

Softwareentwicklung (SW)

Web Security mit Spring Security

Prof. Dr. Alixandre Santana
alixandre.santana@oth-regensburg.de

Wintersemester
2024/2025

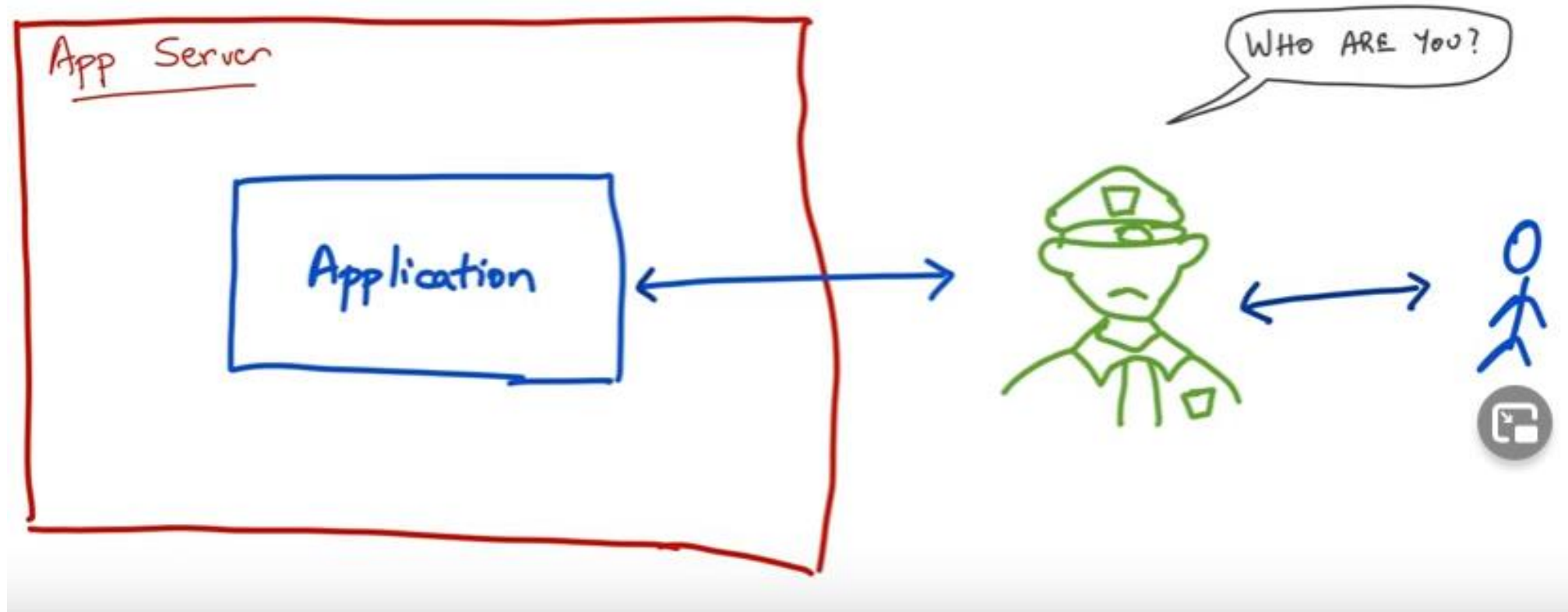
- Die Konzepte von Authentication und Authorization zu beschreiben
- Die Konzepte von Rollen und Authorities zu beschreiben
- Die Architektur des Spring Security Framework zu erklären
- Ein Beispiel von Login mit Spring Security and JDBC zu implementieren

1. Authentication und Authorization zu beschreiben
2. Rollen und Authorities zu beschreiben
3. Komponenten des Spring Security Framework
4. Beispiel

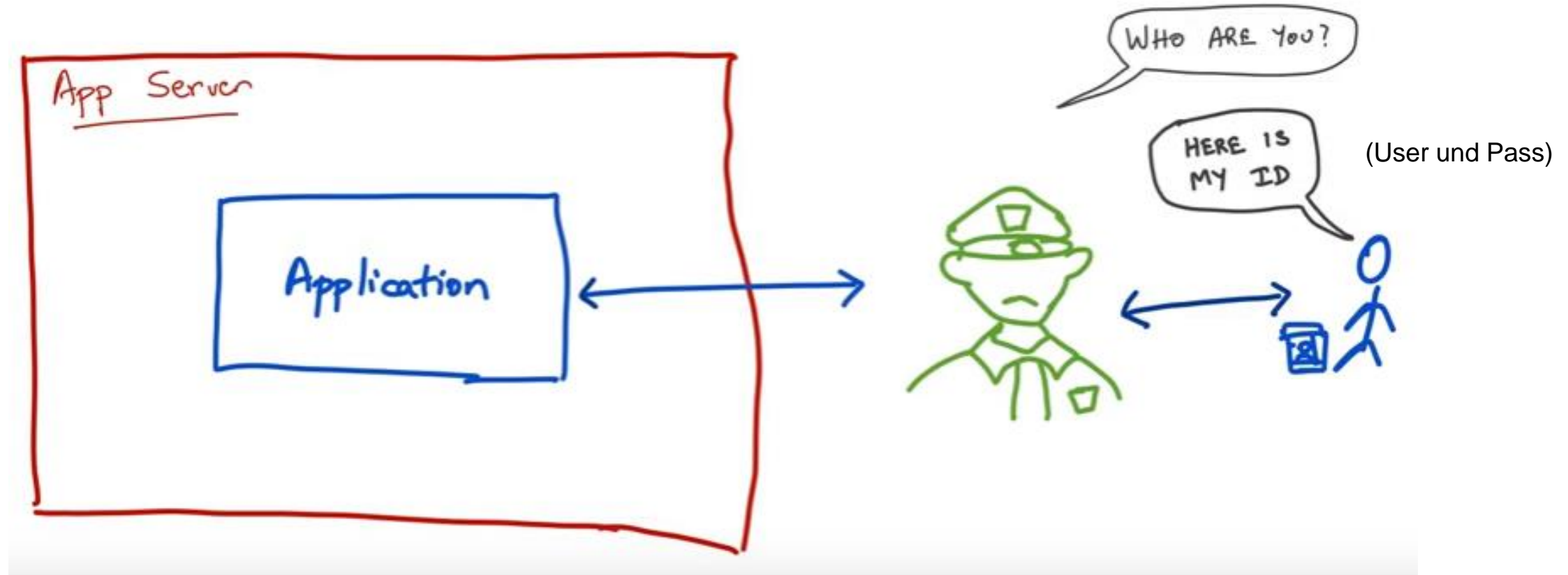
1- Allgemeine Arten der Authentifizierung

