Senior Project

Team 55

Housing Management System

Miras Zakaryanov, Nurbergen Agadil, Ralina Tashenova,

Aktilek Ishanov, Temirlan Sansyzbay

Adviser: Askar Boranbayev

2025

Executive Summary

Addressing the inefficiencies of Nazarbayev University's current dormitory operation facilities that are more dependent on manual process generated by the use of spreadsheets, personal communication, and paper-based workflow function. They cause delayed responses to maintenance, errors on room assignments, and administrative burden.

The central purpose of the project was to create a reformatted, digital platform to simplify the housing operations and improve experience for students while increasing administrative efficiency. All are automated and aligned with essential tasks that include room allocation, maintenance ticketing, contract management and internal communication all within a unified interface that is on both desktops and mobile.

It was a project that followed a structured development methodology with user research, design, and implementation in an Agile way. The technologies used include: backend makes use of Java Spring Boot, frontend is implemented in React.js along with Vite and TypeScript, structured data management is done with PostgreSQL and deployment is consistent using Docker. For collaborative UX/UI design for the team was also used Figma and for code management and version control was used Github.

This system demonstrates that a computing based solution is a design, development and evaluation system. UI/UX prototyping user focused were the design phase, full stack web development which includes role based access and modular architecture was the implementation, and the evaluation was multiple multiple review sessions with Housing Management Office. After multiple testing sessions, a series of user testing sessions happened to get approval from the Housing Management's department head, final approval on putting that system in place.

Overall, the NU Housing Management System provides a flexible and implementable solution to the problem of modernizing dormitory services, establishing a framework for future work on campus infrastructure management.

Introduction

Housing management is crucial part of University Infrastructure, which has major contribution to student satisfaction and operational efficiency of the university. There is a significant number of students, who study at Nazarbayev University and live on campus. However, the existing system for managing housing relies mainly on manual processes, including spreadsheets, Google Forms, in-person interactions and paper-based approach. These methods may lead to issues such as mix-ups with room assignments, slower responses to maintenance requests, and extra work for the administrative team.

Therefore, our team has developed the **NU Housing Management System**, which is a centralized, digital platform designed to streamline the full spectrum of housing operations. The system can be used by administration and residents in order to automate tasks such as room allocation, contract creation, maintenance tracking and general communications. In addition, it provides a responsive user interface to make the interaction with system more comfortable.

The core objective of the project is to enhance the quality and efficiency of housing services through technology. By integrating modern web development frameworks, robust backend infrastructure, the platform ensures reliability, scalability, and ease of use.

Background and Related Work

During the observation phase, our team identified that the current dormitory management system mostly rely on processes, which are manual, or rely on basic digital tools, including Spreadsheets, Google Forms and Email communication. While these methods might work for small operations, they often cause problems in bigger settings—like slow maintenance, inefficient room assignments, and extra work for administration staff. As the need for better housing systems grows, smarter and more automated solutions are being developed.

In the case with Nazarbayev University, there is one point that has to be mentioned. The housing related payments are handled through the centralized portal **my.nu.edu.kz**. This system works well with managing financial transactions related to housing operations, however it operates separately from Housing related processes such as room assignments and maintenance requests. This separation may lead to communication gaps and limits the potential for a fully integrated student housing experience.

Our approach places equal value on administrative efficiency and the student experience. Previous systems, while sometimes successful at automating certain tasks, often fall short on the unifying side. That is, they do not provide a single, customizable interface for both students and administrators in which to work. Ours is an attempt to solve that shortcoming, incorporating as it does some very key features: real-time room allocation (both automatic and manual), automatic generation of contracts, and maintenance tracking with feedback.

By analyzing current literature and available tools, we identified key areas of improvement: real-time data synchronization, user role customization, and mobile accessibility. These insights informed our design decisions and guided the development of a modular, scalable solution tailored to the needs of our university's dormitory system.

