Scitt Transparency and Provenance Saas Service

Author: Serhiy Pikho
Student ID: 13108607
Type 4: BUCI 027S6 Computing

1. Abstract

The subject of the following report is a proposed solution to the problem of untrusted data. The solution roughly follows the definition of a "transparency service" as defined by the emerging SCITT (Supply Chain Integrity, Transparency, and Trust) consortium by the IETF (Internet Engineering Task Force). [1] [2]

This document will cover the development of this service, from the market research, design of the SAAS service, and the implementation of the service.

2. Table of Contents

3. Introduction

3.1. Problem Statement

The landscape of industry and technology is rapidly evolving, increasingly leaning towards automation and digital transformation. This shift necessitates the swift exchange of data between entities, a process that faces significant challenges when the authenticity and provenance of data cannot be assured. The traditional solution has always involved manual verification by personnel, a method that negates the efficiency automation aims to achieve.

This issue spans both tangible and intangible assets. In the realm of physical goods, there is a case study for a nuclear power management company, which needed to ensure that its subcontractors adhere to the proper disposal protocols for nuclear waste. This case study highlighted the need for two-way trust: as the subcontractors also needed to be confident they're following the most current guidelines for waste disposal.[3]

In the digital domain, The vulnerabilities exposed by the log4j incident underscore the fragility of the software supply chain. This culminated in the US government making a software bill of materials (SBOM), a requirement for official projects. [4]

The emergence of AI and synthetic media content has amplified the urgency for transparency mechanisms. The proliferation of these technologies in the last few years [5] has highlighted the need to track the provenance and ownership of photos. Adobe, for example, has introduced the C2PA (Coalition for Content Provenance and Authenticity) framework, a datastructure and eco system that tracks provenance of images and videos.[6]

3.2. Potential solutions

Efforts to enhance digital trust and transparency Have been made. Initiatives such as Google's Certificate Transparency project[7] and Microsoft's Explainable AI initiative[8] seek to address broader aspects of digital security and accountability. Together, these efforts signify a growing recognition of the need to safeguard the integrity of our increasingly digitized world.

These are examples of closed source solutions, which have an inherent cost and distrust associated with them. On the other hand, the Supply Chain Integrity, Transparency, and Trust (SCITT) is an open alternative to meet these requirements, aiming to establish a framework for the secure and transparent exchange of data provenance information within and across supply chains. Its design principles prioritize agnosticism regarding the type of data exchanged and compatibility with pre-existing standards, ensuring broad applicability and integration flexibility.

Despite being in the early stages of its development, with cryptographic assurance mechanisms still under review by the IETF, the concept of a transparency service is gaining significant momentum. This is evidenced by the emergence of multiple startups that are dedicated to developing services aligned with SCITT's principles, signalling a strong movement towards compliance with this proposed standard.

3.3. Market review of the current solutions

General Provenance Platforms

- Data Trails - A platform offering comprehensive provenance services.
- Certifaction - Provides verifiable document authenticity and provenance.
- Data Freedom Foundation - Focuses on open standards for data provenance.
- MEA Integrity - Offers solutions for data integrity and provenance.
- Tractiv - Specializes in supply chain transparency and provenance.

Assurance Against AI-Generated Content

- Open Origins - Works on assurance against AI-generated content.
- BBC Verify - BBC's initiative for verifying content authenticity.
- DeepActual - Provides tools for detecting AI-generated misinformation.

Software Bill of Materials (SBOM) Specific

- Docker Scout - A tool for generating and managing SBOMs in software development.

AI-Specific Transparency

- Trustible AI - Focuses on creating trustworthy AI systems.
- DataRobot - Offers auditing and approval features for AI deployments.

Notification of Compromised Certificates

- Certificate Transparency by Google - A project aimed at logging and monitoring SSL certificates to detect compromises.

Physical Goods Specific

- Bosch Origify - Targets transparency and provenance for physical goods.

4. Design and Implementation Goals

The primary objective of this project is to deploy a Software as a Service (SaaS) platform dedicated implements the core Ideas of the "SCITT transparency service".

structured around three phases phases:

1. Minimum Viable Product (MVP) - Locally developed and tested
2. Deployment - CI and CD pipelines are established for building and deploying the service
3. Extended Goals.

MVP

The MVP phase focuses on laying a solid foundation for the service, ensuring it is both functional and scalable. It achieves the lowest level of functionality required to provide value to users, specifically a neutral third party that can hold attestations of data.

Key components include:

- **API Accessibility:** Develop a robust API that enables seamless integration and interaction with the service over the internet. This API will be the backbone of the service, facilitating data submissions, retrievals, and attestations.
- **Comprehensive Testing and Documentation:** Prioritize thorough testing to guarantee reliability and security from the outset. Comprehensive documentation will accompany the API, providing clear guidelines and support for developers and users, ensuring ease of adoption and effective utilization.
- **Scalability:** Architect the service with scalability in mind to handle varying loads efficiently. This ensures the service can grow with user demand, accommodating spikes in traffic without compromising performance.

Deployment

- **CI/CD Pipelines:** Establish continuous integration and continuous deployment pipelines to automate the build, testing, and deployment processes. This will streamline the development lifecycle, ensuring rapid and reliable updates to the service.

Extended Goals

After establishing a solid MVP and deployment functionalities, the project will shift towards broader functionalities and enhanced user experiences:

- **Web Interface:** Develop an intuitive web interface that allows users to interact with the service directly. This interface will cater to users who prefer graphical interactions over API calls, broadening the service’s accessibility.
- **SCITT-based Cryptographic Assurance Mechanisms:** Integrate advanced cryptographic assurance mechanisms based on the SCITT framework. This will enhance the security and trustworthiness of the data handled by the service, providing verifiable transparency across the supply chain.

- For a full product this will be a hard requirement, If there are no cryptographic assurance mechanisms, the service could get dragged into the legal disputes of its clients, and would need to prove that it did not tamper with the data for the benefit of one of the parties.

5. Use Cases

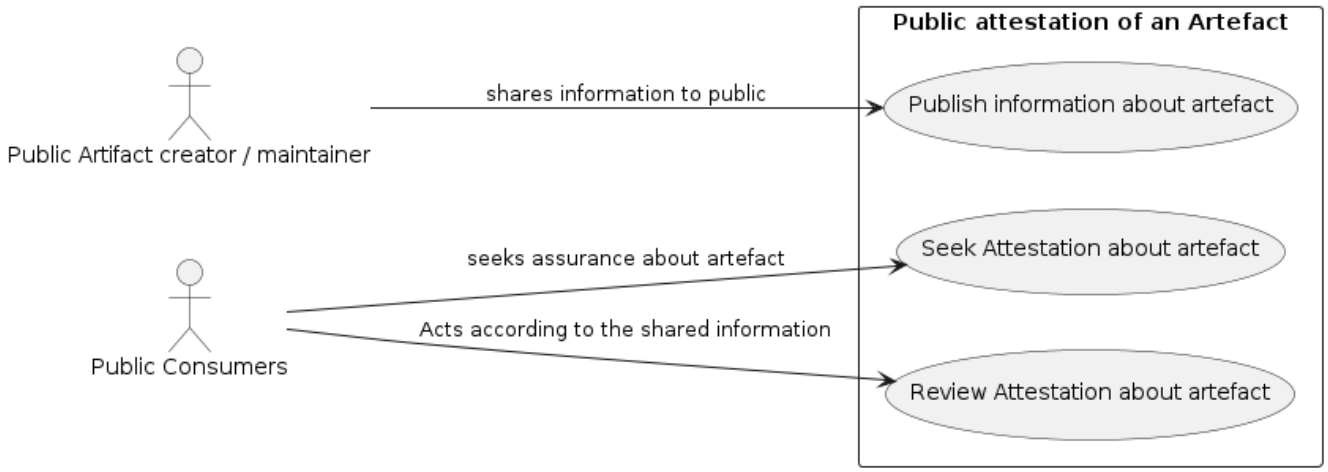
Market research has identified two primary use cases that the service must address to meet the needs of its potential users effectively:

5.1. Public Attestation of Data

This use case addresses the need for users to submit data that is then stored in a publicly accessible manner, ensuring transparency and verifiability.

