Electronic Declaration System (EDS)

CSCI 409 Senior Project II

Nazarbayev University

Group Members:

Azat Akash
Nurlan Serikov
Sultan Turan
Nurtore Onggarbay

Adviser(s):

Askar Boranbayev
Kuanysh Yersakhanov

Spring 2025

Contents

1 Executive Summary

Historically, the control of Conflict of Interest (CoI) declarations at Nazarbayev University (NU) depended on inefficient manual processes characterized by lack of transparency and high compliance risks. The major objective of the **Electronic Declaration System (EDS)** project was to overcome the above challenges by creating a web-based software program capable of automating and enhancing the life-cycle of the CoI declaration process and thus enabling better compliance control and more transparency.

The Electronic Declaration System (EDS) was built based on a modern technological stack, using Java Spring Boot for the backend application programming interface, React (Next) for front-end development, PostgreSQL for data handling, and MinIO for object storage. The system defines separate functionality for **Declarants**, who submit Initial and Ad hoc declarations and acknowledge Management Plans; **Managers** from the NU Compliance Office, whose role is to review declarations, disclose conflicts, and create Management Plans; and **Administrators/Super-Administrator**, IT department staff who are responsible for the system's maintenance, user control, and management [3].

The current project resulted in the successful development of a working EDS platform, where users can submit, track, and evaluate CoI statements. It also equips Managers with the ability to handle possible conflicts through formal Management Plans, and Administrators to preserve the integrity of the system. This project demonstrates the successful design, implementation, and evaluation of an applications-based solution to the recognized problem of inadequate CoI management, in conformity with the requirements of the course.

2 Introduction

2.1 Problem Description

The approach used by Nazarbayev University for the management of Conflict of Interest (CoI) statements has been described as *manual* [1]. This approach comes with many drawbacks, including a significant administrative workload, potential delays in the evaluation and resolution processes, increased risks of non-compliance with institutional policy, and lack of transparency and efficient tracking mechanisms [2]. It has been challenging to follow the NU Policy and Procedure on Conflict of Interest using this approach [1].

Nazarbayev University's approach to managing Conflict of Interest (CoI) statements has been described as *manual* [1]. This approach was subjected to a number of drawbacks, such as administrative costs, the potential for delays in the evaluation and resolution processes, the increased likelihood of a failure to comply with the NU policy, and a lack of adequate transparency along with weak systems of monitoring [2]. As such, maintaining compliance with the NU Policy and Procedure on Conflict of Interest was challenging [1].

2.2 Motivation and Significance

The need to modernize and improve the management process of Conflict of Interest (CoI) in NU was the driving force in the development of the **Electronic Declaration System (EDS)**, aiming to attain complete compliance to the CoI Policy of the university [2]. The benefits of the use of *Digitalization* include more efficient workflow due to automation, improved transparency for the declarants and the managers, better audit trails to monitor compliance, and minimized administrative workload for manual processing [2].

2.3 Proposed Solution Overview

The proposed solution includes the **EDS**, a dedicated web-based application, which was designed to manage the whole life-cycle of Conflict of Interest (CoI) statements [2]. The system's major goal is to automate the submission, evaluation, and resolution tasks of potential conflicts, classified into the major categories of statements known as *Initial Statements* (submitted at the time of hiring or the system's startup), *Ad hoc Statements* (arising from new conflicts or modifications), and *Management Plans* (prepared by Managers to handle known conflicts) [2]. The system provides role-specific access for **Declarants**, **Managers**, and **Administrators**, each of whom receives unique personalized dashboards to carry out the assigned tasks efficiently [3].

3 Background and Related Work

Conflict of Interest (CoI) management is a critical aspect of maintaining integrity and ethical standards within organizations, particularly in academic and research institutions like Nazarbayev University. Traditional approaches often involve manual paperwork, generic electronic forms (like PDFs or Word documents), or sometimes email-based submissions. While functional to a degree, these methods often suffer from inefficiencies, lack of centralized tracking, difficulty in generating compliance reports, and potential for human error or delays. NU's own *Policy and Procedure on Conflict of Interest* served as a primary driver for developing a more robust system [2].

