Kostenloses eBook

LERNEN

meteor

Free unaffiliated eBook created from
Stack Overflow contributors.

#meteor

Inhaltsverzeichnis

Über ... 1

Kapitel 1: Erste Schritte mit Meteor ... 2

Bemerkungen ... 2

Versionen ... 2

Examples ... 3

Fertig machen ... 3

Installieren Sie Meteor ... 3

Unter OS X und Linux ... 3

Unter Windows ... 3

Erstellen Sie Ihre App ... 3

Starte es ... 3

Beispielanwendungen ... 4

Verwalten von Paketen ... 4

Build-Fortschritt verstehen ... 5

Linux / OSX-Beispiel ... 5

Windows-Beispiel ... 5

Überprüfen der Version der Meteor Tool & Meteor-Projekte ... 5

Meteor-Werkzeug ... 5

Meteor-Projekte ... 6

Meteor-Website ... 6

Aktualisieren von Meteor-Projekten und installierten Paketen ... 6

Erstellen Sie mobile Apps ... 7

Kapitel 2: Abnahmeprüfung (mit Nightwatch) ... 8

Bemerkungen ... 8

Examples ... 8

App-Oberfläche ... 8

Benutzerdefinierte Befehle ... 9

Meteorobjekte auf dem Client prüfen ... 10

Formulare und Eingabetypen ... 10

Komponenten & Seitenobjekte ... 12
Kapitel 3: Anfängerleitfaden zur Installation von Meteor 1.4 unter AWS EC2 ... 14
Examples ... 14
Melden Sie sich für den AWS-Service an ... 14
Kapitel 4: Bereitstellung mit Upstart ... 20
Examples ... 20
Upstart-Service ... 20
Dateien auf den Server kopieren und dann erstellen ... 20
Bundle, dann auf Server kopieren ... 20
Schreiben Sie Ihr Upstart-Skript ... 20
Upstart-Skript für Replikatsätze ... 21
Ausführen Ihres Upstart-Skripts ... 21
Einrichten eines Servers zum Hosten mehrerer Meteor-Apps ... 22
Kapitel 5: Blaze Templating ... 23
Einführung ... 23
Examples ... 23
Füllen Sie eine Vorlage aus einem Methodenaufruf aus ... 23
Datenkontext einer Vorlage ... 23
Vorlagenhelfer ... 24
Kapitel 6: Blaze-Benutzeroberflächenrezepte (Bootstrap; keine jQuery) ... 26
Bemerkungen ... 26
Examples ... 26
Dropdown-Menü ... 26
Navbars ... 27
Modals ... 28
Tagging ... 29
Alarme und Fehler ... 31
Workflow mit Registern ... 33
Kapitel 7: Continuous Integration & Device Clouds (mit Nightwatch) ... 36
Bemerkungen ... 36
Examples ... 36
Travis ... 36

Kreis ... 37
SauceLabs ... 39
BrowserStack ... 39
Kapitel 8: Datei hochladen ... 41
Bemerkungen ... 41
Examples ... 41
Server / Client ... 41
Dropzone (mit Eisen: Router) ... 43
Filepicker.io ... 44
CollectionFS ... 44
Server-Uploads ... 45
Kapitel 9: Daten veröffentlichen ... 47
Bemerkungen ... 47
Examples ... 47
Basis-Abonnement und Veröffentlichung ... 47
Globale Veröffentlichungen ... 48
Benannte Publikationen ... 48
Abonnements mit Vorlagenbereich ... 48
In einer ephemeren clientseitigen Sammlung veröffentlichen ... 49
Fehler in einer Publikation erstellen und darauf reagieren ... 49
Erneutes Abonnieren einer Publikation ... 50
Warten Sie in der Blaze-Ansicht, während veröffentlichte Daten abgerufen werden ... 50
Überprüfen des Benutzerkontos beim Veröffentlichen ... 51
Veröffentlichen Sie mehrere Cursor ... 51
Simulieren Sie die Verzögerung in Publikationen ... 51
Publikationen zusammenführen ... 52
Kapitel 10: Daten von einem Meteor.call abrufen ... 53
Examples ... 53
Die Grundlagen von Meteor.call ... 53
Session-Variable verwenden ... 54
Serverseite ... 54
Client-Seite ... 54

ReactiveVar verwenden ... 54
Serverseite ... 54
Client-Seite ... 55
Kapitel 11: Debuggen ... 56
Examples ... 56
Browser-Debugger ... 56
Fügen Sie Ihrer App Debugger-Haltepunkte hinzu ... 56
Serverseitiges Debuggen mit Knoteninspektor ... 56
Serverseitiges Debuggen mit npm debuggen ... 57
Meteor Shell ... 57
Andere Debugging-Dienstprogramme ... 57
Kapitel 12: Electrify - Meteor als lokal installierbare App kompilieren ... 58
Examples ... 58
Installieren von Electrify für eine Meteor-Anwendung ... 58
Verwenden von Electrify bei einer Meteoranwendung ... 59
Kapitel 13: Entwicklungswerkzeuge ... 61
Examples ... 61
Integrierte Entwicklungsumgebungen ... 61
Datenbank-Tools ... 61
Remote-Collaboration-Dienstprogramme für verteilte Entwickler ... 61
REST-Clients ... 62
Debugger ... 62
Mobile Codierung unter iOS ... 62
Kapitel 14: ES2015-Module (Import und Export) ... 64
Bemerkungen ... 64
Examples ... 64
Importieren in App-Modulen ... 64
Importieren in Meteor-Paketen ... 64
Variablen aus App-Modulen exportieren ... 64
Symbole aus Meteor-Paketen exportieren ... 65
Kapitel 15: ESLint ... 66
Examples ... 66

Eslint zu Ihrem Meteor-Projekt hinzufügen 66
Verwenden Sie ein npm-Skript, um Ihren Code einzufahren 66
Kapitel 16: Grundlegendes Codeship-Setup für automatisiertes Testen 67
Examples 67
Codeship einrichten 67
Bereiten Sie das Projekt vor 67
Kapitel 17: Hintergrundaufgaben 69
Bemerkungen 69
Examples 69
Einfacher Cron 69
Kapitel 18: Horizontale Skalierung 70
Examples 70
Bereitstellen einer Anwendung mit separater Datenbank (MONGO_URL) 70
Replikatset-Konfiguration 70
Konfigurieren eines Replikatsatzes zur Verwendung von Oplogging 70
Oplog Upstart-Skript 71
Scherben 71
Kapitel 19: Integration von Drittanbieter-APIs 72
Examples 72
Einfacher HTTP-Aufruf 72
Erstellen Sie ein Paket für Ihren API-Wrapper 72
Erstellen Sie ein Atmosphere-Paket für Ihren API-Wrapper 72
Binden Sie das API-Paket in Ihre Anwendung ein 73
Verwenden des API-Wrapper-Objekts in Ihrer App 73
Kapitel 20: Knoten / NPM 74
Examples 74
Meteor getestete / unterstützte Knotenversion 74
Kapitel 21: Kontinuierliche Bereitstellung von Codeship für Galaxy 75
Bemerkungen 75
Examples 75
Konfiguration 75
Kapitel 22: Leistungsoptimierung 76

Bemerkungen ... 76
Examples ... 76
Entwerfen und Bereitstellen von produktionsbereiter Software ... 76
Kapitel 23: Meteor + React + ReactRouter ... 78
Einführung ... 78
Examples ... 78
Erstellen Sie das Projekt ... 78
Hinweis: ... 78
Fügen Sie React + ReactRouter hinzu ... 79
Hinweis: ... 80
Schritt 3 - Konten hinzufügen ... 80
Hinweis: ... 81
Rollen hinzufügen ... 81
Hinweis: ... 84
Kapitel 24: Meteor + Reaktion ... 85
Bemerkungen ... 85
Examples ... 85
Setup und "Hello World" ... 85
Erstellen Sie einen reaktiven Container mit createContainer ... 85
Anzeigen einer MongoDB-Sammlung ... 86
Kapitel 25: Meteor Benutzerkonten ... 89
Examples ... 89
Meteor-Kontopaket ... 89
Konto-Passwort ... 89
Zugriff auf Benutzerdaten ... 90
Andere Kontofunktionen ... 90
Verwenden Sie nicht das Standardprofilfeld ... 91
Kapitel 26: Meteor mit einem Proxy-Server verwenden ... 92
Examples ... 92
Verwendung der `HTTP [S] _PROXY`-Env-Var ... 92
Einrichten einer Proxy-Ebene ... 92

Kapitel 27: Mobile Apps 93
Examples 93
Seitenlayout auf verschiedenen Geräten - CSS 93
Festgelegte Fenstergröße 93
Offline-Zwischenspeicherung 94
Scroll-Bounce deaktivieren 94
Multitouch & Gesten 94
Erstellen Sie Ihre Symbole und Begrüßungsbildschirme 95
Meteor Cordova Architektur-Pipeline 96
IOS-Entwicklung 97
IOS-Gerätetest 97
Konfigurieren Sie Ihr Cordova-Projekt (config.xml) 97
Erkennen des Geräteereignisses 98
Kapitel 28: Mongo-Datenbankverwaltung 99
Bemerkungen 99
Examples 99
Vererbte Datenbank analysieren 99
Stellen Sie eine Verbindung zu einer Datenbank auf * .meteorapp.com her 99
Laden Sie eine Datenbank von * .meteor.com herunter 100
Daten aus lokaler Meteor-Entwicklungsinstanz exportieren? 100
Wiederherstellen von Daten aus einem Dumpfile 100
Exportieren Sie eine Collection nach JSON 100
Importieren Sie eine JSON-Datei in Meteor 100
Daten zwischen Staging und lokalen Datenbanken kopieren 101
Komprimieren Sie eine Mongo-Datenbank auf einer Ubuntu-Box 101
Setzen Sie einen Replikatsatz zurück 102
Stellen Sie eine Remote-Verbindung zu einer Mongo-Instanz auf * .meteor.com her 102
Zugriff auf Mongo-Protokolldateien in einer lokalen Meteor-Instanz 102
Protokolldateien auf einer Ubuntu-Box drehen 102
Kapitel 29: MongoDB 104
Einführung 104
Examples 104

Exportieren Sie eine entfernte Mongo-Datenbank und importieren Sie sie in eine lokale Mete ... 104
Holen Sie sich die Mongo-URL Ihrer lokalen Meteor-Mongo-Datenbank ... 104
Verbinden Sie Ihre lokale Meteor-App mit einer alternativen Mongo-Datenbank ... 104
Linux / MacOS-Beispiel: ... 104
Windows-Beispiel ... 105
NPM ... 105
Meteor ohne MongoDB ausführen ... 105
Fertig machen ... 105
Dokumente abfragen ... 106
Dokumente einfügen ... 106
Dokumente aktualisieren ... 106
Dokumente löschen ... 106
Kapitel 30: MongoDB-Aggregation ... 108
Bemerkungen ... 108
Examples ... 108
Server Aggregation ... 108
Aggregation in einer Servermethode ... 109
Kapitel 31: Mongo-Sammlungen ... 110
Bemerkungen ... 110
Examples ... 110
Datensätze in einer älteren Datenbank erstellen ... 110
Daten in ein Dokument einfügen ... 110
Abrufen der _id des zuletzt erstellten Dokuments ... 110
Zeitreihendaten ... 111
Filtern mit Regexes ... 111
Geospatial Collections - Weitere Informationen ... 112
Überwachungssammlungsabfragen ... 113
Beobachter & Arbeiterfunktionen ... 113
Kapitel 32: Mongo-Schema-Migrationen ... 115
Bemerkungen ... 115
Examples ... 115
Versionsfeld zu allen Datensätzen in einer Sammlung hinzufügen ... 115

Entfernen Sie das Array aus allen Datensätzen in einer Sammlung ... 115
Sammlung umbenennen ... 115
Feld suchen, das bestimmte Zeichenfolge enthält ... 115
Neues Feld aus Alt erstellen ... 116
Ziehen Sie Objekte aus einem Array und platzieren Sie sie in einem neuen Feld ... 116
Blob-Datensatz aus einer Sammlung in eine andere Sammlung (z. B. Join & Flatten entfernen) ... 116
Stellen Sie sicher, dass das Feld vorhanden ist ... 116
Stellen Sie sicher, dass das Feld einen bestimmten Wert hat ... 116
Datensatz entfernen, wenn ein bestimmtes Feld ein bestimmter Wert ist ... 117
Ändern Sie den spezifischen Wert des Feldes in Neuer Wert ... 117
Spezifisches Feld auf Null setzen ... 117
Konvertieren Sie ObjectId in String ... 117
Konvertieren Sie Feldwerte von Zahlen in Strings ... 117
Konvertieren Sie Feldwerte von Strings in Zahlen ... 117
Erstellen Sie einen Zeitstempel aus einer ObjectID im Feld _id ... 118
Erstellen Sie eine ObjectID aus einem Datumsobjekt ... 118
Finden Sie alle Datensätze, die Elemente in einem Array enthalten ... 118
Kapitel 33: Nightwatch - Konfiguration und Setup ... 119
Bemerkungen ... 119
Examples ... 119
Aufbau ... 119
Installation & Nutzung ... 120
Startskripte einrichten ... 121
Ordnerstruktur ... 122
Datengesteuertes Testen ... 122
Kapitel 34: Offline-Apps ... 124
Bemerkungen ... 124
Examples ... 124
Meteor.status () ... 124
Appcache aktivieren ... 124
Aktivieren Sie GroundDB ... 125
Dinge, vor denen man vorsichtig sein sollte ... 125

Kapitel 35: Polymer mit Meteor verwenden ... 126
Examples ... 126
Mit Differential: vulkanisieren ... 126
Kapitel 36: Protokollierung ... 128
Examples ... 128
Grundlegende serverseitige Protokollierung ... 128
Tools zur clientseitigen Protokollierung ... 128
Erweiterte Serverprotokollierungstools ... 128
Protokollierungsfehler bei Datenbankklappe ... 129
Protokollierung von Informationen zum Datenkontext in einem Vorlagenhelfer ... 129
Protokollierung von Ereignissen und Benutzerinteraktionen ... 129
Protokollierung mit Variablen auf Protokollebene ... 129
Deaktivieren Sie die Protokollierung in der Produktion ... 130
Winston ... 130
Loglevel ... 130
Kapitel 37: Reaktiv (Vars & Wörterbücher) ... 132
Examples ... 132
Reaktive Abfrage ... 132
Kapitel 38: Replikatsätze und Sharding ... 134
Bemerkungen ... 134
Examples ... 134
Replikatsatz-Schnellstart ... 134
Replikatset-Konfiguration ... 135
Kapitel 39: Routing ... 136
Examples ... 136
Routing mit dem Iron Router ... 136
Mit FlowRouter ... 137
Installieren Sie FlowRouter ... 137
Vorlage rendern ... 137
Rendern einer Vorlage mit Parametern und / oder Abfrage ... 138
Kapitel 40: Umgebungserkennung ... 139

Examples ... 139
Erweiterte Umgebungskonfigurationen ... 139
App-Parameter mit METEOR_SETTINGS angeben ... 139
Umgebungserkennung auf dem Server ... 140
Erkennung der Clientumgebung mit Meteor-Methoden ... 140
Erkennung der Clientumgebung mit NODE_ENV ... 141
Kapitel 41: Umgebungsvariablen ... 142
Parameter ... 142
Examples ... 144
Verwenden von Umgebungsvariablen mit Meteor ... 144
Meteor SMTP-Server einstellen ... 144
Kapitel 42: Umschließen asynchroner Methoden in eine Fiber für die synchrone Ausführung. ... 145
Syntax ... 145
Parameter ... 145
Bemerkungen ... 145
Examples ... 145
Asynchronous NPM-Methoden synchron mit Callbacks ausführen ... 145
Kapitel 43: Vermögenswerte ... 147
Examples ... 147
Zugriff auf Assets auf dem Server ... 147
Textdateien ... 147
Binärdateien ... 147
Kapitel 44: Veröffentlichen eines Release-Tracks ... 148
Bemerkungen ... 148
Examples ... 148
Grundlegende Verwendung ... 148
Manifest freigeben ... 148
Anpassen des Meteor-Tools ... 149
Auslösen eines Veröffentlichungsmanifests aus .meteor / versions ... 149
Anzeigen des Release-Manifests für eine bestimmte Version ... 149
Veröffentlichen einer Veröffentlichung aus der Kasse ... 149
Abrufen der neuesten Commits für jedes Paket in einem Release ... 149

Kapitel 45: Verwenden Sie private Meteor-Pakete für das Codeship 151
Bemerkungen 151
Examples 151
Installieren Sie MGP 151
Konfigurieren Sie Codeship für die Installation von privaten Github-Paketen 151
Kapitel 46: Verzeichnisaufbau 153
Einführung 153
Bemerkungen 153
Examples 154
Klassische Verzeichnisstrukturen 154
Verzeichnisstruktur nur für Pakete 154
Verzeichnisstruktur der Imports / Module 155
Verzeichnisstruktur im gemischten Modus 155
Reihenfolge beim Laden des Verzeichnisses 155
Kapitel 47: Vollständige Installation - Mac OSX 157
Examples 157
Installieren Sie Node & NPM 157
Komplettlösung zur Installation von Meteor 157
Mongo-Installation 158
Andere Entwicklungswerkzeuge 158
Kapitel 48: Zugriff auf Meteor Build-Maschinen von Windows aus 160
Bemerkungen 160
Examples 160
PuTTY verwenden (Fortgeschrittene) 160
Cygwin verwenden (Unix-Tools unter Windows) 160
Credits 162

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: meteor

It is an unofficial and free meteor ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official meteor.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit Meteor

Bemerkungen

Meteor ist eine **Full-Stack-** JavaScript-Plattform für die Entwicklung moderner Web- und Mobilanwendungen.

Innerhalb *eines* Projekts können Sie Ihren Client (Browser und / oder Hybrid Mobile App für Android und / oder iOS) *und* Serverseiten erstellen.

Referenzseiten:

- Meteor Guide
- Meteor-API-Dokumente
- Meteor-Tutorials
- Meteor-Foren

Versionen

Ausführung	Veröffentlichungsdatum
0,4,0	2012-08-30
0,5,0	2013-10-17
0,6,0	2013-04-04
0,7,0	2013-12-20
0,8,0	2014-04-21
0,9,0	2014-08-26
0,9,1	2014-09-04
0,9,2	2014-09-15
0,9,3	2014-09-25
0.9.4	2014-10-13
1.0.1	2014-12-09
1.0.2	2014-12-19
1.0.3.1	2014-12-09
1.1.0	2015-03-31

Ausführung	Veröffentlichungsdatum
1.2.0	2015-09-21
1.3.0	2016-03-27
1.4.0	2016-07-25
1.5.0	2017-05-30

Examples

Fertig machen

Installieren Sie Meteor

Unter OS X und Linux

Installieren Sie die neueste offizielle Meteor-Version von Ihrem Terminal:

```
$ curl https://install.meteor.com/ | sh
```

Unter Windows

Laden Sie den offiziellen Meteor-Installer hier herunter .

Erstellen Sie Ihre App

Nachdem Sie Meteor installiert haben, erstellen Sie ein Projekt:

```
$ meteor create myapp
```

Starte es

Führen Sie es lokal aus:

```
$ cd myapp
$ meteor npm install
$ meteor
```

Hinweis: Meteor-Server läuft unter: http: // localhost: 3000 /

Dann gehen Sie zu http: // localhost: 3000 , um Ihre neue Meteor-Anwendung anzuzeigen.

- Weitere Informationen zum Einstieg in Meteor finden Sie im [Meteor Guide] .
- Entdecken Sie Meteor Packages bei atmosphere - ein moderner, schneller und ausgereifter Paketmanager.

Beispielanwendungen

Meteor hat mehrere Beispiel-Apps eingebaut. Sie können ein Projekt mit einem von ihnen erstellen und lernen, wie es erstellt wurde. Um eine Beispiel-App zu erstellen, installieren Sie Meteor (siehe Erste Schritte) und geben Sie Folgendes ein:

```
meteor create --example <app name>
```

Um zum Beispiel eine Beispiel- `todos` App zu erstellen, schreiben Sie:

```
meteor create --example todos
```

Um eine Liste aller Beispiel-Apps zu erhalten, geben Sie Folgendes ein:

```
meteor create --list
```

Verwalten von Paketen

Meteor hat ein eigenes Paket-Repository auf atmospherejs.com

Sie können neue Pakete aus der Atmosphäre hinzufügen, indem Sie Folgendes ausführen:

```
meteor add [package-author-name:package-name]
```

Zum Beispiel:

```
meteor add kadira:flow-router
```

Ebenso können Sie dasselbe Paket entfernen, indem Sie:

```
meteor remove kadira:flow-router
```

Um die aktuellen Pakete in Ihrem Projekt anzuzeigen, geben Sie Folgendes ein:

```
meteor list
```

Eine Liste der Pakete finden Sie auch in der Datei `./meteor/packages` . Um ein Paket hinzuzufügen, fügen Sie den Paketnamen in diese Datei ein und löschen Sie ihn.

Ein Paket lokal (zB nicht veröffentlichte Pakete oder bearbeitete Version der veröffentlichten Pakete), speichern Sie das Paket in hinzuzufügen `packages` Ordner in der Wurzel.

Ab Version 1.3 **fügte** Meteor **Unterstützung für npm-Pakete hinzu** .

Sie können den Befehl `npm` im Verzeichnis des Meteor-Projekts verwenden, wie Sie es normalerweise ohne Meteor tun würden, oder mit dem Befehl `meteor npm` , der die mitgelieferte Version von npm verwendet.

Build-Fortschritt verstehen

Manchmal dauern Builds länger als erwartet. Es gibt einige Umgebungsvariablen, die Sie einstellen können, um besser zu verstehen, was während des Erstellungsprozesses geschieht.

```
METEOR_DEBUG_BUILD=1        (logs progress)
METEOR_PROFILE=<n>          (logs time spent)
METEOR_DEBUG_SPRINGBOARD=1 (?)
METEOR_DEBUG_SQL=1          (logs SQLITE calls)
METEOR_PROGRESS_DEBUG=1     (? looks like it might be useful, but seems confusing)
```

Wobei `<n>` eine Anzahl von ms ist. Jeder Vorgang, der länger dauert, wird protokolliert.

Linux / OSX-Beispiel

```
export METEOR_DEBUG_BUILD=1
export METEOR_PROFILE=100
meteor
```

Windows-Beispiel

```
set METEOR_DEBUG_BUILD=1
set METEOR_PROFILE=100
meteor
```

Überprüfen der Version der Meteor Tool & Meteor-Projekte

Meteor-Werkzeug

Um die installierte Version des Meteor-Tools zu überprüfen, führen Sie einfach den folgenden Befehl außerhalb eines Meteor-Projekts aus:

```
meteor --version
```

Um eine Liste aller offiziellen (empfohlenen) Meteor-Versionen zu erhalten, führen Sie Folgendes aus:

```
meteor show METEOR
```

Meteor-Projekte

Wenn Sie die Projektversion von Meteor überprüfen möchten, können Sie den folgenden Befehl auch innerhalb eines Projekts ausführen:

```
meteor --version
```

oder drucken Sie einfach den Inhalt der Datei `.meteor/release` :

```
cat .meteor/release
```

`.meteor/versions` Sie die Version der Pakete überprüfen möchten, die derzeit in Ihrem Meteor-Projekt installiert sind, drucken Sie den Inhalt der Datei `.meteor/versions` :

```
cat .meteor/versions
```

Meteor-Website

Um zu sehen, welche Version von Meteor eine auf Meteor basierende Website `Meteor.release` Sie den Inhalt von `Meteor.release` während des Besuchs der Website in der `Meteor.release` :

```
Meteor.release
```

Aktualisieren von Meteor-Projekten und installierten Paketen

Das Meteor Tool benachrichtigt Sie, wenn eine neuere Version verfügbar ist.

Führen Sie den folgenden Befehl in einem Meteor-Projekt aus, um Meteor-Projekte auf die neueste Version zu aktualisieren:

```
meteor update
```

Wenn Sie Ihr Meteor-Projekt auf eine bestimmte Meteor-Version aktualisieren möchten, führen Sie den folgenden Befehl innerhalb des Projekts aus:

```
meteor update --release <release>
```

Wenn Sie alle Nicht-Core-Pakete aktualisieren möchten, führen Sie Folgendes aus:

```
meteor update --packages-only
```

Sie können bestimmte Pakete auch aktualisieren, indem Sie deren Namen als

Befehlszeilenargument an das `meteor update` . Beispiel:

```
meteor update [packageName packageName2 ...]
```

Erstellen Sie mobile Apps

Meteor verwendet Cordova, um Ihre Anwendung in eine mobile *Hybrid*- App zu packen. Nach dem Packen kann die App wie native Apps (über den Apple App Store, Google Play Store usw.) verteilt werden.

1. **Fügen Sie** die Zielplattform (en) Ihrem Meteor-Projekt hinzu:

```
meteor add-platform android
meteor add-platform ios # Only available with Mac OS
```

2. **Installieren Sie** das Android SDK und / oder Xcode (für iOS erfordert Mac OS).

3. **Führen Sie** Ihr Projekt aus (beginnen Sie mit dem Entwicklungsmodus):

```
meteor run android # You may need to configure a default Android emulator first
```

Für iOS (nur unter Mac OS verfügbar):

```
meteor run ios # This will auto start an iOS simulator
```

4. **Bauen Sie** Ihr App-Paket für die Verteilung auf:

```
meteor build <output_folder> --server <url_app_should_connect_to>
```

Dadurch werden `android` und / oder `ios` Ordner neben Ihrem Server-Bundle erstellt.

- Der `android` Ordner enthält die Datei `release-unsigned.apk` , die Sie signieren und ausrichten müssen.
- Der Ordner `ios` enthält das Xcode-Projekt, das Sie signieren müssen.

Siehe auch das Thema Meteor Mobile Apps .
Referenzseite: Meteor Guide> Build> Mobile

Erste Schritte mit Meteor online lesen: https://riptutorial.com/de/meteor/topic/439/erste-schritte-mit-meteor

Kapitel 2: Abnahmeprüfung (mit Nightwatch)

Bemerkungen

Nightwatch bietet seit v0.5 Tagen Akzeptanz- und End-to-End-Tests für Meteor-Apps an und hat Migrationen von PHP nach Spark nach Blaze und React verwaltet. und alle wichtigen Continuous Integration-Plattformen. Weitere Hilfe finden Sie unter:

Nightwatch-API-Dokumentation
Nightwatch.js Google-Gruppe

Examples

App-Oberfläche

Auf der einfachsten Ebene sind Abnahmetests im Wesentlichen Black-Box-Tests, bei denen es im Wesentlichen darum geht, die Eingänge und Ausgänge eines geschlossenen Systems zu testen. Daher gibt es drei wesentliche Funktionen für das Akzeptieren von Tests: Lokalisieren einer Ressource, Lesen von Daten und Schreiben von Daten. Bei Browsern und Webapps lassen sich diese drei Funktionen im Wesentlichen auf Folgendes reduzieren:

1. Laden Sie eine Webseiten- oder Anwendungsansicht
2. Benutzeroberflächenelemente (z. B. DOM) prüfen
3. Ereignis auslösen / Benutzerinteraktion simulieren

Wir nennen dies die Oberfläche der Anwendung. Oberfläche ist alles, was ein Benutzer sieht oder erlebt. Es ist die Außenseite eines Blackbox-Systems. Und da Benutzer mit Web-Browsern mit modernen Webanwendungen interagieren, wird die Flächendeckung durch universelle Ressourcen-Locators (URLs) und Viewports definiert. Und so sieht unser allererster Walkthrough so aus:

```
module.exports = {
  "Hello World" : function (client) {
    client
      // the location of our Meteor app
      .url("http://localhost:3000")

      // the size of the viewport
      .resizeWindow(1024, 768)

      // test app output
      .verify.elementPresent('h1')
      .verify.containsText('h1', "Welcome to Meteor!")
      .verify.containsText('p', "You've pressed the button 0 times")
      .verify.elementPresent('button')

      // simulate user input
      .click('button').pause(500)
```

```
      // test app output again, to make sure input worked
      .verify.containsText('p', "button 1 times")

      // saving a copy of our viewport pixel grid
      .saveScreenshot('tests/nightwatch/screenshots/homepage.png')
      .end();
  }
};
```

Benutzerdefinierte Befehle

Nightwatch unterstützt das Erstellen benutzerdefinierter Befehle, mit denen Tastatureingaben, Mausklicks und andere Eingaben simuliert werden können. Ein benutzerdefinierter Befehl kann wie folgt mit anderen Nightwatch-Befehlen verkettet werden:

```
module.exports = {
  "Login App" : function (client) {
    client
      .url("http://localhost:3000")
      .login("janedoe@somewhere.com", "janedoe123")
      .end();
  }
};
```

Um dies zu aktivieren, definieren Sie einen Befehl in `./tests/nightwatch/commands/login` wie `./tests/nightwatch/commands/login` :

```
exports.command = function(username, password) {

  this
    .verify.elementPresent('#login')

      // we clear the input in case there's any data remaining from previous visits
      .clearValue("#emailInput")
      .clearValue("#passwordInput")

      // we simulate key presses
      .setValue("#emailInput", username)
      .setValue("#passwordInput", password)

    // and we simulate a mouse click
    .click("#signInToAppButton").pause(1000)

  return this; // allows the command to be chained.
};
```

Damit dies alles funktioniert, müssen Sie Ihrer Anmeldeseite `id` Attribute hinzufügen. Auf einer gewissen Ebene muss es ungefähr so aussehen:

```
<template name="login">
  <div id="login">
    <input id="emailInput" name="email" type="email" />
    <input id="passwordInput" name="password" type="password" />
    <button id="#signInToAppButton">Sign In</button>
  </div>
```

```
</template>
```

Meteorobjekte auf dem Client prüfen

Da Nightwatch Zugriff auf die Browserkonsole hat, ist es möglich, clientseitige Objekte mithilfe der `.execute()` API zu überprüfen. Im folgenden Beispiel überprüfen wir das Session-Objekt auf eine bestimmte Session-Variable. Zuerst erstellen wir die Datei `./tests/nightwatch/api/meteor/checkSession` , in der wir den folgenden Befehl behalten:

```
// syncrhonous version; only works for checking javascript objects on client
exports.command = function(sessionVarName, expectedValue) {
  var client = this;
  this
    .execute(function(data){
      return Session.get(data);
    }, [sessionVarName], function(result){
      client.assert.ok(result.value);
      if(expectedValue){
        client.assert.equal(result.value, expectedValue);
      }
    })
    return this;
};
```

Wir können es dann so ketten:

```
module.exports = {
  "Check Client Session" : function (client) {
    client
      .url("http://localhost:3000")
      .checkSession("currentUser", "Jane Doe")
      .end();
  }
};
```

Formulare und Eingabetypen

Um eine Datei hochzuladen, müssen Sie zuerst ein / data-Verzeichnis erstellen und die Datei hinzufügen, die Sie hochladen möchten.

```
tests/nightwatch/data/IM-0001-1001.dcm
```

Ihr Formular benötigt eine Eingabe mit Dateityp. (Manche Leute mögen die Stiloptionen, die diese Eingabe bietet, nicht; und es ist ein übliches Muster, diese Eingabe auszublenden; und eine weitere Schaltfläche auf der Seite für den Benutzer anklicken.)

```
<form id="myform">
    <input type="file" id="fileUpload">
    <input type="text" name="first_name">
    <input type="text" name="last_name">

    <input type="date" name="dob_month">
```

```
    <input type="date" name="dob_day">
    <input type="date" name="dob_year">

    <input type="radio" name="gender" value="M">
    <input type="radio" name="gender" value="F">
    <input type="radio" name="gender" value="O">

    <input type="select" name="hs_graduation_year">
    <input type="text" name="city">
    <input type="select" name="state">

    <input type="submit" name="submit" value="Submit">
</form>
```

Ihre Tests müssen dann setValue () verwenden und den Pfad zum lokalen Dateisystem auflösen.

```
module.exports = {
  "Upload Study" : function (client) {
    console.log(require('path').resolve(__dirname +  '/../data' ));

    var stringArray = "Chicago";

    client
      .url(client.globals.url)
      .verify.elementPresent("form#myform")

      // input[type="file"]
      .verify.elementPresent("input#fileUpload")
      .setValue('input#fileUpload', require('path').resolve(__dirname + '/../data/IM-0001-
1001.dcm'))

      // input[type="text"]
      .setValue('input[name="first_name"]', 'First')
      .setValue('input[name="last_name"]', 'Last')

      // input[type="date"]
      .click('select[name="dob_month"] option[value="3"]')
      .click('select[name="dob_day"] option[value="18"]')
      .click('select[name="dob_year"] option[value="1987"]')

      // input[type="radio"]
      .click('input[name="gender"][value="M"]')

      // input[type="number"]
      .click('select[name="hs_graduation_year"] option[value="2002"]')

      // input[type="text"]
      // sometimes Nightwatch will send text faster than the browser can handle
      // which will cause skipping of letters.  In such cases, we need to slow
      // Nightwatch down; which we do by splitting our input into an array
      // and adding short 50ms pauses between each letter
      for(var i=0; i < userIdArray.length; i++) {
        client.setValue('input[name="city"]', stringArray[i]).pause(50)
      }

      // input[type="select"]
      // after an array input above, we need to resume our method chain...
      client.click('select[name="state"] option[value="CA"]')

      // input[type="number"]
```

```
      .setValue('input[name="zip"]', '01234')

      //input [ type="submit" ]
      .click('button[type="submit"]')
      .end();
  }
};
```

Dank an Daniel Rinehart für die Inspiration dieses Beispiels.

