# PROJECT DELIVERABLE REPORT

**Grant Agreement Number: 101058732**

**Joint Industrial Data Exchange Pipeline**

Type: Report

# Deliverable Title: D4.2 Integration Report (Beta)

| | |
|---|---|
| **Issuing partner** | Technovative Solutions Ltd(TVS) |
| **Participating partners** | UCAM, UNITN |
| **Document name and revision** | D4.2 Integration Report |
| **Author(s)** | Rasel Ahmed(TVS) |
| **Reviewer(s)** | Miah Raihan Mahmud Arman(TVS)<br>Tanvir Islam(TVS) |
| **Deliverable due date** | 30-11-2024 |
| **Actual submission date** | |

| | |
|---|---|
| **Project Coordinator** | Vorarlberg University of Applied Sciences |
| **Tel** | +43 (0) 5572 792 7128 |
| **E-mail** | florian.maurer@fhv.at |
| **Project website address** | www.jidep.eu |

| **Dissemination Level** | | |
|---|---|---|
| **PU** | Public | ✓ |
| **PP** | Restricted to other programme participants (including the Commission services) | |
| **CO** | Confidential, only for members of the consortium (including the Commission services) | |
| **SEN** | Sensitive, limited under the conditions of the Grant Agreement | |

## Table of Contents

## List of Tables

## List of Figures

## Executive Summary

This report presents the initial integration reference for the JIDEP project, the initial version of deliverable D4.2. The primary objective is integrate services developed in Work Package-4(WP4) that have been committed in the project and enhancing functionality and usability. These tools have been successfully integrated into the JIDEP platform using an API-driven approach, chosen for its scalability and ability to facilitate data exchange across services. This integration ensures the tools are accessible and operational, supporting the project’s aim to provide valuable, interoperable solutions for the stakeholders. An API-driven integration approach allows each tool to communicate directly with the JIDEP platform, ensuring real-time data synchronization and consistent functionality. This approach enhances modularity, allowing individual components to be updated or replaced without disrupting the entire system. It also promotes scalability, making it easier to add new tools or features as project needs evolve.

# 1. Introduction

In today's fast-changing industrial world, there is a growing need for digital solutions to simplify processes, boost efficiency, and provide clear insights. This report describes these needs by creating and integrating services tailored to the stakeholders' requirements. For these services to give their total value, they must be integrated into a single platform which is the JIDEP Platform, allowing stakeholders easy access and sensible use. This integration ensures the services work smoothly, offer a consistent user experience, and enable cross-functional data use. The current JIDEP platform includes various independent services, each serving different purposes, such as the LCA tool and Circularity Calculator. While each tool adds value, the need for a unified integration limits the potential of combined data insights. The modular platform is more suitable for a scalable integration strategy supporting many services and data sources. To achieve seamless integration, JIDEP has chosen an API-driven approach. This method offers flexibility and enables real-time data sharing between services. It allows easy updates, additions, or removals of services without affecting the whole system. The API-driven approach supports scalability, adding new services and features with minimal effort. By utilising this approach, the project aims to build a reliable platform that improves data access, supports interoperability, and provides a solid foundation for future growth.

# 2. Methodology

The integration methodology for the JIDEP platform is structured to ensure secure, scalable, and efficient communication between diverse services and data sources. This methodology emphasises modularity, API-driven integration, and a hybrid data storage and processing approach, ensuring each component works harmoniously to support the platform's objectives [1].

## 2.1 API-Driven Integration

Ensure seamless communication and modular integration between services.

- **RESTful API Design**:
  - Each service in the platform (e.g., Collaborative Service, LCA Service, Circularity Calculator) is developed as an independent REST API. This modular design enables each service to function as a standalone unit, making it easy to add, remove, or update services without affecting others.
- **Centralised API Gateway**:
  - All service requests are routed through a centralised API Gateway, which manages and secures all interactions.
  - The API Gateway handles request routing, load balancing, and security (authentication and authorisation). It also ensures that requests are directed to the correct service based on user permissions and service requirements.
- **Authentication and Authorisation**:
  - The Authentication Service verifies user identities and permissions through the JWT protocol. This approach ensures only authorised users can access sensitive data and functionalities across different services.

