Kostenloses eBook

LERNEN

cucumber

Free unaffiliated eBook created from
Stack Overflow contributors.

#cucumber

Inhaltsverzeichnis

Über

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

It is an unofficial and free cucumber 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 cucumber.

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 Gurke

Bemerkungen

Über Gurke

Gurke ist ein Werkzeug, das ausführbare Spezifikationen von Software ausführt. Spezifikationen, "Features" genannt, sind in strukturierter natürlicher Sprache verfasst. Cucumber führt ein Feature aus, indem jeder Schritt einer "Schrittdefinition" zugeordnet wird, die in der Programmiersprache geschrieben ist, die von dieser Implementierung von Cucumber unterstützt wird. Gurke ist in vielen Programmiersprachen implementiert, darunter Ruby (das Original), Java und Javascript . Es wird auch in viele menschliche Sprachen übersetzt.

Gurke wurde geschrieben, um die agile Methodik namens Behavior-Driven Development (BDD) zu unterstützen. Bei BDD beginnt man mit der Entwicklung von außerhalb mit dem Schreiben von Abnahmetests, die die Funktionalität der Software aus Sicht des Benutzers beschreiben (und nicht aus Sicht eines Programmierers wie bei Unit-Tests). Gurkenmerkmale dienen als Abnahmetests.

Im Allgemeinen handelt es sich bei Cucumber-Funktionen um eine von Menschen lesbare Dokumentation, die auch eine ausführbare Testsuite ist, sodass Dokumentation und Tests immer übereinstimmen. Gurke ist hilfreich bei der Kommunikation mit Nicht-Programmierer-Stakeholdern über Dokumentation und Tests. Es ermöglicht Programmierern auch, Tests auf konzeptioneller Ebene zu schreiben, ohne irrelevante Probleme in der Programmiersprache zu verursachen.

Gurke wird am häufigsten verwendet, um Webanwendungen mithilfe eines Browsertreibers wie Selenium oder PhantomJS anzugeben und zu testen. Es kann jedoch mit jeder Software verwendet werden, die ausgeführt werden kann und deren Status oder Ergebnisse aus der Programmiersprache bestimmt werden können, die eine Cucumber-Implementierung unterstützt.

Andere Dokumentation

Offizielle Dokumentation finden Sie unter https://cucumber.io/docs . Die Dokumentation, die mit den Gurkenfunktionen erstellt wurde, die die Gurkenimplementierungen beschreiben, ist unter

- JavaScript: https://relishapp.com/cucumber/cucumber-js/docs
- Ruby: https://relishapp.com/cucumber/cucumber/docs

https://relishapp.com/explore enthält einige andere Tools und Beispiele für Cucumber, allerdings nicht Cucumber-JVM.

Dieses Thema

Dieses Thema sollte nur einige Beispiele geben, die den Leser in die Gurkenkonzepte einführen. Andere Abschnitte enthalten vollständige Beispiele für die Installation, die Verwendung der Befehlszeile und der IDE, die Funktionen, Schrittdefinitionen usw.

Examples

Eine Gurkenfunktion

Gurke verwendet die Gherkin-Syntax , um das Verhalten Ihrer Software in strukturierter natürlicher Sprache zu beschreiben.

Daher ist Gurke **kein** Testframework (ein häufiges Missverständnis), sondern ein *Systemdokumentationsframework* , das sich nicht sehr von anderen wie dem Use Case Scenario unterscheidet. Das häufige Missverständnis ist darauf zurückzuführen, dass die Gurkendokumentation *automatisiert werden kann, um sicherzustellen, dass sie das tatsächliche Systemverhalten widerspiegelt* .

Eine Cucumber-Dokumentations-Suite besteht aus `Features` , von denen jede eine Funktion Ihrer Software beschreibt, die in Gherkin geschrieben und in einer eigenen Datei gehostet ist. Durch das Organisieren dieser Dateien in einer Verzeichnisstruktur können Sie Funktionen *gruppieren* und *organisieren* :

- Bankwesen /
  - Rückzug
  - atm.feature
  - persönliches Darlehen
- Handel/
  - portfolio.feature
  - Intraday-Feature
- Hypothek/
  - Bewertung.Feature
  - buchhaltung.merkmal

Jedes `Feature` ist eine Nur-Text-Datei, die aus einem optionalen, unstrukturierten, rein informativen Einführungsabschnitt und einem oder mehreren `Scenarios` , die jeweils eine Verwendungsbedingung oder einen Anwendungsfall darstellen.

Beispiel:

```
Feature: Documentation
As a StackOverflow user or visitor
I want to access the documentation section

    Scenario: search documentation on Stack Overflow
        Given I am on StackOverflow
        And I go to the Documentation section
        When I search for "cucumber"
        And I follow the link to "cucumber"
        Then I should see documentation for "cucumber"
```

Jede Zeile, die mit *Given* , *When* , *And* , *But* oder *Then* beginnt, wird als `Step` . Jeder Schritt kann mit jedem dieser Wörter unabhängig von der Reihenfolge beginnen, es ist jedoch üblich, sie auf möglichst natürliche Weise zu verwenden.

Features können auch über `Tags` organisiert werden, Anmerkungen, die der Editor für ein `Feature` `Scenario` kann, oder ein `Scenario` um es weiter zu kategorisieren.

Die Ausführbarkeit eines Features wird durch *Glue*- Code erreicht, der in vielen verschiedenen Sprachen (Java, Ruby, Scala, C / C ++) geschrieben werden kann: Jeder `Step` wird mit dem Glue-Code abgeglichen, um über `Step Definitions` (üblicherweise als *StepDef* abgekürzt) zu *identifizieren* Reguläre Ausdrücke.

Jedem `Step` kann nur eine `Step Definition` .

Wenn ein `Feature` ausgeführt wird, wird jedes zusammenfassende `Scenario` ausgeführt. Das bedeutet, dass jedes StepDef, das mit den `Step` in jedem `Scenario` übereinstimmt, ausgeführt wird.