- Knowledge base authentication (User and Password)
- Possession based authentication (Mobile Confirmation)

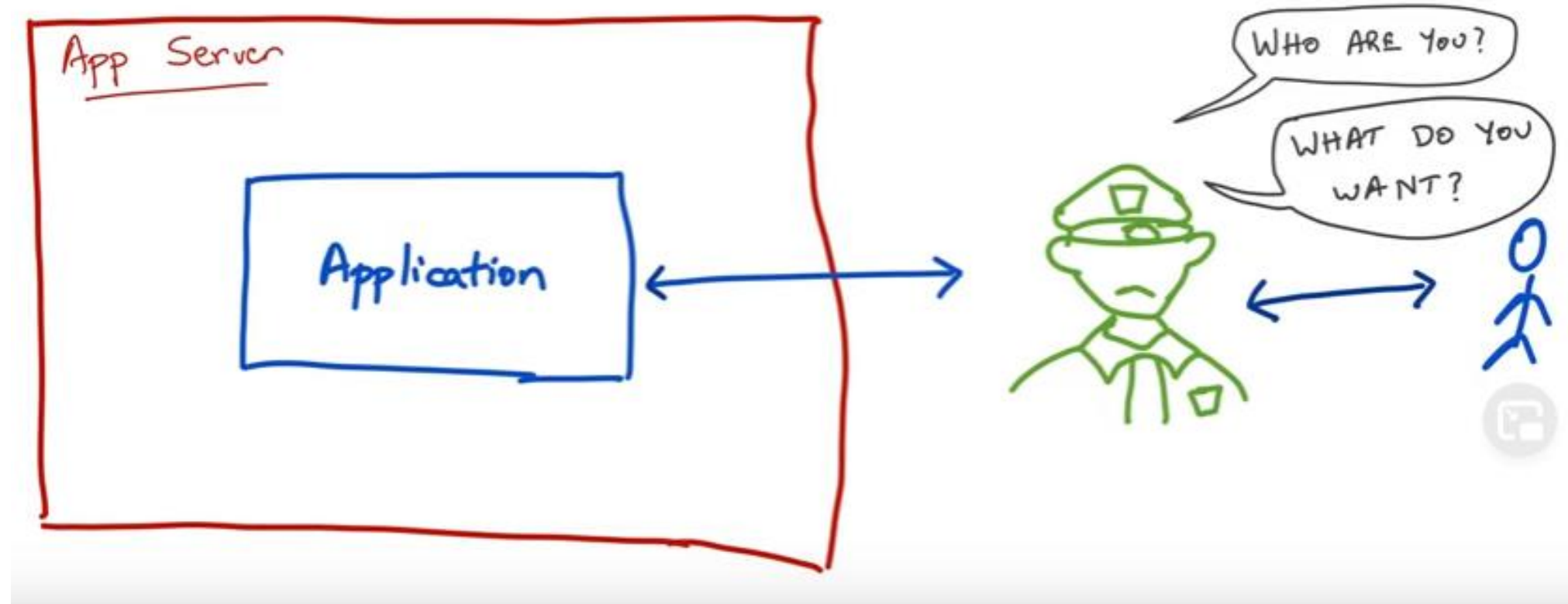
1. 1 Authentifizierung...



