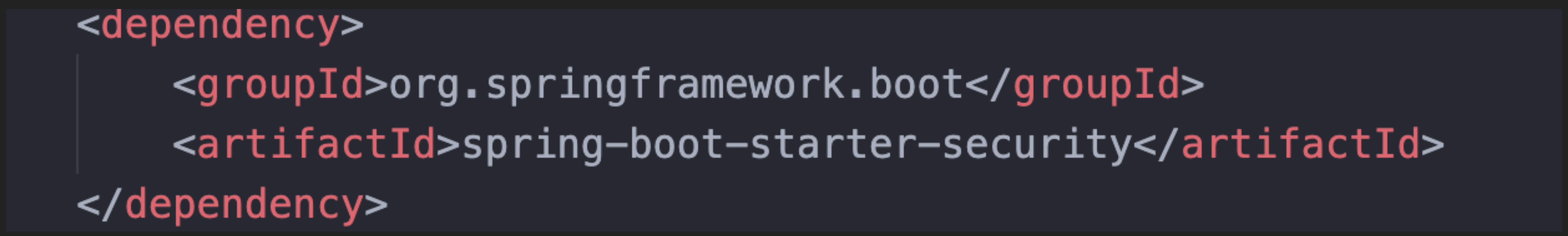
CPAN-252 WEB APPLICATION DEVELOPMENT FOR CPAN DIPLOMA.   PROF. ANTON KOVUNOV

# SPRING SECURITY

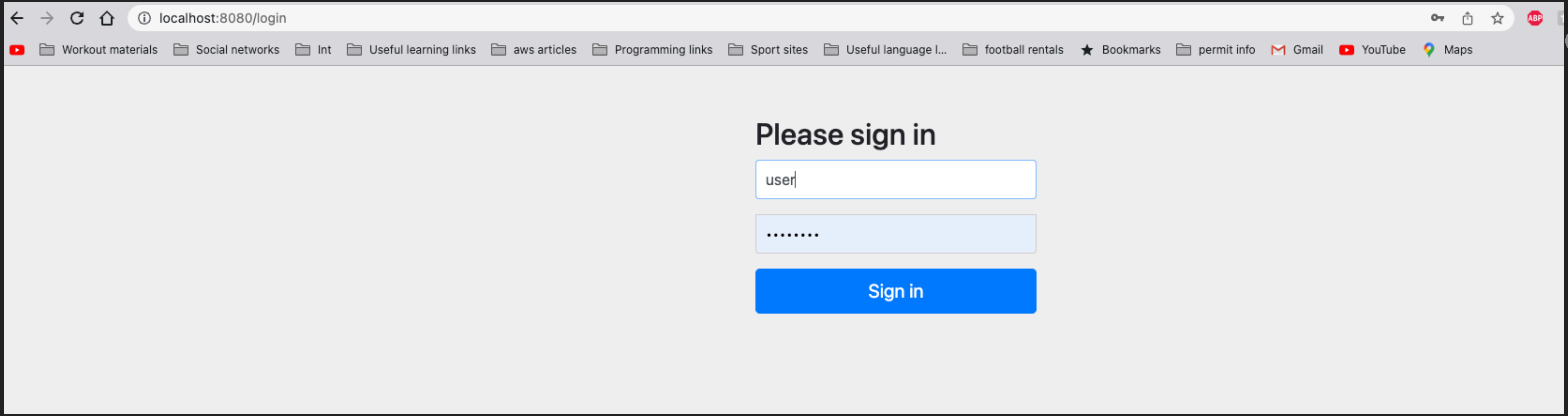
## SECURITY IN MODERN APPLICATIONS

- Enterprises and startups are heavily investing in providing multiple layer security in their applications, and Spring has a lot of tools for providing security in applications
- Adding security to Spring Boot app is starting by adding dependency to pom.xml file