Komponenten & Seitenobjekte

Seitenobjekte ähneln benutzerdefinierten Befehlen. Sie sind jedoch Sammlungen von benutzerdefinierten Befehlen, die einer bestimmten UI-Komponente zugeordnet sind. Dies funktioniert sehr gut mit modernem komponentenbasiertem Design wie in React.

```
module.exports = {
  url: 'http://localhost:3000/login',
  commands: [{
  login: function(email, password) {
    return this
      .clearValue('input[name="emailAddress"]')
      .clearValue('input[name="password"]')

      .setValue('input[name="emailAddress"]', email)
      .setValue('input[name="password"]', password)

      .verify.elementPresent('#loginButton')
      .click("#loginButton");
  },
  clear: function() {
    return this
      .waitForElementVisible('@emailInput')
      .clearValue('@emailInput')
      .clearValue('@passInput')
      .waitForElementVisible('@loginButton')
      .click('@loginButton')
  },
  checkElementsRendered: function(){
    return this
      .verify.elementPresent("#loginPage")
      .verify.elementPresent('input[name="emailAddress"]')
      .verify.elementPresent('input[name="password"]')
  },
  pause: function(time, client) {
    client.pause(time);
    return this;
  },
  saveScreenshot: function(path, client){
    client.saveScreenshot(path);
    return this;
  }
}],
  elements: {
    emailInput: {
      selector: 'input[name=email]'
    },
    passInput: {
```

```
      selector: 'input[name=password]'
    },
    loginButton: {
      selector: 'button[type=submit]'
    }
  }
};
```

Die einzige Einschränkung bei der Verwendung des PageObject-Musters beim Testen von Komponenten besteht darin, dass die Implementierung den Methodenverkettungsfluss `verify.elementPresent` , den das native Nightwatch `verify.elementPresent` bereitstellt. Stattdessen müssen Sie das Seitenobjekt einer Variablen zuweisen und für jede Seite eine neue Methodenkette instanziieren. Ein angemessener Preis für ein konsistentes und zuverlässiges Muster für die Wiederverwendung von Code.

```
module.exports = {
  tags: ['accounts', 'passwords', 'users', 'entry'],
  'User can sign up.': function (client) {

    const signupPage = client.page.signupPage();
    const indexPage = client.page.indexPage();

    client.page.signupPage()
      .navigate()
      .checkElementsRendered()
      .signup('Alice', 'Doe', 'alice@test.org', 'alicedoe')
      .pause(1500, client);

    indexPage.expect.element('#indexPage').to.be.present;
    indexPage.expect.element('#authenticatedUsername').text.to.contain('Alice Doe');
  },
}
```

Abnahmeprüfung (mit Nightwatch) online lesen:
https://riptutorial.com/de/meteor/topic/6454/abnahmeprufung--mit-nightwatch-

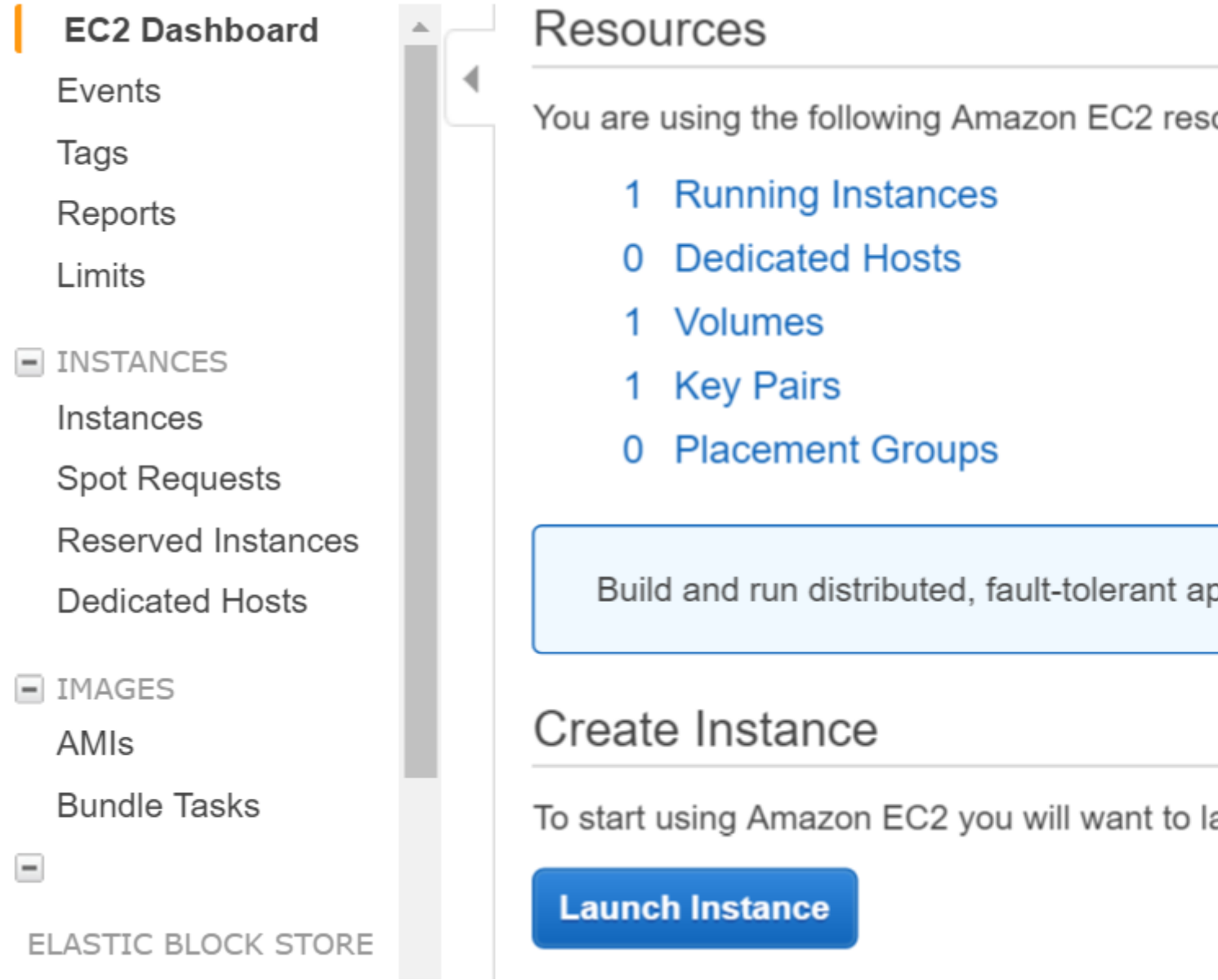
Kapitel 3: Anfängerleitfaden zur Installation von Meteor 1.4 unter AWS EC2

Examples

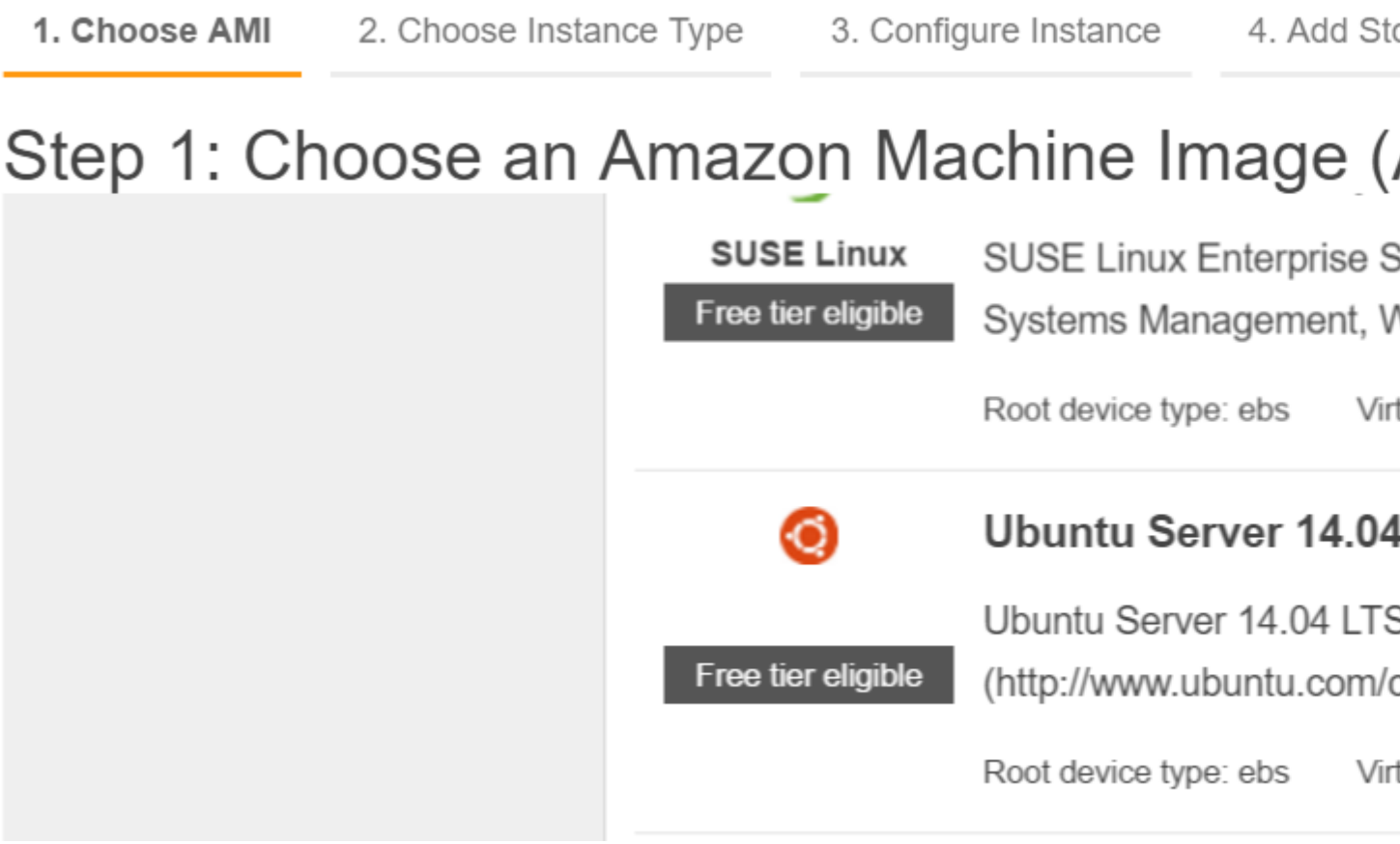
Melden Sie sich für den AWS-Service an

Da viele Anfänger über Cloud-Hosting verwirrt sind. Ich schreibe diesen Leitfaden, um durch das Setzen von Meteor auf AWS mit Ubuntu os zu gehen. Wenn Sie bereits über eine Instanz verfügen, können Sie diesen Schritt überspringen und Meteor auf aws installieren.

Melden Sie sich bei AWS Console an. Wählen Sie EC2. Wechseln Sie zum EC2-Dashboard. Klicken Sie unter Instanz erstellen auf Startinstanz.



Wählen Sie im nächsten Schritt die Ubuntu-Instanz aus



Erstellen Sie ein Schlüsselpaar und laden Sie den privaten Schlüssel auf Ihren lokalen Computer herunter.

Melden Sie sich über die Shell bei aws an (stellen Sie mithilfe des privaten Schlüssels sicher, dass sich der private Schlüssel in Ihrem Pfad befindet, oder führen Sie den Befehl aus dem Verzeichnis aus, das den privaten Schlüssel enthält)

```
ssh -i "myprivatekey.pem" ubuntu@ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com
```

ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com ist ein öffentlicher DNS-Instanzname auf der Amazon-Konsole. ubuntu ist Benutzername. Sie können auch eine öffentliche IP-Adresse verwenden.

SCHRITTE ZUM INSTALLIEREN VON METEOR AUF AWS INSTANCE (mit mupx)

1. Kopieren Sie den privaten Schlüssel vom lokalen Computer in den SSH-Ordner des aws-Servers

Beispiel `/home/ubuntu/.ssh/myprivatekey.pem`

2. Aktualisieren Sie den Packager auf die neueste Version

```
sudo apt-get update
```

3. Installieren Sie die Eigenschaften der Python-Software

```
 sudo apt-get install python-software-properties
```

4. npm und node installieren (optional auch nvm installieren)

```
sudo apt-get install npm
```

Installiere nvm

```
curl https://raw.githubusercontent.com/creationix/nvm/v0.11.1/install.sh | bash
```

Knoten installieren

```
nvm install 4.4.7

nvm use 4.4.7
```

5. Installieren Sie aws cli

```
sudo apt-get install awscli
```

6. Installieren Sie Meteor

```
 sudo npm install -g mupx

 sudo npm install -g mupx-letsencrypt
```

(Meteor 1.4 ist derzeit nur von mpux-letsencrypt verfügbar)

7. Initialisieren Sie mupx, indem Sie in Ihr Projektverzeichnis gehen oder ein neues Verzeichnis erstellen, falls nicht vorhanden

```
mupx-letsencrypt init
```

Wenn Sie einen Fehler wie unten erhalten, ist möglicherweise ein vorhandener Knoten vorhanden, an dem Sie einen Link erstellen müssen

```
/usr/bin/env: node: No such file or directory



sudo ln -s /usr/bin/nodejs /usr/bin/node
```

8. Meteor installieren

```
curl https://install.meteor.com | /bin/sh
```

9. edit mup.json (Vergewissern Sie sich, dass Sie den Benutzernamen: ubuntu und den korrekten Ort des privaten Schlüssels aus Schritt 1 eingegeben haben.)

Nano-Datei-Editor verwenden (um Dateien auf Ubuntu zu bearbeiten, kann auch vi verwendet werden)

```
nano mup.json
```

Beispiel mup.json

```
    {
  // Server authentication info
  "servers": [
    {
      "host": "ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com",
      "username": "ubuntu",
      //"password": "password",
      // or pem file (ssh based authentication)
      "pem": "~/.ssh/myprivatekey.pem",
      // Also, for non-standard ssh port use this
      //"sshOptions": { "port" : 49154 },
      // server specific environment variables
      "env": {}
    }
  ],

  // Install MongoDB on the server. Does not destroy the local MongoDB on future setups
  "setupMongo": true,

  // WARNING: Node.js is required! Only skip if you already have Node.js installed on server.
  "setupNode": false,

  // WARNING: nodeVersion defaults to 0.10.36 if omitted. Do not use v, just the version
number.
  //"nodeVersion": "4.4.7",

  // Install PhantomJS on the server
  "setupPhantom": true,

  // Show a progress bar during the upload of the bundle to the server.
  // Might cause an error in some rare cases if set to true, for instance in Shippable CI
  "enableUploadProgressBar": true,

  // Application name (no spaces).
  "appName": "my-app",

  // Location of app (local directory). This can reference '~' as the users home directory.
  // i.e., "app": "/Users/ubuntu/my-app",
  // This is the same as the line below.
  "app": "/Users/ubuntu/my-app",

  // Configure environment
  // ROOT_URL must be set to https://YOURDOMAIN.com when using the spiderable package & force
SSL
  // your NGINX proxy or Cloudflare. When using just Meteor on SSL without spiderable this is
not necessary
  "env": {
    "PORT": 80,
    "ROOT_URL": "http://myapp.com",
     // only needed if mongodb is on separate server
    "MONGO_URL": "mongodb://url:port/MyApp",
    "MAIL_URL":   "smtp://postmaster%40myapp.mailgun.org:adj87sjhd7s@smtp.mailgun.org:587/"
```

```
  },

  // Meteor Up checks if the app comes online just after the deployment.
  // Before mup checks that, it will wait for the number of seconds configured below.
  "deployCheckWaitTime": 60
}
```

10. Setup-Meteor mit mongo wird mit folgendem Befehl im Projektverzeichnis ausgeführt.

```
mupx-letsencrypt setup
```

11. Projekt mit Mupx bereitstellen

```
mupx-letsencrypt deploy
```

Einige hilfreiche Befehle

So prüfen Sie die Mupx-Protokolle

```
mupx logs -f
```

Docker überprüfen

```
 docker -D info
```

So überprüfen Sie den Netzwerkstatus

```
netstat -a
```

Zur Überprüfung des laufenden Prozesses einschließlich CPU- und Speicherauslastung

```
 top
```

Installieren Sie den Mongo-Client, um den Mongo-Shell-Zugriff auf aws zu erhalten

```
sudo apt-get install mongodb-clients
```

Mongodb-Abfragen ausführen

```
mongo projectName
```

Einmal in der Mongo Shell laufen

```
 db.version()
 db.users.find()
```

Vielen Dank an arunoda für das wunderbare Tool https://github.com/arunoda/meteor-up

Vielen Dank an das mupx-letsencrypt-Team für die gute Arbeit.
https://www.npmjs.com/package/mupx-letsencrypt

Anfängerleitfaden zur Installation von Meteor 1.4 unter AWS EC2 online lesen:
https://riptutorial.com/de/meteor/topic/4773/anfangerleitfaden-zur-installation-von-meteor-1-4-unter-aws-ec2

Kapitel 4: Bereitstellung mit Upstart

Examples

Upstart-Service

In diesem Bereitstellungshandbuch wird davon ausgegangen, dass Sie einen Ubuntu-Server verwenden und sich entweder selbst hosten oder einen Infrastructure as a Service-Anbieter (IaaS) wie Amazon Web Services oder Rackspace verwenden. Auf Ihrem Ubuntu-Server muss ein Daemon ausgeführt werden, um andere Apps zu starten, für die wir den Upstart-Dienst empfehlen. Weitere Informationen zu Upstart finden Sie unter den folgenden Links:

Upstart - Erste Schritte
Erste Schritte mit Upstart-Skripts unter Ubuntu
UbuntuBootupHowTo
Upstart Intro, Kochbuch und Best Practices
Führen Sie NodeJS As a Service auf Ubuntu Karmic aus

Dateien auf den Server kopieren und dann erstellen

Ein bevorzugter Ansatz für die Bereitstellung auf einem Server ist die Verwendung von Git oder GitHub. Dazu müssen Sie sich auf Ihrem Server anmelden, in das Verzeichnis wechseln, von dem aus Sie Ihre App ausführen möchten, und dann Ihre Dateien direkt aus GitHub klonen. Sie erstellen dann Ihre App auf dem Server. Dieser Ansatz stellt sicher, dass plattformspezifische Dateien korrekt erstellt werden, erfordert jedoch die Installation von Meteor auf dem Server (mehr als 500 MB) und kann dazu führen, dass in der Produktion etwas andere Builds entstehen, wenn sich die Server etwas unterscheiden.

```
cd /var/www
sudo git clone http://github.com/myaccount/myapp.git
cd /var/www/myapp
meteor build --directory ../myapp-production
sudo service myapp restart
```

Bundle, dann auf Server kopieren

Alternativ können Sie Ihre Anwendung erstellen und anschließend bereitstellen.

```
cd myapp
meteor build --directory ../output
cd ..
scp output -r username@destination_host:/var/www/myapp-production
```

Schreiben Sie Ihr Upstart-Skript

Sie benötigen ein Upstart-Skript in Ihrem `/etc/init/ directory` `. .conf` es mit dem Namen Ihrer App

und enden mit `.conf` , z. B. `/etc/init/myapp.conf` . Das grundlegende Upstart-Skript sieht in etwa so aus:

```
## /etc/init/myapp.conf
description "myapp.mydomain.com"
author      "somebody@gmail.com"

# Automatically Run on Startup
start on started mountall
stop on shutdown

# Automatically Respawn:
respawn
respawn limit 99 5

script
    export HOME="/root"
    export MONGO_URL='mongodb://myapp.compose.io:27017/meteor'
    export ROOT_URL='http://myapp.mydomain.com'
    export PORT='80'

    exec /usr/local/bin/node /var/www/myapp/main.js >> /var/log/myapp.log 2>&1
end script
```

Upstart-Skript für Replikatsätze

Wenn Sie einen Replikatsatz ausführen oder Ihre Datenbank abteilen müssen, benötigen Sie ein Upstart-Skript, das ungefähr so aussieht:

```
# /etc/init/myapp.conf
description "myapp.mydomain.com"
author      "somebody@gmail.com"

# used to be: start on startup
# until we found some mounts weren't ready yet while booting:
start on started mountall
stop on shutdown

# Automatically Respawn:
respawn
respawn limit 99 5

script
    # upstart likes the $HOME variable to be specified
    export HOME="/root"

    # our example assumes you're using a replica set and/or oplog integreation
    export MONGO_URL='mongodb://mongo-a,mongo-b,mongo-c:27017/?replicaSet=meteor'

    # root_url and port are the other two important environment variables to set
    export ROOT_URL='http://myapp.mydomain.com'
    export PORT='80'

    exec /usr/local/bin/node /var/www/production/main.js >> /var/log/node.log 2>&1
end script
```

Ausführen Ihres Upstart-Skripts

Zum Schluss müssen Sie den Upstart-Daemon starten und Ihre App als Dienst initialisieren.

```
sudo service myapp start
```

Einrichten eines Servers zum Hosten mehrerer Meteor-Apps

https://www.phusionpassenger.com/
https://github.com/phusion/passenger
https://github.com/phusion/passenger/wiki/Phusion-Passenger:-Meteor-tutorial#wiki-installing

Bereitstellung mit Upstart online lesen: https://riptutorial.com/de/meteor/topic/3377/bereitstellung-mit-upstart

Kapitel 5: Blaze Templating

Einführung

Blaze ist eine leistungsstarke Bibliothek zum Erstellen von Benutzeroberflächen durch das Schreiben dynamischer, reaktiver HTML-Vorlagen. Blaze-Templating ermöglicht die direkte Verwendung von Schleifen und bedingter Logik in HTML-Markup. In diesem Abschnitt wird die ordnungsgemäße Verwendung von Vorlagen in Meteor.js mit Blaze erläutert und veranschaulicht.

Examples

Füllen Sie eine Vorlage aus einem Methodenaufruf aus

```
<template name="myTemplate">
  {{#each results}}
    <div><span>{{name}}</span><span>{{age}}</span></div>
  {{/each}}
</template>
```

```
Template.myTemplate.onCreated(function() {
  this.results = new ReactiveVar();
  Meteor.call('myMethod', (error, result) => {
    if (error) {
      // do something with the error
    } else {
      // results is an array of {name, age} objects
      this.results.set(result);
    }
  });
});

Template.myTemplate.helpers({
  results() {
    return Template.instance().results.get();
  }
});
```

Datenkontext einer Vorlage

Wenn eine Vorlage aufgerufen wird, wird der Standarddatenkontext der Vorlage implizit vom Aufrufer abgerufen, da childTemplate im Beispiel den Datenkontext der parentTemplate, dh der Aufrufer-Vorlage, erhält

```
<template name="parentTemplate">
    {{#with someHelperGettingDataForParentTemplate}}
    <h1>My name is {{firstname}} {{lastname}}</h1>
    //some stuffs here
    {{> childTemplate}}
    {{/with}}
</template>
```

In der obigen Situation werden alle Daten, die die Helfer-Extrakte für die übergeordnete Vorlage extrahieren, automatisch von childTemplate abgerufen. Beispiel: Auf {{firstname}} und {{lastname}} kann über childTemplate zugegriffen werden, wie unten gezeigt.

```
<template name="childTemplate">
<h2>My name is also {{firstname}} {{lastname}}</h2>
</template>
```

Wir können sogar den Datenkontext der childTemplate explizit definieren, indem Sie Argumente an die Vorlage übergeben, wie im Beispiel unten.

```
<template name="parentTemplate">
    {{#with someHelperGettingDataForParentTemplate}}
    <h1>My name is {{firstname}} {{lastname}}</h1>
    //some stuffs here
    {{> childTemplate childData=someHeplerReturningDataForChild}}
    {{/with}}
</template>
```

Wenn der Helfer **davon** ausgeht, dass **someHelperReturningDataForChild ein** Objekt wie {profession: "Meteor Developer", Hobby: "stackoverflowing"}, zurückgibt, wird dieses bestimmte Objekt der explizite Datenkontext für die childTemplate. Jetzt können wir in der Kindervorlage etwas tun

```
<template name="childTemplate">
    <h2>My profession is {{profession}}</h2>
    <h3>My hobby is {{hobby}}</h3>
</template>
```

Vorlagenhelfer

Vorlagen-Helfer sind ein wesentlicher Bestandteil von Blaze und bieten sowohl Geschäftslogik als auch Reaktivität für eine Vorlage. Beachten Sie, dass Vorlagenhelfer tatsächlich reaktive Berechnungen sind , die immer dann ausgeführt werden, wenn sich ihre Abhängigkeiten ändern. Je nach Ihren Anforderungen können Vorlagenhelfer global definiert werden oder auf eine bestimmte Vorlage festgelegt werden. Nachfolgend finden Sie Beispiele für jeden Ansatz zur Definition der Vorlagen-Helper.

1. Beispiel für einen Vorlagen-Helfer, der sich auf eine einzelne Vorlage bezieht.

Definieren Sie zuerst Ihre Vorlage:

```
<template name="welcomeMessage">
  <h1>Welcome back {{fullName}}</h1>
</template>
```

Definieren Sie dann den Template-Helfer. Dies setzt voraus, dass der Datenkontext der Vorlage die Eigenschaften firstName und lastName enthält.

```
Template.welcomeMessage.helpers({
```

```
  fullName: function() {
    const instance = Template.instance();
    return instance.data.firstName + ' ' + instance.data.lastName
  },
});
```

2. Beispiel für einen globalen Vorlagen-Helfer (dieser Helfer kann in einer beliebigen Vorlage verwendet werden)

Registrieren Sie zuerst den Helfer:

```
Template.registerHelper('equals', function(item1, item2) {
  if (!item1 || !item2) {
    return false;
  }

  return item1 === item2;
});
```

`equals` Helfer "Gleich" definiert ist, kann ich ihn jetzt in jeder Vorlage verwenden:

```
<template name="registration">
  {{#if equals currentUser.registrationStatus 'Pending'}}
    <p>Don't forget to complete your registration!<p>
  {{/if}}
</template>
```

Blaze Templating online lesen: https://riptutorial.com/de/meteor/topic/2434/blaze-templating

Kapitel 6: Blaze-Benutzeroberflächenrezepte (Bootstrap; keine jQuery)

Bemerkungen

Die oben genannten Blaze-Beispiele sind in hohem Maße mit der http://bootsnipp.com/bibliothek kompatibel, die nur HTML und CSS für Komponenten bereitstellt und das Javascript dem Entwickler überlässt. Auf diese Weise können Komponenten die gleichen zugrunde liegenden Sortier-, Filter-, Abfrage- und Cursormethoden gemeinsam nutzen.

Examples

Dropdown-Menü

Im folgenden Beispiel wird ein Bootstrap-Dropdown-Menü erstellt, das nur Blaze und kein JQuery verwendet.

Dokumentobjektmodell

```
    <nav class="nav navbar-nav">
      <li class="dropdown">
        <a href="#" class="dropdown-toggle" data-toggle="dropdown">{{getSelectedValue}} <span
class="glyphicon glyphicon-user pull-right"></span></a>
        <ul class="fullwidth dropdown-menu">
          <li id="firstOption" class="fullwidth"><a href="#">15 Minutes <span class="glyphicon
glyphicon-cog pull-right"></span></a></li>
          <li class="divider"></li>
          <li id="secondOption"><a href="#">30 Minutes <span class="glyphicon glyphicon-stats
pull-right"></span></a></li>
          <li class="divider"></li>
          <li id="thirdOption"><a href="#">1 Hour <span class="badge pull-right"> 42
</span></a></li>
          <li class="divider"></li>
          <li id="fourthOption"><a href="#">4 Hour <span class="glyphicon glyphicon-heart
pull-right"></span></a></li>
          <li class="divider"></li>
          <li id="fifthOption"><a href="#">8 Hours <span class="glyphicon glyphicon-log-out
pull-right"></span></a></li>
        </ul>
      </li>
    </nav>
```

Javascript

```
Template.examplePage.helpers({
  getSelectedValue:function(){
    return Session.get('selectedValue');
  }
```

```
});
Template.dropDownWidgetName.events({
  'click #firstOption':function(){
    Session.set('selectedValue', 1);
  },
  'click #secondOption':function(){
    Session.set('selectedValue', "blue");
  },
  'click #thirdOption':function(){
    Session.set('selectedValue', $('#thirdOption').innerText);
  },
  'click #fourthOption':function(){
    Session.set('selectedValue', Session.get('otherValue'));
  },
  'click #fifthOption':function(){
    Session.set('selectedValue', Posts.findOne(Session.get('selectedPostId')).title);
  },
});
```

Navbars

Eine sehr häufige Aufgabe ist das Erstellen von responsiven Navigationsleisten und das Erstellen von Aktions- / Fußzeilen mit unterschiedlichen Steuerelementen, je nachdem, auf welcher Seite sich ein Benutzer befindet oder zu welcher Rolle ein Benutzer gehört. Lass uns überlegen, wie man diese Kontrollen macht.

Router

```
Router.configure({
  layoutTemplate: 'appLayout',
});
Router.route('checklistPage', {
    path: '/lists/:_id',
    onBeforeAction: function() {
      Session.set('selectedListId', this.params._id);
      this.next();
    },
    yieldTemplates: {
      'navbarFooter': {
        to: 'footer'
      }
    }
  });
```

Erstellen Sie eine Navbar-Vorlage

```
<template name="navbarFooter">
  <nav id="navbarFooterNav" class="navbar navbar-default navbar-fixed-bottom"
role="navigation">
    <ul class="nav navbar-nav">
      <li><a id="addPostLink"><u>A</u>dd Post</a></li>
      <li><a id="editPostLink"><u>E</u>dit Post</a></li>
      <li><a id="deletePostLink"><u>D</u>elete Post</a></li>
    </ul>
    <ul class="nav navbar-nav navbar-right">
      <li><a id="helpLink"><u>H</u>elp</a></li>
```

```
    </ul>
  </nav>
</template>
```

Definieren Sie die Erträge im Layout

```
<template name="appLayout">
  <div id="appLayout">
    <header id="navbarHeader">
      {{> yield 'header'}}
    </header>

      <div id="mainPanel">
        {{> yield}}
      </div>

    <footer id="navbarFooter" class="{{getTheme}}"">
      {{> yield 'footerActionElements' }}
    </footer>
  </div>
</template>
```

Modals

Dies ist ein reiner Blaze-Ansatz, um Elemente der Benutzeroberfläche in die Existenz hinein und aus ihr heraus zu bewegen. Stellen Sie sich dies als Ersatz für modale Dialoge vor. Tatsächlich gibt es mehrere Möglichkeiten, um modale Dialoge mit dieser Methode zu implementieren (fügen Sie einfach Hintergrundmasken und Animationen hinzu).

Dokumentobjektmodell

```
<template name="topicsPage">
  <div id="topicsPage" class="container">
    <div class="panel">
      <div class="panel-heading">
        Nifty Panel
      </div>
      <!-- .... -->
      <div class="panel-footer">
        <!-- step 1.  we click on the button object -->
        <div id="createTopicButton" class="btn {{ getPreferredButtonTheme }}">Create
Topic</div>
      </div>
    </div>

    <!-- step 5 - the handlebars gets activated by the javascript controller -->
    <!-- and toggle the creation of new objects in our model -->
    {{#if creatingNewTopic }}
    <div>
      <label for="topicTextInput"></label>
      <input id="topicTextInput" value="enter some text..."></input>
      <button class="btn btn-warning">Cancel</button>
      <button class="btn btn-success">OK</button>
    </div>
    {{/if}}
  </div>
```

```
</template>
```

Javascript

```
// step 2 - the button object triggers an event in the controller
// which toggles our reactive session variable
Template.topicsPage.events({
  'click #createTopicButton':function(){
    if(Session.get('is_creating_new topic'){
      Session.set('is_creating_new_topic', false);
    }else{
      Session.set('is_creating_new_topic', true);
    }
  }
});

// step 0 - the reactive session variable is set false
Session.setDefault('is_creating_new_topic', false);

// step 4 - the reactive session variable invalidates
// causing the creatNewTopic function to be rerun
Template.topicsPage.creatingNewTopic = function(){
  if(Session.get('is_creating_new_topic')){
    return true;
  }else{
    return false;
  }
}
```

Tagging

Die Datenbankebene Zunächst wollen wir das Data Distribution Protocol einrichten, um sicherzustellen, dass Daten in der Datenbank gespeichert werden können und diese dem Client zur Verfügung stehen. Es müssen drei Dateien erstellt werden ... eine auf dem Server, eine auf dem Client und eine, die von beiden gemeinsam genutzt wird.

```
// client/subscriptions.js
Meteor.subscribe('posts');

//lib/model.js
Posts  = new Meteor.Collection("posts");
Posts.allow({
    insert: function(){
        return true;
    },
    update: function () {
        return true;
    },
    remove: function(){
        return true;
    }
});


// server.publications.js
Meteor.publish('posts', function () {
  return Posts.find();
```

```
});
```

In diesem Beispiel wird das folgende Dokumentschema für das Tagging-Muster angenommen:

```
{
  _id: "3xHCsDexdPHN6vt7P",
  title: "Sample Title",
  text: "Lorem ipsum, solar et...",
  tags: ["foo", "bar", "zkrk", "squee"]
}
```

Dokumentobjektmodell
Zweitens möchten wir unser Objektmodell in der Anwendungsebene erstellen. Im Folgenden wird beschrieben, wie Sie ein Bootstrap-Bedienfeld verwenden, um einen Beitrag mit Titel, Text und Tags zu rendern. Beachten Sie, dass `selectedPost` , `tagObjects` und `tag` alle Hilfsfunktionen der blogPost-Vorlage sind. `title` und `text` sind Felder aus unserem Dokumentensatz.

```
<template name="blogPost">
  {{#with selectedPost }}
    <div class="blogPost panel panel-default">
      <div class="panel-heading">
        {{ title }}
      </div>
        {{ text }}
      <div class="panel-footer">
        <ul class="horizontal-tags">
          {{#each tagObjects }}
          <li class="tag removable_tag">
            <div class="name">{{tag}}<i class="fa fa-times"></i></div>
          </li>
          {{/each}}
          <li class="tag edittag">
            <input type="text" id="edittag-input" value="" /><i class="fa fa-plus"></i>
          </li>
        </ul>
      </div>
    </div>
  {{/with}}
</template>
```

Javascript
Als Nächstes möchten wir einige Controller so einrichten, dass sie Daten zurückgeben, Dateneingaben implementieren und so weiter.

```
// you will need to set the selectedPostId session variable
// somewhere else in your application
Template.blogPost.selectedPost = function(){
  return Posts.findOne({_id: Session.get('selectedPostId')});
}

// next, we use the _.map() function to read the array from our record
// and convert it into an array of objects that Handlebars/Spacebars can parse
Template.blogPost.tagObjects = function () {
    var post_id = this._id;
    return _.map(this.tags || [], function (tag) {
```

```
        return {post_id: post_id, tag: tag};
    });
};

// then we wire up click events
Template.blogPost.events({
    'click .fa-plus': function (evt, tmpl) {
        Posts.update(this._id, {$addToSet: {tags: value}});
    },
    'click .fa-times': function (evt) {
        Posts.update({_id: this._id}, {$pull: {tags: this.tag}});
    }
});
```

Styling
Schließlich möchten wir einige unterschiedliche Ansichten für Telefone, Tablets und Desktops definieren. und einige grundlegende UI-Gestaltung je nach Benutzereingabe. In diesem Beispiel wird der Precompiler Less verwendet, obwohl die Syntax für Sass und Stylus ungefähr gleich sein sollte.

```
// default desktop view
.fa-plus:hover{
  cursor: pointer;
}
.fa-times:hover{
  cursor: pointer;
}
// landscape orientation view for tablets
@media only screen and (min-width: 768px) {
  .blogPost{
     padding: 20px;
  }
}
// portrait orientation view for tablets
@media only screen and (max-width: 768px) {
  .blogPost{
     padding: 0px;
       border: 0px;
  }
}
// phone view
@media only screen and (max-width: 480px) {
  blogPost{
   .panel-footer{
       display: none;
    }
  }
}
```

Alarme und Fehler

Alarme und Fehler sind fast die einfachsten Muster von Meteor-Komponenten. Sie sind so einfach, dass sie sich kaum als Muster in sich registrieren. Anstatt FlashAlert-Module oder -Muster hinzuzufügen, müssen Sie lediglich eine entsprechende Handlebar-Vorlage formatieren, einen Helper hinzufügen und diese mit einer reaktiven Session-Variablen verbinden.

