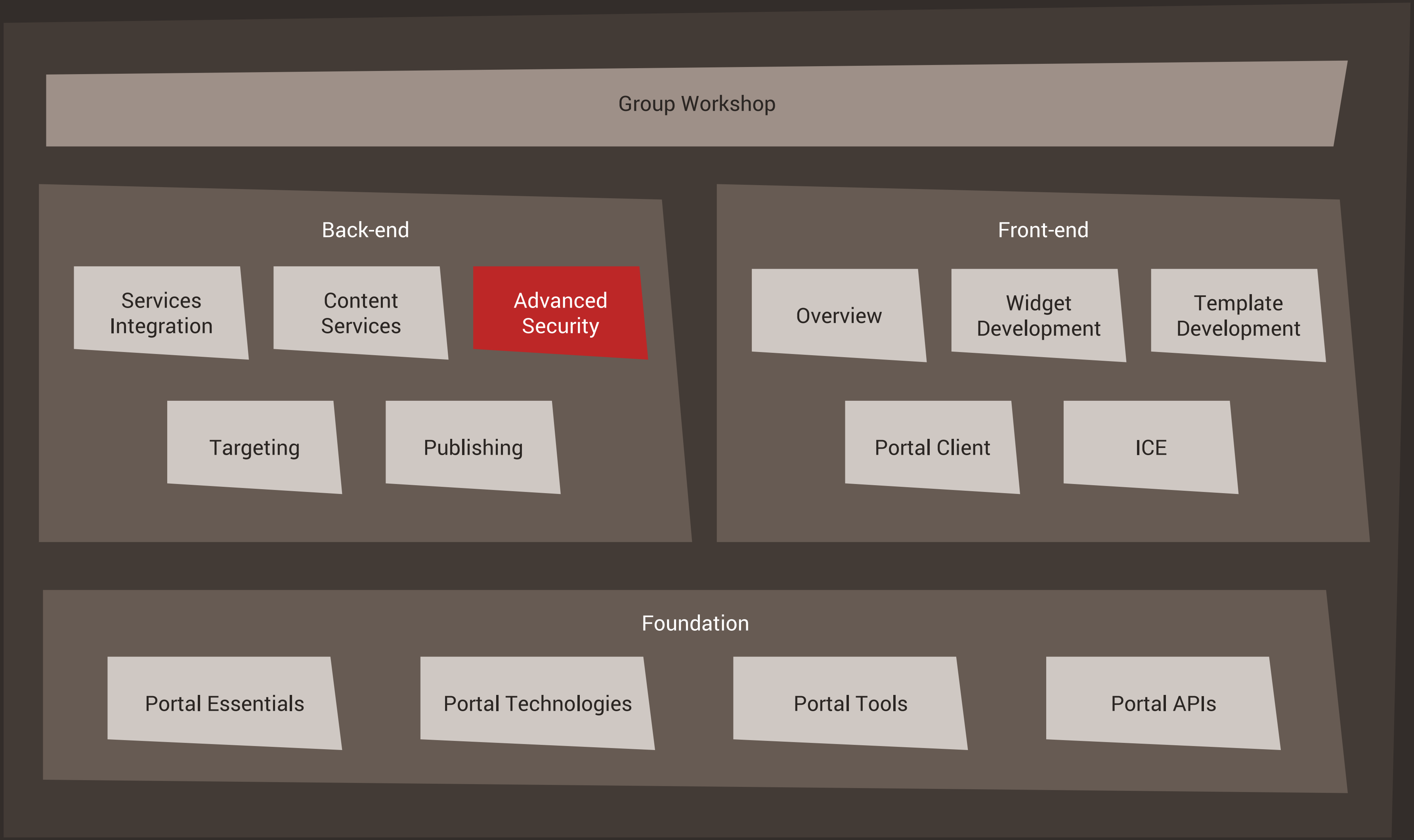
BACKBASE

# Advanced Security

Focus Area: Back-end

BACKBASE

# Training Overview



## Objectives

1. Security Concepts
2. Spring Security
3. Portal Security

# Portal Security Key Concepts

Advanced Security

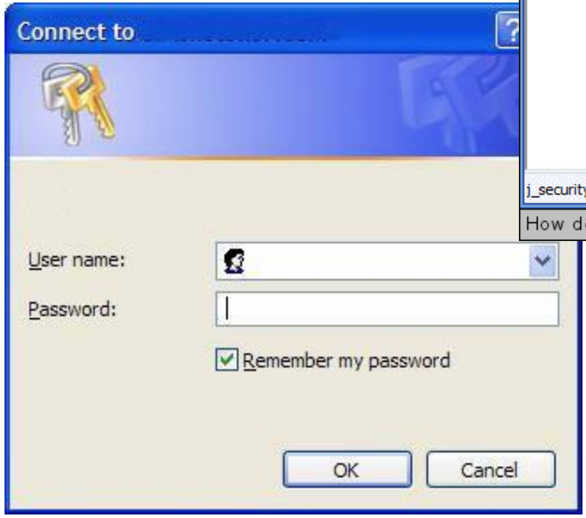
## Key Concepts

- Authentication
- Authorization
- A system administrator role

*Authentication is the process of determining whether someone or something is, in fact, who or what he/she/it is declared to be*

BACKBASE

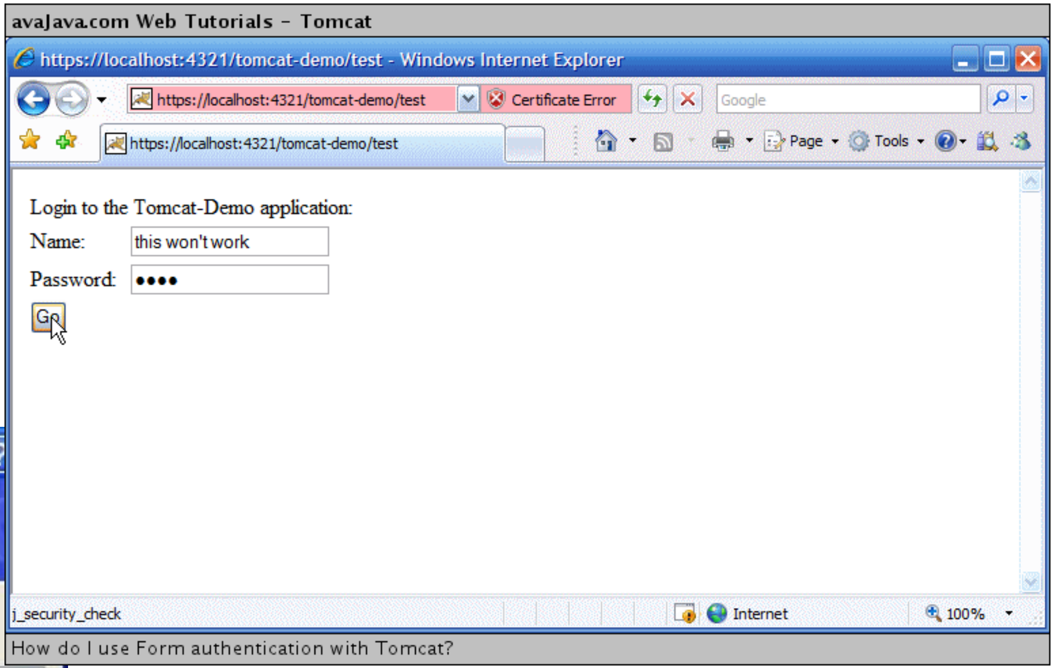
## Authentication Example (1)


Connect to

User name:

Password:

✓ Remember my password


avaJava.com Web Tutorials – Tomcat

https://localhost:4321/tomcat-demo/test - Windows Internet Explorer

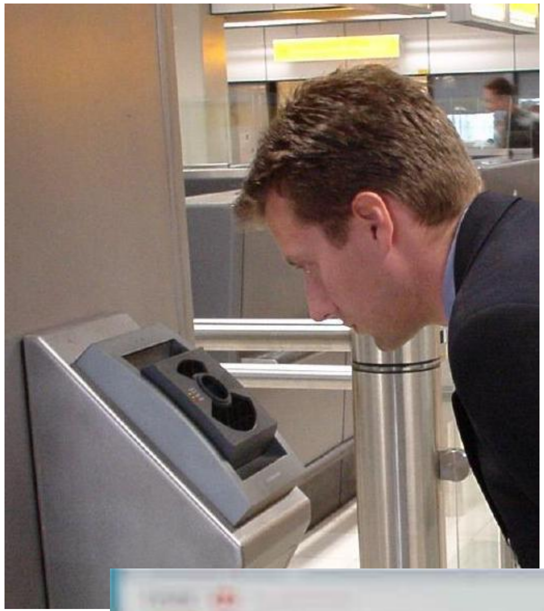
Login to the Tomcat-Demo application:

| | |
|---|---|
| Name: | this won't work |
| Password: | •••• |

j_security_check

How do I use Form authentication with Tomcat?