Project Approach

The proposed solution for the dormitory management system is designed to streamline operations and enhance the experience of residents. Our web-based platform aims to improve administrative workflows, offering an intuitive interface for residents to submit maintenance requests, track room assignments, and stay updated with essential dormitory notifications.

System Architecture

The system is built using a client-server architecture. This architecture separates the user interface from the backend logic. This approach ensures flexibility and ease of maintenance.

- ***Frontend Development***

The team created a complete housing management system through development with current React-based modern technology components. The framework combines React Version 18 and TypeScript for building a robust application with type-safe code throughout the base. The development environment employed Vite as its build tool because it generates results faster and more responsive than older alternatives such as Create React App.

The application makes use of TailwindCSS version 3.4.1 for its style management. The predefined class set in this CSS framework enables developers to write CSS quickly while creating their UIs. The application uses React Router version 6 to manage navigation because it provides a declarative and efficient routing solution. The application features ApexCharts and JSVectorMap APIs to enable interactive representation of dashboard data using charts and geographic displays. chíos API requests are handled by Axios which serves as one of the most popular promise-based HTTP client systems.

The project adheres to a clean and maintainable code structure. The application maintains separate directories specifically for components while also including pages as well as layouts and API services which enables projects to scale better. Every module in the system performs a particular operation that supports the complete housing management process.

User authentication state functions through AuthProvider in React Context which serves as the authentication mechanism for the system. The system prevents unauthorized users from accessing certain application areas through its protected route functionality. Built-in validation features exist within the login form while it also displays user-friendly error messages to the user.

Key metrics obtain summary presentation through the dashboard which shows the resident count alongside applications submitted and active maintenance requests numbers. ApexCharts generate real-time data visualizations which display application status data and the recent incident feed provides users with the latest reports.

The housing management module consists of systems that let administrators control building operations while allowing data filtering and building exports and tracking room utilization rates. The system allows users to handle housing-related contracts within this module. The system allows users to browse resident profiles and change them and manage employee accounts and apply role-based permissions to system access.

Through the dedicated module users can submit applications which allows them to monitor status changes alongside full application visualization. The application provides visible status types including pending, approved and rejected to users along with administrators across both groups. Users can resolve maintenance problems by using structured forms to make their requests while also following the request status and obtaining detailed information about each submission.

From a technical perspective, the application leverages several modern React design patterns. User interfaces that are complex consist of standardized design components which users can reuse throughout the interface. The application implements TailwindCSS grid and flexbox utilities which guarantee responsive layout at any screen dimension. Both the Context API handles global data while local state hooks control individual component operations. The application benefits from TailwindCSS built-in features that support dark mode accessibility alongside user preferences. The application uses React Hot Toast together with React Toastify to give users instant notifications that improve their experience through various user interactions.

As a temporary solution the frontend uses mock data structures as API substitutes to represent real-time data within simulated structures. The applied design structure enables simple adoption of actual backend systems when moving toward future development milestones. The UI component architecture relies on TailwindCSS to style the accessible and usable components that come from Headless UI components. A uniform application interface results from custom components and data display elements alongside validated form inputs and layout wrappers and tables and cards.

- ***Backend Development***

1. Overview

The Housing Management System (HMS) backend is developed on Spring Boot enterprise grade framework which is robust and scalable enough to cater to the varied requirements of university housing operations. It contains a system to manage such properties, users, leases, maintenance requests, application forms and reporting with RESTful API. The system is using Spring Data

JPA and Hibernate to work with data persistence efficiently, while the relational database to print all relevant data is PostgreSQL 16. Docker and Docker Compose are used to deploy systems in a containerized environment, making it easy to scale, portable, and have a consistent environment for development, testing, and production. For version control and collaboration, this project's source code is saved in the Git repository of GitHub. The core architecture, data models, service layer and API endpoints of this section outlines the backend structure, setup, deployment with full reference to other developers and stakeholders by depicting the structure with source code.

