eBook Gratuit

APPRENEZ sails.js

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#sails.js

Table des matières

À propos

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

It is an unofficial and free sails.js 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 sails.js.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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

Chapitre 1: Démarrer avec sails.js

Remarques

sails.js est un framework web MVC (Model View Controller) pour node.js qui émule des frameworks MVC familiers comme Ruby on Rails. sails.js est basé sur Express et fournit un support websocket via socket.io.

sails.js fournit un ensemble de conventions et de configurations par défaut pour lancer rapidement un nouveau projet de site Web. Il est hautement configurable et vous permet de remplacer facilement les conventions par défaut.

sails.js est livré avec un ORM appelé Waterline qui résume l'accès aux données. Waterline vous permet d'utiliser différents magasins de données tels que MySQL, PostgreSQL, MongoDB, Redis, etc. et d'avoir une API claire pour travailler avec les données de votre modèle.

Versions

Version	Notes de version	Changelog	Date de sortie
0.12.13	Notes de version		2017-03-06
0,12.12	Notes de version	Changelog	2017-03-03
0.12.11	Notes de version	Changelog	2016-11-24
0.12.10	Notes de version	Changelog	2016-11-17
0.12.9	Notes de version	Changelog	2016-11-02
0.12.8	Notes de version	Changelog	2016-10-22
0,12,7	Notes de version	Changelog	2016-10-06
0.12.6	Notes de version	Changelog	2016-09-28
0,12,5	Notes de version	Changelog	2016-09-28
0,12,4	Notes de version	Changelog	2016-08-01
0.12.3	Notes de version	Changelog	2016-04-04
0.12.2	Notes de version	Changelog	2016-04-02
0.12.1	Notes de version	Changelog	2016-02-15
0.12.0	Notes de version	Changelog	2016-02-06

Version	Notes de version	Changelog	Date de sortie
0.11.5	Notes de version	Changelog	2016-02-05
0,11,4	Notes de version	Changelog	2016-01-06
0.11.3	Notes de version	Changelog	2015-11-23
0.11.2	Notes de version	Changelog	2015-09-23
0.11.0	Notes de version	Changelog	2015-02-11
0,10,5	Notes de version	Changelog	2014-08-30
0,10,4	Notes de version		2014-08-13
0.10.3	Notes de version		2014-08-07
0,10,2	Notes de version		2014-08-06
0,10,1	Notes de version		2014-08-02

Les versions antérieures à `0.10.1` omises de la liste. Voir les versions précédentes

Examples

Installation

Conditions préalables

- nodejs

Pour installer la dernière version stable de voiles avec le problème de l'outil de ligne de commande suivant la commande:

```
$ sudo npm install sails -g
```

Selon votre système d'exploitation, vous n'avez peut-être pas besoin d'utiliser `sudo` .

Créer un nouveau projet

Une fois que vous avez installé Sails, tapez simplement

```
$ sails new <project_name>
```

Cela créera un projet squelette Sails dans un nouveau dossier appelé <nom_projet>.

Vous pouvez également créer un nouveau projet dans un dossier vide en tapant

```
$ sails new
```

Lancer l'application

Une fois votre projet créé, vous pouvez lancer l'application en tapant

```
$ sails lift
```

Par défaut, vous pouvez accéder à l'application dans le navigateur sur le port 1337. L'URL avec le port est affichée dans le terminal.

Une autre façon de démarrer l'application Sails consiste à `node` commande `node` :

```
$ node app.js
```

Cependant, vous perdez certaines fonctionnalités de développement de la commande `lift` telles que le rechargement automatique de l'application lorsque les fichiers d'affichage et les fichiers sont modifiés.

Pour le développement, vous pouvez également utiliser:

```
$ sails console
```

Cela vous permet d'exécuter une commande directement en ligne de commande. C'est très utile pour le débogage des modèles.

Bonjour le monde

Cet exemple montre comment développer notre première application étape par étape, en supposant que Sails est déjà installé et qu'un projet est créé.

