Spring Security

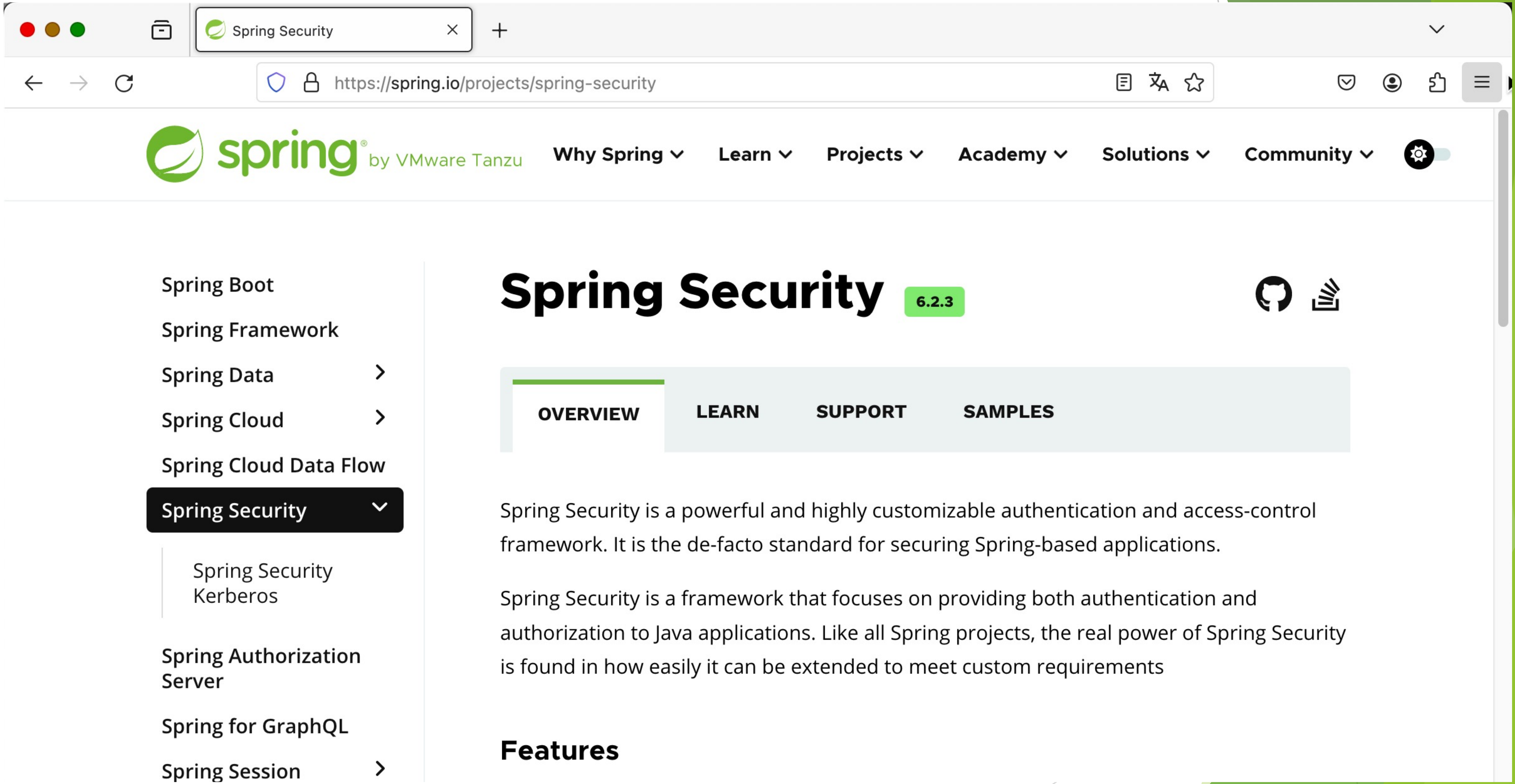
rolf.jufer@letsboot.ch

letsboot.ch
swiss dev training

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

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

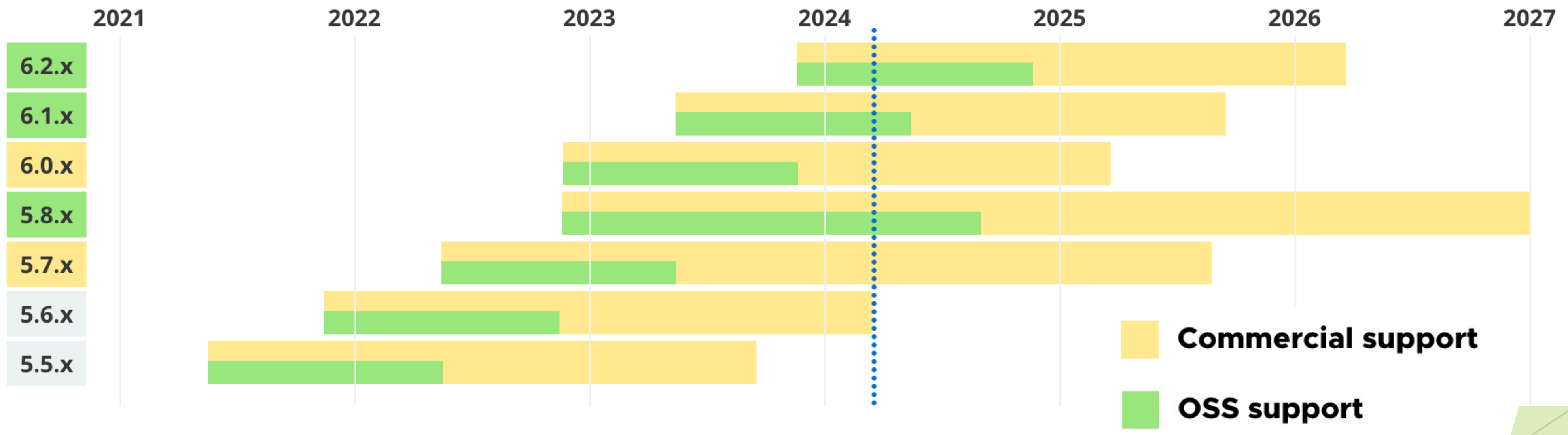
Features

Spring Security

- Spring Security has become a **key project** in the Spring Ecosystem.
- It provides comprehensive support for **authentication**, **authorisation** and **protection** against common security vulnerabilities.
- Spring Security's integration spans multiple frameworks, APIs, and **servlet** and **reactive** stacks.

Focus of this talk

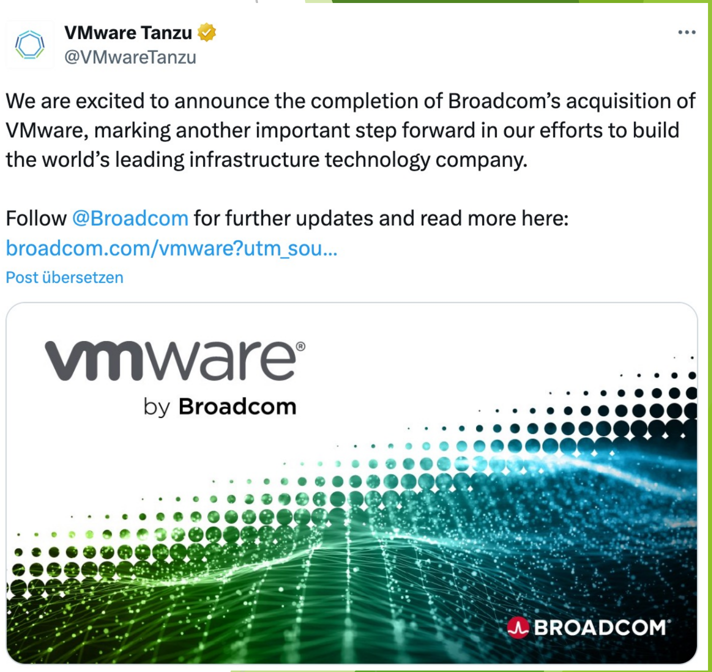
Spring Security Releases and Support



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

Impact of VMware's sale to Broadcom

- The Spring Framework/Ecosystem is **open source** and continues to be developed by the **Spring community**.
- However, the sale of VMware to Broadcom poses potential risks to the future of Spring, particularly with respect to long-term support and strategic direction of the framework.
- In particular, Broadcom may reduce its financial and human resources support for the development of Spring and Spring Security.



VMware Tanzu
@VMwareTanzu

We are excited to announce the completion of Broadcom's acquisition of VMware, marking another important step forward in our efforts to build the world's leading infrastructure technology company.

Follow @Broadcom for further updates and read more here:
broadcom.com/vmware?utm_sou...
Post übersetzen

Source: X (formerly Twitter) 22. Nov. 2023

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.

Note: Spring Security Focus

- This talk specifically covers Spring Security and does **not address general security topics** like the OWASP Top 10.
- We'll focus on using and configuring Spring Security in Spring-based applications to protect against some important security risks and mitigate them.
- For broader security principles, consider additional resources.

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

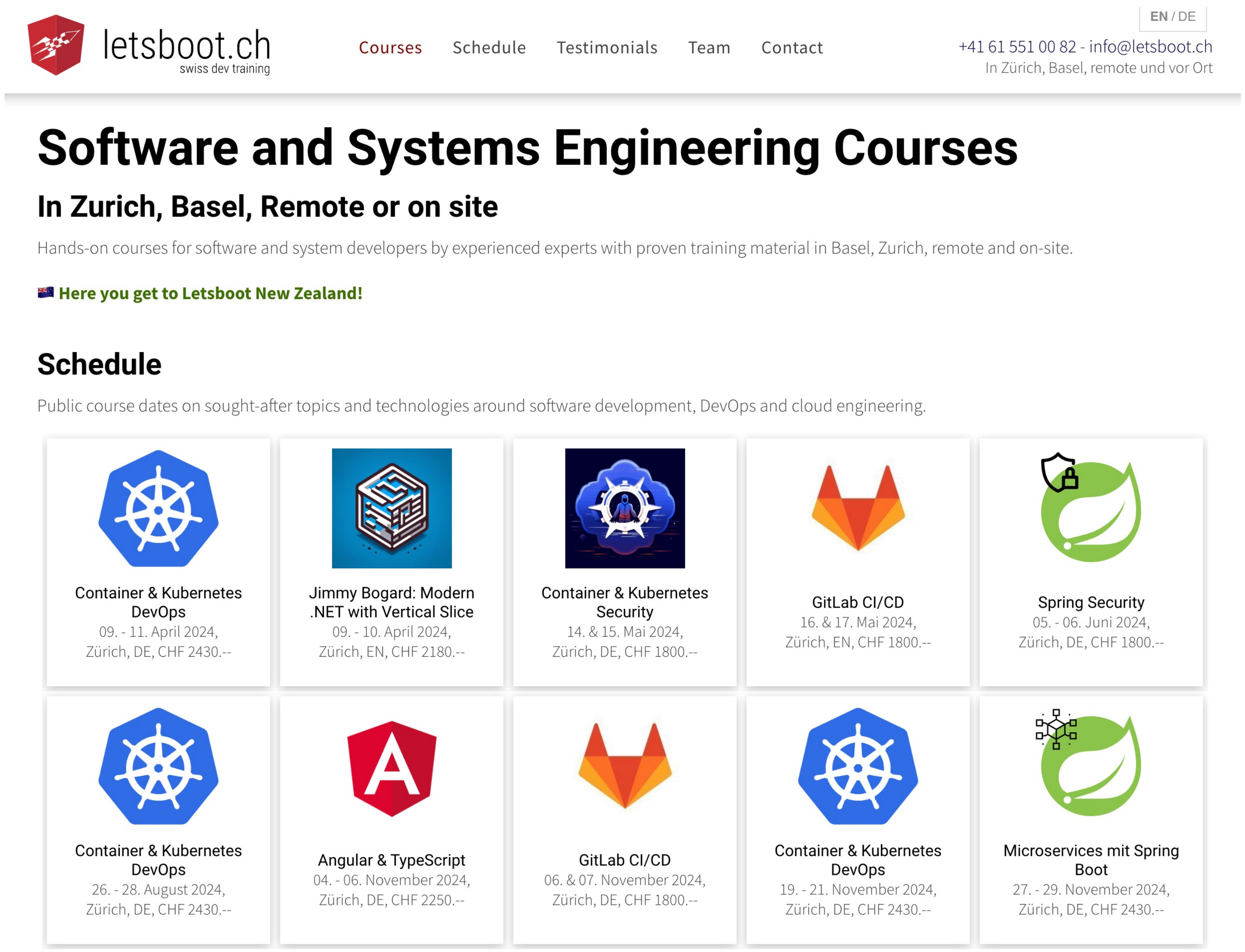
- Please note that I am wearing hearing aids.
- I may not always understand you immediately.