HMS backend is the main aim of simplifying and optimizing the university housing management. The system deals with the management of different types of property, namely dormitories, cottages and apartments, their availability and occupancy status. It also contains functions of roles handling through different user roles such as Student role, Faculty role, Housing Management staff role, Maintenance staff role, and Student Services department with role specific permissions and access to each. The system handles all aspects of lease activity, from creating, renewing, terminating leases, and taking care of the family when warranted. Furthermore, it smoothens the requests maintenance process for users to request and keep track of the resolution. Apart from that, they also have the backend that helps accept housing applications, generate automated reports and many others. There is a complete dashboard to monitor your key metrics and personal data at your fingertips. It is a modular, scalable and extensible system that allows adding new features with ease in the future.

2. Architecture

Housing Management System (HMS) is structured at the back end with layered architecture which helps to separate the concerns clearly, to be maintainable and scalable. This approach facilitates efficient development, easy testing, and future enhancements. Spring Boot 3.x is used as the core framework of the backend which has reduced the development process using its wide ecosystem such as Spring Data JPA and Hibernate for Object Relationship Mapping (ORM). Finally, these components facilitate a smooth integration with PostgreSQL 16 which is used as the relational database to manage the data of the housing. Report data is stored in jsonb and statuses and roles are managed with enums that make PostgreSQL advanced features even more flexible and efficient.

HMS backend is exposing RESTful API endpoints, and every response will be in JSON format following industry best practice. They are very well documented using OpenAPI/Swagger annotations i.e. @Tag and @Operation and developers and integrators can easily understand and interface with the system. A session based mechanism for Authentication is done in DashboardController through HttpSession but there is a space for implementation of better methods for example JWT or Spring Security in following versions of code.

Service interfaces like StudentService and LeaseService contain business logic and attempts are made to make the interfaces modular and reusable. The responsibility of these services is to go perform core operations and to keep data integrity. All the entities in the system are managed for persistence with Spring Data JPA repositories, and data access is managed by the repositories to help querying any data using powerful querying APIs that are provided by Spring Data.

The HMS backend is containerized using Docker and Docker Compose for deployment purposes, enabling easy deployment and scaling the application on different environments. A custom network (hms_network) and a persistent volume (hms_postgres_data) store database data in the containerised way across a container restarts. Application.properties and environment variables are used to configure the environment, and other settings like the connection URL of the PostgreSQL (SPRING_DATASOURCE_URL) is configured in a flexible way.

A feature of the HMS architecture is the use of inheritance hierarchies, in particular the BaseUser and BaseProperty abstract classes. These are the basis for creating user and property entities and hence, reuse and consistency while performing operations on different types of users (student, teacher, housing staff) and properties (dormitory rooms, cottages, apartments). The current design is simplified to add new features or entities easily in the future.

3. Data Model

The HMS backend uses a well-defined data model based on JPA entities and on joined table inheritance for extensibility and flexibility. The model is a biased collection of three main parts: properties, users and operational entities to manage various parts of the housing system.

3.1 Property Entities

Housing units are the housing units of which the HMS is responsible, and those are represented by property entities. Common fields which are contained in the BaseProperty class include the id (UUID), propertyNumber (auto created e.g. PROP-XXXX-YYYY), rent, status (using PropertyStatus for values such as OCCUPIED, VACANT, isPaid, maxOccupant, and depositAmount. With @PrePersist, the propertyNumber is generated if what is not provided explicitly.

BaseProperty is extended by the DormitoryRoom, Cottage, CampusApartment, OffCampusApartment, and Townhouse classes, and each of them adds specific properties. For instance, DormitoryRoom contains a room type (RoomTypeDormitory), CampusApartment and

OffCampusApartment provide extra fields onCampusApartment type and address respectively. The system allows efficient management of different property types in subclass specific attributes.

