

Willkommen bei meinem Test Web-Projektes.

In diesem Artikel wird erläutert, wie sich ein User mit mithilfe von Java Spring Boot-Framework in unserer Webseite anmelden kann.

Es ist sinnvoll, sich vor dem Lesen des Artikels die Vorkenntnisse in Java, HTML, CSS und JavaScript anzueignen.

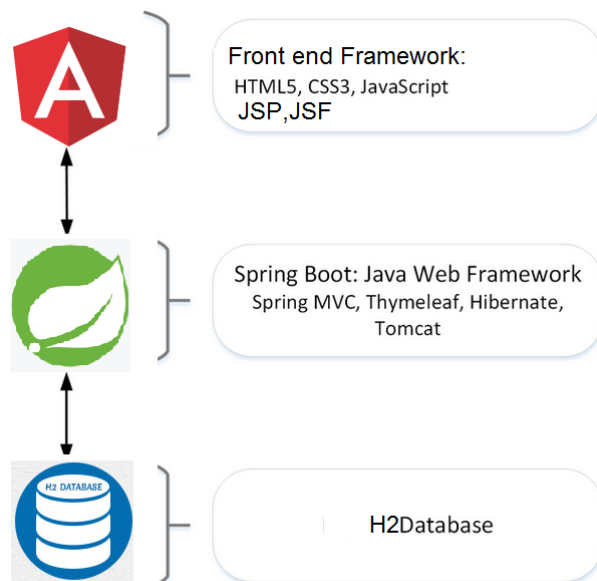
Obwohl meine deutsche Sprache nicht perfekt ist, versuch ich dies Thema zu erklären.

Sie können den Quellcode, Der auf GitHub verfügbar ist, herunterladen.

Die Daten sind in H2database gespeichert, die nur für kleine Datenmenge geeignet ist.

Dieses Projekt wurde mit Spring Tools Suite Version 3 (Eclipse Oxygen.2) erstellt, sodass alle Anweisungen auf dieser IDE basieren. Und enthält auch Maven 4, Junit, Log4j2, und H2database.

Der Leser kann die View dies Projekt mithilfe von Front end Framework verbessern wie Angular, Primface, JQuery.



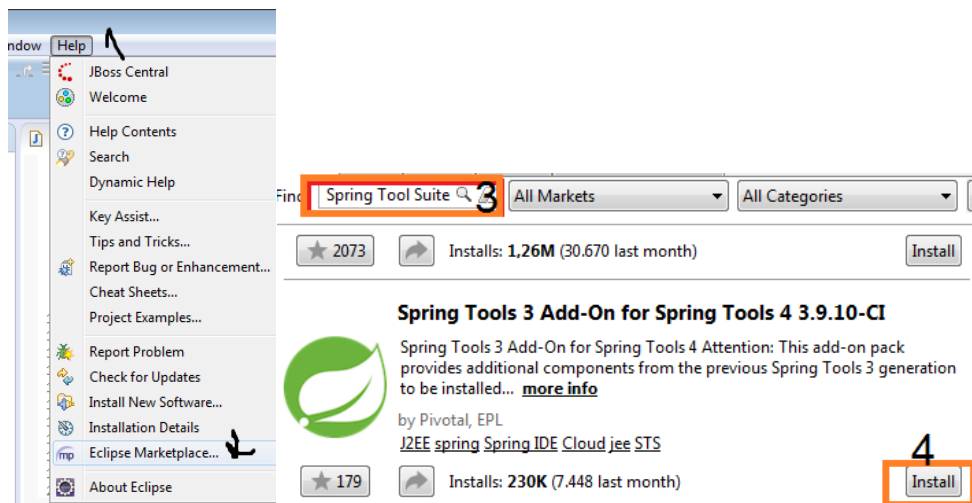
In diesen Tipp *erklären wir* Ihnen *Schritt für Schritt*, wie Sie auch ein Web-Projekt erstellen möchten, können Sie dieser Anleitung *folgen*:

