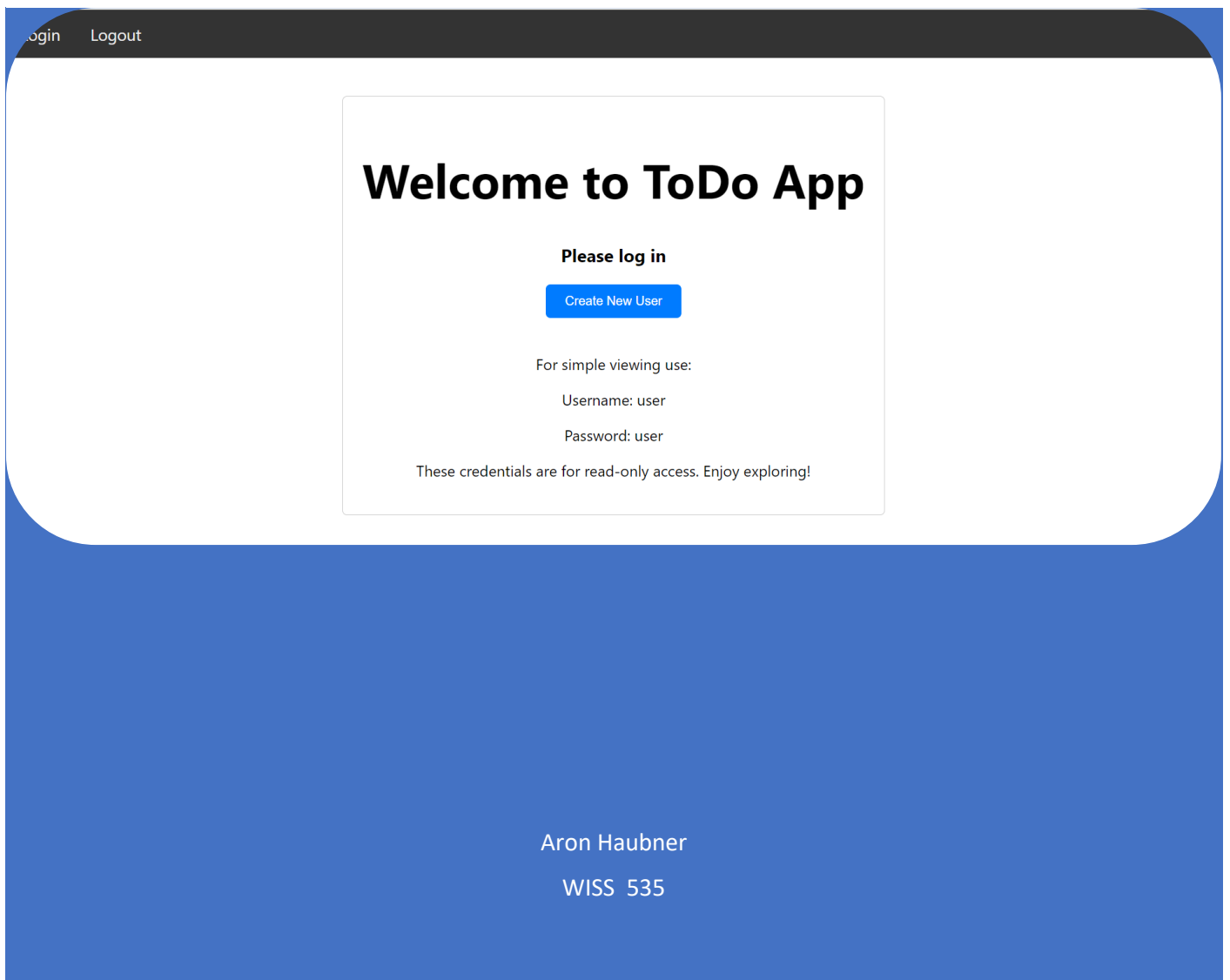


TODO-APP



Inhaltsverzeichnis

Einleitung	1
User Storys	2
User Story #1	2
User Story #2	2
User Story #3	2
FrameWorks	3
Abläufe im Login	3
SicherheitsKonzept	4
1. Authentifizierung und Autorisierung	4
2. Password Encoding	4
3. Session Management	4
4. CSRF-Schutz	4
5. Exception Handling	4
6. Request Authorization	4
7. Security Filter Chain	4
Arbeitsplanung	6
Arbeitsjournal	6
Block #1	6
Block #2	6
Block #3	7
Block #4	7