3.2 User Entities

User entities in HMS represent individuals interacting with the system. The BaseUser as a foundation is composed of fields such as id (UUID), first name, last name, email, password, and gender (using Gender enum). BaseUser is specialized to user types with various roles in the system. Suppose Student has a role field (using StudentRole enum) and the school, and Teacher has the position field in order to represent the teacher's academic rank. HousingManagement, Maintenance, and DSS represent staff roles, with attributes specific to their functions in the housing management process. Base User type represents family members associated with a student or tenant and extends the family member type.

3.3 Operational Entities

HMS can be supported by operational entities in playing a crucial role in supporting the workflows. BaseProperty and BaseUser link with a Lease entity that has status and other fields, e.g., family members and lease terms. Student housing applications are stored in ApplicationForm and application documents can be uploaded using @Lob. It is MaintenanceRequest and tracks maintenance issues, linking users, leases and to the maintenance staff with status and requestType fields. Finally, the BaseUser presents reports generated in jsonb format and is linked to the SavedReport entity.

3.4 Enums

The data model makes extensive use of the enums to categorize different parts of the system and seamlessly ensure type safety and prevent errors. Property Status, Room Type Room, On Campus Apartment Type, and Off Campus Apartment Type are property related enums. Enums related to the user include Student role, Teacher position, Housing Management role, Maintenance role, Department of Student services role, and Gender. LeaseStatus, MaintenanceRequestType, MaintenanceRequestStatus are examples of Operational enums. Such enums assist in maintaining consistency and accuracy of the system.

4. Service Layer

Its job is to encapsulate core business logic of the HMS in the service layer. With the inheritance from BaseUser and BaseProperty, the design of the service layer is consistent using user and

property entities. We implement services as interfaces where we've used @Transactional annotation such that database operations are handled appropriately. The @Service annotation is used for dependency injection of the services that makes them available easily to anywhere in the whole application.

4.1 Common Service Patterns

It is made up of several common patterns to ensure consistency and simplicity of use in the service layer. All entities are implemented with the CRUD (Create, Read, Update, Delete) operations, such as the methods createUser, findById for users, and createProperty, findById for the properties. Role or type based queries can be filtered (e.g. findStudentsByRole) and quick accurate entity statistics can be gained (e.g. countByRole, countByStatus). Because we use DTOs (Data transfer objects), we transfer the data from one layer to another without exposing internal entities directly. At the service level, the transaction management is handled, read only transaction is only for the query and full read write transaction there for the data integrity.

4.2 User Services

The CRUD operations of different roles such as user role for HousingManagementService, StudentService, TeacherService, MaintenanceService as well as DSSService is handled. There are these services which support filtering by role (loggedInUser plays roles that cmap_student, etc), updating user info (can make changes in role of students), things like tracking relatives or roommates.

4.3 Property Services

The CRUD operations for particular types of properties are handled by DormitoryRoomService, CottageService and CampusApartmentService Services. Status updates (e.g., updatePropertyStatus), status filtering (e.g findByStatus), room type filtering (e.g., findByRoomType) are included as well. The LeaseService manages the shared logic such as lease assignments so there is a consistent view across different property types.

4.4 Shared Services

There are common functionalities across various entities hence shared services such as LeaseService, MaintenanceRequestService, and DashboardService. Leases lifecycle is managed by LeaseService, which means that we create, renew and terminate the leases.

4.5 Service Layer Design

The service layer is designed for reusability and extensibility. Generic operations are centralized in base controllers, and entity-specific logic is modularized into specialized services. This design allows for easy additions of new user or property types in the future, making the system adaptable to changing requirements.