Management of Conflict of Interest (CoI) is a key element in maintaining integrity and ethical practices in institutions such as academic and research-based universities like Nazarbayev University. Traditional methods used usually depend on manual documentation, standardized electronic forms (which could be in the format of PDF or Word documents), or, in some cases, email submission. While such methods could work to some extent, they often come with inefficiencies, inadequate centralized control, difficulties in the generation of compliance reports, and the possibility of human error or delay. Nazarbayev University's *Policy and Procedure on Conflict of Interest* has operated as a major driver for the development of a more unified system [2].

There are some generic, free form-building software or basic workflow tools exist, they often lack CoI management-specific features. These features involve tailored review processes, conditional logic based on the contents of the declarations, dedicated tools for the development of management plans, and accurate tracking of acknowledgements. At the outset of the project, research into particular, rival commercial or open-source CoI management systems for academic institutions was limited, and it may reflect the existence of a market gap or the prevalence of customized solutions in similar organisations.

The existing manual or semi-automated processes at NU have been identified as lacking in a number of key areas, such as high *auditability*, *workflow adaptability*, and proactive *notification* features [2]. The creation of a custom web application, like EDS, based on Java Spring Boot and React.js, was considered the most suitable solution. The solution provides a tailored workflow that matches NU's specific requirements regarding the Conflict of Interest (CoI) policy, delivers the necessary functionality for efficient management by the Conflict of Interest, Ethics, and Compliance Office (Managers), ensures data security, and allows for scalability to meet potential future demand increases [2, 3]. The chosen technology stack provides an optimal balance between performance needs, development efficiency, and potential future integrations' compatibility [3].

4 Project Approach

4.1 Detailed Solution Description

A web application called the **Electronic Declaration System (EDS)** was created to automate and oversee Nazarbayev University's Conflict of Interest declaration procedure.

4.1.1 Software/Hardware Architecture

A multi-tier architecture is used by the system [1]. An Nginx load balancer routes user requests from the React.js (Next.js) frontend to one of multiple backend API nodes constructed with Java Spring Boot [1]. In order to store user information, declaration specifics, and workflow status, these backend nodes manage the business logic and communicate with a PostgreSQL relational database [1]. A MinIO S3-compatible object storage solution is used to manage declaration-related document attachments [1]. HTTP/HTTPS is the main protocol used for component-to-component communication [2]. A dual-core CPU, 8GB RAM, and 50GB SSD storage running Linux OS were among the suggested server requirements [1].

4.1.2 Algorithms/Workflows

The declaration lifecycle is at the center of the core workflow [2]. A Declarant uses the web interface to submit an Initial or Ad Hoc declaration. The declaration is saved by the system, and the manager or managers are notified. The submission is examined by the manager. The declaration is marked as resolved if no conflicts are found. The manager drafts a management plan outlining steps to mitigate any potential conflicts. The Declarant must review and approve the plan after receiving notification from the system. As this process progresses, the system keeps an audit trail [2].