## Schritte 1: Spring-Boot Installieren.

Gehen Sie auf der Webseite <https://spring.io/tools3/sts/legacy>

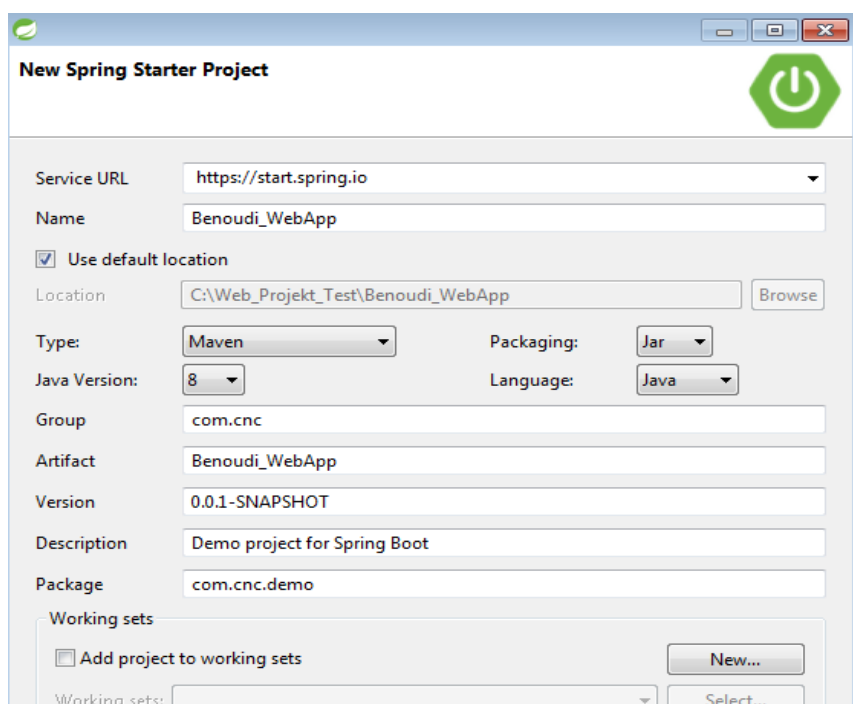


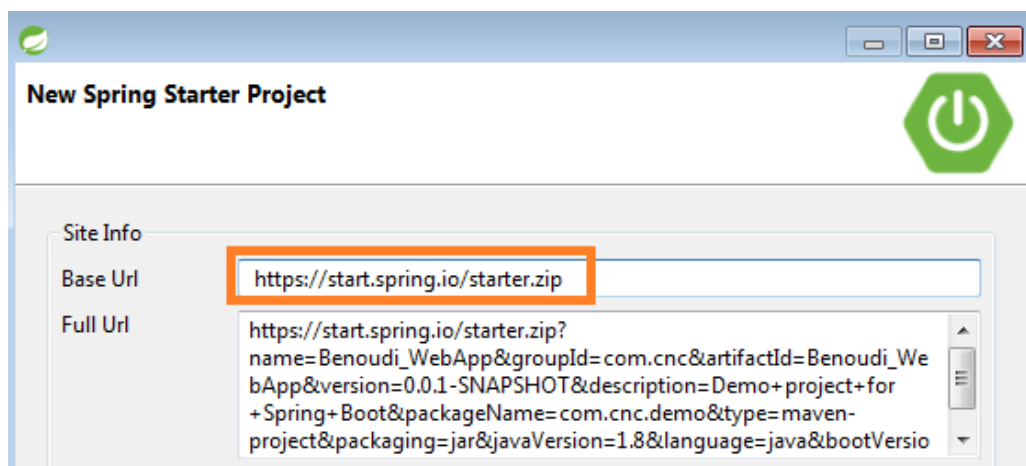
Oder direkt auf der Eclipse IDE installieren.



