Spring Security

rolf.jufer@letsboot.ch

letsboot.ch
swiss dev training

Is Application Security only for those who are afraid to face the Challenges of Cybersecurity?

Facebook-Cambridge Analytica scandal

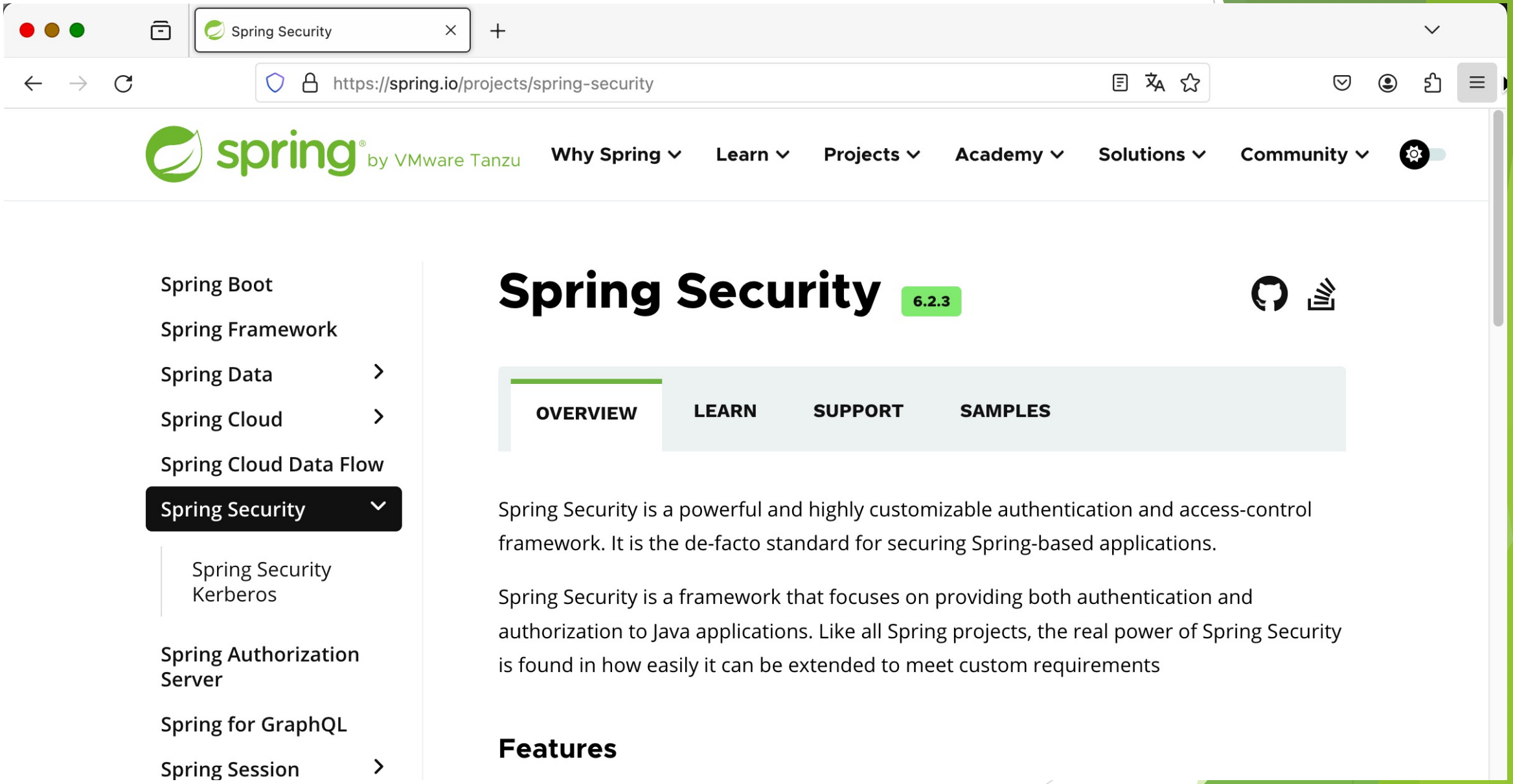
Marriot: three major data leaks within a few years

In 2016, 164 million email addresses and passwords were leaked from LinkedIn.

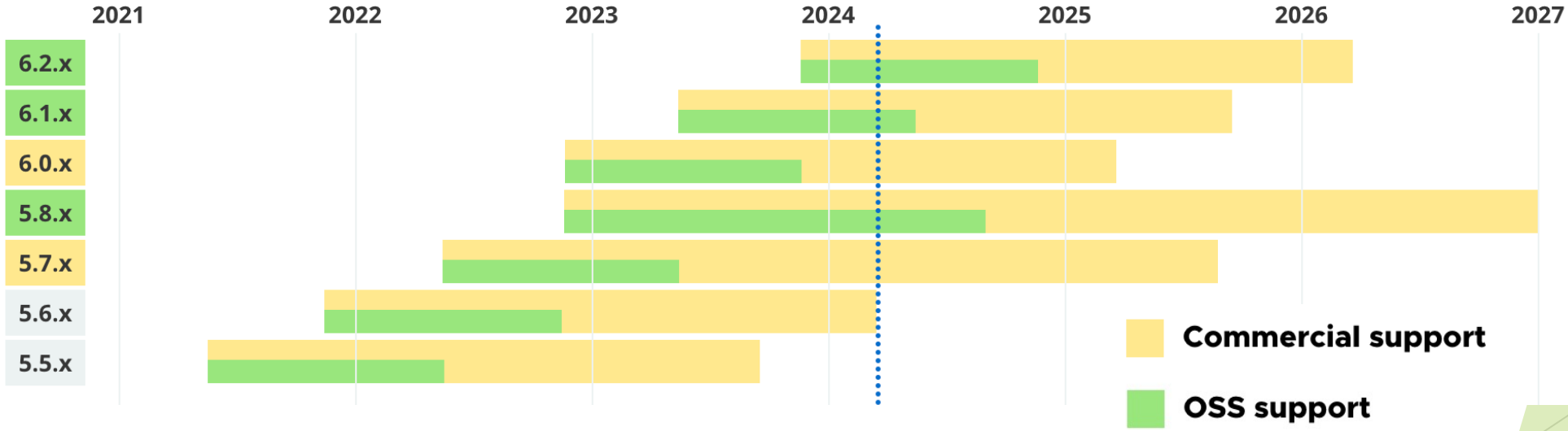
Welcome & Introduction

Notes

- The slides and source code for the demos can be found at https://github.com/rolfjufer/spring-security-jugstalk.
- It's worth noting that the demos are deliberately kept simple. Our aim is **to illustrate** the elements discussed, rather than crafting production-ready code.


Spring Security
https://spring.io/projects/spring-security
spring by VMware Tanzu
Why Spring
Learn
Projects
Academy
Solutions
Community
Spring Boot
Spring Framework
Spring Data
Spring Cloud
Spring Cloud Data Flow
Spring Security
Spring Security Kerberos
Spring Authorization Server
Spring for GraphQL
Spring Session
Spring Security
6.2.3
OVERVIEW
LEARN
SUPPORT
SAMPLES
Spring Security is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.
Spring Security is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements
Features

Spring Security Releases and Support



Sources: *https://spring.io/projects/spring-security#support*
https://tanzu.vmware.com/spring-runtime

Objective of this Talk

- Participants will get a pragmatic **introduction** to using **Spring Security** Version 6.2 using a practical example to integrate security features into RESTful services.
- The seamless integration with **OAuth 2.0** and **OpenID Connect** will also be briefly discussed.

About me

- I am a trainer at letsboot.ch, a lecturer at the Bern University of Applied Sciences and a freelance IT consultant and enthusiastic software developer.
- Over the past 35 years I have worked in many IT fields and industries (eg. mid-sized IT service provider, Swisscom, SRG SSR).
- My current areas of activity include process management with BPMN and Camunda, enterprise application integration with Apache Camel, backend development with the Spring Ecosystem, Docker, Kubernetes, etc.

Personal Note

- Please note that I am wearing hearing aids.
- I may not always understand you immediately.

Photo by Andrea Piacquadio: https://pexels.com

Use Case

+41 61 551 00 82 - info@letsboot.ch
In Zürich, Basel, remote und vor Ort

Software and Systems Engineering Courses

In Zurich, Basel, Remote or on site

Hands-on courses for software and system developers by experienced experts with proven training material in Basel, Zurich, remote and on-site.

🇳🇿 **Here you get to Letsboot New Zealand!**

Schedule

Public course dates on sought-after topics and technologies around software development, DevOps and cloud engineering.