5. API Endpoints

The RESTful API is exposed via controllers, organized by entity type, with Swagger documentation for clarity:

- DashboardController (/api/dashboard):
    - GET /stats: Returns active lease count.
    - GET /: Returns user-specific dashboard data (session-based authentication).
- BaseUserController (/api/users):
    - Generic management for BaseUser subclasses.
    - Endpoints: GET /{id}, DELETE /{id}, GET /by-email, GET /by-nuid, GET /by-national-id, GET /by-name, GET /, GET /count, GET /count-by-type.
- StudentController (/api/students):
    - Manages Student entities, with filtering by StudentRole and SchoolsAndSpecialties.
    - Endpoints: GET /, GET /{id}, DELETE /{id}, GET /search, GET /role, GET /school, GET /specialty, GET /{id}/roommates, GET /{id}/ex-roommates, GET /{id}/leases, GET /count-by-role, GET /count-by-school, GET /count-by-specialty.
- TeacherController (/api/teacher):
    - Manages Teacher entities, with filtering by TeacherPosition.
    - Endpoints: GET /{id}, GET /, GET /search, GET /nuid/{nuid}, GET /school.
- LeaseController (/api/leases):
    - Manages Lease entities for BaseProperty subclasses, with pagination and filtering by LeaseStatus.

    - Endpoints: POST /, PUT /{id}, DELETE /{id}, GET /, GET /{id}, GET /property/{propertyId}, GET /property/{propertyId}/active, GET /tenant/{tenantId}, GET /status/{status}, GET /active, GET /expiring, PUT /{id}/renew, PUT /{id}/terminate, PUT /batch-terminate, GET /search, POST /{leaseId}/add-family-members.

- MaintenanceRequestController (/api/maintenance-requests):
    - Manages MaintenanceRequest entities, with filtering by MaintenanceRequestStatus.
    - Endpoints: POST /, GET /{id}, GET /number/{requestNumber}, GET /user/{userId}, GET /lease/{leaseId}, GET /assigned-to/{staffId}, GET /status/{status}, GET /, PUT /{id}, PUT /{requestId}/assign/{staffId}, PUT /{requestId}/status/{status}, PUT /{requestId}/mark-paid, PUT /{requestId}/mark-completed, DELETE /{id}.
- ApplicationFormController (/api/application-forms):
    - Manages ApplicationForm entities.
    - Endpoints: POST /, GET /, GET /{id}, PUT /{id}, DELETE /{id}.
- DormitoryRoomController (/api/dormitory-rooms):
    - Manages DormitoryRoom entities, with filtering by PropertyStatus and RoomTypeDormitory.
    - Endpoints: POST /, PUT /{id}, DELETE /{id}, GET /{id}, GET /, GET /status, GET /search, GET /available, PUT /{id}/status, GET /price-range.
- Other Property Controllers (assumed):
    - Controllers for Cottage, CampusApartment, OffCampusApartment, and Townhouse, with similar endpoints to DormitoryRoomController.

6. Workflow and Interactions

Housing Management System plays a direct role in the designing of the backend, which would be capable of handling seamless end to end workflow of different housing management tasks. They are workflows that help user management, property management, lease handling, maintenance requests, application processing, and reporting with different set of processes and interactions per workflow.

6.1 User Management

The user creation and management in the HMS are very important roles which are performed by the BaseUserController or by the specific controllers for each user type (e.g. StudentController). For example, admins can create users, for example students, teachers or staff, and assign their roles. The StudentService and HousingManagementService take care of the role assignments, for example changing an student's role or assign a housing management block. The system provides admin the flexibility to filter students by filters like role, school, block using StudentService.findStudentsByRole and other such functions.

6.2 Property Management

With properties instances come dormitory room, cottage and apartment structures created by specific property controllers (e.g DormitoryRoomController). This puts on equal footing property management to the system. Properties' status can be updated through endpoints such as PUT /{id}/status, which will let admins mark properties as occupied, vacant or under maintenance. With property queries, one can filter using specific statuses or types with functions such as DormitoryRoomService.findByRoomType that filters the room types (or other property characteristic).

6.3 Lease Management

It is important to associate tenants to a property by lease management. A user (tenant) is assigned to a property (BaseProperty) by using LeaseService.createLease. It also permits the renewal or termination of leases via LeaseService.renewLease and LeaseService.terminateLease methods on the service, and supports managing the lifecycle of leases. Furthermore, the system enables family members to be linked to a lease by means of the LeaseService.addFamilyMembersToLease method to reflect family relations amongst tenants. LeaseService.getLeasesByProperty allows admin or users to query leases with properties, tenant, or lease status in the returned response.