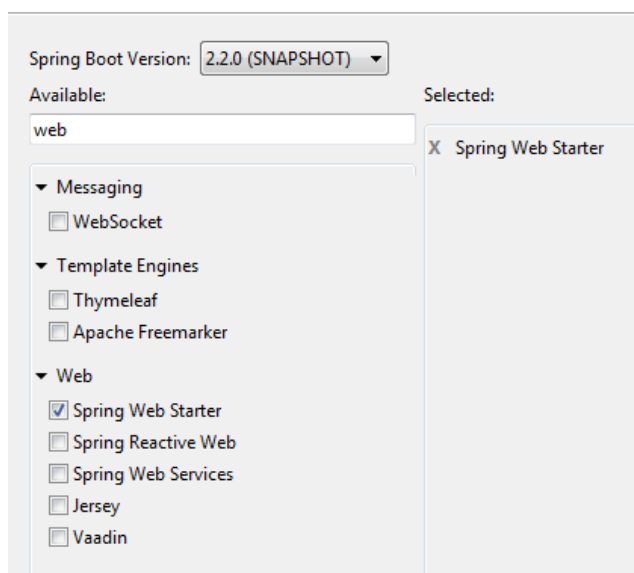
In meinem Projekttest habe ich das STS3 (Spring Tools Suite Version 3) installiert.

## Schritte 2: Nues Web-Projekt erstellen.





### New Spring Starter Project Dependencies



## Schritte 3: POM.xml Datei.

Wir werden für dieses Projekt verwenden. Wir brauchen folgenden Dependencies um unserer Projekt in Maven-4-Tool funktionieren zu können. Der in Pom.xml Datei des Mavenes gelöst werden sollen.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.0.BUILD-SNAPSHOT</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.cnc</groupId>
```

```

<artifactId>Benoudi_WebApp</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>Benoudi_WebApp</name>
<description>Demo project for Spring Boot</description>

<properties>
    <java.version>1.8</java.version>
    <maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>

        <exclusions>
            <exclusion>
                <groupId>org.junit.vintage</groupId>
                <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
    <!-- protection as well as basic authentication on all HTTP endpoints
-->

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.tomcat/tomcat-
jasper -->

    <dependency>
        <groupId>org.apache.tomcat</groupId>
        <artifactId>tomcat-jasper</artifactId>
        <version>9.0.22</version>
    </dependency>

    <!-- https://mvnrepository.com/artifact/com.h2database/h2 -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!--
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-
data-jpa -->

```

```

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <!--
https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-
log4j2 -->
    <!-- Exclude Spring Boot's Default Logging -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</artifactId>
      <exclusions>
        <exclusion>
          <groupId>org.springframework.boot</groupId>
          <artifactId>spring-boot-starter-
logging</artifactId>
        </exclusion>
      </exclusions>
    </dependency>

    <!-- Add Log4j2 Dependency -->
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-log4j2</artifactId>
    </dependency>

  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>

      </plugin>
    </plugins>

  </build>

  <repositories>
    <repository>
      <id>spring-milestones</id>
      <name>Spring Milestones</name>
      <url>https://repo.spring.io/milestone</url>
    </repository>
    <repository>
      <id>spring-snapshots</id>
      <name>Spring Snapshots</name>
      <url>https://repo.spring.io/snapshot</url>
      <snapshots>
        <enabled>true</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>

```