Examples

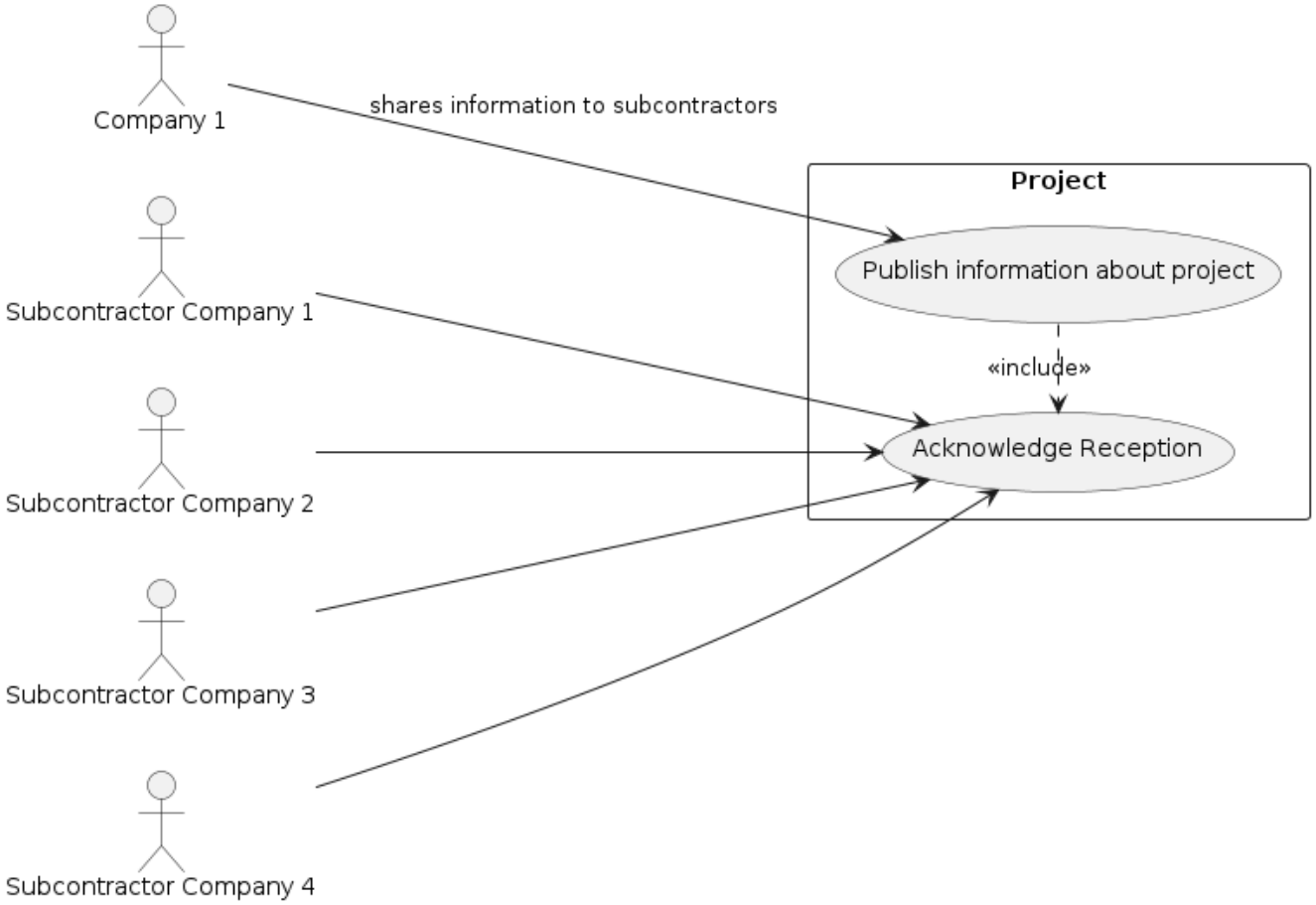
- **SBOM Usage:** The service will support the submission and public attestation of Software Bill of Materials (SBOMs), crucial for software supply chain transparency.
- **Ownership of Data such photos and videos:** Enable users to assert ownership and manage the rights of photos and videos, critical for creators and rights holders in protecting and licensing their digital content. Since the service provides timestamps, these attestations can be used in the court of law to prove ownership of the media, and the public accessibility can be used to prove the intent of any copyright infringement.



5.2. Private Sharing of Data

Catering to the secure, private data exchanges between entities in sensitive business operations:

- **Sensitive Data Exchange:** The platform will facilitate the private sharing of critical information, such as invoices or proprietary data, between companies. This feature is especially vital for manufacturing firms and their suppliers, ensuring secure and verifiable transactions.
    - **Controlled Access and Permissions:** The service will need to provide access control to the shared data, allowing the data owner to manage visibility to collaborators.
    - **Data Permanence:** The service will need to ensure that once data is shared, it is always accessible to the collaborators. This is to prevent data loss in case of a dispute, and to ensure that the data is always available for audit purposes.
    - **Bi-directional communication:** The service will need to provide a way for the collaborators to acknowledge that they have received the data. This is to ensure that the data owner knows that the data has been received and read.



6. MVP Functional requirements

FR1: User Authentication and Authorization

1. FR1.1: The system shall support OAuth for user sign up and authentication.
2. FR1.2: The system shall implement token exchange mechanisms for user login sessions.
3. FR1.3: The system shall enforce secure handling and storage of authentication tokens.

FR2: Project and Event Management

1. FR2.1: The system shall allow users to create and manage "Projects" as primary data containers.
2. FR2.2: Within each Project, the system shall enable the creation and management of "Events" to record specific occurrences or updates related to the Project.
3. FR2.4: The system shall support the classification of Projects as "Public" or "Private" to control access and visibility.
4. FR2.5: For Private Projects, the system shall allow sharing with a specified number of collaborators or subcontractors.

FR3: Relationship Management

1. FR3.1: The system shall feature a relationship manager to control interactions between users.
2. FR3.2: Users must mutually add each other as collaborators before being able to share or modify Projects and Events.
3. FR3.3: The system shall implement safeguards to prevent spamming and unauthorized sharing.

FR4: Application Integration and Automation

1. FR4.1: The system shall allow for app registrations, enabling third-party applications or services to interact with the system through APIs.
2. FR4.2: Registered apps shall receive a unique ID and secret, which can be used to authenticate and obtain access tokens.
3. FR4.3: The system shall provide extensive API documentation, including OpenAPI definitions.

FR5: Project Lifecycle Management

1. FR5.1: The system shall not allow the deletion of Projects. Instead, users can mark Projects as "Inactive" to indicate they are no longer active or being updated.
2. FR5.2: Inactive Projects shall be prevented from receiving new updates or Events, preserving the state of shared information at the time of inactivation.
3. FR5.3: Users shall have the capability to reactivate Inactive Projects, restoring the ability to update the Project and add new Events.
4. FR5.4: Projects that have been created as public or private cannot be changed to the opposite state.

FR6: Collaborator Management and Data Permanence

1. FR6.1: The system shall allow the removal of collaborators from Projects, preventing them from making further interactions or updates to the Project.
2. FR6.2: Despite the removal of a collaborator, all data shared with them up until the point of removal shall remain accessible to them, adhering to the "once shared, always shared" principle.
3. FR6.3: The system shall clearly inform users about the permanence of shared data upon adding collaborators to a Project, ensuring users understand that shared data cannot be made inaccessible to collaborators after sharing.

7. Non-functional requirements

NFR1: High Availability and Fault Tolerance

1. NFR1.1: The system shall be designed for high availability, using Kubernetes' orchestration capabilities to manage and auto-recover from failures.
2. NFR1.2: Deployments on the Google Cloud Kubernetes platform shall utilize managed services and features to ensure the system remains accessible and resilient to outages.

NFR2: Performance and Low Latency

1. NFR2.1: The Docker container shall be optimized for performance, ensuring minimal overhead and efficient use of resources to facilitate low-latency operations.
2. NFR2.2: Network configurations shall be optimized within the Kubernetes environment to support real-time interactions and data exchanges.

NFR3: Maintainability and Modularity

1. NFR3.1: The codebase shall adhere to best practices for readability, documentation, and modularity to support collaborative development and future enhancements.

NFR4: Scalability

1. NFR4.1: The system shall leverage Kubernetes' horizontal scaling capabilities to dynamically adjust resource allocation based on current demand, ensuring scalable performance under varying loads.
2. NFR4.2: The Docker images shall be designed to be lightweight and stateless where possible, facilitating quick scaling and deployment across multiple instances.

