Week 5: Security

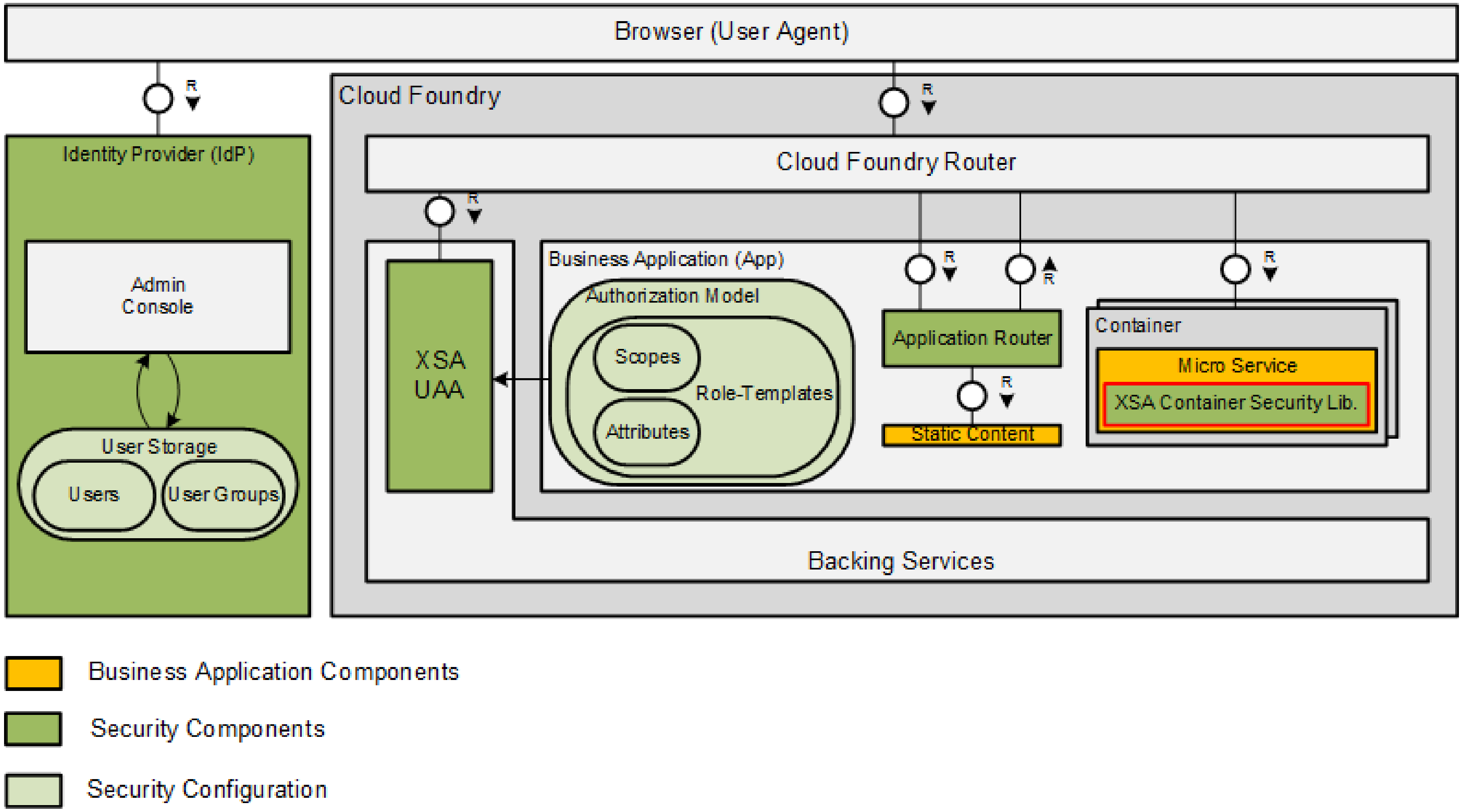
Unit 4: Making Your Application Secure – Part I

Run Simple

openSAP
open.sap.com

Making Your Application Secure – Part I

Domain-specific authorizations – Defining authorizations for business functions



Making Your Application Secure – Part I

SAP java-container-security library (based on Spring security)

Spring security and Spring security OAuth

- Ensures that application calls are authenticated and authorized
- Enablement through servlet filter chain (Example)
- Supports offline JWT token validation
- ...

SAP java-container-security library

- Extends Spring security & Spring security OAuth
- Allows domain-specific authorization checks e.g. "Change Cost Center"
- Implements offline JWT token validation

Making Your Application Secure – Part I

Spring security – Register Spring security filter-chain

Activate security by registering springSecurityFilterChain servlet filter

```
public class AppInitializer implements WebApplicationInitializer {


    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {

      // register Spring Security Filter-Chain
      servletContext.addFilter(AbstractSecurityWebApplicationInitializer.DEFAULT_FILTER_NAME,
          new DelegatingFilterProxy(AbstractSecurityWebApplicationInitializer.DEFAULT_FILTER_NAME))
                  .addMappingForUrlPatterns(EnumSet.allOf(DispatcherType.class), false, "/*");
  }
}
```

Making Your Application Secure – Part I

Spring security – Configure authorizations

Configure Spring security authorization checks

```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends ResourceServerConfigurerAdapter {
            @Override
            public void configure(HttpSecurity http) throws Exception {
                 http
                     .sessionManagement()
                                  .sessionCreationPolicy(SessionCreationPolicy.NEVER)
                                  .and().authorizeRequests()
                      /* from  specific to generic  url */
                     .antMatchers(HttpMethod.POST, "/api/v1/ads/**").access("#oauth2.hasScope(‘bulletinboard.Update’)")
                     .antMatchers(GET, "/", "/health").permitAll() // accepts not-authenticated users for /health endpoint
                     .anyRequest().denyAll();
}
```

