
# Deliverable 2.5 – Third Party Systems Specification Report

| | |
|---|---|
| Deliverable type | R – Document, report |
| Dissemination level | PU - Public |
| Due date (month) | M16 |
| Delivery submission date | 31.05.2024 |
| Work package number | WP2 |
| Lead beneficiary | Universidad Carlos III de Madrid (UC3M) |

This project has received funding from the Horizon Europe Framework Programme of the European Union under grant agreement No. 101094428

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Commission. Neither the European Union nor the granting authority can be held responsible for them.

## Document Information

| | | | |
|---|---|---|---|
| Project number | 101094428 | Acronym | CULTURATI |
| Project name | Customized Games and Routes For Cultural Heritage and Arts | | |
| Call | HORIZON-CL2-2022-HERITAGE-01 | | |
| Topic | HORIZON-CL2-2022-HERITAGE-01-02 | | |
| Type of action | HORIZON-RIA | | |
| Project starting date | 1 February 2023 | Project duration | 36 months |
| Project URL | http://www.culturati.eu | | |
| Document URL | https://culturati.eu/deliverables/ | | |

| | | | | |
|---|---|---|---|---|
| Deliverable number | D2.5 | | | |
| Deliverable name | Third Party Systems Specification Report | | | |
| Work package number | WP2 | | | |
| Work package name | System Development and Evaluation | | | |
| Date of delivery | Contractual | M16 | Actual | M16 |
| Version | 1.0 | | | |
| Lead beneficiary | Universidad Carlos III de Madrid (UC3M) | | | |
| Responsible author(s) | Juan Miguel Gómez Berbís, UC3M, juanmiguel.gomez@uc3m.es<br>David Santiago Garcia, UC3M, davidsga@pa.uc3m.es | | | |
| Reviewer(s) | Juan Miguel Gómez Berbís, UC3M, juanmiguel.gomez@uc3m.es<br>Eda Gürel, Bilkent Universitesi Vakif, eda@tourism.bilkent.edu.tr<br>Arzu Sibel İkinci, Bilkent Universitesi Vakif, aikinci@bilkent.edu.tr | | | |

| | |
|---|---|
| Short Description | This document details the integration of Keycloak for authentication, PostgreSQL for database management, and xWiki for content management within the CULTURATI project. |

| History of Changes | | | |
|---|---|---|---|
| Date | Version | Author | Remarks |
| 06/05/2024 | 0.1 | David Santiago García | First version |
| 15/05/2024 | 0.2 | David Santiago García | Revised after review |
| 06/06/2024 | 1.0 | David Santiago García | Revised after review |

## Executive Summary

This Third Party Systems Specification Report describes the integration of essential third-party systems within our two main applications in the CULTURATI project (content management platform and visitor application), aimed at enhancing its security, performance, and scalability.

Keycloak, an open-source Identity and Access Management (IAM) solution developed by Red Hat, has been implemented to manage user authentication and authorization, offering features such as Single Sign-On (SSO), Identity Brokering, Multi-Factor Authentication (MFA), and User Federation, thereby ensuring robust and secure access control.

PostgreSQL, a reliable and advanced open-source relational database management system, has been integrated to manage and store application data efficiently, supporting ACID transactions and complex queries with high performance and stability.

Additionally, XWiki, an open-source platform chosen for its alignment with the CULTURATI content management application's needs, provides flexibility through its modular architecture, customizable wiki content via Velocity code personalization, and seamless database interaction with Hibernate integration. Integrating XWiki with PostgreSQL ensures a reliable backend database, enhancing collaborative content management.

This report highlights the steps and execution behind these integration points, establishing a secure, scalable, and robust environment that meets our project objectives.

# Table of Contents

# 1. Introduction

This report provides a comprehensive overview of the integration of third-party systems within our core application infrastructure. As our digital ecosystem expands, leveraging robust and reliable third-party solutions becomes crucial for enhancing functionality, security, and scalability. This document focuses on three critical integrations: Keycloak for authentication and identity management, PostgreSQL for database management, and XWiki for collaborative documentation.

The report presents the key aspects of Keycloak, a widely adopted open-source identity and access management solution. It outlines how Keycloak has been seamlessly integrated with our Core Application to provide robust, secure user authentication. In addition, it details how the Core Application leverages Google's User Authentication to enhance its security measures further.

Furthermore, the report looks at PostgreSQL, elucidating its role and significance within both systems: Core Application and the Content Management System. It offers a detailed description of PostgreSQL's architecture and its key components.

Lastly, the report includes a description of the XWiki project, shedding light on its purpose, features, and how it fits into our overall system architecture and details the key aspects to achieve customization for the Content Management System in the CULTURATI project.