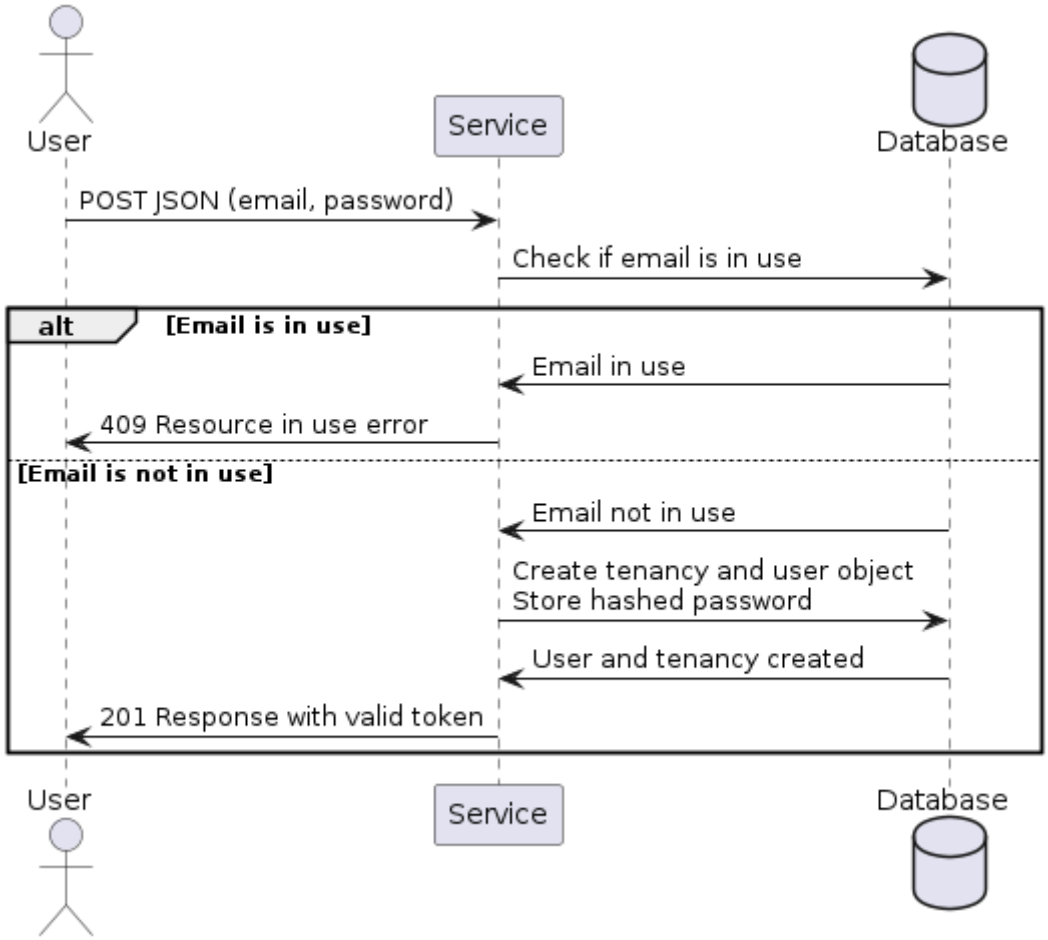
8. Design Diagrams

8.1. User / App registration Authentication and Authorization

This section details the mechanisms for securing access to the system, ensuring that only authenticated and authorized users and applications can interact with it. Utilizing JWT tokens and app registrations, the system establishes a secure environment for both users and automated applications.

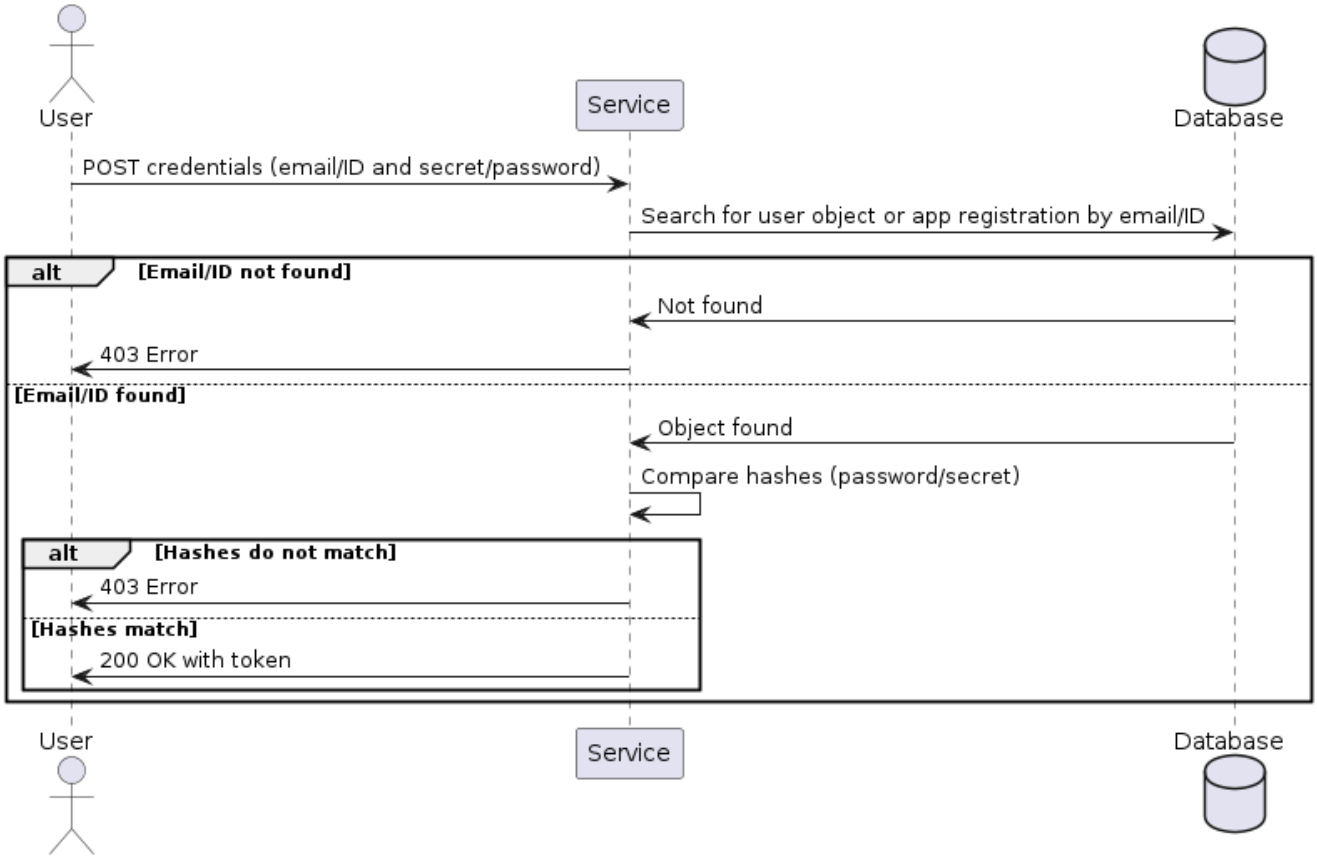
8.1.1. User Sign Up

This diagram illustrates the user signup process. A new user provides their email and password, which the system checks for uniqueness. If the email is not already in use, the system creates a new user account and tenancy, stores a hashed version of the password, and returns a JWT token to the user, signifying successful registration.