1. Créez un fichier de contrôleur vide en tapant

```
$ sails generate controller hello
```

2. Recherchez le nouveau fichier de contrôleur dans `api/controllers/HelloControllers.js` et ajoutez-y la méthode `hello` .

```
module.exports = {

  hello : function (req, res) {
    var myName = 'Luis';
    return res.view('hello' , {name : myName});
    }
}
```

3. Créez un nouveau fichier de vue sous les `views` dossier nommées `hello.ejs` avec le code HTML suivant:

```
<html>
    <head></head>
    <body>
        <p>Hello {{}}.</p>
    </body>
</html>
```

4. Définissez une route dans `config/routes.js` qui appelle la méthode `hello` dans le contrôleur `HelloController` .

```
'GET /' : 'HelloController.hello',
```

Nous avons maintenant implémenté tout le code nécessaire pour cet exemple. Essayons:

1. Démarrer le serveur

```
$ sails lift
```

2. Ouvrez le navigateur et tapez `http://localhost:1337` . Si ce n'est pas le cas, vérifiez l'URL dans la sortie d' `sails lift` . Le port peut être différent.

3. Vous devriez voir la sortie suivante:

   Bonjour Luis

Générer un projet de voiles sans interface

S'il n'y a pas besoin de frontend dans votre prochain projet, vous pouvez lancer de nouvelles voiles avec un indicateur supplémentaire --no-frontend.

```
sails new NameOfProject --no-frontend
```

Cela générera tout le nécessaire pour le backend et omettra les fichiers de vue, les fichiers et les fichiers.

En savoir plus sur la ligne de commande et sails-new:
http://sailsjs.org/documentation/reference/command-line-interface/sails-new

Lire Démarrer avec sails.js en ligne: https://riptutorial.com/fr/sails-js/topic/1205/demarrer-avec-sails-js

Chapitre 2: Adaptateur MongoDB pour les voiles

Examples

Configuration

Vous pouvez configurer les paramètres de la base de données dans `config/connections.js` .

Exemple:

```
someMongoDb: {
  adapter: 'sails-mongo',
  host: 'localhost', // defaults to `localhost` if omitted
  port: 27017, // defaults to 27017 if omitted
  user: 'username_here', // or omit if not relevant
  password: 'password_here', // or omit if not relevant
  database: 'database_name_here' // or omit if not relevant
}
```

Vous pouvez également spécifier votre configuration Mongo en tant qu'URL

```
someMongoDb: {
  adapter: 'sails-mongo',
  url: mongodb://username:password@hostname:port/database
}
```

Installation

Installer à partir de NPM.

```
npm install sails-mongo --save
```

Lire Adaptateur MongoDB pour les voiles en ligne: https://riptutorial.com/fr/sails-js/topic/7047/adaptateur-mongodb-pour-les-voiles

Chapitre 3: Authentification par jeton Web JSON avec Sails

Examples

Configuration

La première étape

Nous devons créer un service appelé *jwToken* . Accédez au répertoire `api/services` et créez `jwToken.js` .

```
'use strict';

const jwt = require('jsonwebtoken'),
      tokenSecret = "secretissecret";

module.exports = {
  // Generates a token from supplied payload
  issue(payload) {
    return jwt.sign(
      payload,
      tokenSecret, // Token Secret that we sign it with
      {
        expiresIn: "30 days" // Token Expire time
      });
  },

  // Verifies token on a request
  verify(token, callback) {
    return jwt.verify(
      token, // The token to be verified
      tokenSecret, // Same token we used to sign
      {}, // No Option, for more see https://github.com/auth0/node-
jsonwebtoken#jwtverifytoken-secretorpublickey-options-callback
      callback //Pass errors or decoded token to callback
    );
  }
};
```