# 2. Core Application

## 2.1 Keycloak

Keycloak is used to manage authentication and authorization within the application. The integration with Keycloak ensures secure access to the application. Keycloak is an open-source Identity and Access Management (IAM) solution developed by Red Hat. It allows to secure the application's by providing the services, such as the following;

- Single Sign-On (SSO): Users can log in once and gain access to multiple applications without having to log in again.
- Identity Brokering and Social Login: Keycloak supports login with external identity providers such as Google, Facebook, and others.
- User Federation: Integrate with existing user directories such as LDAP and Active Directory.
- Strong Authentication: Implement multi-factor authentication (MFA) to enhance security.
- User Management: Manage user accounts, permissions, and roles.
- Authorization Services: Define and enforce fine-grained authorization policies.

## 2.2 Key features of Keycloak

1. Single Sign-On (SSO)

Keycloak allows users to authenticate once and access multiple applications. It supports standard protocols like SAML 2.0, OpenID Connect, and OAuth 2.0 for SSO.

2. Identity Brokering and Social Login

Keycloak can be a broker for various identity providers, allowing users to log in using their accounts from platforms like Google, Facebook, and GitHub.

3. User Federation

Keycloak can integrate with existing user databases, such as LDAP and Active Directory, enabling seamless user management without duplicating user data.

4. Multi-Factor Authentication (MFA)

Enhance security by requiring users to provide multiple verification forms before granting access.

5. User Management

Administrators can manage user accounts, create and assign roles, and set up permissions directly within Keycloak.

6. Authorization Services

Keycloak provides fine-grained authorization capabilities, allowing you to define and enforce access control policies based on roles, groups, and attributes.

7. Standard Protocol Support

Keycloak supports industry-standard authentication protocols, including:

- OAuth 2.0: For token-based authentication and authorization.
- OpenID Connect: An identity layer on top of OAuth 2.0.
- SAML 2.0: For SSO in enterprise environments.

## 2.3 Architecture and Components

1. Realms

A realm in Keycloak represents a space where one can manage a set of users, credentials, roles, and groups. Realms are isolated from one another, allowing one to manage multiple tenants or applications from a single Keycloak instance.

2. Clients

Clients are entities that request authentication and authorization. A client can be a web application, a mobile app, or an API. Clients are configured within a realm and can use different protocols like OpenID Connect and SAML.

3. Users

Users are individuals who interact with the applications. Keycloak manages user credentials, roles, and session information.

4. Roles

Roles define a set of permissions. Users can be assigned roles directly or through group membership. Roles can be defined at the realm level or for specific clients.

5. Groups

Groups are a way to manage users collectively. Users can belong to multiple groups, and groups can have roles associated with them.

6. Federated Identity Providers

Federated identity providers allow users to log in using their credentials from another identity provider, such as social media accounts or enterprise identity systems.

7. Authentication Flows

Keycloak provides customizable authentication flows, allowing to define how users are authenticated. This can include steps like username/password, OTP, or social logins.

## 3. Integration Points

In this section, we detail the integration points, focusing on user authentication and authorization, and utilize Keycloak to manage and streamline these processes within the application.

- User Authentication: Users authenticate through Keycloak's login page.
- User Authorization: User roles and permissions are managed via Keycloak and mapped to the application.

1. User Authentication

User authentication in Keycloak allows users to log in to the application using Keycloak's authentication mechanisms.

Authentication Flow is described below:

- Login Request: When a user tries to access a protected resource, the application redirects the user to Keycloak's login page.
- User Credentials: The user enters their credentials (only a username and password) on the Keycloak login page.
- Verification: Keycloak verifies the user's credentials.
- Token Generation: Keycloak generates a JSON Web Token (JWT) upon successful authentication.
- Redirect: Keycloak redirects the user back to the application, including the JWT in the request.
- Token Validation: The application validates the JWT to ensure Keycloak issues it and has not expired.
- Session Creation: Once validated, the application creates a session for the user and grants access to the requested resource.

2. User Authorization

Authorization within the application in Keycloak leverages its role-based access control (RBAC) features.

Authorization Flow is described below:

- Role Assignment: Users are assigned roles within Keycloak.
- Access Token: After authentication, Keycloak includes user roles in the JWT.

- Token Validation: The application extracts user roles from the JWT and uses this information to enforce access control.
- Resource Access: The application determines which resources and actions the user is authorized to access based on the user's roles.

The following Roles and Permissions are defined for our application;

- Admin: with full access to all application features and management interfaces.
- Content Creator: with access to standard application functionalities for content creation
- Data Entry Operator: with access to standard application functionalities for item creation.
- Visitor: with read-only access to the application through visitor UI.