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

Use Case

Use Case: Letsboot Website

- Letsboot offers a wide range of courses and uses a web-based application that allows interested parties to browse and register for the current courses offered (→ Example).
- At the same time, only authorised administrators should be able to **manage the courses** on offer.



letsboot.ch
swiss dev training

Courses Schedule Testimonials Team Contact

EN / DE

+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.--

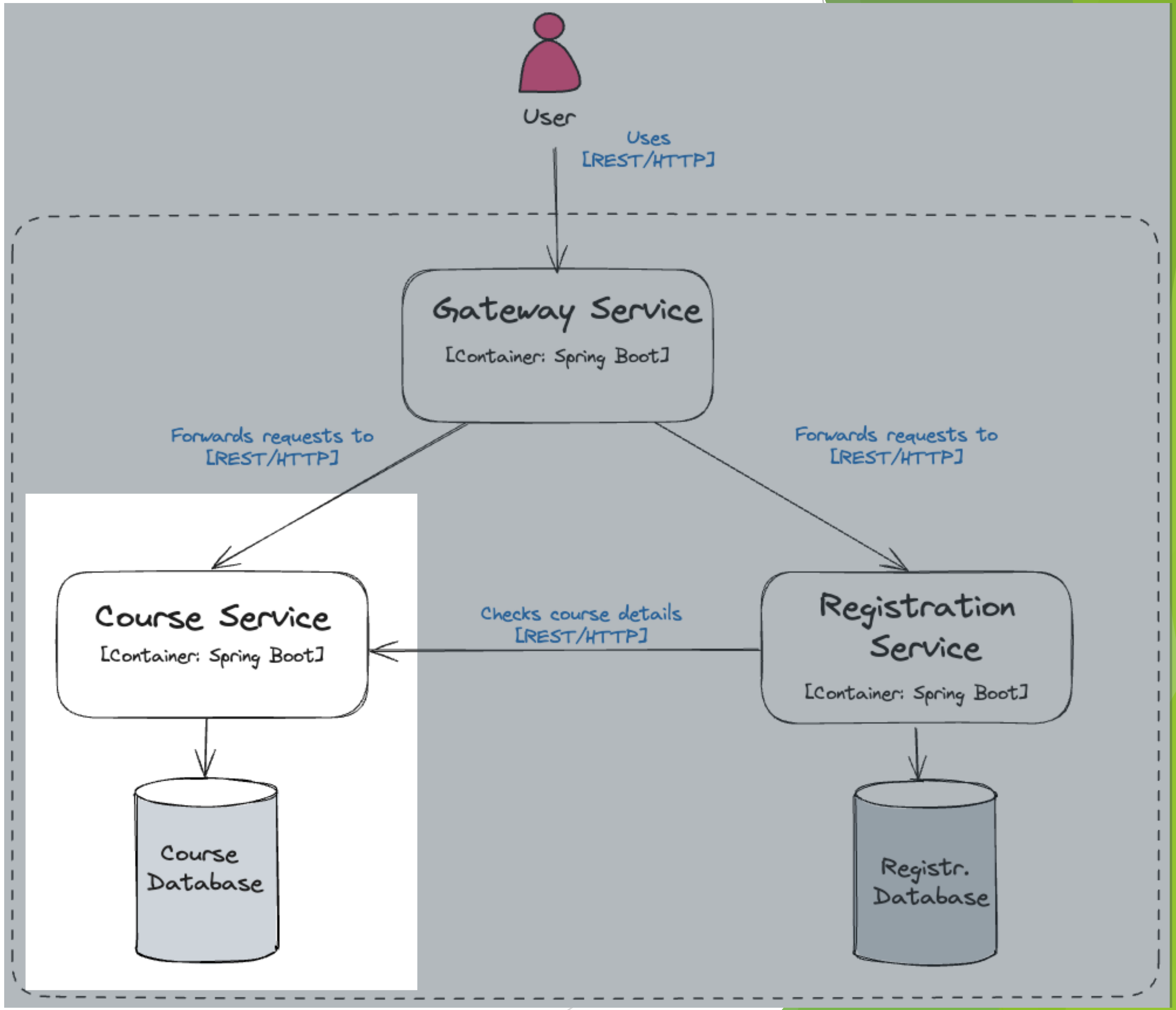
Angular & TypeScript
04. - 06. November 2024,
Zürich, DE, CHF 2250.--

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

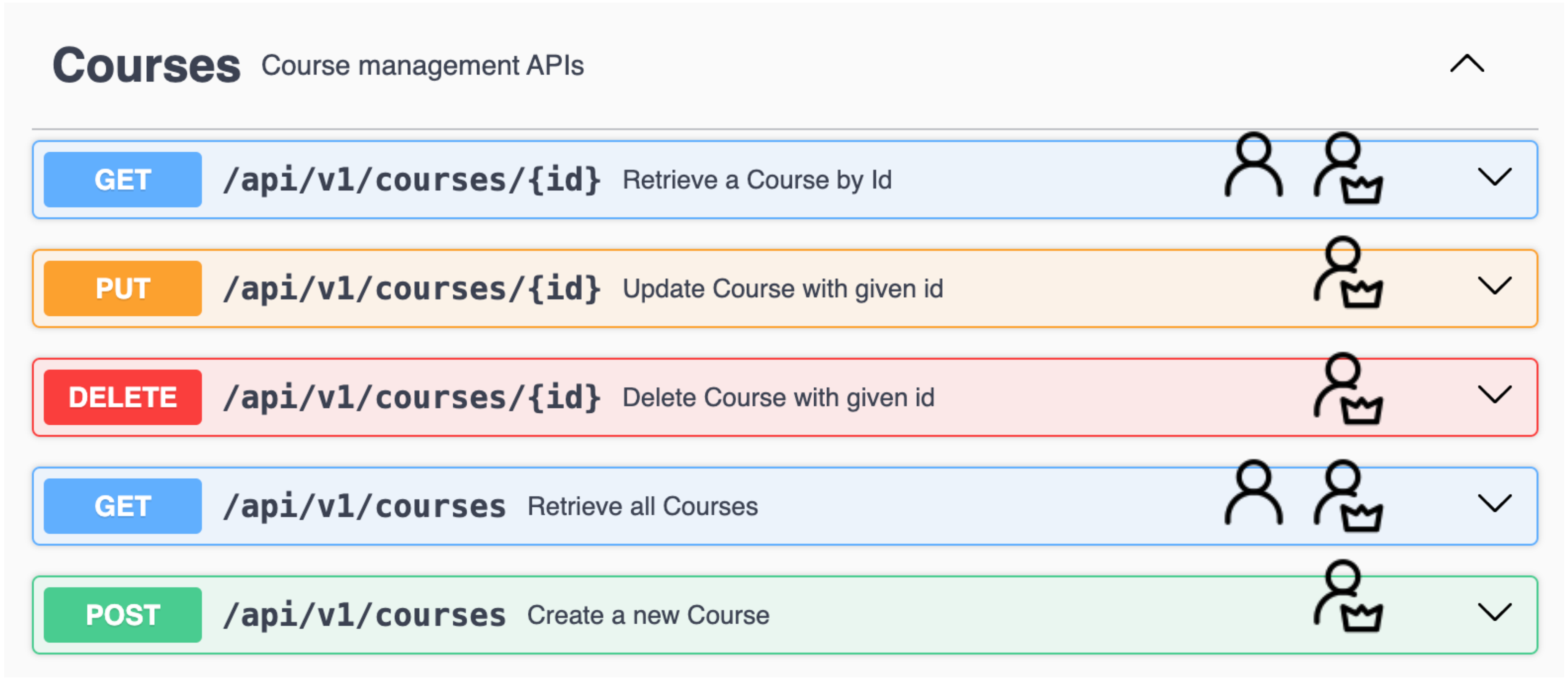
Container & Kubernetes DevOps
19. - 21. November 2024,
Zürich, DE, CHF 2430.--