```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

## SPRING SECURITY



Please sign in

user

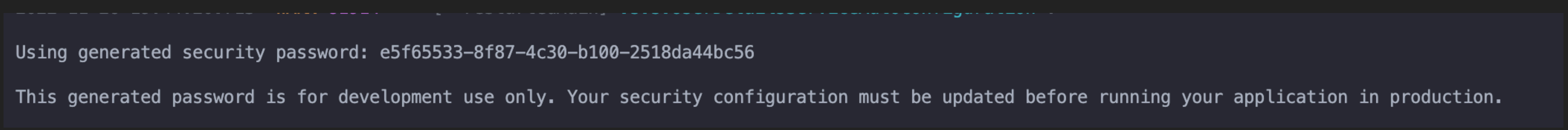
•••••••••

Sign in

e

## SPRING SECURITY

- Spring provides some default credentials for us, username is user and password is automatically generated during application startup, you can see it in application startup logs



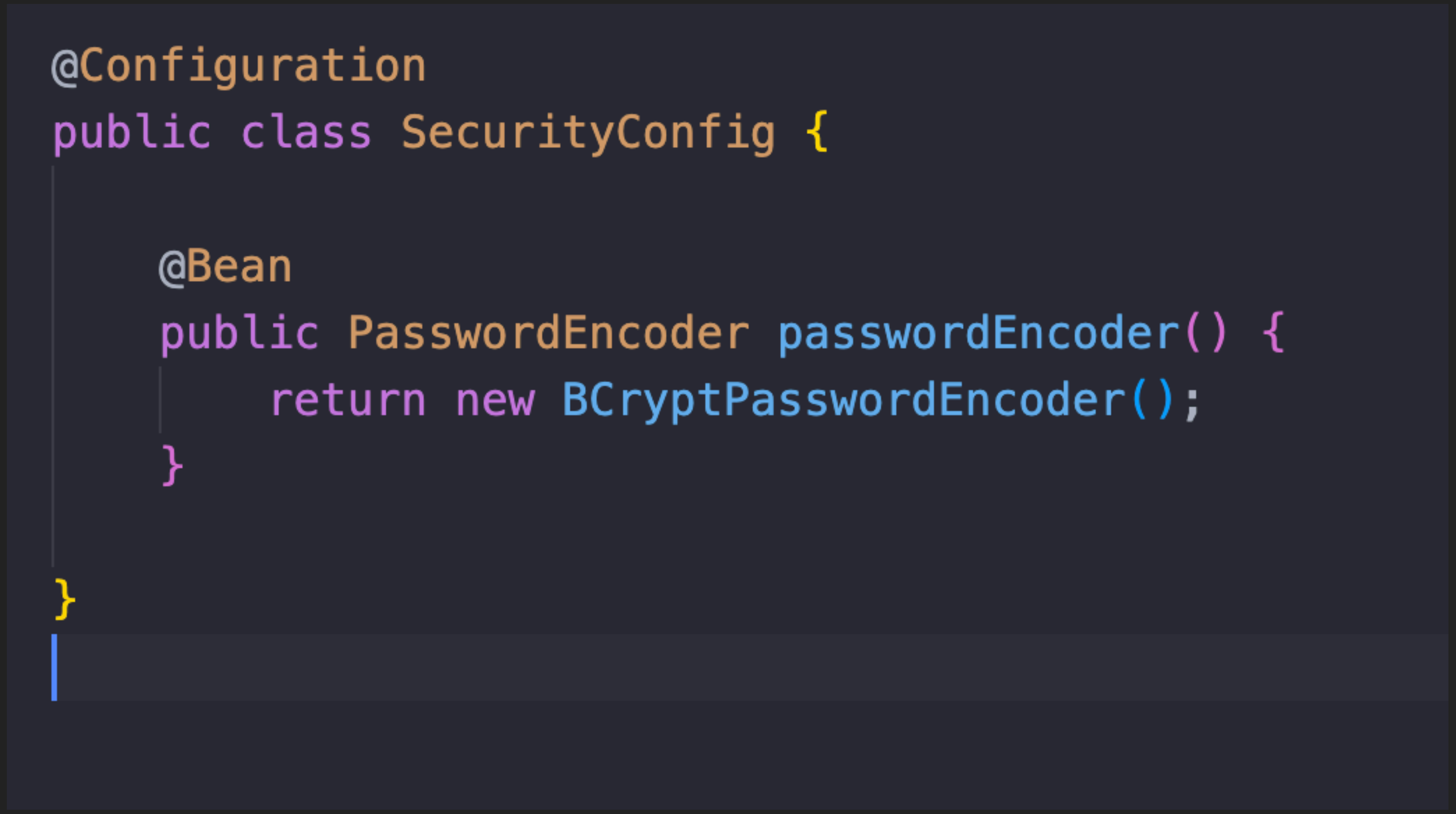
```
Using generated security password: e5f65533-8f87-4c30-b100-2518da44bc56

