



Liferay
Boot Camp 2022
10th Anniversary

Liferay Authentication

How to create a Token-based SSO system

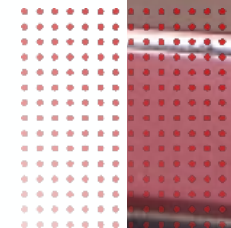


Antonio Musarra

Senior Software Architect

Index of topics

- ☒ **Introduction to the scenario**
- ☐ Securing Liferay
- ☐ Authentication Basics
- ☐ Token-based Single Sign On Authentication
- ☐ Implementation of the scenario
- ☐ Scenario in action (demo)



Introduction to the scenario

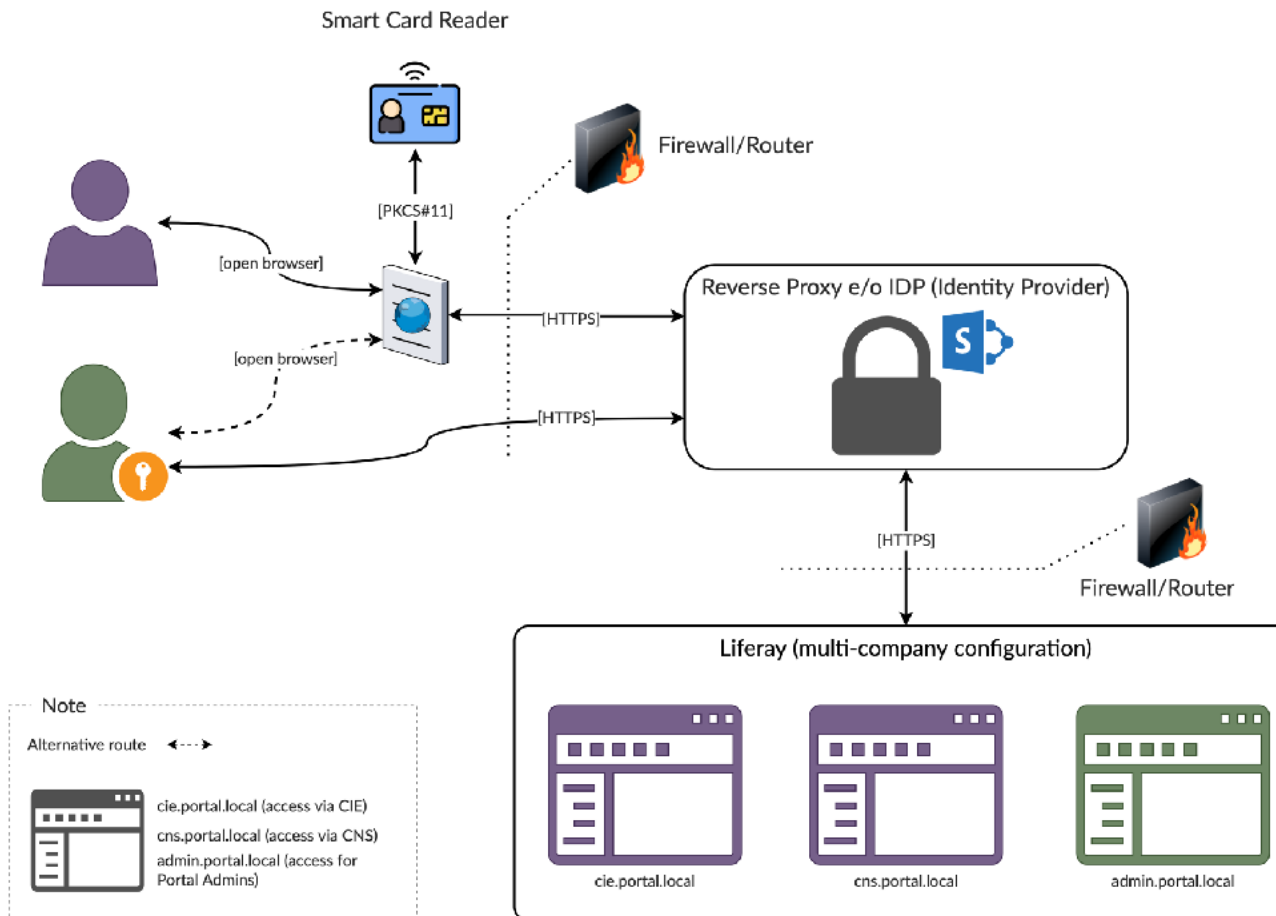
Requirements for an authentication system



- 1 We have two portals configured as Virtual Instances of a Liferay installation
- 2 The two portals respond to two different FQDNs (example: portal-1.local, portal-2.local)
- 3 Access to the two portals must be protected through the use of Smart Cards
- 4 Access to the two portals must be diversified according to the type of Smart Card
- 5 The details on the access data (example: personal data) must be retrieved from an external system
- 6 The external system from which to extract the data associated with the Smart Card can be of different types (database, web service, memory, etc)

Portal access scenario via Smart Card

Macro diagram illustrating access to two Liferay portals configured on two different instances (companies) using the Smart Card as an authentication tool. In this scenario, two types of Smart Cards are used: TS-CNS (Health Card, National Service Card) and CIE (Electronic Identity Card).