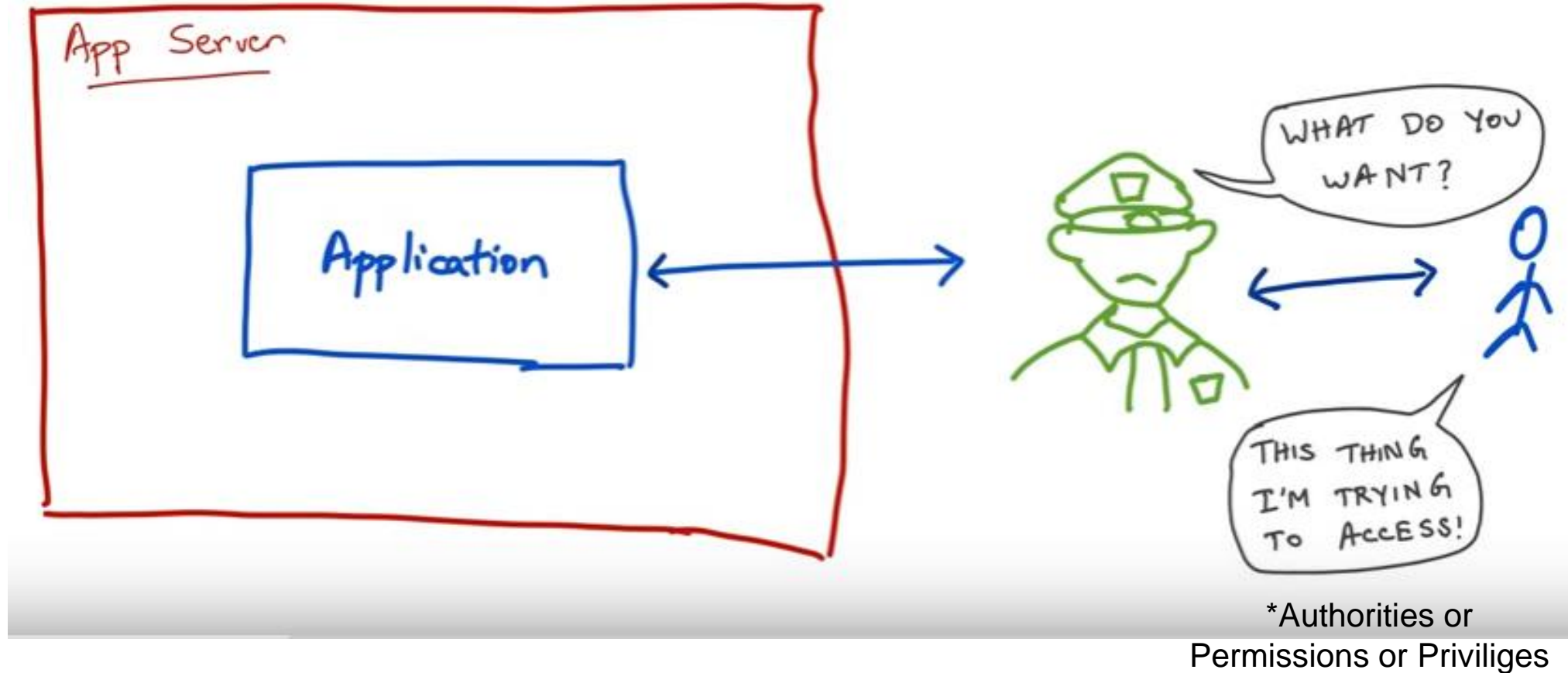
1. 1 Authentifizierung...



1.2 Authorisierung



1.2 Authorisierung



1. Daher...

Authentifizierung...

Wer ist der Benutzer?

Authorisierung...

Was will er? Darf er das tun?

1. Authentication und Authorization zu beschreiben
2. Rollen und Authorities zu beschreiben
3. Komponenten des Spring Security Framework
4. Beispiel

2.1 Roles

- Kann **eine Reihe von Autoritäten** enthalten

Global roles

Role	Overall		Slave		Job						Run			View			SCM	
	Administer	Read	Configure	Delete	Create	Delete	Configure	Read	Build	Workspace	Delete	Update	Artifacts	Create	Delete	Configure	Tag	
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
anonymous	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
job-creator	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Role to add

Add

2.2 Authorities

- Berechtigung auf granularer Ebene

Role	Overall	
	Administer	Read
admin	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Authority

Authority

1. Authentication und Authorization zu beschreiben
2. Rollen und Authorities zu beschreiben
3. Komponenten des Spring Security Framework
4. Beispiel

3.1 Spring Security

- Spring Security ist ein Framework, das **Authentifizierung, Autorisierung und Schutz vor gängigen Angriffen** bietet
- Es ist der De-facto-Standard zur Sicherung von Spring-basierten Anwendungen

```
<!-- Spring security -->
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
</dependency>
```

3.1 Filters

- Das erste, was Ihrer Anwendung hinzugefügt wird, wenn wir Spring Security verwenden, ist ein **Security Filter**....
- Normalerweise sendet der Client eine Anfrage an die Anwendung und der Container erstellt eine **FilterChain**, die die **Filterinstanzen** und das **Servlet** enthält, die die HttpServletRequest basierend auf dem Pfad des Anfrage-URI verarbeiten sollen.
- Der Entwickler sollte jedoch einige vom Filter verwendete **Informationen angeben!**



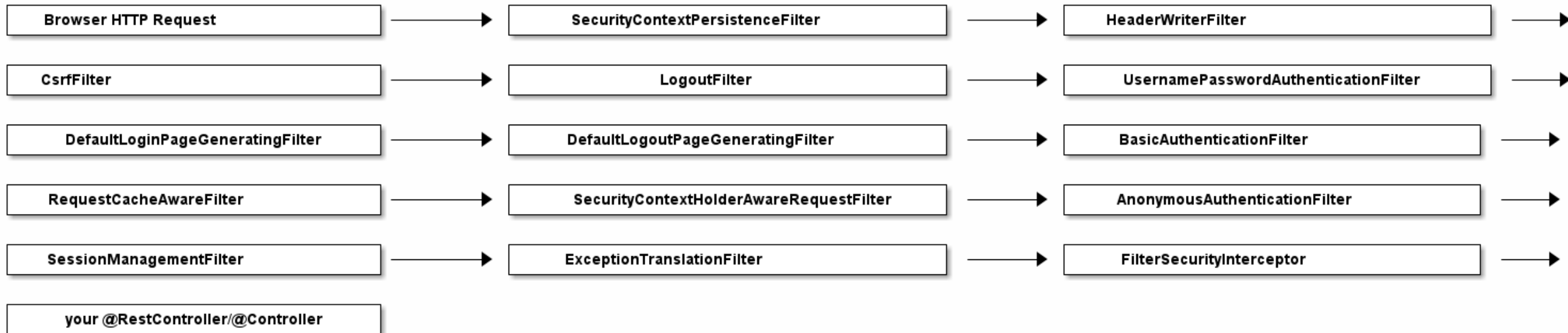
3.1 Security Filters Chain

- **Nur zu wissen, gibt es mehrere Filter..**

- `org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter`
- `org.springframework.security.web.context.SecurityContextPersistenceFilter`
- `org.springframework.security.web.header.HeaderWriterFilter`
- `org.springframework.security.web.authentication.logout.LogoutFilter`
- `org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter`
- `org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter`
- `org.springframework.security.web.authentication.AnonymousAuthenticationFilter`
- `org.springframework.security.web.session.SessionManagementFilter`
- `org.springframework.security.web.access.ExceptionTranslationFilter`
- `org.springframework.security.web.access.intercept.FilterSecurityInterceptor`

3.1 Security Filters Chain

- Nur zu wissen, gibt es mehrere Filter..