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

## BASIC SPRING SECURITY

- When we enter login credentials, we will see our application home page
- Basic security provides following features:
  - All HTTP requests paths require authentication
  - No specific user roles are required
  - Authentication is prompted with a simple page
  - There's only one user, username is user

## TEKKENREBORN APP SECURITY

- Let's improve our app security
- First of all we need to create a security bean that will be responsible for holding configuration

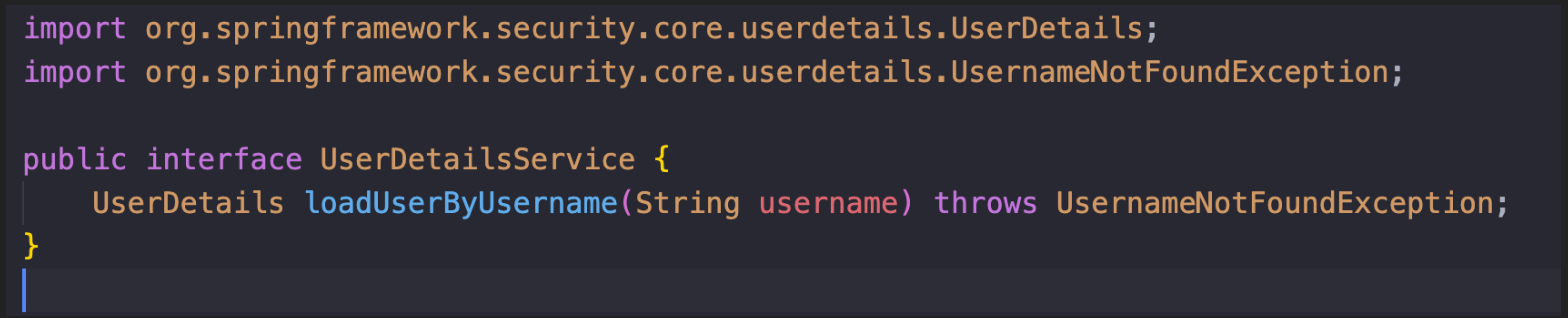


```java
@Configuration
public class SecurityConfig {

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

}
```

## TEKKENREBORN APP SECURITY

- Let’s break down code provided in previous slide:
  - We create a SecurityConfig class annotated with Configuration
  - We create a PasswordEncoder bean, using BCryptPasswordEncoder implementation (Applies bcrypt strong hashing encryption)
  - We use encoder to apply password encryption in the database, so user password is never stored as decrypted and gets encrypted when user logs in and gets compared with encrypted password stored in the database