- ▶ In this scenario, users will be able to access the two portals **cns.portal.local** and **cie.portal.local** using respectively the Smart Cards, **TS-CNS** and **CIE**.
- ▶ The admin user will be able to login without using a Smart Card via FQDN **admin.portal.local** (master instance). If necessary, he will be able to access via Smart Card (on **cns.portal.local** or **cie.portal.local**) and operate as administrator of the specific instance if enabled.
- ▶ The user's workstation must be equipped with a **Smart Card reader**.
- ▶ The user's browser must be configured to be able to use the Smart Card reader.
- ▶ The user must have the **PIN** of the respective Smart Card. For the TS-CNS, before it can be used, it must be activated by the relevant ASL.
- ▶ The Reverse Proxy and/or IDP is the component responsible for the authentication process.
- ▶ Of this scenario, our area of expertise is the one that concerns the Liferay hemisphere only. We will see how to develop the components necessary to allow users previously identified by the Reverse Proxy and/or IDP architectural component to access the portals.

This scenario involves the use of the Smart Card Reader which must conform to the PC/SC, ISO 7816-3 and CCID standard. The readers that I have personally used successfully in this scenario and in other scenarios are produced by Bit4id and in particular.



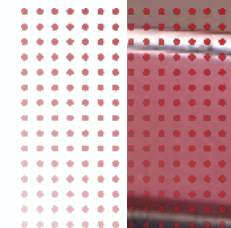
[miniLector CIE](#)



[miniLector EVO](#)

Index of topics

- ☐ Introduction to the scenario
- ☒ **Securing Liferay**
- ☐ Authentication Basics
- ☐ Token-based Single Sign On Authentication
- ☐ Implementation of the scenario



Securing Liferay

Brief introduction on the security mechanisms offered by Liferay



- 1** Liferay is built with security in mind. This includes mitigation of common security vulnerabilities and exploits like those described by the [OWASP Top 10](#) and the [CWE/SANS Top 25](#).
- 2** Liferay [takes care about security](#), in both the community (CE) and the enterprise (DXP) editions, keeping always up-to-date the [known vulnerabilities list](#), and also having their own [security statement](#).
- 3** The security aspects, includes configuring how users authenticate to your Liferay installation, authorizing users with permissions, configuring secure access to Liferay Web Services, and fine-tuning security features on an as-needed basis.

1 The three main aspects of Liferay's security are

- **Authentication**
- **Permission**
- **Securing Web Service**
- **Fine-Tuning Security.**

► Authentication

Liferay authentication is flexible. By default, users log into Liferay by using the ***Sign In*** widget, which uses the database to authenticate the user. By default, guests can use the Sign In widget to create accounts with default permissions. Nearly every element of the default authentication experience can be changed by an administrator. For example,

- You can configure [Multi-Factor authentication](#) (MFA).
- You can [use an SSO to manage](#) authentication.
- Liferay can also be [integrated with LDAP](#) to validate users instead of using the portal database.
- Guest account creation can be [turned off](#).

To learn more, see [Authentication Basics](#).

► Permission

Liferay has a robust **role-based access control (RBAC) system**. Users can be assigned to *Sites, Teams, User Groups, or Organizations*.

Custom Roles can be created, permissions can be assigned to these Roles, and those Roles can be assigned to Users.

Roles are scoped to apply only in a specific context, such as a Site, Organization, or globally.

See [Roles and Permissions](#) for more information.

► Securing Web Services

Liferay Web Services have a multi-layered and configurable approach to security and authorization:

- [Service Access Policies](#) (SAP) control access to remote APIs.
- [Authentication Verifiers](#) verify provided credentials.
- [Cross-Origin Resource Sharing](#) configuration can enable retrieving resources from trusted sources only.

See [Introduction to Securing Web Services](#) to learn more.

► Fine-Tuning Security

There are many ways to fine-tune or disable additional security features:

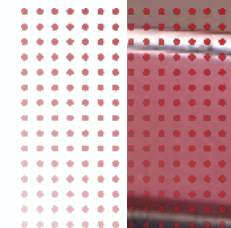
- Configure Liferay Portal's HTTPS [web server](#) address.
- Configure the list of allowed servers to which users can be [redirected](#).
- Configure the list of [portlets](#) that can be accessed from any page.
- Configure the file types allowed to be uploaded and downloaded.

These features can be configured using [portal properties](#).

Liferay Portal's philosophy **is secure by default**. Please exercise significant caution when modifying security-specific defaults or white-lists. Such actions may lead to [security misconfiguration and an insecure deployment](#).

Index of topics

- ✓ Introduction to the scenario
- ✓ Securing Liferay
- ✓ Authentication Basics**
- ✓ Token-based Single Sign On Authentication
- ✓ Implementation of the scenario
- ✓ Scenario in action (demo)



Authentication Basics

Which methods of authentication can be configured to users and/or applications



By default, Liferay uses the *Sign In* widget to authenticate users.

You can configure other methods of authenticating users and/or applications:

- [LDAP](#)
- [SAML](#)
- [Kerberos](#)
- [OpenID Connect](#)
- [Token-Based solutions](#)
- OAuth 2.0

Note

The Sign In widget appears on the default home page at `http[s]://[server-name:port]/web/guest/home`.

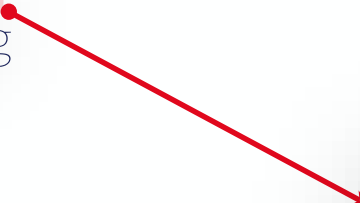
If the Sign In widget is unavailable on any page, it can be accessed directly via its URL: `http[s]://[server-name:port]/c/portal/login`.