Container & Kubernetes DevOps
09. - 11. April 2024,
Zürich, DE, CHF 2430.--

Jimmy Bogard: Modern .NET with Vertical Slice
09. - 10. April 2024,
Zürich, EN, CHF 2180.--

Container & Kubernetes Security
14. & 15. Mai 2024,
Zürich, DE, CHF 1800.--

GitLab CI/CD
16. & 17. Mai 2024,
Zürich, EN, CHF 1800.--

Spring Security
05. - 06. Juni 2024,
Zürich, DE, CHF 1800.--

Container & Kubernetes DevOps
26. - 28. August 2024,
Zürich, DE, CHF 2430.--

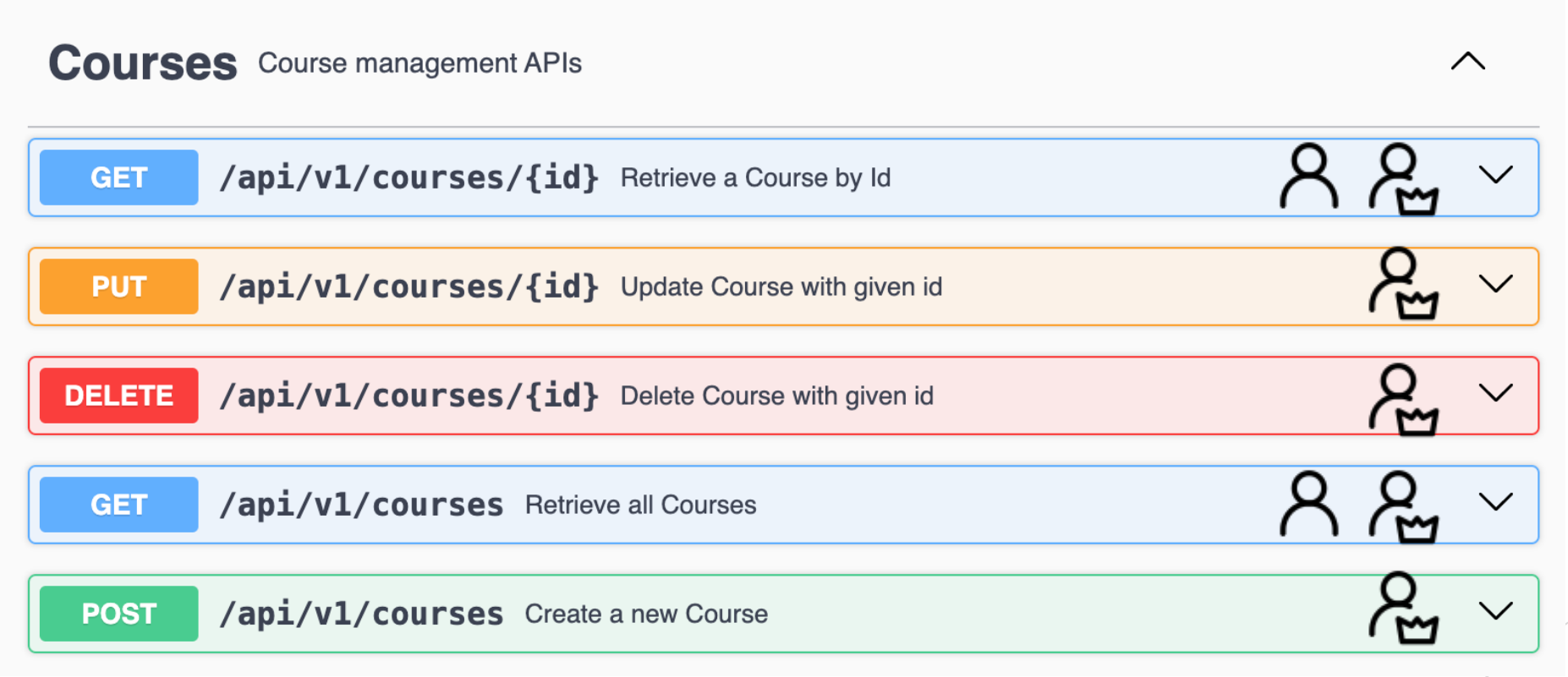
Angular & TypeScript
04. - 06. November 2024,
Zürich, DE, CHF 2250.--

GitLab CI/CD
06. & 07. November 2024,
Zürich, DE, CHF 1800.--

Container & Kubernetes DevOps
19. - 21. November 2024,
Zürich, DE, CHF 2430.--

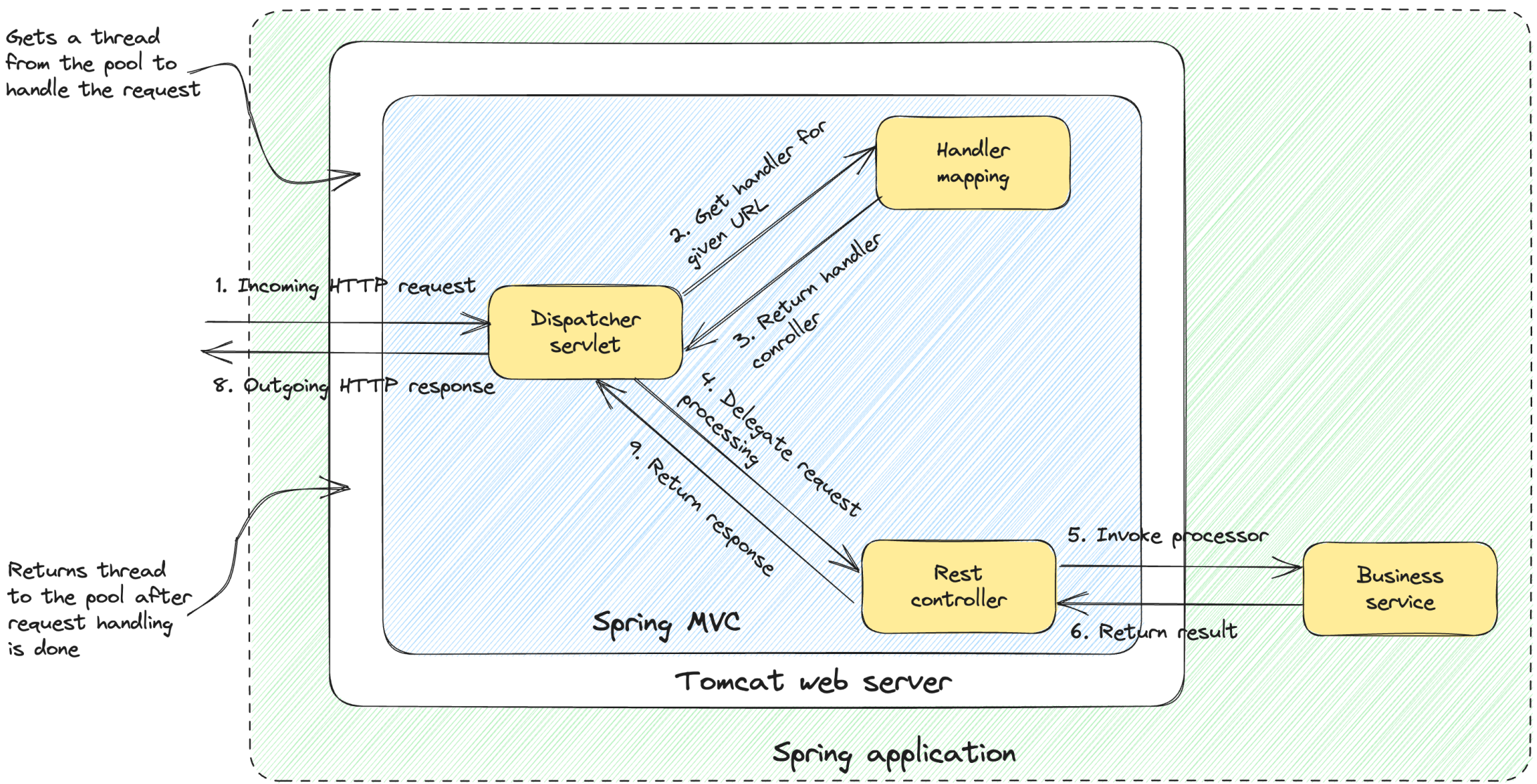
Microservices mit Spring Boot
27. - 29. November 2024,
Zürich, DE, CHF 2430.--

Tom‘s Design for the API of the course management service



Let's dive in!

Request Flow in Spring MVC



Opt-in with Spring Security Starter

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

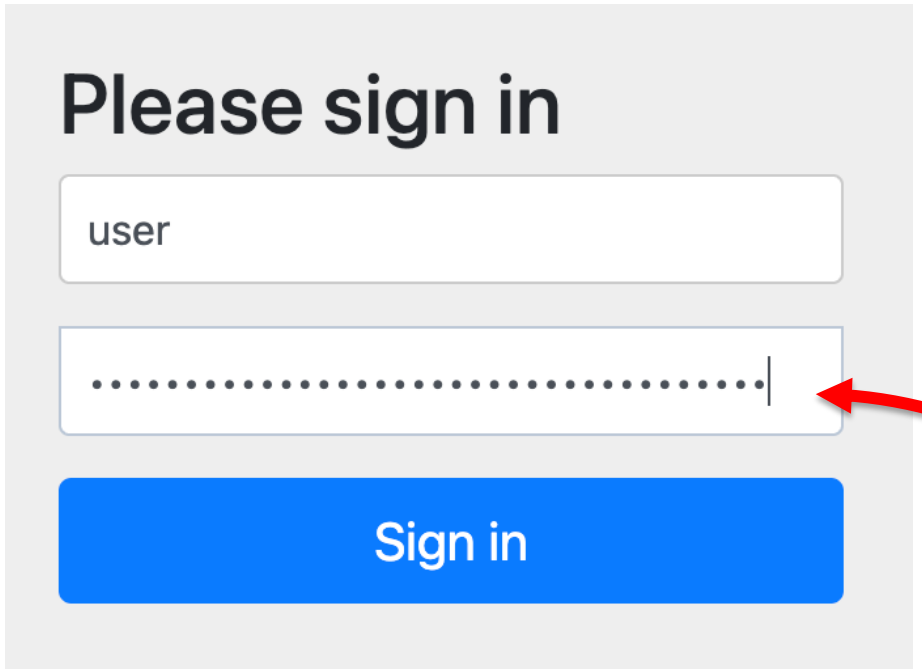
Impact of this Dependency

- Requires **authentication** for all endpoints
- **Default user** with generated password at startup
- **Protects password** storage with Bcrypt etc.
- Supports **form-based login** and **logout**
- Authenticates **form-based login** and **HTTP-Basic**
- Mitigates **CSRF** and Session Fixation attacks
- …