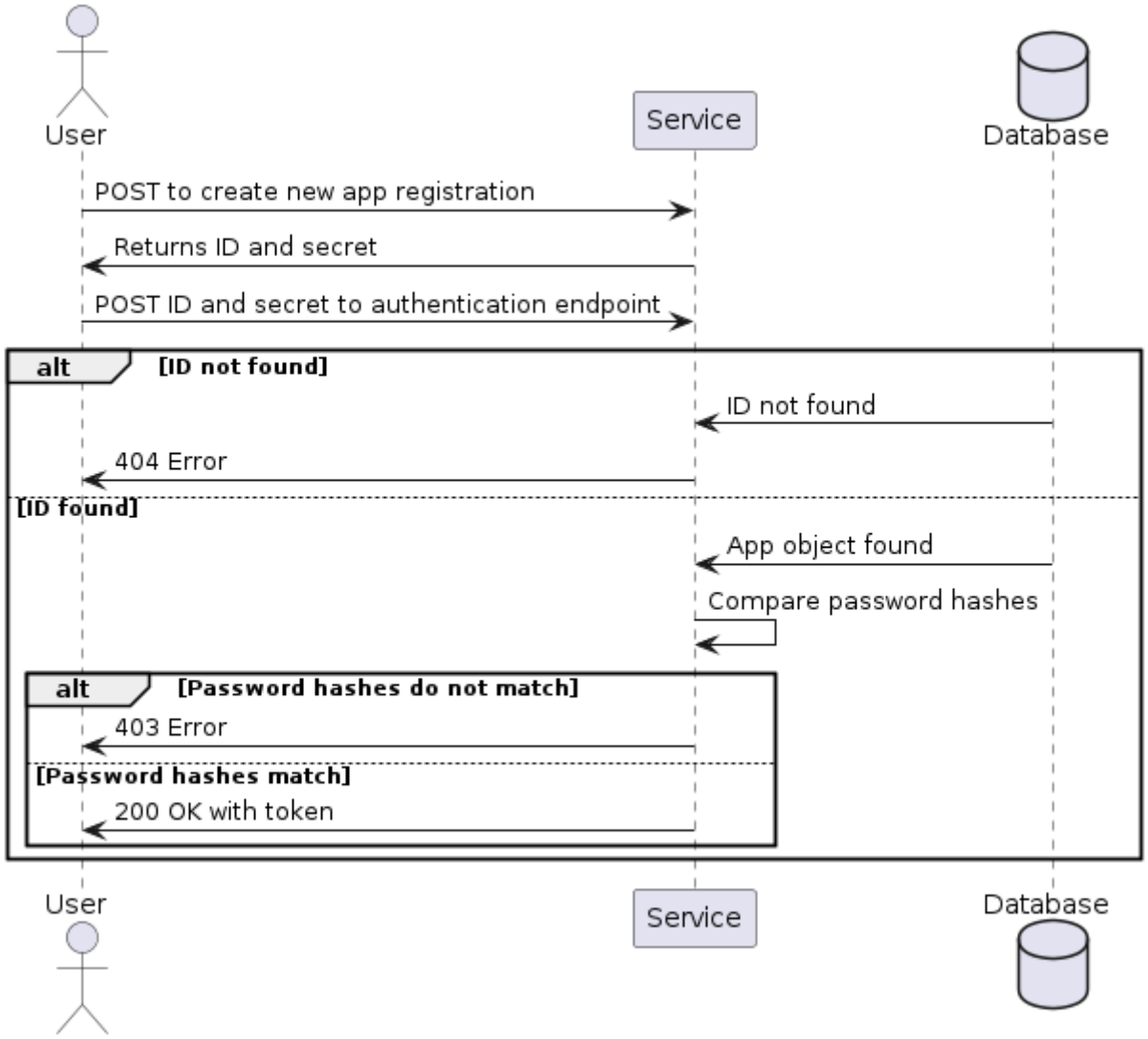
8.1.2. Authentication

This diagram illustrates the user login process. A user provides an email and password or app registration ID + secret, which the system checks against the stored hashed password. If the password / secret matches, the system returns a JWT token to the user, signifying successful login.



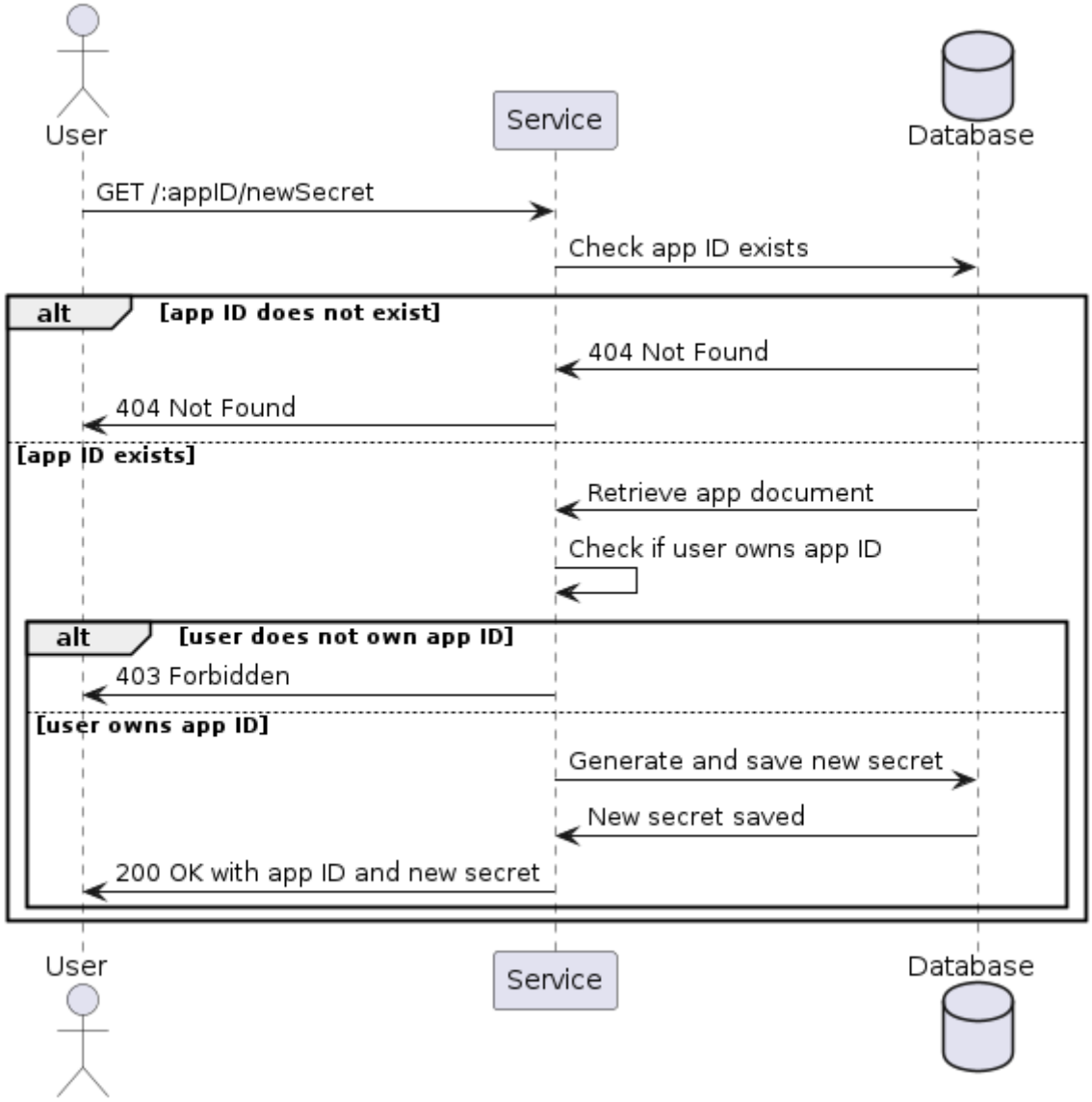
8.1.3. App Registration Creation

This depicts the app registration and authentication process. A user or developer can request a new app registration, receiving an ID and secret in return. This ID and secret are then used to authenticate the app with the system. Upon successful authentication, the app is issued a JWT token, similar to the user login process, allowing the app to interact with the system.



8.1.4. Rolling Secrets for App Registrations

This diagram illustrates the process of rolling secrets for app registrations. The system allows users to request a new secret for an existing app registration, ensuring that the app can continue to authenticate with the system. The old secret is then invalidated, preventing further use.

