Deliverable 2.3 – System Components Design and Specification Report

Deliverable type	R – Document, report
Dissemination level	PU – public
Due date (month)	M7
Delivery submission date	31.08.2023
Work package number	2
Lead beneficiary	NIMBEO ESTRATEGIA E INNOVACION SL

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	2.3			
Deliverable name	System Components Design and Specification Report			
Work package number	WP2			
Work package name	System Development and Evaluation			
Date of delivery	Contractual	M7	Actual	M7
Version	1.0			
Lead beneficiary	NIMBEO ESTRATEGIA E INNOVACION SL			
Responsible author(s)	Santiago Rondón, NIMBEO, srondon@nimbeo.com Ángel Lagares, NIMBEO, alagares@nimbeo.com			
Reviewer(s)	Metin Tekkalmaz, IOTIQ, metin@iotiq.de Neşe Şahin Özçelik, Bilkent Universitesi Vakif, nozcelik@bilkent.edu.tr Arzu Sibel İkinci, Bilkent Universitesi Vakif, aikinci@bilkent.edu.tr Eda Gürel, Bilkent Universitesi Vakif, eda@tourism.bilkent.edu.tr			

Short Description	This deliverable is intended to provide a detailed overview of the individual components, their interactions, and technical specifications within a larger system. This report serves as a blueprint for the development team, outlining the architecture and requirements to ensure a clear understanding and successful implementation of the system.

History of Changes			
Date	Version	Author	Remarks

02.08.2023	0.1	Santiago Rondón	First version
29.08.2023	1.0	Santiago Rondón	Revised after Review

Executive Summary

This System Component Design and Specification Report outlines the development and implementation of CULTURATI, an innovative online platform dedicated to fostering collective content creation for cultural heritage and arts across Europe. The project aims to utilize state-of-the-art digital technologies and cutting-edge approaches to engage end-users, including both institutions and individuals involved in Cultural and Creative Industries (CCIs) as well as citizens.

CULTURATI serves as a global content platform, offering customized games, such as Q&A games and treasure hunt-like experiences, to attract and engage consumers of cultural heritage and arts. It connects various groups in the public, including CCIs, social innovators, local, regional, and national authorities, sectoral partners, local SMEs, young people, minorities, disadvantaged groups, and citizens, actively involving them in content creation.

The platform caters to venue or site-based CCIs, such as museums, art galleries, art fairs, biennial events, palaces, castles, historic town centers, and cities. Additionally, it provides a space for CCI professionals, including artists, authors, handcraft designers, freelancers, painters, film, and animation producers, to share their knowledge and expertise in cultural and artistic merit.

The System Components Design and Specification Report delineates the architecture of CULTURATI, outlining the individual components and their interactions. It details the interface specifications, data flow, and security considerations that contribute to the seamless delivery of content to end-users. Moreover, the report encompasses performance requirements, technical specifications, and maintenance guidelines, ensuring the platform's efficiency and longevity.

By directly connecting the demand and supply sides of the cultural and creative sectors, CULTURATI endeavors to close the gap and promote active participation in content creation for heritage and arts. With its user-friendly interface and state-of-the-art technologies, the platform is poised to revolutionize how cultural heritage and arts are experienced and appreciated across Europe.

Table of Contents

Table of Figures

1. Introduction

CULTURATI emerges as a global content platform, uniquely positioned to deliver a diverse range of cultural and artistic content to end-users in an interactive and captivating manner. Through the utilization of customized games, such as Q&A games and treasure hunt-like experiences, CULTURATI aims to attract and engage individuals with various forms of cultural heritage and arts, forging a deeper connection between the audience and the wealth of creative expressions present across Europe.

The platform's versatility lies in its ability to cater to multiple stakeholders within the Cultural and Creative Industries (CCIs), including institutions, professionals, and citizens. For venue or site-based CCIs, encompassing museums, art galleries, art fairs, biennial events, palaces, castles, historic town centers, and cities, CULTURATI provides an engaging and interactive space to showcase their offerings. Moreover, CCI professionals, such as artists, authors, handcraft designers, freelancers, painters, film, and animation producers, gain a powerful platform to share their knowledge and artistic merit with a wider audience.

This report encapsulates the intricate architecture of CULTURATI, delineating the various components that constitute the platform's functionality. By providing comprehensive interface specifications, data flow insights, and meticulous security considerations, this document ensures the seamless delivery of cultural and artistic content to end-users, while safeguarding the platform from potential vulnerabilities.

Beyond technical aspects, the report addresses performance requirements and technical specifications, providing vital guidelines for maintenance and support, assuring the platform's efficiency and sustained operation. CULTURATI aims to bridge the gap between the demand and supply sides of the cultural and creative sectors by actively involving diverse groups in the content creation process, including CCIs, social innovators, local, regional, and national authorities, sectoral partners, local SMEs, young people, minorities, disadvantaged groups, and citizens.

2. System Overview

The CULTURATI system is a comprehensive solution designed to enhance the learning, experience, and navigation of visitors at indoor or outdoor cultural and natural heritage sites, such as museums, palaces, national parks, gardens, and more. It aims to foster engagement by offering curated information, educational games, and optimized navigation routes, all supported by cloud networking and real-time sensor data.

This innovative system stands as a testament to the harmonious convergence of cutting-edge technology and the timeless allure of heritage and culture. Embracing the trans-formative potential of cloud networking, the CULTURATI system ensures the seamless and instant dissemination of a wealth of knowledge and interactive experiences. Additionally, it harnesses the power of real-time sensor data, effectively transforming the exploration process into a dynamic and adaptive endeavor. By ingeniously combining these elements, the system propels visitors on a journey that transcends traditional boundaries, offering them an unprecedented level of customization and insight into their surroundings.

Moreover, the introduction of educational games brings a captivating layer of interactivity, turning the act of learning into an exhilarating experience. The system's algorithm artfully tailors these games to resonate with the visitors' unique preferences, ensuring that every engagement is both meaningful and memorable.

Integral to the CULTURATI system's efficacy is its navigation subsystem, which guides visitors through these expansive sites with the precision of a compass and the foresight of a guardian. Leveraging real-time sensor data, the system orchestrates an intricate dance of pathways, ensuring that visitors are led to the most enriching experiences while avoiding congested areas. This not only enhances the individual exploration experience but also contributes to the collective harmony of the site, preventing overcrowding and fostering an environment conducive to immersive engagement.