https://docs.spring.io/spring-security/reference/servlet/getting-started.htm

Demo 1

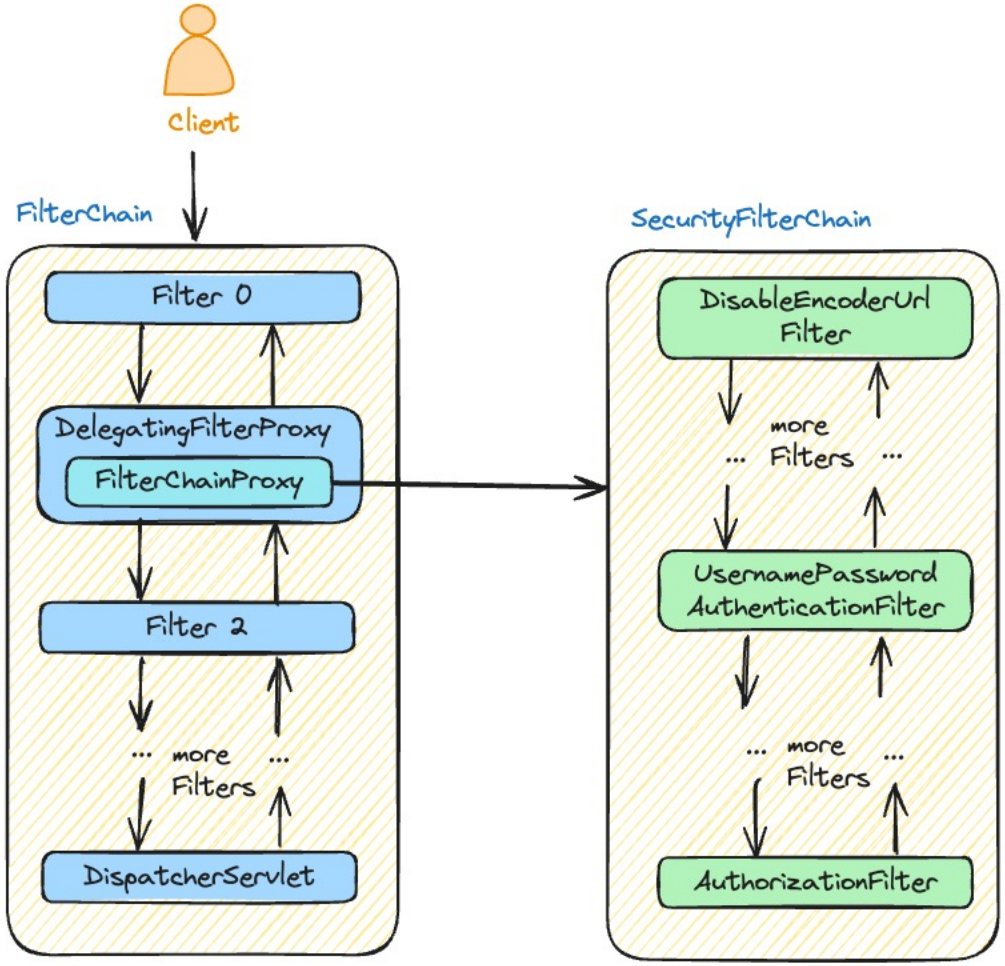
http://localhost:8080/api/v1/courses




```
Using generated security password: 9cff30f3-8795-4eea-8d39-47389e9e3319

This generated password is for development use only.
```

Behind the Scenes: SecurityFilterChain



logging.level.org.springframework.security.web.FilterChainProxy=TRACE

DefaultSecurityFilterChain

```
Invoking DisableEncodeUrlFilter (1/14)
Invoking WebAsyncManagerIntegrationFilter (2/14)
Invoking SecurityContextHolderFilter (3/14)
Invoking HeaderWriterFilter (4/14)
Invoking CsrfFilter (5/14)
Invoking LogoutFilter (6/14)
Invoking UsernamePasswordAuthenticationFilter (7/14)
Invoking DefaultLoginPageGeneratingFilter (8/14)
Invoking DefaultLogoutPageGeneratingFilter (9/14)
Invoking RequestCacheAwareFilter (10/14)
Invoking SecurityContextHolderAwareRequestFilter (11/14)
Invoking AnonymousAuthenticationFilter (12/14)
Invoking ExceptionTranslationFilter (13/14)
Invoking AuthorizationFilter (14/14)
```

https://docs.spring.io/spring-security/reference/servlet/architecture.html

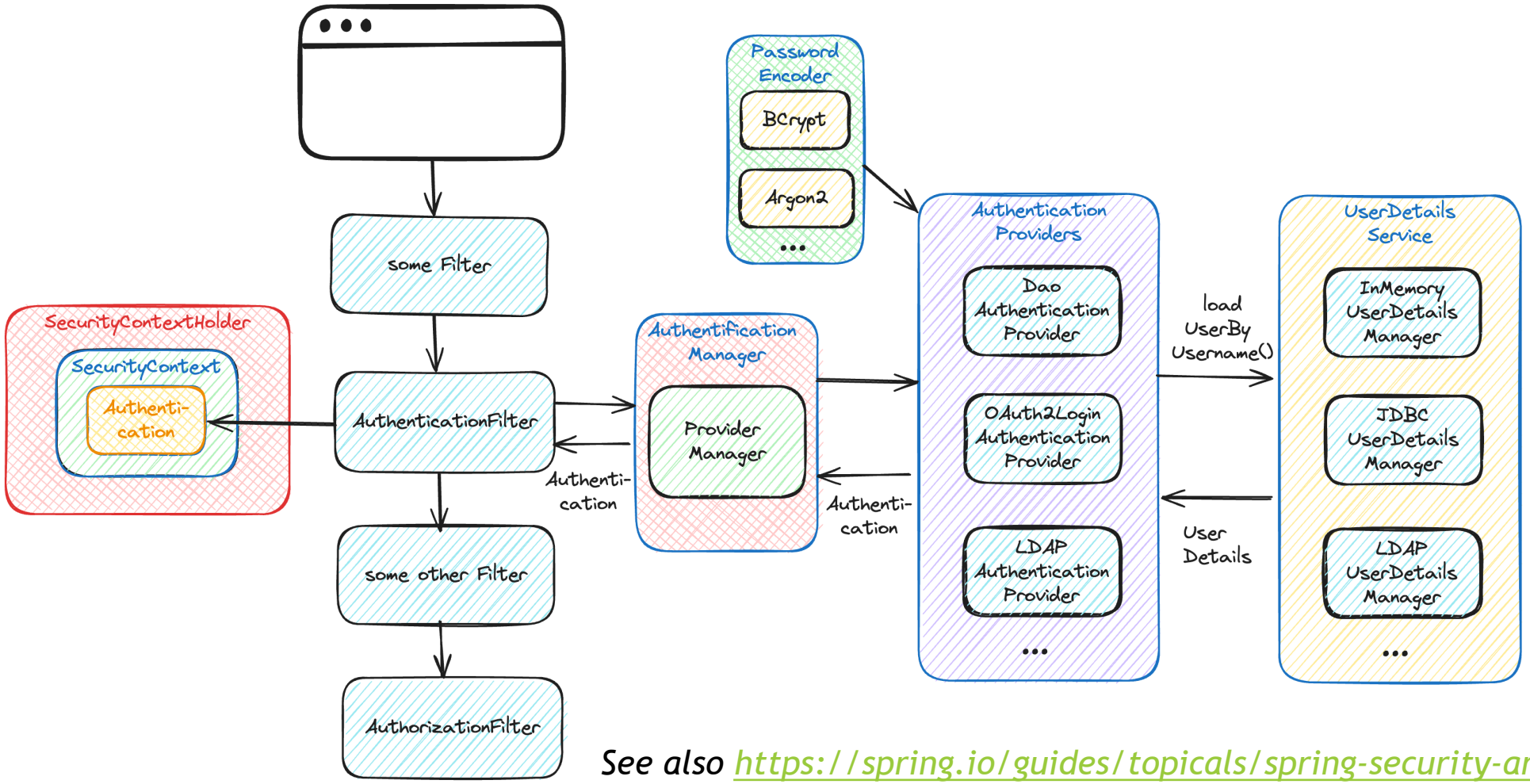
Cross-Site-Request-Forgery (CSRF) Protection

