

CYPRESS DOKUMENTATION

WAS IST CYPRESS?

Cypress ist der Newcomer im Bereich der Testautomatisierung und gewinnt immer mehr Marktanteile. Es ist ein JavaScript-basiertes Front-End-Testwerkzeug, das auch eine Open Source Version hat. Cypress verwendet das Mocha Framework, ein funktionsreiches JavaScript-Testframework, welches sowohl auf als auch im Browser läuft. Auf diese Weise werden asynchrone Tests einfach und möglich gemacht. Cypress bringt zahlreiche integrierte Features mit und bietet viele weitere Vorteile für Softwareentwickler und Testspezialisten.

Tester, Entwickler und QA-Ingenieure können mit Cypress:

- Tests einrichten
- Tests schreiben
 - **End-to-End-Tests:** ermöglichen das Testen von dem gesamten Ablauf einer Web Applikation aus der Endbenutzersicht
 - **Integration-Tests:** ermöglichen Zugriff und Überprüfung der Ergebnisse der API-Calls zwischen Web Browser und verwendeten Back-End Systemen
 - **Unit-Tests:** ermöglichen isolierte Tests der Web Oberflächen ohne Back-End durch das "Mocking von API-Calls"
- Tests durchführend debuggen
- Tests auswerten (Integriertes Reporting mit Screenshots)

FUNKTIONSWEISE UND ARCHITEKTUR

- Cypress setzt sich aus einem kostenlosen, quelloffenen, lokal installierten Test Runner und einem Dashboard Service zur Aufzeichnung der Tests zusammen.
- Die Tests laufen innerhalb von einem Browser, dadurch wird eine hohe Teststabilität gepaart mit hoher Test-Durchführungsgeschwindigkeit ermöglicht.
- Es werden automatische Aufzeichnungen mithilfe von SUT Snapshots angelegt, die für die Fehleranalyse bequem zur Hilfe gezogen werden können.
- Darauf folgend können aufgetretenen Fehler leicht durch Werkzeug wie Developer Tools debuggt werden.
- Cypress wartet automatisch auf Befehle, sodass das Einfügen von Sleeps und Waits nicht nötig ist.
- Das Netzwerkverkehr kann beliebig gesteuert und gemockt werden, so dass ein u. U. noch nicht implementierter Backend-Service u. U. simuliert werden kann (Mocks und Stubs)
- Die gesamte Testpalette wird einfacher, schneller, fehlerfreier und allgemein zuverlässiger ausgeführt, was einen großen Vorteil gegenüber Selenium darstellt.

VOR UND NACHTEILE

Vorteile

- Unabhängig von Selenium WebDriver einsetzbar
- Einfache Einrichtung und Bedienung
- Hohe Stabilität und Test-Durchführungsgeschwindigkeit
- Keine expliziten Sleeps und Waits benötigt
- Einfaches Debugging und Analyse der Fehler
- Unterstützung von API Mocking und Zugriffe auf HTTP Kommunikation

Nachteile

- Cross-Browser Tests nur eingeschränkt möglich
- Multi-Tab und Multi-Session Tests nicht möglich
- Parallele Tests werden nur über das Cypress Dashboard verwaltet
- Fehlende Möglichkeiten des expliziten Wartens in komplexen Use Cases problematisch

CYPRESS VS. SELENIUM

Ein weiteres Testwerkzeug, das oft mit Cypress in Verbindung gebracht wird, ist [Selenium](#).

Beide Tools sind Automatisierungsframeworks, die für Testen von Webanwendungen geeignet sind. Während Selenium eine seit über 10 Jahren auf dem Markt etablierte Testlösung ist, gilt Cypress als Newcomer, welches sich aber rasant entwickelt. Die zwei Technologien unterscheiden sich des Weiteren grundlegend im Hinblick auf ihre Infrastruktur, Features und Beschränkungen. Unten finden Sie einen kurzen Überblick über die Schlüsselunterschiede beider Technologien:

Selenium

- Automatisierung von Cross-Browser Web Testsinkl. Interaktion mit Web Browsern von Außen
- Language Bindingsfür so gut wie alle Programmier-Sprachenverfügbar wie z.B.:Java, JavaScript, C#, Pythonetc.
- Befehle können von der Testumgebung am gewünschtenBrowser auf einer lokalen oder entfernten Umgebung wie Chrome, Firefox, Edge oder Safari gesendet werden
- Vollkommen quelloffen, verwaltet durch Open SourceFoundation
- Nutzung von Cross-Browser Cloud Services wie[BrowserStack](#)und[SauceLabs](#)möglich
- Grundlage für Testframeworks ([Protractor](#),[WebDriverIO](#)etc.) und mobile Anwendungen ([Appium](#))

Cypress

- Automatisierung von Webanwendungenaus dem Browser heraus
- Sprache:nur JavaScript
- Skripte für Web-Testautomatisierung in der de-facto Web-Sprache JavaScript
- Testskriptewerden zum Web Browser lokal übertragen und können von dort aus höchst effizient ausgeführt werden und mit der aktiven Web Seite per JavaScript interagieren.
- Unterstützung von [Mocha](#)–Testframeworkmit Mocha-Syntax

NPM

Um die Dateien und verschiedene Plug-Ins herunterzuladen, die wir für Cypress benötigen, brauchen wir npm-Befehle, die wir in das Cypress-Terminal schreiben müssen.

Npm **steht für Node** Package Manager . Es ist ein Paketmanager für die Node-JavaScript-Plattform.

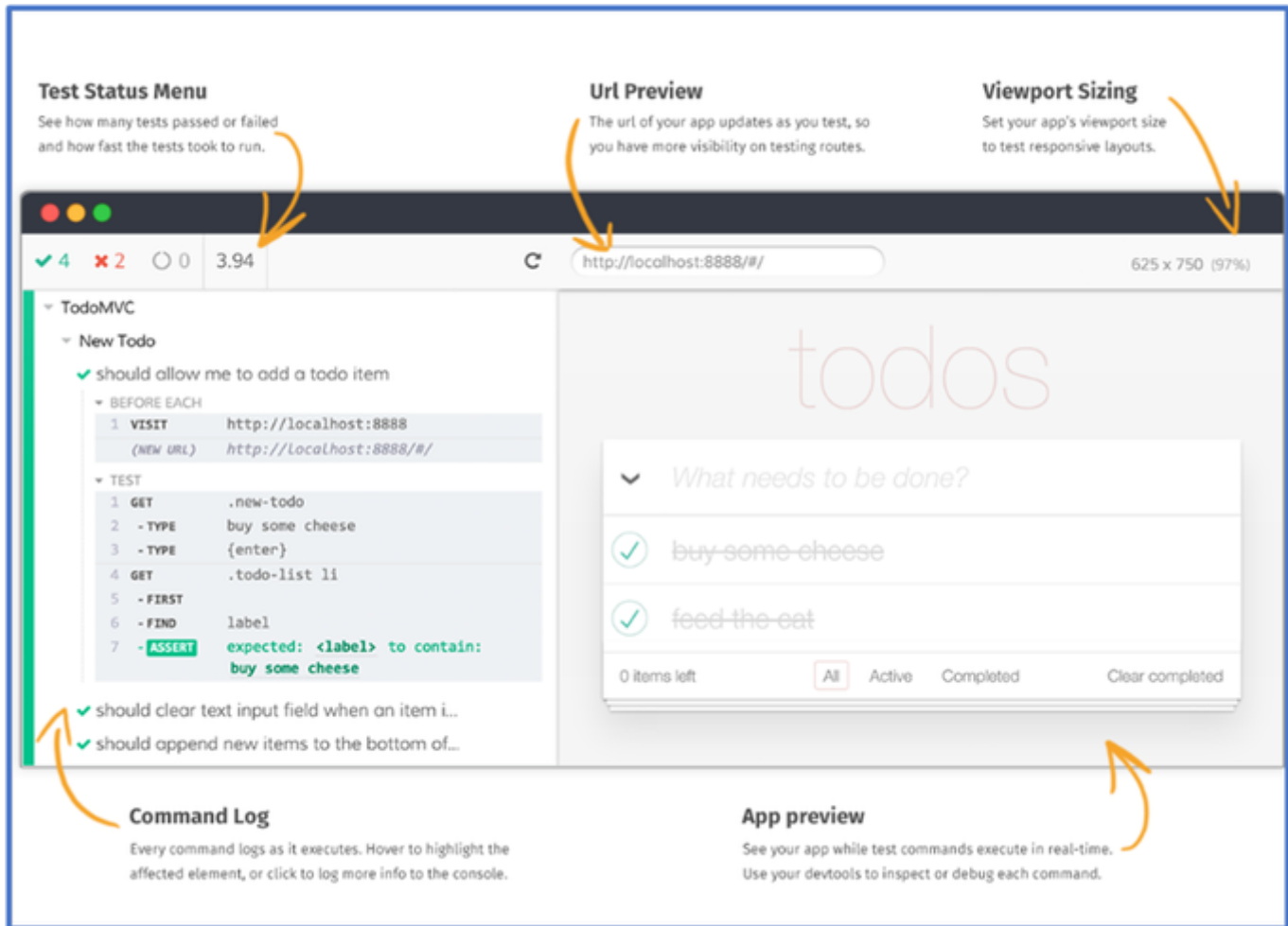
Npm ist als die weltweit größte Softwareregistrierung bekannt. Open-Source-Entwickler auf der ganzen Welt verwenden npm, um ihren Quellcode zu veröffentlichen und zu teilen.