3.1 Security Filters Chain

- Nur zu wissen, gibt es mehrere Filter..

Diese Filter stammen zum größten Teil von Spring Security.

Sie erledigen die ganze Arbeit.

Sie müssen nur noch **konfigurieren** wie sie funktionieren,
d. h. :

- **welche URLs geschützt, welche ignoriert**
- **und welche Datenbanktabellen zur Authentifizierung verwendet werden sollen**

3.1 Security Filter - Configuration

- Der Entwickler sollte jedoch einige vom Filter verwendete **Informationen angeben!**

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(Customizer.withDefaults())
            .authorizeHttpRequests(authorize -> authorize
                .anyRequest().authenticated()
            )
            .httpBasic(Customizer.withDefaults())
            .formLogin(Customizer.withDefaults());
        return http.build();
    }
}
```

Der Code auf der linken Seite erstellt die folgenden Filter ...

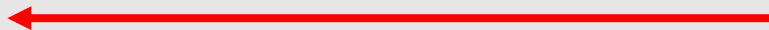
Filter	Added by
CsrfFilter	HttpSecurity#csrf
UsernamePasswordAuthenticationFilter	HttpSecurity#formLogin
BasicAuthenticationFilter	HttpSecurity#httpBasic
AuthorizationFilter	HttpSecurity#authorizeHttpRequests

3.2 Wovon bekommt man User and Password Daten?

- „Spring Security“ bietet zahlreiche Möglichkeiten an, um externe Identity Provider und unterschiedliche Protokolle zu nutzen und einzubinden

- LDAP
- oAuth 2.0
- OpenID
- JDBC
- HTTP Basic
- ...

Wir verwenden die in
unserer Datenbank
gespeicherten Benutzer-
und Passwortinformationen!



3.3 Nach dem Import der Spring Security Dependency...



- Wenn die Anwendung die Spring Security Dependency verwendet, werden standardmäßig alle Anfragen gefiltert und der Benutzer wird **auf eine Anmeldeseite umgeleitet**.
- Die Spring-Security erstellt einen Standardbenutzer („user“) und ein Standardkennwort, das in der Konsole gedruckt wird.
- Von nun an sollten wir diese Anmeldeinformationen angeben, um auf unsere Anwendung zugreifen zu können.

```
2023-11-18T13:10:32.836+01:00 WARN 11528 --- [ restartedMain] .s.s.UserDetailsServiceAutoConfiguration :
```

```
Using generated security password: e31401a6-4a43-40a2-932a-791b71df8f75
```

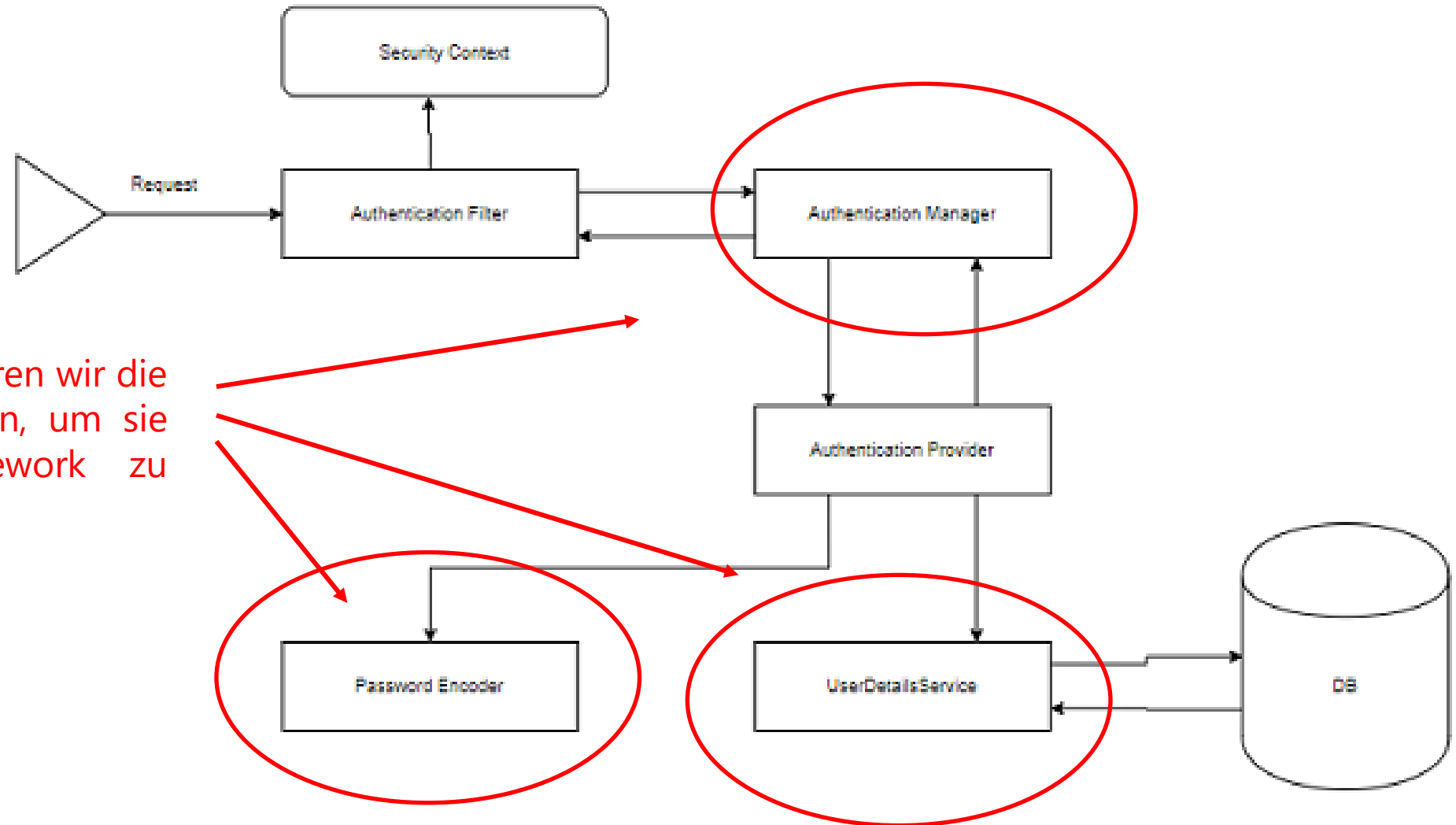
```
This generated password is for development use only. Your security configuration must be updated before running your application i
```