## 2.2 Hybrid Architecture

Combine the strengths of both blockchain and traditional storage systems to support transparency, security, and efficiency.

- **Blockchain Integration**:

    - A Distributed Ledger Technology (DLT) Service is integrated to manage and record immutable, transparent transactions on the public blockchain. Blockchain provides a secure audit trail for data, requiring high transparency and integrity levels
    - Blockchain is used explicitly for critical transaction data to maintain immutability, while other data is stored off-chain for performance and cost efficiency.
- **Off-Chain Storage**:
  Data not requiring blockchain-level immutability is stored in traditional databases and distributed file systems. These include:
    - **Off-Chain Database**: This is for quick and secure access to frequently used or large data sets that do not need to be recorded on the blockchain.
    - **Global Distributed File System (GDFS)**: To handle large files and support distributed data access across multiple locations, providing scalability and resilience.
    - **Knowledge Graph**: For structured data representation, enabling advanced querying and relationship mapping between different data points.

### 2.3 Component-Based Design

Enable flexibility, scalability, and independent management of services.

- **Microservices Architecture**:
    - Each functionality within the JIDEP platform (Collaborative Service, Material Passport Service, etc.) is developed as a separate microservice with its API. This modular design allows for each service's isolated development, testing, and deployment, enhancing overall flexibility.
- **Independent Databases and Resources**:
    - Based on its unique requirements, each service can access specific databases or resources (e.g., the Knowledge Graph using the Ontology Service or the LCA Service interacting with the ecoinvent Database). This design reduces data duplication and optimises resource usage.

## 3. Integration Architecture

The JIDEP platform follows a hybrid architecture integrated through an API-driven approach. The architecture comprises several modules and services, each fulfilling a specific role. This structured design facilitates data sharing, integration, and secure operations across various components.