## Authentication Example (2)



## Authorization

*Authorization is the process of giving someone permission to do or have something.*

## A System Administrator

- In multi-user systems, **a system administrator** defines for the system
  - **which users are allowed** access to the system
  - **privileges of use** (access to which file directories, hours of access, amount of allocated storage space…)

# Portal Security Service

Advanced Security

## Portal Security Service

- Backbase CXP Security Service is **based** on Spring Security
  - Well-established, stable and extensible Java Security Framework (mature and active)
  - Spring security provides connectors to various authentication providers (LDAP, OpenId, CAS…)
  - Enables the implementation of custom connectors to any (non-standard) authentication service
- Responsible for all necessary security services within the Portal Server
- Gives access based on **user credentials**, **roles** and **security profiles**

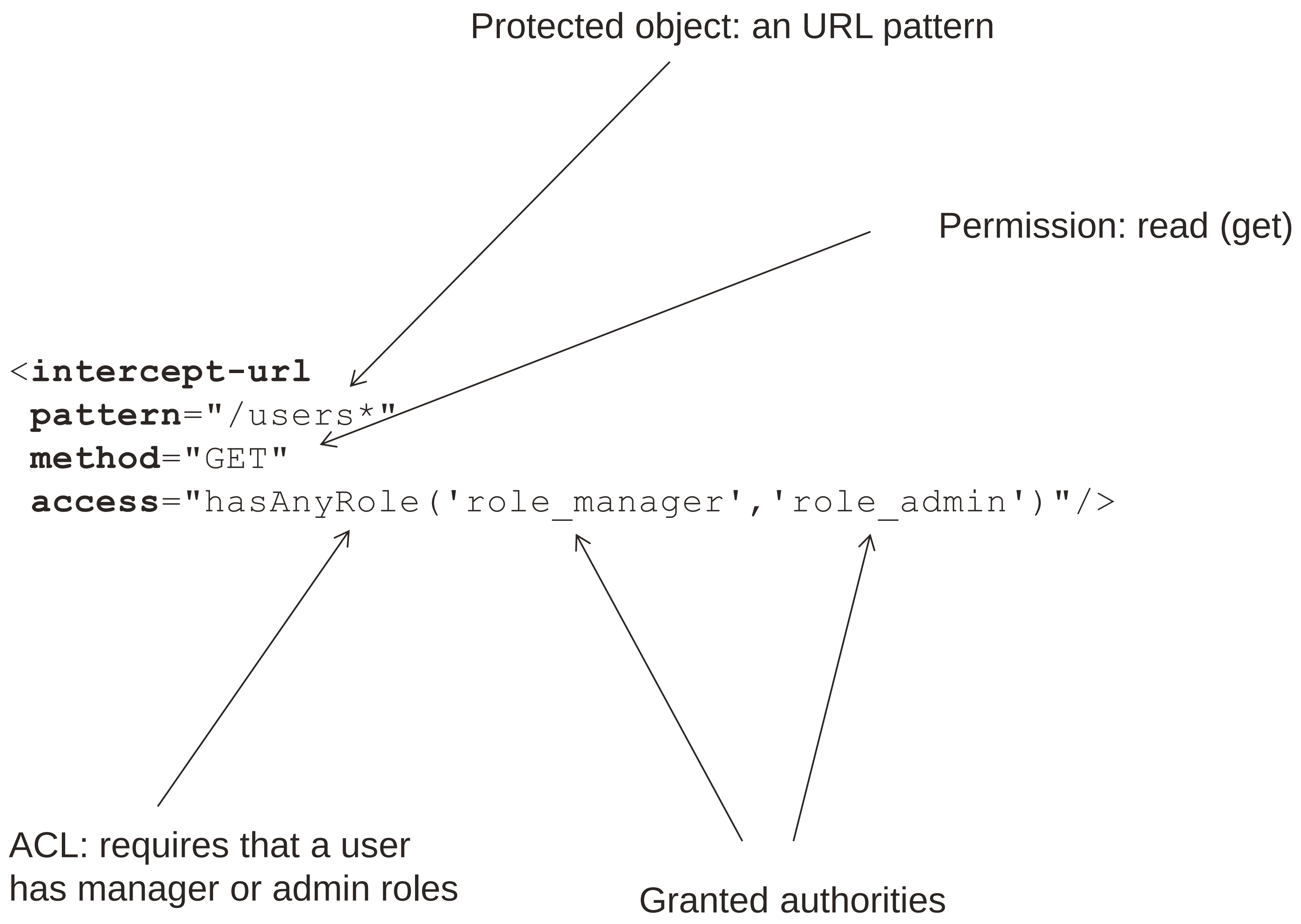
## Authentication and Portal Server

- Portal Server employs Spring authentication
  - Spring login form mechanism
    - CXP Manager uses custom login form
    - Default for new portals
  - JDBC Spring Authentication Connector
  - User Database in the Backbase CXP Data Store
  - mvn clean (with blank archetype and Jetty server)
    - Create a new empty database, with new tables, and adds “admin” user to the database

## Granted Authorities, ACLs, Permissions

- When users log in (authenticate), they are **granted some authorities**
  - Authority is an abstract concept which define user roles (e.g. administrator, collaborator, manager, user)
  - GrantedAuthority in Spring Security are used
- Access to objects are defined through **Access Control Lists (ACLs)**
  - ACLs are normally not defined for individual users, but for user roles (authorities)
  - More flexible (user can belong to multiple groups) and easier to maintain
- An ACL defines a list of authorities and their **permissions** (administration, create, read, write, delete...)

# Example: Spring URL Security



Protected object: an URL pattern

Permission: read (get)

```
<intercept-url
 pattern="/users*"
 method="GET"
 access="hasAnyRole('role_manager','role_admin')"/>
```

ACL: requires that a user
has manager or admin roles

Granted authorities

## Access to Backbase CXP Model Objects

- Protected objects: **URLs** and **Portal Items**
- Granted Authorities = **Portal User Groups + Implicit Roles**
- Access Permissions = **Portal Security Profiles**
  - Security profile is a group of permissions
  - Admin, Creator, Collaborator, Contributor, Consumer
- Access Control Lists (ACLs) = **Portal Items Permissions**
  - Defined for portal items, item properties and content objects
  - A list of pairs: user group + security profile

## Protected Objects

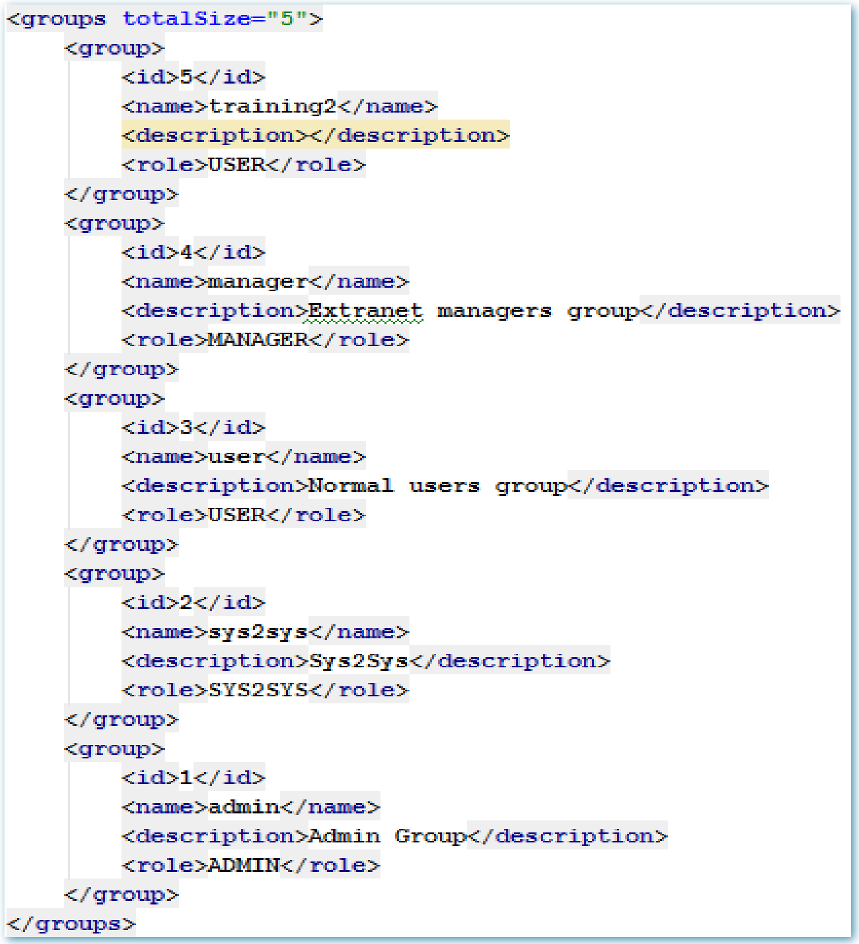
- **Items**
  - Portal, Page, Container, Widget, Link, Template
  - Managed through Portal Manager or Portal APIs
- **Portal REST service URLs**
  - Access to application URLs
    (e.g. Widgets, Services, REST API etc.)
  - Standard Spring Security URL Access Control