```

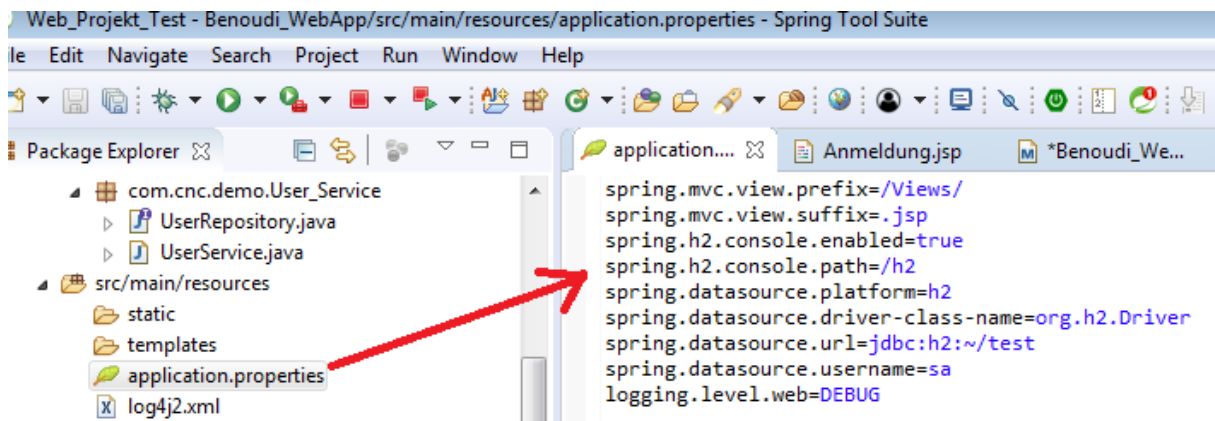
        <id>spring-milestones</id>
        <name>Spring Milestones</name>
        <url>https://repo.spring.io/milestone</url>
    </pluginRepository>
    <pluginRepository>
        <id>spring-snapshots</id>
        <name>Spring Snapshots</name>
        <url>https://repo.spring.io/snapshot</url>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </pluginRepository>
</pluginRepositories>

```

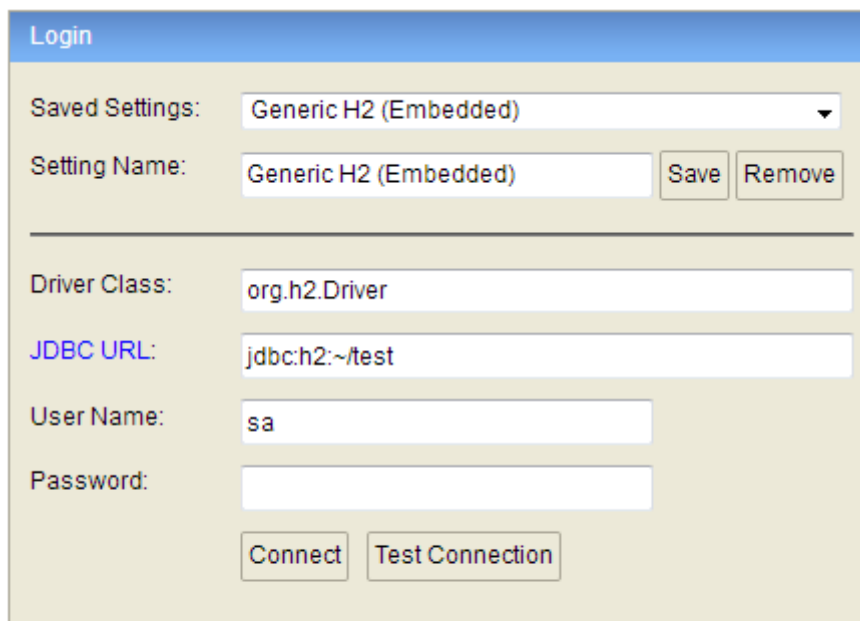
```
</project>
```

## Schritte 4: H2database.

Die Konfiguration der H2database liegt im application.properties



In der Browser logen wir auf (localhost/h2) um die Verbindung zu testen.



The screenshot shows a 'Login' window with a blue header. It contains several input fields and buttons. At the top, 'Saved Settings:' is followed by a dropdown menu showing 'Generic H2 (Embedded)'. Below this, 'Setting Name:' is followed by a text box with 'Generic H2 (Embedded)' and 'Save' and 'Remove' buttons. A horizontal line separates this from the main configuration section. 'Driver Class:' is followed by a text box with 'org.h2.Driver'. 'JDBC URL:' is followed by a text box with 'jdbc:h2:~/test'. 'User Name:' is followed by a text box with 'sa'. 'Password:' is followed by an empty text box. At the bottom are 'Connect' and 'Test Connection' buttons.

Test successful

## Schritte 5: Model User erstellen.

Mithilfe von der Annotation `@Entity` JPA speichert für uns Automatik die Informationen des Benutzer in der H2database.