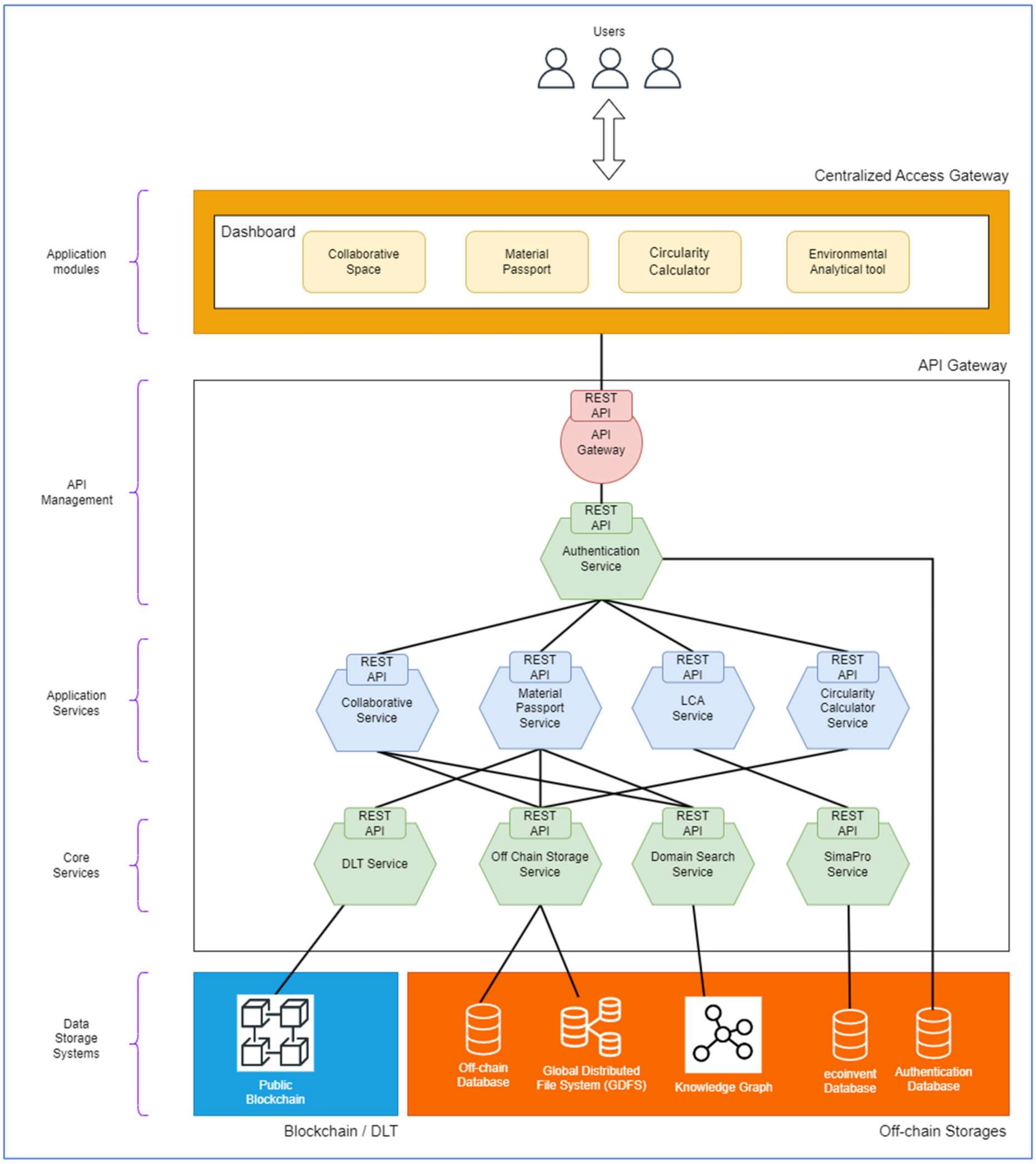

*Figure 1: High Level architectural view of JIDEP platform's components*

**Key Components:**

1. **User Interaction Layer**:
   - **Dashboard**: The user interface consists of a dashboard that provides access to several core tools, including:
     - **Collaborative Space**: For team-based interactions and shared data access.
     - **Material Passport**: For create, update, view and delete material passports.
     - **Circularity Calculator**: A tool for assessing circularity of materials.
     - **Environmental Analytical Tool**: Offers environment-related analysis and insights.

These tools provide stakeholders a seamless, centralised experience, aligning with the primary services at the backend.

2. **Integration and Control Layer**:
    - **API Gateway**: Serves as the central communication point, routing API requests from the user-facing dashboard to various backend services.
    - **Authentication Service**: Ensures secure access and identity management across the platform, managing permissions and authentication for user and service access.
3. **Core Services Layer**: Each of these services is designed as a REST API for efficient communication and scalability. They form the primary functional units that support the dashboard tools and interact with the underlying data sources.
    - **Collaborative Service**: Manages collaboration features such as data sharing and user management.
    - **Material Passport Service**: Handles data related to materials passport and their lifecycle information.
    - **LCA (Life Cycle Assessment) Service**: Supports environmental impact assessments of products and materials.
    - **Circularity Calculator Service**: Provides calculations and assessments related to materials circularity.
    - **DLT (Distributed Ledger Technology) Service**: Manages distributed ledger functionalities, ensuring transparency and immutability for specific transactions.
    - **Off-Chain Storage Service**: Stores large or complex data sets not stored directly on the blockchain for efficiency.
    - **Domain Search Service**: Enables search capabilities across various data domains.
    - **SimaPro Service**: Integrates with SimaPro, a tool for environmental impact assessment, complementing the LCA Service.