```
2023-11-18T13:10:33.068+01:00 INFO 11528 --- [ restartedMain] o.s.s.web.DefaultSecurityFilterChain : Will secure any request
```

3.4 Allgemeine Begriffe

- **Authentication** Vorgang/Ergebnis der Echtheitsprüfung (durch ein geteiltes „Geheimnis“), dt. „Authentifizierung“
- **Authorization** Zuweisen/Überprüfen von Zugriffsrechten dt. „Autorisation“
- **Principal** Einheit (User, Server, ...), die authentifiziert werden kann
- **Role** Rolle eines Principal, die (direkt oder indirekt) für die Zuweisung von Zugriffsrechten steht
- **Authority** Privileg/Recht, etwas zu tun; „feingranularer“ als eine Rolle; in Spring: „GrantedAuthority“
- **Identity Provider** Eigenständiger Authentifizierungsdienst

3.5 Gebrauchte Komponenten



In unserem Projekt codieren wir die hervorgehobenen Klassen, um sie in das Spring Framework zu integrieren

3.5 Gebrauchte Komponenten



- **AuthenticationFilter**

Dies ist der Filter, der Anfragen abfängt und versucht, sie zu authentifizieren. In Spring Security konvertiert er **die Request in ein Authentifizierungsobjekt** und delegiert die Authentifizierung an den AuthenticationManager.

- **AuthenticationManager**

Es ist die Hauptstrategieschnittstelle für die Authentifizierung. Sie verwendet die Methode `authenticate()`, um die Request zu authentifizieren. Die Methode `authenticate()` führt die Authentifizierung durch und gibt bei erfolgreicher Authentifizierung ein **Authentifizierungsobjekt** zurück oder löst bei fehlgeschlagener Authentifizierung eine **AuthenticationException** aus. Der Authentifizierungsprozess wird in diesem Prozess an den **AuthenticationProvider** delegiert.

- **Authentication Provider**

Hier findet die Authentifizierung statt. Wenn der Authentifizierungstyp unterstützt wird, wird der Authentifizierungsprozess gestartet. Hier kann diese Klasse die Methode `loadUserByUsername()` der **UserDetailsService**-Implementierung verwenden. Wenn der Benutzer nicht gefunden wird, kann eine `UsernameNotFoundException` ausgelöst werden.

3.5 Gebrauchte Komponenten

- UserDetailsService

Es ist eine der Kernschnittstellen von Spring Security. Die Authentifizierung einer Anfrage hängt hauptsächlich von der Implementierung der Schnittstelle UserDetailsService ab. **Sie wird am häufigsten bei der datenbankgestützten Authentifizierung zum Abrufen von Benutzerdaten verwendet.** Die Daten werden mit der Implementierung der Methode **loadUserByUsername()** **abgerufen**, in der wir unsere Logik zum Abrufen der Benutzerdetails für einen Benutzer bereitstellen können.