3. Core Application

3.1. Core Back-end (Server)

3.1.1. Technology Stack

The Core Back-end project employs Spring Boot 3.1.0; hence the technology stack can be summarized as;

Java: Spring Boot is a Java-based framework, so the primary programming language used is Java. Spring Boot takes advantage of Java's features for building robust and scalable applications.

Spring Framework: Spring Boot is built on top of the Spring Framework, which provides a comprehensive and modular approach to building enterprise applications. It includes various modules for different concerns such as dependency injection, data access, web development, and more.

Spring Boot: This is the core of the application stack. Spring Boot simplifies the process of setting up and configuring Spring-based applications. It provides auto-configuration, which automatically configures components based on the project's dependencies and application properties.

Spring Data: Spring Data provides abstractions and APIs for working with various data sources, including relational databases,which is what the Culturati application uses. It simplifies data access and persistence. The Core Backend uses **Spring Data JPA** and **Hibernate** to provide a higher level abstraction for data access.

Spring MVC (Model-View-Controller): Spring MVC is a web module within the Spring Framework that provides the infrastructure for building web applications. It handles HTTP requests, routing, and supports the development of RESTful APIs.

Spring Security: Spring Security is used for implementing authentication, authorization, and security features in the application. It helps secure endpoints, control access, and manage user sessions.

Build Tools: We use Apache Maven as our build tool to manage dependencies and build the application.

Database: PostgreSQL is used for both production and testing. Test containers are utilized for testing to ensure data separation.

RESTful APIs: RESTful APIs are provided to both the frontend projects and other modules in the Culturati project.

Testing Frameworks: JUnit and Mockito testing frameworks are used for unit tests and integration tests.

Logging: Log4j2 and SLF4J are used for logging application events and messages.

Docker: Containerization tool Docker is used to package and deploy the applications in a consistent and scalable manner.

IDE (Integrated Development Environment): IntelliJ IDEA or VSCode are used for coding, debugging, and testing the application.

3.1.2. Hardware Requirements

The Core Back-end is expected to be a small to medium-sized application, below are the general recommendations;

CPU: A modern multi-core CPU with 2-4 cores with clock speed of around 2.5 GHz or higher.

Memory (RAM): 4-8 GB of RAM for moderate traffic and data processing.

Storage: 10-20 GB of storage space would be adequate for the application code, dependencies, and database.

Network: A stable internet connection with at least 10 Mbps upload and download speeds is recommended for general connectivity and interactions with external services or APIs.

3.1.3. Software Requirements

Java Development Kit (JDK): The Core Backend application and its submodules are Java-based, so a compatible JDK is needed .

Build Tool: The Apache Maven build tool is required to manage dependencies, compile code, and package the application.

Database System: PostgreSQL is used for both the production and testing environments.

Containerization Tools: Docker is used to package the application and its dependencies into containers.

3.1.4. Architectural Patterns

The application employs the Spring MVC architecture with a client-server model, where a Spring Boot application serves as the backend for both the mobile user interface and the administrator user interface.

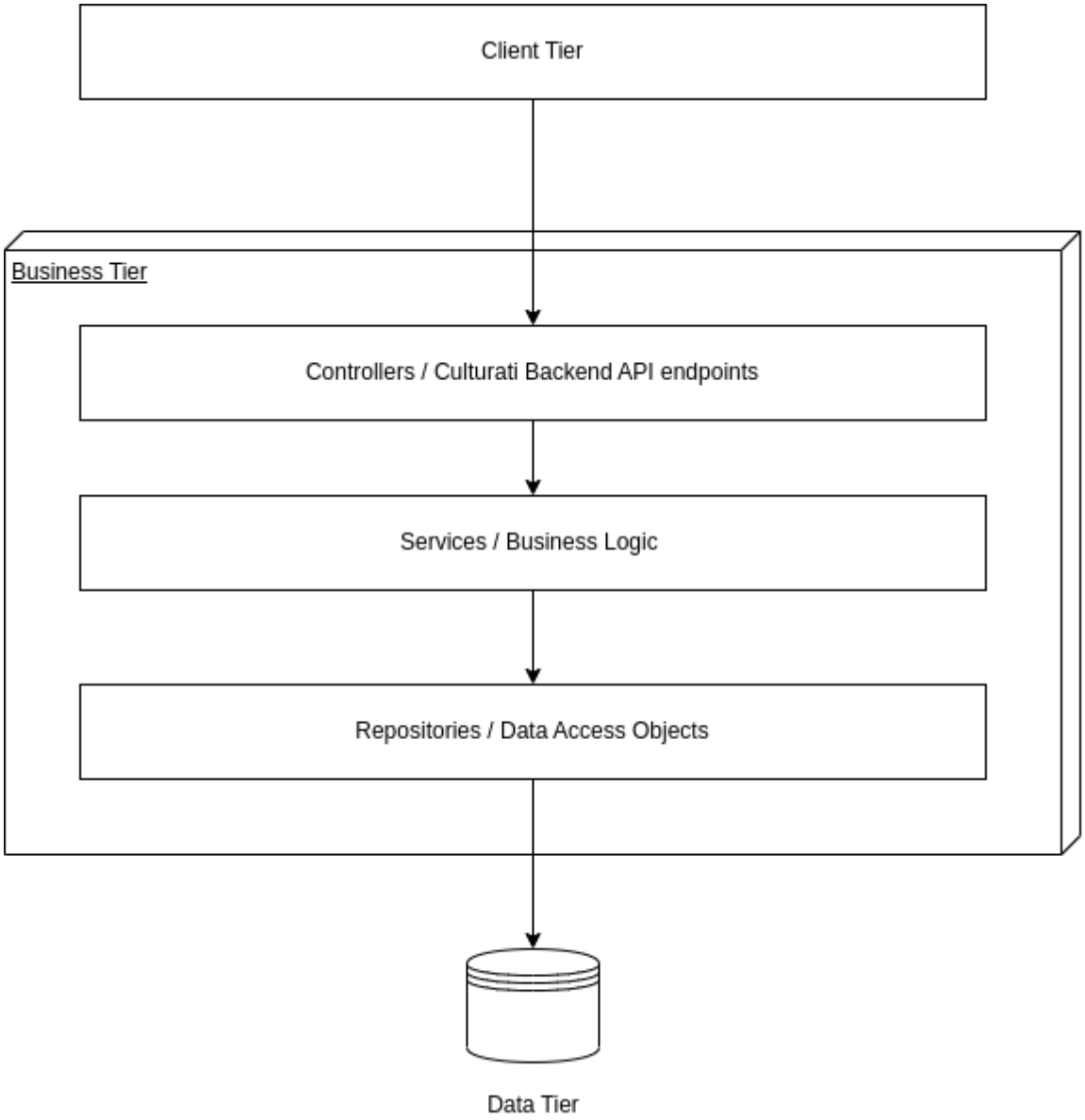

Figure 1: The structure of the Core Back-end application.