This will be the authentication method used as the **basis for implementing the scenario** shown at the beginning of the presentation.

[Authentication Verifiers](#) can manage authentication for remote applications, and Authentication Pipelines define the ways users are validated by one or several systems.

Users can be configured to login using one of three authentication types:

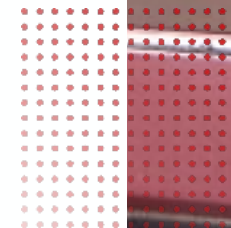
- **Email Address:** Determined by administrator or user at account creation. This is the default setting
- **User ID:** Determined by administrator or user at account creation. This is not the default setting
- **Screen Name:** Determined by administrator or user at account creation. This is not the default setting



This will be the authentication type used as the **basis for implementing the scenario** shown at the beginning of the presentation.

Index of topics

- ✓ Introduction to the scenario
- ✓ Securing Liferay
- ✓ Authentication Basics
- ✓ Token-based Single Sign On Authentication**
- ✓ Implementation of the scenario
- ✓ Scenario in action (demo)



Token-based Single Sign On Authentication


Brief introduction on the Token-based SSO offered OOTB by Liferay



Token-based Single Sign On Authentication (1/3)

Token-based SSO authentication was introduced in Liferay Portal 7.0 to standardize support for **Shibboleth**, **SiteMinder**, **Oracle OAM**, and any **SSO** product that works by propagating a token via one of the following mechanisms:

- HTTP request parameter
- HTTP request header
- HTTP cookie
- Session attribute



This will be the mechanisms type used as the **basis for implementing the scenario** shown at the beginning of the presentation.

Since these providers have a built-in web server module that reads and sets these parameters, headers, cookies, or attributes, you should use the Token SSO configuration.

Token-based SSO authentication mechanism is highly flexible and compatible with any SSO solution that provides it with a valid Liferay Portal user's screen name or email address. These include Shibboleth and SiteMinder.

Token-based Single Sign On Authentication (2/3)

The authentication token contains the User's screen name or email address, whichever has been configured to use for the particular company (portal instance). Liferay Portal supports three authentication methods:

- By email address
- By screen name
- By user ID

This will be the authentication method used to **implement the scenario** shown at the beginning of the presentation.

Token-based authentication only supports email address and screen name. **You must use a security mechanism external to Liferay Portal...**

...such as a fronting web server like Apache with a module or plugin for the authentication mechanism. Having a reverse proxy prevents malicious User impersonation that otherwise might be possible by sending HTTP requests directly to Liferay Portal's app server from the client's web browser.

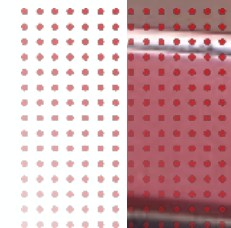
Token-based Single Sign On Authentication (3/3)

The configuration options for the Token-based SSO module are:

- **Enabled:** enable or disable token-based SSO authentication.
- **Import from LDAP:** check this box to import users automatically from LDAP if they don't exist.
- **User token name:** Set equal to the name of the token. This is retrieved from the specified location. (Example: `SM_USER`).
- **Token location:** Set this to the type of user token: HTTP request parameter, HTTP request header, HTTP cookie, Session attribute.
- **Authentication cookies:** Set this to the cookie names that must be removed after logout. (Example: `SMIDENTITY`, `SMSESSION`)
- **Logout redirect URL:** When user logout of Liferay Portal, the user is redirected to this URL.

Index of topics

- ✓ Introduction to the scenario
- ✓ Securing Liferay
- ✓ Authentication Basics
- ✓ Token-based Single Sign On Authentication
- ✓ **Implementation of the scenario**
- ✓ Scenario in action (demo)