## USER DETAILS SERVICE

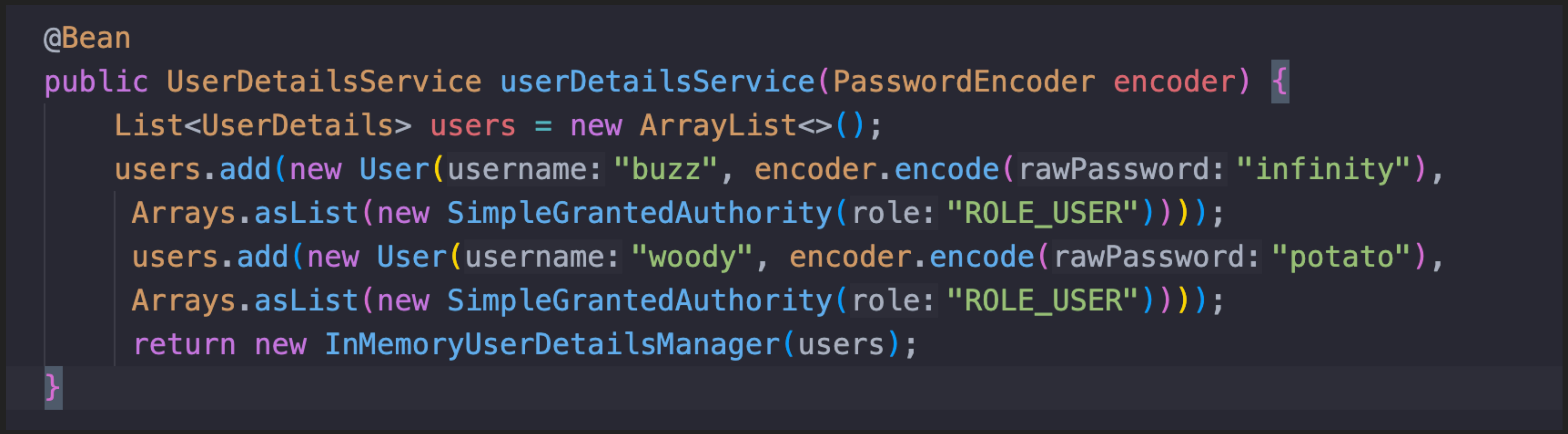


```java
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;

public interface UserDetailsService {
    UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;
}
```

- Spring has implementation for UserDetailsService using in-memory, JDBC and LDAP user store, we will use in memory store
- Also as you see when user can't be found, UsernameNotFoundException gets thrown

## IMPLEMENTATION OF USER DETAILS SERVICE

- We are creating two users with username, encoded password and role, we are using simple User role(it should be SimpleGrantedAuthority class). When we created our users we return InMemoryUserDetailsManager.



```
@Bean
public UserDetailsService userDetailsService(PasswordEncoder encoder) {
    List<UserDetails> users = new ArrayList<>();
    users.add(new User(username: "buzz", encoder.encode(rawPassword: "infinity"),
     Arrays.asList(new SimpleGrantedAuthority(role: "ROLE_USER"))));
     users.add(new User(username: "woody", encoder.encode(rawPassword: "potato"),
     Arrays.asList(new SimpleGrantedAuthority(role: "ROLE_USER"))));
     return new InMemoryUserDetailsManager(users);
}
```

## USER MANAGEMENT IN SPRING

- In-memory user store is good for simple test cases, but it’s not secure and not really extensible
- Now we can use JDBC Store in combination with Spring Data JPA to implement user management
- We will create User entity that will implement UserDetails providing methods needed for security