The backend will be developed using the Spring Boot framework. The backend architecture will include the following components:

Controllers: Controllers in Spring Boot handle incoming HTTP requests from both the mobile user interface and administration user interface. The requests are received through controller endpoints where the controller executes the corresponding business logic.

Webhooks: In addition to its core functionality, the application offers webhooks as a means of integrating with external modules, such as the AI subsystem. Webhooks are endpoints exposed by the application that allow external systems or modules to receive real-time notifications or callbacks from the application. The application may make requests to an external module to perform certain tasks or computations. After processing the requests, the external module can utilize the webhooks provided by the application to notify or send the results back to the application.

Services: Services encapsulate the business logic of the application. They handle complex operations, interact with data repositories, and perform necessary validations or calculations. Services are responsible for implementing the application's rules and workflows.

Data Access Layer: The data access layer consists of repositories or DAOs (Data Access Objects) that interact with the database or other data storage systems. They provide CRUD (Create, Read, Update, Delete) operations and abstract the underlying data access logic.

3.1.4.1. Data Synchronization between different data sources

To address synchronization challenges, the system will employ an eventual consistency approach. The system's replica nodes will originate from the duplicate entity representations in the content management database and the main database. By adopting eventual consistency, the system aims to ensure that all replicas eventually reach a consistent state, despite potential temporary inconsistencies.

To achieve this, the system will implement periodic synchronization of created or updated nodes on a daily basis. This synchronization process will reconcile any differences between the replica nodes, ensuring they converge towards a consistent state over time. Additionally, the system will provide users with the option to manually trigger synchronization if immediate consistency is required, granting them control over the synchronization process.

By utilizing eventual consistency, the system strikes a balance between synchronization efficiency and maintaining data integrity. It acknowledges that temporary inconsistencies may arise but ensures that over time, all replica nodes will align and reach a consistent state.

3.1.4.2. Crosscutting concerns

Aspect-Oriented Programming will be used as a technique or approach used to address crosscutting concerns in the application's architecture. AOP provides a way to encapsulate and separate crosscutting concerns from the core business logic, improving modularity and maintainability.

3.1.4.3. Transaction Management

The application will leverage Spring's transaction management capabilities, which provide declarative transaction demarcation and coordination. Appropriate transaction boundaries will be defined, allowing operations to be executed atomically. Transaction isolation levels and propagation rules will be configured based on the specific requirements of each operation. Error handling and rollback mechanisms will be implemented to handle exceptional conditions and ensure data consistency in case of failures.

3.1.4.4. Caching

To improve application performance and reduce the load on data sources especially when generating tour paths, caching will be implemented for frequently accessed data. The application will integrate with caching frameworks such as Ehcache, Caffeine, or Redis. Caching strategies, such as cache eviction policies and cache update strategies, will be defined based on the characteristics of the data. Caching annotations will be used to mark methods or data sources that can be cached.

3.1.5. Communication with Other Components

3.1.5.1. Front-end

The system architecture of the application involves seamless communication between the back-end and front-end through RESTful APIs. The front-end, developed using React and Next.js, interacts with the back-end by sending HTTP requests to the backend's API endpoints. These requests are responsible for tasks such as retrieving data, submitting forms, or performing various actions.
Upon receiving the HTTP requests, the back-end processes the incoming data and executes the required operations. This involves handling business logic, data manipulation, and any necessary interactions with databases or external services. The back-end then generates responses based on the processing results, which include requested data or appropriate status codes.

Through this communication mechanism, data flow between the front-end and back-end is facilitated, allowing for a dynamic and interactive user experience. The front-end can efficiently retrieve and update information, ensuring that the user interface remains up-to-date and responsive to user interactions. The RESTful API approach promotes a well-organized and standardized communication process, enabling different parts of the application to work harmoniously together.

3.1.5.2. Content Management Subsystem (Wiki)

The wiki instance employs a GraphQl API where it provides CRUD (Create, Read, Update, Delete) operations. These operations are made accessible through endpoints, enabling the core back-end to manipulate the wiki data. The communication is established via GraphQl queries, initiated from the core back-end and directed towards the wiki instance.

When a user intends to engage with the wiki instance through the core application, first the core back-end authorizes through an authorization request. The necessary credentials are created through the wiki instance's UI and are preconfigured in the configuration files specific to the core back-end user's tenant. Using these credentials, the core back-end establishes authorization with the wiki and then interprets and sends the users request to the wiki instance.

The core back-end facilitates CRUD operations of pages in the wiki. However, it is important to note that while the core back-end governs the Create, Read, Update, and Delete processes for pages, it does not directly engage in the creation or modification of the content within these pages.

When a user initiates the creation or modification of page content, the core back-end redirects the user to the user interface of the wiki instance. This redirection ensures that the user interfaces with the wiki instance's dedicated UI environment, where content creation and updates are managed.

This architectural delineation streamlines the overall process, with the core backend responsible for the synchronization and data integrity of the entities in the system, while the task of generating and updating content is delegated to the wiki instance's user interface.

3.1.5.3. AI Subsystem

The interaction between the Core Back-end and AI Subsystem is facilitated via a REST API, offering a comprehensive range of operations. These operations are accessible through designated endpoints, enabling both the Core Back-end and AI Subsystem to exchange and retrieve data.

The AI Subsystem is tasked with a significant role: to enhance accessibility to the extensive knowledge curated by site curators. This will be achieved by offering alternative presentation formats beyond the conventional mediums of text, audio, and video. Given unstructured data that contains valuable insights regarding visiting sites, exhibits, artworks, and narratives intrinsic to the visitor experience, the AI subsystem will focus on two primary objectives.