Voraussetzungen
Für den folgenden Code sind der LESS-Precompiler bzw. Bootstrap-3 erforderlich. Sie müssen die folgenden Befehle an der Eingabeaufforderung ausführen, damit sie funktionieren.

```
meteor add less
meteor add ian:bootstrap-3
```

Dokumentobjektmodell: Definieren des Warnungsobjekts Beginnen Sie mit dem Hinzufügen einiger Elemente zu Ihrem Dokumentobjektmodell. In diesem Fall möchten wir ein div-Element für unsere Warnung erstellen, das mit zwei Handlebar-Helfern verbunden ist.

```
<template name="postsPage">
  <div id="postsPage" class="page">
    <div id="postsPageAlert" class="{{alertColor}}">{{alertMessage}}</div>
    <div class="postsList">
      <!-- other code you can ignore in this example -->
    </div>
    <div id="triggerAlertButton" class="btn btn-default">
  </div>
</template>
```

Javascript: Vorlagenhelfer definieren Dann möchten wir einige Controller miteinander verbinden, die das Objektmodell mit Daten füllen. Wir machen dies mit zwei reaktiven Sitzungsvariablen und zwei Lenkerhelfern.

```
Session.setDefault('alertLevel', false);
Session.setDefault('alertMessage', "");

Template.postsPage.alertColor = function(){
 if(Session.get('alertLevel') == "Success"){
  return "alert alert-success";
 }else if(Session.get('alertLevel') == "Info"){
  return "alert alert-info";
 }else if(Session.get('alertLevel') == "Warning"){
  return "alert alert-warning";
 }else if(Session.get('alertLevel') == "Danger"){
  return "alert alert-danger";
 }else{
  return "alert alert-hidden"
 }
}

Template.postsPage.alertMessage = function(){
  return Session.get('alertMessage');
}
```

Styling: Definieren der DOM-Sichtbarkeit Anschließend möchten wir zu unserem CSS zurückkehren und zwei Ansichten des postsPage-Elements definieren. In der ersten Ansicht zeigen wir alle Inhalte in unserem Objektmodell. In der zweiten Ansicht werden nur einige Inhalte unseres Objektmodells angezeigt.

```
#postsPage{
  .alert{
    display: block;
```

```
  }
  .alert-hidden{
    display: none;
  }
}
```

Javascript: Alarm auslösen
Zum Schluss kehren wir zu unseren Controllern zurück und definieren einen Event-Controller, der beim Anklicken unsere Warnung auslöst.

```
Template.postsPage.events({
  'click #triggerAlertButton':function(){
    Session.set('alertLevel', 'Success');
    Session.set('alertMessage', 'You successfully read this important alert message.');
  }
});
```

Und das ist alles was dazu gehört! Super einfach, richtig? Jetzt können `alertMessage` Sitzungsvariablen `alertLevel` und `alertMessage` beliebigen Stelle in Ihrer Codebase `alertMessage` Ihrer Anwendung werden Alarme und Fehlermeldungen reaktiv `alertMessage` :)

Workflow mit Registern

Dokumentobjektmodell
Beginnen Sie mit der Erstellung Ihrer Registerkarten und Bereiche in Ihrem Objektmodell ...

```
<template name="samplePage">
  <div id="samplePage" class="page">
    <ul class="nav nav-tabs">
      <li id="firstPanelTab"><a href="#firstPanel">First</a></li>
      <li id="secondPanelTab"><a href="#secondPanel">Second</a></li>
    </ul>

    <div id="firstPanel" class="{{firstPanelVisibility}}">
      {{> firstPanel }}
    </div>
    <div id="secondPanel" class="{{secondPanelVisibility}}">
      {{> secondPanel }}
    </div>
  </div>
</template>
```

Javascript

```
// this variable controls which tab is displayed and associated application state
Session.setDefault('selectedPanel', 1);

Template.name.helpers({
  firstPanelVisibility: function (){
    if(Session.get('selectedPanel') === 1){
      return "visible";
    }else{
      return "hidden";
    }
```

```
  },
  secondPanelVisibility: function (){
    if(Session.get('selectedPanel') === 2){
      return "visible";
    }else{
      return "hidden";
    }
  },
  thirdPanelVisibility: function (){
    if(Session.get('selectedPanel') === 3){
      return "visible";
    }else{
      return "hidden";
    }
  },
  firstPanelActive: function (){
    if(Session.get('selectedPanel') === 1){
      return "active panel-tab";
    }else{
      return "panel-tab";
    }
  },
  secondPanelActive: function (){
    if(Session.get('selectedPanel') === 2){
      return "active panel-tab";
    }else{
      return "panel-tab";
    }
  },
  thirdPanelActive: function (){
    if(Session.get('selectedPanel') === 3){
      return "active panel-tab";
    }else{
      return "panel-tab";
    }
  }
});
```

Styling

```
.visible {
  display: block;
  visibility: visible;
}
.hidden {
  display: none;
  visibility: hidden;
}
```

Aktive Registerkarte Für einen zusätzlichen Effekt können Sie dieses Muster erweitern, indem Sie Klassen einfügen, um die aktive Registerkarte anzuzeigen.

```
<li id="firstPanelTab" class="{{firstPanelActive}}"><a href="#firstPanel">First</a></li>
<li id="secondPanelTab" class="{{secondPanelActive}}"><a href="#secondPanel">Second</a></li>
```

```
Template.firstPanel.helpers({
  firstPanelActive: function (){
    if(Session.get('selectedPanel') === 1){
```

```
      return "active";
    }else{
      return "";
    }
  },
  secondPanelActive: function (){
    if(Session.get('selectedPanel') === 2){
      return "active";
    }else{
      return "";
    }
  },
});
```

Blaze-Benutzeroberflächenrezepte (Bootstrap; keine jQuery) online lesen: https://riptutorial.com/de/meteor/topic/4202/blaze-benutzeroberflachenrezepte--bootstrap--keine-jquery-

Kapitel 7: Continuous Integration & Device Clouds (mit Nightwatch)

Bemerkungen

Nightwatch bietet seit v0.5 Tagen Akzeptanz- und End-to-End-Tests für Meteor-Apps an und hat Migrationen von PHP nach Spark nach Blaze und React verwaltet. und alle wichtigen Continuous Integration-Plattformen. Weitere Hilfe finden Sie unter:

Nightwatch-API-Dokumentation
Nightwatch.js Google-Gruppe

Examples

Travis

Travis ist der ursprüngliche Continuous Integration Service, der in der Meteor-Community populär wurde. Es ist solide und zuverlässig, hat schon lange eine Open-Source-Hosting-Klasse und hat im Laufe der Jahre Hunderttausende von Nightwatch-Tests durchgeführt.

.travis.yml
`.travis.yml` einfach eine `.travis.yml` -Datei in das Stammverzeichnis Ihrer Anwendung ein.

```
# this travis.yml file is for the leaderboard-nightwatch example, when run standalone
language: node_js

node_js:
  - "0.10.38"

services:
  - mongodb

sudo: required

env:
  global:
    - TRAVIS=true
    - CONFIG_PREFIX=`npm config get prefix`
    - DISPLAY=:99.0
    - NODE_ENV=`travis`
  matrix:

cache:
  directories:
    - .meteor/local/build/programs/server/assets/packages
    - .meteor

before_install:
  # set up the node_modules dir, so we know where it is
  - "mkdir -p node_modules &"
```

```
  # install nightwatch, selenium, , so we can launch nightwatch and selenium
  - "meteor npm install nightwatch selenium-server-standalone-jar chromedriver"

  # fire up xvfb on port :99.0
  - "sh -e /etc/init.d/xvfb start"

  # set the xvfb screen size to 1280x1024x16
  - "/sbin/start-stop-daemon --start --quiet --pidfile /tmp/custom_xvfb_99.pid --make-pidfile
--background --exec /usr/bin/Xvfb -- :99 -ac -screen 0 1280x1024x16"

  # install meteor
  - "curl https://install.meteor.com | /bin/sh"

  # give meteor a few seconds after installing
  - "sleep 10"

  # setup Meteor app
  - "cd webapp"
  - "meteor &"

  # give Meteor some time to download packages, init data, and to start
  - "sleep 60"

# then run nightwatch using the chromedriver
script: "nightwatch -c .meteor/nightwatch.json"
```

Kreis

Circle ist der neuere Continuous Integration Service, der bei Meteoriten beliebt ist. Es hat alle neusten Schnickschnack, was die kontinuierliche Integration angeht. Das folgende Skript unterstützt viele neue Funktionen, darunter:

- Screenshots
- Artefakte
- Git Submodule
- Umgebungserkennung
- Verzeichnis-Caching
- Parallelitätsoptimierung
- npm-Skripte
- kontinuierliche Bereitstellung
- Webhooks

.circle.yml

```
## Customize the test machine
machine:

  # Timezone
  timezone:
    America/Los_Angeles # Set the timezone

  # Add some environment variables
  environment:
    CIRCLE_ENV: test
```

```
    CXX: g++-4.8
    DISPLAY: :99.0
    NPM_PREFIX: /home/ubuntu/nvm/v0.10.33
    INITIALIZE: true
    NODE_ENV: circle


## Customize checkout
checkout:
 post:
   #- git submodule sync
   #- git submodule update --init --recursive # use submodules

general:
  build_dir: webapp
  artifacts:
    - "./tests/nightwatch/screenshots" # relative to the build directory

## Customize dependencies
dependencies:
  cache_directories:
    - "~/.meteor" # relative to the user's home directory
    - ~/nvm/v0.10.33/lib/node_modules/starrynight
    - ~/nvm/v0.10.33/bin/starrynight

  pre:
    # Install Starrynight unless it is cached
    - if [ ! -e ~/nvm/v0.10.33/bin/starrynight ]; then npm install -g starrynight; else echo
"Starrynight seems to be cached"; fi;
    # Install  Meteor
    - mkdir -p ${HOME}/.meteor
    # If Meteor is already cached, do not need to build it again.
    - if [ ! -e ${HOME}/.meteor/meteor ]; then curl https://install.meteor.com | /bin/sh; else
echo "Meteor seems to be cached"; fi;
    # Link the meteor executable into /usr/bin
    - sudo ln -s $HOME/.meteor/meteor /usr/bin/meteor
    # Check if the helloworld directory already exists, if it doesn't, create the helloworld
app
    # The following doesn't work, because it should be checking ${HOME}/active-
entry/helloworld
    # - if [ ! -e ${HOME}/helloworld ]; then meteor create --release METEOR@1.1.0.3
helloworld; else echo "helloworld app seems to be cached"; fi;

  override:
    #- meteor list

## Customize test commands
test:
  pre:
    #- starrynight fetch
    #- cd packages && rm -rf temp
    #- cd packages && ls -la
    #- starrynight autoconfig
    - meteor update --release METEOR@1.3.3
    - meteor npm install --save jquery bootstrap react react-dom react-router react-bootstrap
react-komposer
    - cat .meteor/nightwatch.json
    - meteor:
          background: true
    - sleep 60
  override:
```

```
    - meteor npm run-script nightwatch


## Customize deployment commands
#deployment:
#  production:
#    branch: master
#    commands:
#      - printf "<Meteor username>\n<Meteor password>\n" | meteor deploy myapp.meteor.com

## Custom notifications
#notify:
  #webhooks:
    # A list of hashes representing hooks. Only the url field is supported.
    #- url: https://someurl.com/hooks/circle
```

SauceLabs

SauceLabs ist eine automatisierte Testplattform für Unternehmen. Es unterstützt sowohl die kontinuierliche Integration als auch Cross-Browser-Tests und eine Cloud für mobile Geräte. Die Kosten sind höher als bei Travis, Circle oder BrowserStack.

```
{
  "selenium" : {
    "start_process" : false,
    "host" : "ondemand.saucelabs.com",
    "port" : 80,
  },
  "test_settings" : {
    "chrome_saucelabs": {
      "selenium_host": "ondemand.saucelabs.com",
      "selenium_port": 80,
      "username": "${SAUCE_USERNAME}",
      "access_key": "${SAUCE_ACCESS_KEY}",
      "use_ssl": false,
      "silent": true,
      "output": true,
      "screenshots": {
        "enabled": false,
        "on_failure": true,
        "path": ""
      },
      "desiredCapabilities": {
        "name": "test-example",
        "browserName": "chrome"
      },
      "globals": {
        "myGlobal": "some_sauce_global"
      }
    },
  }
}
```

BrowserStack

BrowserStack verwendet eine Device Cloud für Cross-Browser-Tests. Die Absicht ist, das Testen

von Selenium-Skripts auf jedem möglichen Gerät zu ermöglichen.

```
{
  "selenium" : {
    "start_process" : false,
    "host" : "hub.browserstack.com",
    "port" : 80,
  },

  "test_settings" : {
    "default" : {
      "launch_url" : "http://hub.browserstack.com",
      "selenium_port"  : 80,
      "selenium_host"  : "hub.browserstack.com",
      "silent": true,
      "screenshots" : {
        "enabled" : false,
        "path" : "",
      },
      "desiredCapabilities": {
        "browserName": "firefox",
        "javascriptEnabled": true,
        "acceptSslCerts": true,
        "browserstack.user": "USERNAME",
        "browserstack.key": "KEY"
      }
    }
  }
}
```

Continuous Integration & Device Clouds (mit Nightwatch) online lesen: https://riptutorial.com/de/meteor/topic/6550/continuous-integration--amp--device-clouds--mit-nightwatch-

Kapitel 8: Datei hochladen

Bemerkungen

Das CollectionFS-Paket wurde vom Autor zurückgestellt und eingestellt. Da es jedoch in Atmosphere oder dem Meteor-Ökosystem kein alternatives Paket für die Verwendung der Mongo-GridFS-Funktionalität gibt, funktioniert der Code weiterhin einwandfrei. Es wird empfohlen, das Beispiel nicht aus der StackOverflow-Dokumentation zu entfernen, bis eine andere GridFS-Lösung als Ersatz dokumentiert werden kann.

Zusätzliche Forschung
Filepicker.io Uploads und Bildkonvertierung
Darios Save File Pattern
Micha Roon's File Upload Pattern
EventedMind-Paket zum Hochladen von Dateien

Examples

Server / Client

Das Hochladen von Dateien kann einfach oder sehr kompliziert sein, je nachdem, was Sie tun möchten. Im Allgemeinen ist das Übertragen einer Datei selbst nicht so schwierig. Es gibt jedoch viele Randfälle in Bezug auf Anhänge, Binärdateien und dergleichen. Der eigentliche Knackpunkt ist die horizontale Skalierung und die Schaffung einer Lösung, die funktioniert, wenn der Server ein zweites, drittes und ntes Mal geklont wird.

Beginnen wir mit einem einfachen Server / Client-Upload-Modell. Wir beginnen mit dem Hinzufügen eines Dateieingabeelements zum Dokumentobjektmodell.

```
<template name="example">
  <input type=file />
</template>
```

Hängen Sie dann ein Ereignis an das Eingabeelement in Ihrem Controller an und rufen Sie die lokale Meteor-Methode `` startFileTransfer '' auf, um die Übertragung zu initiieren.

```
// client/example.js
Template.example.events({
  'change input': function(ev) {
    _.each(ev.srcElement.files, function(file) {
      Meteor.startFileTransfer(file, file.name);
    });
  }
});

// client/save.js
/**
 * @blob (https://developer.mozilla.org/en-US/docs/DOM/Blob)
```

```
 * @name the file's name
 * @type the file's type: binary, text (https://developer.mozilla.org/en-
US/docs/DOM/FileReader#Methods)
 *
 * TODO Support other encodings: https://developer.mozilla.org/en-
US/docs/DOM/FileReader#Methods
 * ArrayBuffer / DataURL (base64)
 */
Meteor.startFileTransfer = function(blob, name, path, type, callback) {
  var fileReader = new FileReader(),
    method, encoding = 'binary', type = type || 'binary';
  switch (type) {
    case 'text':
      // TODO Is this needed? If we're uploading content from file, yes, but if it's from an
input/textarea I think not...
      method = 'readAsText';
      encoding = 'utf8';
      break;
    case 'binary':
      method = 'readAsBinaryString';
      encoding = 'binary';
      break;
    default:
      method = 'readAsBinaryString';
      encoding = 'binary';
      break;
  }
  fileReader.onload = function(file) {
    Meteor.call('saveFileToDisk', file.srcElement.result, name, path, encoding, callback);
  }
  fileReader[method](blob);
}
```

Der Client ruft dann die Server-Methode saveFileToDisk auf, die die eigentliche Übertragung ausführt und alles auf die Festplatte legt.

```
//
/**
 * TODO support other encodings:
 * http://stackoverflow.com/questions/7329128/how-to-write-binary-data-to-a-file-using-node-js
 */
Meteor.methods({
  saveFileToDisk: function(blob, name, path, encoding) {
    var path = cleanPath(path), fs = __meteor_bootstrap__.require('fs'),
      name = cleanName(name || 'file'), encoding = encoding || 'binary',
      chroot = Meteor.chroot || 'public';
    // Clean up the path. Remove any initial and final '/' -we prefix them-,
    // any sort of attempt to go to the parent directory '..' and any empty directories in
    // between '/////' - which may happen after removing '..'
    path = chroot + (path ? '/' + path + '/' : '/');

    // TODO Add file existance checks, etc...
    fs.writeFile(path + name, blob, encoding, function(err) {
      if (err) {
        throw (new Meteor.Error(500, 'Failed to save file.', err));
      } else {
        console.log('The file ' + name + ' (' + encoding + ') was saved to ' + path);
      }
    });
```

```
    function cleanPath(str) {
      if (str) {
        return str.replace(/\.\./g,'').replace(/\/+/g,'').
          replace(/^\/+/,'').replace(/\/+$/,'');
      }
    }
    function cleanName(str) {
      return str.replace(/\.\./g,'').replace(/\//g,'');
    }
  }
});
```

Das ist eine Art nackter Ansatz, der zu wünschen übrig lässt. Es ist vielleicht gut, eine CSV-Datei hochzuladen oder so, aber das war's auch schon.

Dropzone (mit Eisen: Router)

Wenn Sie mit einer integrierten Dropzone-Benutzeroberfläche und einem REST-Endpunkt etwas mehr Feinarbeit haben möchten, müssen Sie mit dem Hinzufügen benutzerdefinierter REST-Routen und -Pakete mit Hilfe von Benutzeroberflächenhelfern beginnen.

Beginnen wir mit dem Import von Iron Router und Dropzone.

```
 meteor add iron:router
 meteor add awatson1978:dropzone
```

Und konfigurieren Sie die URL-Route für Uploads, die im Dropzone-Helper angegeben ist.

```
Router.map(function () {
    this.route('uploads', {
      where: 'server',
      action: function () {
        var fs = Npm.require('fs');
        var path = Npm.require('path');
        var self = this;

        ROOT_APP_PATH = fs.realpathSync('.');

        // dropzone.js stores the uploaded file in the /tmp directory, which we access
        fs.readFile(self.request.files.file.path, function (err, data) {

          // and then write the file to the uploads directory
          fs.writeFile(ROOT_APP_PATH + "/assets/app/uploads/" +self.request.files.file.name,
data, 'binary', function (error, result) {
            if(error){
              console.error(error);
            }
            if(result){
              console.log('Success! ', result);
            }
          });
        });
      }
    });
  });
```

Cool! Wir haben einen Dateiuploader mit schicker Oberfläche und einem programmierbaren REST-Endpunkt. Leider skaliert dies nicht besonders gut.

Filepicker.io

Zum Skalieren müssen wir den lokalen Speicher auf unserem Server nicht mehr verwenden und entweder einen dedizierten Dateispeicherdienst verwenden oder eine horizontale Speicherebene implementieren. Der einfachste Weg, um mit dem skalierbaren Dateispeicher zu beginnen, ist die Verwendung einer Lösung wie Filepicker.io, die S3, Azure, Rackspace und Dropbox unterstützt. loadpicker ist seit einiger Zeit ein beliebtes Filerpicker-Unipaket.

```
meteor add mrt:filepicker
```

Das Filepicker-Muster unterscheidet sich etwas von den anderen Lösungen, da es sich wirklich um die Integration von Drittanbietern handelt. Beginnen Sie mit dem Hinzufügen einer Filepicker-Eingabe, die, wie Sie sehen, stark von data- * -Attributen abhängig ist. Dies ist ein ziemlich ungewöhnliches Muster in Meteor-Apps.

```
<input type="filepicker"
  id="filepickerAttachment"
  data-fp-button-class="btn filepickerAttachment"
  data-fp-button-text="Add image"
  data-fp-mimetypes="image/*"
  data-fp-container="modal"
  data-fp-maxsize="5000000"
  data-fp-services="COMPUTER,IMAGE_SEARCH,URL,DROPBOX,GITHUB,GOOGLE_DRIVE,GMAIL">
```

Sie möchten auch einen API-Schlüssel festlegen, das Filepicker-Widget erstellen, auslösen und dessen Ausgaben beobachten.

```
if(Meteor.isClient){
  Meteor.startup(function() {
    filepicker.setKey("YourFilepickerApiKey");
  });
  Template.yourTemplate.rendered = function(){
    filepicker.constructWidget($("#filepickerAttachment"));
  }
  Template.yourTemplate.events({
  'change #filepickerAttachment': function (evt) {
    console.log("Event: ", evt, evt.fpfile, "Generated image url:", evt.fpfile.url);
  });
});
```

CollectionFS

Wenn Sie es jedoch mit Speicher wirklich ernst meinen und Millionen von Bildern speichern möchten, müssen Sie die GridFS-Infrastruktur von Mongo nutzen und selbst eine Speicherebene erstellen. Dafür benötigen Sie das hervorragende CollectionFS-Subsystem.

Fügen Sie zunächst die erforderlichen Pakete hinzu.

```
meteor add cfs:standard-packages
meteor add cfs:filesystem
```

Und ein Objekt zum Hochladen von Dateien zu Ihrem Objektmodell hinzufügen.

```
<template name="yourTemplate">
    <input class="your-upload-class" type="file">
</template>
```

Fügen Sie dann einen Ereignis-Controller auf dem Client hinzu.

```
Template.yourTemplate.events({
    'change .your-upload-class': function(event, template) {
        FS.Utility.eachFile(event, function(file) {
            var yourFile = new FS.File(file);
            yourFile.creatorId = Meteor.userId(); // add custom data
            YourFileCollection.insert(yourFile, function (err, fileObj) {
                if (!err) {
                   // do callback stuff
                }
            });
        });
    }
});
```

Und definieren Sie Ihre Sammlungen auf Ihrem Server:

```
YourFileCollection = new FS.Collection("yourFileCollection", {
    stores: [new FS.Store.FileSystem("yourFileCollection", {path: "~/meteor_uploads"})]
});
YourFileCollection.allow({
    insert: function (userId, doc) {
        return !!userId;
    },
    update: function (userId, doc) {
        return doc.creatorId == userId
    },
    download: function (userId, doc) {
        return doc.creatorId == userId
    }
});
```

Danke an Raz für dieses hervorragende Beispiel. In der vollständigen CollectionFS-Dokumentation finden Sie weitere Informationen zu allen Funktionen von CollectionFS.

Server-Uploads

Die folgenden Skripts dienen zum Hochladen einer Datei vom Server-Dateisystem auf den Server. Meistens für Konfigurationsdateien und Filewatcher.

```
//https://forums.meteor.com/t/read-file-from-the-public-folder/4910/5

// Asynchronous Method.
Meteor.startup(function () {
```

```
    console.log('starting up');

    var fs = Npm.require('fs');
    // file originally saved as public/data/taxa.csv
    fs.readFile(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8', function (err,
data) {
        if (err) {
            console.log('Error: ' + err);
            return;
        }

        data = JSON.parse(data);
        console.log(data);
    });
});


// Synchronous Method.
Meteor.startup(function () {
    var fs = Npm.require('fs');
    // file originally saved as public/data/taxa.csv
    var data = fs.readFileSync(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8');

    if (Icd10.find().count() === 0) {
        Icd10.insert({
            date:  new Date(),
            data:  JSON.parse(data)
        });
    }
});


Meteor.methods({
  parseCsvFile:function (){
    console.log('parseCsvFile');

    var fs = Npm.require('fs');
    // file originally saved as public/data/taxa.csv
    var data = fs.readFileSync(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8');
    console.log('data', data);
  }
});
```

Datei hochladen online lesen: https://riptutorial.com/de/meteor/topic/3119/datei-hochladen

Kapitel 9: Daten veröffentlichen

Bemerkungen

In Meteors Datensubsystem sind eine Serverpublikation und die entsprechenden Clientabonnements die Hauptmechanismen für reaktiven Live-Datentransport, bei dem die zugrunde liegenden Daten ständig zwischen dem Server und dem Client synchronisiert werden.

Examples

Basis-Abonnement und Veröffentlichung

Entfernen Sie zunächst die `autopublish` . `autopublish` automatisch die gesamte Datenbank auf der Client-Seite, sodass die Auswirkungen von Publikationen und Abonnements nicht sichtbar sind.

So entfernen Sie die `autopublish` :

```
$ meteor remove autopublish
```

Dann können Sie Publikationen erstellen. Unten ist ein vollständiges Beispiel.

```
import { Mongo } from 'meteor/mongo';
import { Meteor } from 'meteor/meteor';

const Todos = new Mongo.Collection('todos');

const TODOS = [
  { title: 'Create documentation' },
  { title: 'Submit to Stack Overflow' }
];

if (Meteor.isServer) {
  Meteor.startup(function () {
    TODOS.forEach(todo => {
      Todos.upsert(
        { title: todo.title },
        { $setOnInsert: todo }
      );
    });
  });

  // first parameter is a name.
  Meteor.publish('todos', function () {
    return Todos.find();
  });
}

if (Meteor.isClient) {
  // subscribe by name to the publication.
  Meteor.startup(function () {
    Meteor.subscribe('todos');
  })
```

```
}
```

Globale Veröffentlichungen

Eine globale Publikation besitzt keinen Namen und erfordert kein Abonnement vom verbundenen Client. Daher steht sie dem verbundenen Client zur Verfügung, sobald der Client eine Verbindung zum Server herstellt.

Um dies zu erreichen, nennt man die Publikation einfach als `null`

```
Meteor.publish(null, function() {
  return SomeCollection.find();
})
```

Benannte Publikationen

Eine benannte Publikation besitzt einen Namen und muss explizit vom Client abonniert werden.

Betrachten Sie diesen serverseitigen Code:

```
Meteor.publish('somePublication', function() {
  return SomeCollection.find()
})
```

Der Kunde muss es anfordern durch:

```
Meteor.subscribe('somePublication')
```

Abonnements mit Vorlagenbereich

Meteors Standard-Templatiersystem Leertasten und das zugrunde liegende Rendering-Subsystem Blaze integriert sich nahtlos in Publikationslebenszyklusmethoden, sodass ein einfacher Vorlagencode seine eigenen Daten abonnieren und seine eigenen Spuren während des Abreißvorgangs der Vorlage stoppen und bereinigen kann.

Um dies zu erschließen, muss man die Template-Instanz abonnieren und nicht das `Meteor` Symbol wie folgt:

Richten Sie zuerst die Vorlage ein

```
<template name="myTemplate">
  We will use some data from a publication here
</template>
```

Tippen Sie dann auf den entsprechenden Lifecycle-Callback

```
Template.myTemplate.onCreated(function() {
  const templateInstance = this;
  templateInstance.subscribe('somePublication')
```

```
})
```

Wenn diese Vorlage nun zerstört wird, wird auch die Veröffentlichung automatisch gestoppt.

Hinweis: Die abonnierten Daten stehen allen Vorlagen zur Verfügung.

In einer ephemeren clientseitigen Sammlung veröffentlichen.

Für, wenn Sie das, was veröffentlicht wird, verfeinern müssen.

```
import { Mongo } from 'meteor/mongo';
import { Meteor } from 'meteor/meteor';
import { Random } from 'meteor/random';

if (Meteor.isClient) {
  // established this collection on the client only.
  // a name is required (first parameter) and this is not persisted on the server.
  const Messages = new Mongo.Collection('messages');
  Meteor.startup(function () {
    Meteor.subscribe('messages');
    Messages.find().observe({
      added: function (message) {
        console.log('Received a new message at ' + message.timestamp);
      }
    });
  })
}

if (Meteor.isServer) {
  // this will add a new message every 5 seconds.
  Meteor.publish('messages', function () {
    const interval = Meteor.setInterval(() => {
      this.added('messages', Random.id(), {
        message: '5 seconds have passed',
        timestamp: new Date()
      })
    }, 5000);
    this.added('messages', Random.id(), {
      message: 'First message',
      timestamp: new Date()
    });
    this.onStop(() => Meteor.clearInterval(interval));
  });
}
```

Fehler in einer Publikation erstellen und darauf reagieren

Auf dem Server können Sie eine solche Veröffentlichung erstellen. `this.userId` ist die ID des aktuell angemeldeten Benutzers. Wenn kein Benutzer angemeldet ist, möchten Sie möglicherweise einen Fehler `this.userId` und darauf reagieren.

```
import Secrets from '/imports/collections/Secrets';

Meteor.publish('protected_data', function () {
  if (!this.userId) {
```

```
    this.error(new Meteor.Error(403, "Not Logged In."));
    this.ready();
  } else {
    return Secrets.find();
  }
});
```

Auf dem Client können Sie mit den folgenden Antworten antworten.

```
Meteor.subscribe('protected_data', {
  onError(err) {
    if (err.error === 403) {
      alert("Looks like you're not logged in");
    }
  },
});
```

Datei / Importe / Sammlungen / Geheimnisse erstellt einen Verweis auf die Geheimnisse-Sammlung wie folgt:

```
const Secrets = new Mongo.Collection('secrets');
```

Erneutes Abonnieren einer Publikation

Ein Vorlagenautorun kann verwendet werden, um eine Publikation (erneut) zu abonnieren. Es wird ein reaktiver Kontext erstellt, der immer dann ausgeführt wird, wenn *reaktive Daten von Änderungen abhängen* . Außerdem wird ein Autorun immer einmal ausgeführt (beim ersten Ausführen).

Vorlagenautoruns werden normalerweise in eine `onCreated` Methode geschrieben.

```
Template.myTemplate.onCreated(function() {
  this.parameter = new ReactiveVar();
  this.autorun(() => {
    this.subscribe('myPublication', this.parameter.get());
  });
});
```

Dies wird einmalig (das erste Mal) ausgeführt und ein Abonnement eingerichtet. Sie wird dann erneut ausgeführt, wenn sich die `parameter` Blindgröße ändert.

Warten Sie in der Blaze-Ansicht, während veröffentlichte Daten abgerufen werden

Template JS-Code

```
Template.templateName.onCreated(function(){
    this.subscribe('subsription1');
    this.subscribe('subscription2');
});
```

Vorlagen-HTML-Code

```
<template name="templateName">
    {{#if Template.subscriptionsReady }}
        //your actual view with data. it can be plain HTML or another template
    {{else}}
        //you can use any loader or a simple header
        <h2> Please wait ... </h2>
    {{/if}}
</template>
```

Überprüfen des Benutzerkontos beim Veröffentlichen

Manchmal ist es eine gute Idee, Ihre Veröffentlichungen weiter zu schützen, indem Sie ein Benutzer-Login erfordern. So erreichen Sie dies über Meteor.

```
import { Recipes } from '../imports/api/recipes.js';
import { Meteor } from 'meteor/meteor';

Meteor.publish('recipes', function() {
  if(this.userId) {
    return Recipe.find({});
  } else {
    this.ready();  // or: return [];
  }
});
```

Veröffentlichen Sie mehrere Cursor

Mehrere Datenbankcursor können aus derselben Publikationsmethode veröffentlicht werden, indem ein Array von Cursor zurückgegeben wird.

Die "Kinder" -Cursors werden als Joins behandelt und sind nicht reaktiv.

```
Meteor.publish('USER_THREAD', function(postId) {
  let userId   = this.userId;

  let comments = Comments.find({ userId, postId });
  let replies  = Replies.find({ userId, postId });

  return [comments, replies];
});
```

Simulieren Sie die Verzögerung in Publikationen

In der `Meteor._sleepForMs(ms);` Verzögerungen in Verbindung und Server auftreten, um Verzögerungen in der Entwicklungsumgebung zu simulieren. `Meteor._sleepForMs(ms);` könnte verwendet werden

```
Meteor.publish('USER_DATA', function() {
    Meteor._sleepForMs(3000); // Simulate 3 seconds delay
    return Meteor.users.find({});
```

```
});
```

Publikationen zusammenführen

Publikationen können auf dem Client zusammengeführt werden, was zu unterschiedlich geformten Dokumenten innerhalb eines einzelnen Cursors führt. Das folgende Beispiel zeigt, wie ein Benutzerverzeichnis eine minimale Menge öffentlicher Daten für Benutzer einer App veröffentlicht und ein detaillierteres Profil für den angemeldeten Benutzer bereitstellt.

```
// client/subscriptions.js
Meteor.subscribe('usersDirectory');
Meteor.subscribe('userProfile', Meteor.userId());

// server/publications.js
// Publish users directory and user profile

Meteor.publish("usersDirectory", function (userId) {
    return Meteor.users.find({}, {fields: {
        '_id': true,
        'username': true,
        'emails': true,
        'emails[0].address': true,

        // available to everybody
        'profile': true,
        'profile.name': true,
        'profile.avatar': true,
        'profile.role': true
    }});
});
Meteor.publish('userProfile', function (userId) {
    return Meteor.users.find({_id: this.userId}, {fields: {
        '_id': true,
        'username': true,
        'emails': true,
        'emails[0].address': true,

        'profile': true,
        'profile.name': true,
        'profile.avatar': true,
        'profile.role': true,

        // privately accessible items, only availble to the user logged in
        'profile.visibility': true,
        'profile.socialsecurity': true,
        'profile.age': true,
        'profile.dateofbirth': true,
        'profile.zip': true,
        'profile.workphone': true,
        'profile.homephone': true,
        'profile.mobilephone': true,
        'profile.applicantType': true
    }});
});
```

Daten veröffentlichen online lesen: https://riptutorial.com/de/meteor/topic/1323/daten-veroffentlichen

Kapitel 10: Daten von einem Meteor.call abrufen

Examples

Die Grundlagen von Meteor.call

```
Meteor.call(name, [arg1, arg2...], [asyncCallback])
```

(1) Name Zeichenfolge
(2) Name der aufzurufenden Methode
(3) arg1, arg2 ... EJSON-fähiges Objekt [optional]
(4) asyncCallback-Funktion [optional]

Zum einen können Sie: (über die **Session- Variable** oder über die **ReactiveVar**)

```
    var syncCall = Meteor.call("mymethod") // Sync call
```

Das bedeutet, wenn Sie so etwas tun, werden Sie auf der Serverseite Folgendes tun:

```
    Meteor.methods({
        mymethod: function() {
            let asyncToSync =  Meteor.wrapAsync(asynchronousCall);
            // do something with the result;
            return  asyncToSync;
        }
    });
```

Auf der anderen Seite möchten Sie es manchmal über das Ergebnis des Rückrufs behalten?