```

@Entity
@Table(name = "Users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "User_id")
    private int User_ID;
    @Column(name = "Vorname")
    private String Vorname;
    @Column(name = "nachname")
    private String nachname;

    @Column(name = "Email")
    // @NotEmpty(message = "Please provide an e-mail")
    private String Email;

    @Column(name = "Password")
    @Transient
    private String Password;

    // Getter und Setter Methoden
    public int getUser_ID() {
        return User_ID;
    }

    public void setUser_ID(int user_ID) {
        this.User_ID = user_ID;
    }

    public String getEmail() {
        return Email;
    }
}

```

## Schritte 6: Repository Erstellen.

Der Interface, oder Service genannt, CrudRepository ist für Create, Read, Update, Delete und die anderen Anfragen von Query SQL verantwortlich.

```

package com.cnc.demo.User_Service;
* @author Benoudi Nasser

import java.util.List;

public interface UserRepository extends CrudRepository<User,Integer> {
    // Achtung bei Generic Klassen nicht primitive Typen wie ( int, long , ... ) als Parametern angeben
    // sondern die Primitive Typen wie Integer, Long,...

    @Query
    List<User> findByNachname(@Param(value = "Nachname") String Nachname );
    // @Query
    // List<User> findByEmailAndPassword(@Param(value = "Email") String email ,@Param(value = "Password")String password);
}

```

## Schritte 7: Service User basiert auf Repository Erstellen.

Mithilfe der Anotation @Service könne wir ein Service der CRUD Methoden löst erstellen.



```

package com.cnc.demo.User_Service;

import java.util.List;

@Service
public class UserService {
    @Autowired
    private UserRepository repository;

    public User addUser(User user) {
        return repository.save(user);
    }

    public List<User> getUsers(){
        List<User> users= (List<User>)repository.findAll();
        return users;
    }

    // get List of User mit Nachname
    public List<User> getUserbyNachname(String nachname){
        List<User> users= (List<User>)repository.findByNachname(nachname);
        return users;
    }

    public User deleteUser(User user) {
        repository.delete(user);
        return user;
    }
}

```

## Schritte 8: Controller Erstellen.

In Model View Controller brauchen wir drei Sachen:

Eine klasse für das Model, und eine andere für das Controller und die letzte für das View.

Wir haben schon in der Schritte 5 das UserModel erwähnt, jetzt kommen wir zur Erstellung der UserController Klasse zu.

```

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.cnc.demo.Model.User;
import com.cnc.demo.User_Service.UserService;

@RestController
public class UserController {
    @Autowired
    private UserService service;

    @GetMapping("/signup")
    public String showSignUpForm(User user) {
        return "Anmeldung";
    }

    @GetMapping("/")
    public String home() {
        // return "Home"

        return "home";
    }
    // to Create benutzen wir Post annotation

    public User speicherUser(@RequestBody User user) {
        user = service.addUser(user);

        return user;
    }

    @GetMapping(value = "/getUsers")
    public List<User> findallUser() {
        return service.getUsers();
    }

    @GetMapping(value = "/getUserByNachname/{nachname}")
    public List<User> getUserByNachname(@PathVariable("Nachname") String nachname) {
        return service.getUserbyNachname(nachname);
    }

    @DeleteMapping(value = "/loeschen")
    public User loeschen(@RequestBody User user) {
        return service.deleteUser(user);
    }
}