    -
      ```
      <intercept-url
       pattern="/users*"
       method="GET"
       access="hasAnyRole('role_manager','role_admin')"/>
      ```

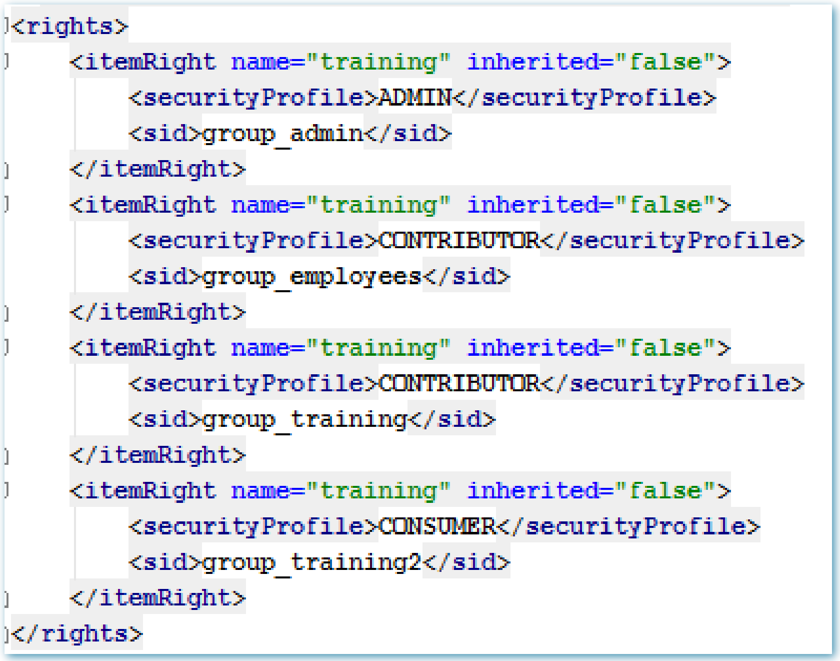
## Groups, Roles, Rights

- Portal users belong to one or more groups
- Each group has exactly one of the predefined roles: ADMIN, MANAGER, USER (default if group created in CXP Manager), SYS2SYS, ANONYMOUS
- When logged in, each user gets two granted authorities for each group it belongs to:
  - GROUP_*GROUPNAME* and ROLE_*ROLENAME*
- You then define rights (permissions) in terms of user granted authorities

## Groups, Roles, Rights - example



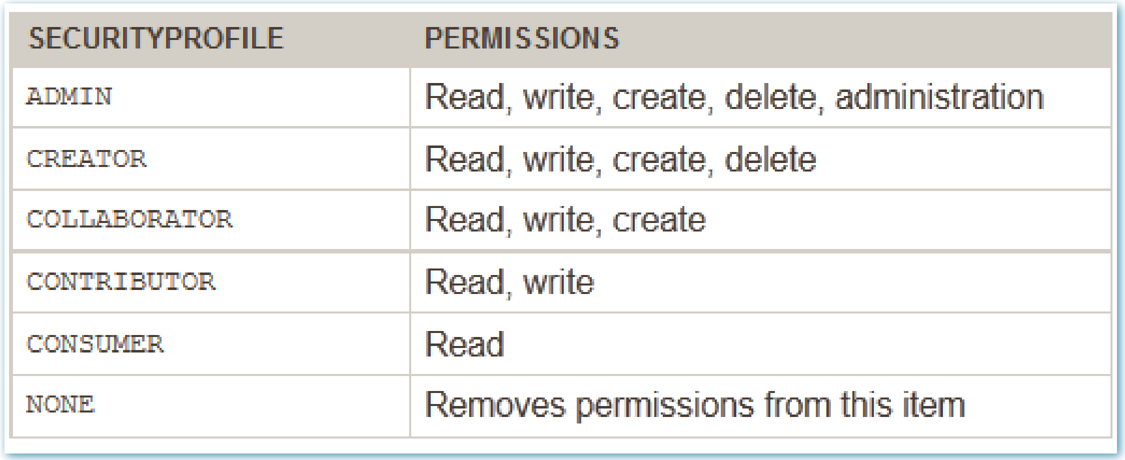
```
<groups totalSize="5">
    <group>
        <id>5</id>
        <name>training2</name>
        <description></description>
        <role>USER</role>
    </group>
    <group>
        <id>4</id>
        <name>manager</name>
        <description>Extranet managers group</description>
        <role>MANAGER</role>
    </group>
    <group>
        <id>3</id>
        <name>user</name>
        <description>Normal users group</description>
        <role>USER</role>
    </group>
    <group>
        <id>2</id>
        <name>sys2sys</name>
        <description>Sys2Sys</description>
        <role>SYS2SYS</role>
    </group>
    <group>
        <id>1</id>
        <name>admin</name>
        <description>Admin Group</description>
        <role>ADMIN</role>
    </group>
</groups>
```



```
<rights>
    <itemRight name="training" inherited="false">
        <securityProfile>ADMIN</securityProfile>
        <sid>group_admin</sid>
    </itemRight>
    <itemRight name="training" inherited="false">
        <securityProfile>CONTRIBUTOR</securityProfile>
        <sid>group_employees</sid>
    </itemRight>
    <itemRight name="training" inherited="false">
        <securityProfile>CONTRIBUTOR</securityProfile>
        <sid>group_training</sid>
    </itemRight>
    <itemRight name="training" inherited="false">
        <securityProfile>CONSUMER</securityProfile>
        <sid>group_training2</sid>
    </itemRight>
</rights>
```

# Permissions: Security profiles

- A group of permissions
- A combination of Spring base permissions: administration, create, read, write, delete

