Modul 223

Multiuser - Apps

Unterrichtsziel heute

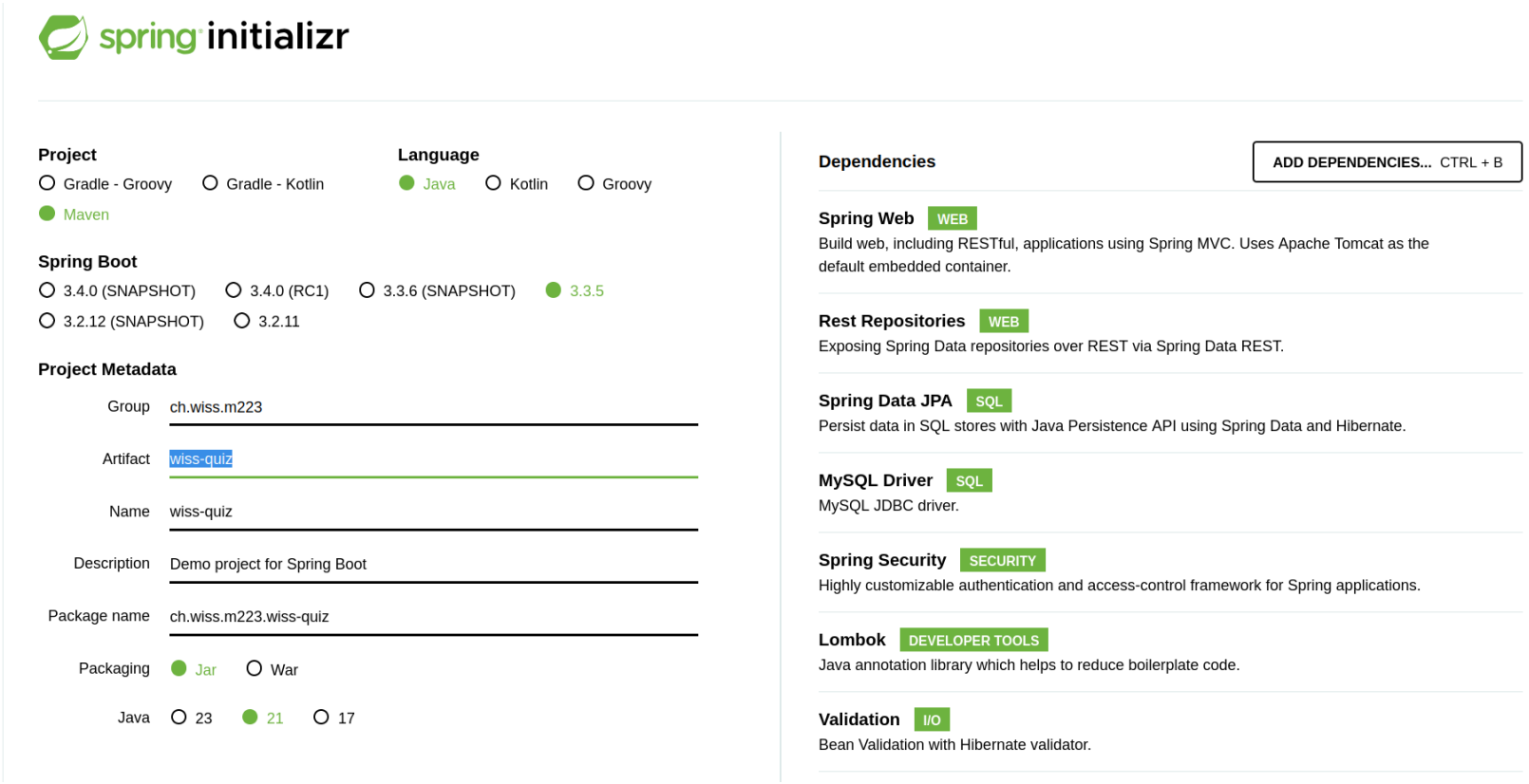
- ~~einrichten Backend-Projektumgebung~~
- nutzen und konfigurieren Spring-Security
- implementieren FilterChain
- implementieren RBAC – Datenmodell mit JPA
- erklären Unterschiede und Abläufe bei sessionbasierter und Token-basierter Auth²

Wie war das nochmal...

- Was war nochmal das Besondere an Multiuser-Apps?
- Worauf muss besonders geachtet werden?
- Was hat es mit RBAC auf sich?
- Wie werden HTTP-Requests in einer Springboot-Webapp verarbeitet?
- Wie funktioniert die persistente Datenspeicherung mit der Java Persistency API (JPA)?