Kundenseite:

```
Meteor.call("mymethod", argumentObjectorString, function (error, result) {
    if (error) Session.set("result", error);
    else Session.set("result",result);
}
Session.get("result") -> will contain the result or the error;

//Session variable come with a tracker that trigger whenever a new value is set to the session
variable. \ same behavior using ReactiveVar
```

Serverseite

```
Meteor.methods({
    mymethod: function(ObjectorString) {
        if (true) {
            return true;
        } else {
```

```
            throw new Meteor.Error("TitleOfError", "ReasonAndMessageOfError"); // This will
and up in the error parameter of the Meteor.call
        }
    }
});
```

Hier soll gezeigt werden, dass Meteor verschiedene Möglichkeiten für die Kommunikation zwischen dem Client und dem Server vorschlägt.

Session-Variable verwenden

Serverseite

```
Meteor.methods({
  getData() {
    return 'Hello, world!';
  }
});
```

Client-Seite

```
<template name="someData">
  {{#if someData}}
    <p>{{someData}}</p>
  {{else}}
    <p>Loading...</p>
  {{/if}}
</template>
```

```
Template.someData.onCreated(function() {
  Meteor.call('getData', function(err, res) {
    Session.set('someData', res);
  });
});

Template.someData.helpers({
  someData: function() {
    return Session.get('someData');
  }
});
```

ReactiveVar verwenden

Serverseite

```
Meteor.methods({
  getData() {
    return 'Hello, world!';
  }
});
```

Client-Seite

```
<template name="someData">
  {{#if someData}}
    <p>{{someData}}</p>
  {{else}}
    <p>Loading...</p>
  {{/if}}
</template>
```

```
Template.someData.onCreated(function() {

  this.someData = new ReactiveVar();

  Meteor.call('getData', (err, res) => {
    this.someData.set(res);
  });
});

Template.someData.helpers({
  someData: function() {
    return Template.instance().someData.get();
  }
});
```

`reactive-var` Paket erforderlich. Um es hinzuzufügen, führen Sie den `meteor add reactive-var` .

Daten von einem Meteor.call abrufen online lesen:
https://riptutorial.com/de/meteor/topic/3068/daten-von-einem-meteor-call-abrufen

Kapitel 11: Debuggen

Examples

Browser-Debugger

Sowohl Chrome als auch Safari verfügen über eingebaute Debugger. Bei Chrome müssen Sie lediglich mit der rechten Maustaste auf eine Webseite und auf "Element prüfen" klicken. Bei Safari müssen Sie auf Einstellungen> Erweitert gehen und auf "Entwicklungsmenü in Menüleiste anzeigen" klicken.

Mit Firefox müssen Sie Firebug installieren

Fügen Sie Ihrer App Debugger-Haltepunkte hinzu

Sie müssen Ihrem Code `debugger` Anweisungen hinzufügen:

```
Meteor.methods({
  doSomethingUself: function(){
    debugger;
    niftyFunction();
  }
});
```

Serverseitiges Debuggen mit Knoteninspektor

Für das serverseitige Debugging müssen Sie ein Tool wie Node Inspector verwenden. Bevor Sie loslegen, lesen Sie einige dieser nützlichen Tutorials.

HowToNode - Debuggen mit Knoteninspektor
Strongloop - Debugging-Anwendungen
Meteor.js- Walkthrough mit Screenshots zum Verwenden von Node Inspector mit Meteor ganz einfach

tl; dr - Es gibt eine Reihe von Dienstprogrammen im Meteor-Ökosystem, die zur gleichen Zeit wie Ihre Meteor-Anwendung ausgeführt werden sollen. Sie funktionieren nur, wenn Ihre Meteor-App läuft und sie eine Verbindung zu einer laufenden Website herstellen können. meteor mongo, Robomongo, Nightwatch ... all dies sind Dienstprogramme, die Ihre Anwendung bereits benötigen. NodeInspector ist eines dieser Dienstprogramme.

```
# install node-inspector
terminal-a$  npm install -g node-inspector

# start meteor
terminal-a$  NODE_OPTIONS='--debug-brk --debug' mrt run

# alternatively, some people report this syntax being better
terminal-a$  sudo NODE_OPTIONS='--debug' ROOT_URL=http://myapp.com meteor --port 80
```

```
# launch node-inspector along side your running app
terminal-b$  node-inspector

# go to the URL given by node-inspector
http://localhost:8080/debug?port=5858
```

Serverseitiges Debuggen mit npm debuggen

Neben Node Inspector berichteten einige Benutzer über den Erfolg eines npm-Dienstprogramms namens `debug` .

MeteorHacks - Debugging von Meteor mit npm debug

Meteor Shell

Ab Meteor 1.0.2 gibt es eine neue Befehlsshell, mit der Sie interaktives Debugging durchführen und Ihre App von der Serverseite aus verwalten können, genau wie mit der Chrome-Konsole auf der Clientseite. Hör zu:

```
meteor shell
```

Andere Debugging-Dienstprogramme

Meteor-Dump
Meteorspielzeug
Konstellation

Meteor DevTools

Debuggen online lesen: https://riptutorial.com/de/meteor/topic/3378/debuggen

Kapitel 12: Electrify - Meteor als lokal installierbare App kompilieren

Examples

Installieren von Electrify für eine Meteor-Anwendung

Electron portiert HTML-Webanwendungen zu nativen Anwendungen für eine Reihe von Geräten, einschließlich der Erstellung nativer Desktopanwendungen. Es ist auch sehr einfach zu beginnen!

Um zu beginnen, müssen wir haben `electron` , `nodejs` , `npm` , `git` und `meteor` installiert. Die Kenntnis dieser Tools ist für die Arbeit mit Meteor von entscheidender Bedeutung. Stellen Sie also sicher, dass Sie diese Dinge zuerst kennen.

Elektron

```
npm install -g electrify
```

- `electron` wir! Lesen Sie hier mehr.
- `electrify` ist ein Tool zum Verpacken von Meteor-Apps. Lesen Sie hier den Modus.

Weitere Anforderungen für die Installation und Verwendung von Electrify with Meteor

Meteor

```
curl https://install.meteor.com/ | sh
```

Es gibt viele Möglichkeiten, Meteor zu installieren, siehe hier .

- `meteor` ist das JavaScript-Framework, mit dem wir unsere Anwendung erstellen. Es bietet uns viele Codiervereinfachungen für einige konzeptionell schwierige Probleme in Webanwendungen. Seine Einfachheit wurde als hilfreich für prototypische Projekte bezeichnet. Lesen Sie hier mehr.

NodeJS

```
apt-get install nodejs build-essentials
```

Es gibt verschiedene Installationsmöglichkeiten, abhängig von Ihrem Betriebssystem. Finden Sie hier heraus, welchen Weg Sie benötigen.

- `nodejs` ist das Paket für Node.js, eine Javascript-Umgebung zum Ausführen von JavaScript auf der Serverseite. Lesen Sie hier mehr.

npm

`npm` sollte in der `nodejs` Installation enthalten sein. Überprüfen Sie dies, indem `nodejs` nach der Installation von `nodejs` den Befehl `npm -v nodejs` .

- `npm` ist der Node Package Manager. Es ist eine riesige Sammlung von Open-Source-Modulen, die Sie problemlos in Ihre Node-Projekte einfügen können. Lesen Sie hier mehr.

Verwenden von Electrify bei einer Meteoranwendung

Laden wir ein Meteor Todos-Beispielprojekt mit einem Linux-Shell-Skript (Befehlszeile) herunter, um das erste Mal das Elektrifizieren eines Projekts zu testen:

Voraussetzungen für diesen Abschnitt:

Git

```
apt-get install git-all
```

Es gibt viele Möglichkeiten, Git zu installieren. Schau sie dir hier an .

- `git` ist ein Versionskontrollsystem für Dateien. Sie können remote (dh online) in öffentlichen Repositorys (GitHub ist ein ziemlich bekannter Repository) oder privaten Repositorys (BitBucket stellt beispielsweise eine begrenzte Anzahl kostenloser privater Repositorys bereit). Lesen Sie mehr [hier] [5].

```
#!/usr/bin/bash

# Change this parameter to choose where to clone the repository to.
TODOSPATH="/home/user/development/meteor-todos"

# Download the repository to the $TODOSPATH location.
git clone https://github.com/meteor/todos.git "$TODOSPATH"

# Change directory (`cd`) into the Todos project folder.
cd "$TODOSPATH"
```

Wir sollten jetzt einen Projektordner mit dem Namen "meteor-todos" an der im Parameter TODOSPATH angegebenen Position haben. Wir haben auch das Verzeichnis (`cd`) in den Projektordner geändert. `cd` wir also Electrify zu diesem Projekt hinzu!

```
# It's really this simple.
electrify
```

Das ist richtig - ein Befehl mit nur einem Wort, und unser Projekt ist fertig. Berechtigungen können Fehler verursachen, wenn Sie versuchen, `electrify` als Befehl auszuführen. In diesem Fall versuchen Sie `sudo electrify` , um die Berechtigungen zu überschreiben.

Versuchen Sie jedoch, diese Berechtigungsprobleme zu lösen - es ist nicht ratsam, unnötig `sudo`

(worauf ich eingehen würde, aber ich könnte ein ganz anderes Thema darüber schreiben, warum das so ist!)

Electrify - Meteor als lokal installierbare App kompilieren online lesen: https://riptutorial.com/de/meteor/topic/2526/electrify---meteor-als-lokal-installierbare-app-kompilieren

Kapitel 13: Entwicklungswerkzeuge

Examples

Integrierte Entwicklungsumgebungen

Die Entwicklung beginnt normalerweise mit einem Editor oder einer integrierten Entwicklungsumgebung. Die folgenden IDEs unterstützen Meteor in gewissem Umfang:

- Atom - Javascript IDE, die das isomorphe Javascript-Framework von Meteor voll nutzen kann. Wenn Sie Ihren Editor selbst hacken möchten, wählen Sie dies aus.
- Cloud9 - Das neueste Cloud-Entwicklungsangebot, das Meteor unterstützt, mit einem Tutorial.
- MeteorDevTools - Chrome-Erweiterung für Blaze, DDP und Minimongo.
- Sublime - Leichter und beliebter Texteditor.
- WebStorm - Die umfassendste IDE, die derzeit für Meteor verfügbar ist.

Datenbank-Tools

Sobald Sie an Ihrer 'Hello World'-App vorbeigefahren sind, müssen Sie sich um Ihre Sammlungs- und Dokumentierungsschemas kümmern und benötigen einige Tools zur Verwaltung Ihrer Datenbank.

- Robomongo - Ein langjähriger Community-Favorit für die Verwaltung von Mongo. Sehr empfehlenswert.
- JSON Generator - Unverzichtbares Hilfsprogramm zum Generieren von Beispieldatensätzen.
- MacOSX Mongo-Voreinstellungsseite - Voreinstellungen GUI für MacOSX.
- MongoHub - Eine weitere Mongo GUI, ähnlich wie RoboMongo Nur MacOSX.
- Mongo3 - Eines der wenigen Cluster-Management-Tools. Kann Replikationssätze visualisieren. Einziger Nachteil ist es in Ruby gebaut.
- Mongo Monitoring Service - Wenn Sie bereit sind, etwas in die Produktion zu bringen, ist MMS von unschätzbarem Wert. Jetzt als MongoDB-Atlas bekannt.
- Mongo Express - Webbasierte MongoDB-Administrationsoberfläche, geschrieben mit Node.js und express

Remote-Collaboration-Dienstprogramme für verteilte Entwickler

Die Entwicklung von Meteor-Apps bedeutet in der Regel die Entwicklung von Multi-Client-Reaktivität, was Collaboration-Tools erfordert. Die folgenden Tools haben sich in der Meteor-Community als beliebt erwiesen.

- Google Hangouts - Videokonferenzen und Chat.
- Zenhub.io - Kanban-Boards für GitHub.
- InVision - kollaboratives Wireframing und Prototyping.

- Meeting Hero - Kollaborative Meetingplanung.
- Hackpad - Collaborative Dokumentenbearbeitung.
- Slack - Kollaborative Projektverfolgungs-Feeds.
- MadEye - kollaborativer Web-Editor.
- Screenhero - Gemeinsame Bildschirmfreigabe.
- Proto.io - Wireframing und Prototyping.
- HuBoard - Kanban-Boards für GitHub.
- Zapier - Die besten Apps. Zusammen.
- Teamwork.com - Traditionelles Projektmanagement und Diagramme.
- Sprint.ly - Weitere Kanban-Boards und Sprint-Planung, die mit GitHub funktionieren.
- LucidChart - Online-Visio-Alternative.
- Waffle.io - Trello / ZenHub-Alternative, die in GitHub integriert ist.

REST-Clients

Wenn Sie Meteor mit einer externen API integrieren möchten, wird es wahrscheinlich als REST-Schnittstelle verfügbar gemacht. Wir neigen dazu, die folgenden Chrome-Apps zum Testen von REST-APIs zu verwenden.

- Briefträger
- DHC Rest Client

Online-Tools:

- Hurl.it
- RequestBin

Debugger

Das Debugging findet hauptsächlich im Terminal oder in den Chrome- oder Safari-Entwicklungstools statt, die für 99% Ihrer Anforderungen ausgereift genug sind. Wenn Sie jedoch unter Firefox debuggen möchten oder zusätzliche Server-Debugging-Funktionen benötigen, können Sie einige zusätzliche Dienstprogramme verwenden.

- Firefox - Feuerbug
- Knoteninspektor
- Meteor Toys oder direkt `meteor add meteortoys:allthings`

Mobile Codierung unter iOS

Texttastic Code Editor - Code-Editor mit Syntaxhervorhebung für iOS-Geräte.

Arbeitskopie - Klonen Sie Github-Repositorys auf Ihr iPad und codieren Sie unterwegs.

CodeHub - Durchsuchen und verwalten Sie Ihre GitHub-Repositorys. Management-Tool

iOctocat - Sozialer Nutzen für die Verfolgung von Github-Projekten.

iMockups für iPad - Wireframes und Mockups. Unterstützt Wireframes für Desktops und Mobile.

Blueprint - iOS Wireframing und Mockups. Hauptsächlich für iOS-Entwicklung, aber etwas für Web-Apps geeignet.

JSON Designer - Datenarchitektur und Datenschemaentwurf.

Entwicklungswerkzeuge online lesen:
https://riptutorial.com/de/meteor/topic/4200/entwicklungswerkzeuge

Kapitel 14: ES2015-Module (Import und Export)

Bemerkungen

MDN-Dokumentation für Importe: https://developer.mozilla.org/de/docs/web/javascript/reference/statements/import MDN-Dokumentation für Exporte: https://developer.mozilla.org/de/docs/web/javascript / reference / statement / export ExploringJS-Kapitel zu Modulen: http://exploringjs.com/es6/ch_modules.html

Examples

Importieren in App-Modulen

Knotenmodule

```
import url from 'url';
import moment from 'moment';
```

Meteorpakete

```
import { Meteor } from 'meteor/meteor';
import { SimpleSchema } from 'meteor/aldeed:simple-schema';
```

Importieren in Meteor-Paketen

In package.js:

```
Npm.depends({
  moment: "2.8.3"
});
```

In einer Paketdatei:

```
import moment from 'moment';
```

Variablen aus App-Modulen exportieren

```
// Default export
export default {};

// Named export
export const SomeVariable = {};
```

Symbole aus Meteor-Paketen exportieren

In Ihrer mainModule- Datei:

```
export const SomeVar = {};
```

ES2015-Module (Import und Export) online lesen:
https://riptutorial.com/de/meteor/topic/3763/es2015-module--import-und-export-

Kapitel 15: ESLint

Examples

Eslint zu Ihrem Meteor-Projekt hinzufügen

Wir werden die beliebte `eslint-config-airbnb` als Starter sowie Meteor-spezifische Regeln verwenden, die `eslint-import-resolver-meteor` .

Wir müssen auch `babel-parser` installieren, um Meteor-fähige ES7-Funktionen wie async / await zu nutzen.

```
cd my-project
npm install --save-dev eslint-config-airbnb eslint-plugin-import eslint-plugin-react eslint-
plugin-jsx-a11y eslint babel-eslint eslint-import-resolver-meteor
touch .eslintrc.json
```

Dann verwenden Sie einfach diese Boilerplate `.eslintrc.json` um loszulegen. Sie können die Regeln nach Belieben überschreiben.

```
{
  "parser": "babel-eslint",
  "settings": {
    "import/resolver": "meteor"
  },
  "extends": "airbnb",
  "rules": {}
}
```

Verwenden Sie ein npm-Skript, um Ihren Code einzufahren

Bearbeiten Sie Ihre `package.json` , um das folgende Skript hinzuzufügen:

```
{
  "scripts": {
    "lint": "eslint .;exit 0"
  }
}
```

Dann führe es mit `npm run lint`

Wir verwenden `exit 0` als Trick, um das Skript `eslint` zu beenden, wenn das Flusen fehlschlägt. Andernfalls verwendet `npm` den `eslint` Rückkehrcode und stürzt ab.

ESLint online lesen: https://riptutorial.com/de/meteor/topic/3772/eslint

Kapitel 16: Grundlegendes Codeship-Setup für automatisiertes Testen

Examples

Codeship einrichten

- Gehen Sie zu Codeship.com und erstellen Sie ein Konto (oder Login).
- Erstellen Sie ein neues Projekt
- Importieren Sie Ihr Projekt über Github oder Bitbucket
- Verwenden Sie auf dem Bildschirm "Konfigurieren Sie Ihre Tests" die folgenden Befehle:
    - Wählen Sie "Ich möchte meine eigenen benutzerdefinierten Befehle erstellen" im Dropdown-Menü "Wählen Sie Ihre Technologie aus, um grundlegende Befehle vorab aufzufüllen".
    - Geben Sie die folgenden Befehle ein:

```
curl -o meteor_install_script.sh https://install.meteor.com/
chmod +x meteor_install_script.sh
sed -i "s/type sudo >\/dev\/null 2>&1/\ false /g" meteor_install_script.sh
./meteor_install_script.sh
export PATH=$PATH:~/.meteor/
meteor --version
meteor npm install
```

    - Lassen Sie die Testbefehle wie folgt:

```
npm test
```

- Ein neues Commit an Github / Bitbucket senden
- Das ist es

Bereiten Sie das Projekt vor

- Schreibe ein paar Tests
- Installieren Sie Versand: Mokka-Phantomjs :

```
meteor add dispatch:mocha-phantomjs
```

- Fügen Sie Ihrem package.json einen Testbefehl hinzu.

```
{
  "name": "awesome meteor package",
  "scripts": {
    "test": "meteor test --driver-package dispatch:mocha-phantomjs --once"
  }
```

```
}
```

- Stellen Sie sicher, dass Sie `npm test` in Ihrem Projektstamm ausführen können.

Grundlegendes Codeship-Setup für automatisiertes Testen online lesen: https://riptutorial.com/de/meteor/topic/6741/grundlegendes-codeship-setup-fur-automatisiertes-testen

Kapitel 17: Hintergrundaufgaben

Bemerkungen

Das Paket **cron-tick** ist ein sehr einfaches Paket für Hintergrundaufgaben, unterstützt jedoch nicht mehrere Prozesse. Wenn Sie Ihre App in mehreren Prozessen (oder Containern) ausführen, verwenden Sie stattdessen **percolate: synced-cron** .

Examples

Einfacher Cron

Verwenden Sie das Paket **percolate: synced-cron**

Einen Job definieren:

```
 SyncedCron.add({
  name: 'Find new matches for a saved user filter and send alerts',
  schedule: function(parser) {
    // parser is a later.parse object
    return parser.text('every 10 minutes');
  },
  job: function() {
    user.alerts.map(a => a.findMatchesAndAlert());
  }
});
```

Starten Sie Ihre definierten Jobs:

```
SyncedCron.start();
```

Es unterstützt das Synchronisieren von Jobs zwischen mehreren Prozessen, z. B. Galaxy mit mehr als einem Container.

Hintergrundaufgaben online lesen:
https://riptutorial.com/de/meteor/topic/4772/hintergrundaufgaben

Kapitel 18: Horizontale Skalierung

Examples

Bereitstellen einer Anwendung mit separater Datenbank (MONGO_URL)

Sie müssen Ihre Anwendungsebene von Ihrer Datenbankebene trennen, und dies bedeutet, dass Sie MONGO_URL angeben. Dies bedeutet, dass Sie Ihre App über den Bundle-Befehl ausführen, dekomprimieren, Umgebungsvariablen festlegen und das Projekt dann als Knoten-App starten. Hier ist wie...

```
#make sure you're running the node v0.10.21 or later
npm cache clean -f
npm install -g n
sudo n 0.10.21

# bundle the app
mkdir myapp
cd myapp
git clone http://github.com/myaccount/myapp
meteor bundle --directory ../deployPath
cd ../deployPath

# make sure fibers is installed, as per the README
export MONGO_URL='mongodb://127.0.0.1:27017/mydatabase'
export PORT='3000'
export ROOT_URL='http://myapp.com'

# run the site
node main.js
```

Replikatset-Konfiguration

Gehen Sie dann in die Mongo-Shell und initiieren Sie den Replikatsatz,

```
mongo

> rs.initiate()
PRIMARY> rs.add("mongo-a")
PRIMARY> rs.add("mongo-b")
PRIMARY> rs.add("mongo-c")
PRIMARY> rs.setReadPref('secondaryPreferred')
```

Konfigurieren eines Replikatsatzes zur Verwendung von Oplogging

Der Replikatsatz benötigt einen Oplog-Benutzer, um auf die Datenbank zuzugreifen.

```
mongo

PRIMARY> use admin
PRIMARY>
```

```
db.addUser({user:"oplogger",pwd:"YOUR_PASSWORD",roles:[],otherDBRoles:{local:["read"]}});
PRIMARY> show users
```

Oplog Upstart-Skript

Ihr Upstart-Skript muss geändert werden, um mehrere IP-Adressen des Replikatsatzes zu verwenden.

```
start on started mountall
stop on shutdown

respawn
respawn limit 99 5

script
    # our example assumes you're using a replica set and/or oplog integreation
    export MONGO_URL='mongodb://mongo-a:27017,mongo-b:27017,mongo-c:27017/meteor'

    # here we configure our OPLOG URL
    export MONGO_OPLOG_URL='mongodb://oplogger:YOUR_PASSWORD@mongo-a:27017,mongo-
b:27017,mongo-c:27017/local?authSource=admin'

    # root_url and port are the other two important environment variables to set
    export ROOT_URL='http://myapp.mydomain.com'
    export PORT='80'

    exec /usr/local/bin/node /var/www/production/main.js >> /var/log/node.log 2>&1
end script
```

Scherben

Oplog Tailing auf Sharded Mongo

Horizontale Skalierung online lesen: https://riptutorial.com/de/meteor/topic/3706/horizontale-skalierung

Kapitel 19: Integration von Drittanbieter-APIs

Examples

Einfacher HTTP-Aufruf

Die Integration von REST-APIs von Drittanbietern kann konzeptionell so einfach sein wie das Hinzufügen des `http` Pakets und das Aufrufen eines externen Endpunkts.

```
meteor add http
```

```
HTTP.get('http://foo.net/api/bar/');
```

Erstellen Sie ein Paket für Ihren API-Wrapper

Einfache HTTP-Aufrufe bieten jedoch keine Wiederverwendbarkeit von Code. Und sie können mit allen anderen Funktionen, die Sie implementieren möchten, verwirrt werden. Aus diesen Gründen ist es üblich, einen API-Wrapper zu implementieren.

```
Foo = {
  identify: function(input){
    return Http.get('http://foo.net/api/identify/' + input);
  },
  record_action_on_item: function(firstInput, secondInput){
    return Http.put('http://foo.net/api/record_action_on_item/' + firstInput + '&' +
secondInput);
  }
}
```

Meteor unterstützt Http.get (), Http.post (), Http.put () usw. usw. Dies ist zweifellos der beste Weg, um Ihre REST-API aufzurufen. http://docs.meteor.com/#http_get

Wenn die API gesprächig und ausführlich ist, erhalten Sie möglicherweise mehrere Pakete. In diesem Fall müssen Sie sie wieder zusammenbauen. Dies ist ein großer Ärger. Wenn Sie der Meinung sind, dass die API mehrere Pakete zurückgibt, möchten Sie wahrscheinlich das npm-Modul "request" auf dem Server verwenden. Sie möchten eine `Npm.require('request')` . https://github.com/mikeal/request

Erstellen Sie ein Atmosphere-Paket für Ihren API-Wrapper

Nach dem Erstellen eines API-Wrappers möchten Sie möglicherweise ein Atmosphere-Paket erstellen, um es neu zu verteilen und zwischen Anwendungen zu teilen. Die Dateien Ihres Pakets werden wahrscheinlich so aussehen.

```
packages/foo-api-wrapper/package.js
packages/foo-api-wrapper/readme.md
packages/foo-api-wrapper/foo.api.wrapper.js
```

Insbesondere möchte `foo-api-wrapper/package.js` Datei `foo-api-wrapper/package.js` etwa wie folgt aussehen:

```
Package.describe({
  summary: "Atmosphere package that impliments the Foo API.",
  name: "myaccount:foo",
  version: '0.0.1'
});

Package.on_use(function (api) {
    api.export('Foo');
    api.addFiles('foo.api.wrapper.js', ["client","server"]);
});
```

Ihr `foo-api-wrapper/foo.api.wrapper.js` sollte das `Foo` API- `foo-api-wrapper/foo.api.wrapper.js` enthalten.

Binden Sie das API-Paket in Ihre Anwendung ein

An diesem Punkt erstellen Sie immer noch Ihr Paket. Sie müssen das Paket also Ihrer Anwendung hinzufügen:

```
meteor add myaccount:foo
```

Und schließlich veröffentlichen Sie es bei Atmosphere:

```
meteor publish myaccount:foo
```

Verwenden des API-Wrapper-Objekts in Ihrer App

Nun, da wir all diese Teile zusammengefügt haben, sollten Sie jetzt wie folgt Anrufe aus Ihrer App heraus tätigen können:

```
Foo.identify('John');
Foo.record_action_on_item('view', "HackerNews');
```

Natürlich sollten Sie Funktionsnamen, Argumente, URLs und Ähnliches anpassen, um die richtige Syntax für die API zu erstellen.

Integration von Drittanbieter-APIs online lesen:
https://riptutorial.com/de/meteor/topic/3118/integration-von-drittanbieter-apis

Kapitel 20: Knoten / NPM

Examples

Meteor getestete / unterstützte Knotenversion

Um die letzte getestete / unterstützte Version von Node zu ermitteln, die mit Ihrer installierten Version von Meteor verwendet werden kann, sichern Sie die Knotenversion direkt von der gebündelten Knoteninstanz des Build-Tools.

```
meteor node -v
```

Knoten / NPM online lesen: https://riptutorial.com/de/meteor/topic/4599/knoten---npm

Kapitel 21: Kontinuierliche Bereitstellung von Codeship für Galaxy

Bemerkungen

Dieses Thema ist stark von Nate Strausers inspiriert, die Meteor-Apps von Modulus auf Galaxy mit kontinuierlicher Bereitstellung von Codeship migrieren .

Examples

Konfiguration

- Erstellen Sie ein `deployment_token.json` :

```
METEOR_SESSION_FILE=deployment_token.json meteor login
```

- Erstellen Sie die folgenden Umgebungsvariablen in Codeship: (https://codeship.com/projects/PROJECT_NUMBER/configure_environment)
    - METEOR_TARGET: Ihre.Domäne.com
    - METEOR_TOKEN: Kopieren / Einfügen des Inhalts von deploy_token.json. Etwas wie: `{"sessions": {"www.meteor.com": {"session": "12345 ...`
    - METEOR_SETTING: Kopieren Sie den Inhalt Ihrer settings.json. So etwas wie: `{"private": {...`
- Erstellen Sie hier eine neue Implementierungspipeline https://codeship.com/projects/YOUR_PROJECT_NUMBER/deployment_branches/new
    - Wir setzen nur den Master-Zweig ein. Also eingestellt: Branch ist genau: Master.
- Fügen Sie als Bereitstellung ein "benutzerdefiniertes Skript" mit folgendem Inhalt hinzu:

```
echo $METEOR_TOKEN > deployment_token.json
echo $METEOR_SETTINGS > deployment_settings.json
meteor npm prune --production
DEPLOY_HOSTNAME=galaxy.meteor.com METEOR_SESSION_FILE=deployment_token.json meteor deploy
$METEOR_TARGET --settings deployment_settings.json
```

Kontinuierliche Bereitstellung von Codeship für Galaxy online lesen:
https://riptutorial.com/de/meteor/topic/6743/kontinuierliche-bereitstellung-von-codeship-fur-galaxy

Kapitel 22: Leistungsoptimierung

Bemerkungen

Es ist zu beachten, dass Meteor einfach Javascript und Node.js ist. Ja, es handelt sich um eine sehr spezifische Implementierung dieser beiden Technologien. Sie verfügt über ein eigenes einzigartiges Ökosystem und nutzt isomorphe APIs und einen JSON-Datastore, um wirklich erstaunliche Ergebnisse zu erzielen. Am Ende des Tages ist Meteor eine Web-Technologie, die in Javascript geschrieben ist. Daher gelten alle Ihre typischen Javascript-Leistungstechniken weiterhin. Beginnen Sie dort.

25 Javascript Performance-Techniken
Leistungsoptimierungen für schnelles Javascript
JavaScript-Code optimieren
Leistungstipps für JavaScript in V8
10 Tipp zur Leistungssteigerung von Javascript
Schreiben Sie effizientes JavaScript
Verbesserung der Leistung Ihrer Meteor JS-Projekte

Examples

Entwerfen und Bereitstellen von produktionsbereiter Software

Denken Sie daran, dass alle bewährten Methoden der typischen Webarchitektur weiterhin gelten. Einen hervorragenden Überblick zu diesem Thema finden Sie in Michael Nygards hervorragendem Buch Release It! Entwickeln und Bereitstellen von produktionsbereiter Software . Wenn Sie Ihre App in Meteor schreiben, müssen Sie keine Bibliotheken von Drittanbietern überprüfen, Leitungsschutzschalter schreiben, Anrufe in Timeouts einschließen, Ihre Ressourcenpools überwachen und alles andere. Wenn Sie möchten, dass Ihre Anwendung eine gute Leistung bringt, müssen Sie sicherstellen, dass Sie Stabilitätsmuster verwenden und Anti-Patterns vermeiden.

Stabilitätsmuster

- Timeouts
- Leistungsschalter
- Schotte
- Händeschütteln
- Stabilität Anti-Patterns

Integrationspunkte

- Drittanbieter-Bibliotheken
- Skalierungseffekte
- Unausgeglichene Fähigkeiten

- Kapazität Anti-Patterns

Ressourcenpoolkonflikt

- AJAX Overkill
- Überstehende Sitzungen
- Übermäßiger weißer Raum
- Eutropification von Daten

Wenn diese Konzepte nicht vertraut sind und Ihnen nicht wie selbstverständlich erscheinen, haben Sie keine größeren Produktionssysteme online gestellt. Kaufen Sie eine Kopie des Buches. Es wird Zeit und Geld gut angelegt sein.

Leistungsoptimierung online lesen:
https://riptutorial.com/de/meteor/topic/3363/leistungsoptimierung

Kapitel 23: Meteor + React + ReactRouter

Einführung

Dieses Dokument zeigt, wie Sie ReactRouter mit Meteor und React verwenden. Von Null bis zu einer funktionierenden App, einschließlich Rollen und Authentifizierung.