Implementation of the scenario

How to implement a custom Token-based SSO module to authenticate users



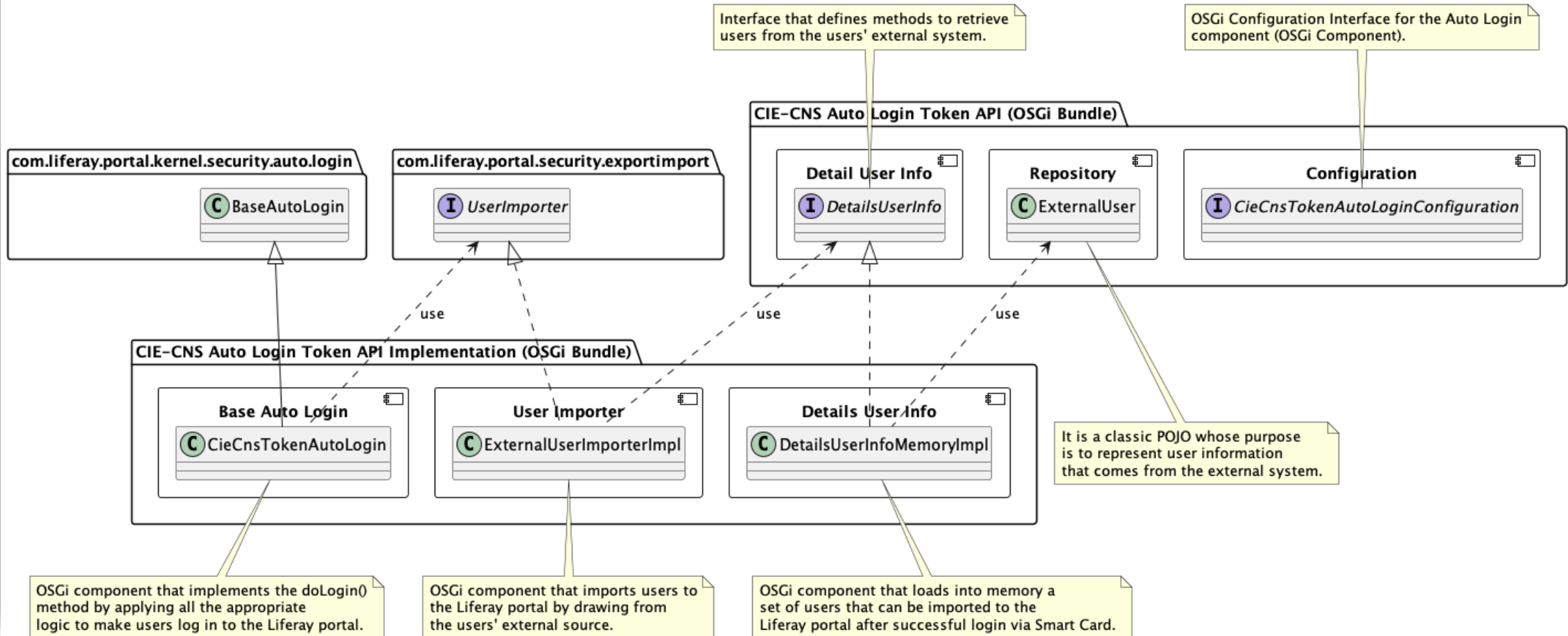
With reference to the scenario described at the beginning of the presentation, we will see which are the main components that must be developed in order to achieve the set objective.

For reasons of time and space we will not be able to see in detail all the components, but no worries, the complete project is available on the [cie-cns-auth-token-sso](#) repository published on GitHub. You will then have the opportunity to calmly view the project and try it out.

The main components that need to be implemented are:

- **Auto Login** (documentation available at <https://help.liferay.com/hc/en-us/articles/360018161711-Auto-Login>)
- **OSGi Configuration** (documentation available at <https://help.liferay.com/hc/en-us/articles/360018161391-Making-Your-Applications-Configurable>)
- **User Importer**


Diagram of the main components of the Auto Login module that is created.



smclab / cie-cns-auth-token-ssobrowseUnwatch5Fork0Star0

CodeIssuesPull requestsActionsProjectsSecurityInsightsSettings

master1 branch0 tagsGo to fileAdd fileCode


amusarra Added the UML docs
 c043ece · 22 hours ago · 13 commits

configs	Added scripts shell on the docker profile	3 days ago
docs/uml	Added the UML docs	22 hours ago
gradle/wrapper	Initial source import	8 days ago
modules/security/token-header-aut...	Added the roles name to the imported users	2 days ago
.gitignore	Added the .DS_Store to the .gitignore	8 days ago
Dockerfile.ext	Initial source import	8 days ago
GETTING_STARTED.markdown	Initial source import	8 days ago
LICENSE	Initial source import	8 days ago
build.gradle	Initial source import	8 days ago
copyright.txt	Initial source import	8 days ago
docker-compose.yml	Added the docker-compose.yml	2 days ago
gradle.properties	Initial source import	8 days ago
gradlew	Initial source import	8 days ago
gradlew.bat	Initial source import	8 days ago
platform.bndrun	Initial source import	8 days ago

About

Repository del Liferay BootCamp 2022 per la presentazione Liferay Authentication: How to create a Token-based SSO system

View license0 stars5 watching0 forks

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Languages

Java95.8%

Shell4.2%

Implementation of the scenario - Auto Login

(2/5)

While Liferay supports a wide variety of [authentication mechanisms](#), you may use a home-grown system or some other product to authenticate users. **To do so, you can write an Auto Login component to support your authentication system.**

Auto Login components can check if the request contains something (a cookie, an attribute) that can be associated with a user in any way. If the component can make that association, it can authenticate that user to Liferay.