Reine Rubin-Installation

Um Gurke zur Verwendung mit Ruby zu installieren, verwenden Sie einfach den Befehl

```
gem install cucumber
```

Wenn Sie Bundler verwenden, können Sie alternativ die folgende Zeile zu Ihrem Gemfile hinzufügen

```
gem 'cucumber'
```

Und dann Bundler ausführen

```
bundle install
```

[Ich denke, das gehört zu seinem eigenen Thema, Installation. Ich habe dieses Thema erstellt und dieses Beispiel dort kopiert. Wenn das Thema genehmigt ist, werde ich es dort verschieben und die Kopie löschen.]

Eine Gurkenschrittdefinition in Ruby

In features / step_definitions / documentation.rb:

```
When /^I go to the "([^"]+)" documentation$/ do |section|
  path_part =
    case section
      when "Documentation"
        "documentation"
      else
        raise "Unknown documentation section: #{section}"
    end
  visit "/documentation/#{path_part}/topics"
end

Then /^I should see the "([^"]+) documentation"$/ do |section|
  expect(page).to have_css('h2.doctag_title a', text: section)
end
```

Diese Schritte üben eine Webanwendung aus. Sie sind so einfach wie sie sein können, während sie praktisch sind.

Jeder Schritt beginnt mit einem Gherkin-Schlüsselwort, das in einer Schrittdefinitionsdatei eine Methode ist, die einen Schritt bei Cucumber registriert. Die schrittbestimmende Methode benötigt einen regulären Ausdruck, der einer Zeile in einem Szenario entspricht, und einen Block, der ausgeführt wird, wenn das Szenario eine übereinstimmende Zeile erreicht. Erfassungsgruppen im regulären Ausdruck werden als Blockparameter an den Block übergeben.

Der Schritt " `When` enthält ein einfaches Inline-Beispiel für den Übergang von einer für Menschen lesbaren Referenz auf eine Seite ("Dokumentation") zu einer URL. Echte Gurkensuiten verwenden diese Logik normalerweise in einer separaten Methode. Die `visit` wird von Capybara bereitgestellt. Capybara ist nicht erforderlich, um Gurke zu verwenden, obwohl es häufig verwendet wird. `visit` teilt dem von Capybara kontrollierten Browser mit, die angegebene URL aufzurufen.

Der `Then` Schritt zeigt, wie der Inhalt einer Seite getestet werden kann. `expect` / `to` durch RSpec bereitgestellt (auch hier nicht von Gurken erforderlich , aber sehr häufig verwendet , um mit ihm). `have_css` wird von Capybara zur Verfügung gestellt. Die Erwartung ist, dass der angegebene CSS-Selektor einem Element auf der Seite entspricht, das den angegebenen Text enthält. Beachten Sie, dass diese Erwartung fehlschlagen würde, wenn die Browseranforderung fehlgeschlagen war.

Weitere Beispiele finden Sie im Thema "Schrittdefinition" .

Erste Schritte mit Gurke online lesen: https://riptutorial.com/de/cucumber/topic/4875/erste-schritte-mit-gurke

Kapitel 2: Eigenschaften

Einführung

Sie können Gurke als Plugin in QTP und Selenium verwenden. Die Schritte im Gurkenszenario sind globale Variablen. Sie können einmal implementieren und mehrmals aufrufen. Dadurch wird die Codewartung reduziert und derselbe Code kann bei Bedarf wiederverwendet werden.

Bemerkungen

Gurke Features sind in der Gherkin Sprache geschrieben und in Dateien mit der Endung gespeichert `.feature` . Dieses Thema enthält Beispiele für jede Funktion von Gherkin.

Examples

Eine minimale Gurkenfunktion

In features / documentation.feature:

```
Feature: Documentation

  Scenario: User views documentation
    When I go to the "Cucumber" documentation
    Then I should see the "Cucumber" documentation
```

Ein minimales Feature enthält eine `Feature` Zeile und ein `Scenario` mit einem oder mehreren Schritten, die mit `When` , `Then` oder einem anderen Gherkin-Schlüsselwort beginnen.

Ein vernünftiges Szenario hätte wahrscheinlich mehr als einen Schritt.

Szenario-Gliederung

Vorlage wie unten

```
Scenario Outline: As a homemaker i want to buy and pay for the below product
  Given I purchase <a product>
    And I require a carry bag to take things to home
  When I pay bill using <payment method> to successfully checkout
  Then I should have a receipt

Examples:
| a product     | payment method |
|  Cake         | Visa           |
|  Coke         | Paypal         |
```

Syntaxverwendung

```
Feature: Some terse yet descriptive text of what is desired
     Textual description of the business value of this feature
     Business rules that govern the scope of the feature
     Any additional information that will make the feature easier to understand

Background:
    Given some precondition needed for all scenarios in this file
        And another precondition

Scenario: Some determinable business situation
  Textual description of the business value of this scenario
  Business rules that govern the scope of the scenario
  Any additional information that will make the scenario easier to understand
    Given some precondition
      And some other precondition
    When some action by the actor
      And some other action
      And yet another action
    Then some testable outcome is achieved
      And something else we can check happens too
      But something else we can check does not happen

Scenario Outline: Some determinable business situation
    Given I am <precondition>
        And some other precondition
    When some action by the actor
    Then I have <outcome> rights

Examples:
    | precondition | outcome  |
    | username1    | customer |
    | username2    | admin    |
```

Die folgenden Schlüsselwörter sind austauschbar, können jedoch je nach Kontext besser verwendet werden:

- `Feature:` | `Ability:` | `Business Need:`
- `Scenario Outline:` | `Scenario Template:`
- `Examples:` | `Scenarios:`
- `Given` | `When` | `Then` | `And` | `But` | `*` |

Eigenschaften online lesen: https://riptutorial.com/de/cucumber/topic/6023/eigenschaften

Kapitel 3: Gurke-Syntax

Einführung

Gherkin ist eine für Unternehmen lesbare Sprache für Testautomatisierung und Testdokumentation. Es wird von Gurke verstanden und existiert zusammen als verhaltensgesteuertes Entwicklungswerkzeug.