Ich zeige jeden Schritt mit einem Beispiel

1- Erstellen Sie das Projekt

2- Fügen Sie React + ReactRouter hinzu

3- Konten hinzufügen

4- Fügen Sie Rollenpakete hinzu

Examples

Erstellen Sie das Projekt

1- Installieren Sie zunächst https://www.meteor.com/install

2- Erstellen Sie ein Projekt. (`--bare` soll ein leeres Projekt erstellen)

```
meteor create --bare MyAwesomeProject
```

3- Erstellen Sie die minimale Dateistruktur (`-p` , um Zwischenverzeichnisse zu erstellen):

```
cd MyAwesomeProject

mkdir -p client server imports/api imports/ui/{components,layouts,pages}
imports/startup/{client,server}
```

4- Erstellen Sie nun eine HTML-Datei in client / main.html

```
<head>
   <meta charset="utf-8">
   <title>My Awesome Meteor_React_ReactRouter_Roles App</title>
</head>

<body>
  Welcome to my Meteor_React_ReactRouter_Roles app
</body>
```

5- Stellen Sie sicher, dass es funktioniert: (3000 ist der Standardport, sodass Sie '-p 3000' tatsächlich überspringen können)

```
meteor run -p 3000
```

und öffnen Sie Ihren Browser unter 'localhost: 3000'

Hinweis:

- Ich überspringe einige andere Dateien, die Sie erstellen müssen, um die Dinge zu verkürzen. Insbesondere müssen Sie einige index.js-Dateien in den Verzeichnissen *client* , import / start / *{client, server}* und *server* erstellen.
- Sie können ein vollständiges Beispiel unter https://github.com/rafa-lft/Meteor_React_Base anzeigen. Suchen Sie nach dem Tag *Step1_CreateProject*

Fügen Sie React + ReactRouter hinzu

Wechseln Sie ggf. in Ihr Projektverzeichnis `cd MyAwesomeProject`

1- Fügen Sie den Reakti- und Reakt-Router hinzu

```
meteor npm install --save react-router@3.0.0 react@15.5.4 react-dom@15.5.4
```

2- Bearbeiten Sie client / main.html und ersetzen Sie den Inhalt wie folgt:

```
<body>
   <div id="react-root"></div>
</body>
```

Was auch immer der ReaktRouter zeigt, der zeigt es im '# Reakt-Wurzel'-Element

3- Erstellen Sie die Layouts-Datei in Importe / ui / layouts / App.jsx

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';


class App extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        {this.props.children}
      </div>
    );
  }
}

App.propTypes = {
  children: PropTypes.node
};

export default App;
```

4- Erstellen Sie die Routes-Datei in Imports / startup / client / Routes.jsx

```
import ReactDOM from 'react-dom';
import React, { Component } from 'react';
import { Router, Route, IndexRoute, browserHistory } from 'react-router';

import App from '../../ui/layouts/App.jsx';

import NotFound from '../../ui/pages/NotFound.jsx';
import Index from '../../ui/pages/Index.jsx';


class Routes extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <Router history={ browserHistory }>
        <Route path="/" component={ App }>
          <IndexRoute name="index" component={ Index }/>
          <Route path="*" component={ NotFound }/>
        </Route>
      </Router>
    );
  }
}

Routes.propTypes = {};


Meteor.startup(() =>{
  ReactDOM.render(
    <Routes/>,
    document.getElementById('react-root')
  );
});
```

Hinweis:

- Ich überspringe einige andere Dateien, die Sie erstellen müssen, um die Dinge zu verkürzen. Suchen Sie insbesondere nach Importen / ui / pages {Index.jsx, NotFound.jsx}.
- Sie können ein vollständiges Beispiel unter https://github.com/rafa-lft/Meteor_React_Base anzeigen. Suchen Sie nach dem Tag *Step2_ReactRouter*

Schritt 3 - Konten hinzufügen

Wechseln Sie ggf. in Ihr Projektverzeichnis `cd MyAwesomeProject`

1- Hinzufügen von Kontopaketen: `meteor add accounts-base accounts-password react-meteor-data`

2- Fügen Sie die Routen für die *Anmeldung* und die *Registrierung von* Seiten in Imports / startup / Routes.jsx hinzu. Die Methode *render ()* lautet wie folgt:

```
  render() {
    return (
      <Router history={ browserHistory }>
        <Route path="/" component={ App }>
          <IndexRoute name="index" component={ Index }/>
          <Route name="login" path="/login" component={ Login }/>
          <Route name="signup" path="/signup" component={ Signup }/>
          <Route name="users" path="/users" component={ Users }/>
          <Route name="editUser" path="/users/:userId" component={ EditUser }/>
          <Route path="*" component={ NotFound }/>
        </Route>
      </Router>
    );
  }
```

Hinweis:

- Ich überspringe einige andere Dateien, die Sie benötigen, um die Dinge zu verkürzen. Überprüfen Sie die Importe / startup / server / index.js und importieren Sie / ui / layouts / {App, NavBar} .jsx und importieren Sie / ui / pages / {Anmelden, Registrieren, Benutzer, EditUser} .jsx
- Sie können ein vollständiges Beispiel unter https://github.com/rafa-lft/Meteor_React_Base anzeigen. Suchen Sie nach dem Tag *Step3_Accounts*

Rollen hinzufügen

1- Rollenpaket hinzufügen (https://github.com/alanning/meteor-roles)

```
meteor add alanning:roles
```

2- Erstellen Sie einige Rollenkonstanten. In Datei importiert / api / accounts / rolls.js

```
const ROLES = {
  ROLE1: 'ROLE1',
  ROLE2: 'ROLE2',
  ADMIN: 'ADMIN'
};

export default ROLES;
```

3- Ich werde nicht zeigen, wie man Rollen zu einem Benutzer hinzufügt / aktualisiert. Ich werde nur erwähnen, dass auf Serverseite Benutzerrollen nach `Roles.setUserRoles(user.id, roles);` Weitere Informationen finden Sie unter https://github.com/alanning/meteor-roles und http://alanning.github.io/meteor-roles/classes/Roles.html

4- Wenn Sie bereits alle *Konten*- und Rollendateien eingerichtet haben (siehe vollständiges Beispiel unter https://github.com/rafa-lft/Meteor_React_Base . Suchen Sie nach dem Tag *Step4_roles*), können Sie jetzt eine Methode erstellen, die die *Genehmigung zulässt* oder nicht) Zugang zu den verschiedenen Routen. In Importe / Startup / Client / Routes.jsx

```
class Routes extends Component {
  constructor(props) {
    super(props);
  }

  authenticate(roles, nextState, replace) {
    if (!Meteor.loggingIn() && !Meteor.userId()) {
      replace({
        pathname: '/login',
        state: {nextPathname: nextState.location.pathname}
      });
      return;
    }
    if ('*' === roles) { // allow any logged user
      return;
    }
    let rolesArr = roles;
    if (!_.isArray(roles)) {
      rolesArr = [roles];
    }
    // rolesArr = _.union(rolesArr, [ROLES.ADMIN]);// so ADMIN has access to everything
    if (!Roles.userIsInRole(Meteor.userId(), rolesArr)) {
      replace({
        pathname: '/forbidden',
        state: {nextPathname: nextState.location.pathname}
      });
    }
  }

  render() {
    return (
      <Router history={ browserHistory }>
        <Route path="/" component={ App }>
          <IndexRoute name="index" component={ Index }/>
          <Route name="login" path="/login" component={ Login }/>
          <Route name="signup" path="/signup" component={ Signup }/>

          <Route name="users" path="/users" component={ Users }/>

          <Route name="editUser" path="/users/:userId" component={ EditUser }
                 onEnter={_.partial(this.authenticate, ROLES.ADMIN)} />


          {/* ********************
           Below links are there to show Roles authentication usage.
           Note that you can NOT hide them by
           { Meteor.user() && Roles.userIsInRole(Meteor.user(), ROLES.ROLE1) &&
           <Route name=.....
           }
           as doing so will change the Router component on render(), and ReactRouter will
complain with:
           Warning: [react-router] You cannot change <Router routes>; it will be ignored

           Instead, you can/should hide them on the NavBar.jsx component... don't worry: if
someone tries to access
           them, they will receive the Forbidden.jsx component
           *************/ }
          <Route name="forAnyOne" path="/for_any_one" component={ ForAnyone }/>

          <Route name="forLoggedOnes" path="/for_logged_ones" component={ ForLoggedOnes }
```

```
                onEnter={_.partial(this.authenticate, '*')} />

          <Route name="forAnyRole" path="/for_any_role" component={ ForAnyRole }
                onEnter={_.partial(this.authenticate, _.keys(ROLES))}/>

          <Route name="forRole1or2" path="/for_role_1_or_2" component={ ForRole1or2 }
                onEnter={_.partial(this.authenticate, [ROLES.ROLE1, ROLES.ROLE2])} />

          <Route name="forRole1" path="/for_role1" component={ ForRole1 }
                onEnter={_.partial(this.authenticate, ROLES.ROLE1)}/>

          <Route name="forRole2" path="/for_role2" component={ ForRole2 }
                onEnter={_.partial(this.authenticate, ROLES.ROLE2)} />


          <Route name="forbidden" path="/forbidden" component={ Forbidden }/>

          <Route path="*" component={ NotFound }/>
        </Route>
      </Router>
    );
  }
}
```

Wir haben einigen Routen einen *onEnter*- Trigger hinzugefügt. Für diese Routen passieren wir auch, welche Rollen betreten dürfen. Beachten Sie, dass der Callback von onEnter ursprünglich 2 Parameter empfängt. Wir verwenden Unterstriche (http://underscorejs.org/#partial) , um eine weitere (Rollen) hinzuzufügen. Die *Authentifizierungsmethode* (von onEnter aufgerufen) erhält die Rollen und:

- Prüfen Sie, ob der Benutzer überhaupt angemeldet ist. Wenn nicht, wird auf '/ login' umgeleitet.
- Bei Rollen === '*' gehen wir davon aus, dass jeder eingeloggte Benutzer teilnehmen kann, also erlauben wir es
- Andernfalls überprüfen wir, ob der Benutzer berechtigt ist (Roles.userIsInRole), und wenn nicht, leiten wir zu verboten weiter.
- Optional können Sie eine Zeile auskommentieren, sodass ADMIN auf alles zugreifen kann.

Der Code enthält mehrere Beispiele für unterschiedliche Routen, die für jeden (kein onEnter-Rückruf), für jeden protokollierten Benutzer, für jeden protokollierten Benutzer mit mindestens einer Rolle und für bestimmte Rollen zulässig sind.

Beachten Sie auch, dass ReactRouter (zumindest in Version 3) die Routen beim Rendern nicht ändern darf. Sie können also die Routen nicht innerhalb der Routes.jsx ausblenden. Aus diesem Grund werden in der Authentifizierungsmethode Weiterleitungen zu / verboten.

5- Ein häufiger Fehler bei ReactRouter und Meteor bezieht sich auf Aktualisierungen des Anwenderstatus. Zum Beispiel hat sich der Benutzer abgemeldet, aber wir zeigen immer noch seinen Namen in der Navigationsleiste. Dies geschieht, weil sich Meteor.user () geändert hat, aber wir werden nicht erneut gerendert.

Dieser Fehler kann durch Aufrufen von Meteor.user () im createContainer behoben werden. Hier

ist ein Beispiel davon, das in import / ui / layouts / NavBar.jsx verwendet wird:

```
export default createContainer((/* {params}*/) =>{
  Meteor.user(); // so we render again in logout or if any change on our User (ie: new roles)
  const loading = !subscription.ready();
  return {subscriptions: [subscription], loading};
}, NavBar);
```

Hinweis:

- Ich überspringe einige andere Dateien, die Sie benötigen, um die Dinge zu verkürzen. Überprüfen Sie die Importe / startup / server / index.js und importieren Sie / ui / layouts / {App, NavBar} .jsx und importieren Sie / ui / pages / {Anmelden, Registrieren, Benutzer, EditUser} .jsx
- Sie können ein vollständiges Beispiel unter https://github.com/rafa-lft/Meteor_React_Base anzeigen. Suchen Sie nach dem Tag *Step4_roles*

Meteor + React + ReactRouter online lesen: https://riptutorial.com/de/meteor/topic/10114/meteor-plus-react-plus-reactrouter

Kapitel 24: Meteor + Reaktion

Bemerkungen

React ist eine JavaScript-Bibliothek zum Erstellen von Benutzeroberflächen. Es ist Open Source , entwickelt und gepflegt von Facebook. Meteor hat produktionsreife Unterstützung für React.

Ressourcen:

- Tutorial reagieren
- Meteor + Reakt Tutorial

Examples

Setup und "Hello World"

Fügen Sie React zu Ihrem Projekt hinzu:

```
meteor npm install --save react react-dom react-mounter
```

Erstellen Sie die Datei `client/helloworld.jsx` , um eine einfache React-Komponente anzuzeigen:

```
import React, { Component } from 'react';
import { mount } from 'react-mounter';

// This component only renders a paragraph containing "Hello World!"
class HelloWorld extends Component {
  render() {
    return <p>Hello World!</p>;
  }
}

// When the client application starts, display the component by mounting it to the DOM.
Meteor.startup(() => {
  mount(HelloWorld);
});
```

Erstellen Sie einen reaktiven Container mit createContainer

Nehmen wir an, es gibt eine Sammlung namens `Todos` und das `autopublish` Paket wird hinzugefügt. Hier ist die Grundkomponente.

```
import { createContainer } from 'meteor/react-meteor-data';
import React, { Component, PropTypes } from 'react';
import Todos from '/imports/collections/Todos';

export class List extends Component {
  render() {
    const { data } = this.props;
```

```
    return (
      <ul className="list">
        {data.map(entry => <li {...entry} />)}
      </ul>
    )
  }
}

List.propTypes = {
  data: PropTypes.array.isRequired
};
```

Unten können Sie einen Container hinzufügen, um die reaktiven Daten in die Komponente einzuspeisen. Es würde so aussehen.

```
export default createContainer(() => {
  return {
    data: Todos.find().fetch()
  };
}, List);
```

Anzeigen einer MongoDB-Sammlung

Dieses Beispiel zeigt, wie eine MongoDB-Sammlung in einer React-Komponente angezeigt werden kann. Die Sammlung wird kontinuierlich zwischen Server und Client synchronisiert, und die Seite wird bei Änderungen der Datenbankinhalte sofort aktualisiert.

Um React-Komponenten und Meteor-Sammlungen zu verbinden, benötigen Sie das Paket `react-meteor-data` .

```
$ meteor add react-meteor-data
$ meteor npm install react-addons-pure-render-mixin
```

In `both/collections.js` Sammlungen wird eine einfache Sammlung deklariert. Jede Quelldatei in `both` Verzeichnissen enthält sowohl clientseitigen als auch serverseitigen Code:

```
import { Mongo } from 'meteor/mongo';

// This collection will contain a list of random numbers
export const Numbers = new Mongo.Collection("numbers");
```

Die Sammlung muss auf dem Server veröffentlicht werden. Erstellen Sie eine einfache Publikation in `server/publications.js` :

```
import { Meteor } from 'meteor/meteor';
import { Numbers } from '/both/collections.js';

// This publication synchronizes the entire 'numbers' collection with every subscriber
Meteor.publish("numbers/all", function() {
  return Numbers.find();
});
```

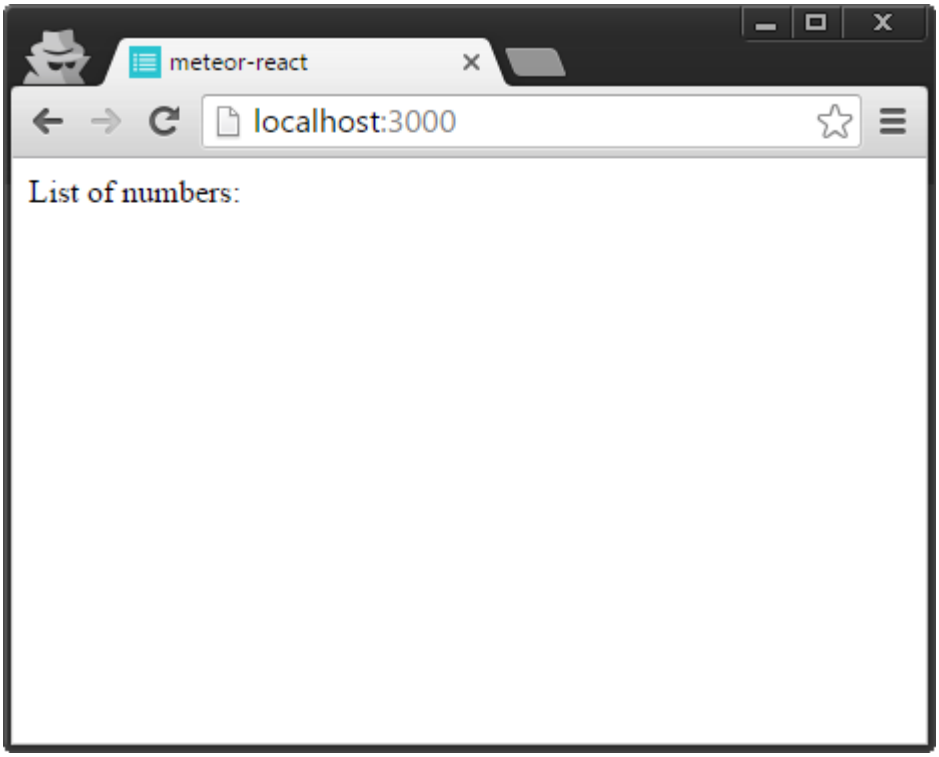
Mit der Funktion `createComponent` können wir reaktive Werte (wie die `Numbers` Auflistung) an eine React-Komponente übergeben. `client/shownumbers.jsx` :

```
import React from 'react';
import { createContainer } from 'meteor/react-meteor-data';
import { Numbers } from '/both/collections.js';

// This stateless React component renders its 'numbers' props as a list
function _ShowNumbers({numbers}) {
  return <div>List of numbers:
    <ul>
      // note, that every react element created in this mapping requires
      // a unique key - we're using the _id auto-generated by mongodb here
      {numbers.map(x => <li key={x._id}>{x.number}</li>)}
    </ul>
  </div>;
}

// Creates the 'ShowNumbers' React component. Subscribes to 'numbers/all' publication,
// and passes the contents of 'Numbers' as a React property.
export const ShowNumbers = createContainer(() => {
  Meteor.subscribe('numbers/all');
  return {
    numbers: Numbers.find().fetch(),
  };
}, _ShowNumbers);
```

Anfangs ist die Datenbank wahrscheinlich leer.



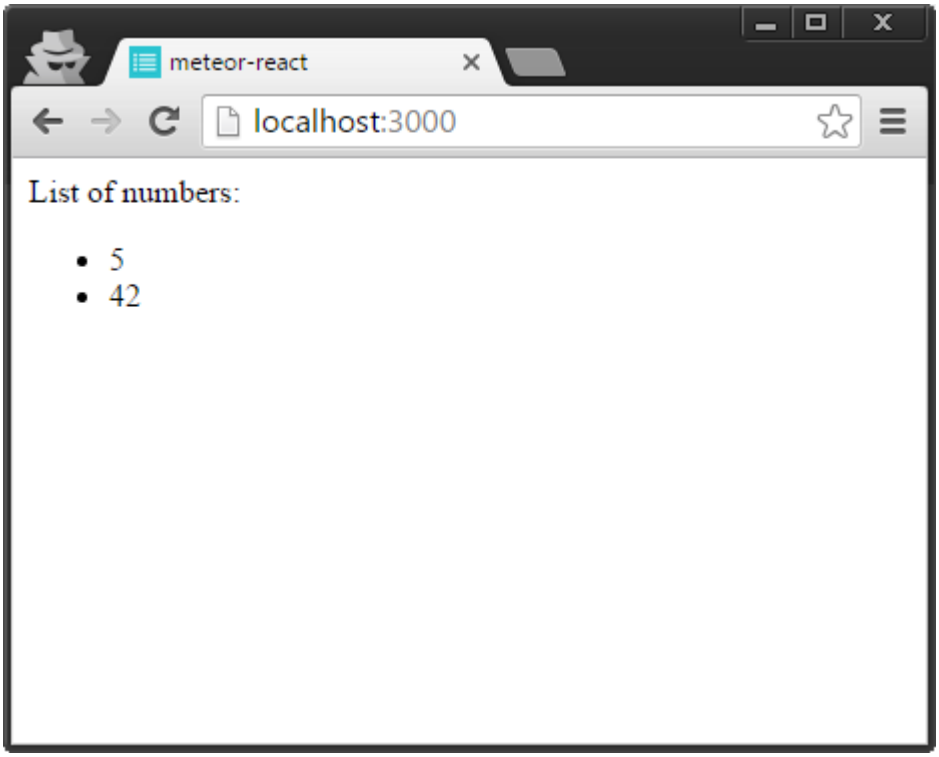
Fügen Sie MongoDB Einträge hinzu und beobachten Sie, wie die Seite automatisch aktualisiert wird.

```
$ meteor mongo
MongoDB shell version: 3.2.6
```

```
connecting to: 127.0.0.1:3001/meteor

meteor:PRIMARY> db.numbers.insert({number: 5});
WriteResult({ "nInserted" : 1 })

meteor:PRIMARY> db.numbers.insert({number: 42});
WriteResult({ "nInserted" : 1 })
```



Meteor + Reaktion online lesen: https://riptutorial.com/de/meteor/topic/3121/meteor-plus-reaktion

Kapitel 25: Meteor Benutzerkonten

Examples

Meteor-Kontopaket

Sie haben einige Möglichkeiten, um sich bei Meteor anzumelden. Die gebräuchlichste Methode ist die Verwendung von `accounts` für Meteor.

Konto-Passwort

Wenn Sie möchten, dass sich Benutzer auf Ihrer Site erstellen und registrieren können, können Sie `accounts-password` .

Installieren Sie das Paket mit dem `meteor add accounts-password` das Konto " `meteor add accounts-password` .

Um einen Benutzer zu erstellen, müssen Sie `Accounts.createUser(options, [callback])`

`options` muss ein Objekt mit den folgenden Eigenschaften sein:

- `username` : Der Benutzername des Benutzers als String.
- `email` : Die E- `email` des Benutzers als Zeichenfolge.
- `password` : Das (nicht verschlüsselte) Kennwort des Benutzers als Zeichenfolge.
- `profile` : Die optionalen zusätzlichen Daten des Benutzers als Objekt. Dies kann beispielsweise der Vor- und Nachname des Benutzers sein. `profile` ist jedoch optional.

Der Callback gibt 1 Variable zurück, wenn ein Fehler vorliegt, bei dem es sich um ein Meteor.Error-Objekt handelt.

Sie müssen lediglich entweder den `username` oder die `email` . Sie können also einen Benutzer mit einem Benutzernamen, aber ohne E-Mail erstellen, und umgekehrt. Sie können auch beides verwenden.

Wenn alles korrekt lief, wird die neu erstellte Benutzer-ID zurückgegeben.

So können Sie beispielsweise Folgendes verwenden:

```
// server side
var id = Accounts.createUser({
    username: "JohnDoe",
    email: "JohnDoe@gmail.com",
    password: "TheRealJohn123",
    profile: {
        firstName: "John",
        lastName: "Doe"
    }
}, function(err) {
```

```
    console.log(err.reason);
});
```

Sie werden automatisch auch angemeldet, wenn der Benutzer erfolgreich erstellt wurde.

Das ist der schaffende Teil. Um sich anzumelden, müssen Sie `Meteor.loginWithPassword(identifier, password, [callback])` auf der Clientseite verwenden.

`identifier` ist der `username` , die `email` `userId` oder die `username` `userId` als Zeichenfolge Ihres Benutzers. `password` ist das (nicht verschlüsselte) `password` des Benutzers.

Der Rückruf gibt eine Variable zurück, wenn ein Fehler vorliegt, bei dem es sich um ein Meteor.Error-Objekt handelt.

Beispiel:

```
// client side
Meteor.loginWithPassword("JohnDoe", "TheRealJohn123", function(err) {
    console.log(err.reason);
});
```

Und das ist es für das grundlegende Erstellen von Konten und das Einloggen.

Zugriff auf Benutzerdaten

Sie können auf der Clientseite prüfen, ob der Benutzer angemeldet ist, indem Sie `Meteor.userId()` `userId` Diese gibt ihre `userId` wenn sie angemeldet ist, und `undefined` wenn sie nicht angemeldet ist.

Sie können einige Informationen von `Meteor.user()` . Es wird undefined zurückgegeben, wenn der Benutzer nicht angemeldet ist, und einige Benutzerdaten, falls dies der Fall ist. Standardmäßig werden keine Kennwörter angegeben. Standardmäßig werden die Benutzer-ID des Benutzers, der Benutzername und das Profilobjekt angezeigt.

Wenn Sie prüfen möchten, ob ein Benutzer auf einer Seite angemeldet ist, können Sie auch den `currentUser` Helper verwenden. Der Inhalt von `Meteor.user()` . Beispiel:

```
{{#if currentUser}}
    <h1>Hello there, {{currentUser.username}}!</h1>
{{else}}
    <h1>Please log in.</h1>
{{/if}}
```

Andere Kontofunktionen

Es gibt einige andere Funktionen, die für jedes Kontopaket funktionieren.

Sie können sich mit `Meteor.logout()`

Verwenden Sie nicht das Standardprofilfeld

Es gibt ein verführerisches Feld namens `profile` , das standardmäßig hinzugefügt wird, wenn sich ein neuer Benutzer registriert. Dieses Feld war früher dazu gedacht, als Notizblock für benutzerspezifische Daten verwendet zu werden - etwa für dessen Avatar, Name, Intro-Text usw. Aus diesem Grund ist **das `profile` jedes Benutzers automatisch von diesem Benutzer vom Client aus beschreibbar** . Es wird auch automatisch für den jeweiligen Benutzer auf dem Client veröffentlicht.

Es stellt sich heraus, dass ein Feld, das standardmäßig beschreibbar ist, ohne dass dies sehr offensichtlich ist, nicht die beste Idee ist. Es gibt viele Geschichten von neuen Meteor-Entwicklern, die Felder wie `isAdmin` im `profile` speichern. Dann kann ein böswilliger Benutzer dies jederzeit auf "true" setzen und sich selbst zum Administrator machen. Selbst wenn Sie sich darüber keine Sorgen machen, ist es nicht ratsam, böswillige Benutzer in der Datenbank beliebige Datenmengen speichern zu lassen.

Anstatt sich mit den Besonderheiten dieses Feldes auseinanderzusetzen, kann es hilfreich sein, seine Existenz einfach zu ignorieren. Sie können dies sicher tun, solange Sie alle Schreibvorgänge vom Client ablehnen:

```
// Deny all client-side updates to user documents
Meteor.users.deny({
  update() { return true; }
});
```

Selbst wenn Sie die Sicherheitsauswirkungen des Profils ignorieren, ist es nicht ratsam, alle benutzerdefinierten Daten Ihrer App in einem Feld zusammenzufassen. Das Datentransferprotokoll von Meteor führt keine tief verschachtelten Unterschiede zwischen Feldern aus. Daher ist es eine gute Idee, Ihre Objekte in viele Felder der obersten Ebene des Dokuments zu glätten.

Meteor Benutzerkonten online lesen: https://riptutorial.com/de/meteor/topic/6219/meteor-benutzerkonten

Kapitel 26: Meteor mit einem Proxy-Server verwenden

Examples

Verwendung der `HTTP [S] _PROXY`-Env-Var

Diese Seite beschreibt die Verwendung des Meteor-Befehlszeilentools (z. B. beim Herunterladen von Paketen, beim Bereitstellen Ihrer App usw.) hinter einem Proxyserver.

Wie viele andere Befehlszeilenprogramme liest das Meteor-Tool die Proxy-Konfiguration aus den Umgebungsvariablen `HTTP_PROXY` und `HTTPS_PROXY` (auch die Kleinbuchstaben-Varianten funktionieren). Beispiele für das Ausführen von Meteor hinter einem Proxy:

- unter Linux oder Mac OS X

```
export HTTP_PROXY=http://user:password@1.2.3.4:5678
export HTTPS_PROXY=http://user:password@1.2.3.4:5678
meteor update
```

- unter Windows

```
SET HTTP_PROXY=http://user:password@1.2.3.4:5678
SET HTTPS_PROXY=http://user:password@1.2.3.4:5678
meteor update
```

Einrichten einer Proxy-Ebene

- Stellen Sie die Meteor-App mit Nginx Proxy auf Ubuntu bereit
- So erstellen Sie ein SSL-Zertifikat unter Nginx für Ubuntu 14
- So stellen Sie eine Meteor JS-App mit Nginx auf Ubuntu bereit
- So installieren Sie ein SSL-Zertifikat von einer kommerziellen Zertifizierungsstelle
- NameCheap SSL-Zertifikate

Meteor mit einem Proxy-Server verwenden online lesen:
https://riptutorial.com/de/meteor/topic/517/meteor-mit-einem-proxy-server-verwenden

Kapitel 27: Mobile Apps

Examples

Seitenlayout auf verschiedenen Geräten - CSS

Wenn Ihre Anwendung auf verschiedenen Geräten ausgeführt werden soll, muss sie je nach Gerätegröße in verschiedenen ViewPorts gerendert werden. Sie können auf zwei Arten damit umgehen: mit Javascript-Regeln oder CSS-Medienstilen. Wenn Sie eine MVC- oder MVVM-Bibliothek wie Angular oder Ember (oder Blaze für diese Angelegenheit) verwendet haben und nur ein einzelnes Gerät oder eine Hardwareplattform anvisiert haben, müssen Sie möglicherweise Ihr MVC-Modell überdenken, da andere Hardware-ViewPorts vorhanden sind in Ihre Bewerbung eingeführt.

```
// desktop
@media only screen and (min-width: 960px) {
}

// landscape orientation
@media only screen and (min-width: 768px) {
}

// portrait orientation
@media only screen and (min-width: 480px) {
}
```

Sie müssen herausfinden, ob Sie die Stile bei 768px (Hochformat) oder 1024 Pixel (Querformat) aufteilen möchten. Angenommen, Ihr mobiles Zielgerät ist das iPad, das ein Verhältnis von 3: 4 verwendet. Andernfalls müssen Sie die Seitenverhältnisse der Geräte ermitteln, auf die Sie abzielen möchten, und die Schwellenwerte von dort aus ermitteln.

Festgelegte Fenstergröße

Wenn Sie Layouts mit Bildschirmen mit fester Größe für verschiedene mobile Geräte entwerfen, möchten Sie möglicherweise dieses Design spiegeln, wenn Sie Ihre App auf einem Desktop ausführen. Mit der folgenden Methode wird die Größe des Fensters AUSSERHALB von PhoneGap festgelegt, wodurch ein Fenster mit fester Größe auf dem Desktop angezeigt wird. Manchmal ist es am einfachsten, die Erwartungen der Benutzer und das Design der Benutzeroberfläche durch Einschränken der Optionen zu verwalten!

```
// create a window of a specific size
var w=window.open('','', 'width=100,height=100');
w.resizeTo(500,500);

// prevent window resize
var size = [window.width,window.height];  //public variable
$(window).resize(function(){
    window.resizeTo(size[0],size[1]);
});
```

Offline-Zwischenspeicherung

Damit all dies funktioniert, benötigen Sie wahrscheinlich eine Offline-Unterstützung, d.

```
meteor add appcache
meteor add grounddb
```

Scroll-Bounce deaktivieren

Bei Desktop-Apps möchten Sie möglicherweise den Scroll-Bounce-Modus deaktivieren, um Ihrer App ein natürlicheres Gefühl zu verleihen. Sie können dies mit Javascript tun, indem Sie deaktivieren, wie der Browser das DOM steuert:

```
// prevent scrolling on the whole page
// this is not meteorish; TODO: translate to meteor-centric code
document.ontouchmove = function(e) {e.preventDefault()};

// prevent scrolling on specific elements
// this is not meteorish; TODO: translate to meteor-centric code
scrollableDiv.ontouchmove = function(e) {e.stopPropagation()};
```

Alternativ können Sie css sowie die Überlauf- und Bildlaufstile verwenden.

```
#appBody {
  overflow: hidden;
}

#contentContainer {
  .content-scrollable {
    overflow-y: auto;
    -webkit-overflow-scrolling: touch;
  }
}
```

Das Objektmodell, das für das Funktionieren des Obigen benötigt wird, sieht in etwa so aus:

```
<div id="appBody">
  <div id="contentContainer">
    <div class="content-scrollable">
      <!-- content -->
    </div>
  </div>
</div>
```

Multitouch & Gesten

Mobile Geräte verfügen im Allgemeinen über keine Tastaturen. Sie müssen also Ihrer Anwendung einige haptische Controller hinzufügen. Die zwei beliebtesten Pakete, die die Leute scheinbar verwenden, sind FastClick und Hammer. Die Installation ist einfach.

```
meteor add fastclick
```

```
meteor add hammer:hammer
```

FastClick erfordert fast keine Konfiguration, während Hammer zum Verdrahten ein bisschen Arbeit erfordert. Das konische Beispiel aus der Todos-App sieht folgendermaßen aus:

```
Template.appBody.onRendered(function() {
  if (Meteor.isCordova) {
    // set up a swipe left / right handler
    this.hammer = new Hammer(this.find('#appBody'));
    this.hammer.on('swipeleft swiperight', function(event) {
      if (event.gesture.direction === 'right') {
        Session.set(MENU_KEY, true);
      } else if (event.gesture.direction === 'left') {
        Session.set(MENU_KEY, false);
      }
    });
  }
});
```

Erstellen Sie Ihre Symbole und Begrüßungsbildschirme

Bevor Sie Ihre App kompilieren und auf Ihrem Gerät ausführen, müssen Sie einige Symbole und `mobile-config.js` erstellen und Ihrer App eine `mobile-config.js` Datei hinzufügen.

```
App.icons({
  // iOS
  'iphone': 'resources/icons/icon-60x60.png',
  'iphone_2x': 'resources/icons/icon-60x60@2x.png',
  'ipad': 'resources/icons/icon-72x72.png',
  'ipad_2x': 'resources/icons/icon-72x72@2x.png',

  // Android
  'android_ldpi': 'resources/icons/icon-36x36.png',
  'android_mdpi': 'resources/icons/icon-48x48.png',
  'android_hdpi': 'resources/icons/icon-72x72.png',
  'android_xhdpi': 'resources/icons/icon-96x96.png'
});

App.launchScreens({
  // iOS
  'iphone': 'resources/splash/splash-320x480.png',
  'iphone_2x': 'resources/splash/splash-320x480@2x.png',
  'iphone5': 'resources/splash/splash-320x568@2x.png',
  'ipad_portrait': 'resources/splash/splash-768x1024.png',
  'ipad_portrait_2x': 'resources/splash/splash-768x1024@2x.png',
  'ipad_landscape': 'resources/splash/splash-1024x768.png',
  'ipad_landscape_2x': 'resources/splash/splash-1024x768@2x.png',

  // Android
  'android_ldpi_portrait': 'resources/splash/splash-200x320.png',
  'android_ldpi_landscape': 'resources/splash/splash-320x200.png',
  'android_mdpi_portrait': 'resources/splash/splash-320x480.png',
  'android_mdpi_landscape': 'resources/splash/splash-480x320.png',
  'android_hdpi_portrait': 'resources/splash/splash-480x800.png',
  'android_hdpi_landscape': 'resources/splash/splash-800x480.png',
  'android_xhdpi_portrait': 'resources/splash/splash-720x1280.png',
  'android_xhdpi_landscape': 'resources/splash/splash-1280x720.png'
```

```
});
```

Meteor Cordova Architektur-Pipeline

Jetzt ist es an der Zeit, die Dokumentation zur Meteor Cordova Phonegap Integration durchzugehen .

Seit der Erstellung dieser Dokumentation wurden XCode und Yosemite veröffentlicht, was bei der Installation zu Problemen führte. Hier sind die Schritte, die wir durchführen mussten, um Meteor auf einem iOS-Gerät zu kompilieren.

- Upgrade auf Yosemite.
- XCode löschen (aus dem Anwendungsordner in den Papierkorb ziehen)
- Installieren Sie XCode 6.1 aus dem App Store.
- Stimmen Sie den verschiedenen Bedingungen zu.