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

Use Case: Course Management of letsboot.ch



Use Case: Course Management API



Courses Course management APIs

GET `/api/v1/courses/{id}` Retrieve a Course by Id

PUT `/api/v1/courses/{id}` Update Course with given id

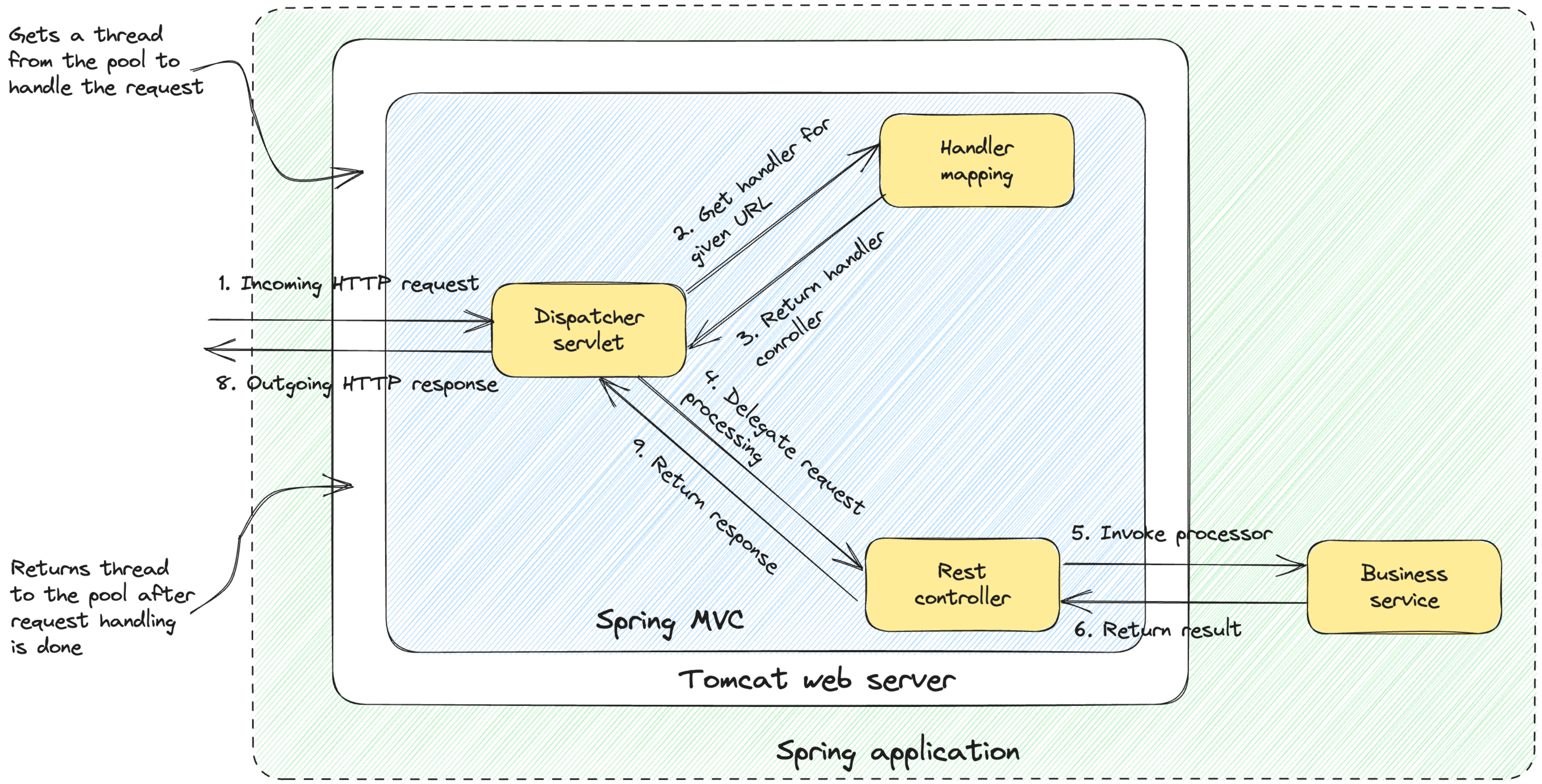
DELETE `/api/v1/courses/{id}` Delete Course with given id

GET `/api/v1/courses` Retrieve all Courses

POST `/api/v1/courses` Create a new Course

Let's dive in!

Request Flow in Spring MVC


Gets a thread from the pool to handle the request
1. Incoming HTTP request
8. Outgoing HTTP response
Returns thread to the pool after request handling is done
Dispatcher servlet
2. Get handler for given URL
Handler mapping
3. Return handler conroller
4. Delegate request processing
9. Return response
Rest controller
5. Invoke processor
6. Return result
Business service
Spring MVC
Tomcat web server
Spring application

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>
```

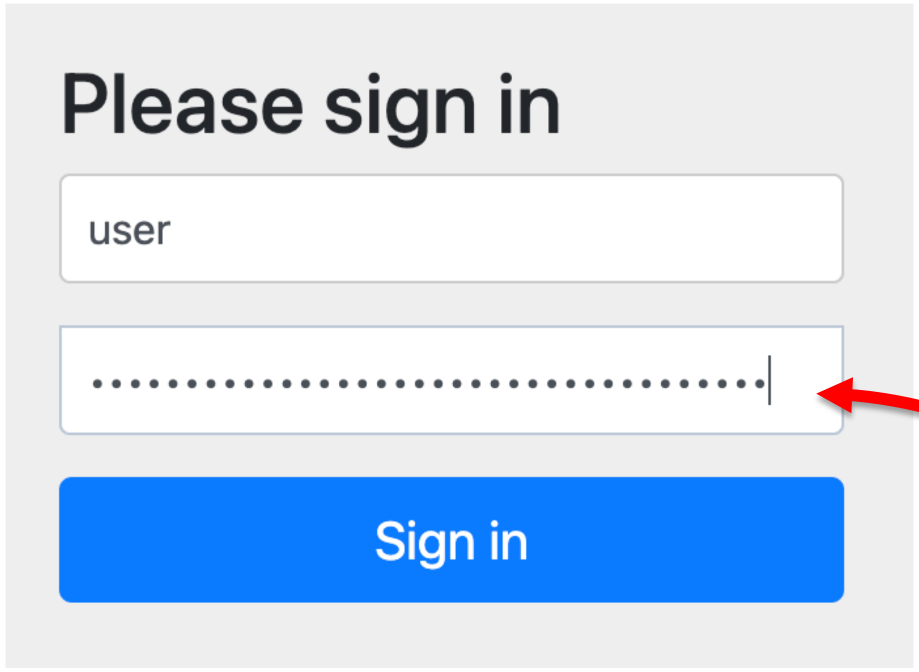
Impact of this Dependency

- 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

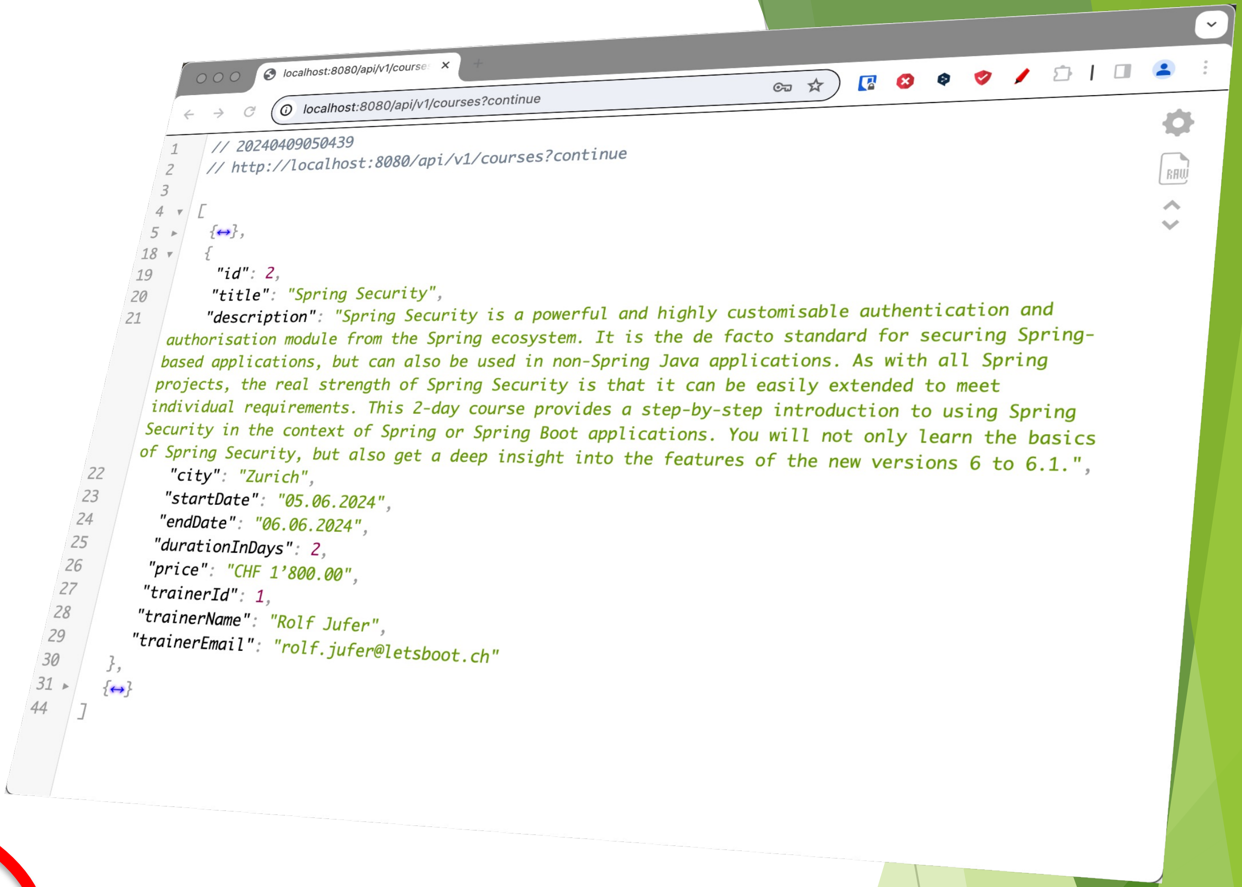
Demo 1

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


Please sign in

user

••••••••••••••••••••••••••••••••••••


localhost:8080/api/v1/courses
localhost:8080/api/v1/courses?continue

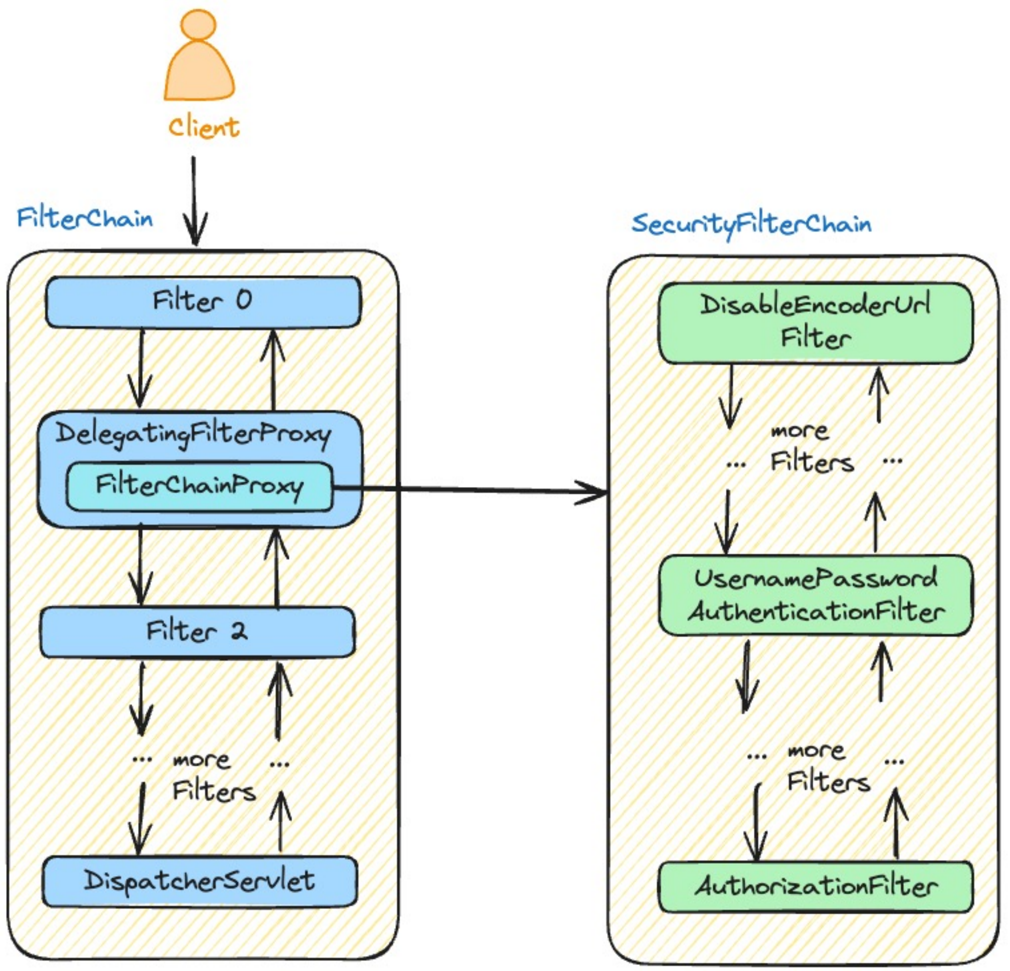
```
// 20240409050439
// http://localhost:8080/api/v1/courses?continue