Deuxième étape

Cryptez notre mot de passe en utilisant `bcrypt` . Accédez à `api/models/User.js`

```
'use strict';
const bcrypt = require('bcrypt');

module.exports = {
```

```
    attributes: {
        // your code...
    },

    // Here we encrypt password before creating a User
    beforeCreate(values, next) {
        bcrypt.genSalt(10, (err, salt) => {
            if (err) {
                sails.log.error(err);
                return next();
            }

            bcrypt.hash(values.password, salt, (err, hash) => {
                if (err) {
                    sails.log.error(err);
                    return next();
                }
                values.encryptedPassword = hash; // Here is our encrypted password
                return next();
            });
        });
    },

    comparePassword(password, encryptedPassword) {

        return new Promise(function(resolve, reject) {
            bcrypt.compare(password, encryptedPassword, (err, match) => {
                if (err) {
                    sails.log.error(err);
                    return reject("Something went wrong!");
                }
                if (match) return resolve();
                else return reject("Mismatch passwords");
            });
        });
    }
};
```

Troisième étape

Créer *une* stratégie *isAuthorized* pour vérifier si un utilisateur dispose d'un jeton valide dans l'en-tête de la demande. Accédez à `api/policies` et créez `isAuthorized.js` .

```
'use strict';

module.exports = (req, res, next) => {
  let token;


  if (req.headers && req.headers.token) {
    token = req.headers.token;
    if (token.length <= 0) return res.json(401, {err: 'Format is Authorization: Bearer
[token]'});

  } else if (req.param('token')) {
    token = req.param('token');
```

```
    // We delete the token from param to not mess with blueprints
    delete req.query.token;
  } else {
    return res.json(401, {err: 'No Authorization header was found'});
  }

  jwToken.verify(token, function (err, token) {
    if (err) return res.json(401, {err: 'Invalid Token!'});
    req.token = token; // This is the decrypted token or the payload you provided
    next();
  });
};
```

Quatrième étape

Nous utilisons config / policies.js pour protéger nos contrôleurs

```
module.exports.policies = {

  '*': ['isAuthorized'], // Everything resctricted here
  'UserController': { // Name of your controller
    'create': true // We dont need authorization here, allowing public access
  }
};
```

Cinquième étape

Testons notre implémentation. Aller à `api/controllers` et créer `UserController.js`

```
'use strict';

module.exports = {
    create(req, res) {
        const data = req.body;
        if (data.password !== data.confirmPassword) return res.badRequest("Password not the
same");

        User.create({
                email: data.email,
                password: data.password,
                name: data.name
                    //etc...
            })
            .then((user) => {
                res.send({ token: jwToken.issue({ id: user.id }) }); // payload is { id:
user.id}
            })
            .catch((err) => {
                sails.log.error(err);
                return res.serverError("Something went wrong");
            });
    },

    login(req, res) {
```

```
        const data = req.body;

        if (!data.email || !data.password) return res.badRequest('Email and password
required');

        User.findOne({ email: email })
            .then((user) => {
                if (!user) return res.notFound();

                User.comparePassword(password, user.password)
                    .then(() => {
                        return res.send({ token: jwToken.issue({ id: user.id }) })
                    })
                    .catch((err) => {
                        return res.forbidden();
                    });
            })
            .catch((err) => {
                sails.log.error(err);
                return res.serverError();
            });
    }
};
```

Installation

Nous avons besoin de deux dépendances:

1. **bcrypt** pour le chiffrement `npm install bcrypt --save`
2. **JSON Web token** `npm install jsonwebtoken --save`

Lire Authentification par jeton Web JSON avec Sails en ligne: https://riptutorial.com/fr/sails-js/topic/7050/authentification-par-jeton-web-json-avec-sails

Chapitre 4: Blueprint API

Remarques

Comment fonctionne l'API Blueprint?

Lorsque les voiles commencent à utiliser le système de `sails lift` , les voiles regardent si un contrôleur est défini. Dans notre exemple, nous avons un contrôleur, le contrôleur utilisateur. Sails fournit ensuite un accès aux actions de Blueprint pour ce contrôleur utilisateur, comme si nous les construisions nous-mêmes dans le contrôleur. Sails crée également automatiquement des itinéraires de plans au moment de la levée du serveur. Donc, même si aucune route n'est définie dans `/config/routes.js` et qu'aucune action n'est définie explicitement dans `/api/controllers/UserController.js` , après avoir soulevé le serveur, toutes ces routes et actions sont disponibles.