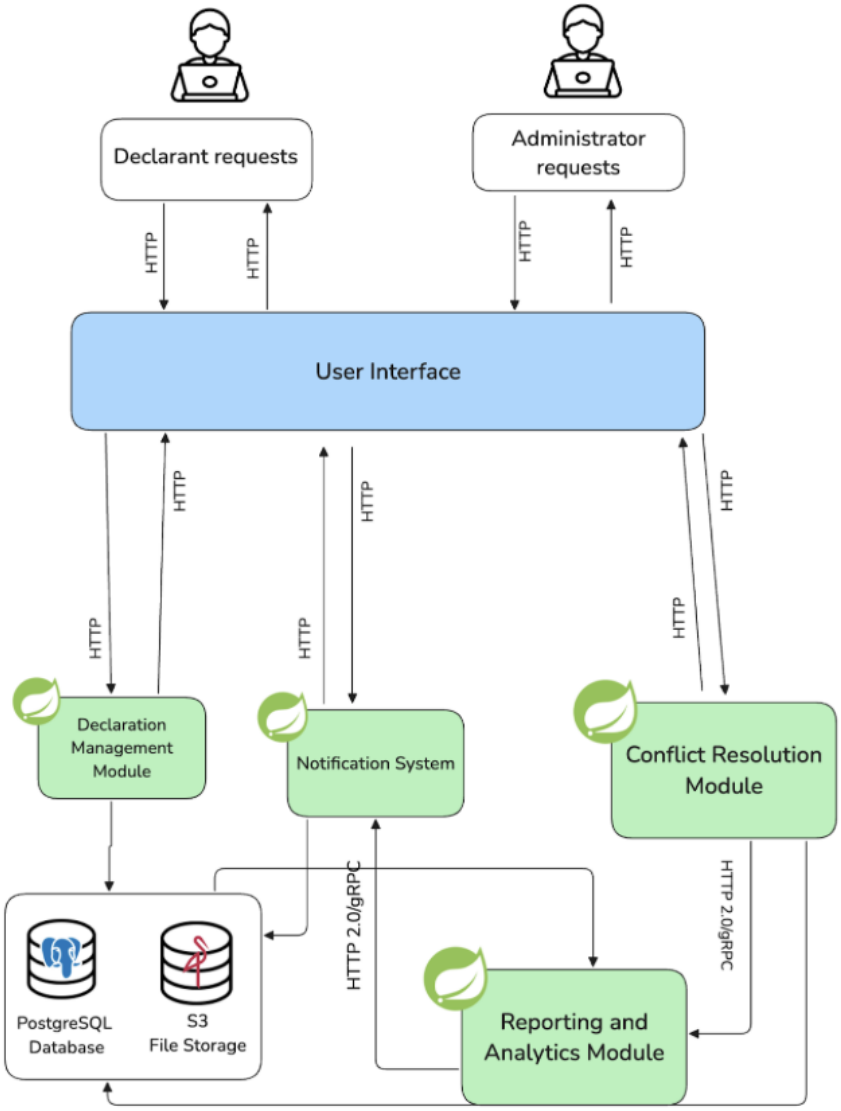

Figure 1: EDS System Architecture Diagram

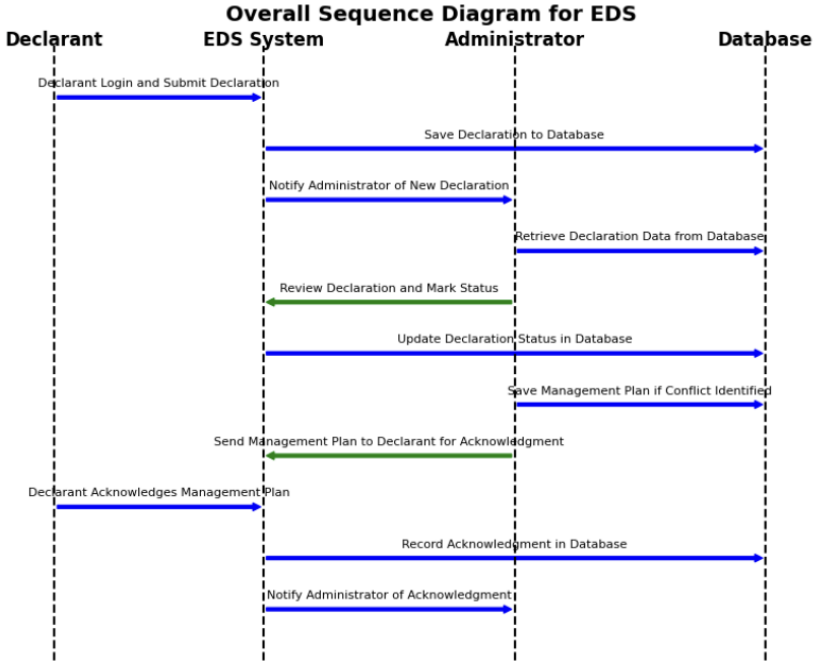

Figure 2: EDS Sequence Diagram

4.1.3 Roles

Three main user roles are defined by the system [3]:

- **Declarant:** NU employees must submit CoI declarations. They can submit *Initial* or *Ad hoc* declarations, view declaration status, review and accept *Management Plans* [2, 3].
- **Manager:** Employees of the *Conflict of Interest, Ethics, and Compliance Office*. In charge of examining declarations that have been submitted, determining possible conflicts, developing and overseeing *Management Plans*, monitoring dashboards, and producing reports pertaining to compliance with the CoI [2, 3].
- **Administrator:** *NU IT department* employees. Responsible for overall system management, which includes user account creation/management (particularly for Managers), monitoring system health/performance and system statistics, and performing technical maintenance and backups.
- **Super-Administrator:** *NU IT department* employee. Same as Administrator, but can create other Administrators.