Backend-Projekt einrichten

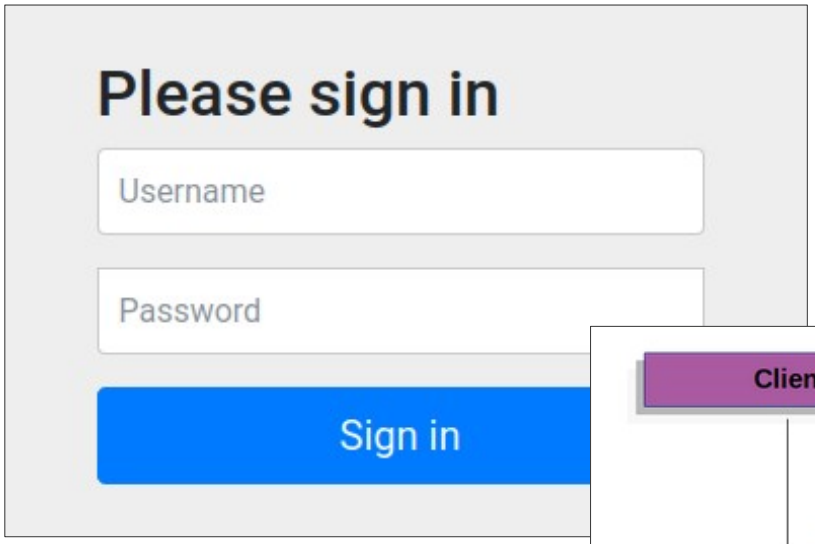
- Template mit Spring-Initializr (https://start.spring.io/) generieren



Backend-Projekt einrichten

- Template entpacken
- in pom.xml: JPA Abschnitt vorerst auskommentieren
- in IDE des Vertrauens ausführen

Spring Security Intro



Spring Security ist eine **zusätzlich Schicht *zwischen*** Client und Spring App, die **jeden Request** anhand der **Security Regeln prüft**. Security Regeln werden als **Filter** definiert.

Die Filter bilden eine Kette (**Filter Chain**). Jeder Request durchläuft die Filter der Kette entsprechend ihrer Reihenfolge. Dabei **prüft jeder Filter**, ob dem Request **Zugriff** auf die Anwendung gewährt wird oder nicht.