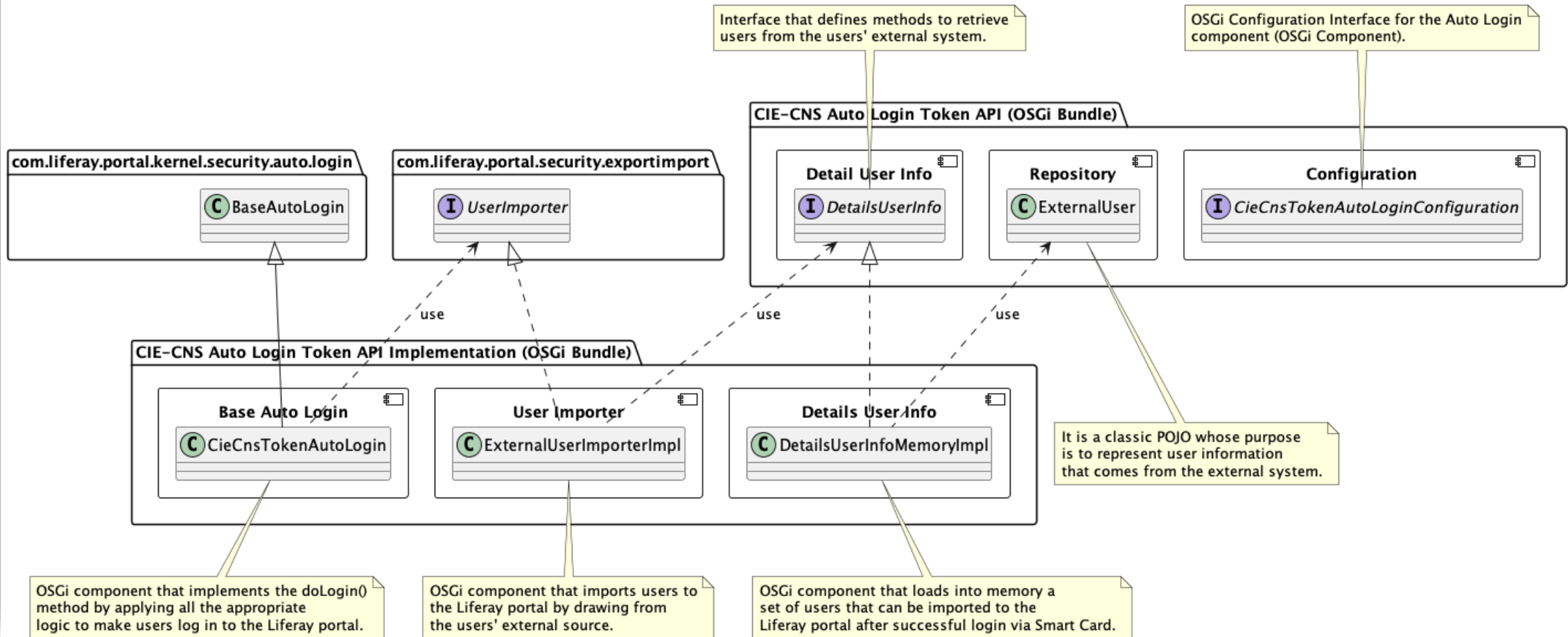
Implementing your own Auto Login consists of creating a [Declarative Services Component](#) that implements the `com.liferay.portal.kernel.security.auto.login.AutoLogin` interface.

In our case we will create the component by extending the base class that Liferay makes available to us and which offers a series of basic services (`com.liferay.portal.kernel.security.auto.login.BaseAutoLogin`).

Implementation of the scenario - Auto Login (2/5)

```
1 import com.liferay.portal.kernel.security.auto.login.AutoLogin;
2
3 import javax.servlet.http.HttpServletRequest;
4 import javax.servlet.http.HttpServletResponse;
5
6 import org.osgi.service.component.annotations.Component;
7
8 @Component(immediate = true)
9 public class MyAutoLogin implements AutoLogin {
10
11     public String[] handleException(
12         HttpServletRequest request, HttpServletResponse response,
13         Exception e)
14         throws AutoLoginException {
15
16         /* This method is no longer used in the interface and can be
17         left empty */
18
19     }
20
21     public String[] login(
22         HttpServletRequest request, HttpServletResponse response)
23         throws AutoLoginException {
24
25         /* Your Code Goes Here */
26
27     }
28
29 }
```

Diagram of the main components of the Auto Login module that is created.



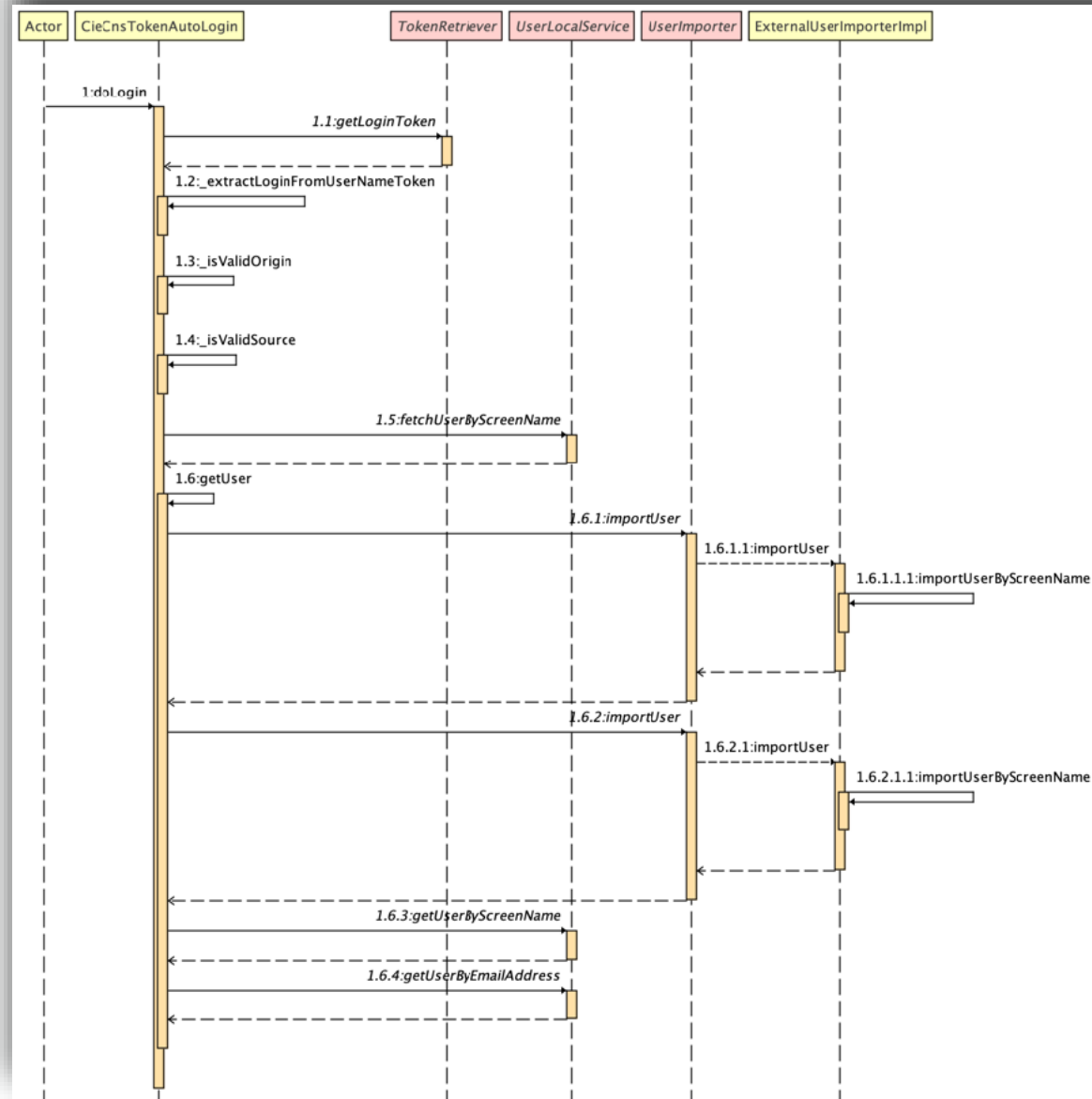
Implementation of the scenario - Auto Login