- Adding the `spring-boot-starter-security` dependency also enables CSRF protection by default.

```
<!DOCTYPE html>
<html lang="en">
    <head> [8 lines]
    <body>
        <div class="container">
            <form class="form-signin" method="post" action="/login">
                <h2 class="form-signin-heading">Please sign in</h2>
                <p>
                    <label for="username" class="sr-only">Username</label>
                    <input type="text" id="username" name="username" class="form-control" placeholder="Username" required autofocus>
                </p>
                <p>
                    <label for="password" class="sr-only">Password</label>
                    <input type="password" id="password" name="password" class="form-control" placeholder="Password" required>
                </p>
                <input name="_csrf" type="hidden" value="RFrllJqXEZqhEpTQBNZ45PSkB58btwzxew4fQqtX5QwLEpK-cmvRoajxd_0MIaziPftM1s2VKqcihW3cGT4vI2o2qz48JfGH" />
                <button class="btn btn-lg btn-primary btn-block" type="submit">Sign in</button>
            </form>
        </div>
    </body></html>
```

Spring Security Architecture

Spring Security Components (a small extract)



See also https://spring.io/guides/topicals/spring-security-architecture

SecurityFilterChain Bean

- The SecurityFilterChain can hold an arbitrary number of security filters.
- Typically you only need to specify your **authentication** and **authorization rules.** Example :

```
@Configuration
public class AuthorizationConfig {

    @Bean
    public SecurityFilterChain configure(HttpSecurity http) throws Exception {
        http.formLogin(Customizer.withDefaults()); // log in with username and pw
        http.authorizeHttpRequests((authorize) -> authorize
                .requestMatchers(HttpMethod.GET, "/api/v1/courses/**").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/v1/courses/**").hasRole("ADMIN")
                .anyRequest().authenticated()
        );
        return http.build();
    }
}
```

Securit

- The Sec
  filters.
- Typically
  authoriz

Configures the authorization of HTTP requests.

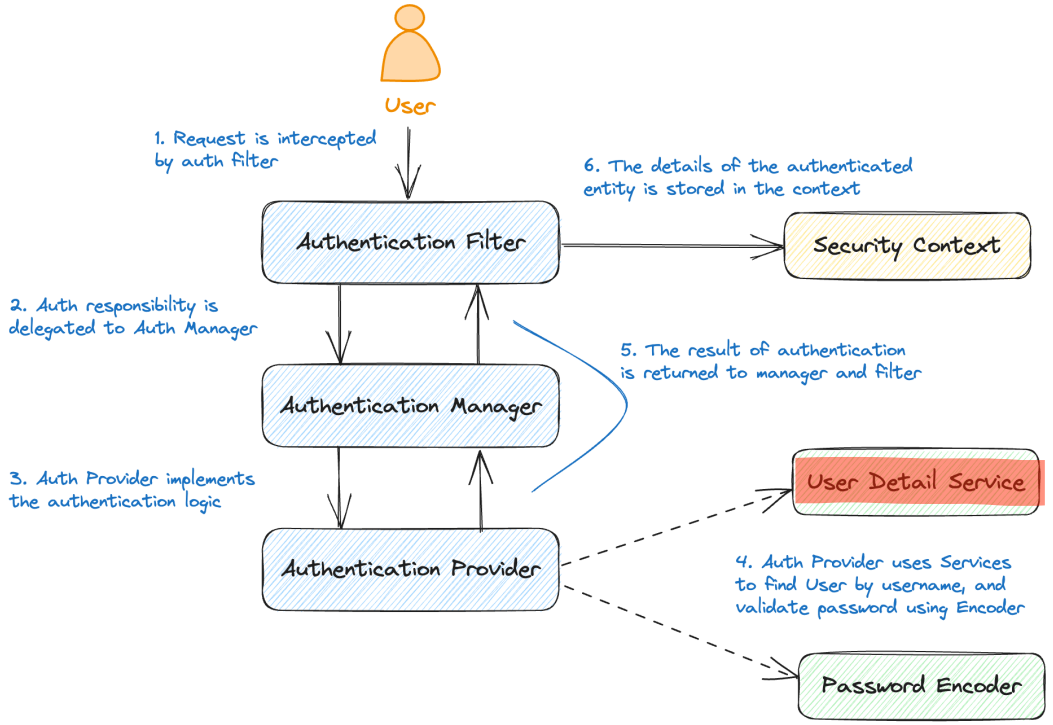
- Allows all GET requests to paths starting with "/api/v1/courses/" for any user.
- However, POST requests to the same paths are only permitted for users with the "ADMIN" role. All other requests require authentication.

```
@Configuration
public class AuthorizationConfig {

    @Bean
    public SecurityFilterChain configure(Ht      rity http) throws Exception {
        http.formLogin(Customizer.withDefault      ); // log in with username and pw
        http.authorizeHttpRequests((authorize) -> authorize
                .requestMatchers(HttpMethod.GET, "/api/v1/courses/**").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/v1/courses/**").hasRole("ADMIN")
                .anyRequest().authenticated()
        );
        return http.build();
    }
}
```

Authentication Mechanisms

- Spring Security provides comprehensive support for Authentication. Example:



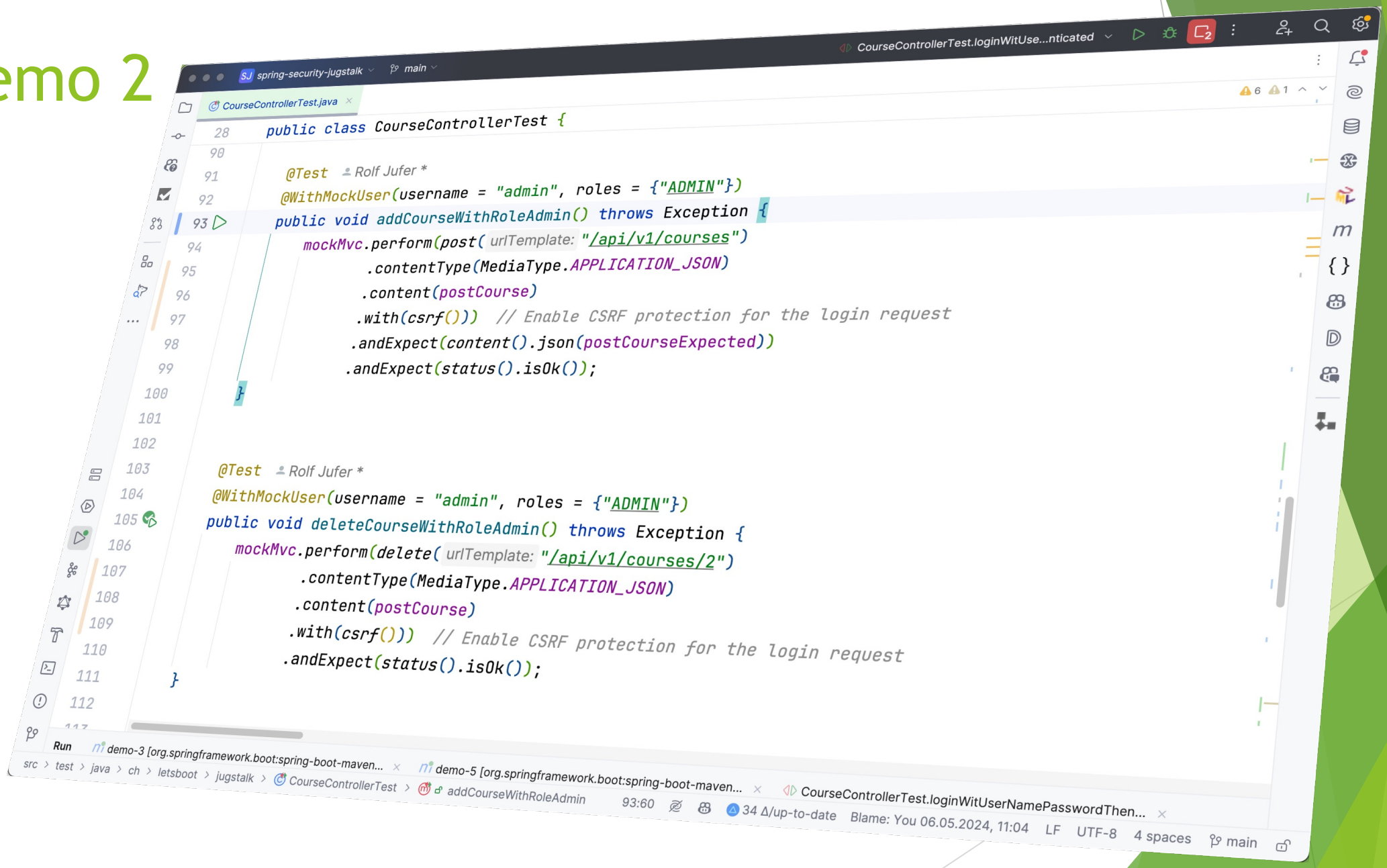

```
@Configuration
public class UserManagementConfig {

    @Bean
    public UserDetailsService users(PasswordEncoder passwordEncoder) {

        UserDetails admin = User.builder()
            .username("admin").password(passwordEncoder.encode("password"))
            .roles("ADMIN").build();

        // InMemoryUserDetailsManager implements UserDetailsService to provide support
        // for username/password based authentication that is stored in memory.
        return new InMemoryUserDetailsManager(admin);
    }
}
```