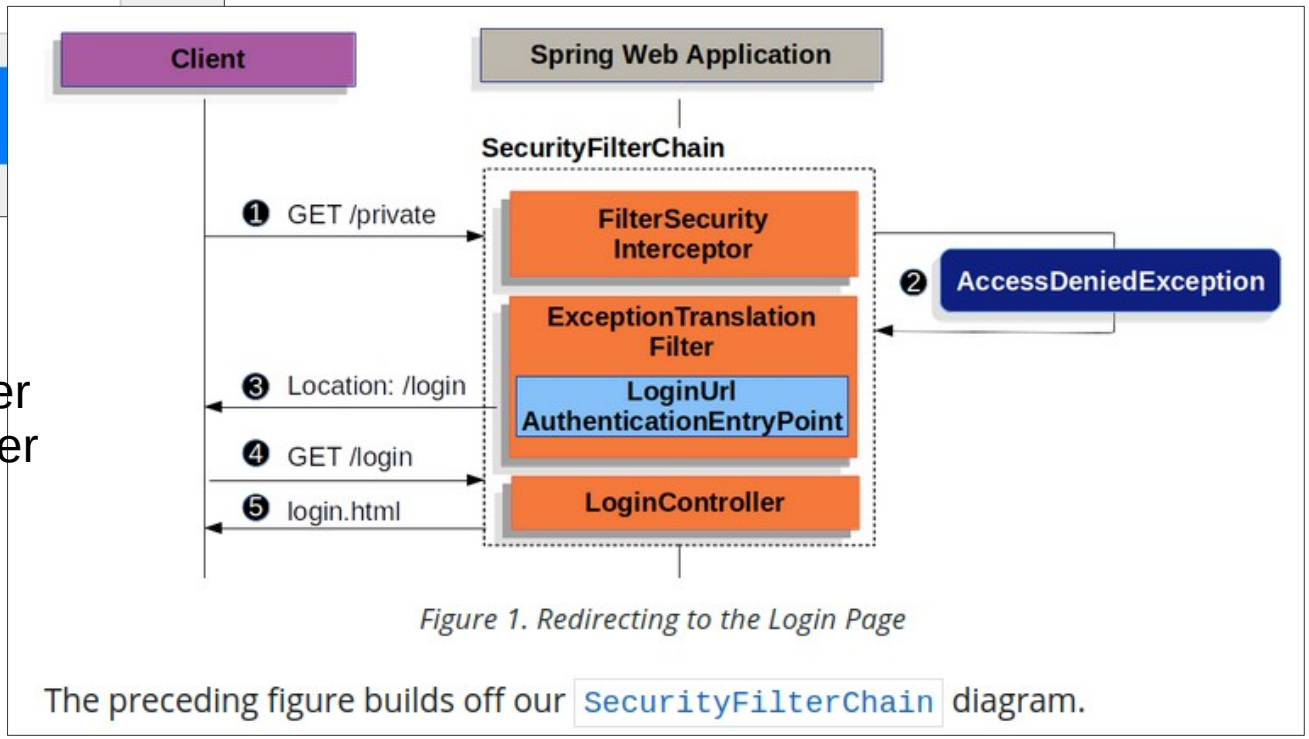


Figure 1. Redirecting to the Login Page

The preceding figure builds off our `SecurityFilterChain` diagram.

https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/form.htm

Spring Security Basiskonfiguration

- alles ist geschützt
- jeder Zugriff mit Benutzer+PWD im Auth-Header
- Standard-"user" mit generiertem Passwort (siehe StartLog), wenn nichts anderes definiert ist
- z.B. eigener Standard-"user" konfigurieren in **application.properties**:

```
spring.security.user.name=admin
spring.security.user.password=geheim
```

  oder in **SecurityConfig** (nächste Folie) oder mit **eigenen Klassen** (später)

- Erklärungen siehe

  https://agile-coding.blogspot.com/2020/12/spring-security-starter.html

Benutzer mit Rolle erstellen

in **SecurityConfiguration.java**

```
@Configuration
public class SecurityConfiguration {
    @Bean
    public UserDetailsService users(@Autowired PasswordEncoder pwEnc){
        UserDetails user = User.builder()
            .username("user")
            .password(pwEnc.encode("top"))
            .roles("USER")
            .build();
        UserDetails admin = User.builder()
            .username("admin")
            .password(pwEnc.encode("secret"))
            .roles("USER", "ADMIN")
            .build();
        return new InMemoryUserDetailsManager(user, admin);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Beispiel SecurityFilterChain

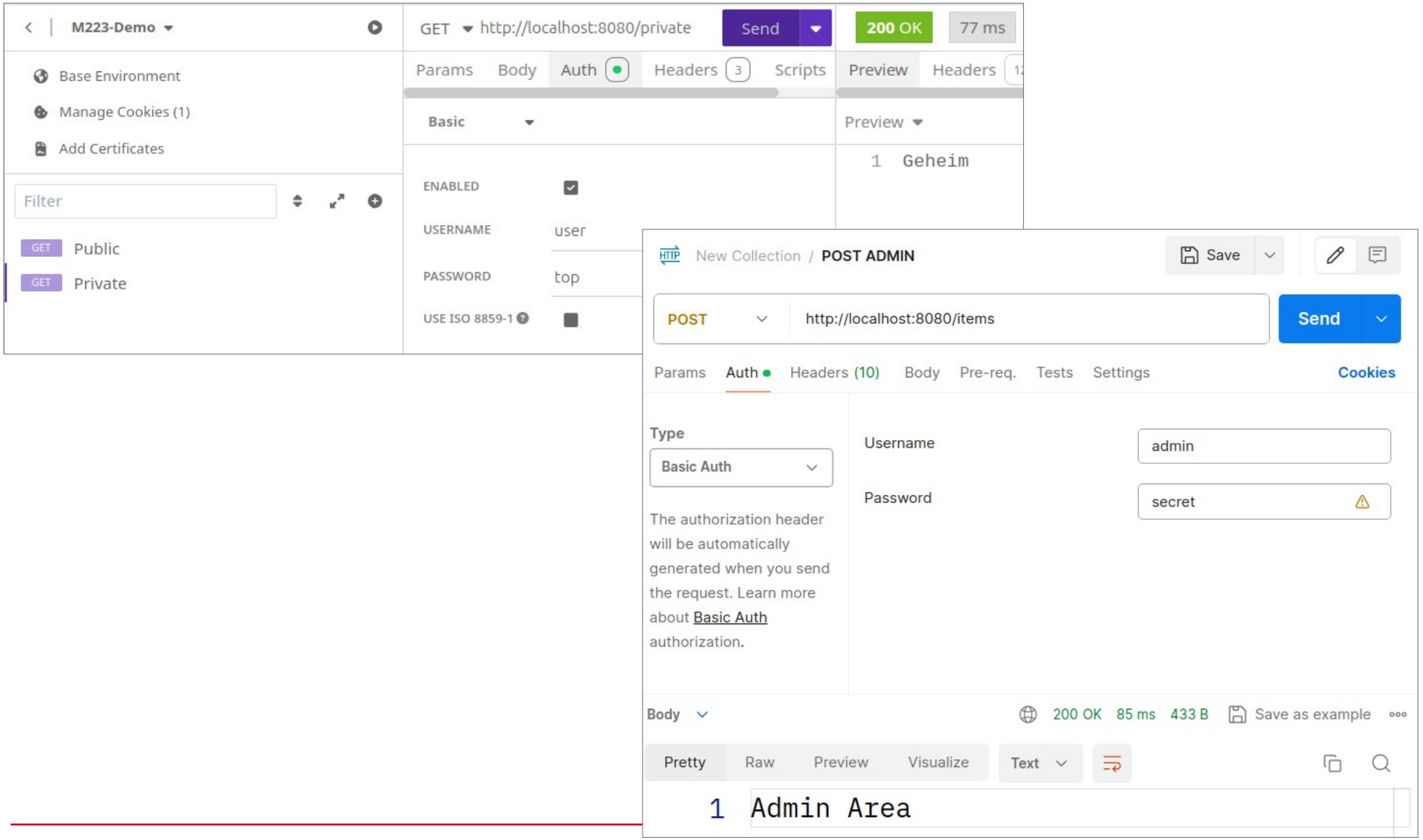
mit öffentlichen und geschützten Bereich(en)

https://docs.spring.io/spring-security/reference/servlet/authorization/authorize-http-requests.html