Einleitung

In dieser Dokumentation wird beschrieben mit welchen Schritten ich vorgegangen bin, um meine Web-Applikation zu erstellen. Am Anfang von diesem Modul 223 wusste ich eigentlich fasst nichts, ich hatte keine Übung mit dem Coden von Web-Apps und hatte auch keine Übung mit dem dazugehörigen. Ich bin ins Modul gestartet mit dem Gedanken „Es wäre schon cool, wenn mehrere Leute eine App benutzen können“ und dem Wunsch es auch zu können. Alles das wurde an Auffahrt (4 Tage Wochenende) Wirklichkeit. Ich hatte so viel Zeit wie noch nie und hatte nichts anderes als die Web-Applikation vor mir. Und da sahs ich mich hin und fing an zu coden, zuerst w3Schools dann Springboot dann Java dann JavaScript dann SQL dann JUnit. „And here we are!“



User Storys

User Story #1

Ich als User möchte ich meine Tasks besichtigen können, um den Überblick über meine Aufgaben zu behalten und sicherzustellen, dass ich nichts vergesse.

Akzeptanzkriterien:

1. Der Benutzer kann eine Liste aller Tasks anzeigen.
2. Jeder Task in der Liste zeigt den Titel und die Beschreibung an.
3. Der Benutzer sieht die Tasks in der richtigen Reihenfolge

User Story #2

Als Administrator möchte ich alle Aufgaben bearbeiten und verwalten können, damit ich neue Tasks erstellen, bestehende Tasks editieren und löschen kann, um eine effiziente Verwaltung und Unterstützung für die Benutzer zu gewährleisten.

Akzeptanzkriterien:

1. Der Administrator kann den Task-Titel und die Beschreibung ins Formular eingeben und Speichern.
2. Nach dem Speichern wird der neue Task in der Task-Liste angezeigt.
3. Der Administrator kann einen bestehenden Task auswählen und bearbeiten.
4. Nach dem Speichern der Änderungen werden diese in der Task-Liste aktualisiert angezeigt.
5. Der Administrator kann einen bestehenden Task auswählen und löschen.
6. Nach der Bestätigung wird der Task aus der Task-Liste entfernt.

User Story #3

Als neuer Benutzer möchte ich einen neuen Account erstellen können, damit ich Zugang zur Anwendung und deren Funktionen erhalten kann.

Akzeptanzkriterien:

1. Auf der Startseite ist eine deutlich sichtbare Schaltfläche mit der Beschriftung "Registrieren" oder "Neuen Account erstellen" vorhanden.
2. Durch Klicken auf die Schaltfläche wird das Registrierungsformular geöffnet.
3. Der Benutzer kann die folgenden Informationen eingeben:
 - Benutzername
 - E-Mail-Adresse
 - Passwort
 - Rolle

FrameWorks

Diese Frameworks bieten eine umfassende Basis für die Entwicklung von sicheren, datenbankgestützten Webanwendungen mit Java. Sie unterstützen Datenzugriff (Spring Data JPA), Sicherheit (Spring Security), Token-Authentifizierung (JWT), Validierung (Jakarta Validation), und Testen (JUnit) Spring Boot: Bietet die Grundlage und erleichtert die Konfiguration und den Betrieb der Anwendung.

1. **Spring Data JPA:** Ermöglicht die einfache Arbeit mit Datenbanken über JPA und Hibernate.
2. **Spring Security:** Fügt Sicherheitsfunktionen hinzu.
3. **JWT:** Handhabt JSON Web Tokens für die Authentifizierung und Autorisierung.
4. **MySQL:** Ermöglicht die Verwendung von MySQL-Datenbanken.
5. **JUnit und andere Testbibliotheken:** Unterstützen das Testen der Anwendung.
6. **Maven**

Abläufe im Login

1. **Benutzer Login (Schritt 1):** Der Benutzer sendet eine POST-Anfrage an den Endpoint `/api/auth/signin` mit seinen Anmeldedaten (Benutzername und Passwort).