Demo 2


spring-security-jugstalk
main
CourseControllerTest.loginWitUse...nticated
CourseControllerTest.java
public class CourseControllerTest {
@Test Rolf Jufer *
@WithMockUser(username = "admin", roles = {"ADMIN"})
public void addCourseWithRoleAdmin() throws Exception {
mockMvc.perform(post(urlTemplate: "/api/v1/courses")
.contentType(MediaType.APPLICATION_JSON)
.content(postCourse)
.with(csrf())) // Enable CSRF protection for the login request
.andExpect(content().json(postCourseExpected))
.andExpect(status().isOk());
}
@Test Rolf Jufer *
@WithMockUser(username = "admin", roles = {"ADMIN"})
public void deleteCourseWithRoleAdmin() throws Exception {
mockMvc.perform(delete(urlTemplate: "/api/v1/courses/2")
.contentType(MediaType.APPLICATION_JSON)
.content(postCourse)
.with(csrf())) // Enable CSRF protection for the login request
.andExpect(status().isOk());
}
Run
demo-3 [org.springframework.boot:spring-boot-maven...
demo-5 [org.springframework.boot:spring-boot-maven...
CourseControllerTest.loginWitUserNamePasswordThen...
src > test > java > ch > letsboot > jugstalk > CourseControllerTest > addCourseWithRoleAdmin
93:60
34 Δ/up-to-date
Blame: You 06.05.2024, 11:04
LF
UTF-8
4 spaces
main

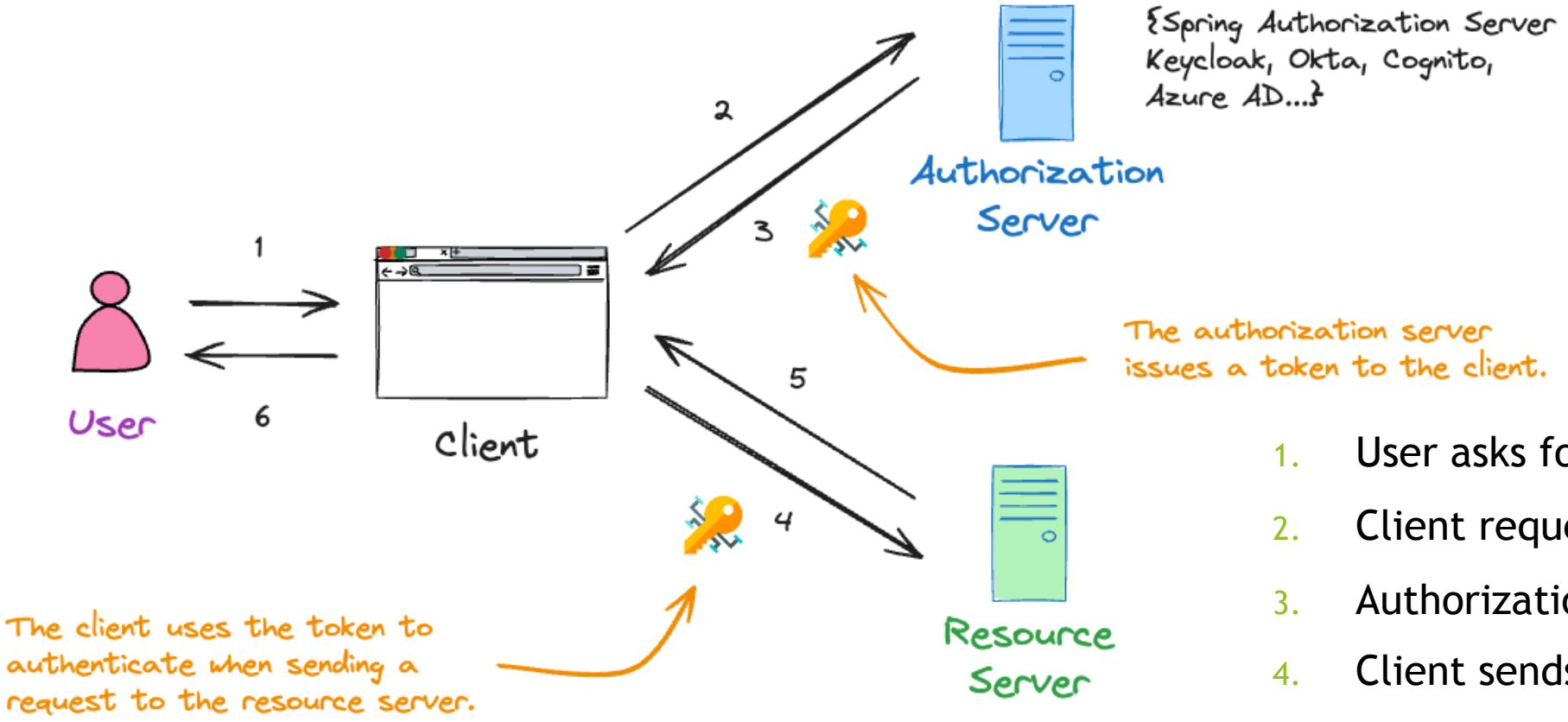
Accessment of Demo 2

- The demo 2 suffers from the fact that it must provide its own **user management.**
- This has significant **drawbacks** such as
  - lack of single sign-on (SSO)
  - fragmented user data
  - potential security risks
  - lack of standardisation
  - and scalability issues

Delegating User Authentication to an Authentication Provider

- Spring Security enables you to **delegate user authentication** to Identity and Access Management (IAM) solutions such as Keycloak, Okta or other OAuth 2.0/OpenID Connect providers.
- This delegation enables **centralised management** of user authentication, simplifying integration and improving security.

Authorization Server – Ressource Server: Token Exchange (Bird's eye view)



1. User asks for secured ressouce.
2. Client requests token.
3. Authorization server issues token.
4. Client sends requests.
5. Resource server authorizes and executes.
6. Client displays result.

Example Response from Authorization Server

*encoded JWT ***

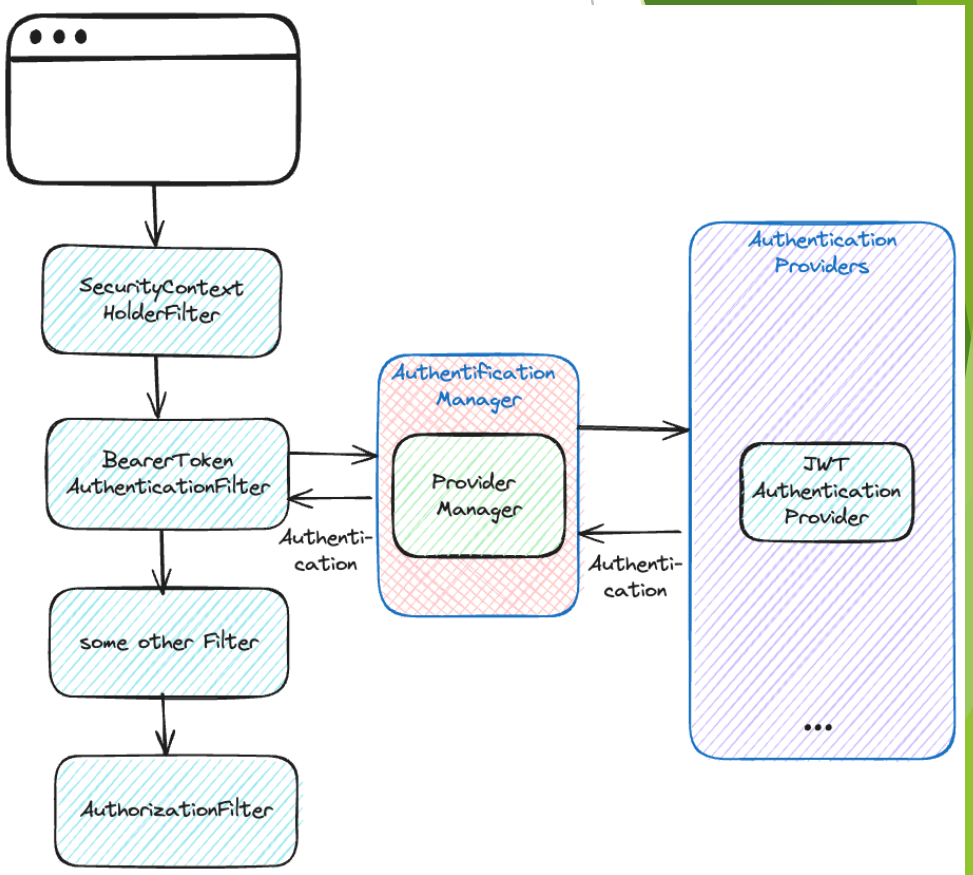
```
{
   "access_token": "eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiS…",
   "expires_in": 300,
   "refresh_expires_in": 1800,
   "refresh_token": "eyJhbGciOiJIUzI1NiIsInR5cCIgOi…",
   "token_type": "Bearer", *
   …
}
```

* *The name "Bearer" comes from the fact that here we simply "presents" or "carries" the token.*

** *OAuth 2.0 Spec itself does not mandate the format of the access token, but it is often JWT.*

JWT (JSON Web Token)

- JWT (JSON Web Token) plays a crucial role as the means of exchanging authentication information between the authentication provider (e.g., Keycloak) and your Spring Boot application.
- It serves as a **secure** and **standardized format** for transmitting authentication data, facilitating **interoperability** and ensuring **data integrity**.



Leverage the Power of OAuth 2.0 and OpenID Connect

Spring Security with OAuth 2.0 and OpenID Connect

- Integrating Spring Security with **OAuth 2.0** and **OpenID Connect** (OIDC) allows you to secure your Spring-based applications by leveraging **industry-standard protocols** for authentication and authorization.
- **In a nutshell:**
  - **OAuth 2.0** is the foundation for controlled access to resources. JWT is generally used as a token format in OAuth 2.0 implementations to represent the access tokens.
  - **OpenID Connect** builds upon OAuth 2.0 to add user authentication and information sharing.