```
@Configuration
@EnableWebSecurity
public class SecurityConfiguration {
  /* ... */
  private static final String[] EVERYONE = {"/public"};


  @Bean
  public SecurityFilterChain filterChain(HttpSecurity http)
     throws Exception {

    http.csrf(csrf -> csrf.disable()) //disable Cross-Site Request Forgery (CSRF) prevention
     .cors(Customizer.withDefaults()) //configure CORS: Cross Origin Request Sharing
     .authorizeHttpRequests(auth -> {
          auth.requestMatchers(HttpMethod.POST, "/items").hasRole("ADMIN");
          auth.requestMatchers(EVERYONE).permitAll()
               .anyRequest().authenticated(); } )

     .formLogin(Customizer.withDefaults())  //für Login-Form im Browser
     .httpBasic(Customizer.withDefaults()); // für CURL, Postman, Insomnia
     return http.build();
   }
}
```

- der hier gezeigte Code entspricht dem Spring Security 7.0 Coding Standard
- Erklärungen siehe https://agile-coding.blogspot.com/2022/09/spring-security-roles.html
- CSRF: https://www.baeldung.com/spring-security-csrf
- CORS: https://www.baeldung.com/spring-cors

Test mit Insomnia oder Postman



Alles klar?

Persistente RBAC + Datenmodel mit JPA

- Annotationen
  - @Entity
  - @Id
  - @ManyToOne
  - @ManyToMany
- JPA-Dependency aktivieren
- MySQL-Zugangsdaten in application.properties konfigurieren:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://127.0.0.1:3306/wiss_quiz
spring.datasource.username=root
spring.datasource.password=example
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```java
@Entity
@Setter @Getter @NoArgsConstructor
public class User {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Integer id;

  @NotBlank
  private String username;
  @NotBlank
  private String email;
  @NotBlank
  private String password;

  public User(String name, String email, String password) {
     this.username = name;
     this.email = email;
     this.password = password;
  }

  @ManyToMany(fetch = FetchType.LAZY) // N:M Mapping
  private Set<Role> roles = new HashSet<>();
}
```

Implementierung: ERole und Role

```java
public enum ERole {
    ROLE_USER,
    ROLE_MODERATOR,
    ROLE_ADMIN
}
```

```java
@Entity @Getter @Setter @NoArgsConstructor
public class Role {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Integer id;

    @Enumerated(EnumType.STRING)
    @Column(length = 20)
    private ERole name;

    public Role(ERole name) {
      this.name = name;
    }
    public String toString(){
      return name.toString();
    }
}
```

Session-basiert vs. Token-basierte AUTH²

- https://youtu.be/UBUNrFtufWo
- Erarbeitet Euch anhand dieser Quelle: https://www.geeksforgeeks.org/session-vs-token-based-authentication/
  ein grundlegendes Verständnis für die Abläufe, sowie die Vor- und Nachteile der beiden Verfahren
- siehe Sidequest 3B
- Zeit: 30 Minuten