2. **Authentifizierung und Token-Erstellung (Schritt 2):** Der

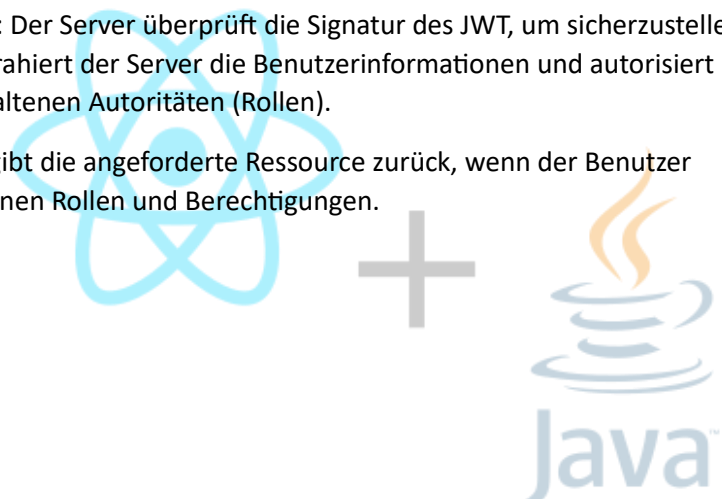
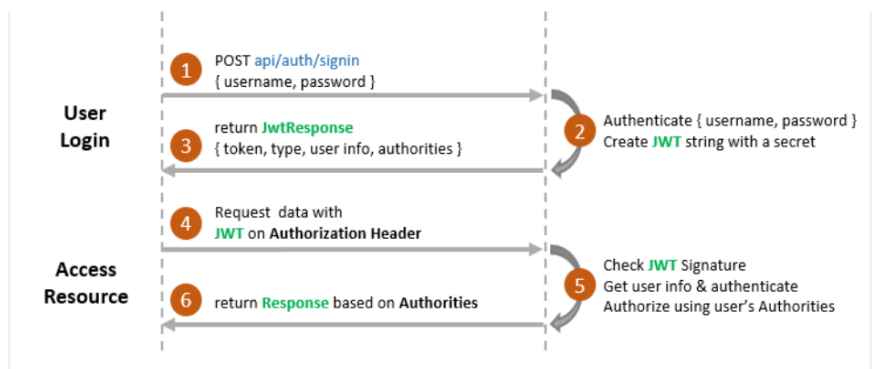
Server authentifiziert den Benutzer mit den übermittelten Anmeldedaten. Wenn die Authentifizierung erfolgreich ist, erstellt der Server ein JWT (JSON Web Token) mit einem geheimen Schlüssel.

3. **JWT-Antwort (Schritt 3):** Der Server gibt eine `JwtResponse` zurück, die das Token, den Token-Typ, Benutzerinformationen und Autoritäten (Rollen) enthält.

4. **Anforderung einer Ressource (Schritt 4):** Der Benutzer fordert eine geschützte Ressource an und übergibt das JWT im `Authorization`-Header der Anfrage.

5. **Token-Prüfung und Autorisierung (Schritt 5):** Der Server überprüft die Signatur des JWT, um sicherzustellen, dass es nicht manipuliert wurde. Danach extrahiert der Server die Benutzerinformationen und autorisiert den Zugriff basierend auf den im Token enthaltenen Autoritäten (Rollen).

6. **Ressourcen-Antwort (Schritt 6):** Der Server gibt die angeforderte Ressource zurück, wenn der Benutzer autorisiert ist, sie zu sehen, basierend auf seinen Rollen und Berechtigungen.



Sicherheitskonzept

1. Authentifizierung und Autorisierung

- **JWT-basierte Authentifizierung:** Das System verwendet JSON Web Tokens (JWT) zur Authentifizierung von Benutzern. Ein Benutzer meldet sich mit einem Benutzernamen und Passwort an und erhält ein JWT, das bei jeder weiteren Anfrage zur Identifizierung und Autorisierung verwendet wird.
- **Rollenbasierte Zugriffskontrolle:** Berechtigungen werden durch Rollen definiert. Nur Benutzer mit spezifischen Rollen (z.B. ROLE_ADMIN) dürfen bestimmte Endpunkte aufrufen.