4.1.4 Features

Key features of the EDS [2, 3]:

- **User Management:** Role-based access control (RBAC), profile management, login/registration (Declarants), and Administrator tools for managing manager accounts.
- **Declaration Management:** *Initial* and *Ad hoc* declaration submission, declarant status monitoring.
- **Conflict Review & Management Plans:** Tools for Managers to review declarations, spot conflicts, and develop and oversee plans for resolving them. Also keeping track of acknowledgements from declarants.
- **Notification System:** Automated email or in-app notifications for reminders, status updates, submissions, and necessary actions.

- **Dashboards:** Separate dashboards for Declarants, Managers, and Administrators.
- **Data Analytics and Reporting:** Managers can export data and generate basic reports on compliance and trends. System-level analytics are only available to administrators.
- **Audit Trail:** Comprehensive logging system of significant actions for compliance and accountability checks across all roles.

4.1.5 Tools

In this project, following primary tools and technologies were utilized [1, 2]:

- **Backend:** Java, Spring Boot (including Spring Data JPA, Spring Security).
- **Frontend:** React.js (using Next.js framework, TypeScript, Axios, Tanstack Query, Zod for validation, shadcn/ui for components).
- **Database:** PostgreSQL.
- **File Storage:** MinIO (S3-compatible).
- **Web Server/Load Balancer:** Nginx.
- **Version Control:** Git.
- **Project Management:** Jira.

4.1.6 Implementation Highlights

The codebase [5, 6] reflects the chosen architecture and technologies. Key implementation aspects include:

- **Backend (Java Spring Boot) [5]:**
  - *Dependencies:* Managed using Maven (`pom.xml`).
  - *Configuration:* Using `application.yaml` for storing database connection and JWT secrets. Docker Compose (`docker-compose.yml`) is used for database setup.
  - *Entities:* Core entities include `User.java`, `InitialDeclaration.java`, `UserInitialDeclaration.java`, `UserAdHocDeclare.java`, and `UserManagementPlan.java`.
  - *Security:* Spring Security is configured in `SecurityConfig.java`, and JWT tokens are used for authentication.
  - *API Endpoints:* Controllers like `AuthenticationController.java`, `UserInitialDeclarationController.ja` and `ManagementPlanActionController.java` define the REST endpoints using standard Spring annotations.
  - *Data Handling:* Repositories (e.g., `UserRepository.java`, `InitialDeclarationRepository.java`) extend JpaRepository. A custom `JsonConverter.java` handles storing multilingual text fields as JSON in the database.
- **Frontend (React/Next.js) [6]:**
  - *Dependencies:* Managed via npm (`package.json`), including React, Next.js, Axios for API calls, Tanstack Query for state management, Zod for schema validation, shadcn/ui components, and Tailwind CSS.
  - *API Client:* A custom client (`client.ts`) uses Axios and Fetch API, handling JWT token injection and refresh logic. Specific API interaction functions are organized (e.g., `login.ts`, `getUserProfile.ts`, `submitDeclarationAnswers.ts`).
  - *State Management:* Tanstack Query (`@tanstack/react-query`) is used for server state management, caching, and background updates (e.g., in `useUser.ts`, `useUserInitialDeclarations.ts`).
  - *Form Handling:* React Hook Form (`react-hook-form`) is used for managing form state, likely integrated with Zod schemas (`declarationAnswersSchema.ts`) via `@hookform/resolvers` for validation.