```
# 5.  clone and rebuild the ios-sim locally
#     (this step will not be needed in future releases)
git clone https://github.com/phonegap/ios-sim.git
cd ios-sim
rake build

# 6.  make sure we can update the .meteor/packages locations
#     (this step will not be needed in future releases)
sudo chmod -R 777 ~/.meteor/packages

# 7.  copy the new build into Meteor locations
#     (this step will not be needed in future releases)
for i in `find ~/.meteor/packages/meteor-tool/ -name ios-sim -type f`; do
  cp -R ./build/Release/ios-sim "$i"
done

# 8.  install the ios platform to your app
cd myapp
meteor list-platforms
meteor add-platform ios
meteor list-platforms

# 9.  and that there aren't dead processes
ps -ax
kill -9 <pid>
# /Users/abigailwatson/.meteor/packages/meteor-
tool/.1.0.35.wql4jh++os.osx.x86_64+web.browser+web.cordova/meteor-tool-
os.osx.x86_64/dev_bundle/mongodb/bin/mongod
# tail -f /Users/abigailwatson/Code/Medstar/dart/webapp/.meteor/local/cordova-
build/platforms/ios/cordova/console.log

# 10.  make sure there are correct permissions on the application (important!)
sudo chmod -R 777 .meteor/local/

# 11.  run app
meteor run ios

# 12.  if that doesn't work, clear the directory
sudo rm -rf .meteor/local
```

```
# 13a.  run meteor again to create the default browser build
meteor

# 13b.  run it a second time so bootstrap and other packages get downloaded into the browser
build
ctrl-x
meteor

# 14.  then run the ios version
ctrl-x
meteor run ios
```

XCode sollte während des Prozesses gestartet werden. Wählen Sie Ihren Simulator aus und drücken Sie die 'Play'-Taste.

IOS-Entwicklung

- Registrieren Sie Ihren Apple Developer Account
- Registrieren Sie eine App-ID für Ihre App
- Registrieren Sie die UUID Ihrer Testgeräte
- Generieren eines Bereitstellungsprofils für die iOS-App-Entwicklung
    - Generieren Sie ein CertificateSigningRequest von KeychainAccess
    - Senden Sie CertificateSigningRequest an https://developer.apple.com/account/ios/profile/profileCreate.action
    - Laden Sie das Zertifikat herunter, und doppelklicken Sie darauf, um es in den Schlüsselbund zu importieren
- Gehen Sie zu XCode> Einstellungen> Konten und registrieren Sie Ihr Apple Developer Account

IOS-Gerätetest

- Stellen Sie sicher, dass Ihre Entwicklungs-Workstation und Ihr iPhone mit demselben WLAN-Netzwerk verbunden sind. Tethering, Hotspots und andere Ad-hoc-Netzwerke funktionieren nicht.
- Führen Sie `sudo meteor run ios-device`
- Bereitstellen auf Ihrem Gerät!

Konfigurieren Sie Ihr Cordova-Projekt (config.xml)

Meteor liest während des `mobile-config.js` eine `mobile-config.js` Datei im Stammverzeichnis Ihres App-Verzeichnisses und verwendet die dort angegebenen Einstellungen, um `config.xml` zu generieren.

```
Project_folder
 ├── /.meteor
 └── mobile-config.js
```

Die meisten Konfigurationen können mit `mobile-config.js` (App-Metadaten, Voreinstellungen, Symbolen und Startbildschirmen sowie Installationsparametern für Cordova-Plugins) erreicht

werden.

```
App.info({
  id: 'com.example.matt.uber',
  name: 'über',
  description: 'Get über power in one button click',
  author: 'Matt Development Group',
  email: 'contact@example.com',
  website: 'http://example.com'
});

// Set up resources such as icons and launch screens.
App.icons({
  'iphone': 'icons/icon-60.png',
  'iphone_2x': 'icons/icon-60@2x.png',
  // ... more screen sizes and platforms ...
});

App.launchScreens({
  'iphone': 'splash/Default~iphone.png',
  'iphone_2x': 'splash/Default@2x~iphone.png',
  // ... more screen sizes and platforms ...
});

// Set PhoneGap/Cordova preferences
App.setPreference('BackgroundColor', '0xff0000ff');
App.setPreference('HideKeyboardFormAccessoryBar', true);
App.setPreference('Orientation', 'default');
App.setPreference('Orientation', 'all', 'ios');

// Pass preferences for a particular PhoneGap/Cordova plugin
App.configurePlugin('com.phonegap.plugins.facebookconnect', {
  APP_ID: '1234567890',
  API_KEY: 'supersecretapikey'
});
```

Bearbeiten Sie die Datei `/.meteor/local/cordova-build/config.xml` *nicht* manuell, da sie bei jedem `meteor run ios/android` oder `meteor build` wird. Dadurch gehen alle Ihre Änderungen verloren.

Referenzseite: Meteor Guide> Build> Mobile> App konfigurieren

Erkennen des Geräteereignisses

Natürlich ist die Erkennung von Mobilgeräten am besten, wenn Sie von der Hardware direkt benachrichtigt werden. Cordova PhoneGap stellt ein "deviceready" -Ereignis bereit, zu dem Sie einen Ereignis-Listener hinzufügen können.

```
document.addEventListener('deviceready', function(){
 Session.set('deviceready', true);
}, false);
```

Mobile Apps online lesen: https://riptutorial.com/de/meteor/topic/3705/mobile-apps

Kapitel 28: Mongo-Datenbankverwaltung

Bemerkungen

Wenn Sie kein Ruby-Programm ausführen möchten, ist Genghis ein Klassiker:
http://genghisapp.com/

Wenn Sie jedoch eine skalierbare Produktionsumgebung benötigen, besuchen Sie MongoHQ.
http://www.mongohq.com/

Auch der Mongo Monitoring Service von 10Gen, die Macher von Mongo:
https://mms.mongodb.com/

MongoClient ist in Meteor, Free, Open Source und Cross-Platform geschrieben.

RoboMongo Native, plattformübergreifendes MongoDB-Verwaltungstool

Examples

Vererbte Datenbank analysieren

Es gibt zwei großartige Dienstprogramme für die Black-Box-Analyse von Datenbanken. Zuerst ist vielfalt.js, die Ihnen einen übergeordneten Überblick verschafft. Die zweite ist schema.js, mit der Sie in die Sammlungen einsteigen können, um mehr Details zu den einzelnen Feldern zu erhalten. Wenn Sie eine Mongo-Produktionsdatenbank übernehmen, können Sie mit diesen beiden Dienstprogrammen verstehen, was vor sich geht und wie die Sammlungen und Dokumente strukturiert sind.

vielfalt.js

```
mongo test --eval "var collection = 'users'" variety.js
```

schema.js

```
mongo --shell schema.js
```

Stellen Sie eine Verbindung zu einer Datenbank auf * .meteorapp.com her

Das Flag --url kann schwierig zu verwenden sein. Es gibt ein 60 Sekunden langes Fenster, um sich zu authentifizieren, und der Benutzername / das Kennwort wird zufällig zurückgesetzt. Stellen Sie sicher, dass robomongo geöffnet ist und bereit ist, eine neue Verbindung zu konfigurieren, wenn Sie den Befehl ausführen.

```
# get the MONGO_URL string for your app
meteor mongo --url $METEOR_APP_URL
```

Laden Sie eine Datenbank von * .meteor.com herunter

Das Gleiche wie zuvor, aber Sie müssen die Informationen in den Befehl mongodump kopieren. Sie müssen die folgenden Befehle recht schnell ausführen, und dies erfordert eine Hand / Auge-Koordination. Sei gewarnt! Dies ist ein unheilvoller Hacker! Aber Spaß! Betrachten Sie es als Videospiel! : D

```
# get the MONGO_URL string for your app
meteor mongo --url $METEOR_APP_URL

# then quickly copy all the info into the following command
mongodump -u username -p password --port 27017 --db meteor_app_url_com --host production-db-
b1.meteor.io
```

Daten aus lokaler Meteor-Entwicklungsinstanz exportieren?

Dieser Befehl erstellt ein / dump-Verzeichnis und speichert jede Sammlung in einer separaten BSON-Blob-Datei. Dies ist der beste Weg, um Datenbanken zwischen Systemen zu sichern oder zu übertragen.

```
mongodump --db meteor
```

Wiederherstellen von Daten aus einem Dumpfile

Der Analog zum `meteordump` Befehl ist `meteorrestore` . Sie können einen teilweisen Import durchführen, indem Sie die bestimmte Sammlung auswählen, die importiert werden soll. Besonders nützlich, nachdem ein Drop-Befehl ausgeführt wurde.

```
# make sure your app is running
meteor

# then import your data
mongorestore --port 3001 --db meteor /path/to/dump

# a partial import after running > db.comments.drop()
mongorestore --port 3001 --db meteor /path/to/dump -c comments.bson
```

Exportieren Sie eine Collection nach JSON

Führen Sie meteor aus, öffnen Sie ein anderes Terminalfenster und führen Sie den folgenden Befehl aus.

```
mongoexport --db meteor --collection foo --port 3001 --out foo.json
```

Importieren Sie eine JSON-Datei in Meteor

Das Importieren in eine Standard-Meteor-Instanz ist ziemlich einfach. Beachten Sie, dass Sie die Option --jsonArray hinzufügen können, wenn Ihre Json-Datei als Array eines anderen Systems

exportiert wird.

```
mongoimport --db meteor --port 3001 --collection foo --file foo.json
```

Daten zwischen Staging und lokalen Datenbanken kopieren

Mongo unterstützt das Kopieren von Datenbank zu Datenbank. Dies ist hilfreich, wenn Sie große Datenbanken in einer Staging-Datenbank haben, die Sie in eine lokale Entwicklungsinstanz kopieren möchten.

```
// run mongod so we can create a staging database
// note that this is a separate instance from the meteor mongo and minimongo instances
mongod

// import the json data into a staging database
// jsonArray is a useful command, particularly if you're migrating from SQL
mongoimport -d staging -c assets < data.json --jsonArray

// navigate to your application
cd myappdir

// run meteor and initiate it's database
meteor

// connect to the meteor mongodb
meteor mongo --port 3002

// copy collections from staging database into meteor database
db.copyDatabase('staging', 'meteor', 'localhost');
```

Komprimieren Sie eine Mongo-Datenbank auf einer Ubuntu-Box

Vorbelegung Mongo legt Festplattenplatz in leeren Containern zur Seite. Wenn also Zeit ist, etwas auf die Festplatte zu schreiben, muss er nicht erst die Bits aus dem Weg räumen. Dies geschieht durch einen Verdopplungsalgorithmus, bei dem der vorab zugewiesene Speicherplatz immer verdoppelt wird, bis er 2 GB erreicht. und dann ist jede Prealloc-Datei von dort 2 GB. Sobald die Daten vorbelegt sind, werden sie nicht aufgehoben, es sei denn, Sie geben dies ausdrücklich an. Die beobachtbare MongoDB-Speicherplatznutzung nimmt also automatisch zu, aber nicht nach unten.

Einige Recherchen zur Mongo-Vorbesetzung ...
Reduzieren der Größe der Mongodb-Datenbankdatei
mongo-prealloc-dateien-aufnahmeraum

```
// compact the database from within the Mongo shell
db.runCommand( { compact : 'mycollectionname' } )

// repair the database from the command line
mongod --config /usr/local/etc/mongod.conf --repair --repairpath /Volumes/X/mongo_repair --
nojournal

// or dump and re-import from the command line
mongodump -d databasename
```

```
echo 'db.dropDatabase()' | mongo databasename
mongorestore dump/databasename
```

Setzen Sie einen Replikatsatz zurück

Löschen Sie die lokalen Datenbankdateien. Verlassen Sie einfach die Mongo-Shell, navigieren Sie zum / dbpath (wo auch immer Sie es einrichten) und löschen Sie die Dateien in diesem Verzeichnis.

Stellen Sie eine Remote-Verbindung zu einer Mongo-Instanz auf * .meteor.com her

Wussten Sie über die Flagge von `--url` ? Sehr praktisch

```
meteor mongo --url YOURSITE.meteor.com
```

Zugriff auf Mongo-Protokolldateien in einer lokalen Meteor-Instanz

Sie sind nicht leicht zugänglich. Wenn Sie den Befehl "meteor bundle" ausführen, können Sie eine tar.gz-Datei generieren und Ihre App dann manuell ausführen. Auf diese Weise sollten Sie in der Lage sein, auf die Mongo-Logs zuzugreifen ... wahrscheinlich im Verzeichnis .meteor / db. Wenn Sie wirklich auf mongodb-Protokolldateien zugreifen müssen, richten Sie eine reguläre mongodb-Instanz ein und verbinden Sie Meteor mit einer externen mongo-Instanz, indem Sie die Umgebungsvariable MONGO_URL festlegen:

```
MONGO_URL='mongodb://user:password@host:port/databasename'
```

Sobald dies erledigt ist, sollten Sie an den üblichen Stellen auf Protokolle zugreifen können ...

```
/var/log/mongodb/server1.log
```

Protokolldateien auf einer Ubuntu-Box drehen

Diese Protokolldateien müssen rotiert werden, da sonst der gesamte Festplattenspeicher aufgebraucht wird. Beginnen Sie mit etwas Forschung ...
Mongodb-Log-Datei-Wachstum
Rotationsprotokolldateien

Protokolldateien können mit dem folgenden Befehl angezeigt werden ...

```
ls /var/log/mongodb/
```

Um die Rotation der Protokolldatei einzurichten, müssen Sie Folgendes tun:

```
// put the following in the /etc/logrotate.d/mongod file
/var/log/mongo/*.log {
```

```
    daily
    rotate 30
    compress
    dateext
    missingok
    notifempty
    sharedscripts
    copytruncate
    postrotate
        /bin/kill -SIGUSR1 `cat /var/lib/mongo/mongod.lock 2> /dev/null` 2> /dev/null || true
    endscript
}

// to manually initiate a log file rotation, run from the Mongo shell
use admin
db.runCommand( { logRotate : 1 } )
```

Mongo-Datenbankverwaltung online lesen: https://riptutorial.com/de/meteor/topic/3707/mongo-datenbankverwaltung

Kapitel 29: MongoDB

Einführung

MongoDB ist ein kostenloses und quelloffenes, Open-Source-Dokumentenorientiertes Datenbankprogramm. Im Gegensatz zu klassischen SQL-Datenbanken verwendet MongoDB BSON (wie JSON) zum Speichern von Daten. Meteor wurde für die Verwendung von MongoDB zum Speichern von Datenbanken entwickelt. In diesem Thema wird beschrieben, wie der MongoDB-Speicher in Meteor-Anwendungen implementiert wird.

Examples

Exportieren Sie eine entfernte Mongo-Datenbank und importieren Sie sie in eine lokale Meteor-Mongo-Datenbank

Hilfreich, wenn Sie sich eine Kopie einer Produktionsdatenbank zum lokalen Wiedergeben holen möchten.

1. `mongodump --host some-mongo-host.com:1234 -d DATABASE_NAME -u DATABASE_USER -p DATABASE_PASSWORD` Dies wird eine lokale erstellen `dump` - Verzeichnis; In diesem Verzeichnis sehen Sie ein Verzeichnis mit Ihrem `DATABASE_NAME` .
2. Führen Sie, `mongorestore --db meteor --drop -h localhost --port 3001 DATABASE_NAME` Ihre lokale Meteor-App im `dump` Verzeichnis ausgeführt wird, `mongorestore --db meteor --drop -h localhost --port 3001 DATABASE_NAME` : `mongorestore --db meteor --drop -h localhost --port 3001 DATABASE_NAME`

Holen Sie sich die Mongo-URL Ihrer lokalen Meteor-Mongo-Datenbank

Während Ihre Meteor-App lokal ausgeführt wird:

```
meteor mongo --url
```

Verbinden Sie Ihre lokale Meteor-App mit einer alternativen Mongo-Datenbank

`MONGO_URL` Umgebungsvariable `MONGO_URL` , bevor Sie Ihre lokale Meteor-App starten.

Linux / MacOS-Beispiel:

```
MONGO_URL="mongodb://some-mongo-host.com:1234/mydatabase" meteor
```

oder

```
export MONGO_URL="mongodb://some-mongo-host.com:1234/mydatabase"
meteor
```

Windows-Beispiel

Hinweis: Verwenden Sie nicht "

```
set MONGO_URL=mongodb://some-mongo-host.com:1234/mydatabase
meteor
```

NPM

```
//package.json

"scripts": {
    "start": "MONGO_URL=mongodb://some-mongo-host.com:1234/mydatabase meteor"
}

$ npm start
```

Meteor ohne MongoDB ausführen

Setzen Sie `MONGO_URL` auf einen beliebigen Wert mit Ausnahme einer Datenbank-URL. `MONGO_URL` Sie `MONGO_URL` , dass in Ihrem Meteor-Projekt keine Sammlungen definiert sind (einschließlich von Meteor-Paketen definierte Sammlungen), um Meteor ohne MongoDB auszuführen.

Beachten Sie, dass ohne MongoDB Server- / Client-Methoden und alle Pakete, die sich auf das Benutzerkontosystem von Meteor beziehen, undefiniert sind. Ex: `Meteor.userId()`

Linux / Mac:

```
MONGO_URL="none" meteor
```

oder

```
export MONGO_URL="none"
meteor
```

Windows:

```
set MONGO_URL=none
meteor
```

Fertig machen

Sie können die `mongo` Shell starten, indem Sie den folgenden Befehl in Ihrem Meteor-Projekt ausführen:

```
meteor mongo
```

Bitte beachten Sie: Das Starten der serverseitigen Datenbankkonsole funktioniert nur, wenn Meteor die Anwendung lokal ausführt.

Danach können Sie alle Sammlungen auflisten, indem Sie den folgenden Befehl über die `mongo` Shell ausführen:

```
show collections
```

Sie können auch grundlegende MongoDB-Vorgänge ausführen, z. B. Abfragen, Einfügen, Aktualisieren und Löschen von Dokumenten.

Dokumente abfragen

Dokumente können mit der `find()` -Methode abgefragt werden, zB:

```
db.collection.find({name: 'Matthias Eckhart'});
```

Dadurch werden alle Dokumente aufgeführt , die die haben `name` Attribut auf `Matthias Eckhart` .

Dokumente einfügen

Wenn Sie Dokumente in eine Sammlung einfügen möchten, führen Sie Folgendes aus:

```
db.collection.insert({name: 'Matthias Eckhart'});
```

Dokumente aktualisieren

Wenn Sie Dokumente aktualisieren möchten, verwenden Sie die `update()` -Methode, zum Beispiel:

```
db.collection.update({name: 'Matthias Eckhart'}, {$set: {name: 'John Doe'}});
```

Die Ausführung dieses Befehls wird , indem der Wert eines **einzelnen** Dokuments aktualisieren `John Doe` für den `name` (zunächst der Wert war `Matthias Eckhart`).

Wenn Sie **alle** Dokumente aktualisieren möchten, die bestimmten Kriterien entsprechen, setzen Sie den `multi` Parameter auf `true` . Beispiel:

```
db.collection.update({name: 'Matthias Eckhart'}, {$set: {name: 'John Doe'}}, {multi: true});
```

Nun werden alle Dokumente in der Sammlung , die zunächst den hatte `name` Attribut auf `Matthias Eckhart` wurden aktualisiert `John Doe` .

Dokumente löschen

Dokumente können einfach mit der `remove()` -Methode entfernt werden. Beispiel:

```
db.collection.remove({name: 'Matthias Eckhart'});
```

Dadurch werden alle Dokumente entfernen, die den Wert in dem angegebenen übereinstimmen `name` Feld.

MongoDB online lesen: https://riptutorial.com/de/meteor/topic/1874/mongodb

Kapitel 30: MongoDB-Aggregation

Bemerkungen

Server Aggregation

Durchschnittliche Aggregationsabfragen in Meteor

Ist es möglich, eine "echte" Mongodb-Bibliothek für die Verwendung auf der * Server * -Seite nur in Meteor 0.6 zu packen

Client Aggregation (Minimongo)

https://github.com/utunga/pocketmeteor/tree/master/packages/mongowrapper

Examples

Server Aggregation

Andrew Maos Lösung. Durchschnittliche Aggregationsabfragen in Meteor

```
Meteor.publish("someAggregation", function (args) {
    var sub = this;
    // This works for Meteor 0.6.5
    var db = MongoInternals.defaultRemoteCollectionDriver().mongo.db;

    // Your arguments to Mongo's aggregation. Make these however you want.
    var pipeline = [
        { $match: doSomethingWith(args) },
        { $group: {
            _id: whatWeAreGroupingWith(args),
            count: { $sum: 1 }
        }}
    ];

    db.collection("server_collection_name").aggregate(
        pipeline,
        // Need to wrap the callback so it gets called in a Fiber.
        Meteor.bindEnvironment(
            function(err, result) {
                // Add each of the results to the subscription.
                _.each(result, function(e) {
                    // Generate a random disposable id for aggregated documents
                    sub.added("client_collection_name", Random.id(), {
                        key: e._id.somethingOfInterest,
                        count: e.count
                    });
                });
                sub.ready();
            },
            function(error) {
                Meteor._debug( "Error doing aggregation: " + error);
            }
```

```
        )
    );
});
```

Aggregation in einer Servermethode

Eine andere Möglichkeit, Aggregationen `Mongo.Collection#rawCollection()` ist die Verwendung der `Mongo.Collection#rawCollection()`

Dies kann nur auf dem Server ausgeführt werden.

Hier ein Beispiel, das Sie in Meteor 1.3 und höher verwenden können:

```
Meteor.methods({
   'aggregateUsers'(someId) {
      const collection = MyCollection.rawCollection()
      const aggregate = Meteor.wrapAsync(collection.aggregate, collection)

      const match = { age: { $gte: 25 } }
      const group = { _id:'$age', totalUsers: { $sum: 1 } }

      const results = aggregate([
         { $match: match },
         { $group: group }
      ])

      return results
   }
})
```

MongoDB-Aggregation online lesen: https://riptutorial.com/de/meteor/topic/4199/mongodb-aggregation

Kapitel 31: Mongo-Sammlungen

Bemerkungen

Eine sinnvolle Möglichkeit, über die Mongo-Sammlungen nachzudenken, besteht in Bezug auf Wer, Was, Wann, Wo, Warum und Wie. Mongo hat die folgenden Optimierungen für verschiedene Datentypen:

Wo - GeoJSON
Wann - ObjectID-Zeitstempel
Wer - Meteor-Konto-Zeichenfolgen
Wie - JSON für Entscheidungsbäume

Damit bleibt das Standarddokument in Mongo ungefähr ein "Was".

Examples

Datensätze in einer älteren Datenbank erstellen

Sie können das normale Mongo-Format standardmäßig verwenden, indem Sie Ihre Sammlungen mit dem Feld idGeneration definieren.

```
MyCollection = new Meteor.Collection('mycollection', {idGeneration : 'MONGO'});
```

Daten in ein Dokument einfügen

Viele Anfänger von Mongo haben Schwierigkeiten mit den Grundlagen, wie z. B. das Einfügen eines Arrays, eines Datums, eines Boolean, einer Sitzungsvariablen usw. in einen Dokumentensatz. Dieses Beispiel enthält einige Hinweise zu grundlegenden Dateneingaben.

```
Todos.insert({
  text: "foo",                      // String
  listId: Session.get('list_id'),   // String
  value: parseInt(2),               // Number
  done: false,                      // Boolean
  createdAt: new Date(),            // Dimestamp
  timestamp: (new Date()).getTime(),  // Time
  tags: []                          // Array
});
```

Abrufen der _id des zuletzt erstellten Dokuments

Sie können es entweder synchron bekommen:

```
var docId = Todos.insert({text: 'foo'});
console.log(docId);
```

Oder asynchron:

```
Todos.insert({text: 'foo'}, function(error, docId){
  console.log(docId);
});
```

Zeitreihendaten

Die Verwendung von MongoDB für Zeitreihendaten ist ein sehr gut dokumentiertes und etabliertes Anwendungsbeispiel mit offiziellen Whitepapers und Präsentationen. Lesen und sehen Sie sich die offizielle Dokumentation von MongoDB an, bevor Sie versuchen, Ihre eigenen Schemata für Zeitreihendaten zu erfinden.

MongoDB für Zeitreihendaten

Im Allgemeinen sollten Sie "Buckets" für Ihre Zeitreihendaten erstellen:

```
DailyStats.insert({
   "date" : moment().format("MM-DD-YYYY"),
   "dateIncrement" : moment().format("YYYYMMDD"),
   "dailyTotal" : 0,
   'bucketA': 0,
   'bucketB': 0,
   'bucketC': 0
   });
```

Erhöhen Sie dann diese Buckets, während Daten in Ihre Anwendung eingespeist werden. Dieses Inkrement kann in eine Meteor-Methode, einen Sammlungsbeobachter, einen REST-API-Endpunkt und verschiedene andere Stellen eingefügt werden.

```
DailyStats.update({_id: doc._id}, {$inc: {bucketA: 1} });
```

Ein vollständigeres Meteor-Beispiel finden Sie in den Beispielen des klinischen Meteor-Tracks:

Echtzeit-Analytics-Pipeline
Klinischer Meteor - Schaubilder - Dailystats

Filtern mit Regexes

Einfaches Muster zum Filtern von Abonnements auf dem Server unter Verwendung von regulären Ausdrücken, reaktiven Sitzungsvariablen und automatischen Ausfällen.

```
// create our collection
WordList =  new Meteor.Collection("wordlist");

// and a default session variable to hold the value we're searching for
Session.setDefault('dictionary_search', '');

Meteor.isClient(function(){
    // we create a reactive context that will rerun items when a Session variable gets updated
```

```
    Deps.autorun(function(){
        // and create a subscription that will get re-subscribe to when Session variable gets
updated
        Meteor.subscribe('wordlist', Session.get('dictionary_search'));
    });

    Template.dictionaryIndexTemplate.events({
        'keyup #dictionarySearchInput': function(evt,tmpl){
            // we set the Session variable with the value of our input when it changes
            Session.set('dictionary_search', $('#dictionarySearchInput').val());
        },
        'click #dictionarySearchInput':function(){
            // and clear the session variable when we enter the input
            Session.set('dictionary_search', '');
        },
    });
});
Meteor.isServer(function(){
    Meteor.publish('wordlist', function (word_search) {
        // this query gets rerun whenever the client subscribes to this publication
        return WordList.find({
            // and here we do our regex search
            Word: { $regex: word_search, $options: 'i' }
        },{limit: 100});
    });
});
```

Und der HTML-Code, der auf dem Client verwendet wird:

```
<input id="dictionarySearchInput" type="text" placeholder="Filter..." value="hello"></input>
```

Dieses Muster selbst ist ziemlich geradlinig, aber die Regexen mögen nicht sein. Wenn Sie sich mit Regex nicht auskennen, finden Sie hier einige nützliche Tutorials und Links:

Tutorial für reguläre Ausdrücke
Regulärer Ausdruck Spickzettel
Reguläre Ausdrücke in Javascript

Geospatial Collections - Weitere Informationen

Geospatial-Sammlungen umfassen im Allgemeinen das Speichern von GeoJSON in der Mongo-Datenbank, das Streaming dieser Daten an den Client, den Zugriff auf die `window.navigator.geolocation` des Browsers, das Laden einer Karten-API, das Konvertieren von GeoJSON in LatLngs und das Plotten auf der Karte. Am liebsten alles in Echtzeit. Hier finden Sie eine Liste mit Ressourcen, um Ihnen den Einstieg zu erleichtern:

- mongodb speichert seine Daten optimal in geoJSON
- geojson.org
- window.navigator.geolocation
- HTML-Geolocation
- Karten-API-Auswahl
- google.maps.LatLng
- Google map.data.loadGeoJson

- Meteor-Cordova-Geolocation-Hintergrund
- phonegap-googlemaps-plugin
- LatLng
- Karten.Dokumentation
- google.maps.LatLng
- 2dsphere indexes
- Erstellen Sie einen 2dsphere-Index
- Abfrage eines 2dsphere-Index
- Geodaten und Abfragen

Überwachungssammlungsabfragen

Im folgenden Beispiel werden alle Ihre Sammlungsabfragen in Echtzeit an der Server-Konsole protokolliert.

```
Meteor.startup(
  function () {
    var wrappedFind = Meteor.Collection.prototype.find;

    // console.log('[startup] wrapping Collection.find')

    Meteor.Collection.prototype.find = function () {
      // console.log(this._name + '.find', JSON.stringify(arguments))
      return wrappedFind.apply(this, arguments);
    }
  },

  function () {
    var wrappedUpdate = Meteor.Collection.prototype.update;

    // console.log('[startup] wrapping Collection.find')

    Meteor.Collection.prototype.update = function () {
      console.log(this._name + '.update', JSON.stringify(arguments))
      return wrappedUpdate.apply(this, arguments);
    }
  }
);
```

Beobachter & Arbeiterfunktionen

Wenn die Node-Ereignisschleife wie eine Fahrradkette wirkt, ist der serverseitige Sammlungsbeobachter wie ein Umwerfer. Es handelt sich um einen Getriebemechanismus, der bei der Datenerfassung auf die Datenerfassung angewendet wird. Er kann sehr leistungsfähig sein, da alle Rennräder über Schaltwerke verfügen. Aber es ist auch eine Quelle, um das gesamte System zu durchbrechen. Es ist eine reaktionsschnelle Hochgeschwindigkeitsfunktion, die Sie sprengen kann. Sei gewarnt.

```
Meteor.startup(function(){
  console.log('starting worker....');

  var dataCursor = Posts.find({viewsCount: {$exists: true}},{limit:20});
```

```
  var handle = dataCursor.observeChanges({
    added: function (id, record) {
      if(record.viewsCount > 10){
         // run some statistics
         calculateStatistics();

         // or update a value
         Posts.update({_id: id}, {$set:{
           popular: true
         }});

      }
    },
    removed: function () {
      console.log("Lost one.");
    }
  });
});
```

Die Grenze von 20 ist die Größe des Umwerfers. Wie viele Zähne hat er? oder genauer gesagt, wie viele Elemente sich im Cursor befinden, wenn er über die Sammlung läuft. Seien Sie vorsichtig mit dem Schlüsselwort 'var' in dieser Art von Funktion. Schreiben Sie so wenige Objekte wie möglich in den Speicher und konzentrieren Sie sich auf die Wiederverwendung von Objekten innerhalb der hinzugefügten Methode. Wenn das opslog eingeschaltet ist und dieses Ding auf Hochtouren läuft, ist dies ein hervorragender Kandidat für das Freilegen von unangenehmen Speicherverlusten, wenn Objekte schneller auf den Speicherheap geschrieben werden, als der Node-Garbage Collector die Dinge aufräumen kann.

Die obige Lösung lässt sich nicht horizontal gut skalieren, da jede Meteor-Instanz versucht, denselben Datensatz zu aktualisieren. Daher ist eine Art Umgebungserkennung erforderlich, damit diese horizontal skaliert werden kann.

In dem Paket `percolatestudios:synced-cron` ein hervorragendes Beispiel für die Synchronisierung von Service-Mitarbeitern auf mehreren Computern in einem Cluster.
Meteorensynchron-Cron

Mongo-Sammlungen online lesen: https://riptutorial.com/de/meteor/topic/5120/mongo-sammlungen

Kapitel 32: Mongo-Schema-Migrationen

Bemerkungen

Es ist häufig erforderlich, Wartungsskripts in Ihrer Datenbank auszuführen. Felder werden umbenannt; Datenstrukturen werden geändert; Funktionen, die Sie zur Unterstützung verwendet haben, werden entfernt. Dienste werden migriert. Die Liste der Gründe, aus denen Sie möglicherweise Ihr Schema ändern möchten, ist unbegrenzt. Das Warum ist also ziemlich selbsterklärend.

Das Wie ist ein bisschen unbekannter. Für die Personen, die an SQL-Funktionen gewöhnt sind, werden die oben genannten Datenbankskripts merkwürdig aussehen. Beachten Sie jedoch, wie sie alle in Javascript sind und wie sie dieselbe API verwenden, die wir in Meteor sowohl auf dem Server als auch auf dem Client verwenden. Wir haben eine konsistente API über unsere Datenbank, Server und Client.

Führen Sie die Schema-Migrationsbefehle über die Meteor-Mongo-Shell aus:

```
# run meteor
meteor

# access the database shell in a second terminal window
meteor mongo
```

Examples

Versionsfeld zu allen Datensätzen in einer Sammlung hinzufügen

```
db.posts.find().forEach(function(doc){
    db.posts.update({_id: doc._id}, {$set:{'version':'v1.0'}}, false, true);
});
```

Entfernen Sie das Array aus allen Datensätzen in einer Sammlung

```
db.posts.find().forEach(function(doc){
    if(doc.arrayOfObjects){
        // the false, true at the end refers to $upsert, and $multi, respectively
        db.accounts.update({_id: doc._id}, {$unset: {'arrayOfObjects': "" }}, false, true);
    }
});
```

Sammlung umbenennen

```
db.originalName.renameCollection("newName" );
```

Feld suchen, das bestimmte Zeichenfolge enthält

Mit der Kraft des Regex kommt eine große Verantwortung ...