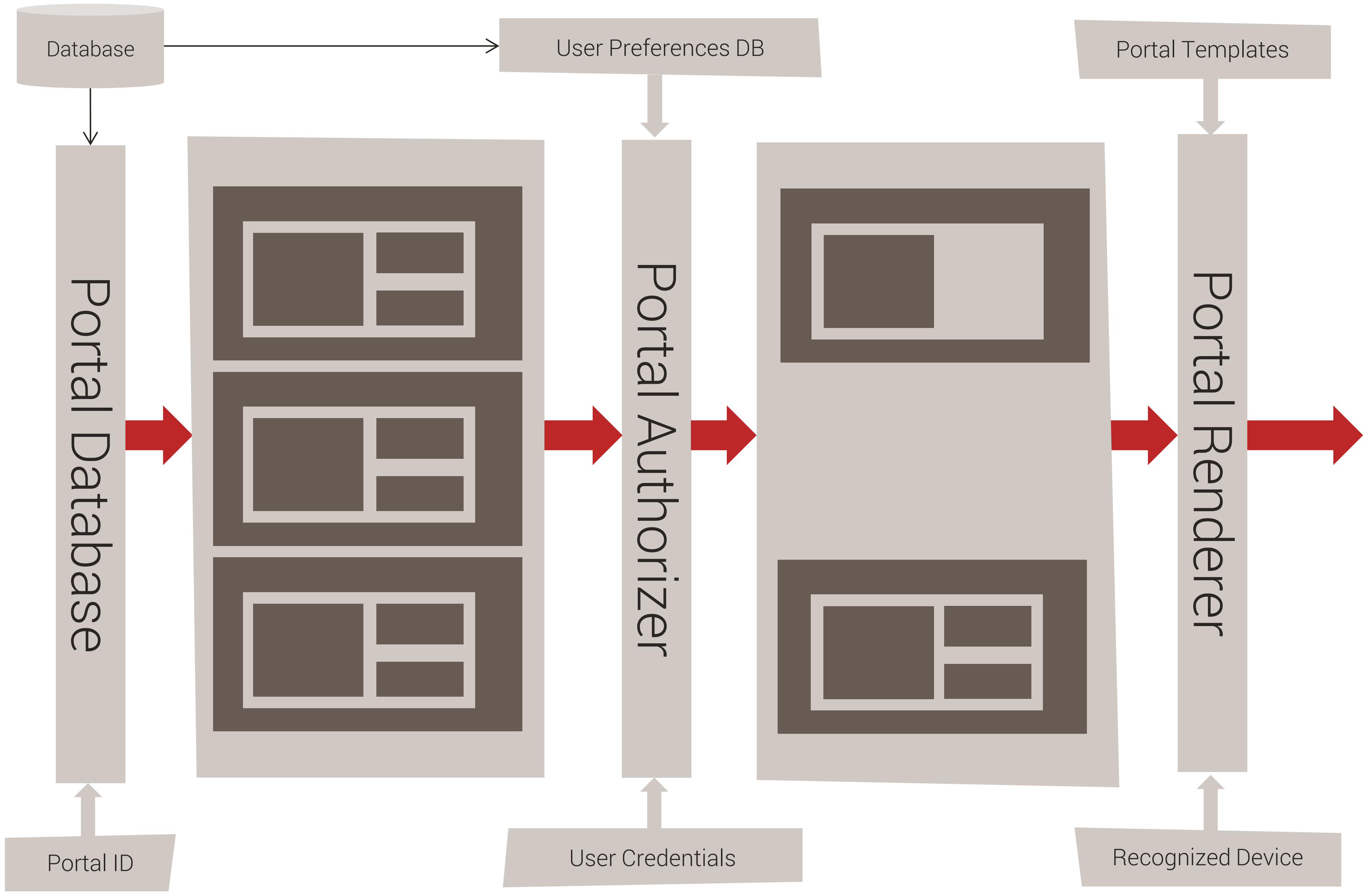


| SECURITYPROFILE | PERMISSIONS | **In CXP Manager** |
|---|---|---|
| ADMIN | Read, write, create, delete, administration | Can Administer |
| CREATOR | Read, write, create, delete | Can Edit, Can Personalize |
| COLLABORATOR | Read, write, create | - |
| CONTRIBUTOR | Read, write | - |
| CONSUMER | Read | Can View |
| NONE | Removes permissions from this item | No Access |

# Aggregated Security Profile

- A user can belong to one or more groups
  - Multiple groups possible to define through Portal APIs but not through CXP Manager
- When aggregating two groups, the more powerful groups is taken
- User { admin, user } = admin
- User { user } = user

# Portal Authorization



# Item Security (Authorization)

Advanced Security

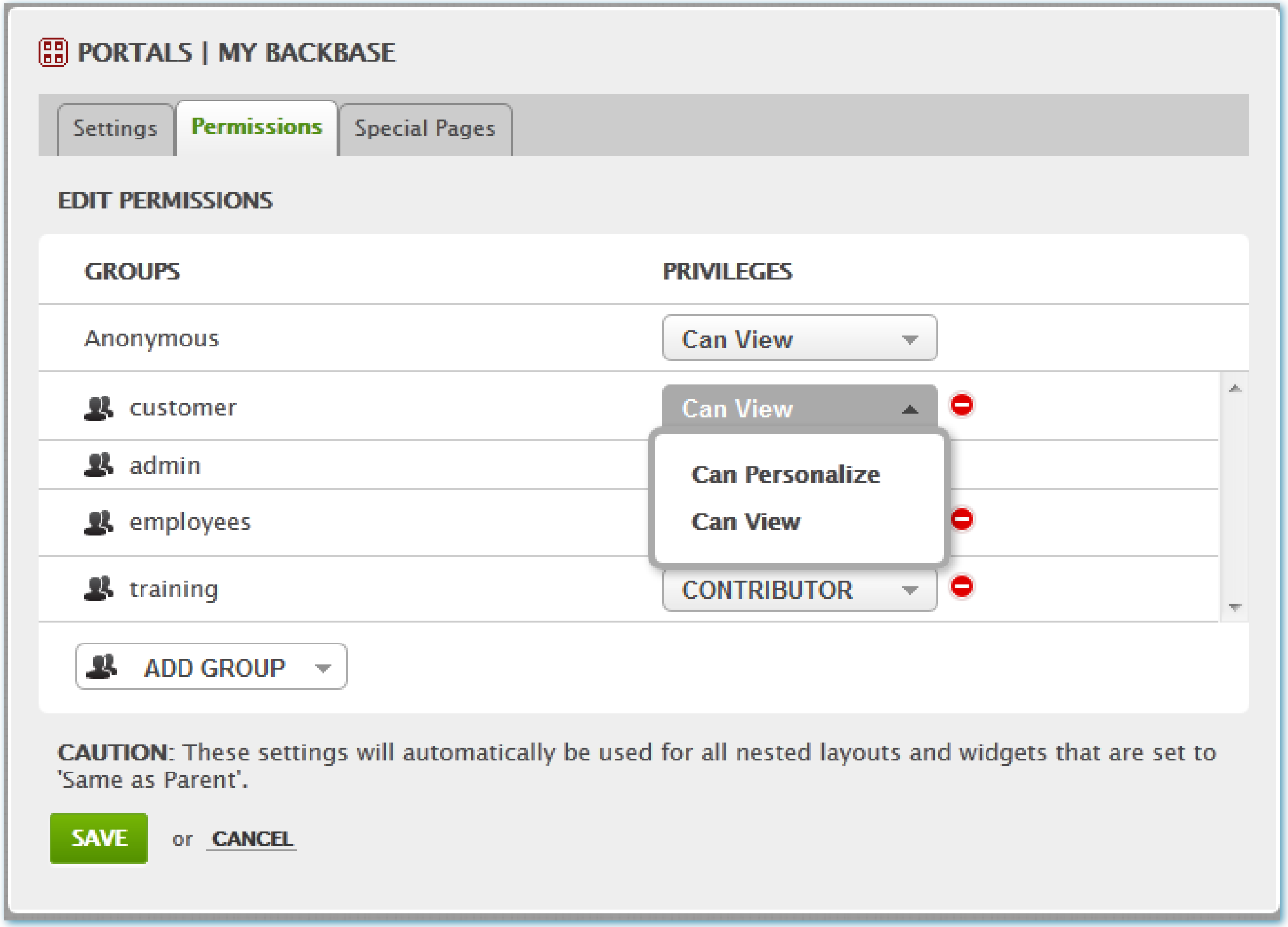
## Item Security

- Items can be secured. The following are regarded as items:
  - Portals
  - Pages
  - Containers
  - Widgets
  - Links
  - Templates
- Security inheritance
  - Items inherit the security of their parent item

## Setting Item Security

- Via CXP Manager
- Via REST API
- Via JAVA  API
- Via XML import scripts

- Permissions tab for each portal item



PORTALS | MY BACKBASE

Settings | Permissions | Special Pages

EDIT PERMISSIONS

| GROUPS | PRIVILEGES |
|---|---|
| Anonymous | Can View |
| customer | Can View |
| admin | |
| employees | |
| training | CONTRIBUTOR |

CAUTION: These settings will automatically be used for all nested layouts and widgets that are set to 'Same as Parent'.