2. Password Encoding

- **BCryptPasswordEncoder:** Passwörter werden mit dem BCrypt-Algorithmus verschlüsselt, um die Sicherheit der gespeicherten Passwörter zu erhöhen und sie gegen Brute-Force-Angriffe zu schützen.

3. Session Management

- **Stateless Session Management:** Die Anwendung ist zustandslos konfiguriert, was bedeutet, dass keine Sitzungsinformationen auf dem Server gespeichert werden. Jede Anfrage muss ein gültiges JWT enthalten.

4. CSRF-Schutz

- **CSRF-Deaktivierung:** CSRF-Schutz ist deaktiviert, da die Anwendung zustandslos ist und JWT für die Authentifizierung verwendet. In einem stateless Setting ist CSRF-Schutz in der Regel nicht notwendig.

5. Exception Handling

- **Custom AuthEntryPointJwt:** Ein benutzerdefinierter Entry Point (AuthEntryPointJwt) wird verwendet, um nicht authentifizierte Zugriffsversuche zu behandeln und entsprechende Fehlermeldungen zurückzugeben.

6. Request Authorization

- **URL-basierte Sicherheitsregeln:** Verschiedene URL-Muster haben unterschiedliche Zugriffsbeschränkungen:
- **Öffentliche Endpunkte:** /home, /getTasks, /api/auth/**, /api/test/** sind für alle Benutzer zugänglich.
- **Admin-geschützte Endpunkte:** /addTask, /myTesting, /delTask/*, /uptTask/* sind nur für Benutzer mit der Rolle ROLE_ADMIN zugänglich.
- **Alle anderen Anfragen:** Alle anderen Endpunkte erfordern eine Authentifizierung.

7. Security Filter Chain

- **JWT Token Filter:** Der AuthTokenFilter wird vor dem UsernamePasswordAuthenticationFilter hinzugefügt, um JWTs in eingehenden Anfragen zu verarbeiten und die Authentifizierung zu handhaben.

The screenshot displays a web application interface. At the top, a notification box shows the message "localhost:3000 says logged in!" with an "OK" button. Below this is a "Login" section containing a "Username:" field with the value "admin" and a "Password:" field with masked characters ".....". A blue "Login" button is positioned at the bottom of the form.

Testprotokoll

Idee #1

Dieses Array (`numberArray`) ist dafür da, dass alle Tasks die korrekte Deskription haben. Alle Daten von `/getTasks` werden mit einem `foreach` gespeichert. Wenn wir die richtige Tasks Deskription anzeigen wollen schauen wir welcher Eintrag in der Array die gleiche ID hat, und dann nehmen wir alle Daten von dort (wie zbsp. Die Task Beschreibung)

Zum Testen habe ich das Array in der Konsole ausgegeben.

```
const numberArray = [];  
// Clear existing numberArray  
  
    numberArray.splice(0, numberArray.length);  
    let currentLength = 1;  
    // Populate numberArray with task IDs  
    //we carefullly put all the data in array here  
    // I did this so that the tasks are numberd  
    data.forEach(task => {  
        numberArray.push({ taskid: task.taskid, taskName: task.taskName,  
taskDescription: task.taskDescription, length: currentLength });  
        currentLength++; // Increment after each element handled  
    });  
  
    console.log(numberArray);  
    // i did this output for testing  
    // console.log(numberArray);  
    })  
    .catch(error => {  
        console.error('Fetch error: ', error);  
    })  
}
```

Idee #2

Um zu testen, ob der User wirklich eingeloggt ist und um unerlaubten zugriff ohne einloggen zu verhindern. Dieser key mit Namen „status“ übernimmt dies.

Um zu testen, ob der User tatsächlich eingeloggt ist, speichern wir im LocalStorage eine Variabel die hinzugefügt wird oder entfernt wird je nach dem.