Firstly, it will distill the content embedded within these unstructured documents into formats that are comprehensible and user-friendly, ensuring ease of understanding for visitors. Secondly, it will contribute to the preservation and advancement of cultural heritage by facilitating the conversion of diverse documents into a standardized textual representation.

A crucial facet of the AI Subsystem's role involves the curation of a comprehensive knowledge base. This will be accomplished through the extraction of relational information between entities, leveraging the aforementioned unstructured data formats. By deciphering and delineating hidden relationships, the knowledge base will evolve into a robust repository of interconnected insights. Moreover, the knowledge base will be seamlessly integrated into the user interface, augmenting visitors' comprehension of the insights they have acquired during their exploration of the site. This integration will serve as a vital bridge between the information gleaned from their journey and its contextual significance, ensuring a holistic and enriched visitor experience. By presenting the curated knowledge base through an intuitive user interface, visitors will have the opportunity to delve deeper into the intricate web of information, gaining a more profound understanding of the subjects that pique their interest.

The user interface will be thoughtfully designed to empower visitors with tools to navigate through the knowledge base effortlessly. It will offer intuitive search functionalities, enabling users to pinpoint specific topics, exhibits, or entities of interest. Additionally, the interface will provide interactive pathways, guiding visitors through curated trails that connect related pieces of knowledge, thereby fostering a sense of continuity and coherence. Visual aids, such as diagrams, infographics, and interactive visualizations, will be employed judiciously within the user interface to elucidate complex concepts or relationships. This visual enhancement will cater to diverse learning preferences and facilitate the absorption of information across various levels of familiarity with the subject matter. In summary, the AI Subsystem's objectives encompass the facilitation of accessible knowledge dissemination and the promotion of cultural heritage preservation. Through innovative

utilization of unstructured data and advanced relational analysis, the subsystem will play a pivotal role in enriching the visitor experience and ensuring the perpetuity of invaluable cultural assets. Whenever one module requires data from the other, the process begins with an authorization request. The essential credentials are preconfigured within the configuration files of their respective applications. These credentials serve as the means by which the modules authenticate with each other and subsequently send requests to fulfill their specific requirements.

3.1.5.4. IoT Subsystem

The communication between an application and sensors is typically established through a combination of protocols, technologies, and architectural patterns to ensure efficient and reliable data exchange. Here's how a back-end application communicates with IoT sensors:

Sensor Protocols: IoT sensors are equipped with various communication interfaces and protocols that allow them to transmit data over networks. Common protocols include MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), HTTP/HTTPS, and LoRaWAN (Long Range Wide Area Network). These protocols ensure that data can be efficiently transmitted over both local and wide-area networks, catering to different IoT deployment scenarios. The selection of the protocol will be made once the specific sensors are chosen.

Data Ingestion: Back-end will provide endpoints or APIs that IoT sensors can communicate with. These endpoints are designed to receive data from sensors in a structured format. Sensors send data packets containing measurements, statuses, or events to these endpoints. The back-end application will authenticate and validate incoming data to ensure its integrity and origin.

Message Queues and Brokers: In scenarios with high data throughput, message queues or brokers can be employed to handle the communication between sensors and the back-end application. MQTT brokers, for example, efficiently manage the delivery of messages from sensors to the back-end. This asynchronous communication pattern decouples the sensors from the back-end, allowing for smoother data flow.

Data Storage: Processed data will be stored in a database, where it can be further analyzed, visualized, and queried. A Time-series database will be used to store sensor data, as they efficiently handle large volumes of timestamped data points.

Security: Ensuring the security of communication is paramount in IoT systems. This involves encrypting data in transit using protocols like TLS (Transport Layer Security) and implementing authentication mechanisms to verify the identity of sensors and the back-end application.

3.1.6. Data Storage and Management

Below, we provide an overview of the core components that drive the application's functionality:

PostgreSQL Database (*postgres:15.2-alpine*):

The project leverages PostgreSQL as the relational database management system. Known for its reliability, extensibility, and adherence to SQL standards, PostgreSQL provides a solid foundation for secure and efficient data storage. Its support for complex queries, indexing, and data integrity enforcement ensures the accuracy and integrity of the stored information. The project uses postgres image *postgres:15.2-alpine.*

Docker (*v20.10*):

The application uses Docker as a containerization platform. Docker enables the creation and management of lightweight, isolated containers that encapsulate the application and its dependencies. This approach ensures consistent deployment across various environments, simplifies scalability, and promotes a streamlined development and testing process. Docker is used to containerize the database image.

Spring Data JPA (*v3.1.2 - dependency managed by Spring Boot*) and Hibernate(*6.2.6.Final*):

To interact with the database, the project integrates Spring Data JPA in conjunction with Hibernate. Spring Data JPA provides an abstraction layer that simplifies database operations through the use of Java Persistence API (JPA) annotations. Hibernate, a popular Object-Relational Mapping (ORM) framework, seamlessly translates Java objects into database records and vice versa. This combination accelerates the development process and reduces the complexity of database interactions.

Multi-Tenancy for Data Management and Storage:

An integral aspect of the project is the implementation of multi-tenancy architecture. This design approach enables the application to efficiently manage and store data for multiple tenants within a single database instance. Multi-tenancy enhances resource utilization, reduces maintenance overhead, and ensures isolation between different clients' data. By utilizing this architecture, the

application optimizes data storage while providing a secure and organized environment for each tenant's information.

Geo Spatial Data Storage:

The geospatial data of indoor maps will be stored as GeoJSON or JSON files in the file system of the server.

3.1.7. Security

Security is a vital concern to protect sensitive data and ensure authorized access to the application. The application will employ Spring Security, a widely adopted security framework, to implement authentication, authorization, and access control mechanisms. User authentication will be implemented using standard techniques such as token-based authentication. JWT tokens will be utilized to securely transmit authentication information between the client and the server, enhancing the overall security of the application.

Role-based access control will be enforced to restrict access to specific resources or functionalities based on user roles. Security configurations, including secure communication over HTTPS, session management, and protection against common security vulnerabilities (e.g., cross-site scripting, cross-site request forgery), will be implemented to mitigate potential security risks. By integrating these robust security measures, the application aims to provide a safe and trustworthy environment for users while safeguarding sensitive data and ensuring proper access controls.