4. **Data and Storage Layer**: The foundational storage and data management components of the JIDEP platform include a mix of databases and file systems, integrated to manage large data sets securely and efficiently.
    - **Public Blockchain**: Provides an immutable ledger for recording critical, verified transactions in JIDEP we are storing the HASH data of a passport.
    - **Off-Chain Database**: Stores data that does not require the full transparency and security of blockchain but still needs robust management.
    - **Global Distributed File System (GDFS)**: A distributed storage system for handling large data files across the platform.
    - **Knowledge Graph**: Facilitates the structured representation of relationships between data points, enabling complex data querying and reasoning through ontology.
    - **Ecoinvent Database**: Houses environmental data to support sustainability and impact assessments.
    - **Authentication Database**: Manages authentication-related data, aiding the Authentication Service in user management.

# 4. API Design and Development

## 4.1 API Standards and Protocols

The JIDEP platform follows a standardised RESTful API approach for all services, ensuring consistency, modularity, and ease of integration.

- **RESTful Design**:
    - Each service uses REST principles, enabling consistent and stateless interactions across the platform.
    - APIs support CRUD operations (Create, Read, Update, Delete) where applicable, allowing easy data manipulation within each service.
- **HTTP/HTTPS Communication**:
    - Secure HTTPS protocols are used across the platform, protecting data in transit from potential security threats.
    - Standardised HTTP methods (GET, POST, PUT, DELETE) are consistently implemented, making the APIs intuitive for developers.
- **Data Format**:
    - JSON is the standard data format for requests and responses, ensuring compatibility and readability across all services

## 4.2 API Specification and Documentation

To ensure easy use and maintenance, each service within the JIDEP platform is well-documented and follows an OpenAPI/Swagger specification. This approach provides developers with an apparent, accessible reference.

- **OpenAPI/Swagger Documentation**:
    - OpenAPI/Swagger documents each service's API endpoints, parameters, response formats, and status codes, serving as a comprehensive guide for interacting with the APIs.
    - Each endpoint specification includes:
        - **Request and response formats**: Detailed schema for expected data structures.
        - **Parameter details**: Query parameters, path variables, and body content.
        - **Example requests and responses**: Sample JSON payloads for typical requests and responses.
        - **Error codes and descriptions**: Defined HTTP status codes and error messages for each possible outcome, aiding in consistent error handling across the platform.
- **Interactive API Documentation**:
    - Swagger UI provides an interactive interface where users can test API endpoints, view example requests and responses, and understand data flows. This makes it easier to onboard developers and troubleshoot issues.

## 4.3 Authentication and Authorisation Mechanisms

The JIDEP platform implements a JWT (JSON Web Token)-based approach for authentication and authorisation, combined with Role-Based Access Control (RBAC) for fine-grained access control.

- **JWT Tokens**:
    - Upon successful authentication, users receive a JWT token to validate their identity and access rights on subsequent API requests.
    - JWTs are passed in the Authorization header (Authorisation: Bearer <token>) of each API request, ensuring that only authenticated users can access the platform's resources.
- **Role-Based Access Control (RBAC)**:

    - Access levels are defined based on roles, such as Admin, Editor, Viewer, etc., with specific permissions for each role. RBAC helps restrict access to sensitive data and functionality based on user roles.
    - Each service enforces role-specific permissions at the API Gateway level, ensuring that only users with the required privileges can perform specific actions.
    - For example:
        - **Admin** roles have full access to all CRUD operations.
        - **Editor** roles have access to create and update but not delete.
        - **Viewer** roles be limited to read-only access.
- **Session Management and Token Expiry**:
    - JWT tokens are set with expiry times to enhance security. Requiring re-authentication after a specified period reduces the risk of unauthorised access.
    - The platform also implemented refresh tokens, which would allow users to renew their access tokens without frequent logins, enhancing user experience without compromising security.