Key	Value
status	isLoggedIn
debug	honey:core-sdk:*
user	{\"id\":5,\"username\":\"admin\",\"email\":\"admin@example.c...
1 isLoggedIn	

Key	Value
status	notLoggedIn
debug	honey:core-sdk:*
1 notLoggedIn	

Ob Daten geladen werden sollen oder nicht wird von dieser Variabel entschieden. (Sicherheitslücken sind vorhanden)

Arbeitsplanung




Meine Arbeitsplanung ist vor allem in meinen Git-Commits sichtbar. Nach jedem Arbeitsblock oder am Ende des Tages habe ich einen Commit durchgeführt und so kann man meine Planung auch sehr gut nachvollziehen.

 **28 Commits**

11:11 PM
5/20/2024

Tag 01 - 09.04.2024

Commits on Apr 9, 2024

Initial commit
 Ratedaron committed last month
Verified 3c9c6f5  

Arbeitsjournal

Block #1

Block #1 bestand aus

Kennenlernen des neuen

Moduls und auch aus der Festsetzung der Idee für die Applikation. Ich habe auch noch entschieden in einer Einzelgruppe zu arbeiten, weil ich so vieles allein machen konnte, was von Vorteil war für mich. Ich hatte viele gute Erfahrungen mit Gruppenarbeit, aber am Ende lernte ich nie alles, was wir machten. Und so entschied ich mich für eine Einzelgruppe mit dem Thema TODO-List, weil dies bekannt ist dafür, dass sie einfach ist. Sonst ist nichts erstellt worden ausser dem GIT-Repository.









Block #2

Block #2 bestand aus den ersten Schritten Richtung Multiuser. Wir erstellten Controllern bei denen nur Benutzer mit bestimmten Attributen zugreifen konnten. Dies hat mir einen guten Einblick ins «what's to come»

gegeben und es hat mir auch gefallen. Noch knapp vor Schulschluss habe ich es geschafft des Controllers zum Funktionieren zu bringen. Ich habe aber nicht viel mitgenommen. Das kommt später....

Commits on Apr 16, 2024

v1
 Ratedaron committed last month
596dc63  

private and public controller
 Ratedaron committed last month
441d8ca  



Commits on May 7, 2024

Block #3

Vom siebten Mai bis 14. Mai habe ich viel gearbeitet. Ich fing an mit den Controllern von meiner TODO-App und ich schaffte es bis zu einem funktionierenden Login am 14. Mai.

working post and del HAHAAHAHA

Ratedaron committed 2 weeks ago

Commits on May 14, 2024

Login done - I used LocalStorage a bit

Ratedaron committed last week

login working

Ratedaron committed last week

Ich hatte viel zu viele Commits in der Zwischenzeit, um diese hier abzubilden, aber sie werden dafür zugänglich sein auf meinem GIT, welches ich noch public machen werde. Die Login Sache war ein bisschen kompliziert, aber als mich einer meiner Klassenkameraden beraten hatte, wusste ich was falsch sein könnte. Die einfachere Methode (Session-based) für das Login war nicht gut genug und ich musste zu Token-Based wechseln und mit JWTs arbeiten.

Block #4

Tag 04 - 14.05.2024

Im Block vier bin ich fertig geworden mit der ganzen Web-Applikation. Alles funktioniert, wie es sollte, aber einige kleinen quirks verstecken sich noch (Sicherheitslücken). Für diese LB langt es aber. Dies ist ein Projekt, welches ich weitertragen werde und weiter ausarbeiten werde. Es hat mir sehr gefallen an diesem Projekt zu arbeiten, weil ich mit jeder Zeile Code mehr gelernt habe und ich würde gern weiterlernen, weil ich noch nie so viel Können gehabt habe. Cya next time!!

Commits on May 20, 2024

added Comments and very simppls model tests, sel tests didnt work

Ratedaron committed 3 hours ago

Commits on May 19, 2024

I'm done, it works, only thing left is implementing tests and comments yuuu huu!!

Ratedaron committed yesterday

```

14 | // most important file in backend!!!!
15 |
16 | @SpringBootApplication
17 | @Import(CorsConfig.class)
18 | public class TodoApplication {
19 |
20 |     public ArrayList<String> tasks = new ArrayList<>(); // new item
21 |
22 |     Run | Debug
23 |     public static void main(String[] args) {
24 |         SpringApplication.run(TodoApplication.class, args);
    }

```