(3/5)

What are the main responsibilities of our Auto Login component which we will call *CieCnsTokenAutoLogin*?

1. Read the module configuration (OSGi).
2. Read the Username Token using the [TokenRetriever](#) service.
3. Verify that the user is present on the Liferay portal (instance/company).
4. If the user is present, allow access to the Liferay portal.
5. If the user is not present, the [User Importer](#) service must be used. The latter is responsible for importing the user from the external source, once imported (created) on Liferay he will be granted access.
6. If User Import is unable to import the user, access to the portal will be denied.

Implementation of the scenario - Auto Login (3/5)



Implementation of the scenario - OSGi Configuration (4/5)

The OSGi configuration component of the Auto Login module that we will call *CieCnsTokenAutoLoginConfiguration* will manage the following main configuration parameters:

- 1.enabled.** Enable or disable the Auto Login module. The default is false.
- 2.importFromExternalSource.** Enable or disable the import from the users' external source. The default is false.
- 3.userTokenName.** Set the HTTP header that will contain the username of the user requesting to access the portal. The default value of the header is X-AUTH-REMOTE-USER
- 4.tokenLocation.** Set where the username token is located. The default is REQUEST_HEADER

Implementation of the scenario - OSGi Configuration (4/5)

```

1 package it.smc.labs.bootcamp.liferay.security.auto.login.token.configuration;
2
3 import aQute.bnd.annotation.metatype.Meta;
4
5 import com.liferay.portal.configuration.metatype.annotations.ExtendedObjectClassDefinition;
6 import com.liferay.portal.security.sso.token.security.auth.TokenLocation;
7
8 /**
9  * @author Antonio Musarra
10  */
11 @ExtendedObjectClassDefinition(
12     category = "sso", scope = ExtendedObjectClassDefinition.Scope.COMPANY
13 )
14 @Meta.OCD(
15     id = CieCnsTokenAutoLoginConfiguration.PID,
16     localization = "content/Language",
17     name = "cie-cns-token-auto-login-configuration-name"
18 )
19 public interface CieCnsTokenAutoLoginConfiguration {
20
21     @Meta.AD(deflt = "false", name = "enabled", required = false)
22     public boolean enabled();
23
24     @Meta.AD(
25         deflt = "false", description = "import-from-external-source-help",
26         name = "import-from-external-source", required = false
27     )
28     public boolean importFromExternalSource();
29
30     @Meta.AD(
31         deflt = "X-AUTH-REMOTE-USER", description = "user-token-name-help",
32         name = "user-token-name", required = false
33     )
34     public String userTokenName();
35
36
37     @Meta.AD(
38         deflt = "REQUEST_HEADER", description = "token-location-help",
39         name = "token-location",
40         optionLabels = {
41             "token-location-" + TokenLocation.REQUEST,
42             "token-location-" + TokenLocation.REQUEST_HEADER
43         },
44         optionValues = {TokenLocation.REQUEST, TokenLocation.REQUEST_HEADER},
45         required = false
46     )
47     public String tokenLocation();
48
49     public final String PID =
50         "it.smc.labs.bootcamp.liferay.security.auto.login.token." +
51         "configuration.CieCnsTokenAutoLoginConfiguration";
52
53 }

```

Implementation of the scenario - User Importer (5/5)

The component that implements the User Importer interface (from Liferay) and which we will call ***ExternalUserImporterImpl*** is responsible for:

1. Search for the user (using the username token received from the Reverse Proxy/IDP) on the external source.
2. If the user exists on the external source, the user is searched on the Liferay portal, in order to verify that he has not been imported previously. If the user is already present on the Liferay portal, no import will be performed.
3. If the user exists on the external source and is not present on the Liferay portal, then the user will be entered as a user of the Liferay portal using data such as name, surname, etc. extracted from the external source
4. If the user does not exist on the external source, a special exception will be raised.