- PasswordEncoder

Bis Spring Security 4 war die Verwendung von PasswordEncoder optional. Der Benutzer konnte Passwörter im Klartext mithilfe der In-Memory-Authentifizierung speichern. Spring Security 5 schreibt jedoch die Verwendung von PasswordEncoder zum Speichern von Passwörtern vor. Dieser kodiert das Passwort des Benutzers mithilfe einer seiner zahlreichen Implementierungen. **Die gebräuchlichste Implementierung ist der BCryptPasswordEncoder.**

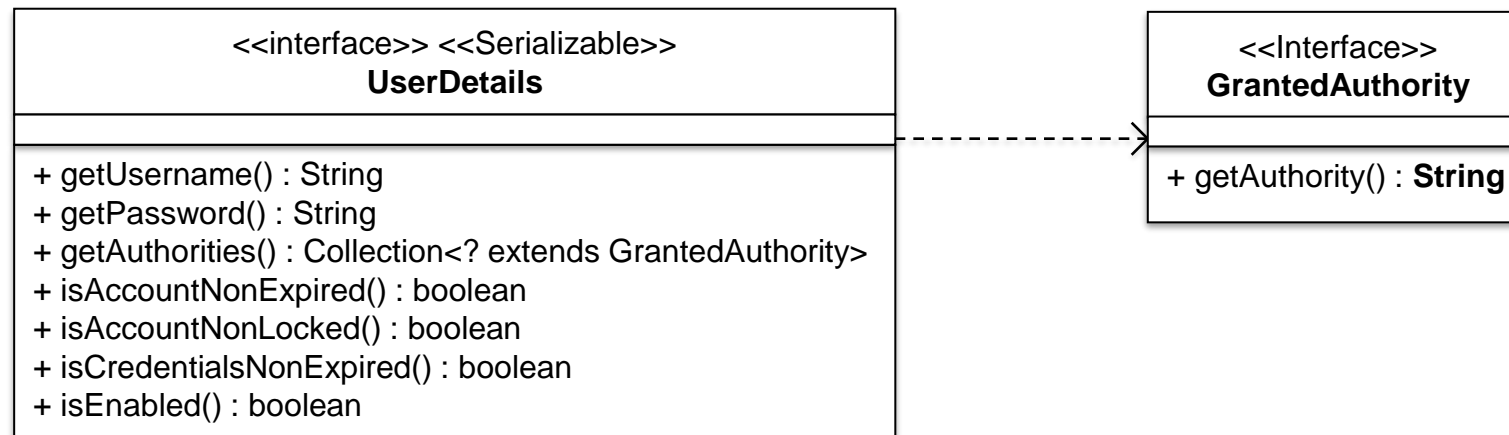
- Spring SecurityContext

Hier werden die Details des aktuell authentifizierten Benutzers nach erfolgreicher Authentifizierung gespeichert. **Das Authentifizierungsobjekt ist dann während der gesamten Anwendung für die Sitzung verfügbar.** Wenn wir also den Benutzernamen oder andere Benutzerdetails benötigen, müssen wir den SecurityContext abrufen. first.

3.5 Gebrauchte Komponenten - UserDetails

- Sofern kein externer Identity Provider integriert wird, können eigene Entity-Klassen die Prinzipal-Attribute speichern
- Für die Attribute *Username*, *Password* (verschlüsselt) und die zugehörigen Authorities werden von entsprechenden Entity-Klassen die Interface **UserDetails** und **GrantedAuthority** genutzt/implementiert

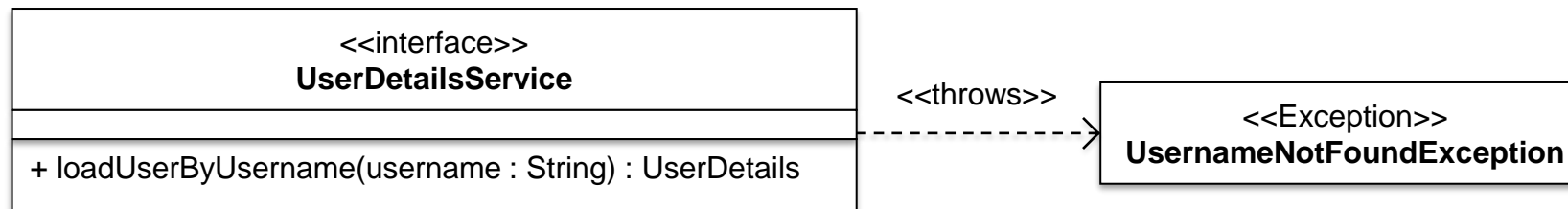
Interfaces UserDetails und GrantedAuthority



3.5 Implementierung eines „Identity Providers“

- Das Interface **UserDetailsService** definiert die API eines minimalen “Identity Provider” und wird durch das Security-Framework automatisch genutzt
- Über dieses Interface können die **UserDetails** in der eigenen Anwendung abgerufen werden
- Die Überprüfung des Passworts findet ausschließlich im Security-Framework statt und wird in der Anwendung nicht implementiert

Interface UserDetailsService



1. Authentication und Authorization zu beschreiben
2. Rollen und Authorities zu beschreiben
3. Komponenten des Spring Security Framework
4. Beispiel

4. Beispiel eines eigenen „Identity Provider“

- Im nachfolgenden Beispiel ist der Entitäts-Typ Kunde der Principal (Kunden sind Nutzer der einer Applikation und loggen sich ein)
- Der Service `MyUserDetailsServiceImpl` übernimmt hier zusätzlich die Funktion des *Identity Provider* (die Userdaten inkl. verschlüsseltem Passwort werden in diesem Beispiel mittels JPA in einer relationalen Datenbank gespeichert)