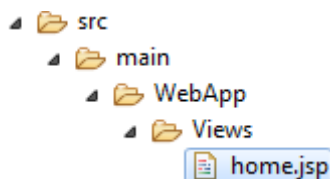
```

## Schritte 9: Log4j2 hinzufügen.

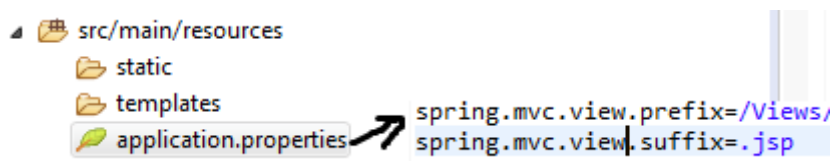
Der dependency spring-boot-starter-logging wird exkludiert und der dependency spring-boot-starter-log4j2 wird inkludiert. Das Log Datei log4j2.xml liegt in src/main/resources Ordner.

## Schritte 10: Views.

Unser **Views** Webseiten liegen in der Ordner **WebApp/Views**



Es ist sinnvoll, dem prefix( /Views) und dem Suffix ( .jsp oder .htm) in der URL zu verstecken. Als Beispiel : mit der Einstellung in der **application.properties**, können wir den URL `http://localhost:8080/views/home.jsp` wechseln zum `http://localhost:8080/home`.



## Schritte 10: Sicherheit.

Es gibt eine Menge Annotationen die für Sicherheit gebraucht sind, wie

`@Configurable`, `@EnableWebSecurity`,

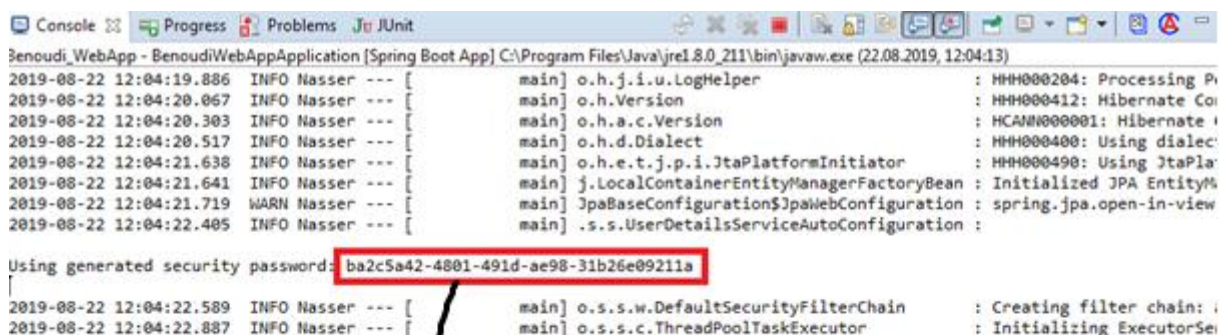
`@AccessTimeout` : gibt ein Time-Out für Konkurrent Zugriff angegeben.

Ein Wert von 0 bedeutet, dass Konkurrent Zugriffe nicht erlaubt ist.

Ein Wert von -1 bedeutet, dass Client-Zugriff unendlich lange warten.

Default Unit ist Millisekunden.

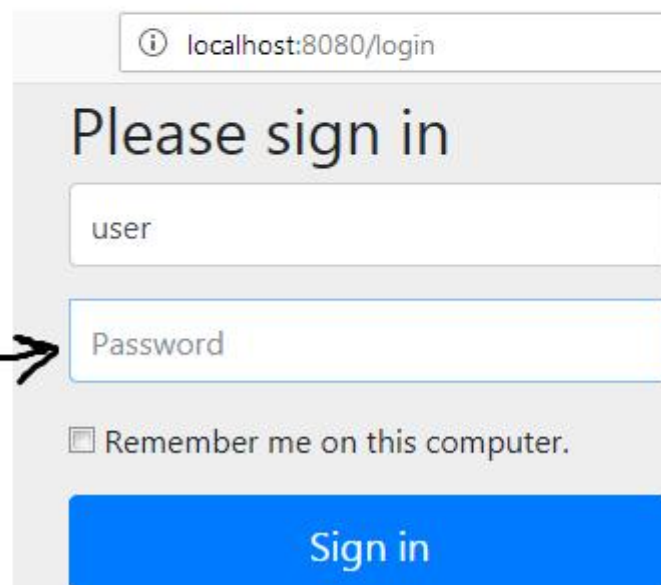
`@TransactionAttribute` :