# 5. Continuous Integration (CI) and Continuous Delivery (CD)

The JIDEP platform leverages a robust CI/CD pipeline to ensure smooth, reliable, and efficient development and deployment processes. This approach supports rapid iteration, high code quality, and seamless deployment to production, making it essential for the platform's agile and scalable development needs. The CI/CD pipeline, facilitated by GitHub Actions, automates the process from code integration to deployment. It includes continuous integration for code quality and testing and continuous delivery to deploy updates via Kubernetes [2]. This setup ensures rapid, reliable deployments while leveraging GitHub's integration and Kubernetes' orchestration capabilities.

## 5.1 Continuous Integration (CI):

GitHub Actions automates the CI pipeline, triggering builds and tests upon pull requests or branch merges. Code changes are validated through automated tests and quality checks, ensuring integration stability.

**Build Automation**:

- GitHub Actions triggers Docker builds upon each merge, creating a container for each service.
- The pipeline validates the Docker image, ensuring it adheres to the configuration needed for Kubernetes.

**Automated Testing**:

- Unit, integration, and end-to-end tests run in the GitHub Actions pipeline. This setup allows quick feedback on code quality and functionality.
- Tests execute within Docker containers to mimic production environments, catching container-specific issues early.

**Merge Workflows**:

- GitHub Actions manages branch protection rules, allowing merges only when all checks pass.
- Pull requests automatically trigger CI workflows, ensuring that only validated code reaches the main branch.

## 5.2 Continuous Delivery (CD):

The CD pipeline uses GitHub Actions to build Docker images and push them to a container registry(Docker Hub). Kubernetes pulls these images from the registry, deploying them in the production environment with minimal downtime.

### 5.2.1 Deployment Automation and Release Processes

- **Docker Image Deployment**:
    - Upon successful CI checks, GitHub Actions builds and tags Docker images, then pushes them to the container registry.
- **Kubernetes Deployment**:
    - Kubernetes pulls the latest images and deploys them across clusters. The platform uses Kubernetes manifests or Helm charts to define deployment configurations, including replica counts, resource allocations, and networking rules [3].
    - Rolling updates in Kubernetes ensures minimal downtime, replacing containers incrementally while monitoring health.

### 5.2.2 Rollback and Recovery Mechanisms

- **Automated Rollbacks**:
    - Kubernetes can automatically roll back to a previous image if issues are detected during deployment, ensuring stability.
    - Versioned Docker images allow the platform to revert to earlier stable versions in case of critical failures.
- **Cluster Monitoring and Health Checks**:
    - Kubernetes performs health checks (readiness and liveness probes) to monitor container status. Failed checks automatically restart containers or trigger rollbacks if the deployment is unstable.

### 5.2.3 Monitoring and Feedback Loops for Production Releases

- **Real-Time Monitoring**:
    - Kubernetes, combined with monitoring tools like Prometheus and Grafana, provides real-time insights into application performance, resource usage, and system health.
- **Feedback Loop**:
    - GitHub Actions notifications and alerts from Kubernetes give developers immediate feedback on deployment success or failure.
    - User feedback is also collected post-deployment to identify potential issues or areas for improvement.

## 5.3 Tools and Platforms

- **GitHub Actions**: Orchestrates the CI/CD pipeline, automating merging, testing, and deployment tasks upon branch changes.
- **Docker**: Containerizes each service, enabling consistency and portability across environments.
- **Kubernetes**: Manages container deployment, scaling, and orchestration, ensuring the platform can handle varying loads and maintain availability.

# 6. Security and Compliance

The JIDEP platform follows stringent security and compliance standards to protect user data, ensure platform reliability, and adhere to regulatory requirements. The primary framework guiding this approach is the OWASP Top 10, which outlines the most critical web application security risks. Below is a table summarising key security considerations aligned with the OWASP Top 10 and additional compliance measures [4].