[
  {↔},
  {
    "id": 2,
    "title": "Spring Security",
    "description": "Spring Security is a powerful and highly customisable authentication and authorisation module from the Spring ecosystem. It is the de facto standard for securing Spring-based applications, but can also be used in non-Spring Java applications. As with all Spring projects, the real strength of Spring Security is that it can be easily extended to meet individual requirements. This 2-day course provides a step-by-step introduction to using Spring Security in the context of Spring or Spring Boot applications. You will not only learn the basics of Spring Security, but also get a deep insight into the features of the new versions 6 to 6.1.",
    "city": "Zurich",
    "startDate": "05.06.2024",
    "endDate": "06.06.2024",
    "durationInDays": 2,
    "price": "CHF 1’800.00",
    "trainerId": 1,
    "trainerName": "Rolf Jufer",
    "trainerEmail": "rolf.jufer@letsboot.ch"
  },
  {↔}
]
```

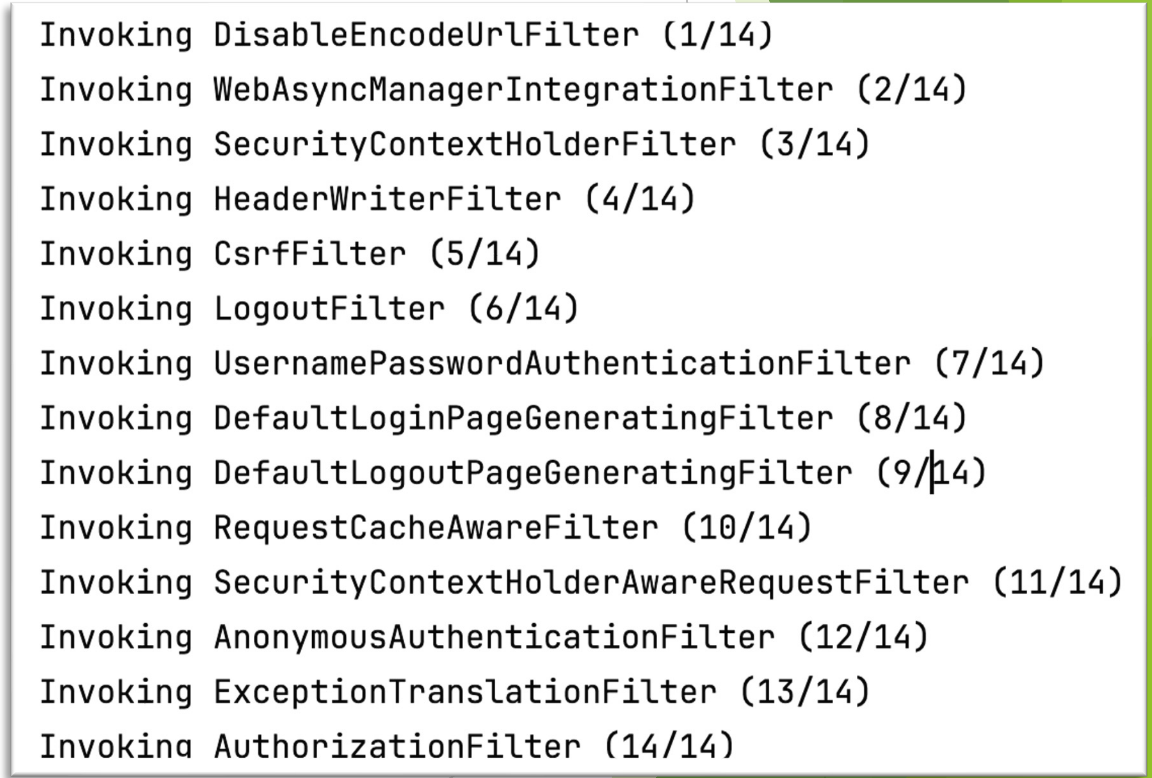
```
Using generated security password: 9cff30f3-8795-4eea-8d39-47389e9e3319

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

Behind the Scenes: SecurityFilterChain



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

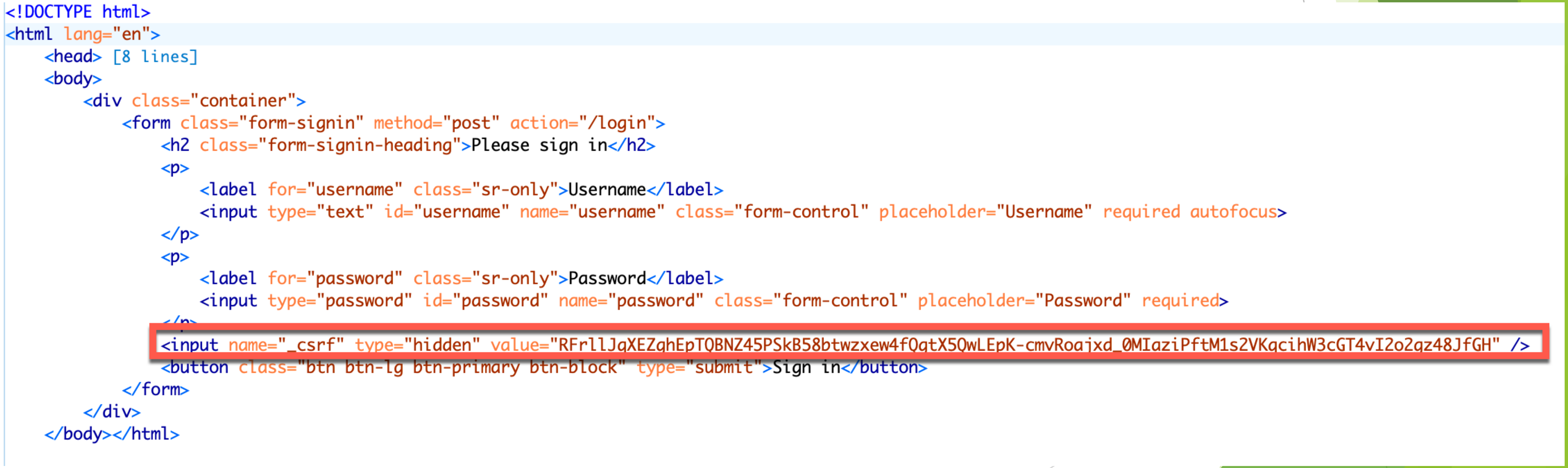


```
Invoking DisableEncodeUrlFilter (1/14)
Invoking WebAsyncManagerIntegrationFilter (2/14)
Invoking SecurityContextHolderFilter (3/14)
Invoking HeaderWriterFilter (4/14)
Invoking CsrfFilter (5/14)
Invoking LogoutFilter (6/14)
Invoking UsernamePasswordAuthenticationFilter (7/14)
Invoking DefaultLoginPageGeneratingFilter (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

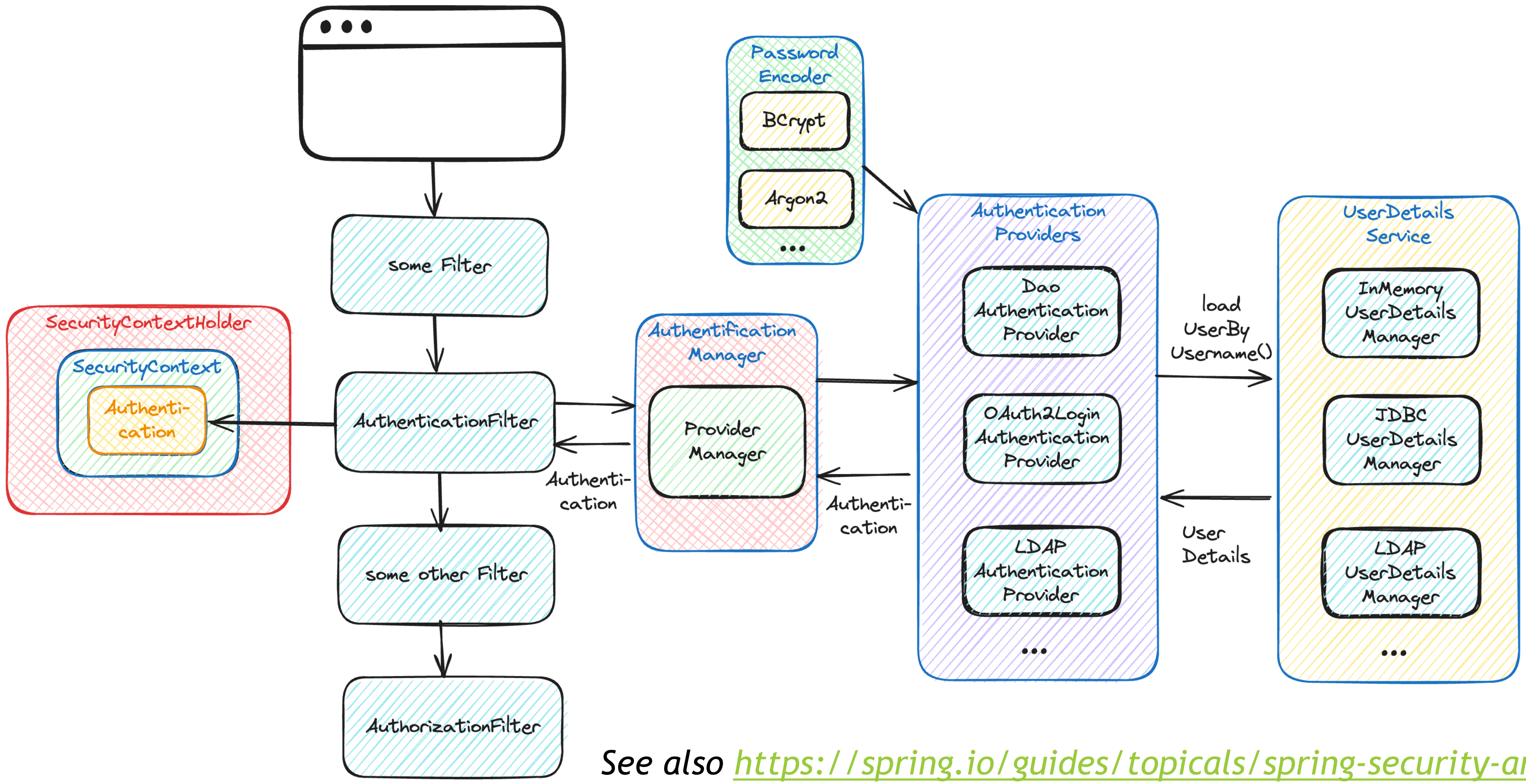
- Adding the `spring-boot-starter-security` dependency also enables CSRF protection by default.



```
<!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.