Syntax

- Feature: Dieses Schlüsselwort gibt an, dass es sich bei der folgenden Beschreibung um eine grundlegende Beschreibung oder den Namen des zu testenden oder dokumentierten Features handelt.
- Hintergrund: Dieses Schlüsselwort kennzeichnet Schritte, die vor jedem Szenario in der Funktion ausgeführt werden.
- Szenario: Dieses Schlüsselwort steht für den Namen oder die grundlegende Beschreibung eines bestimmten Szenarios, das die Funktion testet.
- Szenario-Übersicht: Dieses Schlüsselwort gibt an, dass das Szenario N-mal für jedes Argument ausgeführt wird, das in Beispielen aufgeführt ist, die explizit durch Spaltennamen übergeben werden, die in spitzen Klammern eingeschlossen sind.
- Beispiele: Bei diesem Schlüsselwort wird die Liste der statischen Argumente aufgeführt, die an eine Szenario-Gliederung übergeben werden.
- Gegeben: Dieses Schlüsselwort repräsentiert einen bestimmten Schritt oder eine Voraussetzung, die angenommen wird, bevor Sie fortfahren. Im Arrangement, Act, Assert-Paradigma steht "Arrange" für gegeben.
- When: Dieses Schlüsselwort steht für einen When-Schritt oder das Verhalten, gegen das behauptet werden soll. Im Arrangement, Act, Assert-Paradigma repräsentiert gegebenes "Act".
- Dann: Dieses Schlüsselwort steht für einen Then-Schritt, oder anders ausgedrückt, für den Schritt, in dem das Ergebnis eines Verhaltens validiert wird. Im Arrangement, Act, Assert-Paradigma repräsentiert "Assert".
- Und: Dieses Schlüsselwort wird in Verbindung mit einem der oben genannten Schlüsselwörter verwendet. Wenn Sie zwei gegebene Anweisungen haben, anstatt Given zweimal explizit aufzurufen, können Sie sagen: "A und B".

Examples

Die Grundlagen

In diesem Beispiel wird die grundlegende Struktur einer Gurken-Feature-Datei in Gherkin beschrieben. Feature-Dateien verwenden mehrere Schlüsselwörter in der Grundsyntax.

Schauen wir uns die grundlegenden Schlüsselwörter an:

- **Feature:** Dieses Schlüsselwort gibt an, dass es sich bei der folgenden Beschreibung um eine grundlegende Beschreibung oder den Namen des zu testenden oder dokumentierten Features handelt.
- **Szenario:** Dieses Schlüsselwort steht für den Namen oder die grundlegende Beschreibung eines bestimmten Szenarios, das die Funktion testet.
- **Bei** Angabe dieses Schlüsselworts handelt es sich um einen bestimmten Schritt oder eine Voraussetzung, die angenommen wird, bevor Sie fortfahren. Im Arrangement, Act, Assert-Paradigma steht "Arrange" für gegeben.
- **Wenn** dieses Schlüsselwort einen Wenn-Schritt darstellt oder das Verhalten, gegen das geltend gemacht werden soll. Im Arrangement, Act, Assert-Paradigma repräsentiert gegebenes "Act".
- **Dann** repräsentiert dieses Schlüsselwort einen dann-Schritt, oder anders ausgedrückt, den Schritt, in dem das Ergebnis eines Verhaltens validiert wird. Im Arrangement, Act, Assert-Paradigma repräsentiert "Assert".
- **Und** Dieses Schlüsselwort wird oben in Verbindung mit einem der Schlüsselwörter verwendet. Wenn Sie zwei gegebene Anweisungen haben, anstatt Given zweimal explizit aufzurufen, können Sie sagen: "A und B".
- **Aber** Dieses Schlüsselwort wird in Verbindung **gegeben, als** und **dann** um anzuzeigen , dass etwas sollte nicht passieren. Dann A aber nicht B.

Alle Schlüsselwörter müssen sich in einer neuen Zeile befinden und müssen das erste Wort in einer neuen Zeile sein, um vom Gherkin-Parser erkannt zu werden. Die Schlüsselwörter Feature und Scenario müssen unmittelbar danach einen Doppelpunkt haben, wie im folgenden Beispiel dargestellt. Gegeben, Wann, Dann und Und benötigen keinen Doppelpunkt.

Zusätzlich zu den Schlüsselwörtern können Sie Beschreibungen und Kommentare schreiben. Beschreibungen werden nach dem Schlüsselwort aber in derselben Zeile angezeigt, während Kommentare in Zeilen unter den Schlüsselwörtern auftreten. Beim Schreiben von Feature-Kommentaren ist es üblich, explizite Regeln anzugeben, die Kanten und Bedingungen umreißen, die zu unterschiedlichen Ergebnissen des Verhaltens führen.

```
Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Scenario: The user successfully logs in with valid credentials
        This scenario tests that a user is able to successfully login
        provided they enter a valid username, valid password, and
        currently have an active subscription on their account.

        Given the user is on the login page
        When the user signs in with valid credentials
        Then the user should be logged in
```

Parametrisierte Schritte

Beim Schreiben von Gherkin kann es Zeiten geben, in denen Sie Ihre Schritte für die Wiederverwendbarkeit durch den Ingenieur parametrisieren möchten, der die Testpläne implementiert. Schritte erhalten Parameter über Erfassungsgruppen für reguläre Ausdrücke. (**Technischer Hinweis:** Wenn für jede Erfassungsgruppe in Ihrem regulären Ausdruck keine übereinstimmenden Parameter vorhanden sind, können Sie erwarten, dass "CucumberException: Arity Mismatch" ausgelöst wird) als Ganzzahlen als Argumente akzeptieren.

```
Feature: Product Login
   As a user, I would like to be able to use my credentials to successfully
   login.

   Rules:
   - The user must have a valid username
   - The user must have a valid password
   - The user must have an active subscription
   - User is locked out after 3 invalid attempts
   - User will get a generic error message following
     login attempt with invalid credentials

   Scenario: The user successfully logs in with valid credentials
       This scenario tests that a user is able to successfully login
       provided they enter a valid username, valid password, and
       currently have an active subscription on their account.

       Given the user is on the login page
       When the user signs in with "valid" credentials
       Then the user should be logged in

   Scenario: The user attempts to log in with invalid credentials
       This scenario tests that a user is not able to log in when
       they enter invalid credentials

       Given the user is on the login page
       When the user signs in with "invalid" credentials
       Then the user should be logged in

   Scenario: The user is locked out after too many failed attempts
       This scenario validates that the user is locked out
       of their account after failing three consecutive
       attempts to log in

       Given the user is on the login page
       When the user fails to log in 3 times
       Then the user should be locked out of their account
```