Examples

Itinéraires Blueprint

Lorsque vous exécutez `sails lift` avec les blueprints activés, la structure inspecte vos contrôleurs, modèles et configurations afin de lier automatiquement certains itinéraires. Ces itinéraires bleus implicites permettent à votre application de répondre à certaines demandes sans que vous ayez à les lier manuellement dans votre fichier `config/routes.js` . Par défaut, les itinéraires des plans directeurs pointent vers les *actions* correspondantes du plan directeur, chacune pouvant être remplacée par du code personnalisé.

Il existe trois types d'itinéraires de plan dans Sails:

- **RESTful routes** , où le chemin est toujours `/:model/:id?` . Lorsque le modèle `User` et le contrôleur sont définis, blueprint lie implicitement les routes RESTful de la manière suivante:

```
'GET /user/:id?': {
  controller: 'User',
  action: 'find'
},
'POST /user': {
  controller: 'User',
  action: 'create'
},
'PUT /user/:id?': {
  controller: 'User',
  action: 'update'
},
'DELETE /user/:id?': {
  controller: 'User',
  action: 'destroy'
}
```

  Ces itinéraires utilisent le verbe HTTP pour déterminer l'action à entreprendre même si

l'itinéraire est identique. Ainsi, une requête `POST` à `/user` créera un nouvel utilisateur, une requête `PUT` à `/user/123` mettra à jour l'utilisateur avec la clé primaire 123 et une requête `DELETE` à `/user/123` supprimera l'utilisateur dont la clé primaire est 123. In Dans un environnement de production, les routes RESTful doivent généralement être protégées par des règles pour éviter les accès non autorisés.

- **Itinéraires de raccourci** , où l'action à prendre est codée dans le chemin. Pour notre modèle d'utilisateur et notre contrôleur, Sails se lie implicitement aux quatre routes de raccourci suivantes.

```
'GET /user/find/:id?': {
  controller: 'User',
  action: 'find'
},
'GET /user/create/:id?': {
  controller: 'User',
  action: 'create'
},
'GET /user/update/:id?': {
  controller: 'User',
  action: 'update'
},
'GET /user/destroy/:id?': {
  controller: 'User',
  action: 'destroy'
}
```

Par exemple, le raccourci `/user/create?name=joe` crée un nouvel utilisateur, tandis que `/user/update/1?name=mike` met à jour le champ de nom de l'utilisateur # 1. Notez que ces itinéraires ne répondent qu'aux requêtes `GET` . Les raccourcis sont très pratiques pour le développement, mais devraient généralement être désactivés dans un environnement de production. Il n'est pas conçu pour être utilisé en production.

- **Itinéraires d'action** , qui créent automatiquement des itinéraires pour vos actions de contrôleur personnalisées. Par exemple, laissez `query` être une action personnalisée définie dans le contrôleur utilisateur. Ensuite, les routes suivantes seraient implicitement disponibles pour les voiles -

```
'GET /user/query/:id?': {
  controller: 'User',
  action: 'query'
},
'POST /user/query/:id?': {
  controller: 'User',
  action: 'query'
},
'PUT /user/query/:id?': {
  controller: 'User',
  action: 'query'
},
'DELETE /user/query/:id?': {
  controller: 'User',
  action: 'query'
}
```

Si la demande est faite dans `/user/query/:id?` route alors indépendante sur le verbe HTTP, l'action serait la même. Contrairement aux itinéraires RESTful et aux raccourcis, les itinéraires d'action *ne* nécessitent *pas* qu'un contrôleur dispose d'un fichier de modèle correspondant. Ce qui signifie que si vous définissez un contrôleur dans le fichier `/api/controllers/FooController.js` mais pas de modèle dans le fichier `/api/models/Foo.js` , il n'y aurait pas de route RESTful ou de raccourci avec `/foo` mais il y aura toujours des routes d'action disponible à l'emploi.