3.1.8. Error Handling and Logging

3.1.8.1. Utilizing Aspect-Oriented Programming in Error Handling

AOP is utilized in the Core Back-end in various ways, one being enabling streamlined and consistent global error handling across various components. This technique empowers the application to efficiently address exceptions and errors, ensuring separation of concerns and modularity.

AOP allows the CULTURATI application to separate cross-cutting concerns, such as error handling, from the core logic of individual components. This separation is achieved through the creation of aspects, which encapsulate specific functionalities and can be applied uniformly across multiple parts of the application. In the context of error handling, AOP enables the definition of a central error handling aspect implemented with the @ControllerAdvice annotation specifically designed for error handling, that seamlessly integrates with various controller methods, services, or repositories.

This ensures that error-related code is kept separate from the primary business logic, leading to cleaner and more maintainable code.

One of the major advantages of using AOP for global error handling in the CULTURATI application is its ability to provide a centralized and consistent response to exceptions. When an exception is thrown, the error handling aspect intervenes, allowing for standardized error messages, logging, and potentially even recovery strategies to be applied uniformly across the application. This results in a cohesive and predictable error handling mechanism, enhancing the user experience by providing clear and informative feedback whenever unexpected situations arise.

Moreover, AOP empowers the CULTURATI application with a high degree of flexibility. New error handling behaviors can be introduced or existing ones can be modified without the need to modify the individual components. This modularity not only simplifies the maintenance process but also facilitates rapid adjustments to error handling strategies based on evolving requirements or emerging issues.

3.1.8.2. Message Bundles

The Core Back-end application also utilizes message bundles to enhance the error handling mechanism. These bundles provide a mechanism to efficiently manage and present error messages in a user's preferred language, improving the application's user experience. When an error occurs, the error handling components can dynamically retrieve the appropriate error message from the message bundles based on the user's language settings. This ensures that users receive error notifications and explanations in a linguistically relevant and understandable manner. This localization aspect adds a layer of user-centricity to the error handling process, promoting clearer communication and more effective problem resolution within the Culturati application.

3.1.8.3. Logging

The system seamlessly integrates with the Log4j logging framework, offering robust and customizable logging capabilities. In this integration, a well-defined logging configuration captures appropriate log levels, including debug, info, warning, and error. Logs are generated for critical operations, exceptional conditions, and important events as they occur. The logs include relevant contextual information such as timestamps, log levels, source components, and descriptive messages, providing a comprehensive view of system activities.

Furthermore, the system implements a log rotation strategy to manage log files and prevent excessive disk space consumption. This strategy ensures that log files remain well-maintained over time, allowing for efficient monitoring and troubleshooting of system behavior.

3.1.8.4. Auditing

The application incorporates auditing capabilities for the administration application, tracking and logging important activities to ensure accountability and compliance. Audit logs capture critical events such as user actions, system configuration changes, and data modifications. The audit logs include relevant information such as user identity, timestamp, action performed, and affected resources.

The application takes advantage of Spring Data JPA's lifecycle events to implement auditing functionality. By utilizing Spring Data JPA's lifecycle events, the application can track and record details such as creation and modification timestamps, as well as user identities, associated with data entities.

3.2. User Front-end (Mobile and Web Client)

3.2.1 Technology Stack

We've built the front-end using React as the core frontend library, Redux for state management, and Vite for efficient development. Leveraging the Ionic framework allows us to seamlessly integrate native mobile features into the browser experience, eliminating the need for app downloads while providing a consistent and responsive interface.

3.2.2 Hardware Requirements

The front-end project is accessible via a web browser, optimized for a mobile experience. While we recommend using a mobile phone browser for the best experience, the project can be accessed on any device with a modern web browser.

3.2.3 Software Requirements

To access and interact with the front-end project, all you need is a modern web browser. We support the latest versions of popular browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. No additional software installations are required, ensuring easy accessibility across various devices.

3.2.4 Architectural Patterns

3.2.4.1. Component-Based Architecture

The project utilizes a component-based architecture, with the React library at its core. This approach involves structuring the application's user interface into discrete, reusable units known as components. Each component is responsible for a specific visual or functional aspect and operates independently from others. By dividing the UI into components, development becomes more modular, allowing for easier code maintenance, enhanced reusability, and efficient collaboration among developers. This architectural pattern facilitates the creation of complex interfaces by assembling these components like building blocks, resulting in a streamlined and well-organized front-end.

3.2.4.2. State Management with Redux

To manage application state and ensure a consistent data flow, Redux is employed. This pattern centralizes the state and simplifies state updates, making it easier to manage complex interactions and data changes.

3.2.4.3. Mobile-First Design

Leveraging the Ionic framework, the project follows a mobile-first design approach. This pattern ensures that the user experience is optimized for mobile devices while remaining responsive on larger screens.

3.2.4.4. Front-end Routing with React Router

React Router is used for client-side routing, enabling seamless navigation within the application. This pattern enhances user experience by providing a smooth and dynamic interface.

3.2.4.5. Internationalization (i18n)

The i18next library is integrated to support internationalization. This pattern facilitates the adaptation of the application to different languages and regions, improving its accessibility and usability.

3.2.5 User Experience Design Approaches

3.2.5.1 Crowd Management

The UX design incorporates a map component to effectively address one of the critical aspects of the application: guiding users to less congested areas thereby managing the congestion. The map allows

the user to navigate the area and find where crowds are located with ease. By making the map a central component of the application the design emphasizes the importance of crowd management.

3.2.5.2 Personalized Tours

The design incorporates prompts strategically placed throughout the application to encourage the user to try AI-generated, personalized tours. These prompts are displayed on the main screen when no active route is available to the user, ensuring maximum visibility and user engagement.

3.2.5.3 Accessibility

3.2.5.3.1 Text alternatives

Text alternatives enhance accessibility by providing textual information about visuals on the screen, enabling visually impaired users to engage with the visual content using screen readers on their devices. The system will enable content creators to add text alternatives to visual content and make this information accessible to screen readers.

3.2.5.3.2 Multi-language Support

Multi-language support enables the application to be available in multiple languages, catering to users who speak different languages. The system allows users to select their preferred language from a list of languages. The language selection option is conveniently accessible from all screens of the application, ensuring access is one click away on all screens.

Multi-language support encapsulates both UI elements and the content serving the user the content in their preferred language. By providing a personalized language experience, the application enhances user satisfaction and enables effective communication and comprehension.