Feature-Hintergrund

Wie Sie vielleicht im obigen Beispiel bemerkt haben, schreiben wir denselben Schritt mehrmals:

```
Given the user is on the login page
```

Dies kann außerordentlich langwierig sein, insbesondere wenn mehr als ein bestimmter Schritt verwendet wird, der wiederverwendet wird. Gherkin bietet eine Lösung dafür, indem wir uns ein

weiteres Stichwort geben, mit dem wir arbeiten können: **Background:.**

Das Hintergrundschlüsselwort dient dazu, die darunter deklarierten Schritte vor jedem Szenario im Feature auszuführen. Fügen Sie keinen Hintergrundschritt hinzu, es sei denn, Sie sind sich sicher, dass dies für jedes Szenario erforderlich ist. Wie bei den anderen Schlüsselwörtern folgt auf Background eine Beschreibung oder ein Name und es können Kommentare darunter angezeigt werden. Ähnlich wie bei Feature und Szenario muss der Hintergrund mit einem Doppelpunkt fortgesetzt werden.

```
Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Background: The user starts out on the login page
        Given the user is on the login page

    Scenario: The user successfully logs in with valid credentials
        This scenario tests that a user is able to successfully login
        provided they enter a valid username, valid password, and
        currently have an active subscription on their account.

        When the user signs in with "valid" credentials
        Then the user should be logged in

    Scenario: The user attempts to log in with invalid credentials
        This scenario tests that a user is not able to log in when
        they enter invalid credentials

        When the user signs in with "invalid" credentials
        Then the user should be logged in

    Scenario: The user is locked out after too many failed attempts
        This scenario validates that the user is locked out
        of their account after failing three consecutive
        attempts to log in

        When the user fails to log in 3 times
        Then the user should be locked out of their account
```

Szenario-Gliederung

In einigen Fällen möchten Sie möglicherweise dasselbe Szenario immer wieder ausführen, indem Sie die Argumente ersetzen. In diesem Fall bietet Gherkin mehrere neue Schlüsselwörter für diese Situation an: **Szenario-Gliederung:** und **Beispiel:.** Das Schlüsselwort Szenario-Gliederung teilt Cucumber mit, dass das Szenario mehrmals ausgeführt wird, um Argumente aus einer Liste zu ersetzen. Das Schlüsselwort Examples wird aufgerufen, bevor die Liste explizit notiert wird. Argumente für Szenario-Konturen sollten in spitzen Klammern stehen. Beachten Sie im

nachstehenden Beispiel, dass die in eckigen Klammern eingeschlossenen Argumentnamen den unter Beispielen aufgeführten Spaltennamen entsprechen. Jede Spalte ist durch vertikale Striche getrennt, wobei die Spaltennamen in der ersten Zeile stehen.

```
Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Background: The user starts out on the login page
        Given the user is on the login page

    Scenario Outline: The user successfully logs in with their account
         This scenario outlines tests in which various users attempt
         to sign in successfully

         When the user enters their <username>
         And the user enters their <password>
         Then the user should be successfully logged on

         Examples:
         | username | password |
         | frank    | 1234     |
         | jack     | 4321     |
```

Stichworte

Zu Dokumentationszwecken möchten Sie möglicherweise Testpläne oder Szenarien nach Kategorien filtern. Entwickler möchten möglicherweise Tests ausführen, die auf denselben Kategorien basieren. Mit Gherkin können Sie Features sowie einzelne Szenarien über den Benutzer von Tags kategorisieren. Beachten Sie im Beispiel unten, dass das Schlüsselwort Feature das Tag "@Automation" ist. Gurke erkennt dies als Tag vom Benutzer des Symbols "@". In diesem Beispiel möchte der Ingenieur klarstellen, dass diese Tests für die Automatisierung verwendet werden, wobei nicht jeder Test automatisierbar ist. Einige Tests müssen von Hand durchgeführt werden.

Beachten Sie auch, dass das Tag @Production zum Szenario-Testbenutzer gesperrt wurde. In diesem Beispiel ist dies darauf zurückzuführen, dass dieses Szenario nur in der Produktionsumgebung der Anwendung aktiv ist. Die Entwickler möchten nicht, dass ihre Sandbox-Konten während der Entwicklung gesperrt werden. Mit diesen Tags können sie erzwingen, dass dieser Test nur in der Produktionsumgebung ausgeführt wird.

Schließlich hat die Szenario-Gliederung das Tag @Staging. In diesem Beispiel liegt dies daran, dass die Konten Staging-Konten sind und in den anderen Umgebungen nicht funktionieren. Wie beim @Production-Tag wird sichergestellt, dass diese Tests nur in der Staging-Umgebung ausgeführt werden.