```
db.posts.find({'text': /.*foo.*|.*bar.*/i})
```

Neues Feld aus Alt erstellen

```
db.posts.find().forEach(function(doc){
    if(doc.oldField){
        db.posts.update({_id: doc._id}, {$set:{'newField':doc.oldField}}, false, true);
    }
});
```

Ziehen Sie Objekte aus einem Array und platzieren Sie sie in einem neuen Feld

```
db.posts.find().forEach(function(doc){
    if(doc.commenters){
        var firstCommenter = db.users.findOne({'_id': doc.commenters[0]._id });
        db.clients.update({_id: doc._id}, {$set:{'firstPost': firstCommenter }}, false, true);

        var firstCommenter = db.users.findOne({'_id': doc.commenters[doc.commenters.length -
1]._id });
        db.clients.update({_id: doc._id}, {$set:{'lastPost': object._id }}, false, true);
    }
});
```

Blob-Datensatz aus einer Sammlung in eine andere Sammlung (z. B. Join & Flatten entfernen)

```
db.posts.find().forEach(function(doc){
    if(doc.commentsBlobId){
        var commentsBlob = db.comments.findOne({'_id': commentsBlobId });
        db.posts.update({_id: doc._id}, {$set:{'comments': commentsBlob }}, false, true);
    }
});
```

Stellen Sie sicher, dass das Feld vorhanden ist

```
db.posts.find().forEach(function(doc){
    if(!doc.foo){
        db.posts.update({_id: doc._id}, {$set:{'foo':''}}, false, true);
    }
});
```

Stellen Sie sicher, dass das Feld einen bestimmten Wert hat

```
db.posts.find().forEach(function(doc){
    if(!doc.foo){
        db.posts.update({_id: doc._id}, {$set:{'foo':'bar'}}, false, true);
    }
```

```
});
```

Datensatz entfernen, wenn ein bestimmtes Feld ein bestimmter Wert ist

```
db.posts.find().forEach(function(doc){
    if(doc.foo === 'bar'){
        db.posts.remove({_id: doc._id});
    }
});
```

Ändern Sie den spezifischen Wert des Feldes in Neuer Wert

```
db.posts.find().forEach(function(doc){
    if(doc.foo === 'bar'){
        db.posts.update({_id: doc._id}, {$set:{'foo':'squee'}}, false, true);
    }
});
```

Spezifisches Feld auf Null setzen

```
db.posts.find().forEach(function(doc){
    if(doc.oldfield){
        // the false, true at the end refers to $upsert, and $multi, respectively
        db.accounts.update({_id: doc._id}, {$unset: {'oldfield': "" }}, false, true);
    }
});
```

Konvertieren Sie ObjectId in String

```
db.posts.find().forEach(function(doc){
     db.accounts.update({_id: doc._id}, {$set: {'_id': doc._id.str }}, false, true);
});
```

Konvertieren Sie Feldwerte von Zahlen in Strings

```
var newvalue = "";
db.posts.find().forEach(function(doc){
     if(doc.foo){
         newvalue = '"' + doc.foo + '"';
         db.accounts.update({_id: doc._id}, {$set: {'doc.foo': newvalue}});
     }
});
```

Konvertieren Sie Feldwerte von Strings in Zahlen

```
var newvalue = null;
db.posts.find().forEach(function(doc){
     if(doc.foo){
         newvalue = '"' + doc.foo + '"';
```

```
        db.accounts.update({_id: doc._id}, {$set: {'doc.foo': newvalue}});
    }
});
```

Erstellen Sie einen Zeitstempel aus einer ObjectID im Feld _id

```
db.posts.find().forEach(function(doc){
    if(doc._id){
        db.posts.update({_id: doc._id}, {$set:{ timestamp: new
Date(parseInt(doc._id.str.slice(0,8), 16) *1000) }}, false, true);
    }
});
```

Erstellen Sie eine ObjectID aus einem Datumsobjekt

```
var timestamp = Math.floor(new Date(1974, 6, 25).getTime() / 1000);
var hex       = ('00000000' + timestamp.toString(16)).substr(-8); // zero padding
var objectId  = new ObjectId(hex + new ObjectId().str.substring(8));
```

Finden Sie alle Datensätze, die Elemente in einem Array enthalten

Was wir hier tun, referenziert den Array-Index mit Punktnotation

```
db.posts.find({"tags.0": {$exists: true }})
```

Mongo-Schema-Migrationen online lesen: https://riptutorial.com/de/meteor/topic/3708/mongo-schema-migrationen

Kapitel 33: Nightwatch - Konfiguration und Setup

Bemerkungen

Nightwatch bietet seit v0.5 Tagen Akzeptanz- und End-to-End-Tests für Meteor-Apps an und hat Migrationen von PHP nach Spark nach Blaze und React verwaltet. und alle wichtigen Continuous Integration-Plattformen. Weitere Hilfe finden Sie unter:

Nightwatch-API-Dokumentation
Nightwatch.js Google-Gruppe

Examples

Aufbau

Der Hauptgrund dafür, dass Nightwatch so leistungsfähig ist, liegt in der hervorragenden Konfigurationsdatei. Im Gegensatz zu den meisten anderen Test-Frameworks ist Nightwatch vollständig konfigurierbar und kann an verschiedene Umgebungen und Technologiestacks angepasst werden.

.meteor / nightwatch.json

Die folgende Konfigurationsdatei ist für Meteor v1.3 und höher und unterstützt zwei Umgebungen ... eine `default` , die den chromedriver Browser startet, und eine `phantom` Umgebung , die die Tests in einer Headless - Umgebung ausgeführt wird .

```
{
  "nightwatch": {
    "version": "0.9.8"
  },
  "src_folders": [
    "./tests/nightwatch/walkthroughs"
  ],
  "custom_commands_path": [
    "./tests/nightwatch/commands"
  ],
  "custom_assertions_path": [
    "./tests/nightwatch/assertions"
  ],
  "output_folder": "./tests/nightwatch/reports",
  "page_objects_path": "./tests/nightwatch/pages",
  "globals_path": "./tests/nightwatch/globals.json",
  "selenium": {
    "start_process": true,
    "server_path": "./node_modules/starrynight/node_modules/selenium-server-standalone-
jar/jar/selenium-server-standalone-2.45.0.jar",
    "log_path": "tests/nightwatch/logs",
    "host": "127.0.0.1",
```

```
    "port": 4444,
    "cli_args": {
      "webdriver.chrome.driver":
"./node_modules/starrynight/node_modules/chromedriver/bin/chromedriver"
    }
  },
  "test_settings": {
    "default": {
      "launch_url": "http://localhost:5000",
      "selenium_host": "127.0.0.1",
      "selenium_port": 4444,
      "pathname": "/wd/hub",
      "silent": true,
      "disable_colors": false,
      "firefox_profile": false,
      "ie_driver": "",
      "screenshots": {
        "enabled": false,
        "path": "./tests/nightwatch/screenshots"
      },
      "desiredCapabilities": {
        "browserName": "chrome",
        "javascriptEnabled": true,
        "acceptSslCerts": true,
        "loggingPrefs": {
          "browser": "ALL"
        }
      },
      "exclude": "./tests/nightwatch/unittests/*",
      "persist_globals": true,
      "detailed_output": false
    },
    "phantom": {
      "desiredCapabilities": {
        "browserName": "phantomjs",
        "javascriptEnabled": true,
        "databaseEnabled": false,
        "locationContextEnabled": false,
        "applicationCacheEnabled": false,
        "browserConnectionEnabled": false,
        "webStorageEnabled": false,
        "acceptSslCerts": true,
        "rotatable": false,
        "nativeEvents": false,
        "phantomjs.binary.path": "./node_modules/starrynight/node_modules/phantomjs-
prebuilt/bin/phantomjs"
      }
    },
    "unittests": {
      "selenium": {
        "start_process": false,
        "start_session": false
      },
      "filter": "./tests/nightwatch/unittests/*",
      "exclude": ""
    }
  }
}
```

Installation & Nutzung

Um Nightwatch zum Laufen zu bringen, benötigen Sie eine lokale Kopie von **Selen** , einem Befehls- und Steuerungsserver, der automatisierte Browserinstanzen verwaltet. Sie benötigen außerdem einen Webbrowser, den Selen steuern kann, beispielsweise **Chromdriver** oder **Phantomjs** .

Fügen Sie Ihrer `package.json` die folgenden devDependencies `package.json` :

```
{
  "devDependencies": {
    "nightwatch": "0.9.8",
    "selenium-server-standalone-jar": "2.45.0",
    "chromedriver": "2.19.0",
    "phantomjs-prebuilt": "2.1.12"
  }
}
```

Dann installieren Sie alle Abhängigkeiten.

```
cd myapp
meteor npm install
```

Sie sollten Nightwatch dann mit den folgenden Befehlen ausführen können:

```
nightwatch -c .meteor/nightwatch.json
nightwatch -c .meteor/nightwatch.json --env phantom
```

Wenn Sie noch keine Tests geschrieben oder Ihre Ordnerstruktur eingerichtet haben, werden möglicherweise Fehler angezeigt.

Startskripte einrichten

Im Stammverzeichnis Ihrer Anwendung sollte sich eine `package.json` Datei befinden, in der Sie Skripts und devDependencies definieren können.

```
{
  "name": "myapp",
  "version": "1.0.0",
  "scripts": {
    "start": "meteor --settings settings-development.json",
    "nightwatch": "nightwatch -c .meteor/nightwatch.json",
    "phantom": "nightwatch -c .meteor/nightwatch.json --env phantom",
  }
}
```

Anschließend können Sie Nightwatch mit den folgenden Befehlen starten:

```
meteor npm run-script nightwatch
meteor npm run-script phantom
```

In diesem Beispiel wäre es fast einfacher, `nightwatch -c .meteor/nightwatch.json` . Mit komplexeren Befehlen, mit komplexen Umgebungsvariablen, Optionen und Einstellungen wird dies jedoch sehr

nützlich, um Devops-Skripts für ein Team anzugeben.

Ordnerstruktur

Bei einer grundlegenden Nightwatch-Installation für Meteor werden die folgenden Verzeichnisse und Dateien installiert.

```
/myapp
/myapp/.meteor/nightwatch.json
/client/main.html
/client/main.js
/client/main.css
/tests
/tests/nightwatch
/tests/nightwatch/assertions
/tests/nightwatch/commands
/tests/nightwatch/data
/tests/nightwatch/logs
/tests/nightwatch/pages
/tests/nightwatch/reports
/tests/nightwatch/screenshots
/tests/nightwatch/walkthroughs
/tests/nightwatch/walkthroughs/critical_path.js
/tests/nightwatch/globals.json
```

Datengesteuertes Testen

Nightwatch akzeptiert eine zweite Konfigurationsdatei von `globals.json` , die Daten in den Testläufer selbst injiziert. `Meteor.settings` ähnelt der `Meteor.settings` der `Meteor.settings` Daten von der Befehlszeile aus in der App verfügbar macht.

globals.json

```
{
  "default" : {
    "url" : "http://localhost:3000",
    "user": {
      "name": "Jane Doe",
      "username" : "janedoe",
      "password" : "janedoe123",
      "email" : "janedoe@test.org",
      "userId": null
    }
  },
  "circle" : {
    "url" : "http://localhost:3000",
    "user": {
      "name": "Jane Doe",
      "username" : "janedoe",
      "password" : "janedoe123",
      "email" : "janedoe@test.org"
      "userId": null
    }
  },
  "galaxy" : {
    "url" : "http://myapp.meteorapp.com",
```

```
    "user": {
      "name": "Jane Doe",
      "username" : "janedoe",
      "password" : "janedoe123",
      "email" : "janedoe@test.org"
      "userId": null
    }
  }
}
```

Sie können dann Ihre Tests schreiben, die nicht mit bestimmten Benutzern, Kennwörtern, Sucheingaben usw. fest codiert sind.

```
module.exports = {
  "Login App" : function (client) {
    client
      .url(client.globals.url)
      .login(client.globals.user.email, client.globals.user.password)
      .end();
  }
};
```

Nightwatch - Konfiguration und Setup online lesen:
https://riptutorial.com/de/meteor/topic/5901/nightwatch---konfiguration-und-setup

Kapitel 34: Offline-Apps

Bemerkungen

Weitere Appcache-Forschung

http://www.html5rocks.com/de/tutorials/indexeddb/todo/
http://grinninggecko.com/2011/04/22/erhöhende-chromes-offline-application-cache-storage-limit/
http://www.html5rocks.com/de/tutorials/offline/quota-research/
https://developers.google.com/chrome/apps/docs/developers_guide?csw=1#installing
https://developers.google.com/chrome/apps/docs/developers_guide?csw=1#manifest

Examples

Meteor.status ()

Wenn Sie Ihre Meteor-App offline nehmen, müssen Sie zunächst visuell darauf hinweisen, ob die lokale Client-App mit dem Server verbunden ist oder nicht. Es gibt viele Möglichkeiten, dies zu tun, aber die einfachste Möglichkeit ist, wahrscheinlich so etwas zu tun:

```
Template.registerHelper('getOnlineStatus', function(){
  return Meteor.status().status;
});

Template.registerHelper('getOnlineColor', function(){
  if(Meteor.status().status === "connected"){
    return "green";
  }else{
    return "orange";
  }
});
```

```
  <div id="onlineStatus" class="{{getOnlineColor}}">
    {{getOnlineStatus}}
  </div>
```

```
.green{
  color: green;
}
.orange{
  color: orange;
}
```

Appcache aktivieren

Einer der einfacheren Schritte ist das Hinzufügen des Appcache. Appcache ermöglicht das Laden Ihrer Anwendungsinhalte, auch wenn kein Internetzugang vorhanden ist. Sie können keine Daten von Ihren Mongo-Servern erhalten, der statische Inhalt und die Assets sind jedoch offline

verfügbar.

```
meteor add appcache
```

Aktivieren Sie GroundDB

Schließlich möchten wir, dass einige unserer dynamischen Daten offline gespeichert werden.

```
meteor add ground:db
```

```
Lists = new Meteor.Collection("lists");
GroundDB(Lists);

Todos = new Meteor.Collection("todos")
GroundDB(Todos);
```

Dinge, vor denen man vorsichtig sein sollte

- Der Appcache führt zu Verwirrung in Ihrem Entwicklungsworkflow, da die automatischen Aktualisierungsfunktionen von Meteor ausgeblendet werden. Wenn Sie die Serverkomponente Ihrer App deaktivieren, funktioniert der Clientbereich in Ihrem Browser weiterhin. Das ist eine gute Sache! Sie erhalten jedoch nicht das unmittelbare Feedback, dass Ihre App deaktiviert wurde oder dass Updates vorhanden sind.
- Versuchen Sie, den Inkognito-Modus von Chrome zu verwenden, wenn Sie Ihre App entwickeln, da sie keinen Appcache verwendet.
- GroundDB funktioniert mit IronRouter nicht besonders gut.

Offline-Apps online lesen: https://riptutorial.com/de/meteor/topic/3375/offline-apps

Kapitel 35: Polymer mit Meteor verwenden

Examples

Mit Differential: vulkanisieren

Stellen Sie im Stammverzeichnis Ihres Projekts sicher, dass Bower installiert ist (`npm install -g bower`), und führen Sie Bower `bower init` . Dadurch wird eine `bower.json` Datei im Verzeichnis Ihres Projekts erstellt.

Erstellen Sie eine neue Datei mit dem Namen `.bowerrc` in Ihrem Stammverzeichnis. Es sollte Folgendes enthalten:

```
{
  "directory": "public/bower_components"
}
```

Dadurch kann Bower wissen, dass Komponenten im Ordner " `bower_components` " im öffentlichen Verzeichnis Ihrer App `bower_components` .

Fügen Sie nun die Polymerkomponenten hinzu, die Sie mit Ihrer App verwenden möchten.

Installieren Sie im Stammverzeichnis Ihrer App jede Komponente, die Sie verwenden möchten.

```
bower install --save PolymerElements/paper-button#^1.0.0 PolymerElements/paper-checkbox#^1.0.0
```

Fügen Sie Ihrem Projekt Vulcanize hinzu

```
Meteor add differential:vulcanize
```

Erstellen Sie eine neue Datei mit dem Namen config.vulcanize im Stammverzeichnis Ihres Projekts. Es sollte Folgendes enthalten:

```
{
    "polyfill": "/bower_components/webcomponentsjs/webcomponents.min.js",
    "useShadowDom": true, // optional, defaults to shady dom (polymer default)
    "imports": [
        "/bower_components/paper-button/paper-button.html",
        "/bower_components/paper-checkbox/paper-checkbox.html"
    ]
}
```

`"imports"` sollte jede Komponente auflisten, die Sie in Ihrer App verwenden werden.

Komponenten, die Sie in Ihre Blaze-Vorlagen importiert haben, können Sie jetzt wie jedes andere Element verwenden:

```
<template name="example">
```

```
    <div>
        this is a material design button: <paper-button></paper-button>
        this is a material design checkbox: <paper-checkbox></paper-checkbox>
    </div>
</template>
```

Polymer mit Meteor verwenden online lesen: https://riptutorial.com/de/meteor/topic/4598/polymer-mit-meteor-verwenden

Kapitel 36: Protokollierung

Examples

Grundlegende serverseitige Protokollierung

Der erste Schritt zur Protokollierung besteht darin, Meteor von der Shell aus auszuführen, und die Serverprotokolle werden in der Befehlskonsole abgerufen.

```
meteor
```

Der nächste Schritt besteht darin, den Inhalt von std_out und std_err wie folgt in eine Protokolldatei zu leiten:

```
meteor > my_app_log.log 2> my_app_err.log
```

Tools zur clientseitigen Protokollierung

Sobald Sie sich auf der Serverseite angemeldet haben, können Sie zur Clientseite wechseln. Wenn Sie die Konsolen-API noch nicht erforscht haben, bereiten Sie sich auf eine Belohnung vor. Mit der integrierten Console-API, die in jeder Chrome- und Safari-Installation enthalten ist, können Sie alle möglichen Dinge tun. So sehr, dass Sie Winston oder andere Protokollierungs-Frameworks möglicherweise nicht benötigen.

Als Erstes sollten Sie clientseitige Protokollierungs- und Entwicklertools installieren. Chrome und Safari werden mitgeliefert, Firefox erfordert jedoch die Firebug-Erweiterung.

Firebug-Erweiterung

Dann sollten Sie sich die Dokumentation zur Console API ansehen. Die folgenden zwei Dokumente sind von unschätzbarem Wert für die Protokollierung der Lernkonsolen.

Chrome-Entwicklertools

Firebug (Client)

Erweiterte Serverprotokollierungstools

Nachdem Sie sowohl die serverseitige Protokollierung als auch die clientseitigen Entwicklungstools ausgeführt haben, können Sie sich die Meteor-spezifischen Erweiterungen wie die Meteor Chrome DevTools-Erweiterung anschauen. So können Sie tatsächlich die Serverprotokollierung im Client beobachten! Weil die Datenbank überall ist. Da ist die Protokollierung.

Chrome DevTools Extension (Server)

Protokollierungsfehler bei Datenbankklappe

Das folgende Beispiel reicht von 0,5 bis 0,7 Tagen und veranschaulicht, wie ein Fehler protokolliert wird, wenn die Datenbank den clientseitigen Cursor noch nicht gefüllt hat.

```
Template.landingPage.postsList = function(){
  try{
    return Posts.find();
  }catch(error){
    //color code the error (red)
    console.error(error);
  }
}
```

Protokollierung von Informationen zum Datenkontext in einem Vorlagenhelfer

Im Folgenden wird die Chrome-Protokollierungs-API verwendet. Wenn die `.group()` in mehreren Vorlagen verwendet wird, werden die Konsolenprotokolle aus verschiedenen Vorlagen grafisch in einer hierarchischen `.group()` organisiert.

Sie können auch sehen, wie Sie den aktuellen Datenkontext untersuchen und wie Sie Daten in einer Reihe festlegen.

```
Template.landingPage.getId = function(){
  // using a group block to illustrate function scoping
  console.group('coolFunction');

  // inspect the current data object that landingPage is using
  console.log(this);

  // inspect a specific field of the locally scoped data object
  console.log(JSON.stringify(this._id);

  // close the function scope
  console.groupEnd();
  return this._id;
}
```

Protokollierung von Ereignissen und Benutzerinteraktionen

Ein einfaches Beispiel für die Verwendung der Chrome-Protokollierungs-API.

```
Template.landingPage.events({
  'click .selectItemButton':function(){
    // color code and count the user interaction (blue)
    console.count('click .selectItemButton');
  }
});
```

Protokollierung mit Variablen auf Protokollebene

Die Protokollierung kann die Konsole oft unübersichtlich machen. Daher ist es üblich,

Protokollebenen festzulegen, um zu steuern, welche Details der Daten protokolliert werden. Ein übliches Muster ist das Angeben von Variablen auf Protokollebene.

```
var DEBUG = false;
var TRACE = false;
Template.landingPage.events({
  'click .selectItemButton':function(){
    TRACE && console.count('click .selectItemButton');

    Meteor.call('niftyAction', function(errorMessage, result){
        if(errorMessage){
            DEBUG && console.error(errorMessage);
        }
    });
  }
});
```

Deaktivieren Sie die Protokollierung in der Produktion

Einige Teams möchten, dass sie Konsolenprotokollanweisungen in ihrem Code belassen möchten, diese jedoch nicht in der Produktion anzeigen. Sie setzen die Protokollierungsfunktionen außer Kraft, wenn keine Variable festgelegt ist (möglicherweise eine Umgebungsvariable). Darüber hinaus kann dies in einigen Situationen als Sicherheitsmerkmal gelten.

```
if (!DEBUG_MODE_ON) {
    console = console || {};
    console.log = function(){};

    console.log = function(){};
    console.error = function(){};
    console.count = function(){};
    console.info = function(){};
}
```

Winston

Wenn Sie etwas Stärkeres als die Standardprotokollierungsoptionen benötigen, sollten Sie sich ein Tool wie Winston ansehen. Gehen Sie zu Atmosphere und suchen Sie einfach nach einem der vielen verfügbaren Winston-Pakete.

https://atmospherejs.com/?q=winston

Seien Sie jedoch gewarnt: Winston ist ein ausgereiftes Produkt, und obwohl es viele Funktionen bietet, fügt es Ihrer Anwendung eine komplexere Ebene hinzu.

Loglevel

Besonders zu erwähnen ist das von der Community entwickelte LogLevel-Paket. Es scheint ein Gleichgewicht zu sein zwischen geringem Gewicht und einfacher Handhabung, während es gut mit der Bündel-Pipeline von Meteor zusammenarbeitet und Zeilennummern und Dateinamen beibehält.

https://atmospherejs.com/practicalmeteor/loglevel

Protokollierung online lesen: https://riptutorial.com/de/meteor/topic/3376/protokollierung

Kapitel 37: Reaktiv (Vars & Wörterbücher)

Examples

Reaktive Abfrage

Beispielcode:

In main.html

```
<template name="test">
     <input type="checkbox" id="checkbox1" name="name" value="data">Check Me
      {{showData}}
</template>
```

In Main.js

```
 var check_status='';
 //Reactive Var Initialization
 Template.main.onCreated(function (){
       check_status=new ReactiveVar({});

 });

 Template.main.helpers({
       showData : function(){
           return Collection.find(check_status.get());
       }
 });

 Template.main.events({
      "change #checkbox1" : function(){
              check_status.set({field: 'data'});
       }
 });
```

Erläuterung:

Wenn wir das reaktive var `check_status` initialisieren, `check_status` wir den Wert gleich `{}` . Im `Collection.find(check_status.get())` zum Zeitpunkt des Renderns die gleichen Daten an die Abfrage `Collection.find(check_status.get())` die so gut ist wie *alle* Daten `Collection.find(check_status.get())` .

Sobald Sie auf das Kontrollkästchen klicken, wird das in `Template.main.events` beschriebene Ereignis ausgelöst, das den Wert von `check_status` auf `{field: data}` . Da dies eine *reaktive* `showData` ist, wird die `showData` Vorlage erneut ausgeführt, und diesmal lautet die Abfrage `Collection.find({field: data})` werden also nur Felder zurückgegeben, bei denen das `field 'data'` hat.

Bevor Sie diese Befehle verwenden, müssen Sie das `reactive var` Paket `meteor add reactive-var` (

`meteor add reactive-var`).

Reaktiv (Vars & Wörterbücher) online lesen: https://riptutorial.com/de/meteor/topic/6535/reaktiv--vars--amp--worterbucher-

Kapitel 38: Replikatsätze und Sharding

Bemerkungen

Für nicht vertraute Personen ist ein Replikatsatz als redundante Konfiguration von drei Servern definiert. Eine Sharded-Datenbank ist eine horizontal gescannte Datenbank, in der jeder Shard als Replikatsatz definiert ist. Daher umfasst ein Mongo-Cluster mit Shards mindestens 11 Server für einen 2-Shard-Cluster und erhöht sich für jedes weitere Shard um drei Server. Ein Sharded-Cluster hat also immer Serverinstanzen von 11, 14, 17, 20, 23 usw. Das heißt, es gibt 2 Shards mit jeweils 3 Servern, 3 weitere Konfigurationssteuerungen und 2 Router. Insgesamt 11 Server für einen 2-Shard-Cluster.

Examples

Replikatsatz-Schnellstart

Bauen Sie **drei** Server mit beliebiger physischer oder virtueller Hardware auf. (In diesem Lernprogramm wird davon ausgegangen, dass Sie Ubuntu als Betriebssystem verwenden.) Wiederholen Sie dann die folgenden Anweisungen dreimal ... einmal für jeden Server.

```
# add the names of each server to the host file of each server
sudo nano /etc/hosts
  10.123.10.101 mongo-a
  10.123.10.102 mongo-b
  10.123.10.103 mongo-c

# install mongodb on the server
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee
/etc/apt/sources.list.d/mongodb.list
sudo apt-get update
sudo apt-get install mongodb-10gen


# create the /data/ directories
sudo mkdir /data
sudo mkdir /data/logs
sudo mkdir /data/db

# make sure the mongodb user and group have access to our custom directories
sudo chown -R mongodb:mongodb /data

# edit the mongo upstart file in /etc/init/mongodb.conf
sudo nano /etc/init/mongodb.conf
  start on started mountall
  stop on shutdown
  respawn
  respawn limit 99 5
  setuid mongodb
  setgid mongodb
  script
    exec /usr/bin/mongod --config /etc/mongodb.conf >> /data/logs/mongo-a.log 2>&1
```

```
  end script


# edit mongodb configuration file
sudo nano /etc/mongodb.conf
    dbpath=/data/db
    logpath=/data/logs/mongod.log
    logappend=true
    port=27017
    noauth=true
    replSet=meteor
    fork=true

# add a mongo log-rotation file
sudo nano /etc/logrotate.d/mongod
  /data/logs/*.log {
    daily
    rotate 30
    compress
    dateext
    missingok
    notifempty
    sharedscripts
    copytruncate
    postrotate
        /bin/kill -SIGUSR1 `cat /data/db/mongod.lock 2> /dev/null` 2> /dev/null || true
    endscript
  }

# make sure mongod service is started and running
sudo service mongodb start
sudo reboot
```

Replikatset-Konfiguration

Gehen Sie dann in die Mongo-Shell und initiieren Sie den Replikatsatz,

```
meteor mongo

  > rs.initiate()
  PRIMARY> rs.add("mongo-a")
  PRIMARY> rs.add("mongo-b")
  PRIMARY> rs.add("mongo-c")
  PRIMARY> rs.setReadPref('secondaryPreferred')
```

Replikatsätze und Sharding online lesen: https://riptutorial.com/de/meteor/topic/4332/replikatsatze-und-sharding

Kapitel 39: Routing

Examples

Routing mit dem Iron Router

Installieren Sie den Iron Router

Vom Terminal:

```
meteor add iron:router
```

Grundlegende Einstellung

```
Router.configure({
    //Any template in your routes will render to the {{> yield}} you put inside your layout
template
    layoutTemplate: 'layout',
    loadingTemplate: 'loading'
});
```

Rendern ohne Daten

```
//this is equal to home page
Router.route('/', function (){
    this.render('home')
});

Router.route('/some-route', function () {
    this.render('template-name');
});
```

Rendern mit Daten und Parametern

```
Router.route('/items/:_id', function () {
    this.render('itemPage', {
        data: function() {
            return Items.findOne({_id: this.params._id})
        }
    });
});
```

Rendern Sie auf einen sekundären Ertrag

```
Router.route('/one-route/route', function() {
    //template 'oneTemplate' has {{> yield 'secondary'}} in HTML
    this.render('oneTemplate');

    //this yields to the secondary place
    this.render('anotherTemplate', {
        to: 'secondary'
```

```
    });

    //note that you can write a route  for '/one-route'
    //then another for '/one-route/route' which will function exactly like above.
});
```

Abonnieren Sie und warten Sie auf Daten, bevor Sie die Vorlage rendern

```
Router.route('/waiting-first', {
    waitOn: function() {
        //subscribes to a publication
        //shows loading template until subscription is ready
        return Meteor.subscribe('somePublication')
    },

    action: function() {
        //render like above examples
    }
});
```

Abonnieren Sie mehrere Publikationen und warten Sie auf Daten, bevor Sie die Vorlage für das Rendern erstellen

```
Router.route('/waiting-first', {
    waitOn: function() {
        //subscribes to a publication
        //shows loading template until subscription is ready
        return [Meteor.subscribe('somePublication1'),Meteor.subscribe('somePublication2')];
    },

    action: function() {
        //render like above examples
    }
});
```

Leitfaden für Iron Router: http://iron-meteor.github.io/iron-router/

Mit FlowRouter

FlowRouter ist im Vergleich zum Iron Router modularer.

Installieren Sie FlowRouter

```
meteor add kadira:flow-router
```

Vorlage rendern

Insbesondere müssen Sie manuell ein Layout-Rendering-Paket hinzufügen, um eine Verknüpfung mit Ihrer Rendering-Engine herzustellen:

- Blaze Layout für Blaze: `meteor add kadira:blaze-layout`
- React Layout für React: `meteor add kadira:react-layout` React `meteor add kadira:react-layout`

Dann können Sie durch dynamisches Templating rendern (im Fall von Blaze):

```
<template name="mainLayout">
  {{> Template.dynamic template=area}}
</template>
```

```
FlowRouter.route('/blog/:postId', {
  action: function (params) {
    BlazeLayout.render("mainLayout", {
      area: "blog"
    });
  }
});
```

Rendern einer Vorlage mit Parametern und / oder Abfrage

Die Parameter werden wie beim Iron Router auf der Route angegeben:

```
FlowRouter.route("/blog/:catId/:postId", {
  name: "blogPostRoute",
  action: function (params) {
    //...
  }
})
```

Die Parameter werden jedoch nicht als Datenkontext an die untergeordnete Vorlage übergeben. Stattdessen muss die untergeordnete Vorlage sie lesen:

```
// url: /blog/travel/france?showcomments=yes
var catId = FlowRouter.getParam("catId"); // returns "travel"
var postId = FlowRouter.getParam("postId"); // returns "france"

var color = FlowRouter.getQueryParam("showcomments"); // returns "yes"
```

Routing online lesen: https://riptutorial.com/de/meteor/topic/5119/routing

Kapitel 40: Umgebungserkennung

Examples

Erweiterte Umgebungskonfigurationen

Für komplexere Anwendungen sollten Sie ein `` settings.json`-Objekt mit mehreren Umgebungsvariablen erstellen.

```
if(Meteor.isServer){
  Meteor.startup(function()){
    // this needs to be run on the server
    var environment, settings;

    environment = process.env.METEOR_ENV || "development";

    settings = {
      development: {
        public: {
          package: {
            name: "jquery-datatables",
            description: "Sort, page, and filter millions of records. Reactively.",
            owner: "LumaPictures",
            repo: "meteor-jquery-datatables"
          }
        },
        private: {}
      },
      staging: {
        public: {},
        private: {}
      },
      production: {
        public: {},
        private: {}
      }
    };

    if (!process.env.METEOR_SETTINGS) {
      console.log("No METEOR_SETTINGS passed in, using locally defined settings.");
      if (environment === "production") {
        Meteor.settings = settings.production;
      } else if (environment === "staging") {
        Meteor.settings = settings.staging;
      } else {
        Meteor.settings = settings.development;
      }
      console.log("Using [ " + environment + " ] Meteor.settings");
    }
  });
}
```

App-Parameter mit METEOR_SETTINGS angeben

Die Umgebungsvariable METEOR_SETTINGS kann JSON-Objekte akzeptieren und macht dieses Objekt im `Meteor.settings` Objekt `Meteor.settings` . `settings.json` in Ihrem App-Stammverzeichnis eine `settings.json` mit Konfigurationsinformationen hinzu.

```
{
  "public":{
    "ga":{
      "account":"UA-XXXXXXX-1"
    }
  }
}
```

Dann müssen Sie Ihre Anwendung mit Ihrer Einstellungsdatei starten.

```
# run your app in local development mode with a settings file
meteor --settings settings.json

# or bundle and prepare it as if you're running in production
# and specify a settings file
meteor bundle --directory /path/to/output
cd /path/to/output
MONGO_URL="mongodb://127.0.0.1:27017" PORT=3000 METEOR_SETTINGS=$(cat /path/to/settings.json)
node main.js
```

Auf diese Einstellungen kann dann über Meteor.settings zugegriffen und in Ihrer App verwendet werden.

```
Meteor.startup(function(){
  if(Meteor.isClient){
    console.log('Google Analytics Account', Meteor.settings.public.ga.account);
  }
});
```

Umgebungserkennung auf dem Server

Umgebungsvariablen stehen dem Server auch über das Objekt `process.env` .

```
if (Meteor.isServer) {
  Meteor.startup(function () {
    // detect environment by getting the root url of the application
    console.log(JSON.stringify(process.env.ROOT_URL));

    // or by getting the port
    console.log(JSON.stringify(process.env.PORT));

    // alternatively, we can inspect the entire process environment
    console.log(JSON.stringify(process.env));
  });
}
```

Erkennung der Clientumgebung mit Meteor-Methoden

Um die Umgebung auf dem Server zu ermitteln, müssen Sie eine Hilfemethode auf dem Server

erstellen, da der Server die Umgebung ermittelt, in der er sich befindet, und dann die Hilfemethode vom Client aus aufrufen. Grundsätzlich geben wir nur die Umgebungsinformationen vom Server an den Client weiter.

```
//-------------------------------------------------------------------------------------------
----------
// server/server.js
// we set up a getEnvironment method

Meteor.methods({
  getEnvironment: function(){
    if(process.env.ROOT_URL == "http://localhost:3000"){
        return "development";
    }else{
        return "staging";
    }
  }
 });

//-------------------------------------------------------------------------------------------
----------
// client/main.js
// and then call it from the client

Meteor.call("getEnvironment", function (result) {
  console.log("Your application is running in the " + result + "environment.");
});
```

Erkennung der Clientumgebung mit NODE_ENV

Seit Meteor 1.3 macht Meteor nun `NODE_ENV` Variable `NODE_ENV` auf dem Client `NODE_ENV` .