3.2.5.3.3 Color Contrast and Color Modes

The application utilizes color contrast and color modes to aid both visually impaired users and non-visually impaired users.

Color modes and contrast play a crucial role in making an application accessible to users with visual impairments or color vision deficiencies. By providing alternative color modes, such as high contrast or dark mode, users can customize the UI to their specific needs and preferences. Additionally, ensuring sufficient color contrast between text and background elements enhances readability for all users.

3.3. Admin Frontend (Web Client)

3.3.1. Technology Stack

Admin frontend is powered by React and TypeScript for dynamic interfaces. Styling is accomplished using Emotion and Material-UI. i18next handles internationalization, while date handling employs date-fns and Moment.js. Axios manages data communication, and React Router aids in smooth navigation. Development benefits from Vite's speed, while ESLint ensures code quality.

3.3.2. Hardware Requirements

For seamless access to the admin frontend project, a standard computer with a modern web browser is sufficient. While the project is optimized for desktop and laptop use, it remains accessible on various devices, ensuring flexibility in usage.

3.3.3. Software Requirements

To interact with the admin front-end project, all that's needed is a modern web browser. The project is designed to be compatible with the latest versions of popular browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. No additional software installations are required, ensuring effortless access across devices.

3.3.4. Architectural Patterns

3.3.4.1. Component-Based Architecture

Embracing a component-based structure, the project divides the user interface into modular, reusable components. This approach enhances maintainability, reusability, and overall code organization.

3.3.4.2. Theming with MUI Styled Components

Leveraging Material-UI's styled components, the project introduces dynamic theming and a dark mode switch. This pattern empowers visual customization, allowing different users to experience the application with color schemes that suit their preferences. The dark mode switch enhances accessibility and reduces eye strain during low-light usage.

3.3.4.3. Adaptive Navigation with React Router

React Router is employed for adaptive navigation, ensuring a smooth and intuitive user journey through the application's various sections.

3.3.5. User Experience Design Approaches

3.3.5.1. Administrative Dashboard

The interface features a dedicated administrative dashboard, providing quick access to essential management tasks, user data, and project metrics.

3.3.5.2. Data Visualization

Utilizing interactive charts and graphs, admins can easily visualize project performance, user engagement, and other vital metrics, enabling informed decision-making.

3.3.5.3. Dynamic Tables and Data Grids

To facilitate effective data management, the application integrates dynamic tables and data grids. Admins can seamlessly navigate, organize, edit, and delete various data points, enhancing operational efficiency.

3.3.5.4. Accessibility

3.3.5.4.1. Multi-Language Support

The application offers comprehensive multi-language support, allowing users to access content in their preferred language. This ensures that language barriers are minimized.

3.3.5.4.2. Color Modes and Customization

Users have the flexibility to choose from different color modes, including dark mode, light mode, or custom color schemes. This empowers users to personalize the interface according to their visual preferences and comfort.

3.3.5.4.3. Text Alternatives

All images and visual elements are accompanied by meaningful text alternatives, ensuring that users who rely on screen readers can understand and engage with the content.

3.3.5.4.4. Keyboard Navigation

We have optimized keyboard navigation throughout the application, enabling users who may have difficulty using a mouse to navigate and interact seamlessly.

3.4. Sensors

As mentioned earlier, one of the core functions of CULTURATI is to guide visitors to the “right place” at the “right time”. One important factor affecting “right place” is the interests of the visitor. Another important factor is the current state of the place. To monitor the current state, CULTURATI is designed to make efficient use of sensors. In CULTURATI, the most important group of sensors is crowd detection sensors, since they provide necessary information to guide visitors to places of interest when it is not over-crowded. On the other hand, although being an important one, people density might not be the only factor affecting the comfort of the visit. Other factors such as temperature, air quality, weather conditions (for outdoor sites), etc. might play an important role for a pleasant visit. Even though sensors and data sources providing such information will not be integrated into CULTURATI in its early stages, they are still kept in mind for further improvement. In the following subsections, considerations related to the sensors are summarized.

3.4.1. Crowd Detection

The main purpose of crowd detection sensors is to monitor the number of people within a given area in real-time. Then this information can be used for different purposes, such as managing the capacity and occupancy levels of an area, detecting and analyzing the behavior and movement of a crowd, and implementing contingency plans or smart solutions based on the crowd density status. Aims of CULTURATI overlaps with all these purposes.

There are sensor types that apply different approaches to detect the crowd. Following is a list of the most common approaches with their own advantages and disadvantages.

- Optical sensors: These sensors use cameras or infrared beams to capture images or videos of people and analyze them using computer vision techniques.
- Radar sensors: These sensors use millimeter wave (mmWave) signals to detect and track the movement and position of people in an area.
- Time-of-flight (ToF) sensors: These sensors use infrared light pulses to measure the distance and depth of objects in a 3D space.
- Thermal sensors: These sensors use heat signatures to detect the presence of people in a specific area, without capturing identifiable visual information.
- Active infrared sensors: These sensors use infrared beams to detect the interruption of a beam of light by people passing by.
- Signal sensors: These sensors use signals (for WiFi, GSM, etc.) to detect and track the devices that are connected (or trying to connect) to a network or hotspot.

In theory, for the CULTURATI system, it is not important with which approach the crowd information is obtained as long as it is accurate enough. But there are other concerns CULTURATI tries to address requirements of different sites with different nature. For that reason, crowd detection solutions are analyzed using following criteria

- Sensing approach: As listed above, sensors may employ different approaches to detect crowds. Normally, it is not important by which approach the crowd information is obtained, as long as it is accurate enough. On the other hand, the approach has an effect on important factors, such as privacy, mounting requirements (ceiling, sides of a passage, wall, etc.), resistance to weather conditions, lighting requirements, cost, etc.
- Power requirements: AC, battery, solar, PoE, etc. Depending on the site, power may not be available and it may not be feasible to have additional cabling for power. Therefore the options to power the sensor are important.
- Communication protocol: Ethernet, WiFi, GSM, LoRa, etc. Depending on the site, one or more communication options may not be available for different reasons such as lack of infrastructure, thick walls, etc.
- Data API: This is how CULTURATI can get the data from the sensor system. In some cases, it is possible to get the data directly from the sensors, but in other cases the data should be obtained via a server which processes the sensor data.
- Real Time Data: To be able to guide people to the right places, it is necessary to have up to date data. More than a few minutes of delay is expected to hinder the performance of the system.
- Environmental specifications: In some cases, the sensors need to be installed outdoors. In such cases it is important the sensors resist the weather conditions of the site.
- Coverage: Depending on the sensor type, how big of an area or how long of a crossing line the sensor can cover. In some cases, this affects the number of sensors that need to be used, in others (e.g. big outdoor areas) this property may prevent usage of the sensor.
- Accuracy/Reliability: Although exact number of people is not always required, it is important to have a certain error window which does not accumulate over time.
- Statistical data / Dashboards: Although not a must, some systems provide some additional data such as heat maps, statistics over hours of day, days of week etc. which may be directly consumed and prevent additional implementation on the CULTURATI side.
- GDPR compliance: Privacy is one of the most important topics in CULTURATI, therefore it is important to have a GDPR compliant sensor system in place.