Npm besteht aus drei Komponenten:

- [Auf der Website](#) können Sie Pakete von Drittanbietern finden, Profile einrichten und Ihre Pakete verwalten.
- Die Befehlszeilenschnittstelle oder npm-CLI, die von einem Terminal ausgeführt wird, um Ihnen die Interaktion mit npm zu ermöglichen.
- Die Registrierung ist eine große öffentliche Datenbank mit JavaScript-Code.

CYPRESS TEST RUNNER

- Cypress führt Tests in einem einzigartigen interaktiven Runner durch, der es Ihnen ermöglicht, Befehle während der Ausführung zu sehen und gleichzeitig die zu testende Anwendung anzuzeigen.



- Die linke Seite des Test Runners ist eine visuelle Darstellung Ihrer Testsuite. Jeder Testblock ist richtig verschachtelt und jeder Test zeigt beim Anklicken jeden Cypress-Befehl.
- Jeder Befehl, jede Behauptung oder jeder Fehler zeigt beim Anklicken zusätzliche Informationen in der Entwicklertools-Konsole an. Durch Klicken auf wird auch die zu testende Anwendung (rechte Seite) in ihrem vorherigen Zustand „angeheftet“, als der Befehl ausgeführt wurde.
- **Fehlernamen** : Dies ist die Art des Fehlers (z. B. AssertionError, CypressError)

Fehlermeldung : Dies sagt Ihnen im Allgemeinen, was schief gelaufen ist. Es kann in der Länge variieren. Einige sind kurz wie im Beispiel, während andere lang sind und Ihnen möglicherweise genau sagen, wie Sie den Fehler beheben können.

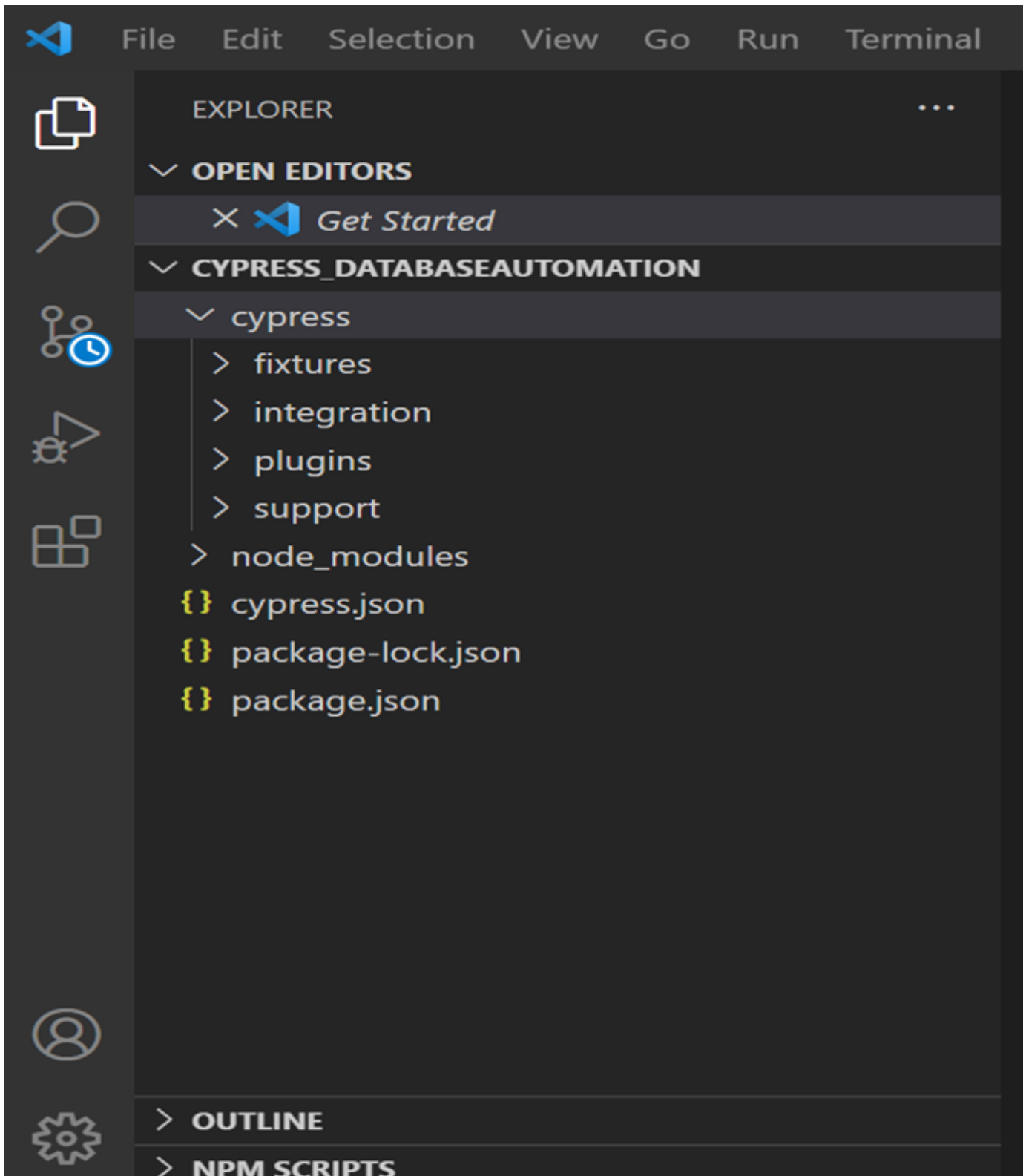
- **Weitere Informationen**: Einige Fehlermeldungen enthalten einen Link Weitere Informationen, der Sie zur relevanten Cypress-Dokumentation führt.
- **Coderaahmen** : Dies zeigt einen Codeausschnitt, an dem der Fehler aufgetreten ist, wobei die relevante Zeile und Spalte hervorgehoben sind.
- **Schaltfläche**: „Auf Konsole drucken“ : Klicken Sie hierauf, um den vollständigen Fehler in Ihrer DevTools-Konsole zu drucken. Dadurch können Sie normalerweise auf Zeilen im Stack-Trace klicken und Dateien in Ihren DevTools öffnen.

ERSTEN TEST MIT CYPRESS

- Angenommen, Sie haben den Test Runner erfolgreich installiert und die Cypress-App geöffnet.
- Erstellen Sie eine `sample_spec.js`-Datei.
- Starten Sie den Cypress Test Runner.
- Sobald wir die Datei erstellt haben, sollte Cypress Test Runner sie sofort in der Liste der Integrationstests anzeigen.
- Cypress überwacht Ihre Spezifikationsdateien auf Änderungen und zeigt automatisch alle Änderungen an.

ORDNERSTRUKTUR

Nach dem Hinzufügen eines neuen Projekts erstellt Cypress automatisch eine vorgeschlagene Ordnerstruktur. Standardmäßig wird Folgendes erstellt.



Obwohl Sie mit Cypress konfigurieren können, wo sich Ihre Tests, Fixtures und Support-Dateien befinden, empfehlen wir Ihnen, die obige Struktur zu verwenden, wenn Sie Ihr erstes Projekt starten.

- **Testdateien**

Cypress unterstützt auch ES2015 Out-of-the-Box. Sie können entweder ES2015 modules oder verwenden CommonJS modules. Dies bedeutet, dass Sie sowohl npm-Pakete als auch lokale relative Module verwenden können `.require`.