- ➢ @EnableWebSecurity defines Spring security configuration (similar to security.xml)

Making Your Application Secure – Part I

Extract from the SAP Java container security library

SecurityContext.getUserInfo() : UserInfo

- Static Method – Returns an object of type **UserInfo**

userInfo.checkLocalScope(String scopeName) : Boolean

- Checks a scope of the current application – scope without prefix

userInfo.checkScope(String scopeName) : Boolean

- Checks a scope of an external application – scope prefixed by application name

userInfo.getAttribute(String attributeName) : String[]

- Returns attribute values (e.g. Cost Center) passed from authentication

userInfo.getIdentityZone() : String

- Returns identity zone which is equal to the **tenant ID**

userInfo.getDBToken() : String

- Get a personalized DB access token for connecting to the HANA DB

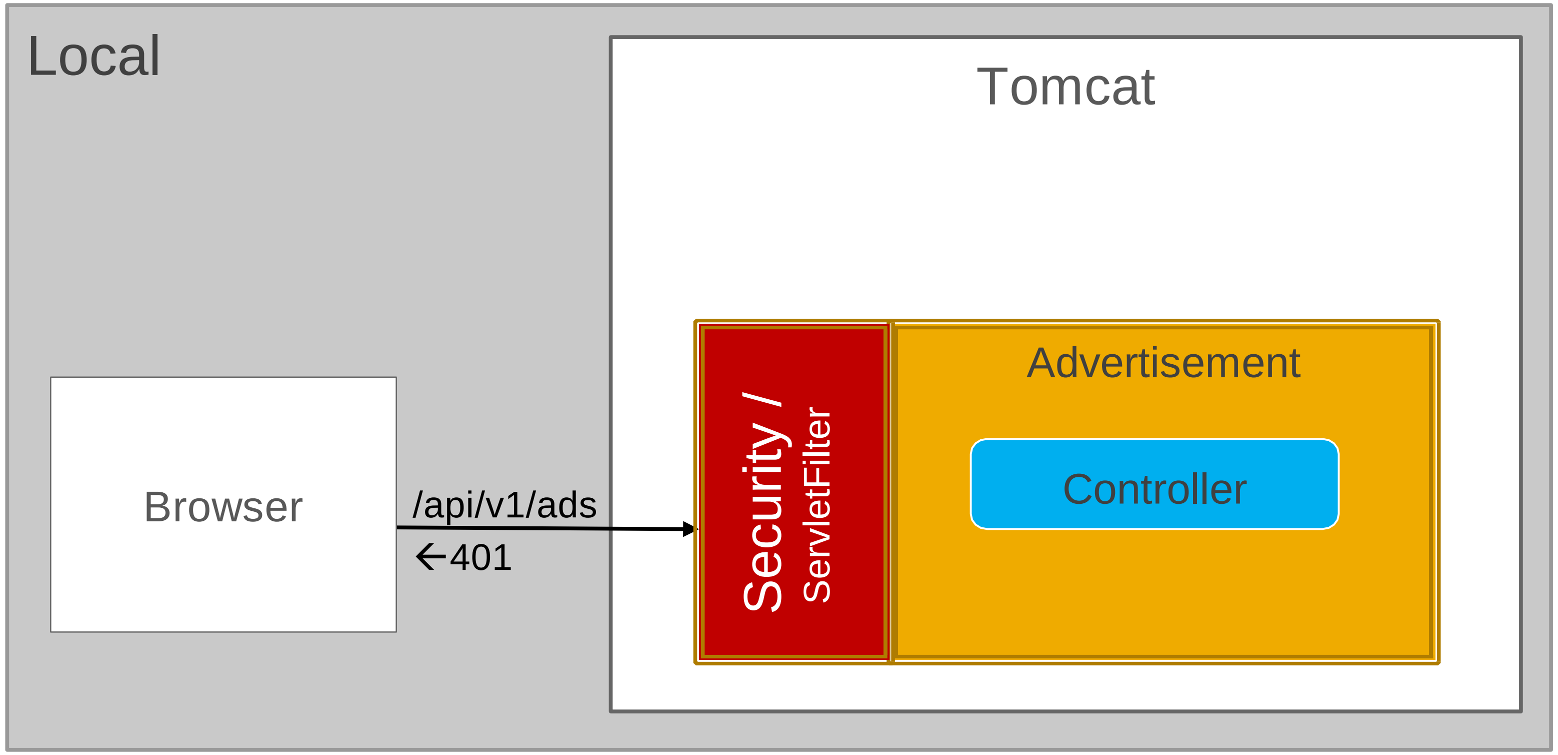
Making Your Application Secure – Part I

Demo

Making Your Application Secure – Part I

Exercise 24 Steps 1-3

Exercise 24 Steps 1-3:
Make your Application Secure



Making Your Application Secure – Part I

What you’ve learned in this unit

- How to set up domain-specific authorization checks
- Spring security
  - What it is
  - Why we use it
  - How to use it
- SAP Java container security library (based on Spring security)
  - What it is
  - Why we use it
  - How to use it

Thank you.

Contact information:

open@sap.com

Run Simple

openSAP
open.sap.com

© 2018 SAP SE or an SAP affiliate company. All rights reserved.