Dies sind nur einige Beispiele dafür, wo, wie und warum Sie Tags verwenden. Letztendlich werden diese Tags für Sie und die Entwickler eine Bedeutung haben und können alles sein, um sie zu kategorisieren, wie Sie es für richtig halten.

```
@Automation
Feature: Product Login
    As a user, I would like to be able to use my credentials to successfully
    login.

    Rules:
    - The user must have a valid username
    - The user must have a valid password
    - The user must have an active subscription
    - User is locked out after 3 invalid attempts
    - User will get a generic error message following
      login attempt with invalid credentials

    Background: The user starts out on the login page
        Given the user is on the login page

    Scenario: The user successfully logs in with valid credentials
        This scenario tests that a user is able to successfully login
        provided they enter a valid username, valid password, and
        currently have an active subscription on their account.

        When the user signs in with "valid" credentials
        Then the user should be logged in

    Scenario: The user attempts to log in with invalid credentials
        This scenario tests that a user is not able to log in when
        they enter invalid credentials

        When the user signs in with "invalid" credentials
        Then the user should be logged in

    @Production
    Scenario: The user is locked out after too many failed attempts
        This scenario validates that the user is locked out
        of their account after failing three consecutive
        attempts to log in

        When the fails to log in 3 times
        Then the user should be locked out of their account

    @Staging
    Scenario Outline: The user successfully logs in with their account
         This scenario outlines tests in which various users attempt
         to sign in successfully

         When the user enters their <username>
         And the user enters their <password>
         Then the user should be successfully logged on

         Examples:
         | username | password |
         | frank    | 1234     |
         | jack     | 4321     |
```

Gurken-Tipps

- Jedes Szenario testet ein Verhalten
- Szenarien werden deklarativ geschrieben
- Vermeiden Sie zufällige Details innerhalb des Szenarios
- Lassen Sie das Offensichtliche weg
- Vermeiden Sie konjunktive Schritte
- Halten Sie Ihre Szenarien kurz
- Sie müssen nicht viele Szenarien in derselben Funktion haben
- Verwenden Sie beschreibende Szenarionamen
- Habe nur einen Wannschritt
- Verwenden Sie das "sollte" in den Then-Schritten

Gurke-Syntax online lesen: https://riptutorial.com/de/cucumber/topic/9296/gurke-syntax

Kapitel 4: Installieren Sie das Gurken-Plugin in Intellij

Einführung

Die Cucumber-Plugins für IntelliJ IDEA bieten praktische IDE-Funktionen zum Arbeiten mit Gherkin-Feature-Dateien in einem IntelliJ-Projekt unter Verwendung des Cucumber-Frameworks. Plugins sind für Java, Scala oder Groovy verfügbar.

Bemerkungen

Das IntelliJ-Plugin Cucumber für Java bietet IDE-Funktionen für das bequeme Entwickeln mit Cucumber, einschließlich

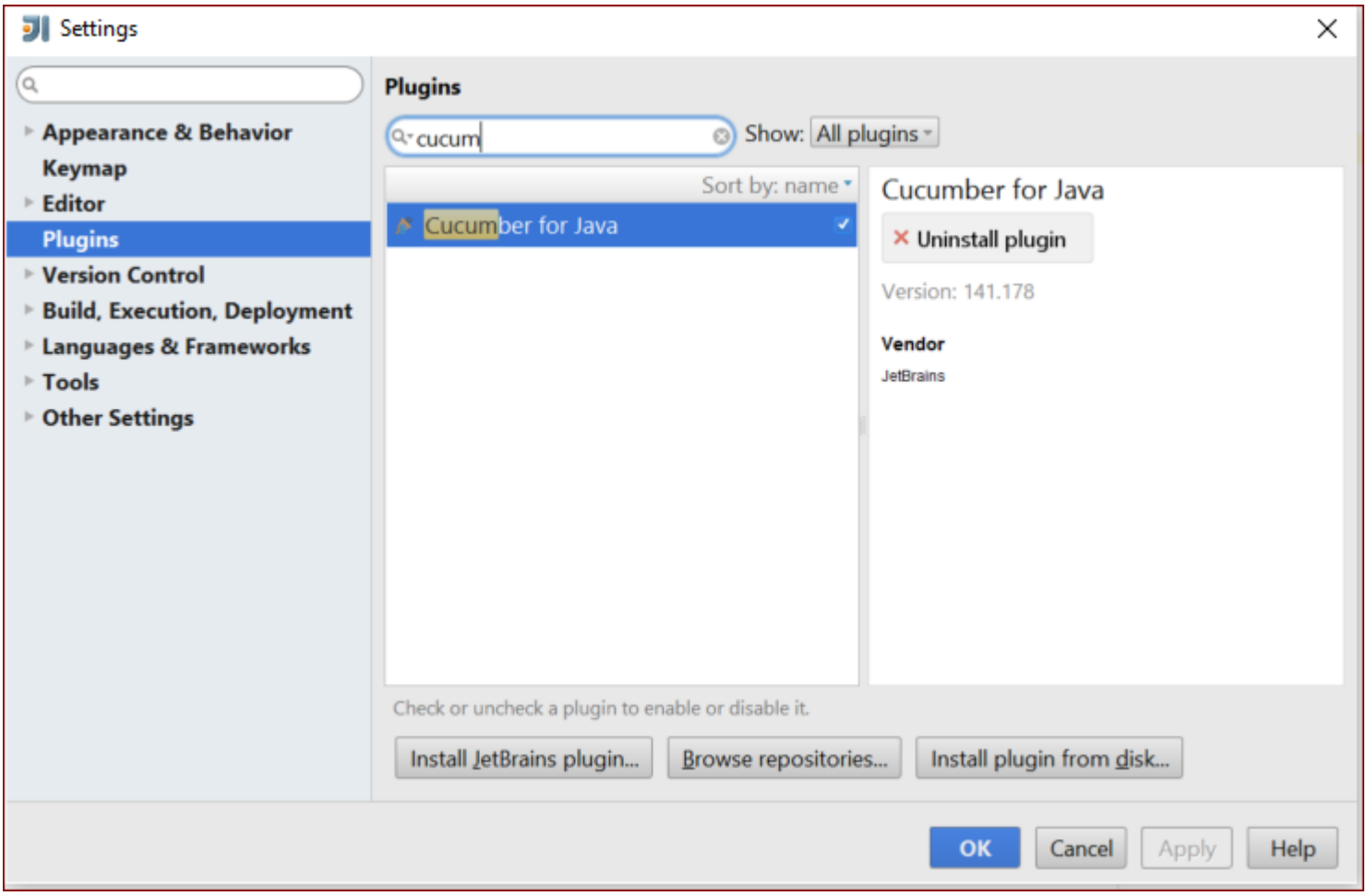
- Gewürzgurken-Schrittklebererzeugung für unimplementierte Schritte
- Gurke-Schrittcode-Vervollständigung
- Step-to-Glue-Methode Code springen
- Gurke-Syntax-Hervorhebung in ".feature" -Dateien, die mit dem Schritt regex übereinstimmen

und andere praktische Funktionen.