Ordre des itinéraires correspondant

Lorsqu'une demande arrive, les voiles vont d'abord correspondre à l'itinéraire par rapport à des itinéraires explicitement définis. Si elle correspond, aucune autre correspondance n'est effectuée et l'action correspondante est exécutée. Mais si elle ne correspond pas, la route est comparée en premier lieu aux routes d'action du plan, si elle ne correspond pas aux routes de repos et si elle ne correspond pas aux routes de raccourci. Donc, si votre fichier `/config/routes.js` a une entrée comme la suivante-

```
'/user/query/:id?': {
  controller: 'User',
  action: 'find'
}
```

Ensuite, vous ne pouvez pas vous attendre à ce que `query` routes d'action de `query` fonctionnent. Étant donné que le même itinéraire que l'itinéraire d'action de `query` serait mis en correspondance avec les routes à définition explicite et `find` action du contrôleur d'utilisateur serait exécutée.

Actions Blueprint

Les actions Blueprint (à ne pas confondre avec les *routes d'* action Blueprint) sont des actions génériques conçues pour fonctionner avec n'importe lequel de vos contrôleurs ayant un modèle du même nom (par exemple, `ParrotController` nécessiterait un modèle `Parrot`). Considérez-les comme le comportement par défaut de votre application. Par exemple, si vous avez un modèle `User.js` et un contrôleur `UserController.js` vide, `find` , `create` , `update` , `destroy` , `populate` , `add` et `remove` actions implicitement, sans que vous ayez à les écrire.

Par défaut, les itinéraires et les raccourcis RESTful du plan directeur sont liés à leurs actions correspondantes. Cependant, toute action de plan directeur peut être remplacée pour un contrôleur particulier en créant une action personnalisée dans ce fichier de contrôleur (par exemple, `ParrotController.find`). Vous pouvez également remplacer l'action de plan directeur *partout dans votre application* en créant votre propre action de plan directeur personnalisée.

Sails est livré avec les actions suivantes:

- trouver
- trouverOne
- créer
- mettre à jour

- détruire
- peupler
- ajouter
- retirer

Désactivation des itinéraires du plan directeur

- **Base globale** : La configuration de l'API Blueprint est définie dans le fichier `/config/blueprint.js` . Vous pouvez activer ou désactiver les trois types d'itinéraires de plan directeur pour tous les contrôleurs à partir de là. Par exemple, si vous souhaitez désactiver les itinéraires de raccourcis blueprint pour tous vos contrôleurs, mais que vous souhaitez conserver les routes d'action et de repos activées, alors votre `/config/blueprint.js` devrait être comme ça -

```
module.exports.blueprints = {
  action: true,
  rest: true,
  shortcut: false
}
```

- **Par contrôleur** : vous pouvez également remplacer les paramètres de `/config/blueprints.js` par contrôleur en définissant une clé '_config' dans la définition de votre contrôleur et en lui affectant un objet de configuration avec les paramètres de remplacement. dans ce fichier Par exemple, si vous souhaitez que les raccourcis soient uniquement activés pour votre contrôleur utilisateur, mais pas pour d'autres contrôleurs, vous devez disposer de la paire de valeurs clé suivante dans le contrôleur utilisateur avec la configuration de plan ci-dessus.

```
module.exports = {
  _config: {
    actions: true,
    shortcuts: true,
    rest: true
  }
}
```

Lire Blueprint API en ligne: https://riptutorial.com/fr/sails-js/topic/3749/blueprint-api

Chapitre 5: Configurer mysql avec sails.js

Examples

Comment configurer la connexion à la base de données mysql dans sails.js

Pour ce faire, localisez d'abord le dossier de configuration dans votre racine. Puis ouvrez `connections.js`

Localiser

```
// someMysqlServer: {
//   adapter: 'sails-mysql',
//   host: 'YOUR_MYSQL_SERVER_HOSTNAME_OR_IP_ADDRESS',
//   user: 'YOUR_MYSQL_USER', //optional
//   password: 'YOUR_MYSQL_PASSWORD', //optional
//   database: 'YOUR_MYSQL_DB' //optional
// },
```

Décommentez ces lignes.