*Table 1: OWASP Top 10 and additional compliance measures*

| OWASP | Security Considerations | Compliance Measures |
|---|---|---|
| **A01 - Broken Access Control** | Implement strict Role-Based Access Control (RBAC) and JWT-based authentication to enforce user permissions. | Conduct regular access reviews, and enforce least privilege principles across services. |
| **A02 - Cryptographic Failures** | Ensure all data in transit is encrypted with HTTPS and enforce secure protocols (e.g., TLS 1.2+). Encrypt sensitive data at rest. | Regularly review encryption standards, and comply with GDPR and ISO 27001 for data protection. Ensure keys and certificates are rotated periodically. |
| **A03 - Injection** | Use parameterised queries for database interactions and sanitise inputs to prevent SQL, NoSQL, and command injection. | Perform static code analysis with tools like SonarQube and CodeQL to detect injection vulnerabilities. |
| **A04 - Insecure Design** | Implement security by design principles. Review and enhance API endpoints, data validation, and error handling. | Regular security reviews during development, align with Secure Development Lifecycle (SDL) practices. |
| **A05 - Security Misconfiguration** | Harden Docker images, Kubernetes configurations, and ensure least privilege for services. Avoid default configurations. | Run regular configuration scans using tools like Aqua Security or Kube-bench to detect misconfigurations in Docker and Kubernetes. |
| **A06 - Vulnerable and Outdated Components** | Use dependency management tools (e.g., Dependabot in GitHub) to track and update libraries. | Regularly scan dependencies for vulnerabilities with tools like OWASP Dependency-Check. Maintain an up-to-date inventory of components. |
| **A07 - Identification and Authentication Failures** | Use multi-factor authentication (MFA) for sensitive accounts, secure password storage (e.g., bcrypt hashing), and enforce strong password policies. | Periodic security audits of authentication mechanisms, and require MFA for administrative users. Comply with ISO 27001 access management. |
| **A08 - Software and Data Integrity Failures** | Implement integrity checks on data transmissions, and ensure that automated CI/CD processes are secure (e.g., signed commits). | Enforce signed images in Docker and container registry policies, use GitHub Actions Secrets to manage sensitive information in pipelines. |

| | | |
|---|---|---|
| **A09 - Security Logging and Monitoring Failures** | Use centralised logging with SIEM tools for real-time monitoring. Enable detailed logging for access and security events. | Ensure logs are retained according to GDPR requirements. Regularly review and respond to security alerts. |
| **A10 - Server-Side Request Forgery (SSRF)** | Implement strict URL whitelisting, disable unused services, and ensure firewalls restrict server-to-server requests. | Perform regular security testing and monitoring for SSRF vulnerabilities. Restrict access to sensitive internal services. |

**Additional Compliance Considerations**

- **Data Privacy**: Compliance with GDPR and CCPA by ensuring data minimisation, user consent management, and the ability for users to access or delete their data.
- **Security Audits**: Conduct regular security audits and penetration tests to identify and mitigate vulnerabilities, ensuring compliance with industry standards (e.g., ISO 27001, NIST).
- **Secure Development Lifecycle (SDL)**: Integrate security assessments at each stage of development, from design to deployment. This includes threat modelling, secure code training for developers, and automated security testing.
- **Incident Response**: Define and test an **incident response plan** to address security breaches quickly, including notifications, containment, and post-incident review.

# 7. Performance Analysis

This section will be discussed in the final version of the deliverable.

# 8. Challenges and Resolutions

The JIDEP platform encountered challenges primarily in communication, scalability, data consistency, and dependency management. Solutions included optimising services communication, implementing robust access control, enhancing CI/CD efficiency, and managing dependency risks effectively. These resolutions ensure platform stability, scalability, and security.