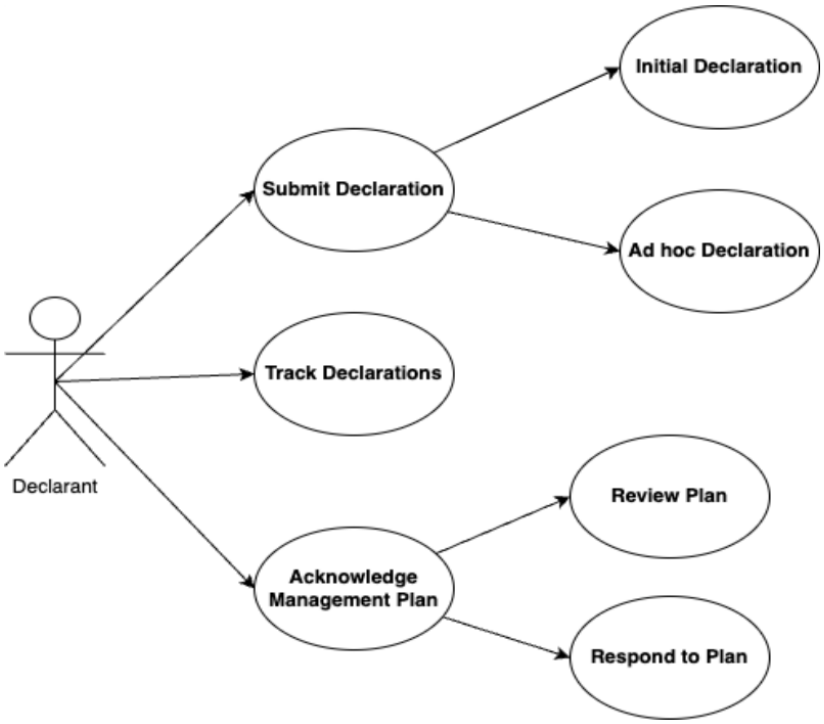

Figure 3: Declarant Use Cases

  - *UI Components:* Built using shadcn/ui library, leveraging Radix UI primitives and Tailwind CSS for styling.

- **Development Status:** The project is currently under development. Some features, particularly around error handling, logging, testing, and specific workflow details (like round-robin assignment), are noted as future tasks in the backend `README.md` [5].

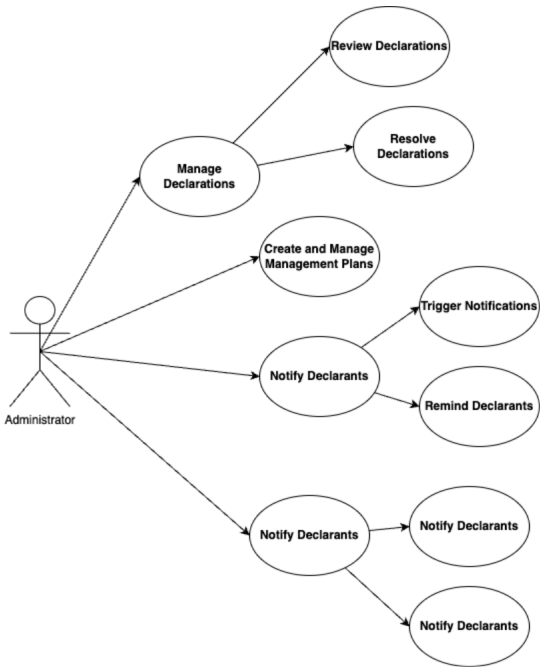

Figure 4: Administrator Use Cases

4.1.7 Use Cases

Main use cases of the system are [2]:

- Declarant can submit an Initial Declaration.
- Declarant can submit an Ad hoc Declaration.
- Manager can review a Declaration.
- Manager can create a Management Plan.
- Declarant can acknowledge a Management Plan.
- Administrator can create/manage user accounts.
- Administrator can view system statistics.

4.2 Third-Party Components

While developing EDS project our team used some third-party software components/libraries/frameworks [1, 2]:

- **Spring Boot:** Powerful framework for Java for building the robust and scalable RESTful backend API [8].
- **React.js (with Next.js):** A JavaScript library/framework used to build the dynamic and interactive UI for the web application.
- **PostgreSQL:** An open-source relational DMBS, used for persistent storage of system/application data [9].
- **MinIO:** Object storage server used for storing file attachments [10].
- **Nginx:** Used for hosting the server and load balancing, increasing the performance and reliability of the EDS [11].

4.3 Team Functioning

The project team was structured to effectively tackle the development of the EDS. Responsibilities were divided based on expertise and interest:

Nurtore Onggarbay handled DevOps tasks, including hosting in development and production stages, documentation (report), Project Management and communication with stakeholders;

Nurlan Serikov was engaged in Frontend development, UI design and testing.

Sultan Turan contributed most to Frontend/UI development;

Azat Akash managed the PostgreSQL database design, contributed the most of the Backend development, and shared Project Management responsibilities [3].

Communication and coordination were managed using **Telegram** messenger, **Jira/Trello** for task tracking [7] and **Git** for collaborative code management [2, 3].