- **Fixture-Dateien**

Vorrichtungen werden als externe statische Daten verwendet, die von Ihren Tests verwendet werden können. Fixture-Dateien befinden sich cypress/fixturesstandardmäßig in, können aber in einem anderen Verzeichnis konfiguriert werden.

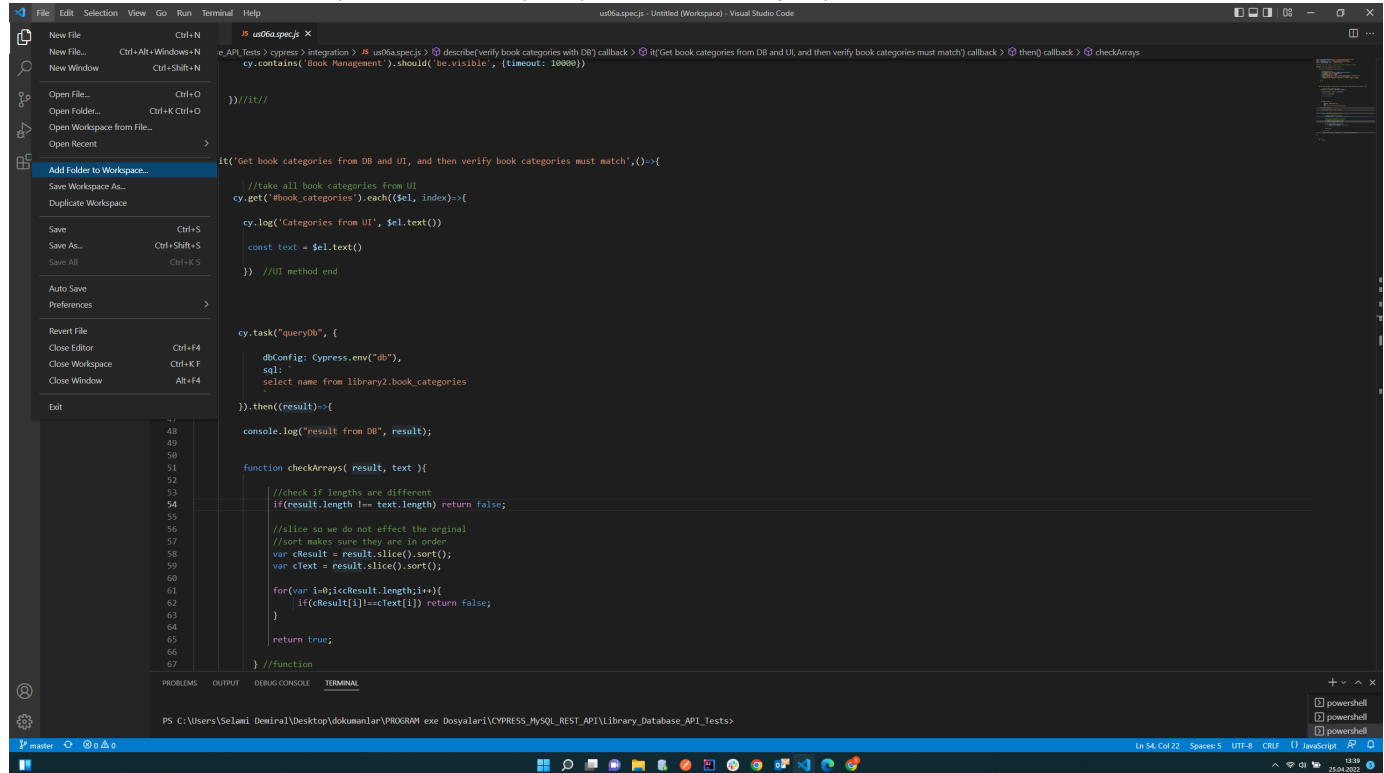
- **Plugins-Datei**

Die Plug-in-Datei ist eine spezielle Datei, die in Node ausgeführt wird, bevor das Projekt geladen wird, bevor der Browser gestartet wird und während Ihrer Testausführung. Während die Cypress-Tests im Browser ausgeführt werden, wird die Plug-in-Datei im Node-Hintergrundprozess ausgeführt, sodass Ihre Tests durch Aufrufen des Befehls `cy.task()` auf das Dateisystem und den Rest des Betriebssystems zugreifen können.

CYPRESS INSTALLIEREN AUF IDE (Visual Studio Code oder IntelliJ)

Erstellen Sie ein leeres Verzeichnis mit dem Project_name auf Desktop usw..

Wählen dieses Verzeichnis im IDE (Visual Studio Code) aus (Add Folder to Workspace).

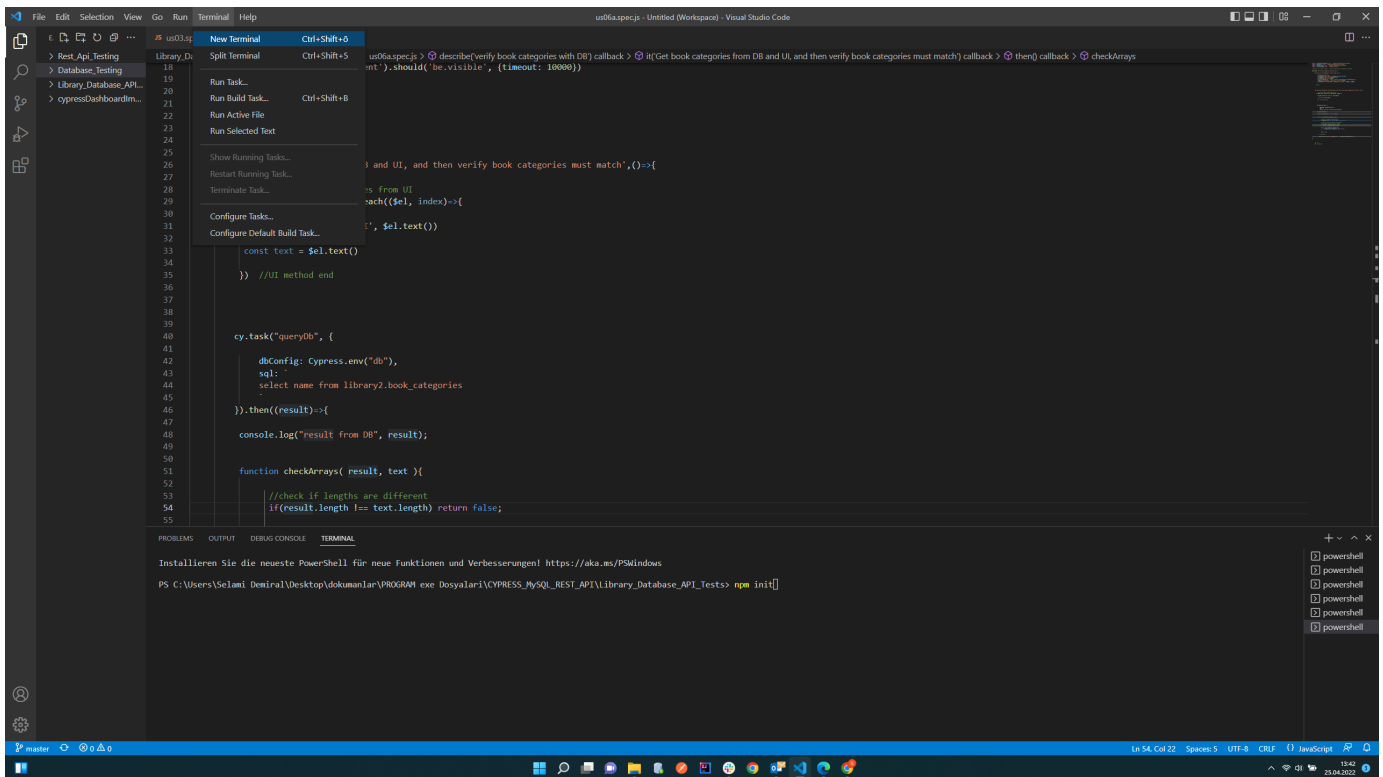


Öffnen neuer Terminal, um die "package.json" datei im Framework zu erstellen ;

```
# npm init
```