*Table 2: Challege and Resolutions*

| Challenge | Description | Risks Identified | Solutions and Workarounds |
|---|---|---|---|
| **Microservices Communication Issues** | Occasional latency and failures in communication between microservices, especially under high load. | Risk of service unavailability and degraded performance. | Implemented retries with exponential backoff, optimised API Gateway configurations, and introduced service discovery mechanisms. |
| **Database Performance Bottlenecks** | High query latency observed in certain complex queries under load. | Risk of slower response times affecting overall platform performance. | Applied indexing to optimise query speed, used read replicas for load distribution, and |

| | | | |
|---|---|---|---|
| | | | optimised database queries. |
| **External Service Dependency Latency** | Latency issues with third-party service dependencies affected platform response times. | Risk of user frustration due to delayed responses from external dependencies. | Implemented caching strategies for frequent data, used async calls where possible, and set timeouts with fallback data handling. |
| **Managing Deployment Rollbacks** | Rollbacks in Kubernetes were complex, especially with interconnected services and database dependencies. | Risk of downtime and data inconsistency during rollbacks. | Implemented versioned deployments and blue-green deployment strategies to ensure smooth rollbacks with minimal impact. |
| **Managing Deployment Rollbacks** | Rollbacks in Kubernetes were complex, especially with interconnected services and database dependencies. | Risk of downtime and data inconsistency during rollbacks. | Implemented versioned deployments and blue-green deployment strategies to ensure smooth rollbacks with minimal impact. |

## 9. Testing and Validation

This section will be discussed in the Deliverable 4.2 final version.

## 10. Documentation and Training

Comprehensive documentation and training were provided to ensure the effective use and management of the JIDEP platform. Detailed user manuals and technical documentation were delivered, covering system functionalities, configuration, and troubleshooting. Training sessions were conducted for both end users and administrators, equipping them with the necessary skills to operate the platform efficiently. Additionally, knowledge transfer sessions and support materials, including FAQs and quick reference guides, were made available to foster long-term usability and enable self-sufficient troubleshooting and maintenance. This approach ensures that all stakeholders are well-prepared to leverage the platform to its full potential.

## 11. Conclusions

In conclusion, the development and deployment of the JIDEP platform have successfully addressed the project's core objectives of delivering a secure, scalable, and user-friendly solution. Through a carefully structured architecture leveraging microservices, Kubernetes orchestration, and robust CI/CD pipelines, the platform is well-equipped to handle diverse workloads and integrate seamlessly with external systems. Comprehensive documentation and targeted training have empowered users and administrators to utilise the platform effectively, while the established processes for continuous improvement will support future growth and adaptation.

## References

[1] "System Integration Explained: Methods and Approaches," Snaplogic, 12 11 2024. [Online]. Available: https://www.snaplogic.com/blog/system-integration-types-and-approaches.

[2] B. Douglas, "How to build a CI/CD pipeline with GitHub Actions in four simple steps," 23 07 2023. [Online]. Available: https://github.blog/enterprise-software/ci-cd/build-ci-cd-pipeline-github-actions-four-steps/.

[3] "Deployments," 12 11 2024. [Online]. Available: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

[4] "OWASP Top Ten," 12 11 2024. [Online]. Available: https://owasp.org/www-project-top-ten/.

## Acronyms and Abbreviations

| Acronym | Meaning |
|---|---|
| API | Application Programming Interface |
| CCPA | California Consumer Privacy Act |
| CD | Continuous Delivery |
| CI | Continuous Integration |
| DLT | Distributed Ledger Technology |
| FAIR | Findability, Accessibility, Interoperability, and Reusability |
| GDPR | General Data Protection Regulation |
| GDFS | Global Distributed File System |
| HTTPS | Hypertext Transfer Protocol Secure |
| ISO | International Organization for Standardization |
| JSON | JavaScript Object Notation |
| JWT | Json Web Token |
| LCA | Life Cycle Analysis |
| MFA | Multi-Factor Authentication |
| OBDI | Ontology Based Data Integration |
| OWSAP | The Open Worldwide Application Security Project |
| RBAC | Role-Based Access Control |
| SDL | Secure Development Lifecycle |
| SIEM | Security Information and Event Management |
| SQL | Structured Query Language |
| SSRF | Server-Side Request Forgery |
| TLS | Transport Layer Security |
| URL | Uniform Resource Locator |