5 Project Execution

5.1 Project Course

Two academic semesters were dedicated to the **Electronic Declaration System (EDS)** project. Project planning, requirements gathering, system design, and documentation were the main focus of the **Fall 2024** semester [1]. Stakeholder definition, functional and non-functional requirements, use case definition, preliminary architecture design, and initial task planning were all part of this [2, 1]. The implementation phase consisted of database schema finalization, backend API development, frontend interface creation, component integration, system testing, and deployment readiness during the **Spring 2025** semester [1].

5.2 Decisions and Changes

Important choices during the project centered on the technology stack (*Spring Boot, React, PostgreSQL, MinIO*), which was chosen for its scalability, resilience, and compatibility with team capabilities [3]. Nginx was chosen as a load balancer in the architecture to guarantee performance and dependability right away [1]. Even though the original plan was strictly adhered to, feedback from the development or early testing stages may have led to small changes in the order of features or UI details. For clarity, the definition of roles changed to include separate **Manager**, **Administrator** and **Super-Administrator** responsibilities.

5.3 Challenges and Successes

The risk of hardware/software unavailability, possible delays in developing complex features (such as analytics or notifications), integration issues with current NU systems (**1C/EDMS** were mentioned as a potential future integration point, not necessarily within the project scope but a known complexity), and maintaining strong security and compliance given the sensitive nature of CoI data were among the potential challenges that the project anticipated [1]. Prioritizing essential features, conducting early integration testing, and following security best practices were all part of the contingency plans [1].

Successful teamwork was made possible by well-defined roles (even with shared responsibilities) and the application of project management software such as Jira and Git [3]. A significant accomplishment is the provision of the essential features, which allow declarants to submit forms and managers to examine and resolve disputes. The project remained on schedule in part because of the methodical approach taken from planning to execution. Although still in development, the codebase shows how the selected technologies and architectural patterns are applied [5].

5.4 Problem Solving

The group took proactive measures to address possible obstacles. For example, prioritizing core functionalities reduced the risk of development delays [1]. Planning for data encryption, role-based access control (implemented with Spring Security), secure authentication (JWT-based), and thorough testing all helped to address security concerns [1, 5]. Early testing and backup options, such as independent modules, were intended to address potential integration problems in the event that direct integration proved too difficult at first [1]. During collaborative development, version control (**Git**) was used to manage code changes and settle disputes [3].

5.5 Teamwork

The project's execution relied heavily on teamwork. Members **Nurtore** (DevOps/Azure/Documentation/PM), **Nurlan** (Frontend/UI), **Sultan** (Frontend/UI/Backend), and **Azat** (DB/Backend/PM) were clearly assigned tasks [3]. While **Git** allowed for collaborative development and version control [2, 3], **Jira** was the primary tool for task management and progress tracking [7]. Coordination was ensured by regular communication, most likely through platform comments and meetings. Coordinating activities and monitoring the project's course were part of the shared project management duties [3].

6 Evaluation

6.1 Evaluation Method

User Beta Testing was the main technique used to assess the EDS solution, and it involved important stakeholders such as potential *Declarants* (NU employees) and *Managers* (representatives from the Conflict of Interest, Ethics, and Compliance Office) [1]. System management features might also be tested by *Administrators* (IT personnel). This method focuses on verifying whether the system successfully addresses the fundamental issue of ineffective manual CoI management, going beyond basic bug testing [4]. Participants in the test were given particular tasks and scenarios that reflected real-world usage [4], like:

- Entering a *Initial Declaration* and logging in.
- To report a new possible conflict, submit a *Ad hoc Declaration.*
- (For Managers) examining a declaration that has been submitted and looking for conflicts.
- (For Managers) drafting a *Management Plan* for a conflict that has been identified.
- (For Declarants) Receiving notification, reviewing, and acknowledging a *Management Plan.*
- (For Administrators) Managing users.

Methods as observation, task completion analysis (success rates, time taken), and structured questionnaires emphasizing usability, clarity, and overall satisfaction were used to gather feedback [1, 4].

6.2 Validation

The user beta testing was done to confirm a number of important EDS features:

- **Usability:** How users (Declarants, Managers, Administrators/Super-Administrator) could complete the necessary tasks more quickly and easily than with manual methods or standard system administration procedures [1].
- **Requirement Fulfillment:** Whether the functional requirements for each role were directly addressed by the features that were implemented [1].
- **Improvement Over Manual Process:** Whether users thought the system was more dependable, transparent, and efficient than the current manual workflows (mainly for managers and declarants).
- **Effectiveness:** Whether the system fulfilled its intended role in supporting the end-to-end CoI management lifecycle and system administration tasks [1].