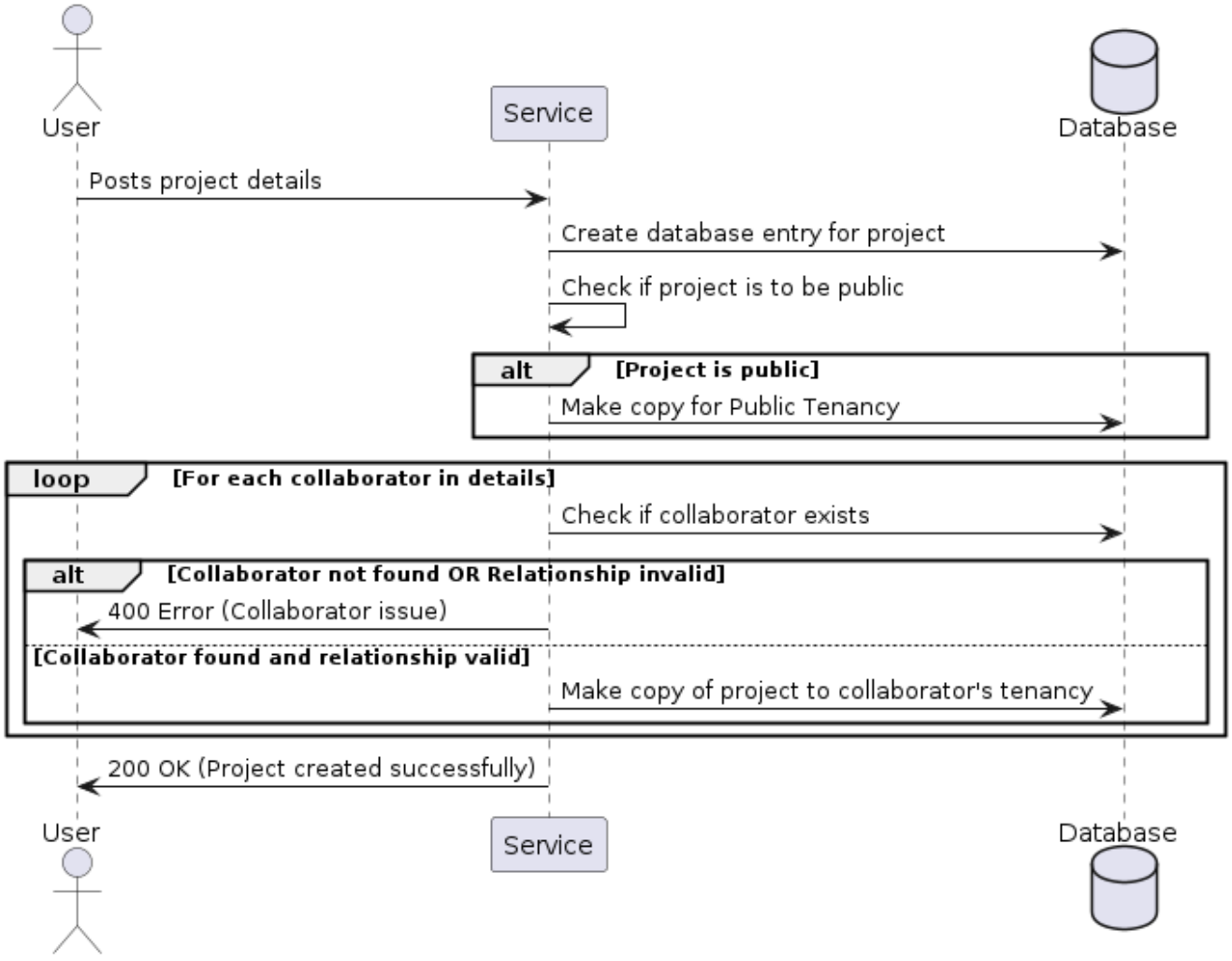


8.2. Project and Event Creation

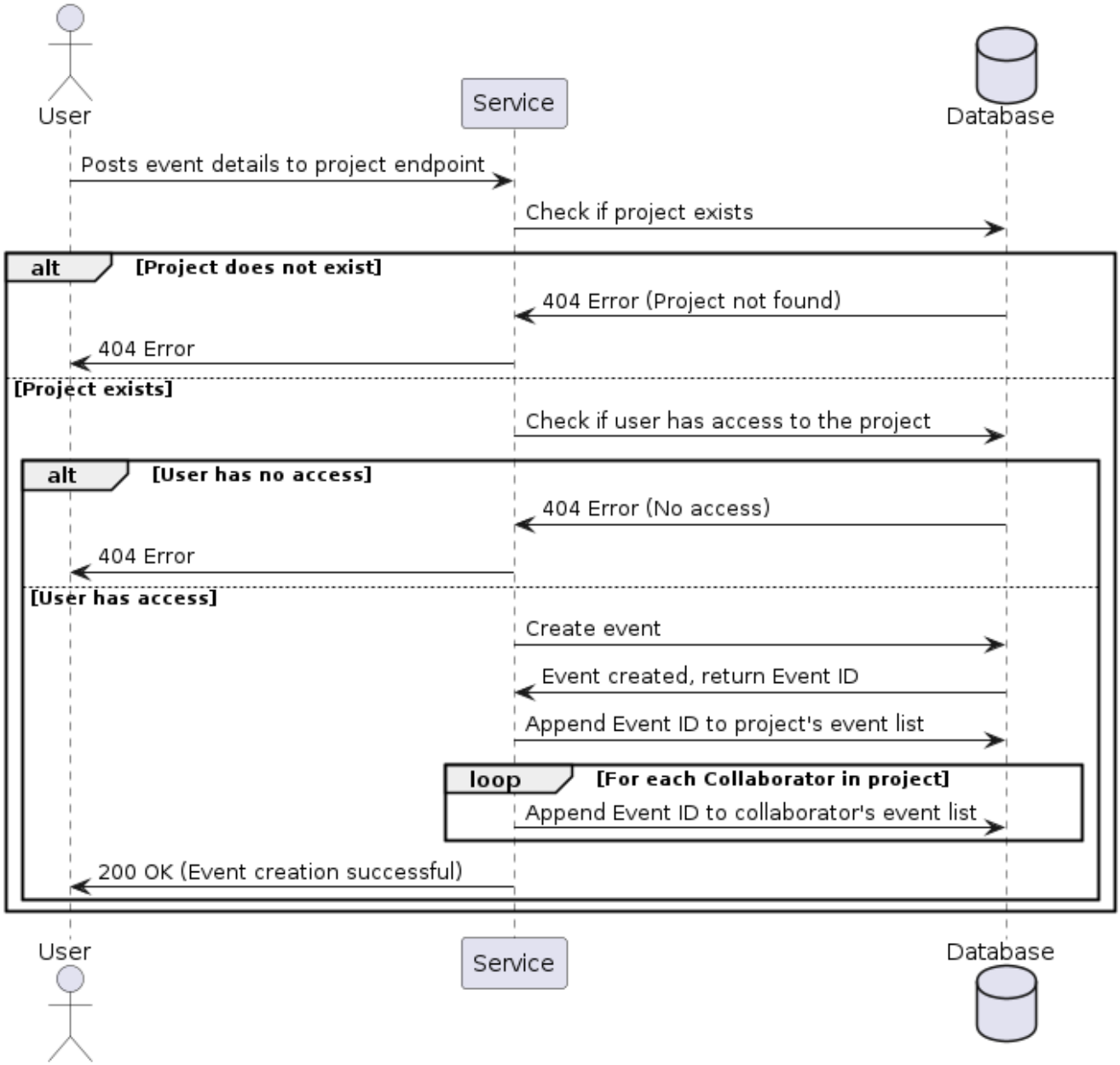
Projects serve as containers for events within the system, acting as organizational units to which changes (events) are logged. Here's an elaboration on the characteristics and functionalities related to projects and events:

- Nature of Projects: Projects can be designated as either public or private at their creation. This visibility setting determines who can access and view the project and its associated events. Once set, the visibility of a project cannot be altered, ensuring a consistent access level.
- Permanence of Projects: Projects are designed to be permanent structures within the system. They cannot be deleted to preserve the integrity and history of the work done within them. However, they can be marked as inactive, indicating that no new events will be associated with the project, but its history remains accessible. Projects can be reactivated if necessary.
- Direct Updates and Event Logging: Any direct updates made to a project, such as changing its description or adding metadata, are logged as events. This creates a transparent history of changes that anyone with access to the project can review.
- Events as Change Logs: Events within a project act as a detailed log of changes. They can include attachments and custom metadata, providing flexibility in documenting the nature and details of each change.