OOTB Liferay has the LDAP importer, in this case an ad hoc importer has been implemented that uses an in-memory repository for the users to import. The interface defined and implements is ***DetailsUserInfo***.

Implementation of the scenario - User Importer (5/5)

```

1 package it.smc.labs.bootcamp.liferay.security.auto.login.token.impl.exportimport;
2
3 ...
4
5 import it.smc.labs.bootcamp.liferay.security.auto.login.token.repository.DetailsUserInfo;
6 import
7 it.smc.labs.bootcamp.liferay.security.auto.login.token.repository.model.ExternalUser;
8 ...
9
10 import org.osgi.service.component.annotations.Component;
11 import org.osgi.service.component.annotations.Reference;
12
13 /**
14  * @author Antonio Musarra
15  */
16 @Component(immediate = true, service = UserImporter.class)
17 public class ExternalUserImporterImpl implements UserImporter {
18
19     @Override
20     public long getLastImportTime() throws Exception {
21         return 0;
22     }
23
24     ...
25
26     @Override
27     public User importUserByScreenName(long companyId, String screenName)
28         throws PortalException {
29
30         ExternalUser externalUser = _getExternalUser(
31             companyId, StringPool.BLANK, screenName);
32
33         User user = _getUser(companyId, externalUser);
34
35         if (user != null) {
36             return user;
37         }
38
39         return _addUser(companyId, externalUser);
40     }
41
42     private User _addUser(long companyId, ExternalUser externalUser)
43         throws PortalException {
44         ...
45
46         return _userLocalService.updatePassword(
47             user.getUserId(), password, false);
48     }
49
50     private ExternalUser _getExternalUser(
51         long companyId, String email, String screenName)
52         throws PortalException {
53
54         if (Validator.isNotNull(screenName)) {
55             return _detailsUserInfo.getUserByScreenName(companyId, screenName);
56         }
57
58         if (Validator.isNotNull(email)) {
59             return _detailsUserInfo.getUserByEmail(companyId, screenName);
60         }
61
62         throw new PortalException("Email or ScreenName cannot be null");
63     }
64
65     private User _getUser(long companyId, ExternalUser externalUser) {
66
67         ...
68         return _userLocalService.fetchUserByEmailAddress(
69             companyId, externalUser.getEmailAddress());
70     }
71
72     @Reference
73     private DetailsUserInfo _detailsUserInfo;
74
75     @Reference
76     private UserLocalService _userLocalService;
77
78 }