Spring Security OAuth2 Dependencies

(see Spring Initializr)

OAuth2 Client SECURITY

Spring Boot integration for Spring Security's OAuth2/OpenID Connect client features.

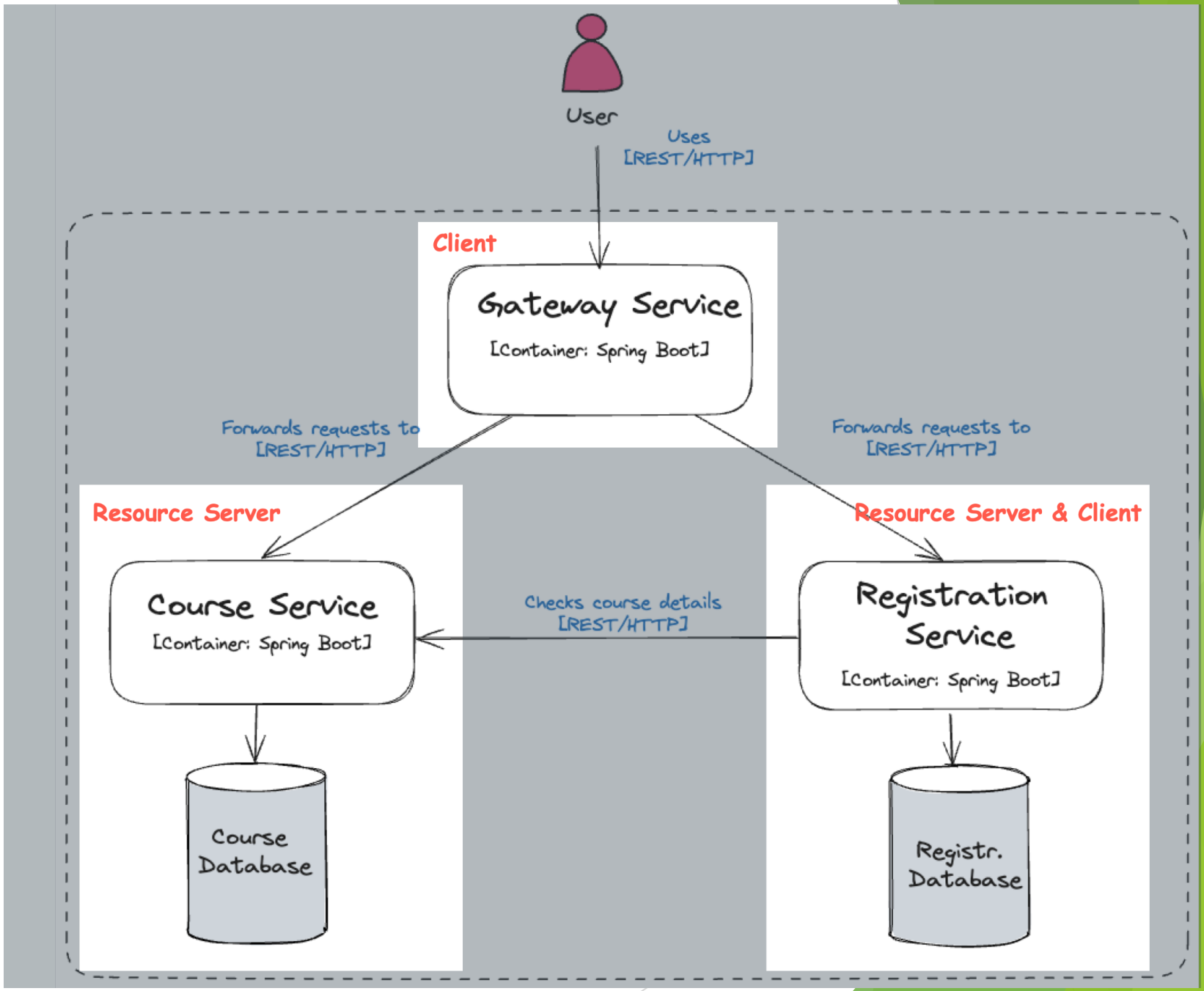
OAuth2 Resource Server SECURITY

Spring Boot integration for Spring Security's OAuth2 resource server features.

OAuth2 Authorization Server SECURITY

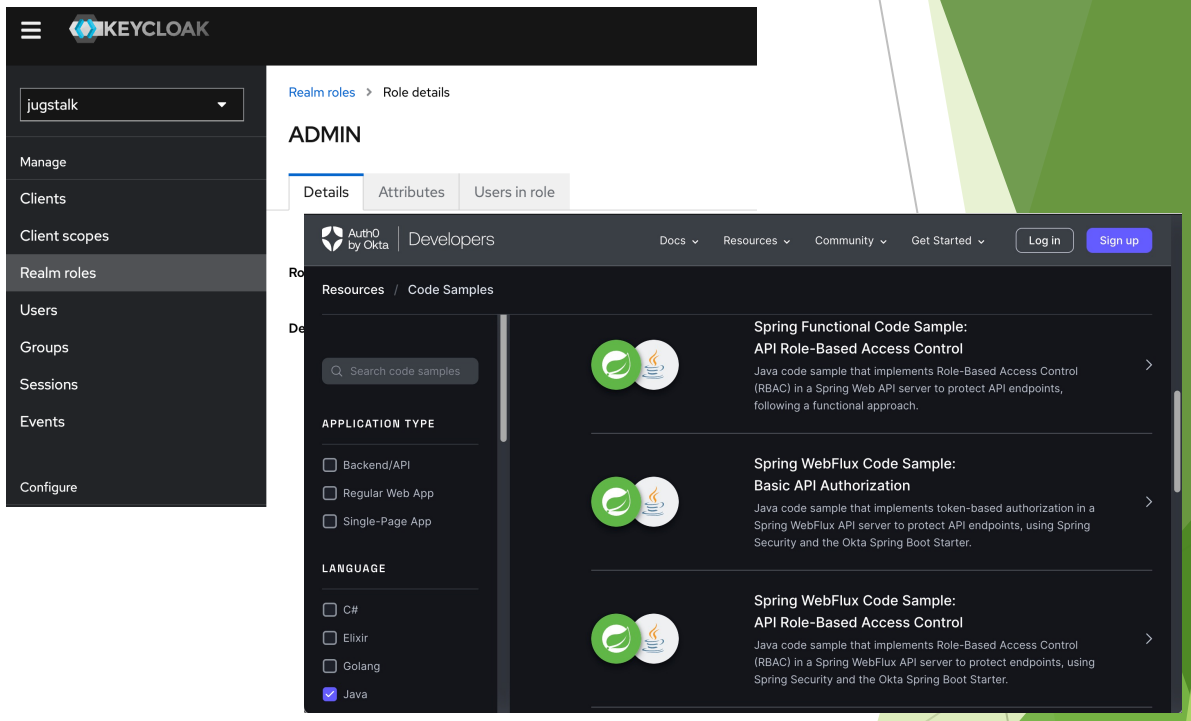
Spring Boot integration for Spring Authorization Server.

OAuth2 Roles: Client and Rescource Server



OAuth2 Authorization Server

- Instead of Spring Security's native Authorization Server, you may use external identity providers like Keycloak, Okta, Cognito etc.



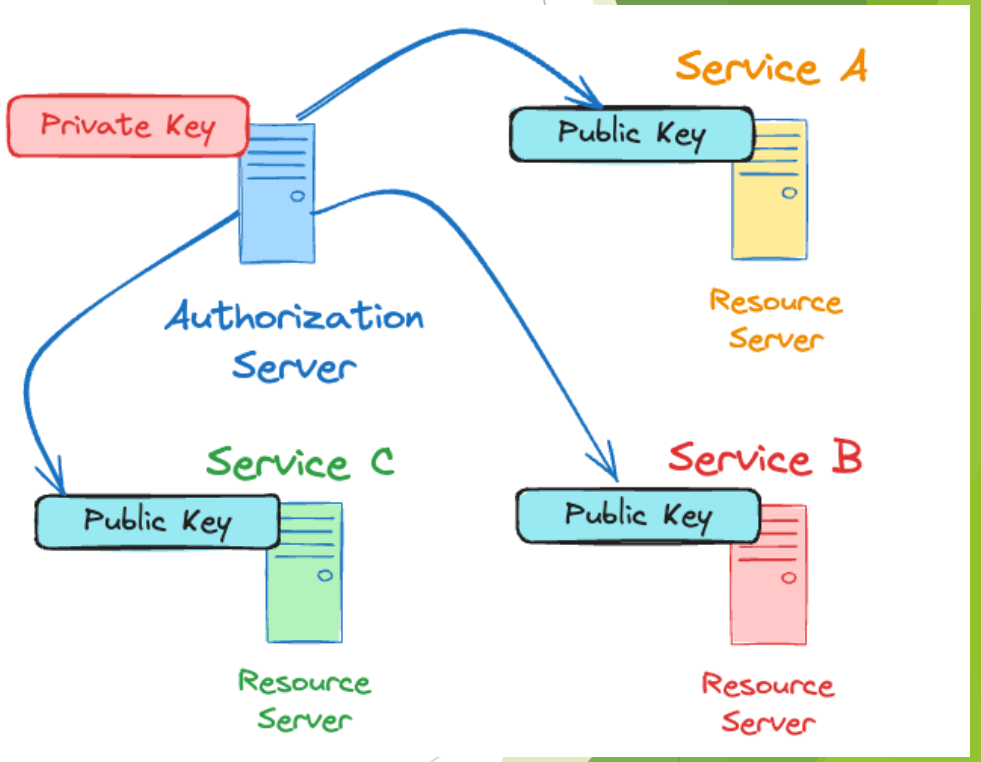
Authorization Server Config

(Example Keycloak)

```
// http://localhost:8081/realms/jugstalk/.well-known/openid-configuration
{
  "issuer": "http://localhost:8081/realms/jugstalk",
  "authorization_endpoint": "http://localhost:8081/realms/jugstalk/protocol/openid-
                                    connect/auth",
  "token_endpoint": "http://localhost:8081/realms/jugstalk/protocol/openid-
                              connect/token",
  "introspection_endpoint": "http://localhost:8081/realms/jugstalk/protocol/openid-
                                    connect/token/introspect",
  "jwks_uri": "http://localhost:8081/realms/jugstalk/protocol/openid-connect/certs",
  "grant_types_supported": [
    "authorization_code", "refresh_token", "password", "client_credentials",… ]
…
}
```