- Installation requirements: Each pilot site has its own specific features which restricts the installation possibilities. Therefore the installation requirements/options for the sensors are important.
- Server Locations: For the systems that require an intermediate cloud server, it is important that the servers reside in certain geographical locations (e.g. Europe) due to privacy concerns.
- Sales and support: It is important to have sales representatives and support possibilities in the pilot site countries.
- Cost: Initial and recurring costs are other factors affecting the choice of sensor solutions.

As seen above there are multiple criteria affecting the sensor choice and due to the variety of the pilot sites there is no one size fits all sensor solution. Therefore, the sensor options will be evaluated based on these criteria on a case by case basis. The back-end component that will get the data from the crowd detection sensor will be flexible to incorporate different solutions easily.

3.4.2. Other

As mentioned earlier, the people density of the site may not be the only factor affecting the quality of the visits. Additional data regarding temperature, humidity, air quality, weather conditions, noise, etc. may also be integrated into CULTURATI to improve the comfort of the visitors. In some cases, the data may be obtained from external sources (e.g. weather condition), in some cases, sites might already have installed sensors for this purpose, and in other cases, additional sensors may be required. Even though CULTURATI does not aim to incorporate such sensors initially, it will be designed to be flexible enough to add such data sources in the future. In the case of additional sensors, it is expected to have criteria similar to the ones listed in the Crowd Detection section.

4. Content Management Subsystem (Wiki)

The Wiki shall function as a centralized hub, housing vital particulars concerning museums and locales. This encompasses comprehensive exhibition insights, diverse multimedia assets, and pertinent accessibility details. Powered by an intuitive user interface, adaptable personalization features, fluid content administration, and meticulous version oversight, the Wiki guarantees an effortless upkeep of information, fortified data accuracy, and streamlined updates. Beyond this, its collaborative editing capabilities, fortified security protocols, and seamless integrations equip NIMBEO to present an immersive cultural exploration arena. This platform not only enriches visitor encounters but also fosters community-wide knowledge dissemination and interaction.

4.1. Technology Stack

Front-end technologies:

Vue.js: Progressive JavaScript framework that's designed to build user interfaces. It provides a structured and component-based approach to creating web applications. Vue.js allows developers to build reusable UI components, making it easier to manage complex user interfaces.

Quasar Framework: Quasar is a high-performance framework built on top of Vue.js. It provides a set of ready-to-use components and utilities for building responsive single-page applications (SPAs), server-side rendered apps, progressive web apps (PWAs), and mobile apps using a single codebase. Quasar's design philosophy aligns well with the CULTURATIs Wiki need for a responsive and versatile user interface.

Webpack: Webpack is a popular module bundler for JavaScript applications. It's used to bundle and optimize the front-end code, including JavaScript, CSS, and other assets. Webpack helps manage dependencies and efficiently serve the application to users.

Back-end technologies:

Node.js: Node.js is a server-side JavaScript runtime built on the V8 JavaScript engine. It enables developers to write server-side code in JavaScript, making it possible to use the same programming language on both the front end and back end. Node.js's non-blocking, event-driven architecture is well-suited for building scalable and performant applications.

Express.js: Express.js is a minimal and flexible web application framework for Node.js. It provides a set of tools and utilities for building web applications and APIs. Express simplifies tasks like routing, middleware management, and handling HTTP requests and responses.

Sequelize: Promise-based Object-Relational Mapping (ORM) library for Node.js. It simplifies working with databases by allowing developers to interact with relational databases using JavaScript objects and methods. Sequelize supports multiple database systems, including SQLite, PostgreSQL, MySQL, and more.

PostgreSQL: This is a relational database management system (RDBMS) that the Culturati Wiki supports. It is used to store the wiki content, user data, and other application-related information. The Wiki interacts with the PostgreSQL database using SQL (Structured Query Language) queries. These queries are used to perform operations such as creating, reading, updating, and deleting data. For example:

Insert: Adding new records, such as creating a new page or user account.

Select: Retrieving data, such as fetching the content of a specific page.

Update: Modifying existing data, such as editing the content of a page.

Delete: Removing data, such as deleting a page or user account.

4.2. Hardware Requirements

Server-side computer:

CPU: A quad-core or higher processor might be more suitable for handling increased user traffic and concurrent editing.

RAM: As the number of users and the amount of content increase, at least it needs 4 GB or more of RAM to maintain good performance.

Storage: As we expect a lot of content, we will need additional storage space. We are considering using solid-state drives (SSDs) for improved performance, especially in order to read and write operations.

User-side computer:

Standard computer or phone with a modern web browser is sufficient.

4.3. Software Requirements

The CULTURATI's Wiki is designed to be compatible with the latest versions of popular browsers, including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. No additional software installations are required, ensuring effortless access across devices.

4.4. Key Aspects of the Wiki

The CULTURATI's Wiki stands as an interface prioritizing user-friendliness, meticulously crafted to foster efficient and collaborative knowledge-sharing within organizations. Centered around

accessibility and simplicity, this Wiki provides an extensive and feature-enriched platform dedicated to crafting, structuring, and sustaining internal documentation and knowledge repositories.

Central to the Wiki's essence is its contemporary and user-intuitive interface, offering users a seamless means to craft and revise pages, thereby encouraging active participation from team members. By allowing real-time edits, the Wiki amplifies the spirit of cooperation, presenting an excellent solution for organizations and teams operating across various locations.