Examples

Installieren Sie das Gurken-Plugin

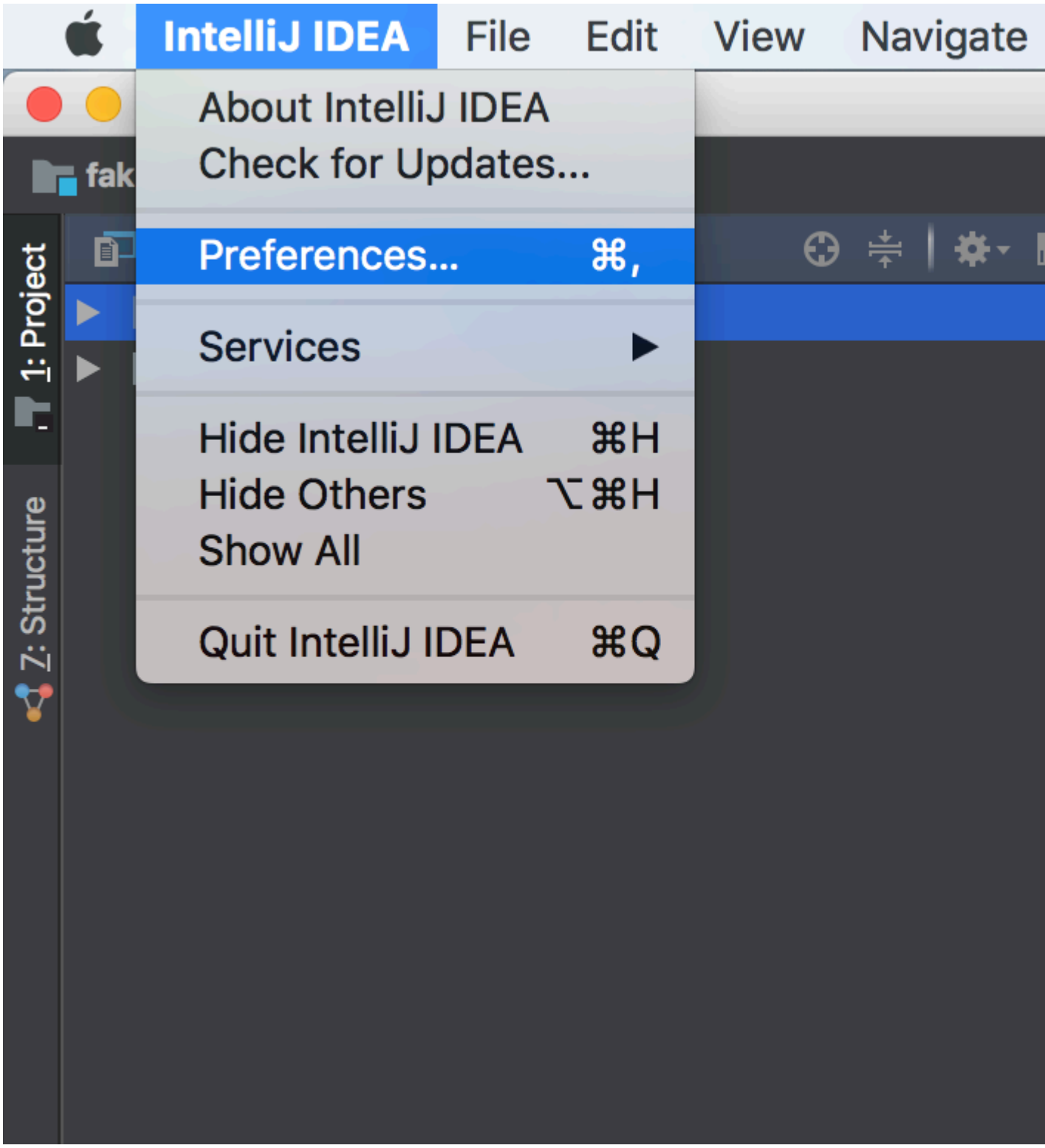
Gehen Sie zu Datei -> Einstellungen -> klicken Sie auf Plugins im linken Fensterbereich -> Suche nach Gurke -> Plugin installieren