JSON Web Key Set

- JWKS (JSON Web Key Set) is used for securely **distributing public keys** in a JSON format.
- It allows clients to verify the authenticity and **integrity** of JSON Web Tokens (JWTs) by providing a set of public keys.



Authorization & Token Endpoint

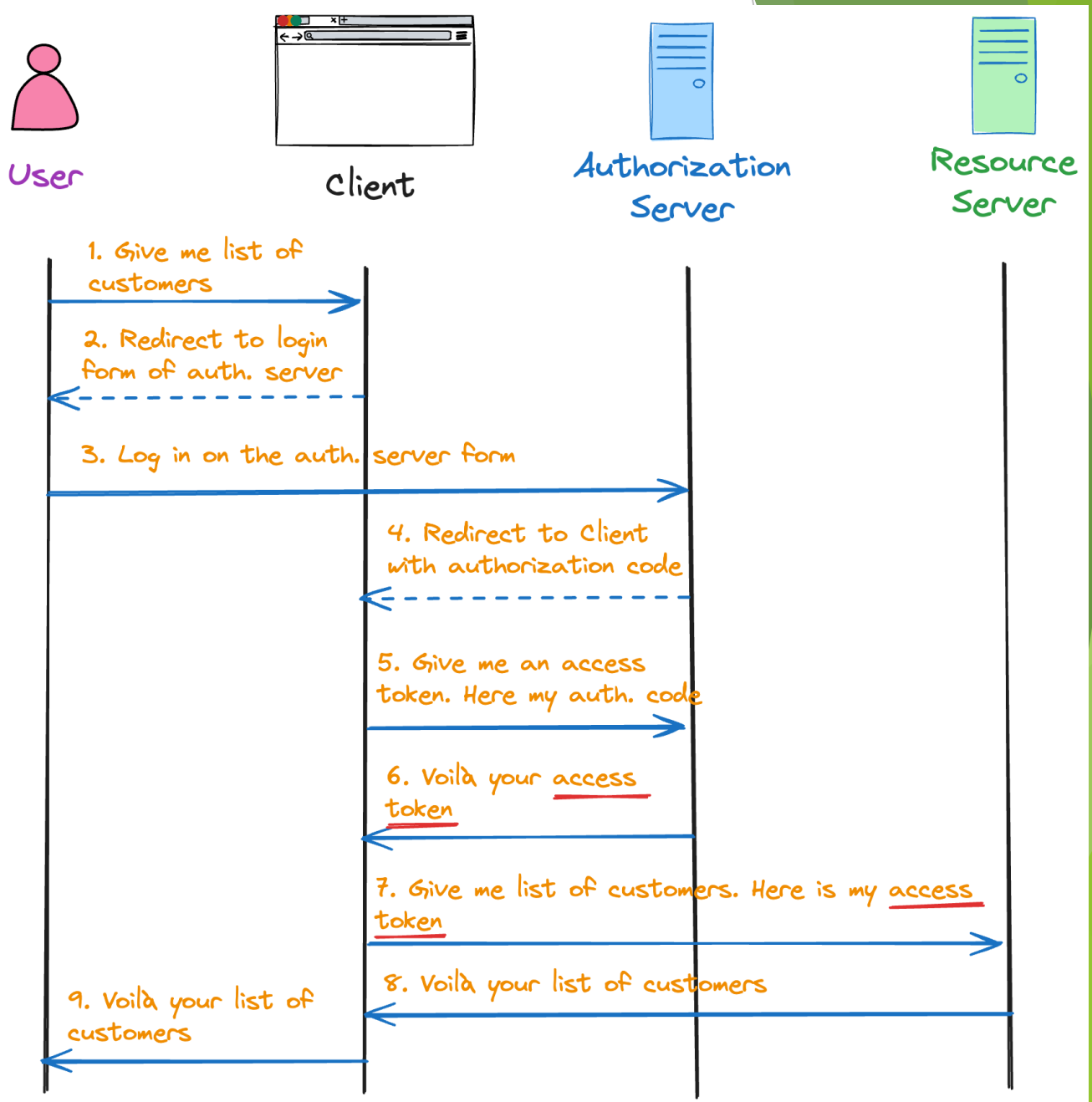
- The **Authorization Endpoint** is where the OAuth 2.0/OpenID Connect authorization process begins. It handles user authentication, consent, and authorization, ultimately granting the client application the permissions needed to access protected resources on behalf of the user.
- The **Token Endpoint** specifies the endpoint where OAuth 2.0/OpenID Connect clients can exchange authorization codes or refresh tokens for access tokens (see below).

Getting a Token from Authorization Server

- An OAuth2 **grant type** is the method by which a client obtains a token. There is a whole range of approaches to how clients obtain their token from the authentication server.
- The most **common grant types** are
  - authorization code grant type
  - authorization code grant type with PKCE (“pixy”, proof key for code exchange)
  - client credentials grant type

Also see: https://oauth.net/2/grant-types/ or https://www.oauth.com/playground/index.html

Example Authorization Code Grant Type



Nota bene:
We could improve this flow by adding a verifier together with a code challenge and a code challenge method during the Authorisation Code Grant Flow. This is achieved through PKCE (Proof Key for Code Exchange).

OAuth2 Ressource Server Dependency

- By adding `spring-boot-starter-oauth2-resource-server` to a Spring Boot application, it can act as a protected API and validate **access tokens** from clients.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-oauth2-resource-
   server</artifactId>
</dependency>
```

Authentication Measures for JWTs in Spring Security

Spring Security authenticates each JWT by performing checks:

1. **Validating the Signature**: Ensuring the integrity of the JWT by verifying its signature.
2. **Time Validation**: Confirming that the current time falls within the timeframe specified by the iat (Issued At) and exp (Expires At) claims.
3. Furthermore, Spring Security offers **additional validation** capabilities, such as verifying the issuer (iss claim) and the audience (aud claim).

SecurityFilterChain Bean

(as applied in Spring Ressource Server)

```
@Configuration
public class AuthorizationConfig {

    @Bean
    public SecurityFilterChain configure(HttpSecurity http) throws Exception {
        // Configures OAuth 2.0 resource server support for the application.
        // This enables the application to act as a resource server, capable of
        // accepting and responding to protected resource requests using access tokens.
        http.oauth2ResourceServer(oauth2 -> oauth2.jwt(jwt ->
                jwt.jwtAuthenticationConverter(jwtConverter)));
        // Converter is responsible for extracting relevant information from the JWT
        // token (like roles, expiry date etc.)
        // ...
        );
        return http.build();
    }

}
```

See also https://docs.spring.io/spring-security/reference/servlet/oauth2/index.html

OAuth2 Client Dependency

- By adding the `spring-boot-starter-oauth2-client` dependency to a Spring Boot application, it can seamlessly act as an **OAuth 2.0 client**. This simplifies the process of integrating with OAuth authorization servers and accessing protected resources from resource servers

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-oauth2-
   client</artifactId>
</dependency>
```

SecurityWebFilterChain Bean

(as applied in Spring Cloud API Gateway)

- **SecurityWebFilterChain** is specifically designed for reactive applications using Spring WebFlux.
- Note: Spring Cloud Api Gateway is reactive.

```
@Configuration
@EnableWebFluxSecurity // Enables Spring Security for reactive applications
public class SecurityConfig {

    @Bean
    public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity http) {
        http.authorizeExchange(auth -> auth.anyExchange().authenticated())
        //Convenient way to enable OAuth2 Login with some pre-configured settings.
        //However, these defaults might not always align with your specific requirements.
                .oauth2Login(withDefaults())
        return http.build();
    }
}
```