Key attributes of the Wiki encompass a versatile markdown editor, page version tracking, finely detailed access management, and an influential search capability. These facets ensure the perpetuation of current and accurate content, all while affording administrators effective oversight over permissions.

The markdown editor's adaptability further elevates collaborative efforts by facilitating users in effortlessly structuring and stylizing content. This encompassing tool enables the insertion of headings, lists, images, and links in a straightforward and streamlined manner. This editor's user-friendly nature transcends technical barriers, emboldening non-technical contributors to actively partake in the shaping and updating of content.

Furthermore, the Wiki's granular access controls empower administrators with the authority to tailor user roles and permissions, aligning them with distinct sections and functions based on each person's responsibilities and their place within the organization's hierarchy. This precision-guided approach safeguards sensitive information while ensuring that appropriate resources are available to the right individuals. Administrators are provided with seamless control over permissions, extending edit access to content creators, read-only access to stakeholders, and comprehensive access to designated leaders or supervisors. This meticulous oversight system underlines the Wiki's commitment to a safe, controlled, and collaborative space for knowledge exchange.

4.5. Wiki security component

Security constitutes a foundational pillar within the architecture of the CULTURATI's Wiki. The framework incorporates resilient authentication and authorization systems to exercise control over user entry and safeguard data confidentiality. Administrators wield the capability to formulate user roles that come with distinct permissions, thus ensuring that content creation, editing, or deletion is the prerogative of only authorized personnel.

A pivotal facet of the platform's security strategy is its emphasis on encryption for safeguarding sensitive information during both transit and storage. Data traversing between the front-end and back-end, as well as interactions with external services, are enveloped in encryption via well-established protocols like TLS (Transport Layer Security). Furthermore, data housed within the database, encompassing vital credentials and confidential data, remains encrypted even when at rest. This security measure guarantees that even if unauthorized access to the storage system were to occur, the information remains securely shielded.

A robust defense against prevalent security vulnerabilities is achieved through stringent input validation and sanitation protocols. This proactive approach acts as a formidable barrier, preventing malicious inputs—such as SQL injection or cross-site scripting (XSS) attacks—from compromising the platform's integrity or divulging sensitive data. Through meticulous validation and sanitation of user inputs at every stage of data processing, the wiki effectively slashes the chances of data breaches or unauthorized infiltration. This comprehensive approach not only bolsters security but also establishes a robust foundation upon which user trust and data integrity flourish.

4.6. Wiki side overview

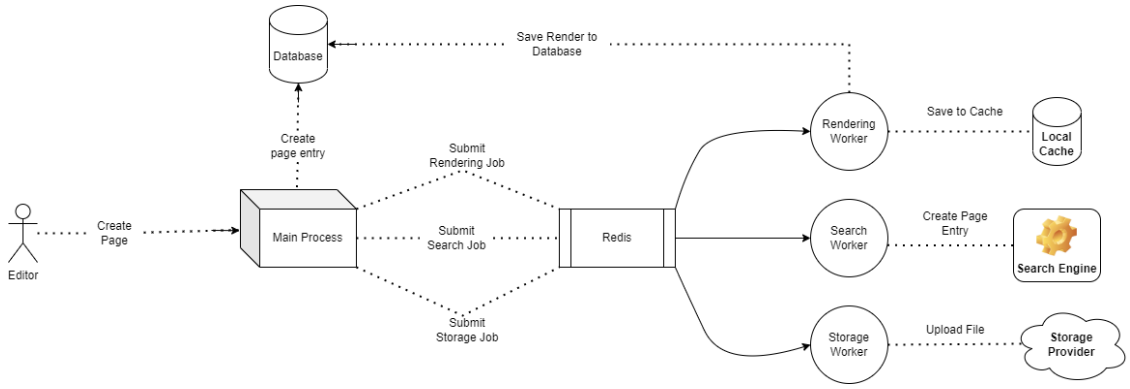

Figure 2: Wiki General Process

The infrastructure of the Wiki is established upon a cutting-edge and adaptable architecture that greatly enhances the efficiency of knowledge-sharing and documentation management. Embedded within the heart of the CULTURATI's Wiki is a client-server framework, which forms the backbone for fluid interaction between the front-end and back-end elements.

This advanced architecture not only paves the way for effective communication between these two key components but also engenders a harmonious environment that seamlessly supports the

processes of sharing insights and effectively managing critical documentation. By integrating such a robust framework, the CULTURATI's Wiki empowers users with a dynamic and cohesive platform that's engineered to cater to their knowledge-sharing and documentation needs.

Conclusion

In summary, the System Component Design and Specification Report culminates in envisioning CULTURATI, an innovative online platform set to transform the landscape of cultural heritage and art appreciation throughout Europe. At its core, CULTURATI represents the culmination of our efforts to harness cutting-edge digital technologies and novel approaches, uniting institutions and individuals within the Cultural and Creative Industries (CCIs) and citizens across the continent.

Rooted in collective content creation, CULTURATI serves as a dynamic global platform, hosting engaging and personalized experiences such as Q&A games and treasure hunts. Its reach extends far and wide, engaging diverse segments of the public - from CCIs and social innovators to local authorities, SMEs, minorities, and citizens. CULTURATI redefines content creation by providing a canvas for interaction and involvement, weaving a tapestry of insights, ideas, and perspectives that reflect the rich diversity of European culture.

Designed with precision, CULTURATI's architecture is meticulously outlined in this report. Each component finds its place within the intricate network, forming an ecosystem that seamlessly delivers content to end-users. Through detailed interface specifications, data flow mechanisms, and robust security considerations, the platform ensures a smooth and enriching experience for all.

Performance requirements, technical specifications, and maintenance guidelines lay the foundation for CULTURATI's longevity and operational efficiency. This technical backbone guarantees that the platform's transformative potential is matched with a reliable and enduring infrastructure.

As the demand and supply sides of the cultural and creative sectors find convergence within CULTURATI, it bridges gaps and fosters an atmosphere of vibrant participation. The platform's user-friendly interface and utilization of state-of-the-art technologies promise a revolution in the way we engage with cultural heritage and arts.

The System Component Design and Specification Report captures the essence of CULTURATI - an embodiment of innovation, collaboration, and the inevitable connection between technology and culture. With CULTURATI as the beacon, we venture into a future where the boundaries between creators, consumers, and cultures blur, ultimately enriching our shared narrative and understanding of European heritage and arts.