Donnez un nom approprié pour le connecteur comme thisMysqlServer à mysql_connection ou n'importe quel nom pour votre souhait

```
mysql_connection: {
  adapter: 'sails-mysql',
  host: '127.0.0.1', // can user localhost or mysql connection either
  user: 'root', // your mysql username
  password: 'xxxxxxxxx', // your mysql password
  database: 'your database name here' // database name
},
```

Enregistrer le fichier

Accédez à votre dossier racine et exécutez la commande suivante:

```
$ npm install sails-mysql --save
```

Note: en exécutant la commande ci-dessus, nous installons le package du pilote mysql pour sails.js.

Vous avez terminé.

Maintenant, vous pouvez utiliser mysql_connection comme nom de connexion dans votre configuration de modèle. Lorsque vous soulevez votre application en utilisant la commande suivante:

```
$ sails lift
```

vous modélisez le schéma sera automatiquement mis à jour dans la base de données MySQL

Lire Configurer mysql avec sails.js en ligne: https://riptutorial.com/fr/sails-js/topic/5317/configurer-mysql-avec-sails-js

Chapitre 6: Contrôleurs

Remarques

Les contrôleurs (le **C** dans **MVC**) sont les principaux objets de votre application Sails chargés de répondre aux requêtes provenant d'un navigateur Web, d'une application mobile ou de tout autre système capable de communiquer avec un serveur. Ils agissent souvent comme un intermédiaire entre vos modèles et vos points de vue. Pour de nombreuses applications, les contrôleurs contiendront l'essentiel de la logique métier de votre projet.

Examples

ES2015 Syntaxe

```
'use strict';

// This is an example of a /api/controllers/HomeController.js
module.exports = {
  // This is the index action and the route is mapped via /config/routes.js
  index(req, res) {
    // Return a view without view model data
    // This typically will return the view defined at /views/home/index.<view engine
extension>
    return res.view();
  },
  foo(req, res) {
    // Return the 'foo' view with a view model that has a `bar` variable set to the query
string variable `foobar`
    return res.view({
      bar: req.param('foobar'),
    });
  },
};
```

Utilisation des générateurs ES2015 avec co.js

```
'use strict';

const co = require('co');

module.exports = {
  // This is the index action and the route is mapped via /config/routes.js
  index(req, res) {
    co(function* index() {
      // Return a view without view model data
      // This typically will return the view defined at /views/home/index.<view engine
extension>
      return res.view();
    }).catch(res.negotiate); // Catch any thrown errors and pass the error to the `negotiate`
policy.
  },
  foo(req, res) {
```

```
    co(function* foo() {
      // Get an array of `FooBar` items from the database
      const items = yield FooBar.find();

      // Return the 'foo' view with a view model containing the array of `FooBar` items
      return res.view({
        items,
      });
    }).catch(res.negotiate); // Catch any thrown errors and pass the error to the `negotiate`
policy.
  },
};
```

Lire Contrôleurs en ligne: https://riptutorial.com/fr/sails-js/topic/3521/controleurs

Chapitre 7: Des modèles

Remarques

Sails est livré avec un puissant ORM / ODM appelé Waterline, un outil indépendant du magasin de données qui simplifie considérablement l'interaction avec une ou plusieurs bases de données. Il fournit une couche d'abstraction au-dessus de la base de données sous-jacente, ce qui vous permet d'interroger et de manipuler facilement vos données sans écrire de code d'intégration spécifique au fournisseur.

Examples

Modèle de base

Cet exemple montre comment définir un modèle simple dans Sails.js

Vous pouvez générer un fichier modèle vide en tapant

```
sails generate model car
```

Vous trouverez le nouveau fichier `Car.js` dans `api/models/` .

Ensuite, vous remplissez certains détails.

```
modules.exports = {

  tableName : 'cars',
  connection : 'mongodb',

  attributes : {

    id : {
      type : 'integer',
      unique : true,
      primaryKey : true,
      autoIncrement : true
    },

    brand : {
      type : 'string',
      size : 25
    },

    description : {
      type: 'text',
      defaultsTo : ''
    },

    price : {
      type : 'float',
      required : true
```

```
    },

    seats : {
      type : 'integer'
    },

    sell_date : {
      type : 'datetime'
    },

    has_cooler : {
      type : 'boolean',
      columnName : 'cooler'
    },

    chassis_number : {
      unique : true,
      type : 'string'
    },

    color : {
      type : 'string',
      enum: ['white', 'red', 'black']
    }

  }

};
```

L'exemple ci-dessus utilise presque toutes les options de modèle possibles, expliquées ci-dessous.