The EDS offers a better and more practical computing-based solution to the identified problem, as confirmed by positive feedback and successful task completion during testing [4].

6.3 Data and Analysis

Data collected during the User Beta Testing phase included [4]:

- **Quantitative Data:** Task success rates (the proportion of users who finish tasks without making significant mistakes), average time spent on each task, usability rating scales (such as the System Usability Scale, or SUS scores, if applicable), and error rates.

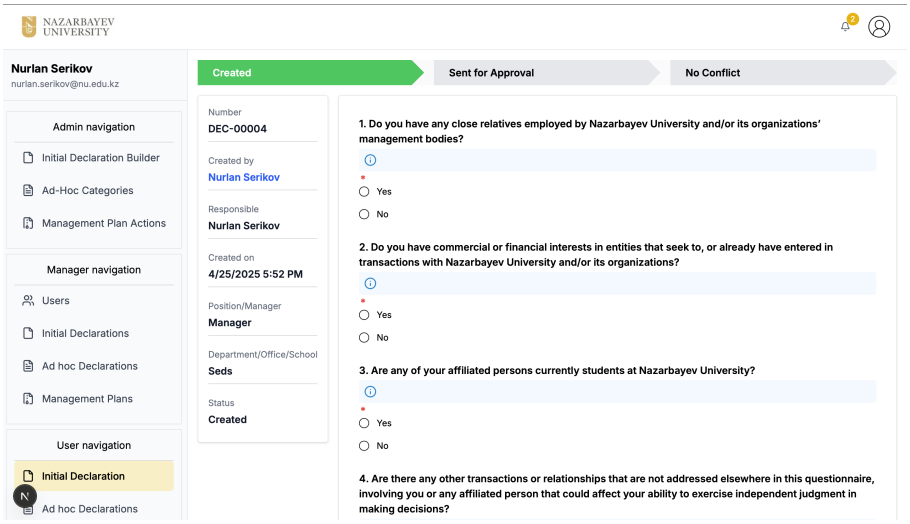

Figure 5: Declarant initial declaration page.

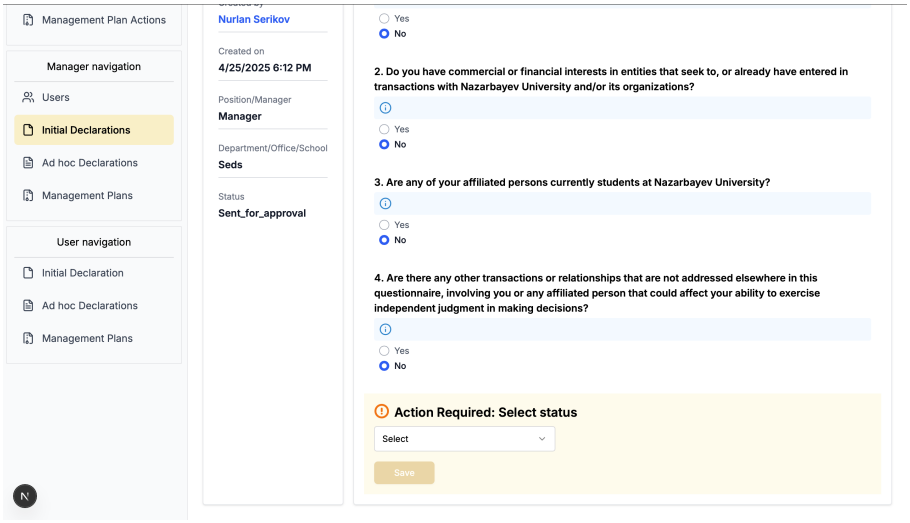

Figure 6: Manager initial declaration ecaluation page.

- **Qualitative Data:** Direct user answers from surveys (e.g., ratings for a feature's usability, instructions' clarity), remarks made during testing, and recommendations for enhancements.

To determine usability bottlenecks, validate feature efficacy, and assess general user acceptance, this data would be examined [4]. High success rates and favorable usability ratings, for example, would suggest a successful design, whereas detailed comments might point out areas that require improvement.