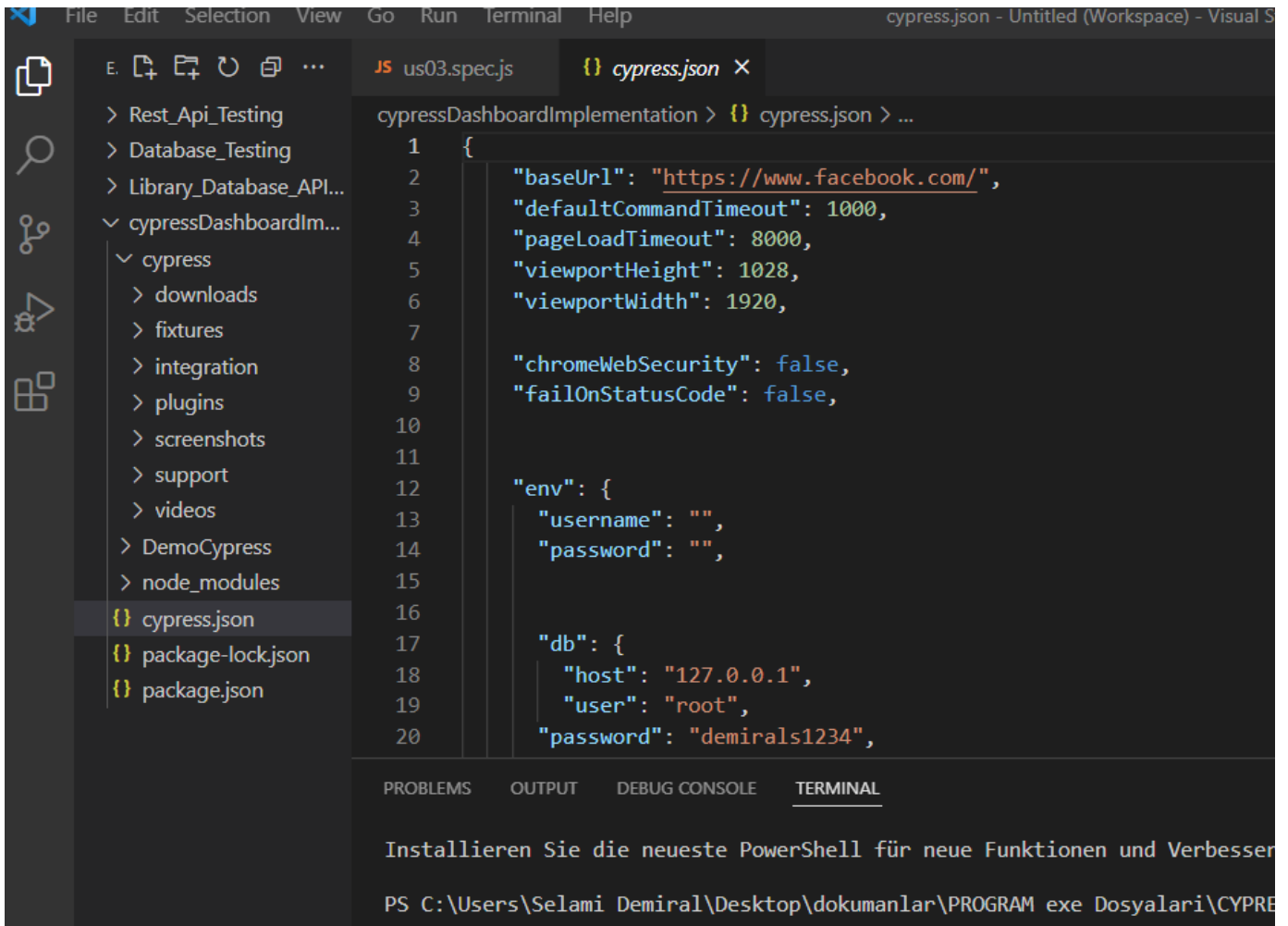
Zur Erstellung restlicher Pakete ;

```
# npm install cypress --save-dev  
  
# npx cypress open
```



Nutzung_Beispiel für baseUrl : `cy.visit('/')` in "cypress.json" Datei :

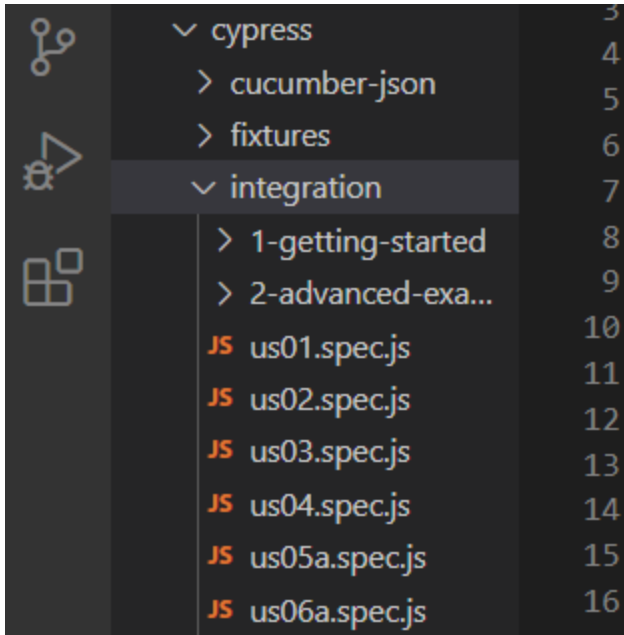
```
{
  "baseUrl": "https://...",
  "password": "12345"
}
```



Erstellen Sie neue Folder und Datei.spec.js unter Integration Pakete (für noncucumber Framework)

Treten Sie in die Folder von der Datei.spec.js und um die datei zu laufen im CLI Modus;

```
# node datei.spec.js
```



CUCUMBER FRAMEWORK *

(*) Wir empfehlen, das Cucumber Framework nicht zu verwenden, da wir festgestellt haben, dass der Austausch von Parametern und Konstanten zwischen GIVEN-AND-THEN nicht korrekt erfolgt. Darüber hinaus bietet das Cucumber-Framework nicht den Komfort von Selenium Cucumber während der Erstellung von Schrittdefinitionen. Darüber hinaus enthält Cucumber Reporting auch nicht viele Details. Stattdessen wäre die Verwendung von Cypress Dashboard eine viel bessere Lösung.

Tutorial Link : <https://testersdock.com/cypress-cucumber-bdd/>

1. Installieren Sie das cypress-cucumber-preprocessor-Plugin (Sehe Plugin in package.json datei).

npm install --save-dev cypress-cucumber-preprocessor

```

us03.spec.js  package.json X
Database_Testing > package.json > {} scripts
18 | "test:echo": "echo \"Error: no test specified\" && exit 1"
19 | },
20 | "author": "demirals",
21 | "license": "ISC",
22 | "devDependencies": {
23 |   "cucumber-html-reporter": "^5.5.0",
24 |   "cypress": "^9.5.4",
25 |   "cypress-cucumber-preprocessor": "^4.3.1",
26 |   "multiple-cucumber-html-reporter": "^1.19.0",
27 |   "mysql": "^2.18.1"
28 | },
29 | "cypress-cucumber-preprocessor": {
30 |   "nonGlobalStepDefinitions": true,
31 |   "cucumberJson": {
32 |     "generate": true,
33 |     "outputFolder": "cypress/reports/cucumber-json",
34 |     "filePrefix": "",
35 |     "fileSuffix": ".cucumber"
36 |   }
37 | }

```

2. Schreiben Sie unten genannten Text in Ihre package.json. Damit soll sichergestellt werden, dass cypress-cucumber-preprocessor keine globalen Schrittdefinitionen verwendet.

```

"cypress-cucumber-preprocessor": {
  "nonGlobalStepDefinitions": true
}

```

3. Gehen Sie zu cypress/plugins/index.js und schreiben Sie:


```
const cucumber = require('cypress-cucumber-preprocessor').default
module.exports = (on, config) => {
  on('file:preprocessor', cucumber())
}
```

```
Database_Testing > cypress > plugins > JS index.js > ...
40
41   })
42   })
43 }
44
45
46 const cucumber = require('cypress-cucumber-preprocessor').default
47
48
49 module.exports = (on, config) => {
50   on('task', { queryDb: query => { return queryTestDb(query, config) }, }), //For running sql query
51   on('file:preprocessor', cucumber());
52 }
53
54
```

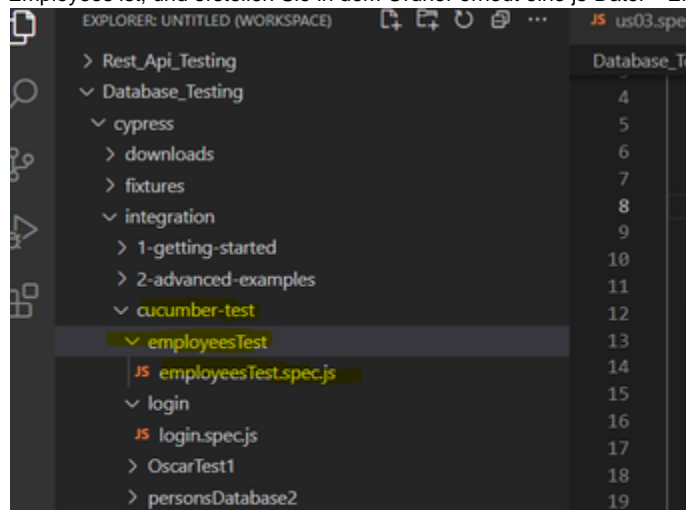
4. Gehen Sie zu cypress.json und fügen Sie unten hinzu. Dies dient dazu, unserer Cypress-Konfiguration Unterstützung für Feature-Dateien hinzuzufügen.