```
@Service
public class MyUserDetailsServiceImpl implements UserDetailsService{

    UserRepositoryI userRepository;
    public MyUserDetailsServiceImpl (UserRepositoryI userRepository) {
        this.userRepository= userRepository;
    }
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        // TODO Auto-generated method stub
        Optional<User> oUser= userRepository.findByLoginIgnoreCase(username);
        oUser.orElseThrow(()-> new UsernameNotFoundException("Not found "+username));
        System.out.println("User found at the UserDetailsService="+ oUser.get().getLogin());
        return new MyUserDetails(oUser.get());
    }
}
```

weitere Methode/Erweiterung ein paar Folien später!

4.1 SecurityConfig.java - Hilfsklasse zum „Passworthashen“


- Passwörter werden immer verschlüsselt („secure hash“) gespeichert
- Für die Verschlüsselung wird eine PasswordEncoder-Implementierung des Security-Frameworks genutzt

Interface PasswordEncoder

<<interface>> PasswordEncoder	
+ encode(pwd : CharSequence) : String + matches(pwd1 : CharSequence, pwd2 : String) : boolean	

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return PasswordEncoderFactories.createDelegatingPasswordEncoder();
    }
}
```



Klassen können sich diesen Encoder injizieren lassen (@Autowired)

4.2 SecurityConfig.java - SecurityFilterChain

```
@Bean
public SecurityFilterChain getSecurityFilterChain(HttpSecurity http,
HandlerMappingIntrospector introspector) throws Exception {

    MvcRequestMatcher.Builder mvcMatcherBuilder = new MvcRequestMatcher.Builder(introspector);

    http.csrf().disable();
    http.csrf(csrfConfigurer ->
csrfConfigurer.ignoringRequestMatchers(new AntPathRequestMatcher("/h2-console/**")));

    http.headers(headersConfigurer ->
headersConfigurer.frameOptions(HeadersConfigurer.FrameOptionsConfig::sameOrigin));

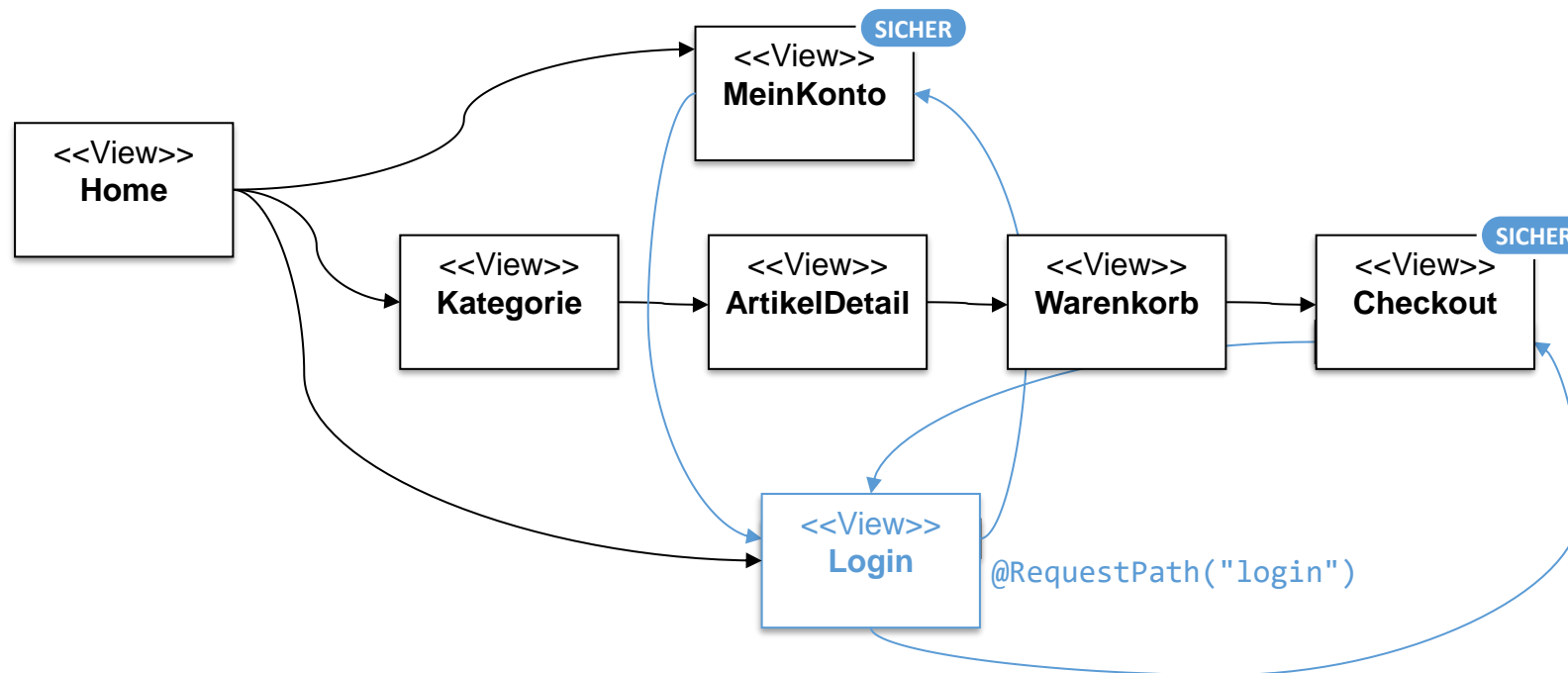
    http.authorizeHttpRequests(auth ->
auth
    .requestMatchers(new AntPathRequestMatcher("/h2-console/**")).permitAll()
    .requestMatchers(new AntPathRequestMatcher("/login")).permitAll()
    .requestMatchers(new AntPathRequestMatcher("/logout")).permitAll());
    http.authorizeHttpRequests()
    .requestMatchers(new AntPathRequestMatcher("/registration/**")).hasAuthority("REGISTRATION")
    .requestMatchers(new AntPathRequestMatcher("/student/add")).hasAuthority("CREATE_STUDENT")
    .requestMatchers(new AntPathRequestMatcher("/student/all")).hasAuthority("LIST_STUDENT");
    //andere URLs....
    http.headers(headers -> headers.frameOptions(FrameOptionsConfig::disable));
    http.formLogin(Customizer.withDefaults());
    http.httpBasic(Customizer.withDefaults());

    return http.build();
}
```

4.3 Web-Security wird „konfigurativ“ genutzt

- Für „Request-Pfade“ (zu Views) wird eine Konfiguration festgelegt, ob für deren Aufruf eine Authentifizierung nötig ist
- Die „Umleitung“ zu einer Login-View erfolgt automatisch

Pageflow-Beispiel



4.4 Beispiel Login-View

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:url value="/Login" var="loginUrl"/>