1. tableName

Ce paramètre définit le nom de la table qui sera créée dans la base de données. S'il n'est pas défini, le nom du modèle sera utilisé (`car` dans cet exemple).

2. connexion

Ce particulier définit la connexion à la base de données utilisée pour le modèle. Les détails de cette connexion sont définis sous la clé `mongodb` dans `config/connections.js` . Voici le format d'une connexion:

```
mongodb : {

  // The driver that connect our models with the database
  adapter : '<adapter>',

  // The database parameters
  user : '<username>',
  port : <port>,
  host : '<host>',
  database : '<database>'

}
```

3. attributs

Chaque attribut référence une colonne dans la table de base de données pour le modèle. Dans cet exemple, neuf colonnes seront créées. Chaque colonne peut être configurée avec une ou plusieurs des clés suivantes:

- **type** : Le type de données de la colonne. Cette page répertorie tous les types disponibles.
- **unique** : si la valeur est true, une erreur se produira si vous essayez de créer un objet ayant la même valeur pour cette colonne que celui déjà présent dans la base de données.
- **primaryKey** : Si true, la colonne fonctionnera comme clé primaire.
- **autoIncrement** : Une séquence sera associée à la colonne avec un numéro auto incrémentable commençant à 0.
- **size** : la longueur maximale de la colonne.
- **required** : Si true, il ne peut pas être nul.
- **columnName** : configure la colonne dans la base de données, qui utilise par défaut le nom de l'attribut.
- **enum** : Nous pouvons définir un tableau d'options possibles pour un attribut.

Lire Des modèles en ligne: https://riptutorial.com/fr/sails-js/topic/4646/des-modeles

Chapitre 8: Le routage

Remarques

Les routes sont des règles qui indiquent à Sails quoi faire face à une requête entrante.

Les routes sont définies dans `config/routes.js` . L'ordre des routes est important, car les routes sont appariées de haut en bas. Cela signifie que si un itinéraire spécifique peut également être associé à un itinéraire générique, l'itinéraire spécifique doit être défini au-dessus de l'itinéraire générique.

Lorsqu'une requête entre dans votre application, sails.js récupère tous les paramètres qui l'accompagnent et les met à votre disposition en tant que `params` sur l'objet de requête.

Les propriétés de l'objet cible de routage seront transmises au gestionnaire de routage dans l'objet req.options. Les propriétés réservées suivantes peuvent affecter le comportement du gestionnaire de routage:

Propriété	Types de cibles applicables	Type de données	Détails
sauterAssets	tout	Booléen	Défini sur `true` si vous *ne* souhaitez *pas* que la route corresponde aux URL contenant des points (par exemple, myImage.jpg). Cela gardera vos routes avec une notation générique à partir des URL correspondantes des ressources statiques. Utile lors de la création de slugs URL.
skipRegex	tout	Regexp	Si ignorer chaque URL contenant un point est trop permissif ou si vous devez ignorer un gestionnaire de route en fonction de différents critères, vous pouvez utiliser `skipRegex` . Cette option vous permet de spécifier une expression régulière ou un tableau d'expressions régulières pour correspondre à l'URL de la demande; Si l'une des correspondances réussit, le gestionnaire est ignoré. Notez que contrairement à la syntaxe de liaison d'un gestionnaire avec une expression régulière, `skipRegex` attend des objets _Exact RegExp, pas des chaînes.
des locaux	contrôleur,	dictionnaire	Définit les variables locales par défaut à

Propriété	Types de cibles applicables	Type de données	Détails
	vue, plan, réponse		transmettre à toute vue rendue lors du traitement de la demande.
cors	tout	Dictionnaire ou booléen ou chaîne	Spécifie comment gérer les demandes pour cette route à partir d'une origine différente.
peupler	plan	Booléen	Indique si les champs de modèle associés doivent être remplis dans les résultats d'une action blueprint "find" ou "findOne". Valeur par défaut à la valeur définie dans `config/blueprints.js` .
sauter , **limiter** , **trier** , **où**	plan	dictionnaire	Définir les critères pour "trouver" le plan.