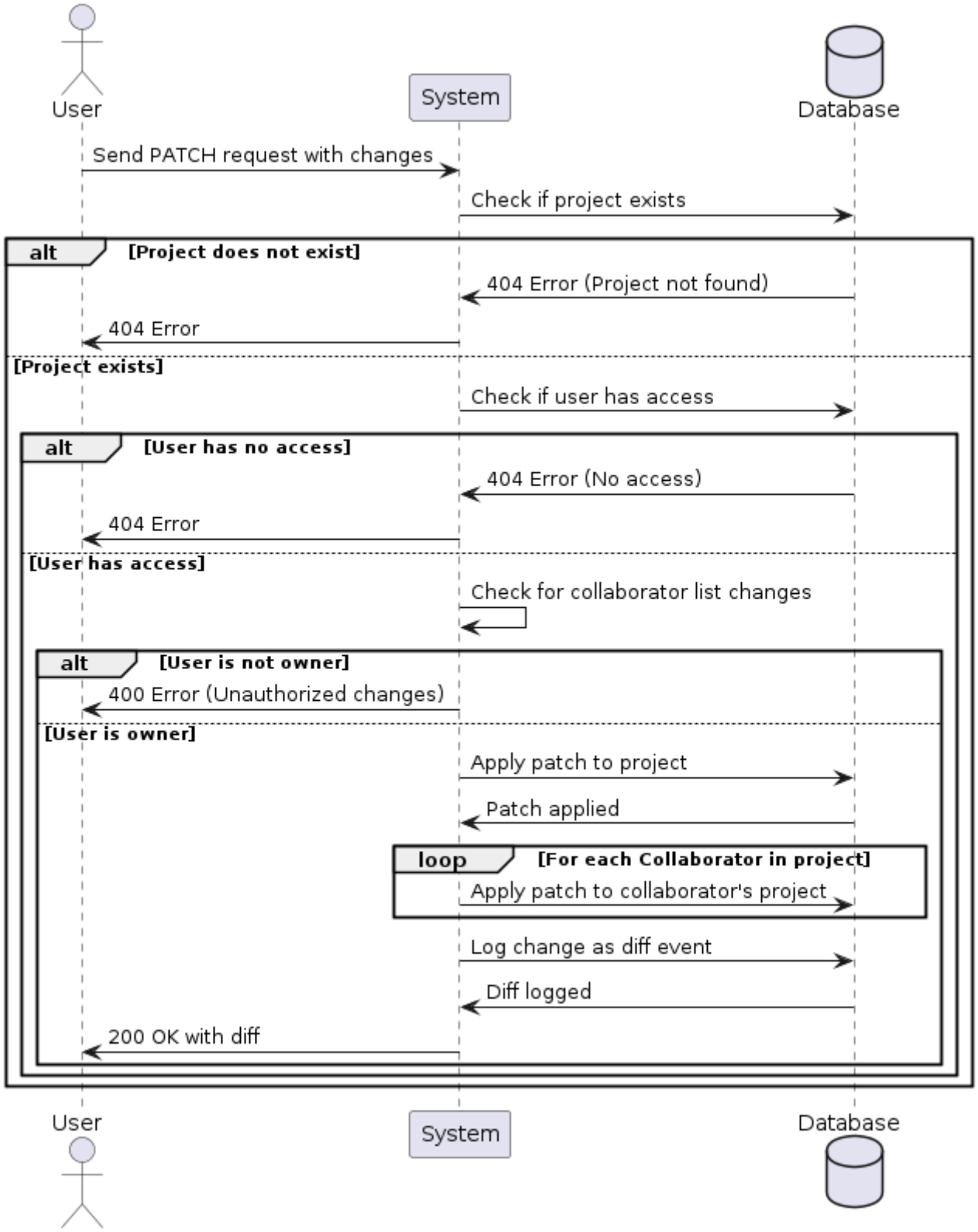
8.2.1. Project Creation



8.2.2. Event Creation



8.2.3. updating a project

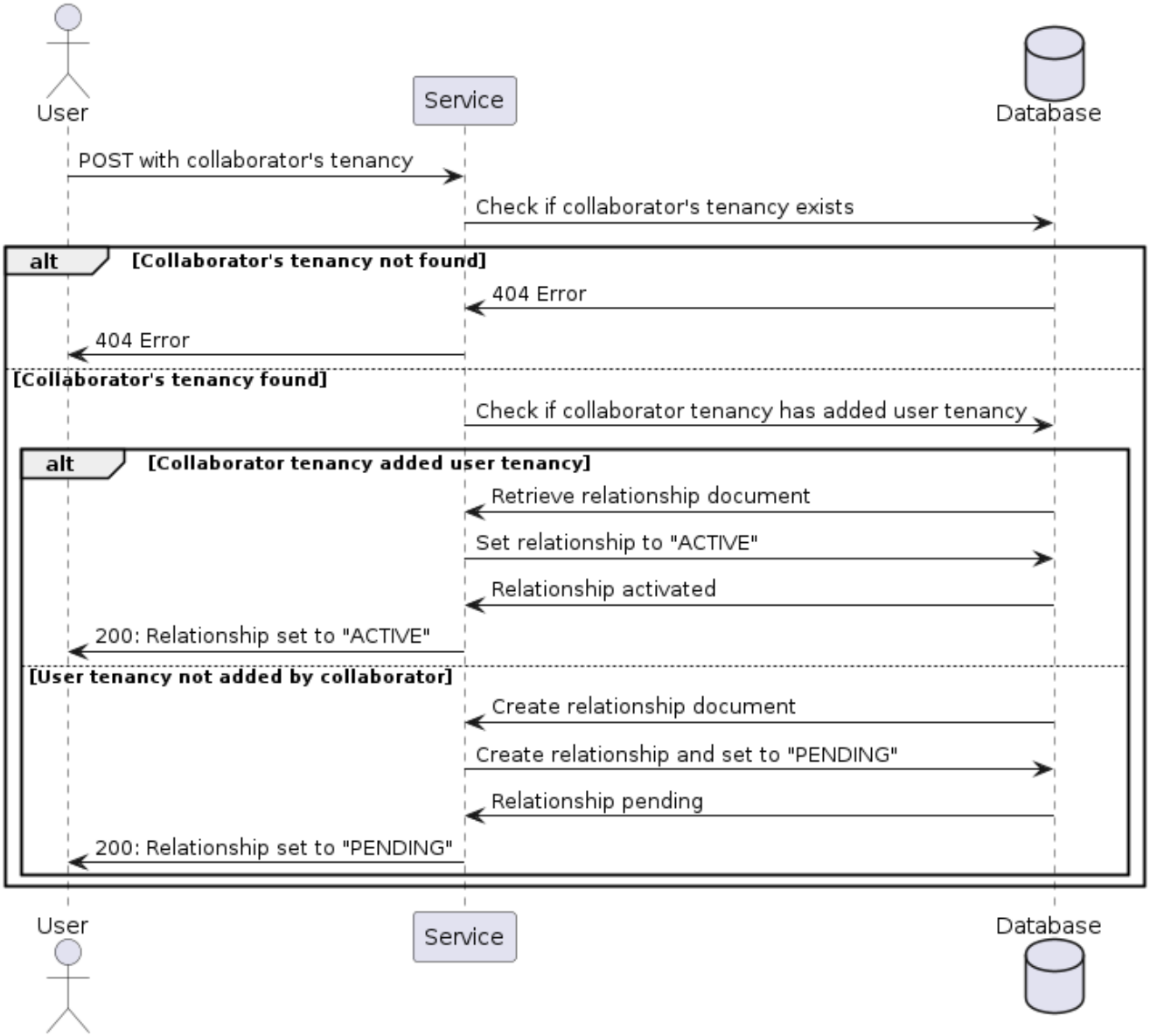


8.3. Collaborator Management

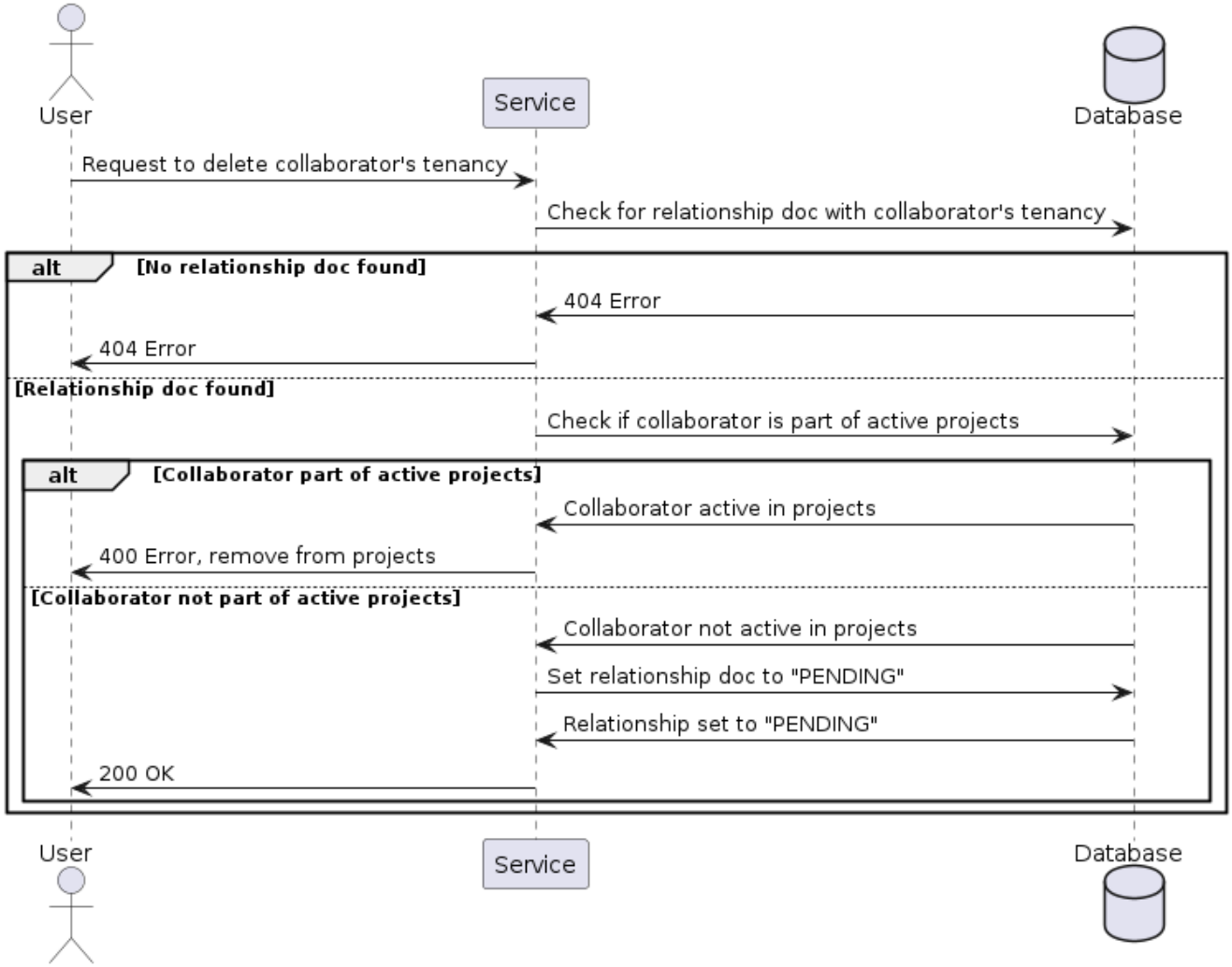
keeping in line of preventing spamming and unauthorized sharing, the system shall implement a relationship manager to control interactions between users. Users must mutually add each other as collaborators before being able to share or modify Projects and Events.

When Removing collaborators, the system needs to make sure that any shared active projects are not affected. It forces project owners to remove collaborators from all projects before they can remove them from their collaborators list. This is to ensure that there is a log of the removal actions, this covers the legal issue of when a party is trying to deny that they had access to the data.

8.3.1. Adding a collaborator



8.3.2. Removing a collaborator



9. Technologies and Tools

This section outlines the key languages, frameworks, and tools utilized in the development, documentation, and deployment of the service. These technologies were selected for their support, and ability to integrate seamlessly with each other, providing a foundation for building and maintaining the service.

9.1. Languages

- **OpenAPI** [9]: The OpenAPI Specification is a widely adopted standard for documenting APIs. It provides a language-agnostic way to describe RESTful APIs, enabling both humans and computers to understand the capabilities of a service without accessing its source code.
- **TypeScript** [10]: TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. It offers static type checking at compile time, enhancing code quality and maintainability. The static type system allowed the consumption of the OpenAPI specification using the openapi-typescript [11] tool automatically generating types and interfaces, that were then consumed by express.js [12]. This workflow ensured that the API specification was adhered to, and the code was type safe.
- **Dockerfile / Bash**
  - **Dockerfile** syntax defines the service's environment within a Docker container, including the setup of necessary software, dependencies, and runtime configurations.
  - **Bash** [13]: As the Docker container is based on Ubuntu, Bash was the default shell for executing commands and scripts, for automation of deployment and setup tasks.
- **AsciiDoc** [14]: A human-readable document format, chosen because it was semantically richer than Markdown but not as cumbersome as LATeX. AsciiDoc's flexibility and support for rich formatting met the requirements for the documentation of this project.
  - **PlantUML** [15] This tool is used to generate diagrams using a declarative DSL. It supports all the major UMl diagrams, and is used to generate the diagrams in this document.

9.2. Tools

This subsection details the array of tools that support the project's lifecycle, from development and testing to deployment and management. Selected for their robust functionality and integration capabilities, these tools form the backbone of the project's operational workflow.

- **Visual Studio Code**: IDE for development, chosen for its extensive plugin system, which significantly enhances productivity and personal experience with it.
  - The **Remote Development extension** allows for seamless development inside Docker containers, mirroring production environments closely.