## 4. User authentication with Google

User authentication through Keycloak is configured to use Google as an identity provider. This allows users to log in to our application using their Google credentials.

Google Authentication Flow is described below:

- Login Request: When a user attempts to access a protected resource, the application redirects the user to Keycloak's login page.
- Provider Selection: The user selects Google as the login provider.
- Redirection to Google: Keycloak redirects the user to Google's OAuth 2.0 authentication endpoint.
- Google Login: The user logs in with their Google credentials.
- Consent: If this is the first time the user is logging in, Google asks for consent to share their profile information with Keycloak.
- Authorization Code: After successful authentication, Google redirects the user back to Keycloak with an authorization code.
- Token Exchange: Keycloak exchanges the authorization code for an ID token and access token from Google.
- User Provisioning: Keycloak processes the information received from Google. If the user is logging in for the first time, Keycloak may create a new user record.
- Token Generation: Keycloak generates a JSON Web Token (JWT) for the user.
- Redirect: Keycloak redirects the user back to the application with the JWT.
- Token Validation: The application validates the JWT to ensure Keycloak issues it and has not expired.

- Session Creation: Once validated, the application creates a session for the user and grants access to the requested resource.

## 4.1 Configuration Details

The following steps outline the process (covering the necessary steps for creating a Google API project, configuring Keycloak, and updating our application to handle authentication through Google) we completed to set up Google as an identity provider in Keycloak.

**Create a Google API Project:**

- Access the Google Developers Console.
- Create a new project.
- Navigate to the "OAuth consent screen" and configure the consent screen.
- Go to "Credentials" and create OAuth 2.0 credentials.
- Note the Client ID and Client Secret provided by Google.

**Configure Keycloak:**

- Log in to the Keycloak Admin Console.
- Select the realm where Google authentication will be configured.
- Navigate to Identity Providers in the left-hand menu.
- Click on Add provider and select Google.
- Fill in the required fields:
  - Alias: google
  - Client ID: The client ID from the Google Developers Console.
  - Client Secret: The client secret from the Google Developers Console.
  - Default Scopes: openid email profile
- Save the configuration.

**Client Configuration in Keycloak:**

- Ensure the application client is configured to handle external identity providers.
- In the Keycloak Admin Console, navigate to Clients and select the application client.
- Under Settings, ensure Access Type is set to confidential or public, depending on the application's requirements.
- Configure Valid Redirect URIs to include the URL(s) that Google can redirect after authentication.

**Application Configuration:**

- Update the application configuration to handle the JWT provided by Keycloak.
- Ensure the application validates and parses the JWT to extract user information and roles.

## 5. Integration with PostgreSQL

Our application integrates with a PostgreSQL database to manage and store application data. PostgreSQL, also known as Postgres, is a free and open-source relational database management system (RDBMS) that emphasizes extensibility and SQL compliance. PostgreSQL features transactions with atomicity, consistency, isolation, durability (ACID) properties, automatically updatable views, materialized views, triggers, foreign keys, and stored procedures. It is supported on all major operating systems, including Linux, FreeBSD, OpenBSD, macOS, and Windows. It handles a range of workloads from single machines to data warehouses or web services with many concurrent users.

This section details the configuration and usage of PostgreSQL within our Spring Boot application.

### 5.1 Integration Points: Database Configuration

The application is configured to connect to the PostgreSQL database using the JDBC (Java Database Connectivity) driver. This configuration includes specifying the database URL, username, and password. The PostgreSQL server runs in a docker container in the same docker network with our main backend application. The following are the application properties for integrating into PostgreSQL;

```
datasource:
  url: jdbc:postgresql://culturati-db:5432/postgres
  username: <username>
  password: <password>
```

## 6. Content Management Application

In order to expedite the development of the CULTURATI content management application, a strategic decision was made to adopt the open-source platform XWiki. This choice was predicated upon several factors that rendered XWiki an optimal fit for our project requirements. Firstly, XWiki already boasted a suite of functionalities that aligned closely with the core objectives of CULTURATI, thus minimizing the need for extensive custom-made development efforts. Moreover, XWiki's architecture's flexibility

and customizability provided a fertile foundation upon which to tailor the platform to our specific needs.

The XWiki is built on a modern, flexible architecture that facilitates efficient knowledge-sharing and documentation management. At its core, the CULTURATI XWiki follows a client-server architecture, enabling smooth communication between the front-end and back-end components. Furthermore, the open-source nature of XWiki conferred many benefits, chief among them being the ability to augment and extend its capabilities per the evolving demands of the CULTURATI project. Leveraging the extensibility afforded by its open-source framework, we integrated custom-made features and functionalities, ensuring that the resultant solution was finely tuned to the unique requirements of our initiative.