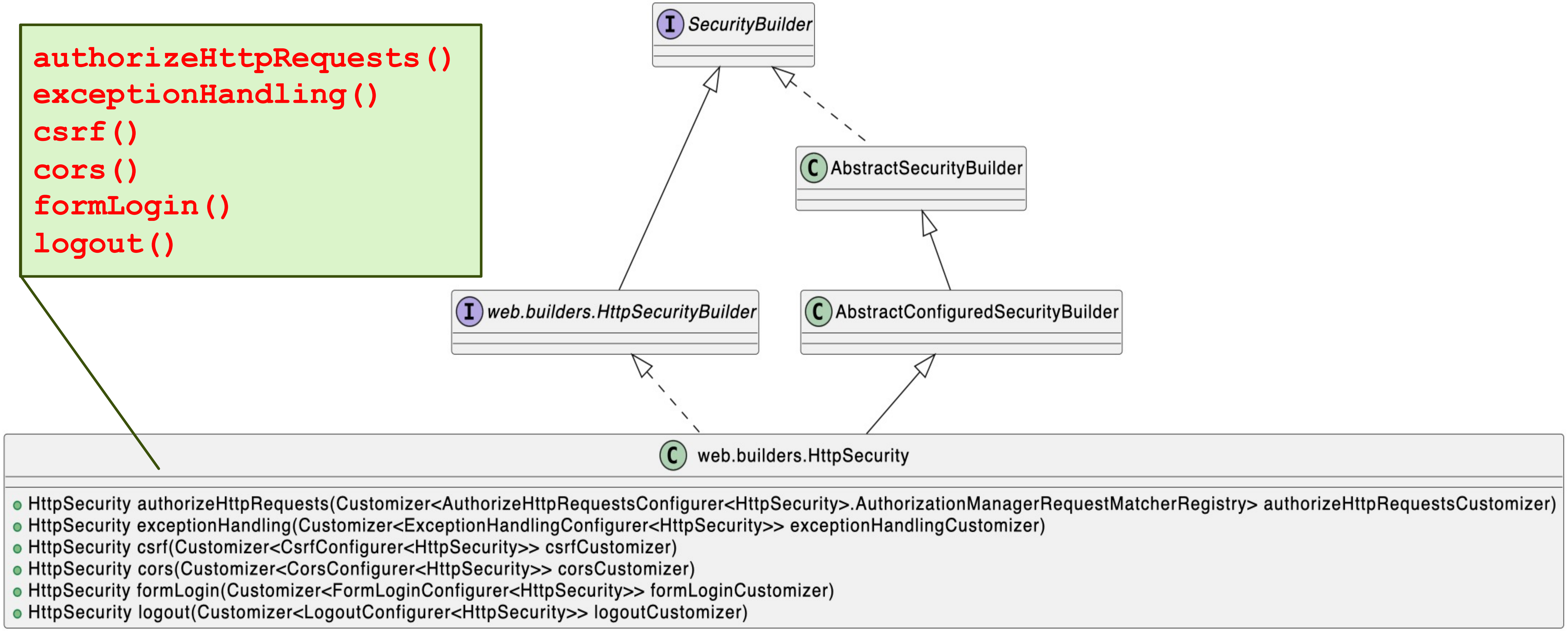
- Allows all GET requests to paths starting with "/api/v1/courses/" for any user.
- However, POST requests to the same paths are only permitted for users with the "ADMIN" role. All other requests require authentication.

```
@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();
    }
}
```

HttpSecurity: Security Rules for Endpoints

```
authorizeHttpRequests()
exceptionHandling()
csrf()
cors()
formLogin()
logout()
```



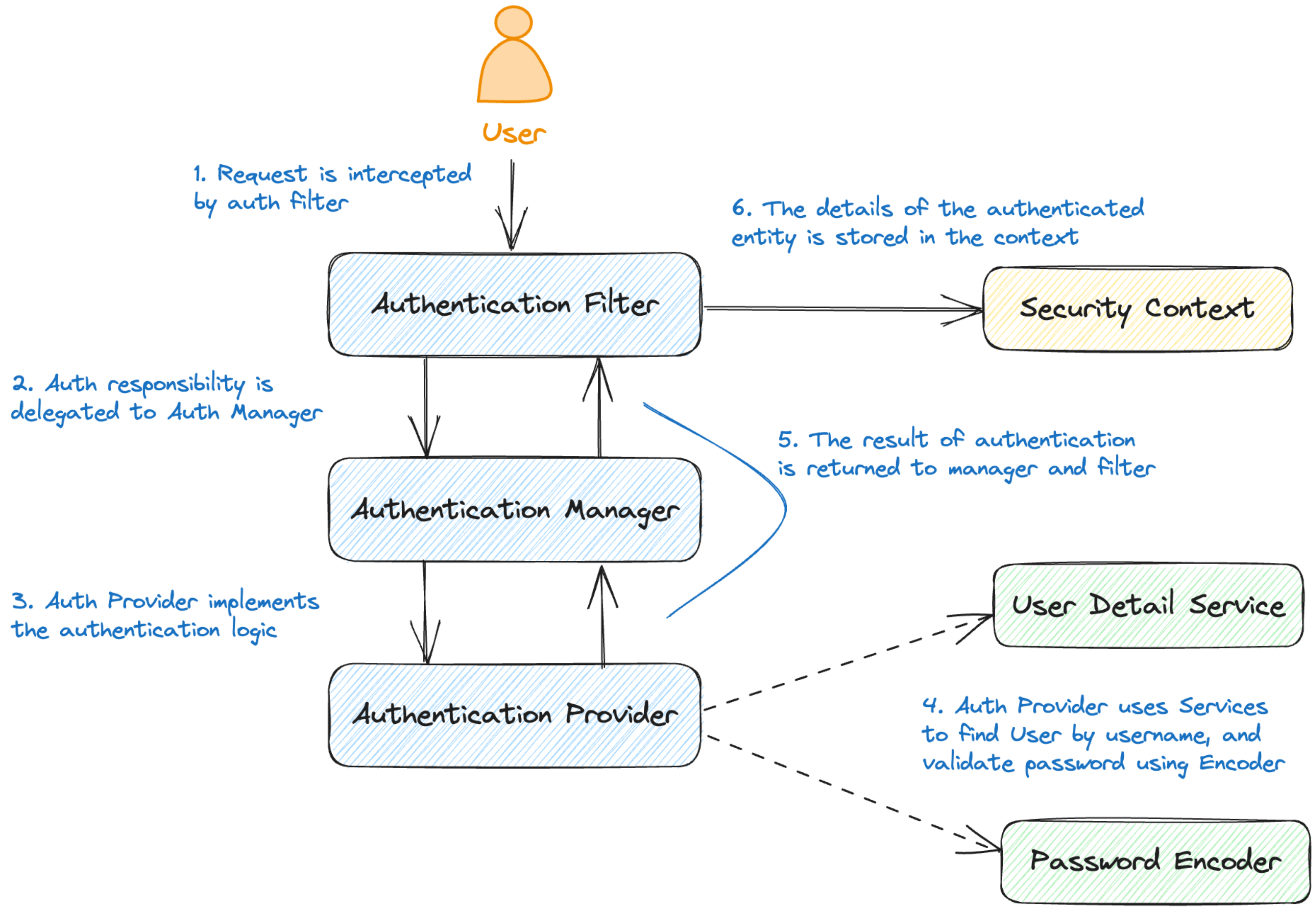
SecurityBuilder
AbstractSecurityBuilder
web.builders.HttpSecurityBuilder
AbstractConfiguredSecurityBuilder
web.builders.HttpSecurity

- HttpSecurity authorizeHttpRequests(Customizer<AuthorizeHttpRequestsConfigurer<HttpSecurity>.AuthorizationManagerRequestMatcherRegistry> authorizeHttpRequestsCustomizer)
- HttpSecurity exceptionHandling(Customizer<ExceptionHandlingConfigurer<HttpSecurity>> exceptionHandlingCustomizer)
- HttpSecurity csrf(Customizer<CsrfConfigurer<HttpSecurity>> csrfCustomizer)
- HttpSecurity cors(Customizer<CorsConfigurer<HttpSecurity>> corsCustomizer)
- HttpSecurity formLogin(Customizer<FormLoginConfigurer<HttpSecurity>> formLoginCustomizer)
- HttpSecurity logout(Customizer<LogoutConfigurer<HttpSecurity>> logoutCustomizer)

Note: Request- vs. Method-Level Autho-rization

- The previous example models authorisation at the **request level**.
- There is also the option of modelling authorisation at the **method level**, with annotations such as:
  - *@PreAuthorize("hasAuthority('ADMIN')")*
- Due to time constraints, we will not discuss this option further.

Authentication Process


User
1. Request is intercepted by auth filter
Authentication Filter
6. The details of the authenticated entity is stored in the context
Security Context
2. Auth responsibility is delegated to Auth Manager
Authentication Manager
5. The result of authentication is returned to manager and filter
3. Auth Provider implements the authentication logic
User Detail Service
Authentication Provider
4. Auth Provider uses Services to find User by username, and validate password using Encoder
Password Encoder

Authentication Mechanisms

- Spring Security provides comprehensive support for Authentication. Example:

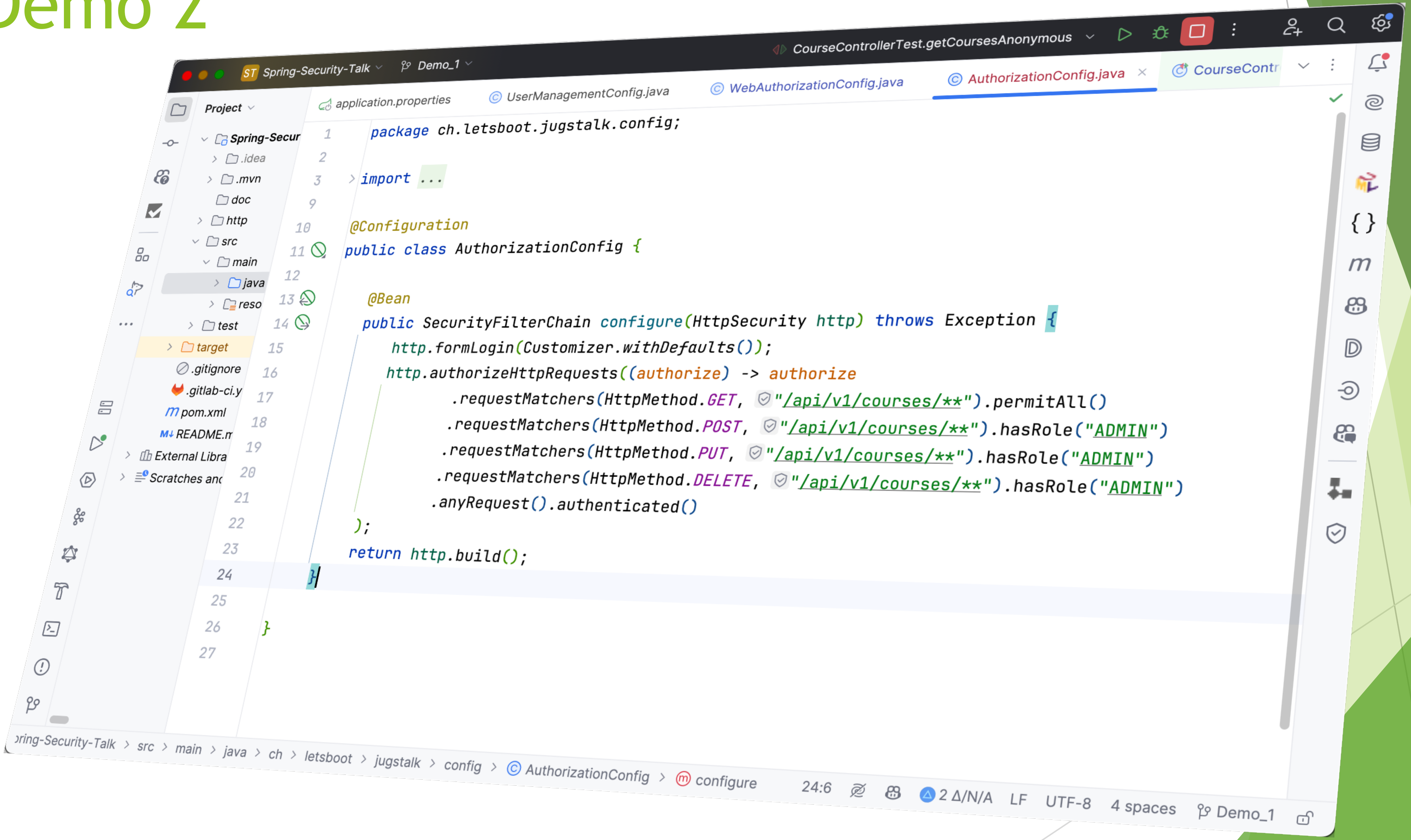
```java
@Configuration
public class UserManagementConfig {

  @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



```java
package ch.letsboot.jugstalk.config;

import ...