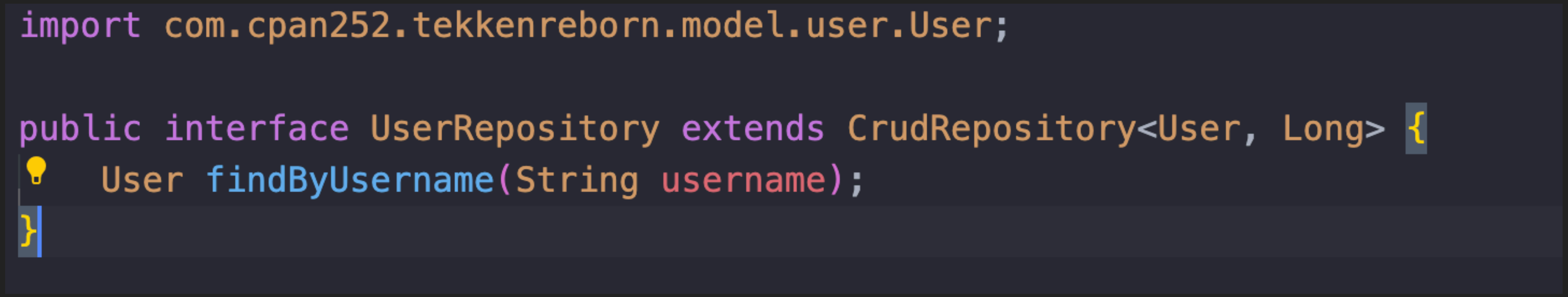
## USER MANAGEMENT IN SPRING

- Implementation of UserDetails provide some essential user data to the framework, most interesting for us are:
  - getAuthorities - returns a collection of user authorities, we will have only user role for this entity
  - Various is* methods provide flags to understand user validity

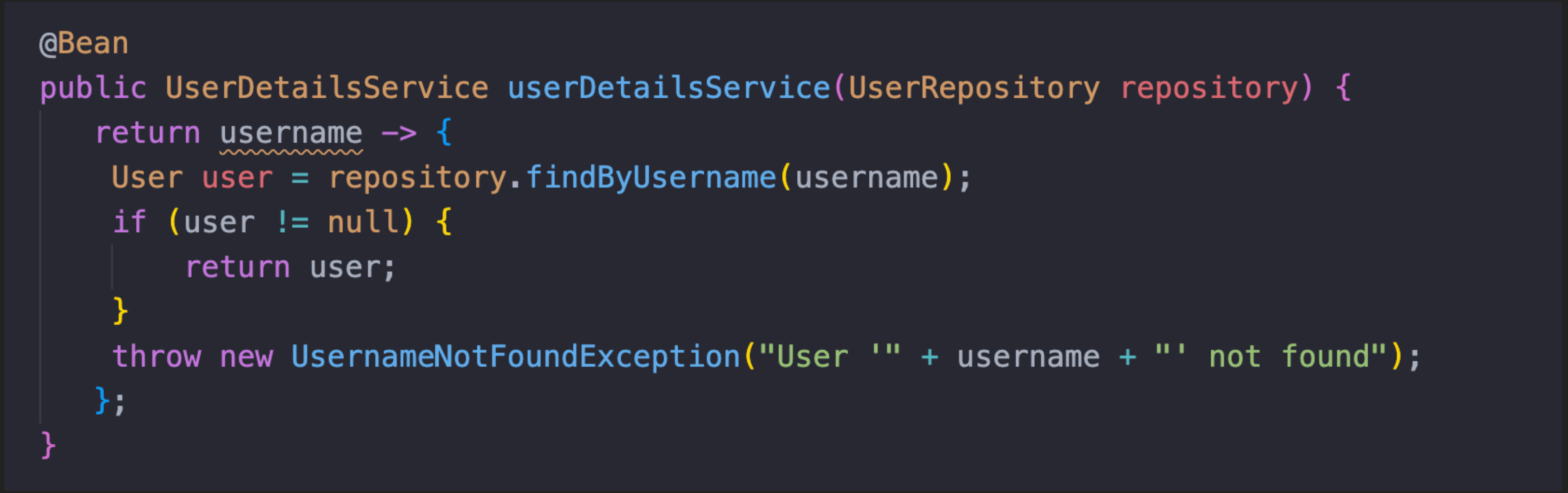
## USER MANAGEMENT IN SPRING

- As entity is created, we need to create repository to do operations with Users

- Next we will implement UserDetailsService, by using user repository



```java
import com.cpan252.tekkenreborn.model.user.User;

public interface UserRepository extends CrudRepository<User, Long> {
    User findByUsername(String username);
}
```



```java
@Bean
public UserDetailsService userDetailsService(UserRepository repository) {
    return username -> {
     User user = repository.findByUsername(username);
     if (user != null) {
         return user;
     }
     throw new UsernameNotFoundException("User '" + username + "' not found");
    };
}
```

## QUESTIONS?

Next we will implement registration for users by providing registration form and registering users in database