  - The **Jest Extension** facilitates efficient test execution and debugging directly within the IDE, including the ability to set breakpoints and step through test and service code.
- **Docker**: The core of the project's deployment strategy, Docker containers encapsulate the service environment, ensuring consistency across all stages of deployment.

    - The service's deployment utilizes Kubernetes for orchestration, managing the containerized service across a cluster for high availability and scalability.
- **K9s**: An essential tool for Kubernetes cluster management, K9s offers a terminal-based interface that simplifies monitoring and debugging tasks.
    - Its use in the project streamlines the observation and management of service deployments, enhancing the efficiency of operational tasks.
- **GitHub Actions**: The platform of choice for automating CI/CD workflows, GitHub Actions facilitates the continuous integration and delivery of the service.
    - Configured pipelines automate the processes of building, testing, and deploying code changes, ensuring that the service remains up-to-date and stable.
- **GitHub Projects**: Implements an agile workflow within GitHub, providing a Kanban board for task management.
    - This tool aids in organizing the development process, tracking progress, and prioritizing work effectively.
- **Mongo-memory-server**: An in-memory database server for MongoDB, enabling rapid and isolated testing environments.
    - Its integration into the testing workflow allows for efficient execution of database-related tests without persisting data, speeding up the test cycles.
- **Jest**: The testing framework selected for its simplicity and compatibility with TypeScript, offering a wide range of testing capabilities from unit to integration tests.
    - Used in conjunction with **supertest** for comprehensive API testing, ensuring that service endpoints function as expected.
- **Git**: The version control system underpinning the project, enabling collaborative development and feature branching.
    - Alongside Git, **GitKraken** is used for its graphical interface, simplifying the visualization of git history and branch management, enhancing the team's version control practices.

10. Architecture and Database Design

This section delineates the structural design of the service and its underlying database. Emphasis is placed on scalability, efficiency, and maintainability.

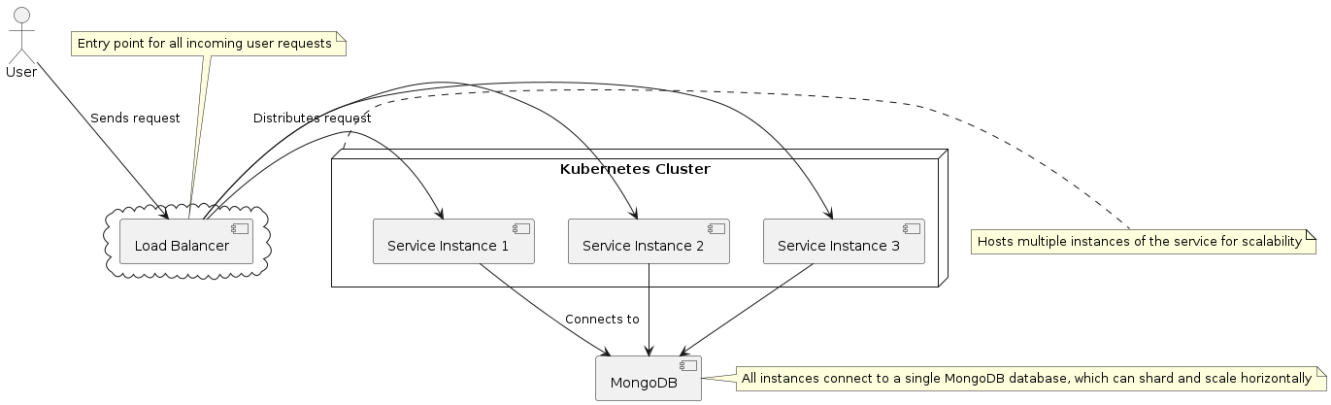
10.1. Architecture

The service adopts a multi-instance monolithic architecture, characterized by:

- A singular codebase that constitutes the monolith, All of the code is present in a single repository, and gets built and deployed as a single unit. This approach simplifies development and deployment processes, allowing for rapid iteration and feature addition.
- Google cloud App Engine is selected as the deployment platform, providing a managed environment for running the service. This platform abstracts away infrastructure management, which is important for the time constraints of the project.
- Horizontal scalability is a key feature, achieved through the deployment of multiple service instances behind a load balancer. This approach allows the service to accommodate varying loads by adjusting the number of active instances, as well as providing redundancy and fault tolerance.
- MongoDB is selected as the database solution, since all of the metadata is formatted as JSON, A scalable document-oriented database was chosen to store the service's data.