```

Benoudi_WebApp - BenoudiWebAppApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_211\bin\javaw.exe (22.08.2019, 12:04:13)
2019-08-22 12:04:19.886 INFO Nasser --- [main] o.h.j.i.u.LogHelper : HHH000204: Processing P
2019-08-22 12:04:20.067 INFO Nasser --- [main] o.h.Version : HHH000412: Hibernate Co
2019-08-22 12:04:20.303 INFO Nasser --- [main] o.h.a.c.Version : HCAHH000001: Hibernate
2019-08-22 12:04:20.517 INFO Nasser --- [main] o.h.d.Dialect : HHH000400: Using dialect
2019-08-22 12:04:21.638 INFO Nasser --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPla
2019-08-22 12:04:21.641 INFO Nasser --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityM
2019-08-22 12:04:21.719 WARN Nasser --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view
2019-08-22 12:04:22.405 INFO Nasser --- [main] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: ba2c5a42-4801-491d-ae98-31b26e09211a
2019-08-22 12:04:22.589 INFO Nasser --- [main] o.s.s.w.DefaultSecurityFilterChain : Creating filter chain: i
2019-08-22 12:04:22.887 INFO Nasser --- [main] o.s.s.c.ThreadPoolTaskExecutor : Initializing ExecutorSe
  
```



localhost:8080/login

### Please sign in

user

Password

☐ Remember me on this computer.

Sign in

## Schritte 11: Test mit Junit.

```

package com.cnc.demo;

import static org.junit.jupiter.api.Assertions.assertEquals;
//import com.cnc.demo.User_Service.UserService;

@SpringBootTest

class BenoudiWebAppApplicationTests {

    @MockBean
    private UserRepository repository;

    @Test
    public void getUsersTest() {
        // Bevor das Test zu testen ich habe User(1,n.benoudi@gmail,Benoudi,123,Nasser
        List<User> users = repository.findByNachname("Benoudi");
        assertEquals(users.get(0).getNachname(), "Benoudi");
    }

    @Test
    void contextLoads() {
    }

}

```

## Schritte 11: REST-API.

Um unser REST-API zu testen, kann man mithilfe von Postman oder Curl Software.

Wenn der Client die Daten als xml oder als Json von Server brauchte, kann er Postman oder Curl benutzen.

```

curl http://localhost:8080 /
curl -H "Accept: text / plain " http://localhost:8080 /getUsers
curl -H "Accept: text / html " http://localhost:8080 /getUsers
curl -H "Accept: application /xml " http://localhost:8080 /getUsers

```



## curl for 64 bit

Size: 3.1 MB

sha256: 19384567361d89261b92373ef4950e257c2fb4c72a7c4fbc4bbdaf463f83ff39

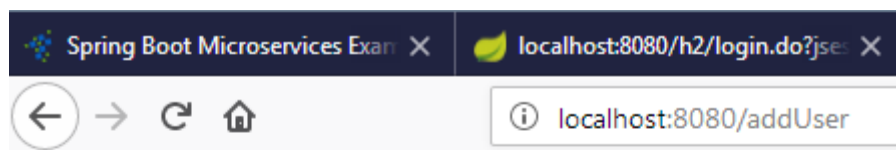
```
C:\Windows\system32\cmd.exe
X-Frame-Options: DENY
Location: http://localhost:8080/login
Content-Length: 0
Date: Thu, 22 Aug 2019 12:29:05 GMT
Connection #0 to host localhost left intact

:\Users\itto\Desktop\curl-7.65.3-win32-mingw\bin>curl -v localhost:8080/getUsers
Trying ::1:8080...
TCP_NODELAY set
Connected to localhost (::1) port 8080 (#0)
GET /getUsers HTTP/1.1
Host: localhost:8080
User-Agent: curl/7.65.3
Accept: */*

Mark bundle as not supporting multiuse
HTTP/1.1 302
Set-Cookie: JSESSIONID=60EC3D20F6D118EECC1C07FFCA03DB9D; Path=/; HttpOnly
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
X-Frame-Options: DENY
Location: http://localhost:8080/login
Content-Length: 0
Date: Thu, 22 Aug 2019 12:30:15 GMT
```

curl -v localhost:8080/getUsers -h "Accept: application/json"

Die im Formular erfassten *Daten* werden nun in der *Datenbank* gespeichert. Die Vorgehensweise wird gezeigt.



## Register Form

Vorname

Nachname

Email

Password

Address

Reference:

<https://developer.paypal.com/docs/api/overview/#get-credentials>

[http://h2database.com/html/features.html#database\\_url](http://h2database.com/html/features.html#database_url)