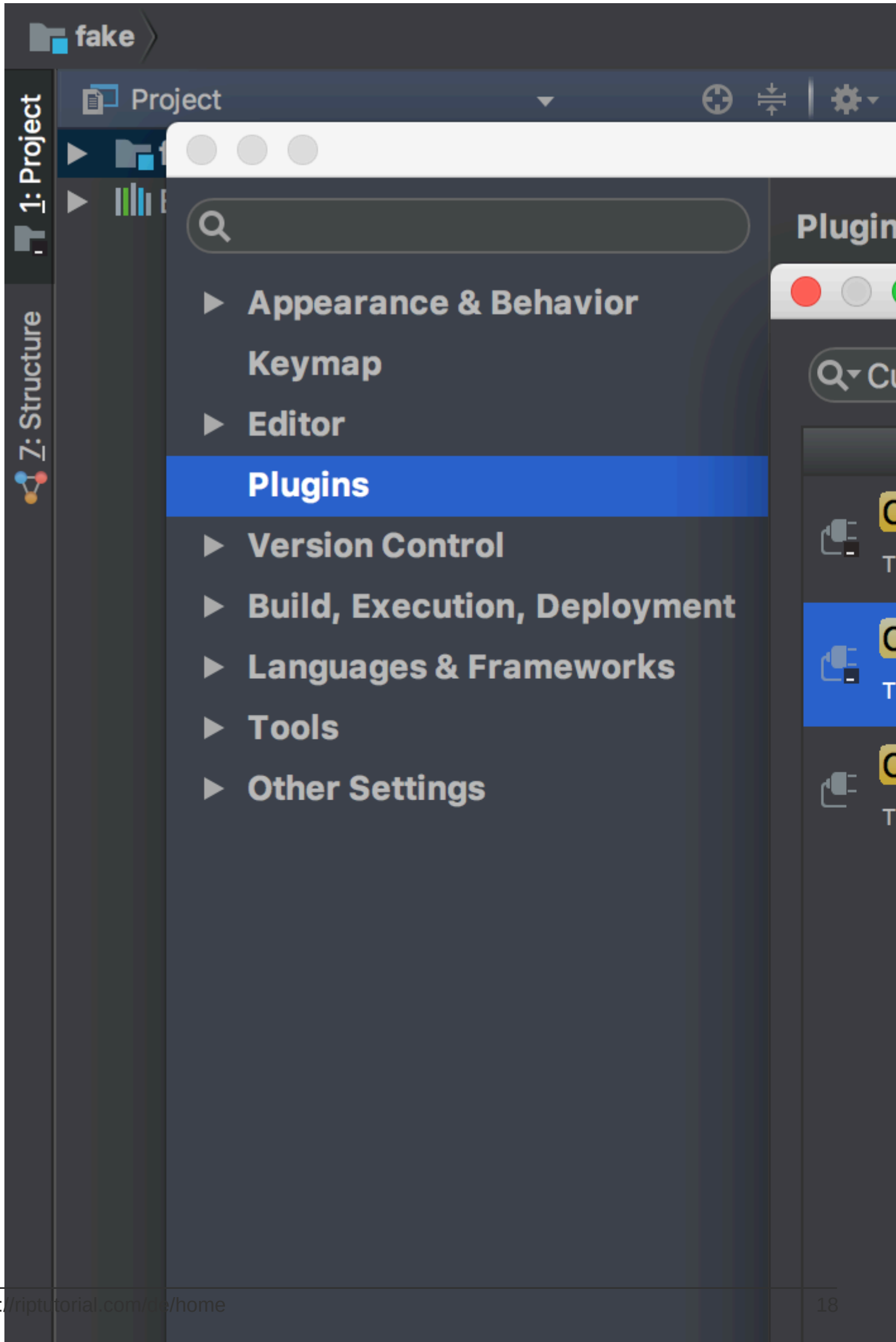
Installieren Sie IntelliJ Cucumber für Java Plugin (Mac)

So installieren Sie das Cucumber für Java-Plugin für IntelliJ auf einem Mac:

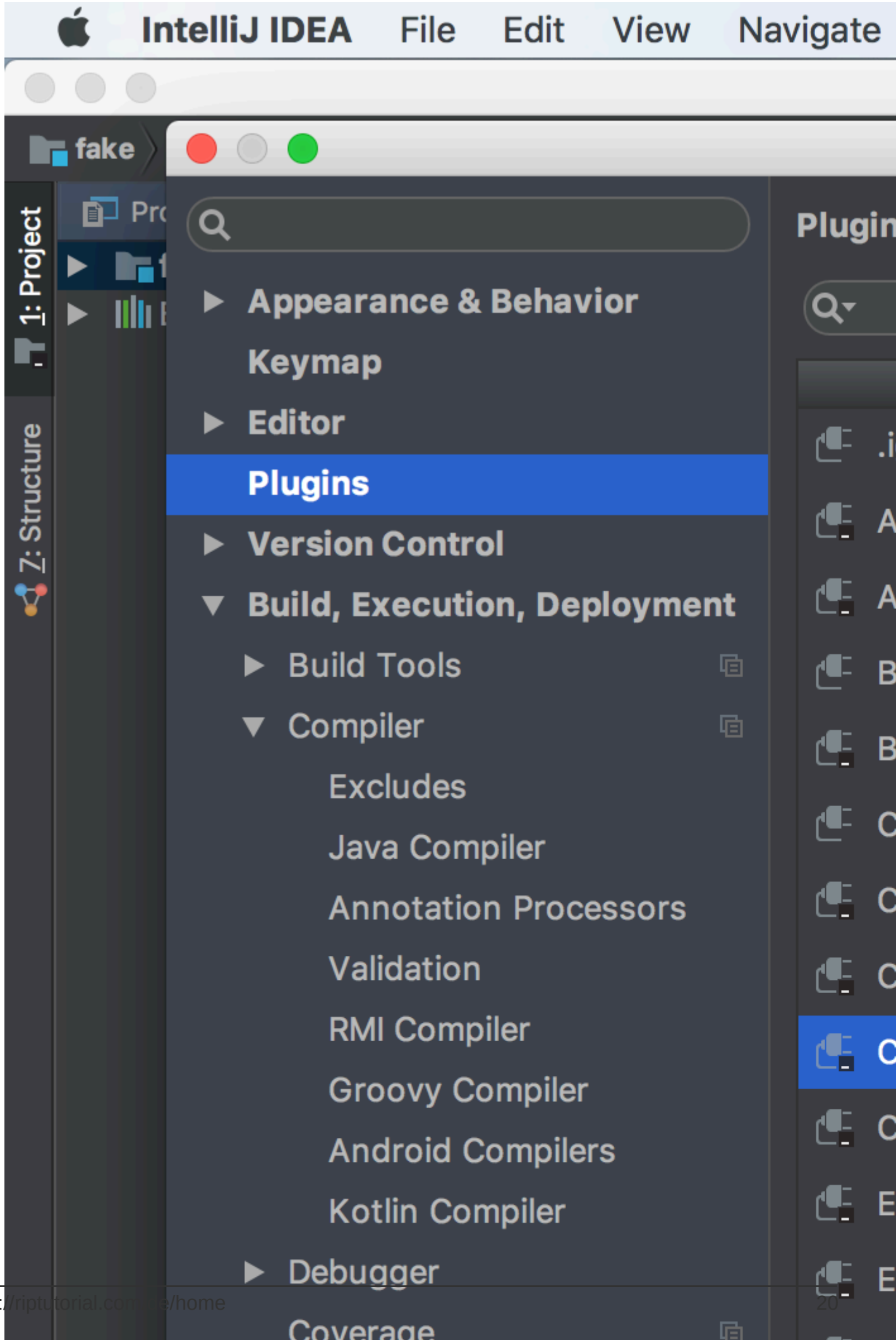
1. Starten Sie IntelliJ IDEA.
2. Klicken Sie in der oberen Leiste auf die Registerkarte "IntelliJ IDEA".