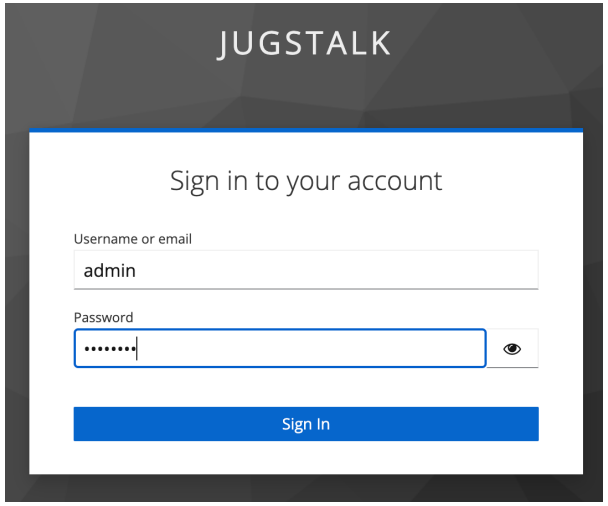
Demo 3 (1/2)

Get Access Token (from Authorization Server → Keycloak)

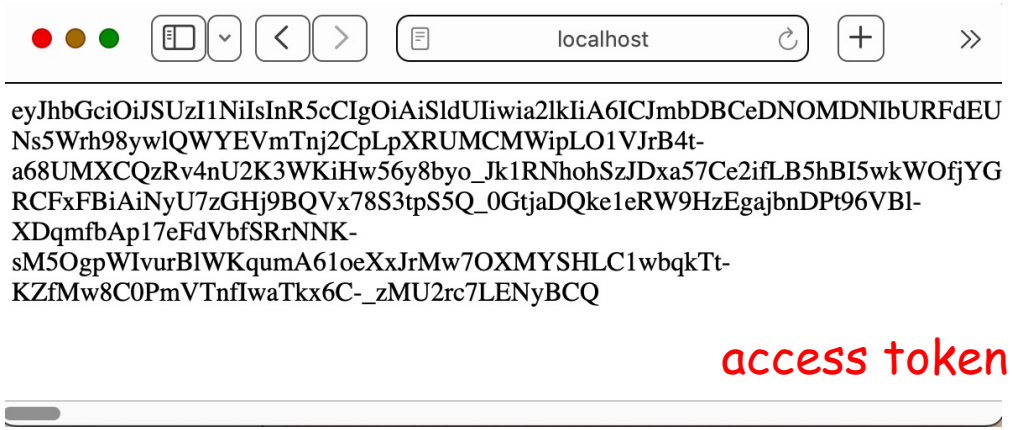
1

http://localhost:8081/realms/jugstalk/protocol/openid-connect/auth?response_type=code&client_id=spring-client&redirect_uri=http://localhost:8082/token&scope=openid&code_challenge=w3e-9_A456pK_x90KCRI9_JQpw3-mf4sh48KgcmhB3k&code_challenge_method=S256

2



3



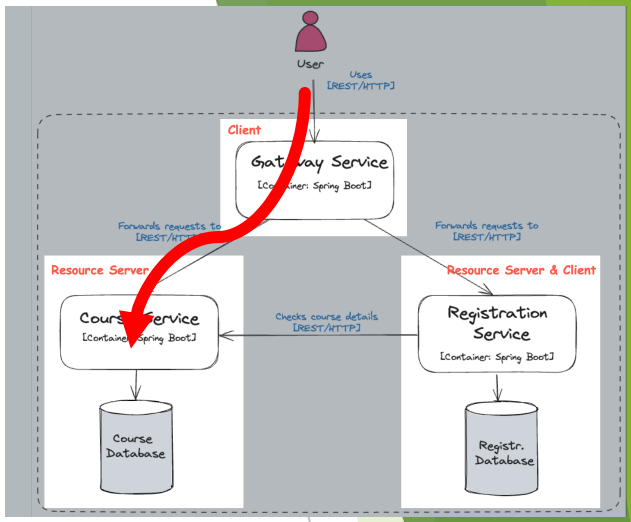
Demo 3 (2/2)

Create new Course

4

```
### Create course
POST http://localhost:8082/api/v1/courses
Authorization: Bearer eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6IC
Content-Type: application/json

{
  "id": 8,
  "title": "Microservices with Spring Boot",
  "description": "Nowadays, enterprise applications are increasingly buil
  "city": "Zurich",
  "startDate": "26.11.2024",
  "endDate": "28.11.2024",
  "durationInDays": 3,
  "price": 2430.00,
  "trainerId": 1
}
```



5

```
HTTP/1.1 200 OK
(Headers) …Content-Type: application/json…

{
  "id": 8,
  "title": "Microservices with Spring Boot",
  "description": "Nowadays, enterprise applications are increasingly buil
  "city": "Zurich",
  "startDate": "26.11.2024",
  "endDate": "28.11.2024",
  "durationInDays": 3,
  "price": "CHF 2'430.00",
  "trainerId": 1,
  "trainerName": "Rolf Jufer",
```

Demos and more Details about Oauth2 Client

- Due to time constraints, it is not possible to do another demo on this topic.
- However, there are many demos and additional explanations on youtube or on various tech blogs (e.g. Piotr's TechBlog, Dan Vegas Blog, Baeldungs Blog...).
- I would also like to point out that I offer a 2-day course on Spring Security. More information on the website of letsboot.ch. I offer free CHF 300 vouchers.

Comparison of Alternatives to Spring Security

(randomly chosen)

Feature	Spring Security	Apache Shiro	JAAS*	Apache Fortress	PicketLink
Authentication	Yes	Yes	Yes	Yes	Yes
Authorization	Yes	Yes	Yes	Yes	Yes
Session Management	Yes	Yes	Yes	Yes	Yes
Encryption	Yes	Yes	Yes	Yes	Yes
Integrability	Spring, JEE, Servlet, etc.	Spring, JEE, Servlet, etc.	Java EE	Java EE, Spring, etc.	Java EE, Spring, etc.
Flexibility	High	High	Medium to High	High	High
Community Support	Active Community	Active Community	JDK Support	Active Community	Active Community
Complexity	Medium to High	Medium to High	High	High	High
Identity Management	Partial (depends on integrations)	Partial (depends on integrations)	No	Yes	Yes
SSO (Single Sign-On)	Yes	Partial (depends on integrations)	Partial (depends on configuration)	Yes	Yes
Social Login	Yes	Partial (depends on integrations)	No	Yes	Yes

JAAS = Java Authentication and Authorization Service

Conclusion

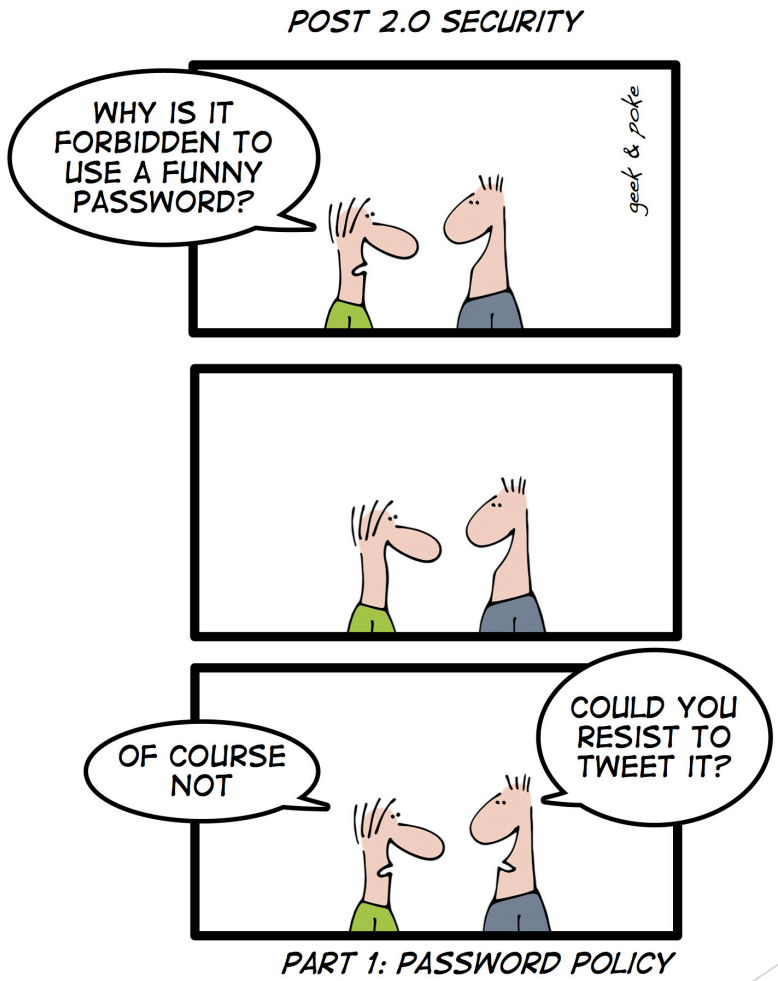
Spring Security at a Glance

- Spring Security is the cornerstone **security solution** for Java applications, seamlessly integrated into the Spring ecosystem.
- Its **flexibility**, combined with effortless **integration** and **continuous evolution**, enables developers to secure their applications with confidence.
- *From **authentication** to **authorisation**, Spring Security is the **essential choice** for secure Java development within the Spring framework.*

Additional Sources

- **Spring Security Website:** https://docs.spring.io/spring-security/reference/index.html
- **Spring Security in Action**, Second Edition, Manning, 2024, ISBN 978-1633437975
- **Authentifizierung und Autorisierung in der IT,** Grundlagen und Konzepte, Hanser 2024, ISBN 978-3-446-47949-4
- **Marcobehler-guide**: https://github.com/marcobehler/marcobehler-guides/blob/main/spring-security.adoc

Thank you!



Source: Geek & Poke