6.4 Maintenance Requests

The maintenance requests are part of the workflow of the system in the MaintenanceRequestController, and thus users will be able to prepare maintenance requests. The MaintenanceRequestService will once assigned to maintenance staff and will track where the request is. MaintenanceRequestService.updateStatus is used to make updates on the progress of the maintenance task as it was updated. Functions such as MaintenanceRequestService.getRequestsByLease allow users to filter maintenance requests by status, user, or lease to guarantee an economical and efficient process.

6.5 Application Forms

Housing applications can be submitted by the students using the ApplicationFormController who can fill in the essential information required for housing assignments. The DSSService (Department of Student Services) will review these applications after submission. Then, this workflow ensures that the students and DSS staff feel comfortable and ordered in the application process.

6.6 Dashboard

The DashboardService provides detailed reporting anduel time metric for the HMS backend as well. This service aggregates important data, such as active leases, user-specific data, and other key metrics. The insights are critical for housing management staff to monitor the status of housing operations and determine decisions based on this. It also includes SavedReport entities for storing the custom reports as jsonb, which provides the flexibility to users and administrators to generate and access different type of reports.

Features and Workflows

The platform provides several features designed to address both resident needs and administrative tasks:

- **Room Assignment:** The assignment of rooms is managed by dormitory administration. Administrators have the ability to assign rooms based on availability and residents' preferences.
- **Maintenance Requests:** Residents can easily submit maintenance tickets for issues within their dormitory, such as plumbing problems or heating malfunctions. They can track the status of these requests in real-time, providing transparency and reducing the need for follow-up inquiries.
- **Administrative Control Panel:** This feature provides dormitory staff with a comprehensive view of room assignments, maintenance requests, and other administrative settings. It allows for the management of the entire dormitory system in one centralized platform.

Roles and Responsibilities

The system accommodates different user roles with varying levels of access and control:

- **Residents**: Residents can submit maintenance requests, check their room assignments, and receive updates on the status of service tickets. They have access to a personalized dashboard to manage their activities within the dormitory.
- **Housing Management Staff:** Staff members have the ability to manage room assignments, handle service requests, and send important announcements to all residents. They have administrative control over system settings.
- **Maintenance Personnel**: Maintenance staff can access submitted tickets, update their status, and resolve issues in a timely manner. They are responsible for ensuring that maintenance requests are handled efficiently.

Tools and Technologies

To build and deploy this platform, the following tools and technologies were used:

- React.js for developing a responsive and user-friendly frontend.
- Java Spring Boot for a powerful and secure backend infrastructure.
- PostgreSQL for data storage, ensuring reliable and structured information management.
- Docker for containerization, which makes the deployment process smoother and ensures consistent performance across different environments.
- IntelliJ IDEA as the primary integrated development environment (IDE), providing a feature-rich workspace that enhances development efficiency.
- Figma for creating detailed wireframes and UI/UX prototypes before development.
- GitHub for version control and team collaboration, ensuring efficient code management and progress tracking.

Team Collaboration

Agile methodology was used by the team to efficiently develop the system. Each and every member played a vital part in the project and through his or her expertise did well.

- Miras Zakaryanov and Nurbergen Agadil: Led system design, architecture planning, and database setup with PostgreSQL. Moreover, they worked on backend using Java SpringBoot, implemented Redis and Kafka for better performance.
- Temirlan Sansyzbay and Aktilnik Ishanov: It is mostly managed with using Figma to developed an user interface and user experience design.
- Temirlan Sansyzbay, Aktilek Ishanov: Focusing about frontend development with React, developing core functionalities, such as room selection UI. Security testing and usability improvements, based on user feedback, were also something they alsoprovide on.
- Miras Zakaryanov: CTO Dockerization (took responsibility for Dockerizing the system). In addition, he played a key role in performance optimization.
- Ralina Tashenova: Led the Whole project management, Running Sprints of 2-week length as per rules of Agile methodology. A system of sprint management was set up in Athe tlas Jira platform, with each task and subtask managed as tickets and assigned to each member of the team. Upon the completion of the project, Ralina was responsible for video presentation production (organization, implementation and post-production parts.)