Extended Requirements

- If the project would progress beyond the MVP, google cloud storage would be selected as the solution for attachments and rich metadata. As the service is already deployed on Google Cloud, keeping within the same ecosystem would simplify the management of the service.
- To enable the cryptographic assurance mechanisms of the SCITT framework, the service would also need to integrate message queues. This is because the cryptographic processes are computationally expensive and would the impact latency of responses, which would be unacceptable for the service's core functionality. By offloading these processes to a message queue, the service can maintain low latency while ensuring the cryptographic assurance mechanisms are eventually applied, in accordance of an SLA.



10.2. Database Design

The database design focuses on cost efficiency and performance:

- An emphasis on using references and arrays within database objects to minimize operational costs. Arrays are especially advantageous for storing time series data, such as events and diffs, because they inherently support efficient sorting and filtering without additional cost.
- The database schema is designed to be flat, avoiding nested objects to simplify data access patterns and improve performance.
- Mongoose, an Object Data Modeling (ODM) library for MongoDB, is utilized to define and enforce the schema of database objects. It provides a rich set of features for schema validation, query building, and business logic implementation.
- Functions attached to database objects play a critical role in implementing business logic and validation directly within the database layer. This encapsulation of functionality ensures that data integrity and business rules are consistently enforced across the application.

The architecture and database design collectively support the service's operational and developmental requirements, ensuring that the system remains scalable, efficient, and easy to maintain as it evolves.

11. development process

- Generally followed kanban and agile development process. updated solutions and requirements as the project progressed, with regards to skills and time constraints.

11.1. Planning

In a typical project, the development process begins with market research and the delineation of requirements use cases, which inform the initial idea.

- Subsequently, for each requirement identified and solution proposed, issues are created on the kanban board. These issues are then broken down into smaller, actionable tasks, which are assigned to different team members. This methodical breakdown ensures clarity of responsibility and facilitates efficient tracking of progress.
- Additionally, the kanban board serves as a dynamic roadmap, guiding the project's feature development. It is complemented by detailed task management tools, which allow for the granulation of tasks and assignment to specific team members. This approach ensures that all aspects of the project are systematically addressed and progress is transparently tracked.

However, given that I was the sole contributor to this project:

- The kanban board was adapted to serve as a rough guide for feature development rather than as a detailed task manager. This was due to the absence of a team to assign tasks to, which altered the utility of the kanban board from a collaborative tool to a personal project roadmap.
- Instead of breaking down issues into smaller tasks within the kanban board, I managed the finer details directly within the code base using TODO comments. This approach allowed for a more fluid and flexible development process, suited to a solo project environment where the developer is fully aware of all aspects of the project.

11.2. Development and Testing

This section describes the development and testing phase, emphasizing the structured approach taken to ensure functionality and stability.

- **Test-Driven Development**: The project utilized a test-driven development approach, starting with the creation of end-to-end test stubs based on the OpenAPI specification and project requirements. This guided the subsequent development of service code and database models to fulfill these test conditions.
- **Integration of OpenAPI and TypeScript**: Productivity was enhanced by using the `openapi-typescript` tool, which generated TypeScript types and interfaces directly from the OpenAPI specification. This ensured adherence to the API specification and maintained type safety across the codebase.
- **Execution of Tests**: After developing service code and database models, the focus shifted to completing and running the tests. This step verified that the service met all specified requirements. Following successful end-to-end tests, further unit tests for database models were conducted to cover additional scenarios and maintain test coverage at or above the target

threshold.

- **Use of mongo-memory-server and Jest**: The project employed `mongo-memory-server` in conjunction with Jest for testing, creating an efficient feedback loop. This setup allowed for rapid testing without the need for service deployment or data mocking, enabling a swift execution of both end-to-end and database model tests.
- **Coverage Analysis**: Jest's built-in tool was used for measuring test coverage, with a target set to ensure the code was extensively tested. This approach helped identify untested paths and contributed to the overall reliability of the service.

The development and testing stages were marked by a systematic approach to coding and testing, facilitated by specific tools and practices designed to optimize the workflow and ensure a high level of code quality and service reliability.

11.3. Deployment

- The service was deployed on a Kubernetes cluster on the Google Cloud Platform
- Master branch was used for deployment, with the CI/CD pipeline being triggered on each merge to master

12. Evaluation of Product and Future Work

The development of the service aimed to fulfill the minimum viable product (MVP) criteria and was successfully deployed. However, to enhance its functionality and competitive edge in the market, several user interface features and technical enhancements are planned. These additions are crucial for improving user experience, meeting real-world organizational needs, and ensuring the service's robustness and security.

12.1. Functional enhancements

12.1.1. User Interface Enhancements

The introduction of a web interface is a primary objective for future development. A graphical user interface will significantly improve the management of relationships and projects, offering users comprehensive overviews of their data. This feature is essential for a more intuitive and efficient user interaction with the service.

File handling capabilities, including upload and download functions, need to be integrated. The current implementation requires users to manually calculate and submit file hashes, a process that is not user-friendly and could deter potential users. Automating this process will streamline user operations and align the service with market standards.

Notifications and alerts represent another area for improvement. Implementing notifications for key events such as the addition or removal of collaborators, changes in project status, and updates to projects will enhance user engagement and project management.

12.1.2. User Management and Permissions

Expanding the service to support multiple users with varying permission levels is critical. The current model, which supports only a single user and their app registrations, does not adequately cater to organizational structures that typically comprise multiple users. Developing a more sophisticated user management system will allow the service to better serve real-world business and organizational needs.

12.1.3. Cryptographic Assurance Mechanisms

Implementing cryptographic assurance mechanisms based on the SCITT framework is crucial. The current MVP lacks the legal robustness required for court scrutiny, a significant disadvantage against competitors. Developing and integrating these mechanisms will provide the necessary legal and security assurances for users, making the service a viable option for businesses requiring transparent and trustworthy data provenance.

12.2. Technical Enhancements

On the technical front, the service's scalability and fault tolerance need to be rigorously tested. Planned testing includes:

- Load testing to assess the service's performance under various stress conditions and ensure it can handle anticipated user loads.
- Fault injection testing to identify potential points of failure and verify the system's response to and recovery from faults.
- Security testing to uncover vulnerabilities and address them, thereby strengthening the service's defense against attacks and unauthorized access.

These tests are imperative for validating the service's reliability and security, ensuring it meets the high standards expected by users and the industry.

The roadmap for future work is designed to address current gaps in the service's offerings and technical foundations. By focusing on these areas, the service aims to enhance user experience, meet the complex needs of organizations, and ensure a secure, scalable, and robust platform.

[1] SCITT (2024). "Introduction to SCITT." Retrieved from https://scitt.io/

[2] SCITT (2024). "About SCITT." Retrieved from https://datatracker.ietf.org/group/scitt/about/

[3] Data Trails AI (2021). "Sellafield Ltd: Streamlining Nuclear Waste Handling." Retrieved from https://www.datatrails.ai/case_study/sellafield-ltd-streamlining-nuclear-waste-handling/

[4] NTIA (2024). "Software Bill of Materials." Retrieved from https://www.ntia.gov/sbom

[5] Why Pope Francis Is the Star of A.I.-Generated Photos https://www.nytimes.com/2023/04/08/technology/ai-photos-pope-francis.html

[6] Adobe (2024). "C2PA Explainer." Retrieved from https://c2pa.org/specifications/specifications/1.3/explainer/Explainer.html

[7] Google (2024). "Certificate Transparency." Retrieved from https://certificate.transparency.dev/

[8] Microsoft (2024). "Explainable AI." Retrieved from https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/innovate/best-practices/trusted-ai

[9] https://www.openapis.org/

[10] https://www.typescriptlang.org/

[11] https://www.npmjs.com/package/openapi-typescript

[12] https://expressjs.com/

[13] https://www.gnu.org/software/bash/

[14] https://docs.asciidoctor.org/asciidoc/latest/

[15] http://plantuml.com/