@Configuration
public class AuthorizationConfig {

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

}
```

Who provides UserManagement?

- The demo 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
  - security risks
  - lack of standardisation
  - and scalability issues

OAuth 2.0 and OpenID Connect

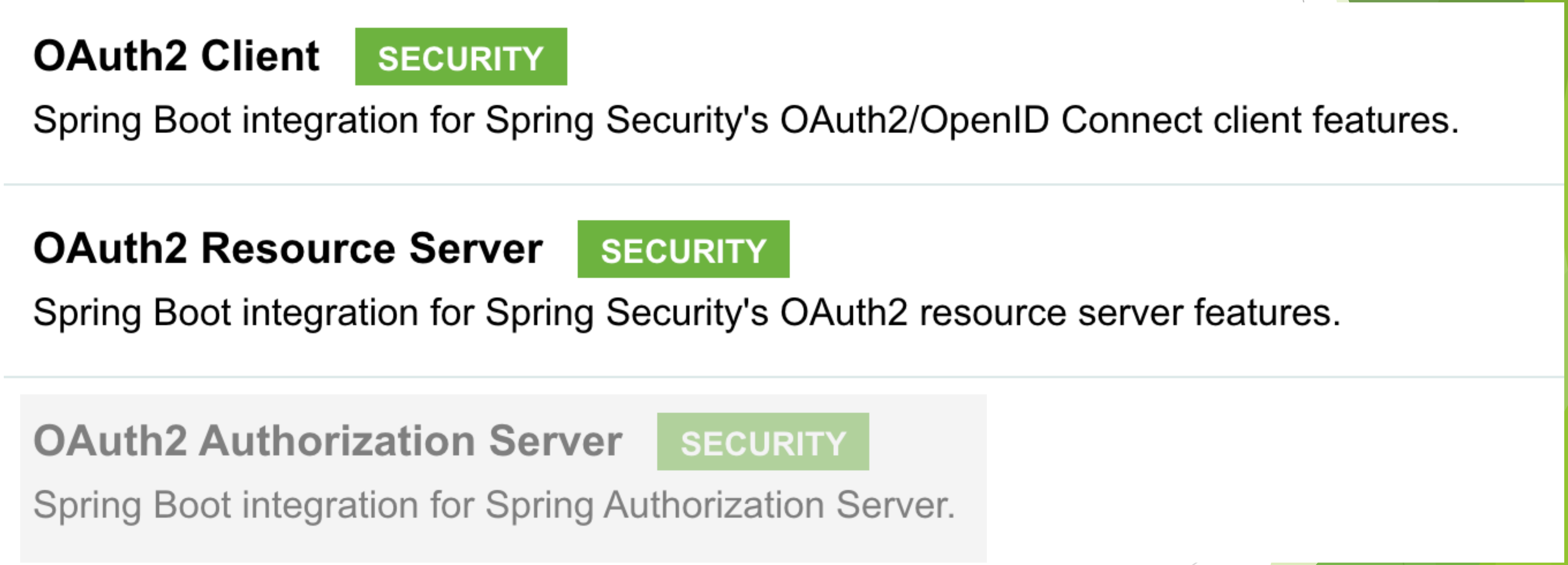
Motivation

- **Balancing security** and **user convenience** is key in authentication. Managing multiple credentials for various apps can be cumbersome and disrupt user experience.
- **OAuth 2.0** and **OpenID Connect** offer a robust framework for authentication and authorization, promoting both security and user convenience across diverse applications.

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.
  - **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.

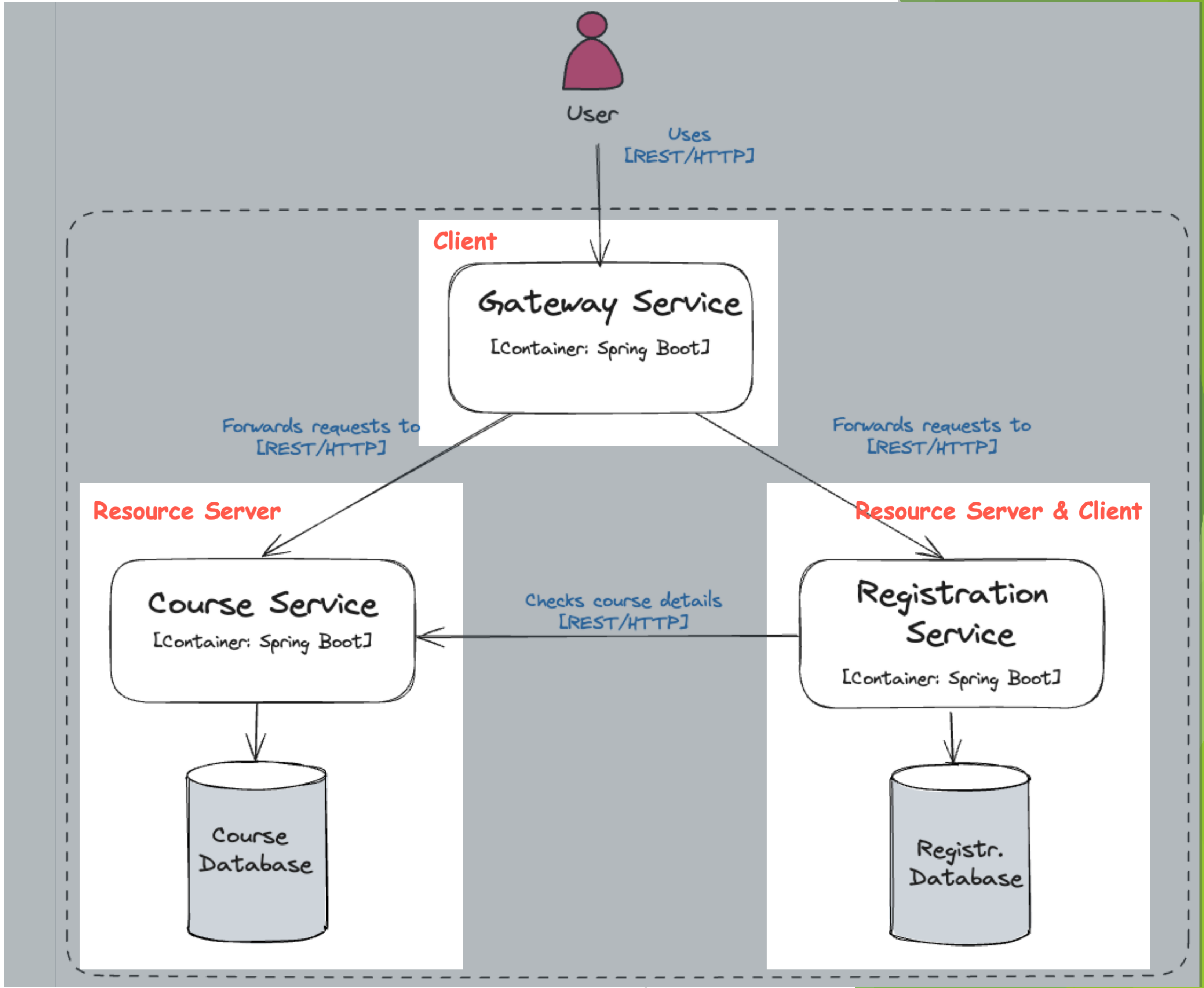
OAuth2 Resource Server SECURITY

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

OAuth2 Authorization Server SECURITY

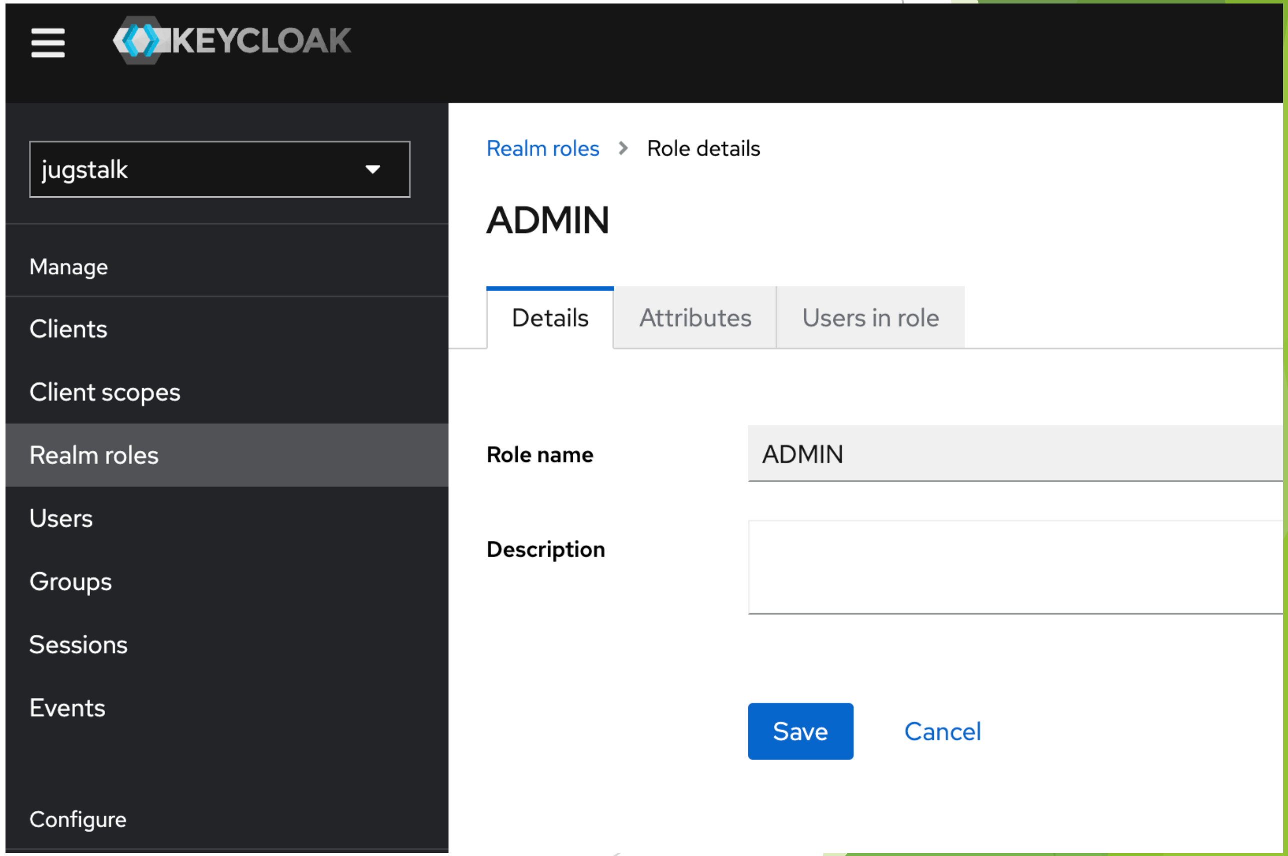
Spring Boot integration for Spring Authorization Server.

Example OAuth2 Roles



OAuth2 Authorization Server

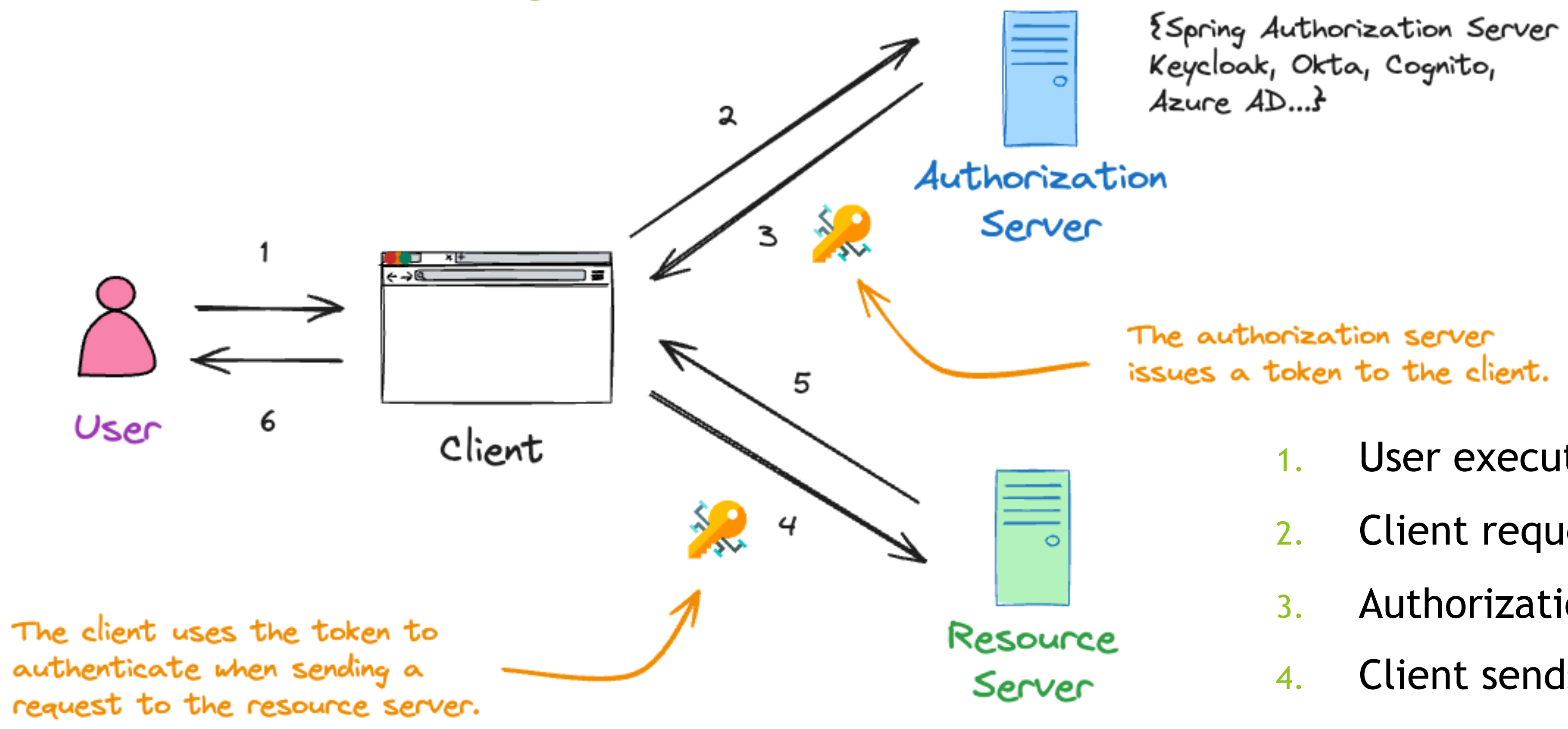
- Instead of Spring Security's native Authorization Server, we may use Keycloak (as it is widely used in practice).



ADMIN

Role name	ADMIN
Description	

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



{Spring Authorization Server
Keycloak, Okta, Cognito,
Azure AD...}
Authorization Server
2
3
1
6
5
4
User
Client
The authorization server issues a token to the client.
Resource Server
The client uses the token to authenticate when sending a request to the resource server.

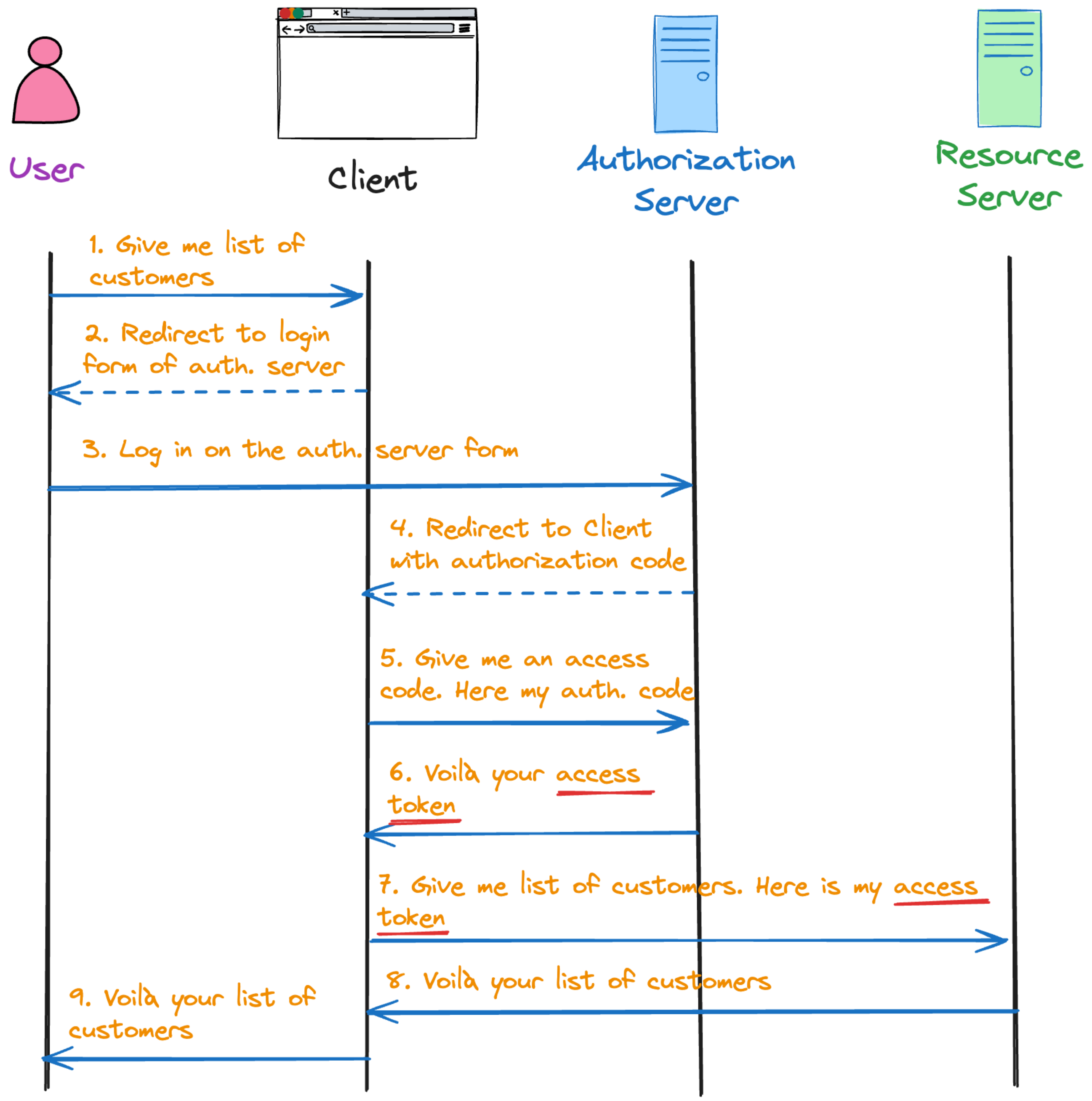
1. User executes use case.
2. Client requests token.
3. Authorization server issues token.
4. Client sends requests.
5. Resource server authorizes and executes.
6. Client displays result.

Getting a Token from Authorization Server with OAuth2

- 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 (--> *access token, see next slide*)
  - authorization code grant type with PKCE ("pixy", proof key for code exchange)
  - client credentials grant type

Also see: https://oauth.net/2/grant-types/

Example Authorization Code Grant Type


User
Client
Authorization Server
Resource Server
1. Give me list of customers
2. Redirect to login form of auth. server
3. Log in on the auth. server form
4. Redirect to Client with authorization code
5. Give me an access code. Here my auth. code
6. Voilà your access token
7. Give me list of customers. Here is my access token
8. Voilà your list of customers
9. Voilà your list of customers

Access Tokens

	Access Tokens (OAuth 2.0)
Purpose	**Access tokens** are used for authorization purposes in OAuth 2.0.
Contents	Access tokens carry information about the permissions granted to the client application, such as the scope of access and possibly additional user attributes.
Usage	Access tokens are presented by the client application to the resource server to gain access to protected resources.

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>
```