BACKBASE

## Securing Items Via Rest API 1/2

- For most of the portal items, you can set the access permissions through REST API using appropriate URLs:
  - `/portals/[portal_name]/`**`rights`**
  - `/portals/[portal_name]/pages/[page_name]/`**`rights`**
  - `/portals/[portal_name]/containers/[container_name]/`**`rights`**
  - `/portals/[portal_name]/widgets/[widget_name]/`**`rights`**
  - `/portals/[portal_name]/links/[link_name]/`**`rights`**
  - `/templates/[template_name]/`**`rights`**

## Securing Items Via REST API 2/2

- XML format for defining rights:

```
<rights>
    <itemRight>
    ...
    </itemRight>
    <propertyRight>
    ...
    </propertyRight>
</rights>
```

```
<itemRight name="index" inherited="false">
    <securityProfile>ADMIN</securityProfile>
    <sid>group_admin</sid>
</itemRight>
```

```
<propertyRight name="index" inherited="false">
    <securityProfile>ADMIN</securityProfile>
    <sid>group_admin</sid>
</propertyRight>
```

## Securing Items Via Java API

- In Java, you get managed items rights through the `ItemBusinessService<T extends Item>` interface:

  -
    ```
    List<Right<? extends Sid>>
    getRightsForItem(String portalName, String itemName)
    ```

  -
    ```
    void
    updateRightsForItem(String portalName, String itemName,
                                     List<Right<? extends Sid>> rights)
    ```

## Securing Items Via XML Import Scripts

- In `importPortal.xml` you can include XML definition of rights

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<backbase-portal-import>
 <portal-import>
   ...
     <rights-file>rights-anonymous.xml</rights-file>
   ...
     <pages>
       ...
       <rights-file>rights-restricted.xml</rights-file>
   </pages>
   ...
</portal-import>
</backbase-portal-import>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<rights>
    <propertyRight name="ItemRef">
            <sid>role_anonymous</sid>
        <securityProfile>CONTRIBUTOR</securityProfile>
    </propertyRight>
    ...
</rights>
```

## Exercise - Authorization

- Start portal and login to CXP Manager
- Create new pages
- Secure a page via CXP Manager
- Secure a page via the REST API

BACKBASE

## Authentication Providers

Advanced Security

## Default Authentication Providers

- Portal Authentication Provider  (Set up as blank archetype dependency. Remove for production!)

```
<!-- default authentication plugin -->
<dependency>
    <groupId>com.backbase.portal.foundation</groupId>
    <artifactId>security-portalserver</artifactId>
    <version>${portal.server.version}</version>
</dependency>
```

- JDBC Authentication
- LDAP
- JAAS
- CAS
- Pre-Authentication (e.g. application container)
- NOTE: You need to override/implement the mapping!

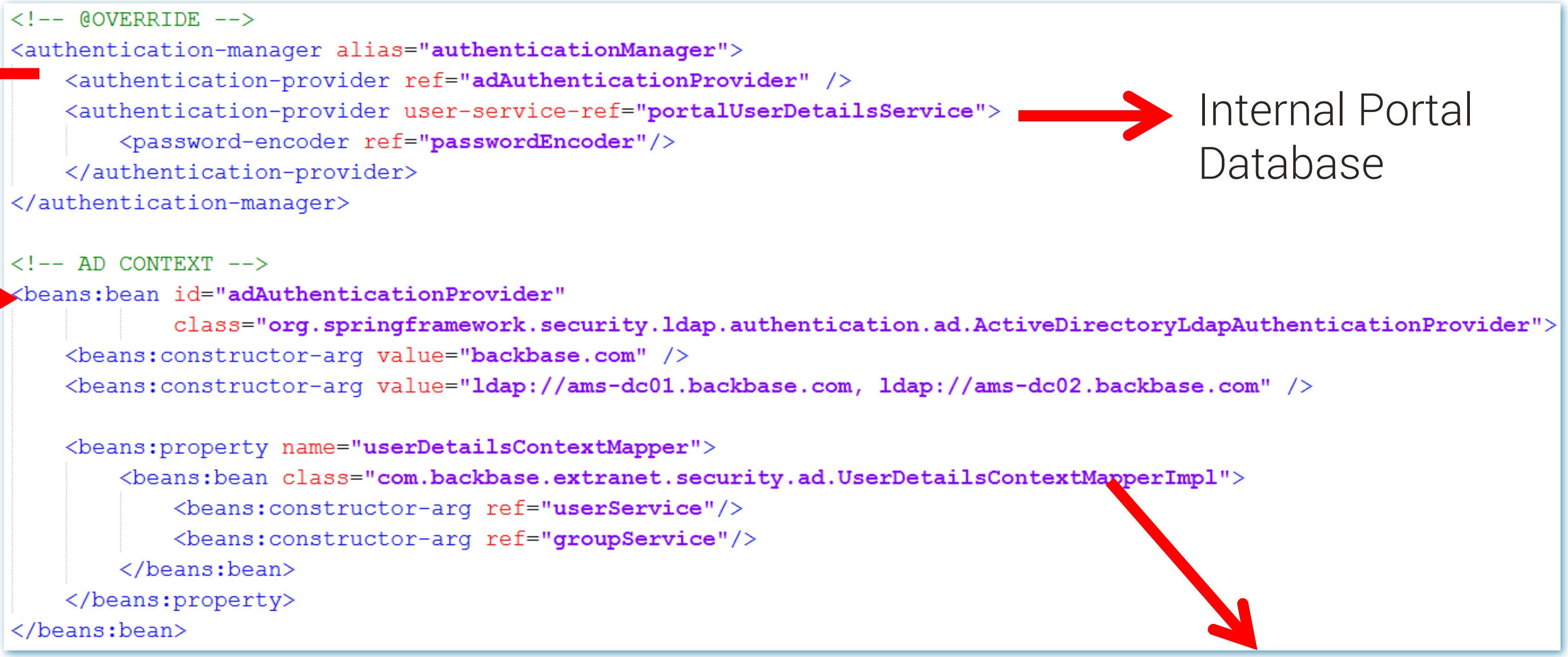
## Authentication Manager

```
<authentication-manager>
    <authentication-provider>
      <user-service>
        <user name="jimi" password="jimispassword"
            authorities="ROLE_USER, ROLE_ADMIN" />
        <user name="bob" password="bobspassword"
            authorities="ROLE_USER" />
      </user-service>
    </authentication-provider>
</authentication-manager>
```

In-memory authentication provider

## Configuring backbase-portal-business-security.xml

- Example with two authentication providers



```xml
<!-- @OVERRIDE -->
<authentication-manager alias="authenticationManager">
    <authentication-provider ref="adAuthenticationProvider" />
    <authentication-provider user-service-ref="portalUserDetailsService">
        <password-encoder ref="passwordEncoder"/>
    </authentication-provider>
</authentication-manager>

<!-- AD CONTEXT -->
<beans:bean id="adAuthenticationProvider"
            class="org.springframework.security.ldap.authentication.ad.ActiveDirectoryLdapAuthenticationProvider">
    <beans:constructor-arg value="backbase.com" />
    <beans:constructor-arg value="ldap://ams-dc01.backbase.com, ldap://ams-dc02.backbase.com" />