```
if (Meteor.isClient) {
  Meteor.startup(function () {
    if(process.env.NODE_ENV === "testing"){
      console.log("In testing...");
    }
    if(process.env.NODE_ENV === "production"){
      console.log("In production...");
    }
  });
}
```

Umgebungserkennung online lesen:
https://riptutorial.com/de/meteor/topic/4198/umgebungserkennung

Kapitel 41: Umgebungsvariablen

Parameter

Parameter	Einzelheiten
HAFEN	Port, auf dem die Meteor-App verfügbar sein wird.
MONGO_URL	URL für die Verbindung zur Mongo-Instanz.
ROOT_URL	...
OPLOG_URL	...
MONGO_OPLOG_URL	...
METEOR_ENV	...
NODE_ENV	...
NODE_OPTIONS	...
DISABLE_WEBSOCKETS	...
MAIL_URL	...
DDP_DEFAULT_CONNECTION_URL	...
HTTP-PROXY	...
HTTPS_PROXY	...
METEOR_OFFLINE_CATALOG	...
METEOR_PROFILE	...
METEOR_DEBUG_BUILD	...
TINYTEST_FILTER	...
MOBILE_ROOT_URL	...
NODE_DEBUG	...
BIND_IP	...
PACKAGE_DIRS	...

Parameter	Einzelheiten
DEBUGGEN	...
METEOR_PRINT_CONSTRAINT_SOLVER_INPUT	...
METEOR_CATALOG_COMPRESS_RPCS	...
METEOR_MINIFY_LEGACY	...
METEOR_DEBUG_SQL	...
METEOR_WAREHOUSE_DIR	...
AUTOUPDATE_VERSION	...
USE_GLOBAL_ADK	...
METEOR_AVD	...
DEFAULT_AVD_NAME	...
METEOR_BUILD_FARM_URL	...
METEOR_PACKAGE_SERVER_URL	...
METEOR_PACKAGE_STATS_SERVER_URL	...
DEPLOY_HOSTNAME	...
METEOR_SESSION_FILE	...
METEOR_PROGRESS_DEBUG	...
METEOR_PRETTY_OUTPUT	...
APP_ID	...
AUTOUPDATE_VERSION	...
CONSTRAINT_SOLVER_BENCHMARK	...
DDP_DEFAULT_CONNECTION_URL	...
SERVER_WEBSOCKET_COMPRESSION	...
USE_JSESSIONID	...
METEOR_PKG_SPIDERABLE_PHANTOMJS_ARGS	...
WRITE_RUNNER_JS	...

Parameter	Einzelheiten
TINYTEST_FILTER	...
METEOR_PARENT_PID	...
METEOR_TOOL_PATH	...
RUN_ONCE_OUTCOME	...
TREE_HASH_DEBUG	...
METEOR_DEBUG_SPRINGBOARD	...
METEOR_TEST_FAIL_RELEASE_DOWNLOAD	...
METEOR_CATALOG_COMPRESS_RPCS	...
METEOR_TEST_LATEST_RELEASE	...
METEOR_WATCH_POLLING_INTERVAL_MS	...
EMACS	...
METEOR_PACKAGE_STATS_TEST_OUTPUT	...
METEOR_TEST_TMP	...

Examples

Verwenden von Umgebungsvariablen mit Meteor

Umgebungsvariablen können vor dem Meteoraufruf wie folgt definiert werden:

```
PORT=4000 meteor
NODE_ENV="staging" meteor
```

Meteor SMTP-Server einstellen

Google Mail-Beispiel

```
MAIL_URL=smtp://username%40gmail.com:password@smtp.gmail.com:465/
```

Hinweis: Bei diesem Setup können nur 2000 E-Mails pro Tag gesendet werden. Unter https://support.google.com/a/answer/176600?hl=de finden Sie alternative Konfigurationen.

Umgebungsvariablen online lesen:
https://riptutorial.com/de/meteor/topic/3154/umgebungsvariablen

Kapitel 42: Umschließen asynchroner Methoden in eine Fiber für die synchrone Ausführung.

Syntax

1. Meteor.wrapAsync (func, [context])

Parameter

Parameter	Einzelheiten
Funktion: Funktion	Eine asynchrone / synchrone Funktion, die in eine Glasfaser eingebettet werden soll und einen Rückruf mit Parametern `(error, result)` erfordert.
Kontext: beliebig (optional)	Ein Datenkontext, in dem die Funktion ausgeführt wird.

Bemerkungen

Eine asynchron umschlossene Funktion kann immer noch asynchron ausgeführt werden, wenn ein Rückruf mit Parametern `(error, result) => {}` als Parameter für die umschlossene Funktion angegeben wird.

Die Einbindung von `Meteor.wrapAsync` ermöglicht die Vereinfachung von Code mit Callbacks, da Callbacks jetzt als Kompensation für die Sperrung des `Meteor.wrapAsync` seiner aktuellen `Fiber` vernachlässigt werden können.

Um zu verstehen, wie Fasern funktionieren, lesen Sie hier: https://www.npmjs.com/package/fibers .

Examples

Asynchronous NPM-Methoden synchron mit Callbacks ausführen.

In diesem Beispiel wird die asynchrone Methode `oauth2.client.getToken(callback)` aus dem Paket NPM-Paket `simple-oauth2` in eine Glasfaser eingeschlossen, sodass die Methode synchron aufgerufen werden kann.

```
const oauth2 = require('simple-oauth2')(credentials);

const credentials = {
```

```
    clientID: '#####',
    clientSecret: '#####',
    site: "API Endpoint Here."
};

Meteor.startup(() => {
    let token = Meteor.wrapAsync(oauth2.client.getToken)({});
    if (token) {
        let headers = {
            'Content-Type': "application/json",
            'Authorization': `Bearer ${token.access_token}`
        }

        // Make use of requested OAuth2 Token Here (Meteor HTTP.get).
    }
});
```

Umschließen asynchroner Methoden in eine Fiber für die synchrone Ausführung. online lesen: https://riptutorial.com/de/meteor/topic/2530/umschlie-en-asynchroner-methoden-in-eine-fiber-fur-die-synchrone-ausfuhrung-

Kapitel 43: Vermögenswerte

Examples

Zugriff auf Assets auf dem Server

Statische Server-Assets müssen im `private` Verzeichnis abgelegt werden.

Textdateien

Auf Textdateien kann mit der **`Assets.getText(assetPath, [asyncCallback])`** werden. Die folgende JSON-Datei hat beispielsweise den Namen `my_text_asset.json` und befindet sich im `private` Verzeichnis:

```
{
    "title": "Meteor Assets",
    "type": "object",
    "users": [{
        "firstName": "John",
        "lastName": "Doe"
    }, {
        "firstName": "Jane",
        "lastName": "Doe"
    }, {
        "firstName": "Matthias",
        "lastName": "Eckhart"
    }]
}
```

Sie können auf diese Datei auf dem Server mit dem folgenden Code zugreifen:

```
var myTextAsset = Assets.getText('my_text_asset.json');
var myJSON = JSON.parse(myTextAsset);
console.log(myJSON.title); // prints 'Meteor Assets' in the server's console
```

Binärdateien

Wenn Sie als EJSON-Binärdatei auf Assets auf dem Server zugreifen möchten, verwenden Sie die **`Assets.getBinary(assetPath, [asyncCallback])`** . Hier ein Codebeispiel für den Zugriff auf ein Image namens `my_image.png` das sich im `private/img` Verzeichnis befindet:

```
var myBinaryAsset = Assets.getBinary('img/my_image.png');
```

Vermögenswerte online lesen: https://riptutorial.com/de/meteor/topic/3379/vermogenswerte

Kapitel 44: Veröffentlichen eines Release-Tracks

Bemerkungen

Das Veröffentlichen eines Release-Tracks ist ziemlich einfach, wenn Sie wissen, a) dass der Befehl zum Publizieren und Freigeben eine .json-Datei als Parameter erfordert und b) wie diese Datei aussieht. Das ist definitiv die größte Hürde beim Einstieg, weil es so ziemlich nirgends dokumentiert ist.

Denken Sie daran, dass jedes Paket in der Veröffentlichung veröffentlicht werden muss und auf Atmosphere. Die .meteor / versions-Datei einer App ist ein besonders guter Ort, um alle erforderlichen Pakete und Versionen zu finden, die in das Release aufgenommen werden sollen.

Danach geht es darum, herauszufinden, was Sie unterstützen möchten, was Sie mit einbeziehen möchten usw. Hier ist ein partielles Venn-Diagramm, woran die klinische Veröffentlichung derzeit arbeitet; und sollte Ihnen eine allgemeine Vorstellung davon vermitteln, wie wir über den Entscheidungsfindungsprozess der Inhalte vorgehen.

Weitere Informationen finden Sie im Thema in den Meteor-Foren:
https://forums.meteor.com/t/custom-meteor-release/13736/6

Examples

Grundlegende Verwendung

Die Idee ist, dass ein Distributor der Distribution so etwas wie den folgenden Befehl ausführen möchte:

```
meteor publish-release clinical.meteor.rc6.json
```

Dadurch können Benutzer der Distribution Folgendes ausführen:

```
meteor run --release clinical:METEOR@1.1.3-rc6
```

Manifest freigeben

Ein `package.json` ähnelt einer NPM-Datei `package.json` , da es in erster `package.json` , eine Liste von Atmosphere-Paketen anzugeben und einige Metadaten zu dieser `package.json` . Das Basisformat sieht folgendermaßen aus:

```
{
  "track":"distroname:METEOR",
  "version":"x.y.z",
```

```
  "recommended": false,
  "tool": "distroname:meteor-tool@x.y.z",
  "description": "Description of the Distro",
  "packages": {
    "accounts-base":"1.2.0",
    "accounts-password":"1.1.1",
    ...
  }
}
```

Anpassen des Meteor-Tools

Wenn Sie das Meteor-Tool oder die Befehlszeile erweitern müssen, müssen Sie ein eigenes Meteor-Tool-Paket erstellen und veröffentlichen. Ronens Dokumentation ist die beste für diesen Prozess:

http://practicalmeteor.com/using-meteor-publish-release-to-extend-the-meteor-command-line-tool/1

Es ist leicht, einen Meteor helloworld-Befehl zum Laufen zu bringen, aber danach fand ich es einfacher, eine separate Knoten-App zu erstellen, um Befehle auszuprobieren. So entstand StarryNight. Es ist so etwas wie ein Staging Ground und ein Scratchpad für Befehle, bevor versucht wird, sie in eine Version des Meteor-Tools zu integrieren.

Auslösen eines Veröffentlichungsmanifests aus .meteor / versions

StarryNight enthält ein kleines Dienstprogramm, das die `.meteor/versions` Datei einer Anwendung `.meteor/versions` und in ein Veröffentlichungsmanifest konvertiert.

```
npm install -g starrynight
cd myapp
starrynight generate-release-json
```

Wenn Sie StarryNight nicht nutzen wollen, einfach den Inhalt Ihrer kopieren `.meteor/versions` Datei in den `packages` Feld Ihrer Manifest - Datei. Stellen Sie sicher, dass Sie in die JSON-Syntax konvertieren und Doppelpunkte und Anführungszeichen hinzufügen.

Anzeigen des Release-Manifests für eine bestimmte Version

```
meteor show --ejson METEOR@1.2.1
```

Veröffentlichen einer Veröffentlichung aus der Kasse

```
meteor publish-release --from-checkout
```

Abrufen der neuesten Commits für jedes Paket in einem Release

Beim Erstellen einer benutzerdefinierten Release-Spur werden Pakete im Verzeichnis `/packages`

normalerweise als git-Submodule gespeichert. Mit dem folgenden Befehl können Sie alle neuesten Commits für die Submodule in Ihrem `/packages` Verzeichnis gleichzeitig abrufen.

```
git submodule foreach git pull origin master
```

Veröffentlichen eines Release-Tracks online lesen:
https://riptutorial.com/de/meteor/topic/4201/veroffentlichen-eines-release-tracks

Kapitel 45: Verwenden Sie private Meteor-Pakete für das Codeship

Bemerkungen

Beachten Sie, dass wir nicht besprochen haben, wie Sie Ihre lokalen Pakete verwenden und entwickeln. Es gibt mehrere Möglichkeiten, ich schlage vor, die von David Weldon auf seiner Website beschriebene Umgebungsvariable `PACKAGE_DIRS` zu verwenden.

Examples

Installieren Sie MGP

Wir nutzen das Paket " Meteor Github Packages (mgp)" von "Dispatches":

```
npm install --save mgp
```

`package.json` Sie dann den folgenden Befehl zu Ihren `package.json` Skripts hinzu:

```
"mgp": "mgp"
```

Erstellen Sie eine Datei mit dem Namen `git-packages.json` in Ihrem Projektstammverzeichnis. Fügen Sie eine Konfiguration für jedes (private) Meteor Github-Paket hinzu, von dem Ihr Projekt abhängig ist:

```
{
  "my:yet-another-private-package": {
    "git": "git@github.com:my/private-packages.git",
    "branch": "dev"
  }
}
```

Weitere Informationen zum Konfigurieren Ihrer privaten Pakete finden Sie im Projekt Github Repo.

Konfigurieren Sie Codeship für die Installation von privaten Github-Paketen

Fügen Sie den Codeship-Setupbefehlen den folgenden Befehl hinzu:

```
meteor npm run mgp
```

Nun müssen wir Codeship Zugriff auf diese privaten Repositories gewähren. Es gibt einen Codeship-Dokumentationsartikel , der diesen Prozess ausführlich beschreibt, aber hier sind die Schritte, die Sie für Github durchführen müssen:

- Erstellen Sie ein neues Github-Konto. Ein sogenannter Machine-Benutzer .

- Entfernen Sie den Bereitstellungsschlüssel aus dem zu testenden Repo. Hier: https://github.com/YOUR_USERNAME/REPO_UNDER_TEST/settings/keys
- Holen Sie sich den öffentlichen SSH-Schlüssel aus den Einstellungen Ihrer Codeship-Projekte. Irgendwo hier: https://codeship.com/projects/PROJECT_NUMBER/configure
- Fügen Sie diesen öffentlichen SSH-Schlüssel den SSH-Schlüsseln Ihres Computerbenutzers hinzu: https://github.com/settings/keys
- Geben Sie diesem Computerbenutzer Zugriff auf alle referenzierten Repositorys

Es sollte für BitBucket und andere ähnlich sein.

Verwenden Sie private Meteor-Pakete für das Codeship online lesen: https://riptutorial.com/de/meteor/topic/6742/verwenden-sie-private-meteor-pakete-fur-das-codeship

Kapitel 46: Verzeichnisaufbau

Einführung

Vor der Veröffentlichung von Meteor 1.3 waren Meteor-Entwickler mit der Handhabung von Dateiabhängigkeiten und globalen Variablen durch Meteor.js frustriert. Als Reaktion darauf setzte Meteor neue Maßstäbe für Projektstrukturen, um das Projektabhängigkeitssystem schlanker zu gestalten. Dieses Thema erläutert die standardisierte Projektstruktur und die dahinter stehenden Prinzipien.

Bemerkungen

Klient
Der gesamte Code im Clientverzeichnis wird nur auf der Clientseite oder im Webbrowser ausgeführt.

Client / Kompatibilität
Das Kompatibilitätsverzeichnis enthält Legacy- oder Drittanbietercode, wie z. B. jQuery-Bibliotheken usw.

lib
Das lib-Verzeichnis wird vor anderen Verzeichnissen in Ihrem Meteor-Projekt geladen und sowohl auf dem Server als auch auf dem Client geladen. Dies ist der bevorzugte Ort, um Datenmodelle, isomorphe Bibliotheken und Geschäftslogik zu definieren.

Einfuhren
Das Importverzeichnis ist ein Verzeichnis auf dem Server, das sowohl für den Server als auch für den Client verfügbar ist, jedoch nur, bevor das Client-Paket an den Client gesendet wird.

Pakete
Im Paketverzeichnis werden während der lokalen Entwicklung benutzerdefinierte Pakete gespeichert. Wenn Sie den Standardbefehl `meteor add package:name` , um ein Paket hinzuzufügen, sucht Meteor zuerst in diesem Verzeichnis, ob ein lokales Paket in seiner `package.js` Datei den entsprechenden Beschreibungsnamen hat. Wenn nicht, wird Atmosphere wie üblich abgefragt.

Privatgelände
Das private Verzeichnis enthält statische Dateien, die nur auf dem Webserver verfügbar sein sollten.

Öffentlichkeit
Das öffentliche Verzeichnis enthält statische Dateien, die nur auf dem Anwendungsclient verfügbar sind. Dies kann Branding-Assets usw. umfassen.

Server
Das Serververzeichnis enthält serverseitige Assets. Dazu können Authentifizierungslogik, Methoden und anderer Code gehören, der möglicherweise Sicherheitsaspekte berücksichtigt.

Tests
Das Testverzeichnis wird standardmäßig ausgelassen, wenn Ihre Anwendung gebündelt und bereitgestellt wird.

Wie von Richard Silverton vorgeschlagen , ist es eine bequeme Idee, nicht nur das Meteor-Projektverzeichnis unter Versionskontrolle zu stellen, sondern auch das übergeordnete Verzeichnis.

Auf diese Weise können Sie Dateien unter Versionskontrolle halten, ohne sich mit einem Meteor auseinandersetzen zu müssen.

Examples

Klassische Verzeichnisstrukturen

Das Erste, was Sie beim Strukturieren Ihrer Apps wissen müssen, ist, dass das Meteor-Tool einige Verzeichnisse enthält, die mit einer bestimmten Logik hartcodiert sind. Grundsätzlich werden die folgenden Verzeichnisse im Meteor-Bundler "gebacken".

```
client/                                 # client application code
client/compatibility/                   # legacy 3rd party javascript libraries
imports/                                # for lazy loading feature
lib/                                    # any common code for client/server.
packages/                               # place for all your atmosphere packages
private/                                # static files that only the server knows about
public/                                 # static files that are available to the client
server/                                 # server code
tests/                                  # unit test files (won't be loaded on client or
server)
```

Referenzseite: Meteor Guide> Spezielle Verzeichnisse

Verzeichnisstruktur nur für Pakete

Viele Menschen unterstützen schließlich mehrere Anwendungen und möchten Code zwischen Apps austauschen. Dies führt zum Konzept der Microservice-Architektur und zu Apps für alle Pakete. Im Wesentlichen wird der Code aus der gesamten klassischen Verzeichnisstruktur in Pakete umgestaltet.

Obwohl es keine fest codierte Logik für Verzeichnisse in Paketen gibt, ist es ratsam, beim Erstellen von Paketen die klassische Verzeichnisstruktur zu verwenden. Dadurch wird ein natürlicher Refaktorpfad erstellt, da Features in der App prototypisiert und dann in Pakete extrahiert werden, die veröffentlicht und gemeinsam genutzt werden können. Die Verzeichnisnamen werden gemeinsam genutzt, sodass die Teammitglieder weniger verwirrt sind.

```
client/                                 # client application code
packages/                               # place for all your atmosphere packages
packages/foo/client                     # client application code
packages/foo/lib                        # any common code for client/server
packages/foo/server                     # server code
```

```
packages/foo/tests                      # tests
server/                                 # server code
```

Verzeichnisstruktur der Imports / Module

Die neuesten Versionen von Meteor werden mit Unterstützung für `ecmascript` (ES6 oder ES2015) ausgeliefert. Anstelle von Paketen unterstützt Javascript jetzt `import` und -module, was die Notwendigkeit von Paketanwendungen ersetzt. Die neueste Verzeichnisstruktur ähnelt der Paketstruktur, verwendet jedoch das Verzeichnis `/imports` anstelle von `/packages` .

```
imports                                 #
imports/api                             # isomorphic methods
imports/lib                             # any common code for client/server
imports/client                          # client application code
imports/server                          # server code
```

Verzeichnisstruktur im gemischten Modus

Natürlich können Sie diese Ansätze kombinieren und Pakete sowie Importe neben Ihrem anwendungsspezifischen Code verwenden. Eine Mix-Modus-Struktur kommt am häufigsten in drei Situationen vor: Eine Franken-App, die einfach ohne eine übergeordnete Strategie ein bisschen von hier und da abzieht; Eine App, die von Classic- oder Package-Only-Strukturen aktiv in die Imports / Modules-Struktur überarbeitet wird.

```
client/                                 # client application code
client/compatibility/                   # legacy 3rd party javascript libraries
imports                                 #
imports/api                             # isomorphic methods
imports/lib                             # any common code for client/server
imports/client                          # client application code
imports/server                          # server code
lib/                                    # any common code for client/server.
packages/                               # place for all your atmosphere packages
packages/foo/client                     # client application code
packages/foo/lib                        # any common code for client/server
packages/foo/server                     # server code
packages/foo/tests                      # tests
private/                                # static files that only the server knows about
public/                                 # static files that are available to the client
server/                                 # server code
tests/                                  # unit test files (won't be loaded on client or
server)
```

Reihenfolge beim Laden des Verzeichnisses

HTML-Vorlagendateien werden immer vor allem anderen geladen

Dateien, die mit *main beginnen.* werden zuletzt geladen

Als nächstes werden Dateien in einem beliebigen lib / -Verzeichnis geladen

Als nächstes werden Dateien mit tieferen Pfaden geladen

Die Dateien werden dann in alphabetischer Reihenfolge des gesamten Pfads geladen

Referenzlink

Referenzseite: Meteor-Handbuch> Anwendungsstruktur> Standardreihenfolge beim Laden von Dateien

Verzeichnisaufbau online lesen: https://riptutorial.com/de/meteor/topic/3072/verzeichnisaufbau

Kapitel 47: Vollständige Installation - Mac OSX

Examples

Installieren Sie Node & NPM

Dieser Schnellstart wurde für Mac OSX Mavericks geschrieben und ist etwas ausführlicher als andere Installationsanweisungen. Es sollte hoffentlich einige Randfälle abdecken, z. B. das Festlegen Ihres Pfads und das Konfigurieren von NPM, was dazu führen kann, dass eine Installation schief geht.

```
    # install node
    # as of OSX Mavericks, we need the GUI installer (?!)
    # when a good command line alternative is found, we'll post it
    http://nodejs.org/download/

    # install npm
    curl -0 -L https://npmjs.org/install.sh | sh

    # check node is installed correctly
    node --version

    # check npm is installed correctly
    npm -version

    # find your npm path
    which npm

    # make sure npm is in your path
    sudo nano ~/.profile
      export PATH=$PATH:/usr/local/bin
```

Komplettlösung zur Installation von Meteor

Dieser Schnellstart wurde für Mac OSX Mavericks geschrieben und ist etwas ausführlicher als andere Installationsanweisungen. Es sollte hoffentlich einige Randfälle abdecken, z. B. das Festlegen Ihres Pfads und das Konfigurieren von NPM, was dazu führen kann, dass eine Installation schief geht.

```
# install meteor
    curl https://install.meteor.com | sh

    # check it's installed correctly
    meteor --version

    # install node
    # as of OSX Mavericks, we need the GUI installer (?!)
    # when a good command line alternative is found, we'll post it
    http://nodejs.org/download/
```

```
    # install npm
    curl -0 -L https://npmjs.org/install.sh | sh

    # check node is installed correctly
    node --version

    # check npm is installed correctly
    npm -version

    # find your npm path
    which npm

    # make sure npm is in your path
    sudo nano ~/.profile
      export PATH=$PATH:/usr/local/bin
```

Mongo-Installation

Meteor existiert nicht isoliert, und es ist üblich, eine Reihe zusätzlicher Tools für die Entwicklung zu installieren, beispielsweise Mongo, Robomongo, Atom, Linters usw.

```
# make sure mongo is in your local path
nano ~/.profile
  export PATH=$PATH:/usr/local/mongodb/bin

# or install it to the global path
nano /etc/paths
  /usr/local/mongodb/bin

# create mongo database directory
mkdir /data/
mkdir /data/db
chown -R username:admin /data

# run mongodb server
mongod
ctrl-c

# check that you can connect to your meteor app with stand-alone mongo
terminal-a$ meteor create helloworld
terminal-a$ cd helloworld
terminal-a$ meteor

terminal-b$ mongo -port 3001

# install robomongo database admin tool
http://robomongo.org/

# check you can connect to your mongo instance with robomongo
terminal-a$ meteor create helloworld
terminal-a$ cd helloworld
terminal-a$ meteor

Dock$ Robomongo > Create > localhost:3001
```

Andere Entwicklungswerkzeuge

```
# install node-inspector
terminal-a$  npm install -g node-inspector

# start meteor
terminal-a$  cd helloworld
terminal-a$  NODE_OPTIONS='--debug-brk --debug' mrt run

# alternatively, some people report this syntax being better
terminal-a$  sudo NODE_OPTIONS='--debug' ROOT_URL=http://helloworld.com meteor --port 80

# launch node-inspector along side your running app
terminal-b$  node-inspector

# go to the URL given by node-inspector and check it's running
http://localhost:8080/debug?port=5858

# install jshint
npm install -g jshint
```

Vollständige Installation - Mac OSX online lesen:
https://riptutorial.com/de/meteor/topic/3294/vollstandige-installation---mac-osx

Kapitel 48: Zugriff auf Meteor Build-Maschinen von Windows aus

Bemerkungen

Das `meteor` Befehlszeilentool setzt unter Mac und Linux voraus, dass das `ssh` Befehlszeilentool, mit dem sichere Verbindungen zu anderen Computern hergestellt werden, immer vorhanden ist. Unter Windows muss dieses Tool installiert werden. Nachfolgend sind zwei Optionen zum Einrichten und Verwenden aufgeführt.

Examples

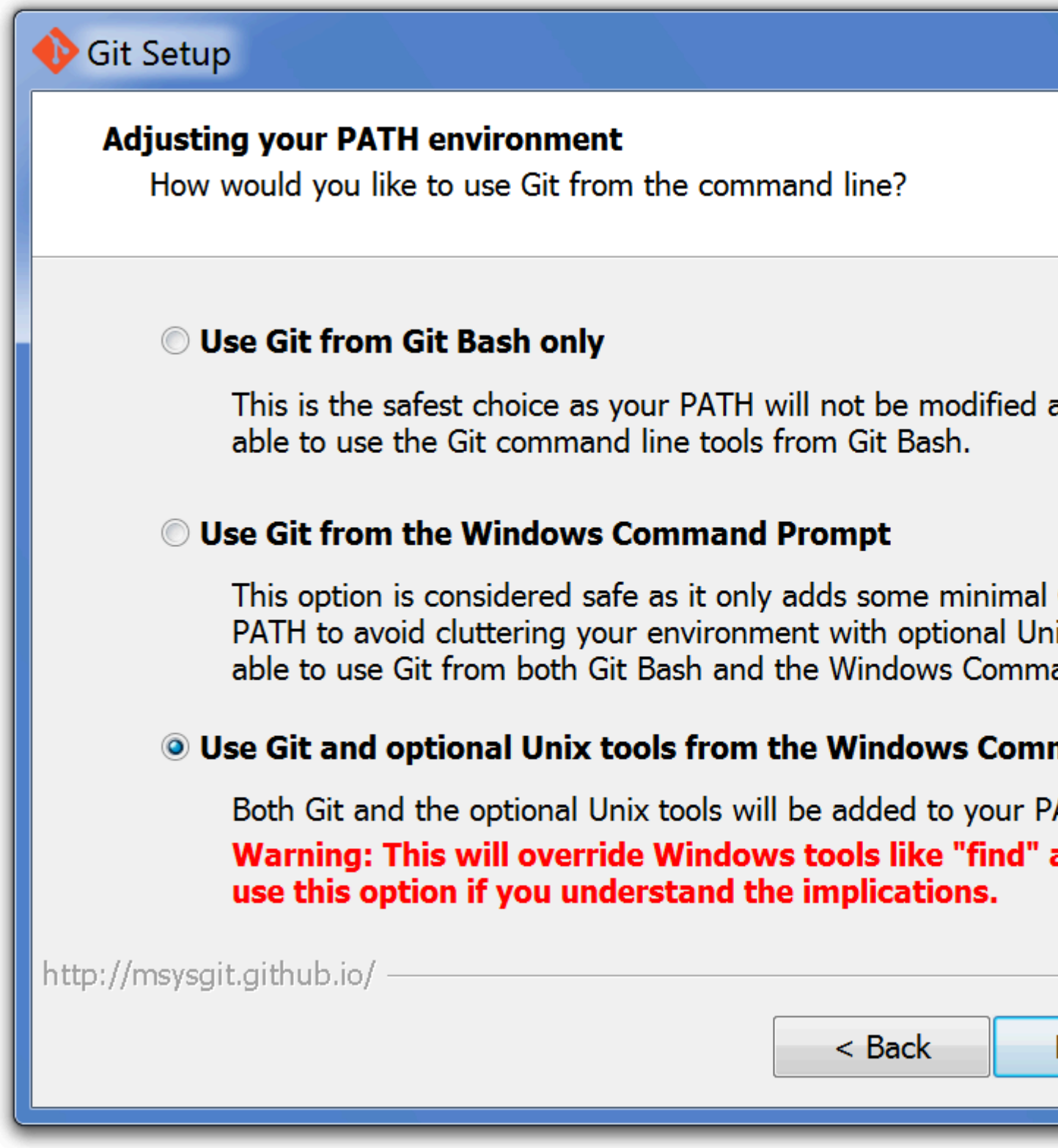
PuTTY verwenden (Fortgeschrittene)

Wenn Sie Ihrem PATH unter Windows keine Unix-Befehle hinzufügen möchten, können Sie einen eigenständigen SSH-Client wie PuTTY herunterladen. Laden Sie PuTTY hier herunter und folgen Sie den Anweisungen unten, um eine Build-Maschine zu erhalten.

1. Rufen Sie den `meteor admin get-machine <os-architecture> --json`
2. Kopieren Sie den privaten Schlüssel und speichern Sie ihn aus den zurückgegebenen JSON-Daten
3. Folgen Sie den Anweisungen hier , um den privaten Schlüssel in ein Format zu konvertieren, das von PuTTY akzeptiert wird
4. Geben Sie den Hostnamen, den Benutzernamen und den privaten Schlüssel in PuTTY ein, und schon kann es losgehen!

Cygwin verwenden (Unix-Tools unter Windows)

Der einfachste Weg zum Einstieg ist die Installation von Git für Windows von dieser Download-Seite . Wählen Sie "Git und optionale Unix-Tools von der Windows-Eingabeaufforderung verwenden" wie in der Abbildung unten aus.



Danach funktioniert `meteor admin get-machine <os-architecture>` genauso wie auf Linux und Mac. Beachten Sie, dass Sie möglicherweise Ihr Terminal neu starten müssen, um die neuen Befehle zu erhalten.

Zugriff auf Meteor Build-Maschinen von Windows aus online lesen:
https://riptutorial.com/de/meteor/topic/518/zugriff-auf-meteor-build-maschinen-von-windows-aus

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Meteor	Ankit, Christian Fritz, Community, Gal Dreiman, ghybs, grahan, hwillson, João Rodrigues, levon, Matthias Eckhart, mav, mertyildiran, Ray, reoh, robfallows, Tom Coleman, Zoltan Olah
2	Abnahmeprüfung (mit Nightwatch)	AbigailW
3	Anfängerleitfaden zur Installation von Meteor 1.4 unter AWS EC2	AGdev
4	Bereitstellung mit Upstart	AbigailW, ghybs
5	Blaze Templating	Dan Cramer, distalx, ghybs, jordanwillis, khem poudel, RamenChef, robfallows, Thomas Gerot
6	Blaze-Benutzeroberflächenrezepte (Bootstrap; keine jQuery)	AbigailW, Anis D
7	Continuous Integration & Device Clouds (mit Nightwatch)	4444, AbigailW
8	Datei hochladen	AbigailW
9	Daten veröffentlichen	Abdelrahman Elkady, AbigailW, Chris Pena, corvid, Dair, dangsonbk, Eliezer Steinbock, Faysal Ahmed, ghybs, j6m8, Maciek, RamenChef, Ramil Muratov, robfallows, Serkan Durusoy
10	Daten von einem Meteor.call abrufen	Ramil Muratov, Rolljee, Sacha
11	Debuggen	AbigailW, distalx
12	Electrify - Meteor als lokal installierbare App kompilieren	AbigailW, JuanGesino, Nick Bull, RamenChef
13	Entwicklungswerkzeuge	AbigailW, Ankit, Ankit Balyan, Fermuch, Ilya Lyamkin

14	ES2015-Module (Import und Export)	reoh
15	ESLint	saimeunt
16	Grundlegendes Codeship-Setup für automatisiertes Testen	schmidsi
17	Hintergrundaufgaben	Filipe Névola
18	Horizontale Skalierung	AbigailW
19	Integration von Drittanbieter-APIs	AbigailW
20	Knoten / NPM	hwillson
21	Kontinuierliche Bereitstellung von Codeship für Galaxy	schmidsi
22	Leistungsoptimierung	AbigailW, RamenChef, reoh
23	Meteor + React + ReactRouter	rafahoro
24	Meteor + Reaktion	AbigailW, aedm, corvid, ghybs, RamenChef, Teagan Atwater, zliw
25	Meteor Benutzerkonten	Barry Michael Doyle, KrisVos130
26	Meteor mit einem Proxy-Server verwenden	AbigailW, Serkan Durusoy, Tom Coleman
27	Mobile Apps	AbigailW, Anis D, Antti Haapala, ghybs
28	Mongo-Datenbankverwaltung	AbigailW, distalx, RamenChef, TechplexEngineer
29	MongoDB	distalx, Dranithix, hwillson, Matthias Eckhart, robfallows, Thomas Gerot
30	MongoDB-Aggregation	AbigailW, levon
31	Mongo-Sammlungen	AbigailW
32	Mongo-Schema-Migrationen	AbigailW
33	Nightwatch - Konfiguration	AbigailW

	und Setup	
34	Offline-Apps	AbigailW
35	Polymer mit Meteor verwenden	Thaum Rystra
36	Protokollierung	AbigailW
37	Reaktiv (Vars & Wörterbücher)	Ankit
38	Replikatsätze und Sharding	AbigailW, Anis D
39	Routing	Ankit, ghybs, Luna, Michael Balmes
40	Umgebungserkennung	AbigailW, ghybs
41	Umgebungsvariablen	AbigailW, hcvst
42	Umschließen asynchroner Methoden in eine Fiber für die synchrone Ausführung.	Dranithix
43	Vermögenswerte	Matthias Eckhart
44	Veröffentlichen eines Release-Tracks	AbigailW
45	Verwenden Sie private Meteor-Pakete für das Codeship	schmidsi
46	Verzeichnisaufbau	AbigailW, anomepani, ghybs, Michael Balmes, Nick Carson, Phe0nix, reoh, Thomas Gerot
47	Vollständige Installation - Mac OSX	AbigailW, RamenChef
48	Zugriff auf Meteor Build-Maschinen von Windows aus	Tom Coleman