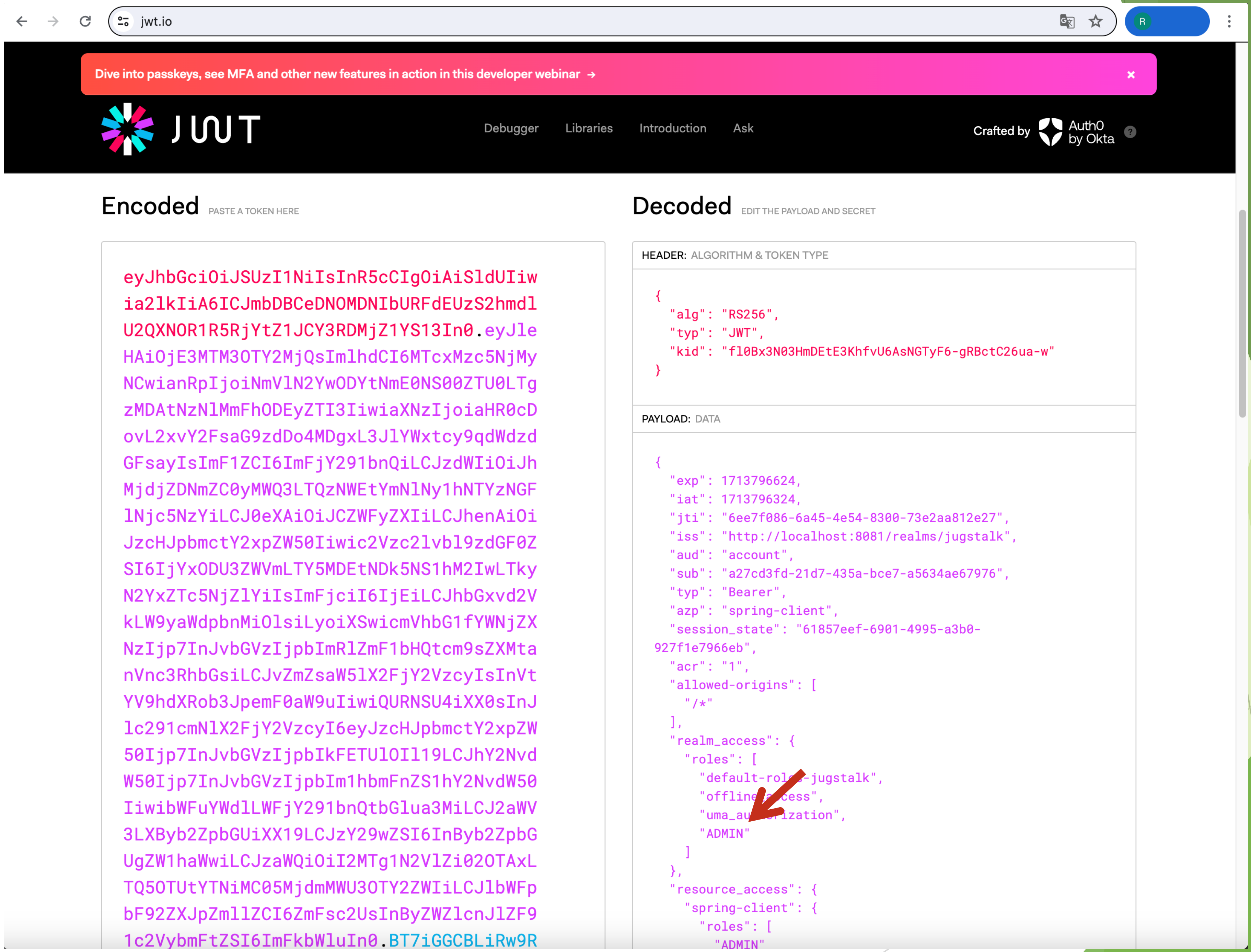
Example Response from Authorization Server