    <beans:property name="userDetailsContextMapper">
        <beans:bean class="com.backbase.extranet.security.ad.UserDetailsContextMapperImpl">
            <beans:constructor-arg ref="userService"/>
            <beans:constructor-arg ref="groupService"/>
        </beans:bean>
    </beans:property>
</beans:bean>
```

Internal Portal Database

Our implementation class

## Custom UserDetailsContextMapper (ActiveDir.)

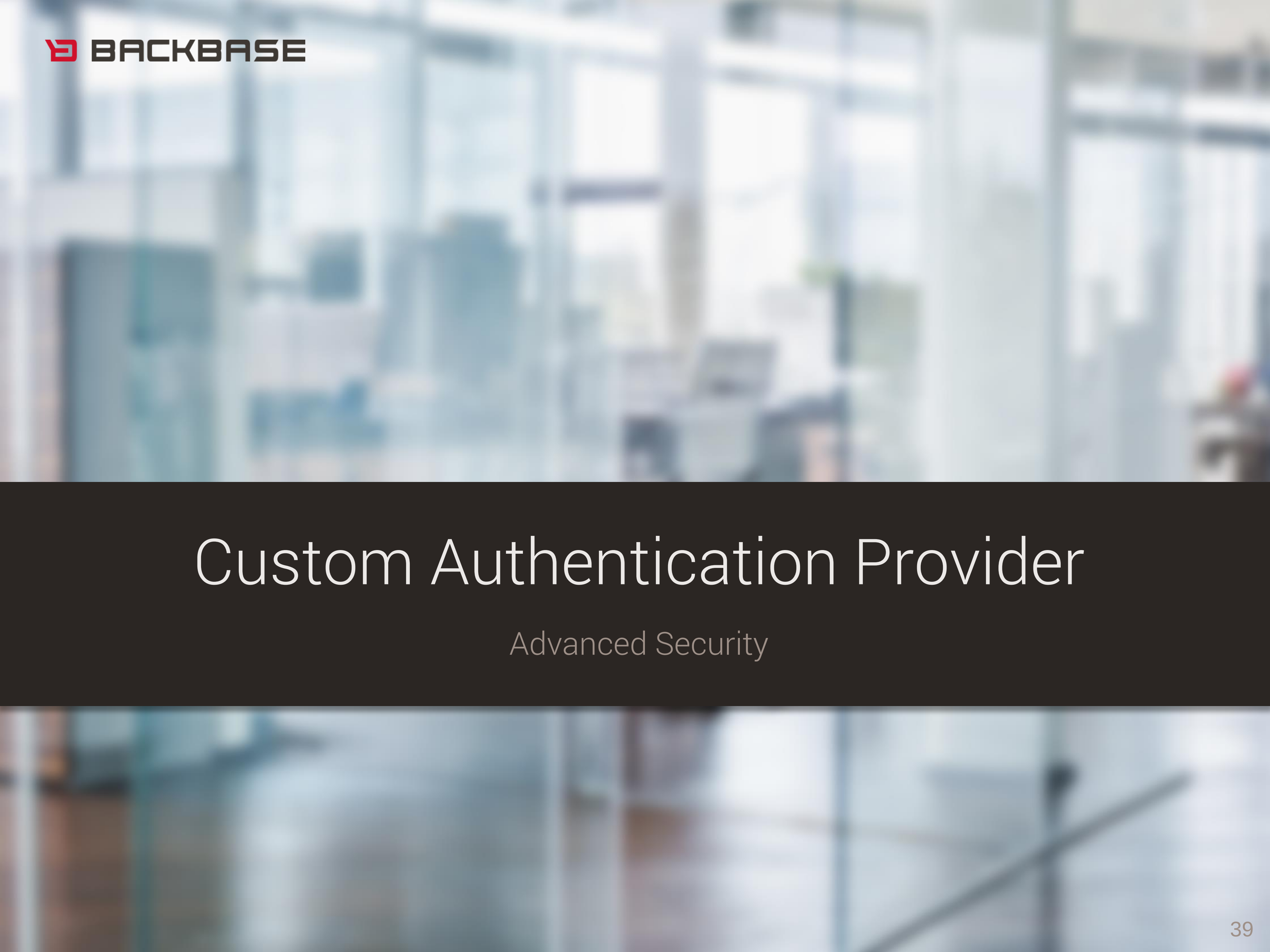
```
import com.backbase.portal.foundation.domain.model.Group;
import com.backbase.portal.foundation.domain.model.Role;
import com.backbase.portal.foundation.domain.model.User;

public class UserDetailsContextMapperImpl implements UserDetailsContextMapper {
    private UserBusinessService users;
    private GroupBusinessService groups;

    public UserDetailsContextMapperImpl(UserBusinessService users, GroupBusinessService groups)  {
         this.users = users;    this.groups = groups;
    }
    ...
    @Override  public UserDetails
     mapUserFromContext(DirContextOperations ctx, String userName, Collection authorities ) {
          ...
          User user = users.getUser(userName);
          user.getGroups().add(groups.getGroup( “EMPLOYEES” ));
          ...
          return user;
    }
}
```

## Authentication Mapping - Challenges

- Be aware of the complex mapping of users and groups
- Authentication provider should map authenticated users to portal users
- You get personalization conflicts if user ID’s are not matching

## Integrate and configure an LDAP authentication provider

How To Guides » Security » **Integration and Configuring Spring Security Providers – LDAP Integration**



# Custom Authentication Provider

Advanced Security

## Authentication Provider Interface

- You can write your own authentication provider by implementing the authentication provider interface:
  - org.springframework.security.authentication.AuthenticationProvider
- Two methods:
  - authenticate(Authentication authentication)
    - Performs authentication
    - In portal you may need to create portal a new user if necessary
  - supports(Class<? extends Object> authentication)
    - True if provider supports the indicated Authentication object
    - Default token is UserNamePasswordToken

## Exercise – Custom Authentication Provider

- Create a custom authentication provider
- Implement the AuthenticationProvider interface
- Use UsernamePasswordAuthenticationToken
- Keep users locally in Class.
- Automatically create users in portal database
- Assign default user groups

# Multitenancy

Advanced Security

BACKBASE

## Multitenancy

*Multi-tenancy is an architecture in which a single instance of a software application serves multiple customers.*

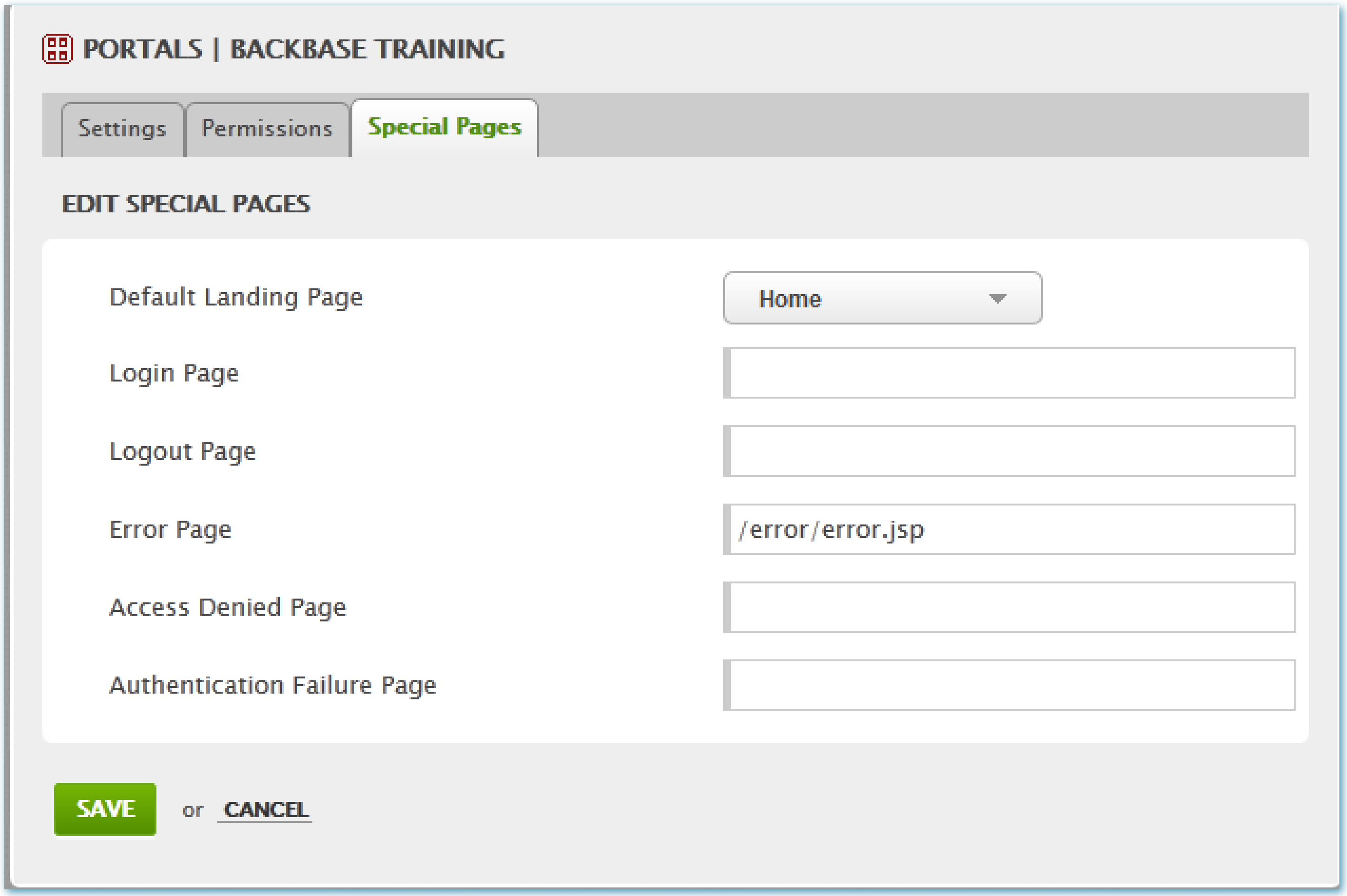
- A single portal server can hold multiple portals.
- A “portal” equals “tenant”
- Each portal can have different authentication mechanisms, different authorizations, different login/logout pages.
- Problem: How to configure a secure multi-tenancy architecture in one single web application configuration (web.xml, spring-security-context.xml)

## Multitenancy: Configuration

- Spring Security > 3.1.x
- Multiple <http> elements possible
- Each <http> element can have a “pattern”
- Override backbase-portal-presentation-security.xml