The system was thoroughly tested, and secure and was optimized for final deployment by the team working cohesively. The project was a success only because of each member's work.

Project Execution

During two semesters, our team has worked together to come up with and build a web based dormitory management system from the backend of java SpringBoot, and from react.js on the front end. From the get go, how we envisioned administrative workflows and the users' experience, that's people living within the community, we wanted to simplify with a little bit of separation separation in roles and responsive interfaces.

We visited the Housing Management Office three times to better get a feel of user needs and current administrative pain points that directly influenced our feature set and user flows. We based on their requests and successfully implemented a manual room allocation feature for staff control and a feedback system for residents to rate and comment on maintenance services (both that were specially requested during our consultations).

We evolved early design choices as we came up against practical constraints. One example of this relates to multiusing of the database schema to provide the room assignment logic and maintenance tracking. For the storage of structured data, we relied on PostgreSQL and used Docker in order to have a consistent development and deployment environment.

Jira was used for task coordination and mates contributed wherever they had skills. Regarding backend and system architecture, Miras Zakaryanov and Nurbergen Agadil were responsible, whereas Ralina Tashenova and Temirlan Sansyzbay as well as Aktilek Ishanov mostly worked on building the user interface and basic frontend features. We then quickly resolved blockers using collaborative problem solving sessions, and were able to iterate the system by refining it iteratively.

While integration lagged and dealt with some bugs, the team consistently made progress exactly as needed in response and eventually delivered a working prototype that matched user requirements and technical scalability.

But not everything planned to be included in it was completed. Due to time constraints, electronic signature was not provided for housing contracts functionality. The payment system integration, which would have enabled students to pay housing fees in the platform, was not implemented, as well. These limitations are potential areas for future development as the system continues to evolve.

Evaluation

In order to make sure our system addressed the heart of the outdated dormitory processes, we did a round of evaluation with the Housing Management Department. We held three reviews officially with the Housing Management representatives across the development timeline, where we would show them our progress, get feedback and adjust features according to their needs.

We presented real use cases, as well as actually did some testing during these sessions of the room assignment tracking, maintenance ticketing, administrative overviews. The feedback then was used to evolve frontend usability and backend workflows.

The system was evaluated in the final evaluation and Housing Management was fully satisfied. Ayagoz Kulekeyeva, the head manager approved the solution as it fitted with their operational expectations and she believed that it would be more efficient and provide better user experience for the staff and the residents.

We seek direct collaboration with the end users, and their approval about final results validate that such a solution was successfully done as per the goals that were stated in the introduction.

Conclusion

The contribution of this project included a web based dormitory management system that was specially designed and catered for the requirements of Nazarbayev University's Housing Management. The system is built using Java SpringBoot and React.js, replacing outdated manual processes, thus making the administration more efficient and more satisfactory. We have made key contributions through streamlined room assignment, tracking the upkeep digitally and with a clear, user friendly interface.

In addition, full permission from the department head followed after three evaluation meetings following close collaboration with Housing Management to ensure that the platform remained inline with real operational workflow.

Several enhancements will lead to further impact of the system looking ahead. These include adding real time notification of maintenance updates, creating analytics dashboards for resource planning, and adding multilingual interface support to the software mentioned earlier. It could also integrate with existing university services (i.e., student portals) for better integration into back-end solutions.

In conclusion, the project acts as a solid base for a scalable as well as intelligent dormitory management solution.

Reference List

1. *Housing Management System - Frontend*. GitHub Repository. Available at: https://github.com/nuraga20/hms-front

2. *Housing Management System - Backend*. GitHub Repository. Available at: https://github.com/Buzhukovv/hms