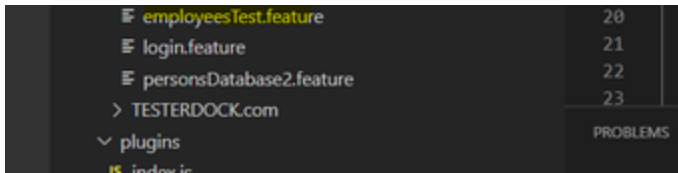
```
"testFiles": "**/*.feature"
```

```
Database_Testing > {} cypress.json > ...
4   "defaultCommandTimeout": 1000,
5   "pageLoadTimeout": 8000,
6   "viewportHeight": 1028,
7   "viewportWidth": 1920,
8
9   "chromeWebSecurity": false,
10  "failOnStatusCode": false,
11  "testFiles": "**/*.feature",
12
13
```

5. Als nächstes schreiben Sie unseren Cucumber-Test im Gherkin-Format (Given When Then). Erstellen Sie dazu einen Ordner namens cucumber-test im Ordner cypress/integration und erstellen Sie dann eine Datei namens login.feature.

6. Der nächste Schritt besteht darin, eine Schrittdefinitionsdatei zu erstellen, die jeden Schritt der Feature-Datei in Aktionen übersetzt, die Cypress ausführt. Erstellen Sie dazu in cucumber-tests einen Ordner mit dem gleichen Namen wie die Feature-Datei, die in unserem Fall Employees ist, und erstellen Sie in dem Ordner erneut eine js-Datei – EmployeesTest.spec.js.



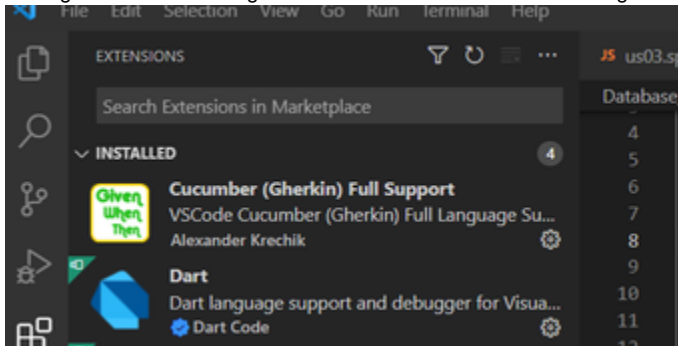


7. Executing the tests via Test Runner oder Executing the tests via **CLI** using the command:

npx cypress run --spec cypress/integration/cucumber-test/EmployeesTest.feature //For single feature file
oder

npx cypress run --spec cypress/integration/cucumber-test/*.feature //For all feature files

8. Vergessen Sie bei farbigen Feature-Sätzen nicht das erste Plugin im Marketplace (given when then)



Tutorial Link : https://www.youtube.com/watch?v=qupybITFqd8&list=PLzDWIPKHyNmK9NX9_ng2ldrEr8L4WwB0&index=14

CUCUMBER HTML REPORTING

Tutorial Link : <https://blog.knoldus.com/generating-cucumber-html-report-cypress/>

1. Da wir das Cypress-Cucumber-Preprocessor-Plugin bereits in unserer package.json haben, müssen wir es jetzt nur noch ein wenig modifizieren. Wir müssen diesem Plugin Folgendes hinzufügen.

```
"cypress-cucumber-preprocessor": {
  "nonGlobalStepDefinitions": true,
  "cucumberJson": {
    "generate": true,
    "outputFolder": "cypress/cucumber-json",
    "filePrefix": "",
    "fileSuffix": ".cucumber"
  }
}
```

WICHTIG: In package.json datei in "scripts" >> Ändern Sie den Satz "test": "echo "Error: no test specified" && exit 1", wie unten ;

"test:echo": "echo "Error: no test specified" && exit 1",

und fügen Sie die Sätze unten hin;

```
"test:open": "cypress open",
"test:qabox": "npx cypress-tags run --env \"TAGS=@qabox\"",
"test:tn1NOTtn2": "npx cypress-tags run --env \"TAGS=@tagname1 and not @tagname2\"",
"test:tn1ANDtn2": "npx cypress-tags run --env \"TAGS=@tagname1 AND @tagname2\"",
"test:tn1ORTn2": "npx cypress-tags run --env \"TAGS=@tagname1 OR @tagname2\"",
"clean:reports": "if exist cypress\\reports rmdir /S/Q"
```

```

cypress\\reports",
"pretest": "npm run clean:reports",
"scripts": "cypress run --browser chrome",
"posttest": "node cucumber-html-report.js",
"test": "npm run scripts || npm run posttest",

```

```

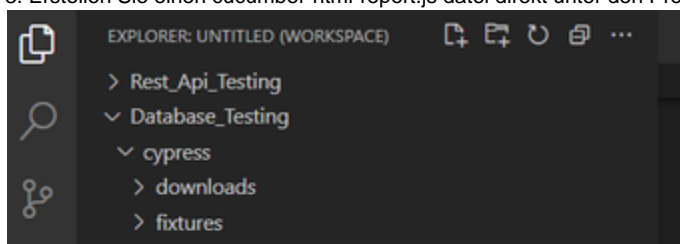
package.json - Untitled (workspace) - Visual Studio Code
us03.spec.js  {} package.json x
Database_Testing > {} package.json > {} scripts
4  "description": "",
5  "main": "index.js",
   ▶ Debug
6  "scripts": {
7    "test:open": "cypress open",
8    "test:qabox": "npx cypress-tags run --env \"TAGS=@qabox\"",
9    "test:tn1NOTtn2": "npx cypress-tags run --env \"TAGS=@tagname1 and not @tagname2\"",
10   "test:tn1ANDtn2": "npx cypress-tags run --env \"TAGS=@tagname1 AND @tagname2\"",
11   "test:tn1ORTn2": "npx cypress-tags run --env \"TAGS=@tagname1 OR @tagname2\"",
12   "clean:reports": "if exist cypress\\reports rmdir /S/Q cypress\\reports",
13   "pretest": "npm run clean:reports",
14   "scripts": "cypress run --browser chrome",
15   "posttest": "node cucumber-html-report.js",
16   "test": "npm run scripts || npm run posttest",
17
18   "test:echo": "echo \"Error: no test specified\" && exit 1"
19 },
20 "author": "demirals",
21 "license": "ISC",
22 "devDependencies": {
23   "cucumber-html-reporter": "^5.5.0",
24   "cypress": "^9.5.4",
25   "cypress-cucumber-preprocessor": "^4.3.1",
26   "multiple-cucumber-html-reporter": "^1.19.0",
27   "mysql": "^2.18.1"
28 },
29 "cypress-cucumber-preprocessor": {
30   "nonGlobalStepDefinitions": true,
31   "cucumberJson": {
32     "generate": true,
33     "outputFolder": "cypress/reports/cucumber-json",
34     "filePrefix": "",
35     "fileSuffix": ".cucumber"
36   }
37 }
38 }
39

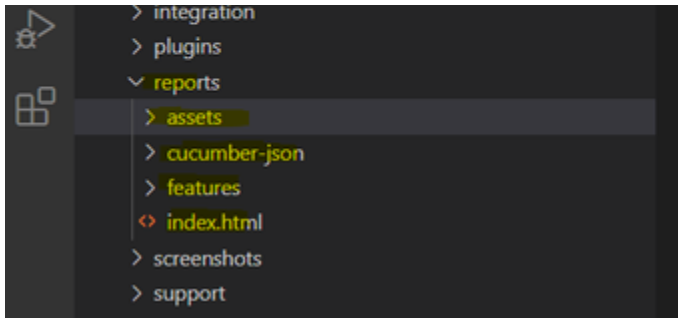
```