*encoded JWT (see next slide) ***

```
{
  "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.io
Dive into passkeys, see MFA and other new features in action in this developer webinar →
JWT
Debugger
Libraries
Introduction
Ask
Crafted by Auth0 by Okta

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiw
ia2lkIiA6ICJmbDBCeDNOMDNIbURFdEUzS2hmdl
U2QXNOR1R5RjYtZ1JCY3RDMjZ1YS13In0.eyJle
HAiOjE3MTM3OTY2MjQsImlhdCI6MTcxMzc5NjMy
NCwianRpIjoiNmVlN2YwODYtNmE0NS00ZTU0LTg
zMDAtNzNlMmFhODEyZTI3IiwiaXNzIjoiaHR0cD
ovL2xvY2FsaG9zdDo4MDgxL3JlYWxtcy9qdWdzd
GFsayIsImF1ZCI6ImFjY291bnQiLCJzdWIiOiJh
MjdjZDNmZC0yMWQ3LTQzNWEtYmNlNy1hNTYzNGF
lNjc5NzYiLCJ0eXAiOiJCZWFyZXIiLCJhenAiOi
JzcHJpbmctY2xpZW50Iiwic2Vzc2lvbl9zdGF0Z
SI6IjYxODU3ZWVmLTY5MDEtNDk5NS1hM2IwLTky
N2YxZTc5NjZlYiIsImFjciI6IjEiLCJhbGxvd2V
kLW9yaWdpbnMiOlsiLyoiXSwicmVhbG1fYWNjZX
NzIjp7InJvbGVzIjpbImRlZmF1bHQtcm9sZXMta
nVnc3RhbGsiLCJvZmZsaW5lX2FjY2VzcyIsInVt
YV9hdXRob3JpemF0aW9uIiwiQURNSU4iXX0sInJ
lc291cmNlX2FjY2VzcyI6eyJzcHJpbmctY2xpZW
50Ijp7InJvbGVzIjpbIkFETUlOIl19LCJhY2Nvd
W50Ijp7InJvbGVzIjpbIm1hbmFnZS1hY2NvdW50
IiwibWFuYWdlLWFjY291bnQtbGlua3MiLCJ2aWV
3LXByb2ZpbGUiXX19LCJzY29wZSI6InByb2ZpbG
UgZW1haWwiLCJzaWQiOiI2MTg1N2VlZi02OTAxL
TQ5OTUtYTNiMC05MjdmMWU3OTY2ZWIiLCJlbWFp
bF92ZXJpZmllZCI6ZmFsc2UsInByZWZlcnJlZF9
1c2VybmFtZSI6ImFkbWluIn0.BT7iGGCBLiRw9R
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "fl0Bx3N03HmDEtE3KhfvU6AsNGTyF6-gRBctC26ua-w"
}
```

PAYLOAD: DATA

```
{
  "exp": 1713796624,
  "iat": 1713796324,
  "jti": "6ee7f086-6a45-4e54-8300-73e2aa812e27",
  "iss": "http://localhost:8081/realms/jugstalk",
  "aud": "account",
  "sub": "a27cd3fd-21d7-435a-bce7-a5634ae67976",
  "typ": "Bearer",
  "azp": "spring-client",
  "session_state": "61857eef-6901-4995-a3b0-
927f1e7966eb",
  "acr": "1",
  "allowed-origins": [
    "/*"
  ],
  "realm_access": {
    "roles": [
      "default-roles-jugstalk",
      "offline_access",
      "uma_authorization",
      "ADMIN"
    ]
  },
  "resource_access": {
    "spring-client": {
      "roles": [
        "ADMIN"
```

Revised SecurityFilterChain Bean

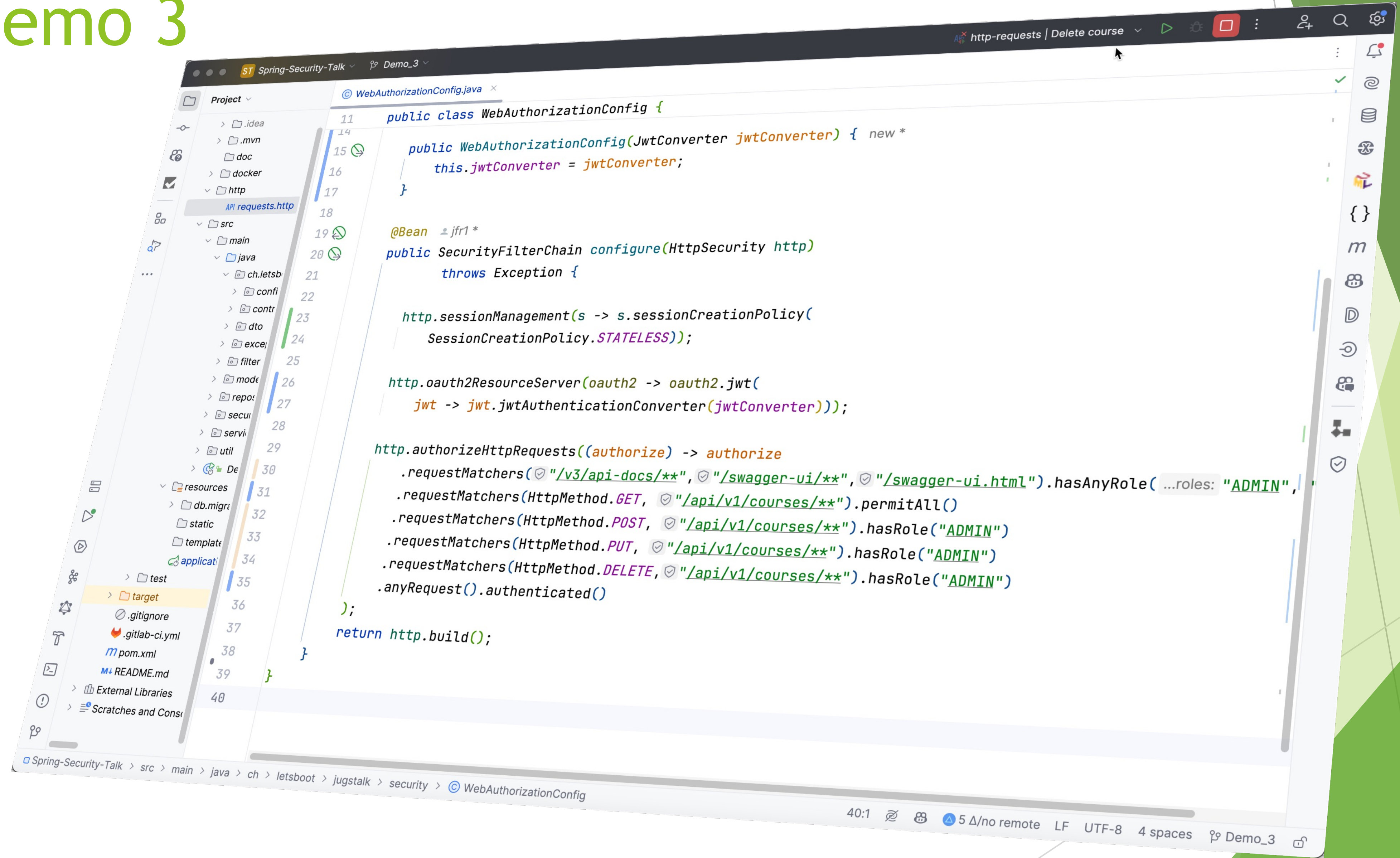
```
@Configuration
public class AuthorizationConfig {

    @Bean
    public SecurityFilterChain configure(HttpSecurity http) throws Exception {
        // ~~http.formLogin(Customizer.withDefaults());~~
        // 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

Demo 3



```java
public class WebAuthorizationConfig {

    public WebAuthorizationConfig(JwtConverter jwtConverter) {  new *
        this.jwtConverter = jwtConverter;
    }

    @Bean  jfr1 *
    public SecurityFilterChain configure(HttpSecurity http)
            throws Exception {

        http.sessionManagement(s -> s.sessionCreationPolicy(
                SessionCreationPolicy.STATELESS));

        http.oauth2ResourceServer(oauth2 -> oauth2.jwt(
                jwt -> jwt.jwtAuthenticationConverter(jwtConverter)));

        http.authorizeHttpRequests((authorize) -> authorize
                .requestMatchers("/v3/api-docs/**", "/swagger-ui/**", "/swagger-ui.html").hasAnyRole( ...roles: "ADMIN", '
                .requestMatchers(HttpMethod.GET, "/api/v1/courses/**").permitAll()
                .requestMatchers(HttpMethod.POST, "/api/v1/courses/**").hasRole("ADMIN")
                .requestMatchers(HttpMethod.PUT, "/api/v1/courses/**").hasRole("ADMIN")
                .requestMatchers(HttpMethod.DELETE, "/api/v1/courses/**").hasRole("ADMIN")
                .anyRequest().authenticated()
        );
        return http.build();
    }
}
```

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())
                .oauth2Login(withDefaults())
                .oauth2ResourceServer((oauth2) -> oauth2.jwt(Customizer.withDefaults()));
        return http.build();
    }
}
```

Demos and more Details about Oauth2 Client

- Due to time constraints, I am not able 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!