```
<http pattern="/css/**" security="none"/>
<http pattern="/login.jsp*" security="none"/>

<http auto-config='true'>
  <intercept-url pattern="/**" access="ROLE_USER" />
   <form-login login-page='/login.jsp'/>
</http>
```

## Multitenancy: Portal Special Pages


PORTALS | BACKBASE TRAINING

EDIT SPECIAL PAGES

| | |
|---|---|
| Default Landing Page | Home |
| Login Page | |
| Logout Page | |
| Error Page | /error/error.jsp |
| Access Denied Page | |
| Authentication Failure Page | |

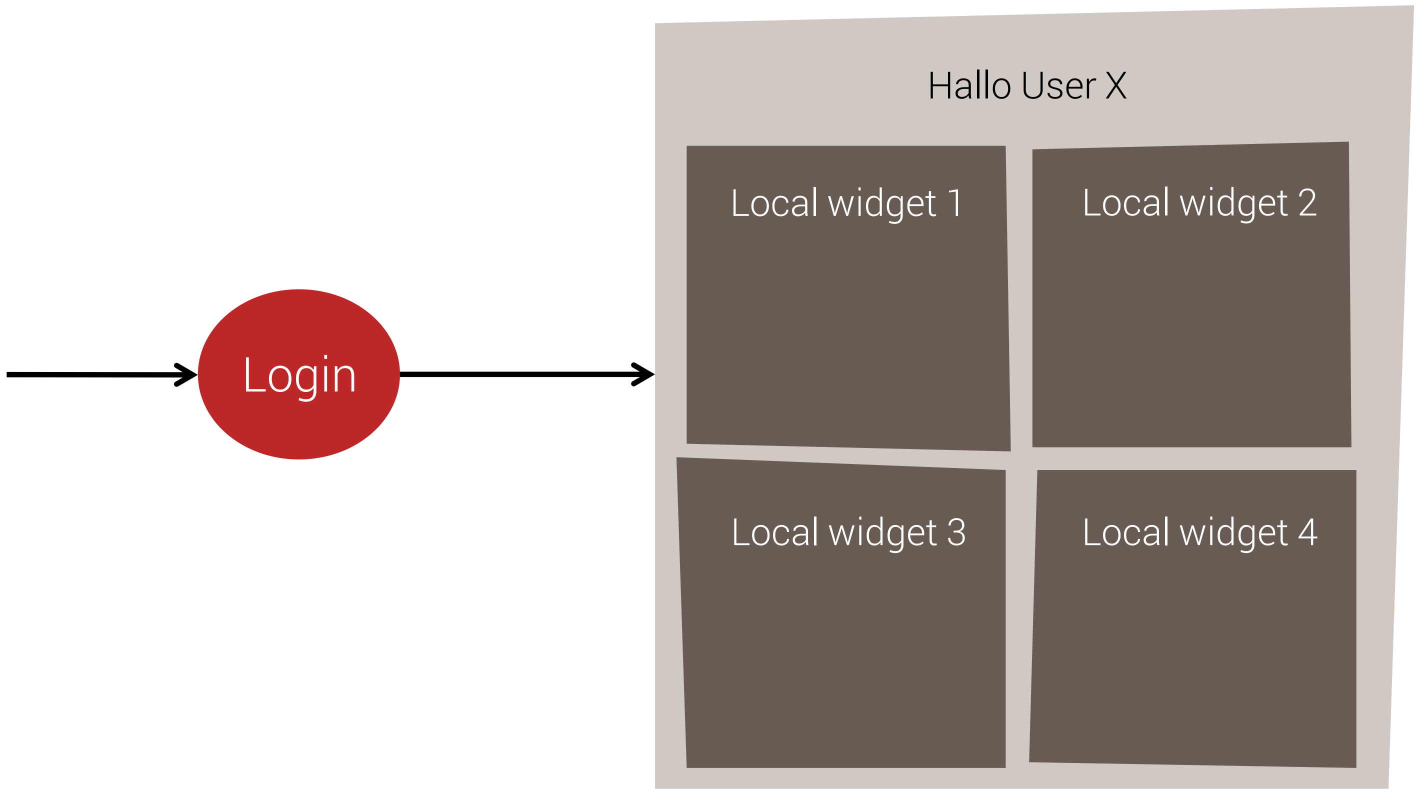
# Widget Security And Single Sign-On (SSO)

Advanced Security

## Widget Security

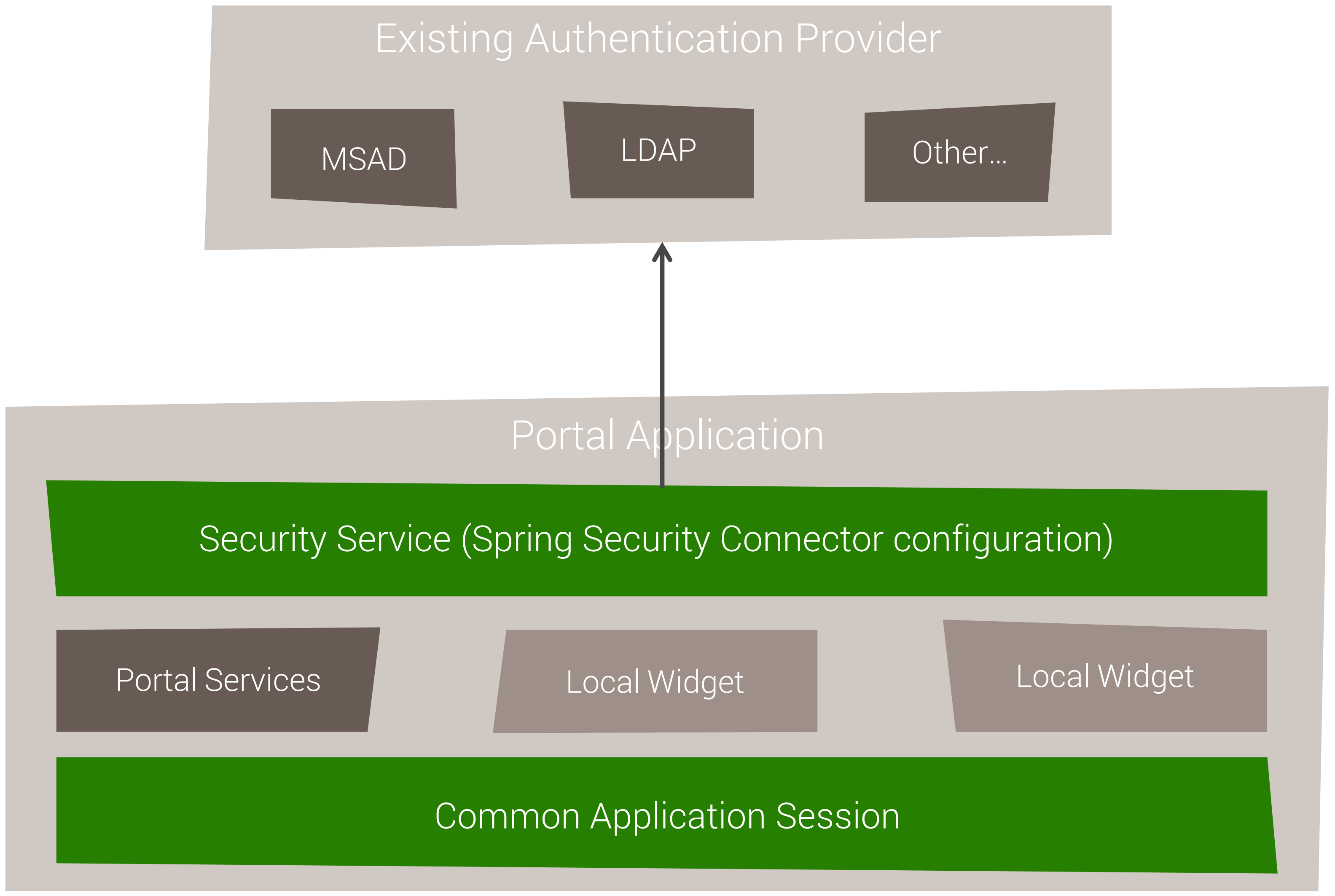
- Widgets are small applications with their own security.