2. Installieren Sie dieses unten erwähnte Plugin in Ihrem Projekt

```
# npm install multiple-cucumber-html-reporter --save-dev
```

3. Erstellen Sie einen cucumber-html-report.js datei direkt unter den Projekt neben .json dateien.



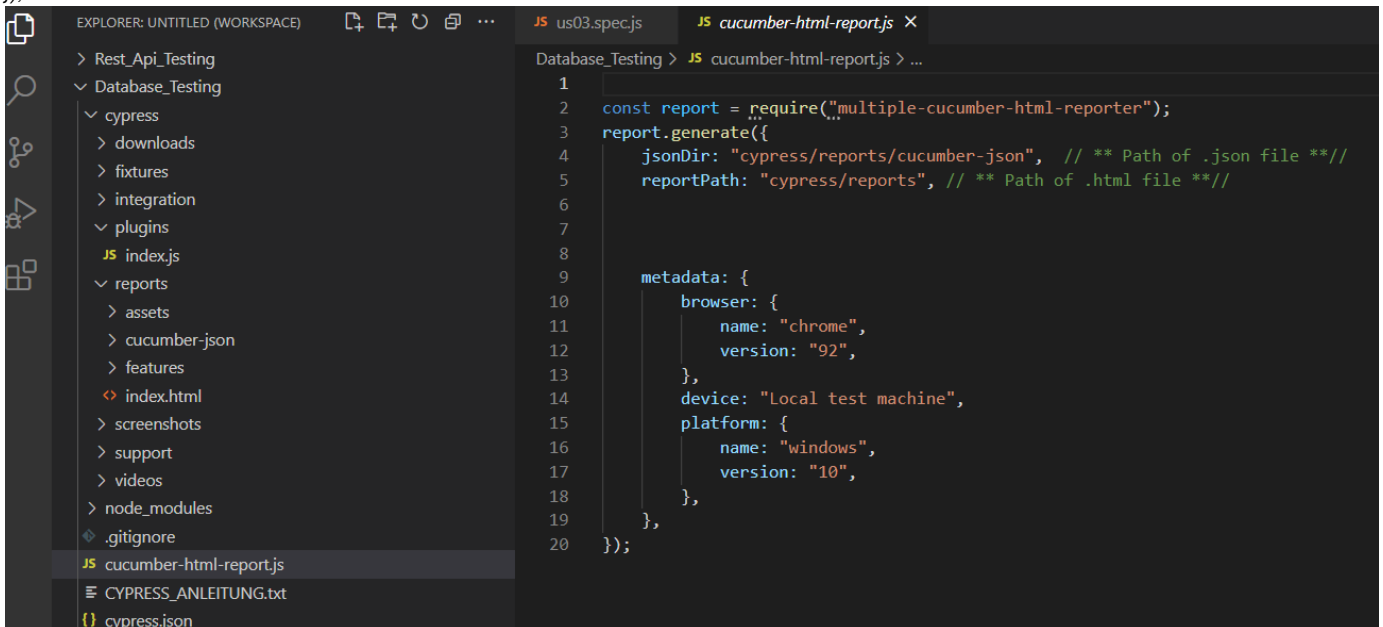


4. Fügen Sie die Sätze cucumber-html-report.js Datei hin;

```
const report = require("multiple-cucumber-html-reporter");
report.generate({
  jsonDir: "cypress/reports/cucumber-json", // ** Path of .json file **//
  reportPath: "cypress/reports", // ** Path of .html file **//
```

```
    metadata: { browser: { name: "chrome", version: "92", }, device: "Local test machine", platform: { name: "windows", version: "10", }, },
```

```
});
```

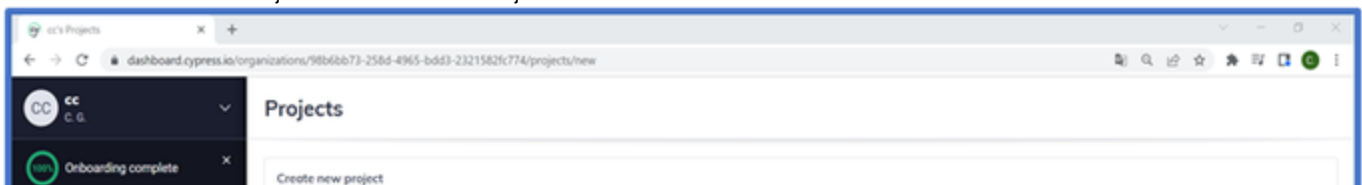


4. Um nun den HTML-Bericht zu generieren, führen Sie einfach diese .js-Datei aus.

```
# node cucumber-html-report.js
```

DASHBOARD ERSTELLUNG

1. Alle Dateien müssen in GitHub sein (git add . / git commit -m "" / git push)
2. Navigieren zu Webseite <http://dashboard.cypress.io> und login mit dem Google Konto* oder GitHub
3. Erstellen Sie ein neues Projekt in Dashboard mit Projektname



4. Fügen Sie Projekt_ID in cypress.json Datei hin

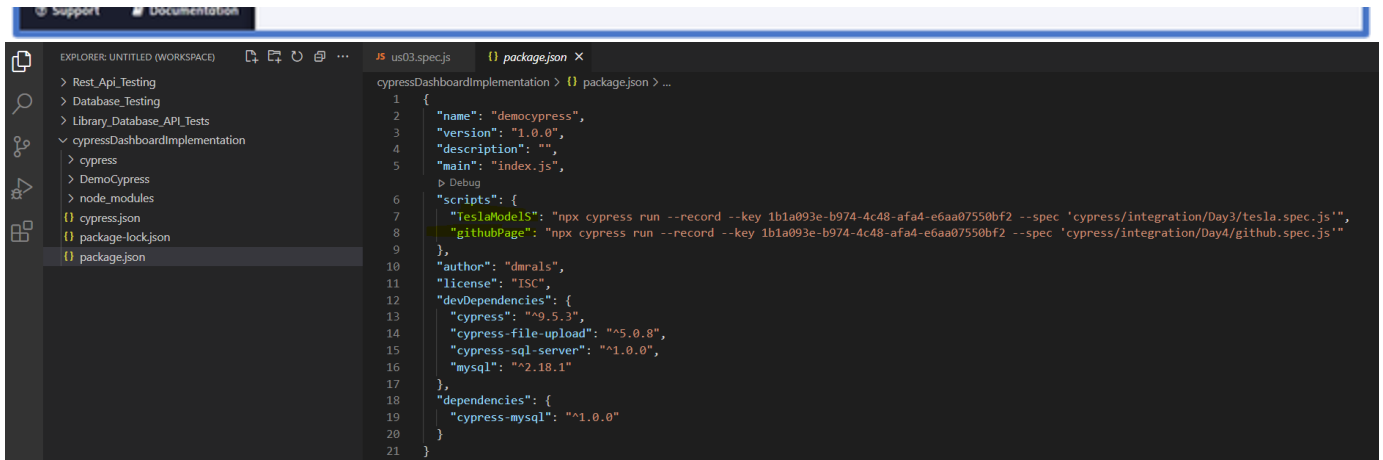
```
{
  "projectId": "z8tnphk"
}
```

5. Fügen Sie den Record Key and die Phath von spec. datei (ab cypress/....) in Datei Package.json in "scripts" :

```
"scripts": {
```