```

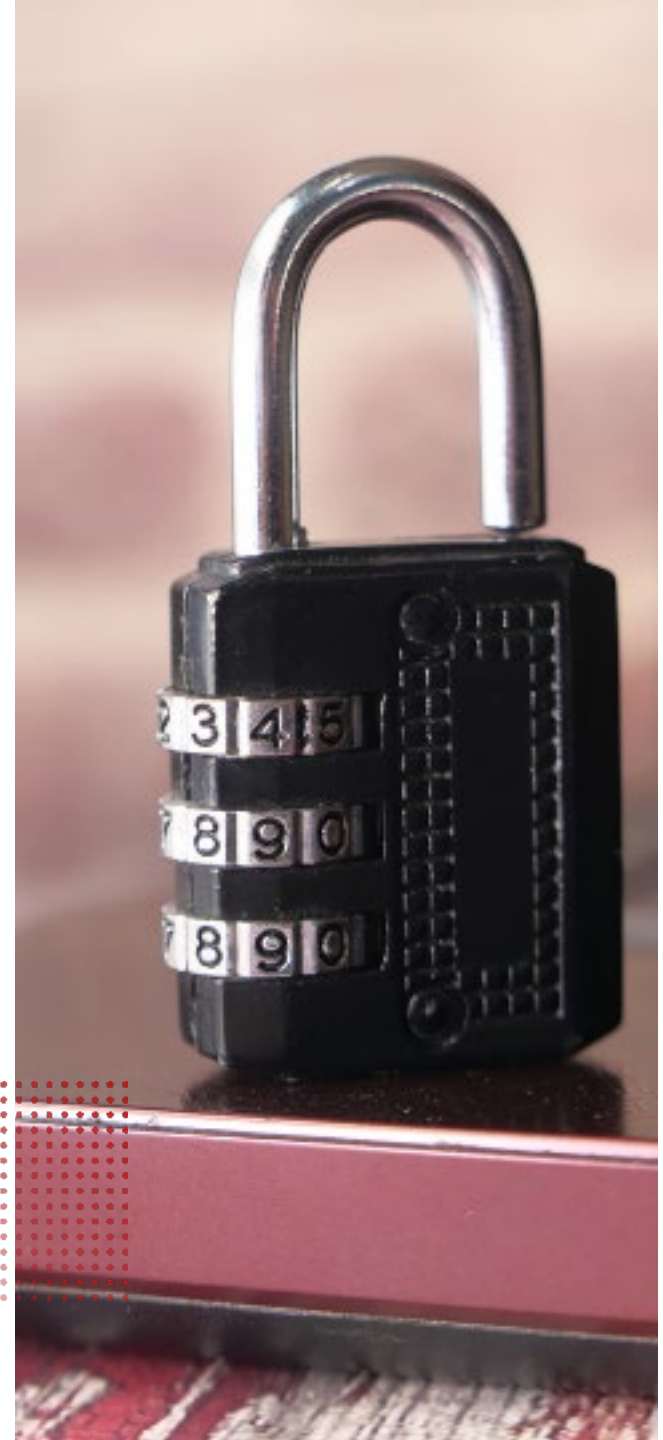
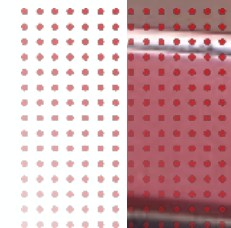
```

1 package it.smc.labs.bootcamp.liferay.security.auto.login.token.impl.remote;
2
3 ...
4 import it.smc.labs.bootcamp.liferay.security.auto.login.token.repository.DetailsUserInfo;
5 import it.smc.labs.bootcamp.liferay.security.auto.login.token.repository.model.ExternalUser;
6
7 ...
8 import org.osgi.service.component.annotations.Activate;
9 import org.osgi.service.component.annotations.Component;
10 import org.osgi.service.component.annotations.Deactivate;
11 import org.osgi.service.component.annotations.Reference;
12
13 /**
14  * @author Antonio Musarra
15  */
16 @Component(immediate = true, service = DetailsUserInfo.class)
17 public class DetailsUserInfoMemoryImpl implements DetailsUserInfo {
18
19     ...
20     public ExternalUser getUserByScreenName(long companyId, String screenName)
21         throws PortalException {
22
23         if (_externalUserHashMap.containsKey(screenName)) {
24             return _externalUserHashMap.get(screenName);
25         }
26
27         throw new NoSuchElementException(
28             String.format(
29                 "No such user for companyId: %s and screenName: %s", companyId,
30                 screenName));
31     }
32
33     @Activate
34     protected void activate() {
35         if (_externalUserHashMap.isEmpty()) {
36             ExternalUser externalUser_1 = new ExternalUser();
37             ExternalUser externalUser_2 = new ExternalUser();
38             ExternalUser externalUser_3 = new ExternalUser();
39
40             externalUser_1.setFirstName("Antonio");
41             externalUser_1.setLastName("Musarra");
42             externalUser_1.setGender("M");
43             externalUser_1.setFiscalCode("MSRNTN77H15C351X");
44             externalUser_1.setBirthDate(new Date(235173600));
45             externalUser_1.setEmail("antonio.musarra@smc.it");
46             externalUser_1.setLanguageId(LocaleUtil.toLanguageId(Locale.ITALY));
47             externalUser_1.setStatus(1);
48             externalUser_1.setUuid(_portalUUID.generate());
49             externalUser_1.setCreateDate(new Date());
50
51             _externalUserHashMap.put(
52                 externalUser_1.getFiscalCode(), externalUser_1);
53             ...
54         }
55     }
56
57     @Deactivate
58     protected void deactivate() {
59         _externalUserHashMap.clear();
60
61         if (_log.isDebugEnabled()) {
62             _log.debug("Unloaded the external users from memory");
63         }
64     }
65
66     private static final Log _log = LogFactoryUtil.getLog(
67         DetailsUserInfoMemoryImpl.class);
68
69     private static final Map<String, ExternalUser> _externalUserHashMap =
70         new HashMap<>();
71
72     @Reference
73     private PortalUUID _portalUUID;
74
75 }

```

Index of topics

- ✓ Introduction to the scenario
- ✓ Securing Liferay
- ✓ Authentication Basics
- ✓ Token-based Single Sign On Authentication
- ✓ Implementation of the scenario
- ✓ **Scenario in action (demo)**



Resources

All the coordinates of the resources used for this presentation and those deemed useful are indicated below.

- Repository GitHub of the project [smclab/cie-cns-auth-token-sso](https://github.com/smclab/cie-cns-auth-token-sso) that we have developed.
- Repository GitHub of the project [amusarra/cie-cns-apache-docker-extended-for-liferay-bootcamp-2022](https://github.com/amusarra/cie-cns-apache-docker-extended-for-liferay-bootcamp-2022) which implements the Reverse Proxy and/or IDP.
- Docker Image of the Reverse Proxy CNS/CIE on the my Docker Hub account [amusarra/cie-cns-apache-docker-extended-for-liferay-bootcamp-2022](https://hub.docker.com/u/amusarra/cie-cns-apache-docker-extended-for-liferay-bootcamp-2022).
- [Injection attacks - how to prevent with Liferay](#) on the SMC Tech Blog.
- [How to connect Keycloak and Liferay via OpenID Connect](#) on the SMC Tech Blog.
- [Carta Nazionale dei Servizi](#) on the AGID Portal
- [CIE - Carta d'Identità Elettronica](#) on the Ministero Portal



Contact

Antonio Musarra

antonio.musarra@smc.it

GitHub (@amusarra) <https://github.com/amusarra>

Twitter @antonio_musarra
https://twitter.com/antonio_musarra

LinkedIn <https://www.linkedin.com/in/amusarra/>

Thank you 