<h2> Custom login page</h2>
<form action="${loginUrl}" method="post" >
  <c:if test="${param.error != null}">
    <p>
      Invalid username and password.
    </p>
  </c:if>
  <c:if test="${param.logout != null}">
    <p>
      You have been logged out.
    </p>
  </c:if>
  <p>
    <label for="username">Username</label>
    <input type="text" id="username" name="username"/>
  </p>
  <p>
    <label for="password">Password</label>
    <input type="password" id="password" name="password"/>
  </p>
  <button type="submit" class="btn">Log in</button>
</form>
```

Dieser Action-Link wird nur dann genutzt/ausgeführt, sofern der Login direkt aufgerufen wurde. Sofern das Security-Framework einen Login vor einen Seitenaufruf mit nötiger Authentifizierung einbauen muss, wird der ursprünglich gewählt Request-Pfad ausgeführt

Die name-Attribute der <input>-Felder lauten entsprechend **username** und **password** (analog dem Interface UserDetails)

4.4 Beispiel - Empfohlene Reihenfolge

- 1- Definieren Sie Benutzer (Profile) und Ressourcen (URLs)
- 2-Erstellen Sie Modelle und Tabellen für sie
- 3-Erstellen Sie das UserRepository
- 4-Erstellen Sie MyUserDetails und MyUserDetailsService
- 5-Erstellen Sie die SecurityConfiguration
- 6-Erstellen Sie den HomeController
- 7- Erstellen Sie verschiedene Home-Ansichten für jedes Profil (Thymeleaf-Vorlagen)

4.5 Thymeleaf-Security-Erweiterungen (Optional)

Benötigt weitere Dependency in pom.xml:

```
<dependency>
  <groupId>org.thymeleaf.extras</groupId>
  <artifactId>thymeleaf-extras-springsecurity6</artifactId>
</dependency>
```

Auf Spring-Version
achten!

```
<html xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
  <!-- Auszug -->
  <div sec:authorize="isAuthenticated()">gerendert, falls Principal authentifiziert</div>

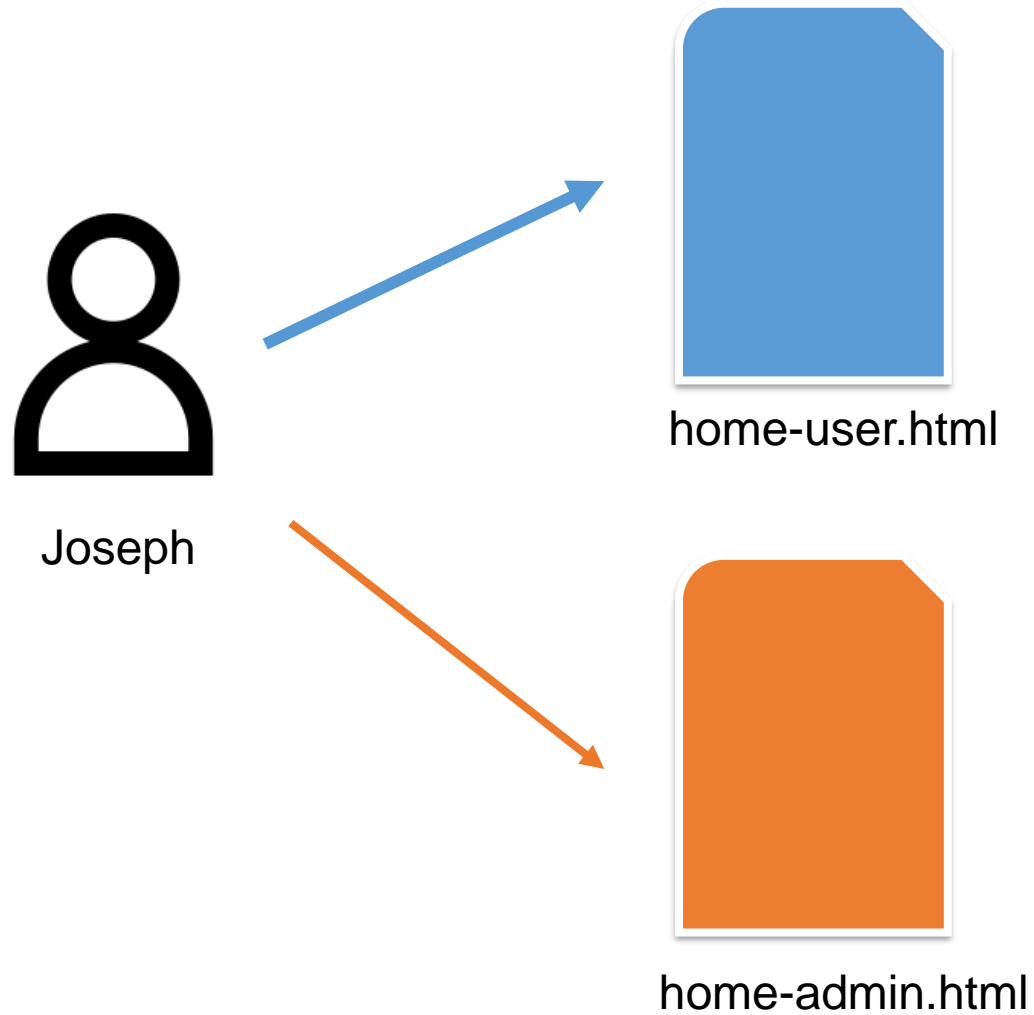
  <div sec:authorize="hasRole('IRGENDEINE_ROLLE')">gerendert, falls Rolle zugeordnet</div>
  <div sec:authorize="hasAuthority('IRGENDEINE_AUTHORITY')">gerendert, falls Authority zugeordnet</div>

  <div sec:authorize="hasAnyRole('ROLLE_1', 'ROLLE_2', 'ROLLE_3')">wie oben</div>
  <div sec:authorize="hasAnyAuthority('AUTHORITY_1', 'AUTHORITY_2')">wie oben</div>

  <div sec:authorize="isAnonymous()">gerendert für "anonymen User"</div>

  <span sec:authentication="name"/> <!-- Gibt den Username des eingeloggten Principal aus -->
  <span sec:authentication="principal.authorities"/> <!-- Gibt die Namen der GrantedAuthorities aus -->
```

4.5 Thymeleaf-Specific Views for different Profiles



Referenzen

<https://www.marcobehler.com/guides/spring-security>

<https://www.baeldung.com/spring-security-csrf>

<https://www.baeldung.com/spring-security-thymeleaf>