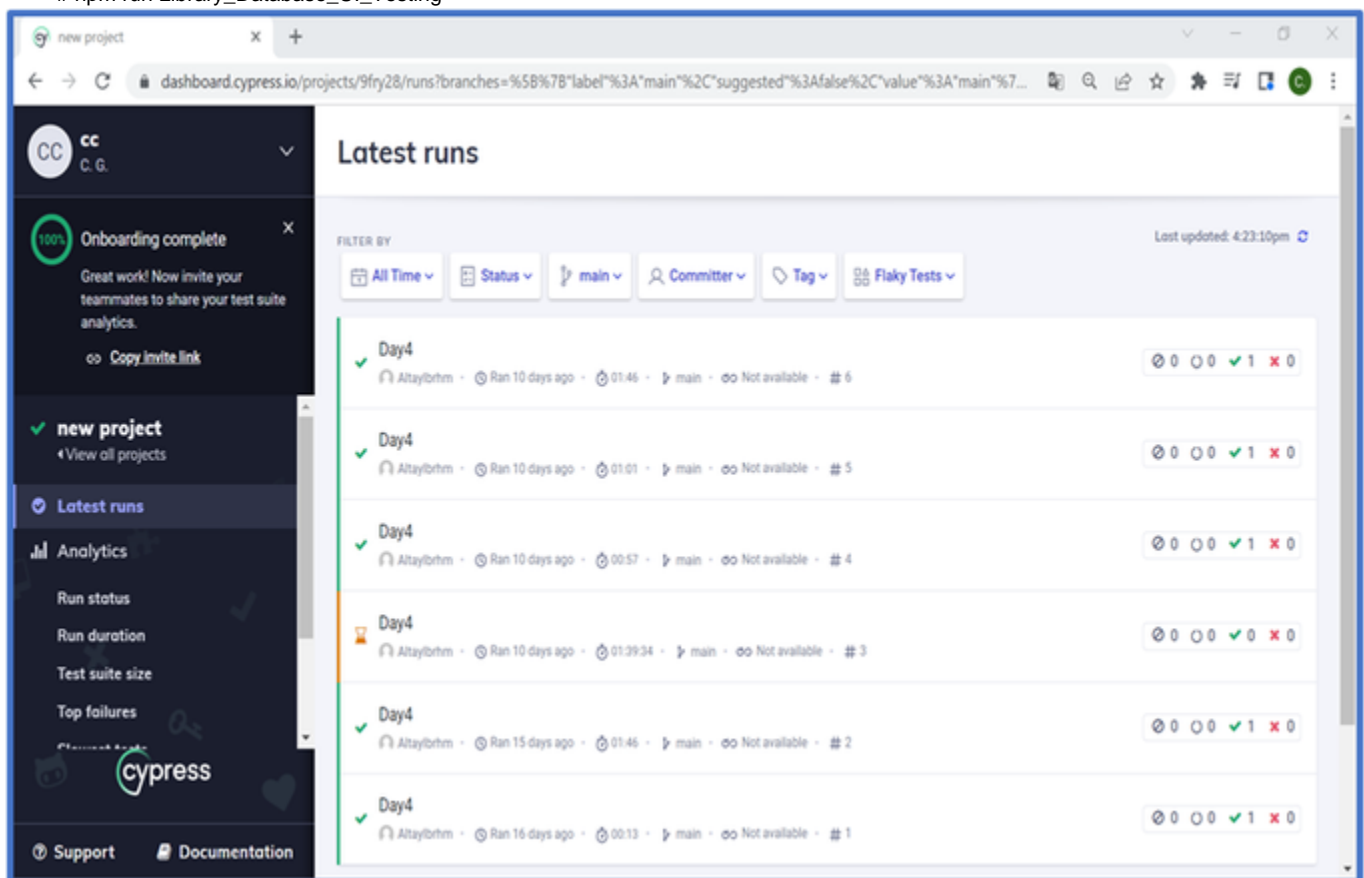
```
"Library_Database_UI_Testing": "npx cypress run --record --key 2d92a570-1258-4436-111 --spec 'cypress/integration/cucumber-test/us01/us01.spec.js'"
```

```
}
```



```
1 {
2   "name": "democypress",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "TeslaModelS": "npx cypress run --record --key 1b1a093e-b974-4c48-afa4-e6aa07550bf2 --spec 'cypress/integration/Day3/tesla.spec.js'",
8     "githubPage": "npx cypress run --record --key 1b1a093e-b974-4c48-afa4-e6aa07550bf2 --spec 'cypress/integration/Day4/github.spec.js'",
9   },
10  "author": "dmrals",
11  "license": "ISC",
12  "devDependencies": {
13    "cypress": "^9.5.3",
14    "cypress-file-upload": "^5.0.8",
15    "cypress-sql-server": "^1.0.0",
16    "mysql": "^2.18.1"
17  },
18  "dependencies": {
19    "cypress-mysql": "^1.0.0"
20  }
21 }
```

6. Laufen Sie den Code mit der Projektname in packace.json datei und sehen Sie Testergebnisse in <http://dashboard.cypress.io> webseite
npm run Library_Database_UI_Testing



DATENBANK TEST CASE UND KONSTRUKTUR BEISPIEL

Tutorial Link : <https://testersdock.com/cypress-database-testing/>

1. Installieren Sie das MySQL-Plugin. Sobald es installiert ist, sollte es sich unter Ihrer package.json widerspiegeln.

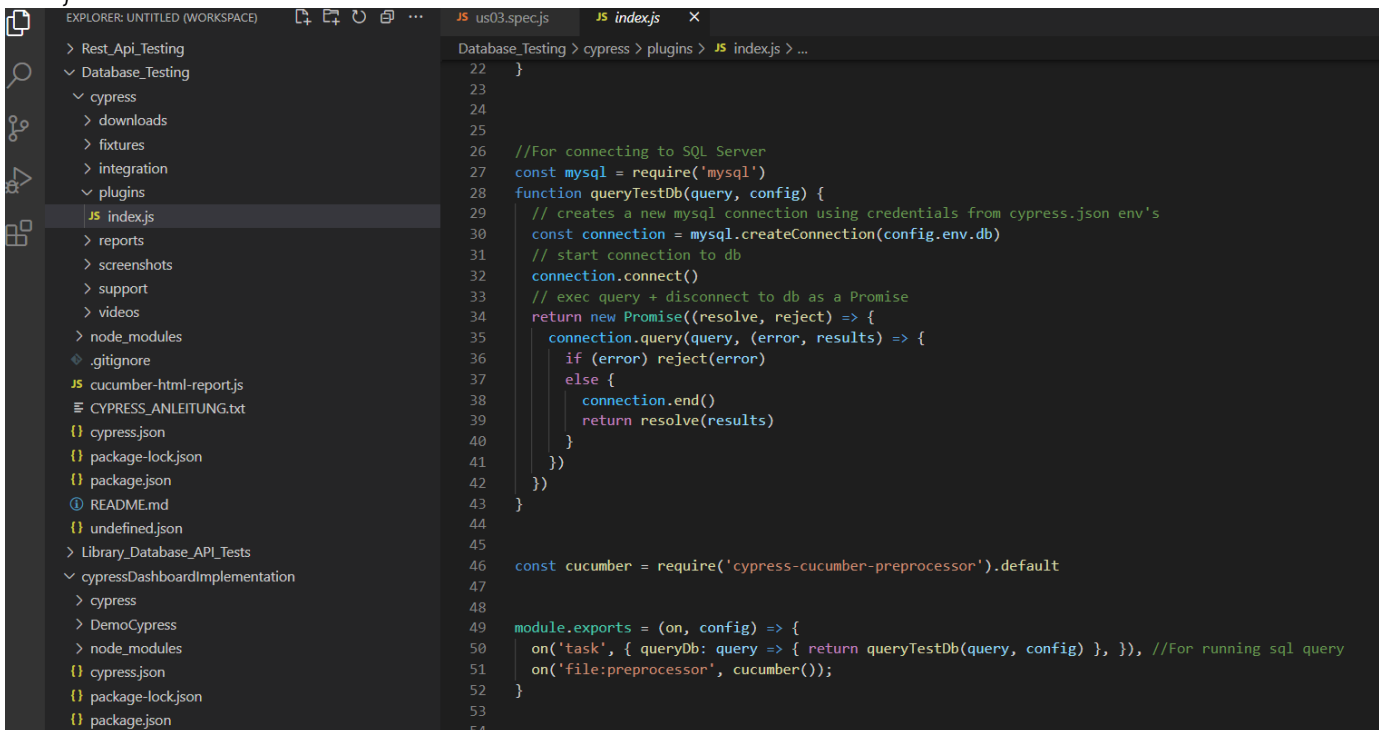
```
# npm i mysql -D
```

2. Wir verwenden das mysql-Plugin, um eine Verbindung zu unserer Datenbank herzustellen. Gehen Sie zu cypress/plugins/index.js und schreiben Sie:

```
//For connecting to SQL Server
const mysql = require('mysql')
function queryTestDb(query, config) {
  // creates a new mysql connection using credentials from cypress.json env's
  const connection = mysql.createConnection(config.env.db)
  // start connection to db
  connection.connect()
  // exec query + disconnect to db as a Promise
  return new Promise((resolve, reject) => {
    connection.query(query, (error, results) => {
      if (error) reject(error)
      else {
        connection.end()
        return resolve(results)
      }
    })
  })
}
```