  - **Local widget** - portal and widgets are within the same application context
  - **Remote widget** - portal and widgets are in separate application contexts (and could even be in different technologies)

# Local Widgets, Share the Same Context

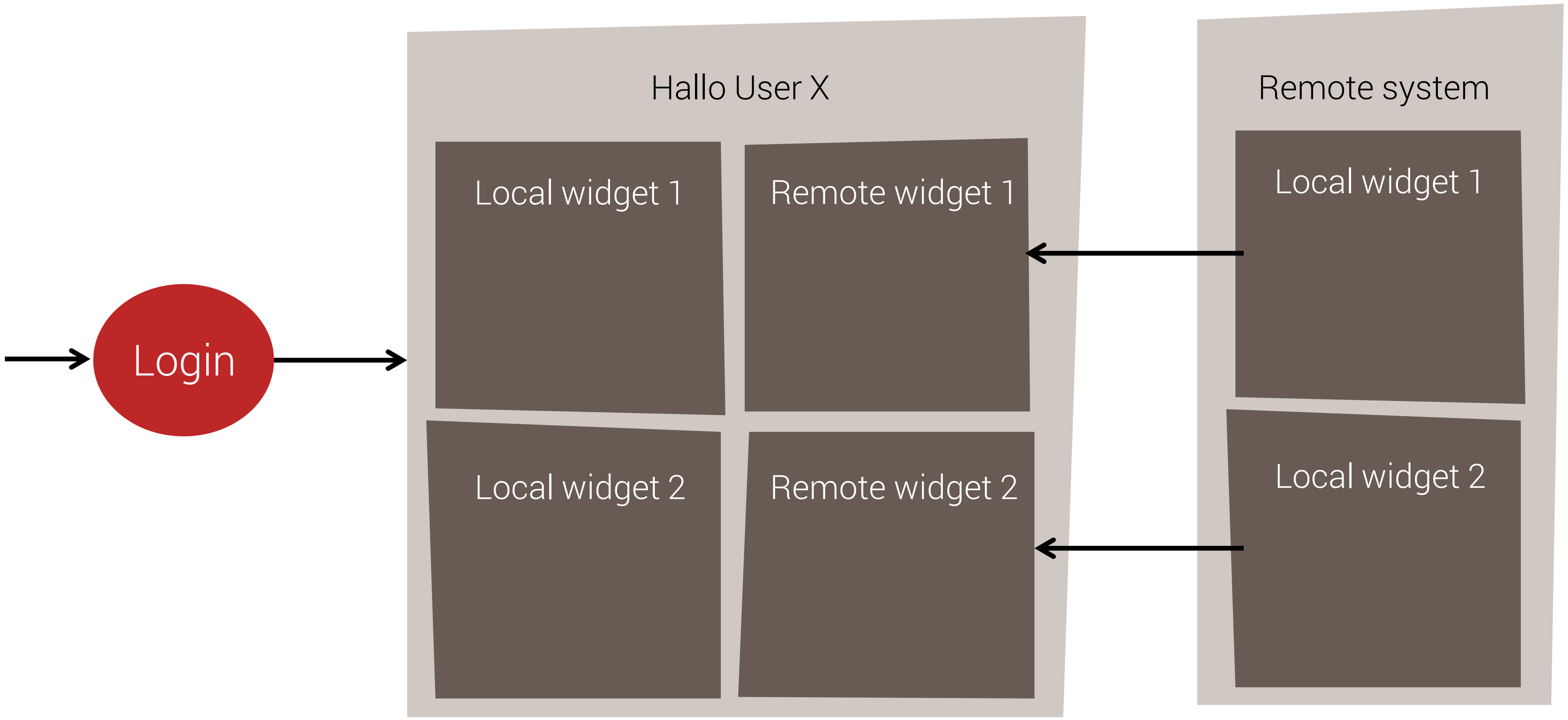


BACKBASE

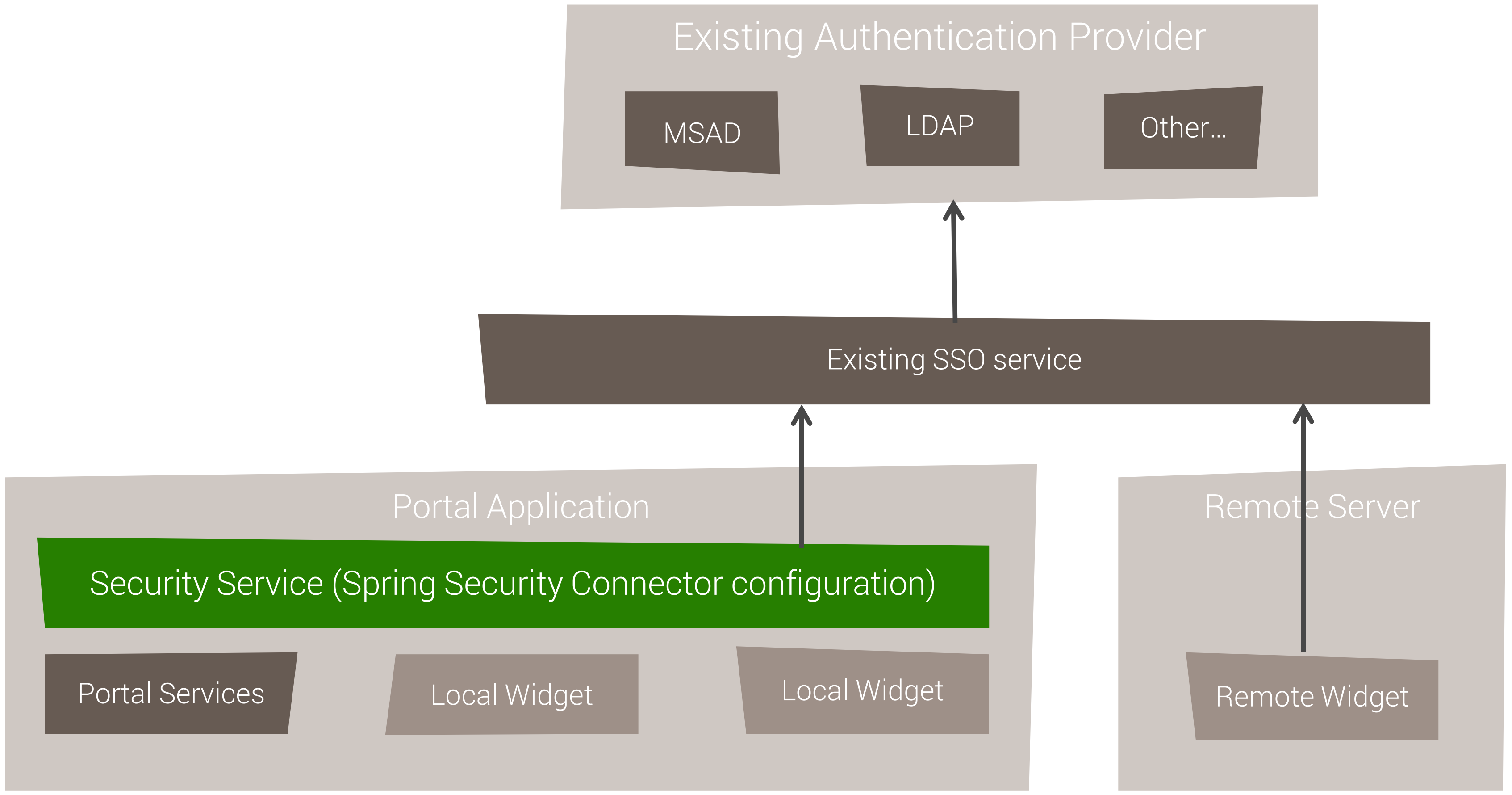
## Security Service: Local Widgets


Existing Authentication Provider
MSAD
LDAP
Other...
Portal Application
Security Service (Spring Security Connector configuration)
Portal Services
Local Widget
Local Widget
Common Application Session

# Remote Widgets, Shared Session



# Security Service: Remote Widgets



# Thank you!

www.backbase.com
sales-eu@backbase.com

**New York:** +1 646 478 7538
**Amsterdam:** +31 20 465 8888