3. Klicken Sie auf "Einstellungen".
4. Klicken Sie in den Voreinstellungen / Einstellungen im linken Bereich auf "Plugins".
5. Klicken Sie auf die Schaltfläche "Repositories durchsuchen", um ein neues Fenster zu öffnen.
6. Suchen Sie in der Suchleiste nach "Gurke".


fake
Project
1: Project
Z: Structure
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Other Settings
Plugin

7. Installieren Sie das Plugin "Cucumber for Java".
8. Starten Sie die IDE neu, damit das Plugin wirksam wird. Die Gurke für Java ist jetzt installiert.


IntelliJ IDEA
File
Edit
View
Navigate
fake
1: Project
7: Structure
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Build Tools
Compiler
Excludes
Java Compiler
Annotation Processors
Validation
RMI Compiler
Groovy Compiler
Android Compilers
Kotlin Compiler
Debugger
Coverage

https://riptutorial.com/de/cucumber/topic/8356/installieren-sie-das-gurken-plugin-in-intellij

Kapitel 5: pom.xml für das Maven_ Gurkenprojekt.

Einführung

Das untenstehende Projektobjektmodell ist die Vorlage pom.xml. Wenn Sie ein Maven mit Gurkenprojekt erstellen möchten, können Sie das folgende Beispiel als Vorlage verwenden

Examples

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

4.0.0

```
<groupId>Project name</groupId>
<artifactId>MulitClients</artifactId>
<version>1.0-SNAPSHOT</version>


<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-core</artifactId>
        <version>1.2.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-testng</artifactId>
        <version>1.2.0</version>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-junit</artifactId>
        <version>1.2.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>cucumber-java</artifactId>
        <version>1.2.0</version>
        <scope>test</scope>
    </dependency>
```

```
    <dependency>
        <groupId>info.cukes</groupId>
        <artifactId>gherkin</artifactId>
        <version>2.12.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>2.53.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-firefox-driver</artifactId>
        <version>2.53.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-htmlunit-driver</artifactId>
        <version>2.53.0</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.yaml</groupId>
        <artifactId>snakeyaml</artifactId>
        <version>1.13</version>
    </dependency>
    <dependency>
        <groupId>com.esotericsoftware.yamlbeans</groupId>
        <artifactId>yamlbeans</artifactId>
        <version>1.06</version>
    </dependency>


</dependencies>
```

pom.xml für das Maven_ Gurkenprojekt. online lesen:
https://riptutorial.com/de/cucumber/topic/8331/pom-xml-fur-das-maven--gurkenprojekt-

Kapitel 6: Schrittdefinitionen

Bemerkungen

Schrittdefinitionen werden in der Programmiersprache von einer bestimmten Implementierung von Cucumber unterstützt. Dieses Thema enthält Beispiele für Schrittdefinitionen in jeder unterstützten Programmiersprache und Beispiele für die Verwendung von Cucumber-API-Aufrufen in Schrittdefinitionen.

Examples

Einige einfache Ruby-Schrittdefinitionen

In features / step_definitions / documentation.rb:

```
When /^I go to the "([^"]+)" documentation$/ do |section|
  path_part =
    case section
      when "Documentation"
        "documentation"
      else
        raise "Unknown documentation section: #{section}"
    end
  visit "/documentation/#{path_part}/topics"
end

Then /^I should see the "([^"]+) documentation"$/ do |section|
  expect(page).to have_css('h2.doctag_title a', text: section)
end
```

Diese Schritte üben eine Webanwendung aus. Sie sind so einfach wie sie sein können, während sie praktisch sind.

Jeder Schritt beginnt mit einem Gherkin-Schlüsselwort, das in einer Schrittdefinitionsdatei eine Methode ist, die einen Schritt bei Cucumber registriert. Die schrittbestimmende Methode benötigt einen regulären Ausdruck, der einer Zeile in einem Szenario entspricht, und einen Block, der ausgeführt wird, wenn das Szenario eine übereinstimmende Zeile erreicht. Erfassungsgruppen im regulären Ausdruck werden als Blockparameter an den Block übergeben.

Der Schritt " `When` enthält ein einfaches Inline-Beispiel für den Übergang von einer für Menschen lesbaren Referenz auf eine Seite ("Dokumentation") zu einer URL. Echte Gurkensuiten verwenden diese Logik normalerweise in einer separaten Methode. Die `visit` wird von Capybara bereitgestellt. Capybara ist nicht erforderlich, um Gurke zu verwenden, obwohl es häufig verwendet wird. `visit` teilt dem von Capybara kontrollierten Browser mit, die angegebene URL aufzurufen.

Der `Then` Schritt zeigt, wie der Inhalt einer Seite getestet werden kann. `expect` / `to` durch RSpec bereitgestellt (auch hier nicht von Gurken erforderlich , aber sehr häufig verwendet , um mit ihm).

`have_css` wird von Capybara zur Verfügung gestellt. Die Erwartung ist, dass der angegebene CSS-Selektor einem Element auf der Seite entspricht, das den angegebenen Text enthält. Beachten Sie, dass diese Erwartung fehlschlagen würde, wenn die Browseranforderung fehlgeschlagen war.

Schrittdefinitionen online lesen: https://riptutorial.com/de/cucumber/topic/5681/schrittdefinitionen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit Gurke	Community, Dave Schweisguth, Mo H., Roberto Lo Giacco, SirLenz0rlot, user3554664
2	Eigenschaften	Dave Schweisguth, Kyle Fairns, Priya
3	Gurke-Syntax	jordiPons, tramstheman, user3554664
4	Installieren Sie das Gurken-Plugin in Intellij	George Pantazes, Priya
5	pom.xml für das Maven_ Gurkenprojekt.	user
6	Schrittdefinitionen	Dave Schweisguth