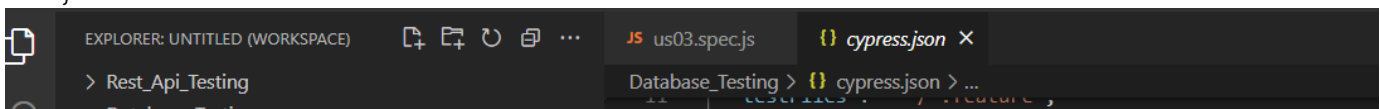
3. Jetzt werden wir cy.task() verwenden, um cypress die Ausführung von SQL-Abfragen zu ermöglichen. Gehen Sie dazu wieder zu cypress /plugins/index.js und schreiben Sie:

```
module.exports = (on, config) => {
  on('task', { queryDb: query => { return queryTestDb(query, config) }, }); //For running sql query
}
```



4. Gehen Sie zu cypress.json und schreiben Sie Folgendes. Diese Details würden verwendet, um eine Verbindung zu unserer Datenbank herzustellen.

```
"env": {
  "db": {
    "host": "http://db4free.net ",
    "user": "admin",
    "password": "password",
    "database": "db_name"
  }
}
```



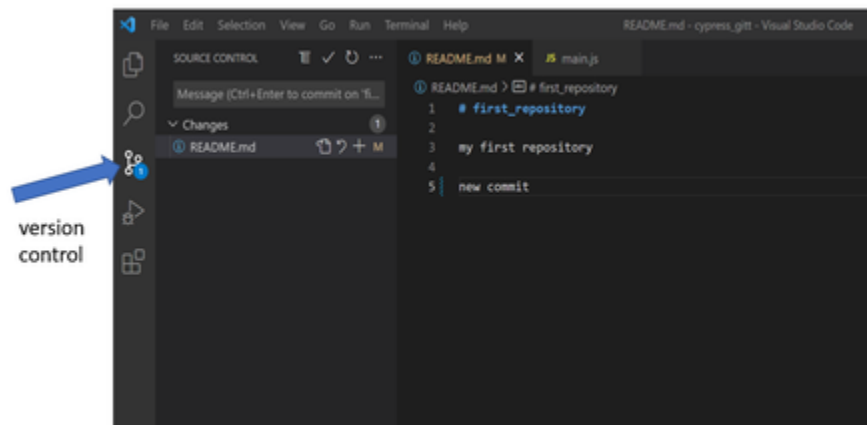


5: Lassen Sie uns nun unseren **Beispiel Test** schreiben, um einige Datenbankaktionen auszuführen.

```
describe('Example to Demonstrate SQL Database Testing in Cypress', () => {
  it('Create a Table', function () { cy.task('queryDb', 'CREATE TABLE Persons (PersonID int, FirstName varchar(255), Address varchar(255), City varchar(255))') })
  it('Input Entries into the table', function () { cy.task('queryDb', 'INSERT INTO Persons (PersonID, FirstName, Address, City) VALUES (001, "John", "House No. 01", "Helsinki"), (002, "Pam", "House No. 02", "Espoo"), (003, "Dwight", "House No. 03", "Lapland"), (004, "Michael", "House No. 04", "Vantaa");').then((result) => { expect(result.affectedRows).to.equal(4) }) })
  it('Update an Entry into the table and verify', function () { cy.task('queryDb', 'UPDATE Persons SET FirstName = "Kevin" WHERE City="Vantaa").then((result) => { expect(result.changedRows).to.equal(1) }) cy.task('queryDb', 'SELECT FirstName FROM Persons WHERE City="Vantaa").then((result) => { expect(result[0].FirstName).to.equal("Kevin") }) })
  it('Verify that there is only one row where the city is Espoo', function () { cy.task('queryDb', 'SELECT COUNT(*) as "rowCount" FROM Persons WHERE City="Espoo").then((result) => { expect(result[0].rowCount).to.equal(1) }) })
  it('Delete a Table and Verify', function () { cy.task('queryDb', 'DROP TABLE Persons').then((result) => { expect(result.message).to.equal("") }) })
})
```

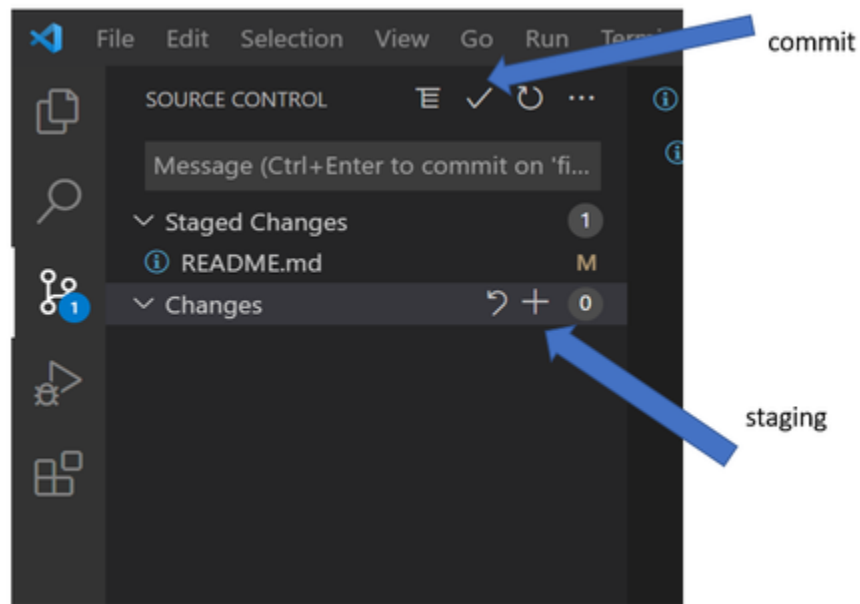
CYPRESS GITHUB

1. Erstellen Sie ein Repository
2. Kopieren Sie den Github-Link
3. Gehen Sie auf die Kommandozeile
4. C:\Benutzer\Benutzer>cd Desktop
C:\Benutzer\Benutzer\Desktop>git clone "github link paste here"
5. Sie sollten den Text sehen:
Clone in 'cypress_gitt'....
remote: Aufzählung der Objekte: 3, fertig.
remote: Zählen von Objekten: 100% (3/3), erledigt.
ferngesteuert: Insgesamt 3 (delta 0), wiederverwendet 0 (delta 0), pack-reused 0.
Empfangen von Objekten: 100% (3/3), erledigt.
6. Erstellen Sie einen Branch und fügen Sie einige Änderungen hinzu.
7. Klicken Sie auf das Versionskontrollsymbol auf der linken Seite.



8. Klicken Sie auf das Plus-Zeichen für die Änderungen, die Sie übertragen wollen, das heißt "staging".

9. Klicken Sie auf das Häkchen und dann auf die "drei punkte" und "push".



10. Gehen Sie zu Github in den neuen BRANCH, erstellen Sie einen Pull Request und führen Sie ihn zusammen.

Um MySQL employees tables zu erstellen :

<https://www.youtube.com/watch?v=-ksz8J6FQ0A>

https://github.com/datacharmer/test_db

Andere nützliche Ressourcen :

Routines :

<https://maxschmitt.me/posts/cypress-routines/#introducing-cypress-routines>

Video-text tutorials :

https://www.youtube.com/playlist?list=PLzDWIPKHyNmK9NX9_ng2IdrkEr8L4WwB0

<https://www.youtube.com/watch?v=Txir3jAooMI&t=511s>

<https://testersdock.com/cypress-tutorial/>

https://www.youtube.com/watch?v=3ir_dyMSu_8&t=363s

https://www.youtube.com/playlist?list=PLMZdod-kiMhKiRztQX_rng7Efcl5OteMR

<https://www.youtube.com/watch?v=3Tn58KQvWtU>

<https://www.youtube.com/watch?v=mFFXuXjVgkU&t=282s>

<https://www.youtube.com/watch?v=Gf0gxV1bYbc&t=870s>