In evaluating XWiki for the CULTURATI project, we highlight its key strengths in flexibility, scalability, community support, and continuous development, collectively ensuring a robust and adaptable content management solution.

- Flexibility: XWiki's modular architecture and extensible plugin ecosystem afford unparalleled flexibility, allowing for seamless customization and adaptation to the unique requirements of the CULTURATI project.
- Scalability: Built upon robust Java technologies and leveraging industry-standard frameworks such as Hibernate, XWiki is inherently scalable, capable of accommodating growing user bases and expanding content repositories without compromising performance or reliability.
- Community Support: With a vibrant open-source community and dedicated support resources, XWiki offers a wealth of knowledge and expertise to aid in the successful implementation and maintenance of the CULTURATI project.
- Continuous Development: Backed by an active development community and regular release cycles, XWiki benefits from ongoing enhancements, bug fixes, and feature updates, ensuring it remains at the forefront of innovation and best practices in content management.

## 6.1 Velocity Code Personalization

XWiki uses Velocity, a Java-based template engine, which allows for the customization of wiki content. It provides a powerful development platform that enables to customize the wiki to specific needs. Using structured data and in-page scripting, one can create macros and applications that extend the capabilities of the wiki.

In addition, XWiki has the Velocity template engine to facilitate dynamic content generation and customization. Velocity's intuitive syntax and extensibility enable developers to seamlessly integrate personalized logic and dynamic content rendering within XWiki's pages, enhancing the platform's flexibility and adaptability to diverse use cases.

### 6.2 Hibernate Integration

As an established Java-based ORM (Object-Relational Mapping) framework, Hibernate serves as the underlying persistence layer for XWiki, facilitating seamless interaction with relational databases. By abstracting database operations and providing transparent object-relational mapping, Hibernate simplifies data access and manipulation, enhancing the scalability and maintainability of XWiki deployments.

### 6.3 Multiple Database Support

XWiki offers native support for various relational databases, including MySQL, PostgreSQL, and HSQLDB. This versatility allows users to leverage their preferred database technologies while ensuring seamless integration with XWiki's data management functionalities, optimizing performance, reliability, and data integrity.

### 6.4 Personalized Groups and Roles

One of XWiki's defining features is its granular role-based access control mechanism, which enables administrators to define custom roles and groups with finely tuned permissions. By assigning specific privileges to user cohorts based on their roles and responsibilities, XWiki facilitates secure and controlled access to sensitive content and functionality, thereby safeguarding data integrity and confidentiality.

## 7. Integration points: PostgreSQL

For the CULTURATI project, we have chosen the PostgreSQL database. Integrating PostgreSQL and XWiki ensures the collaborative platform has a reliable and efficient backend database. PostgreSQL, known for its advanced features and stability, provides an ideal solution for managing the data needs of an XWiki platform. This section details the key steps followed for successful integration.

**Key Steps for Integration:**

- Installation: PostgreSQL was installed on the server where XWiki is running. This involves downloading the PostgreSQL software and completing the installation process.

- Database Creation: A dedicated database for XWiki was created in PostgreSQL. Along with this, a user is set up with appropriate permissions to manage this database.
- Connection Configuration: XWiki's configuration files were updated to include the connection details for the PostgreSQL database. This ensures that XWiki knows where to find and how to connect to the PostgreSQL database.
- Initial Setup: During the first-time setup of XWiki, the platform was connected to the PostgreSQL database to initialize and configure its data storage.

## Conclusion

This Third Party Systems Specification Report detailed the integration of critical third-party systems within our core application. By leveraging the capabilities of Keycloak, PostgreSQL, and XWiki, we aimed to enhance the security, performance, and functionality of our digital infrastructure.

Keycloak is a robust identity and access management solution that manages user authentication and authorization. Through features such as Single Sign-On (SSO), Identity Brokering, Multi-Factor Authentication (MFA), and User Federation, Keycloak ensures secure and efficient access control. Its ability to integrate with external identity providers and support industry-standard authentication protocols is a pivotal component of our authentication strategy.

PostgreSQL has been chosen for its reliability, advanced features, and compliance with SQL standards. It plays a crucial role in managing and storing application data for both the core application and the XWiki platform. PostgreSQL's support for ACID transactions, extensibility, and robust performance underpins the stability and efficiency of our data management processes.

The XWiki platform has been adopted to develop the CULTURATI content management application. XWiki's flexibility, robustness, and extensibility make it an optimal fit for our project requirements. It uses Velocity, a Java-based template engine, to customise wiki content and Hibernate, a Java-based ORM framework, for seamless interaction with relational databases. XWiki also offers a granular role-based access control mechanism and supports a range of relational databases while facilitating the content creation process for the final users.