7 Conclusion and Possible Future Work

7.1 Key Findings and Contributions

To overcome the inefficiencies and lack of transparency in Nazarbayev University's manual Conflict of Interest management process, the **Electronic Declaration System (EDS)** project successfully designed and implemented a working web-based platform. The main contribution is a customized computing solution that, with the help of separate Manager, Administrator and Super-Administrator roles, automates the CoI declaration lifecycle, from submission and review to creation and acknowledgement of management plans. Compared to earlier approaches, the EDS improves administrative efficiency, facilitates compliance tracking, and offers a more transparent and user-friendly interface.

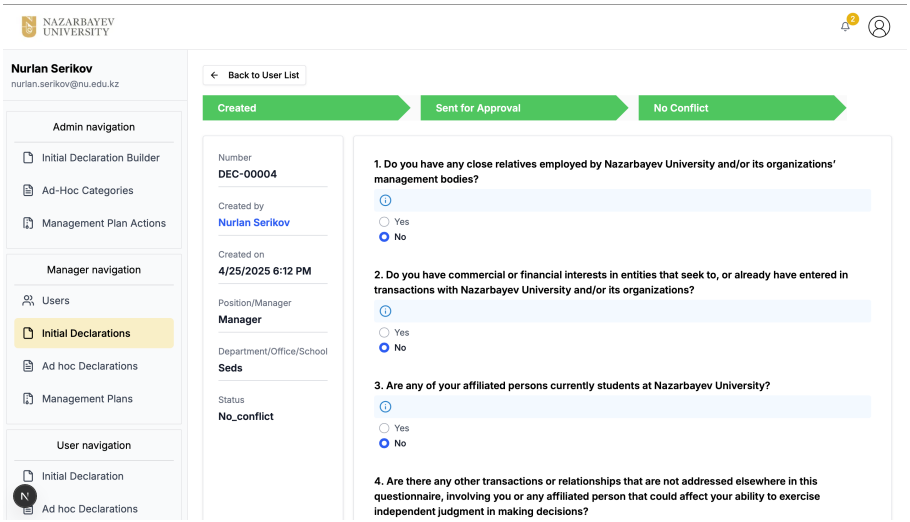

Figure 7: Completed declaration with no conflict

7.2 Future Enhancements

While the current version delivers core functionality, several potential enhancements could further increase its value:

- **Enhanced Data Analytics and Reporting:** Creating improved system of monitoring, dashboards for administrators and more advanced reporting tools for managers to give them a better understanding of CoI trends, compliance rates, and possible risk areas.
- **Integration with NU Systems:** Integration with other university systems, like the EDMS, 1C or Bitrix.
- **Mobile Interface:** Creating a native mobile app, possibly with **Swift** for iOS and **Jetpack Compose** for Android, to enable users to conveniently manage declarations from mobile devices.
- **Workflow Customization:** By using an Administrator interface, managers will have the ability to modify specific parts of the review process or notification templates as policies change.
- **Improved Logging and Error Handling:** Implementation of more comprehensive logging and error handling mechanisms, integrated into NU logging/error detection systems.
- **Testing Coverage:** Creating and conduction of more comprehensive unit and integration tests for greater reliability [5].

7.3 Areas for Improvement

Potential areas for improvement could include:

- Fine-tuning UI elements based on user feedback for improving usability for users.
- Optimizing database queries and backend processes, removing performance bottlenecks under high load.
- Adding more granular controls or customizable reminder schedules to the notification system.

Resolving these issues would enhance the user experience and further refine the system.

8 References

References

[1] Kickoff Document

[2] Requirements and Design document

[3] Updated Senior Project Requirements and Design document

[4] Final Project Report Template

[5] Electronic Declaration System - Backend Codebase - https://github.com/azatAkash/EDS

[6] "Electronic Declaration System - Frontend Codebase - https://github.com/Eager-coder/eds-frontend

[7] Atlassian. Jira Software Documentation: Getting Started.

[8] Pivotal Software. Spring Boot Reference Guide.

[9] PostgreSQL Global Development Group. PostgreSQL Documentation.

[10] MinIO, Inc. MinIO Object Storage.

[11] F5 Networks (Nginx). Nginx Documentation.

[12] Nazarbayev University. Policy and Procedure on Conflict of Interest. *(Placeholder: Add specific document title, version, date, and URL/reference if available).*