Examples

Définitions de route RESTful personnalisées

```
module.exports.routes = {
    'GET /foo': 'FooController.index',
    'GET /foo/new': 'FooController.new',
    'POST /foo/create': 'FooController.create',
    'GET /foo/:id/edit': 'FooController.edit',
    'PUT /foo/:id/update': 'FooController.update',
    'GET /foo/:id': 'FooController.show',
    'DELETE /foo/:id': 'FooController.delete',
};
```

Réorienter

```
module.exports.routes = {
  '/foo': '/bar',
  'GET /google': 'http://www.google.com'
};
```

Définir une variable personnalisée pour toutes les vues

```
module.exports.routes = {
  // This function will be executed for all http verbs on all urls
  'all /*', function (req, res, next) {
    // Expose the function `fooBar` to all views (via the locals object)
    res.locals.fooBar = function (arg1) {
```

```
      return 'foobar' + arg1;
    };
  },
};
```

Ignorer les actifs (URL contenant des points) de la route générique

```
module.exports.routes = {
  'GET /foo/*': {
     fn: function(req, res) {
       res.send("FOO!");
     },
     skipAssets: true
  },
};
```

Routes avec RegEx

```
module.exports.routes = {
    // sends matching regex (few patterns) as 'page' param
    'r|/foo/([0-9]+)|page': 'FooController.get',
    'r|/foo/(.*)|page': 'FooController.get',
    'r|/foo/(\\w+)|page': 'FooController.get'
};
```

Slugs d'URL

```
module.exports.routes = {
  'GET /blog/:year/:month/:day/:posttitle/': 'BlogController.showPost',
  'GET /blog/:year/:month/:day/': 'BlogController.showDayArchive',
  'GET /blog/:year/:month/': 'BlogController.showMonthArchive',
  'GET /blog/:year/': 'BlogController.showYearArchive',
};
```

Les paramètres transmis dans l'URL sont alors accessibles dans les actions de contrôleur correspondantes à l'aide de `req.param('year')` , `req.param('month')` etc.

Par exemple, un `GET` demande `/blog/2016/08/` déclenche l' `BlogController.showMonthArchive` action du contrôleur, avec `req.param('year')` ayant la valeur `2016` et `req.param('month')` ayant la valeur `08` .

Lire Le routage en ligne: https://riptutorial.com/fr/sails-js/topic/4577/le-routage

Chapitre 9: PostgreSQL Database Adapter pour Sails

Examples

Installer

Vous pouvez installer l'adaptateur postgreSQL via NPM.

```
npm install sails-postgresql
```

Configuration

Vous pouvez configurer les paramètres de la base de données dans `config/connections.js` .

Voici un exemple:

```
postgresql: {
  database: 'databaseName',
  host: 'localhost',
  user: 'root',
  password: '',
  port: 5432,
  poolSize: 10,
  ssl: false
};
```

Vous pouvez également fournir les informations de connexion au format URL:

```
postgresql: {
  url: 'postgres://username:password@hostname:port/database',
  ssl: false
};
```

Lire PostgreSQL Database Adapter pour Sails en ligne: https://riptutorial.com/fr/sails-js/topic/4676/postgresql-database-adapter-pour-sails

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec sails.js	Bonanza, Community, hlozancic, Jim Geurts, Lost in OWL, Luis González, mouche, Sebastialonso, Viktor, Yann Bertrand
2	Adaptateur MongoDB pour les voiles	SkyQ
3	Authentification par jeton Web JSON avec Sails	SkyQ, Viktor
4	Blueprint API	taufique
5	Configurer mysql avec sails.js	vijeeshin, Yann Bertrand
6	Contrôleurs	Jim Geurts
7	Des modèles	Luis González, mouche, Viktor
8	Le routage	Bonanza, Jim Geurts, Viktor
9	PostgreSQL Database Adapter pour Sails	Luis González, mouche