WOLKITE UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATICS

DEPARTMENT OF SOFTWARE ENGINEERING

WEB BASED INTERNSHIP MANAGEMENT SYSTEM

INDUSTRIAL PROJECT

BY

NAME OF THE STUDENTS	ID NO
ABEL ZELEKE	NSR/0055/12
BEREKET TADELE	NSR/0285/12
REBECCA ASRAT	NSR/1208/12
TSION TEGENE	NSR/1480/12

PROJECT ADVISOR: Mr. HAILEMICHAEL K.

Wolkite University, Wolkite, Ethiopia

May 09, 2024

WOLKITE UNIVERSITY

COLLEGE OF COMPUTING AND INFORMATICS

DEPARTMENT OF SOFTWARE ENGINEERING

WEB BASED INTERNSHIP MANAGEMENT SYSTEM

SUBMITED TO DEPARTMENT OF SOFTWARE ENGINEERING IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHLER OF SCIENCE IN SOFTWARE ENGINEERING

BY

NAME OF THE STUDENTS	**ID NO**
ABEL ZELEKE	NSR/0055/12
BEREKET TADELE	NSR/0285/12
REBECCA ASRAT	NSR/1208/12
TSION TEGENE	NSR/1480/12

PROJECT ADVISOR: Mr. HAILEMICHAEL K.

Wolkite University, Wolkite, Ethiopia

May 09, 2024

DECLARATION

This is to declare that this project work which is done under the supervision of Mr. Hailemichael and having the title Web Based Internship Management System is the sole contribution of:

- Abel Zeleke,
- Bereket Tadele,
- Rebecca Asrat, and
- Tsion Tegene.

No part of the project work has been reproduced illegally (copy and paste) which can be considered as Plagiarism. All referenced parts have been used to argue the idea and have been cited properly. We will be responsible and liable for any consequence if violation of this declaration is proven.

Date: Jan 25, 2024

Group Members:

Full Name	Signature
_____	_____
_____	_____
_____	_____
_____	_____

APPROVAL FORM

This is to confirm that the project report entitled Web Based Internship Management System submitted to Wolkite University, College of Computing and Informatics Department of Software Engineering by: Abel Zeleke, Bereket Tadele, Rebecca Asrat, and Tsion Tegene is approved for submission.

Advisor Name	Signature	Date
Department Head Name	Signature	Date
Examiner 1 Name	Signature	Date
Examiner 2 Name	Signature	Date
Examiner 3 Name	Signature	Date

ACKNOWLEDGEMENT

We begin by expressing our sincere thanks to Almighty God, the foundation of our strength and achievements. Our heartfelt appreciation goes out to our families for their support and encouragement. We also extend our gratitude to the colleges for their assistance and provision of essential information needed for our work.

Our special thanks to our supervisor, Mr. H/Micheal, for his outstanding guidance and mentorship during this project. His consistent support and expertise have been instrumental in guiding and ensuring the success of our project.

ABSTRACT

Internships are vital for students' educational and career growth, providing essential real-world work experience. Recognizing the need to streamline this process, we propose the development of a Web-Based Internship Management System. This innovative platform will simplify the search for internship, and management of interns, enhancing communication and collaboration between interns and supervisors for a more effective internship experience. Designed to be user-friendly and adaptable.

This system promises to significantly improve the efficiency of internship management, benefiting interns, Universities and organizations.

Tabel of Content

List of Tables

List of Figures

LIST OF ACRONYMS & ABBREVIATIONS

API	**Application Programming Interface**
AUTH	Authentication
CPU	Central Processing Unit
CSS	Cascading Style Sheet
GUI	Graphical User Interface
JS	JavaScript
NextAUTH	Next Authentication
NPM	Node Package Manager
OOD	Object Oriented Design
OOSAD	Object Oriented System Analysis and Design
RAM	Random Access Memory
UML	Unified Modelling Language
URL	Uniform Resource Locator
WIMS	Web based Internship Management System Management System

CHAPTER ONE

INTRODUCTION

Wolkite University currently uses a paper-based system to manage internships, which is inefficient and difficult to track. To address this issue, a web-based Internship Management System (IMS) is being developed. This system will be a one-stop-shop for students, employers, and faculty to handle everything internship-related, from student registration to performance feedback.

The benefits of the IMS include improved efficiency and effectiveness, a clearer view of internship opportunities for students, and better communication among everyone involved. The system will also be tailored to the specific needs of Wolkite University, taking into account the university's curriculum, student needs, and employer network.

The developers of the IMS are working closely with faculty, staff, students, and employers to ensure that the system meets the needs of all stakeholders. The goal is to create a system that is easy to use, affordable, and effective in helping students gain valuable internship experience.

1.1 Background of Wolkite University

Mission: Wolkite University is driven by a mission to provide quality education, training, research, and community services that align with the needs of the country and the region.

Vision: Wolkite University’s vision is to stand among the top ten universities in East Africa by 2027 E.C.

Objectives

- To produce graduates who are not only competent but also ethical, prepared to make meaningful contributions to the development of our nation and the region.
- To conduct research that addresses societal needs and advances knowledge.

- To provide community services that elevate the quality of life for the people in our region.

Organizational Structure

Wolkite University is structured into the following colleges and schools:

- College of Engineering
- College of Computing and Informatics
- College of Natural and Computational Sciences
- College of Social Sciences and Humanities
- College of Business and Economics
- College of Medicine and Health Sciences
- College of Agriculture and Natural Resources
- School of Law

The Internship Management System (IMS) holds a pivotal role in aiding Wolkite University in helping with its mission, vision, and objectives. The IMS serves as a central platform for students, employers, and faculty, facilitating the comprehensive management of internships, including:

- Student registration and application
- Internship posting and matching
- Performance evaluation and feedback
- Reporting and analytics

By enhancing the efficiency and effectiveness of internship management, our IMS plays a pivotal role in cultivating graduates who are not only skilled but also well-prepared for the workforce. It serves as a bridge, connecting students with the best companies and work opportunities, ensuring they step into their careers with a solid foundation and a wealth of opportunities.

1.2 Statement of the Problem

Current Internship Management System

At Wolkite University, the existing internship management system operates through a combination of manual and semi-automated processes. Students seeking internships are required to independently search for internship opportunities with companies (employers).

In cases where students are unable to secure internships on their own, the university assigns them to government-affiliated companies.

Throughout the internship period, advisors from the university visit the hosting companies to assess the progress of the students and gather information for evaluation.

Communication between the university and students primarily takes place through email. The culmination of the internship experience is marked by a final presentation from the students.

Challenges with the Existing System

The current system presents several challenges and limitations:

- ✓ Unclear Company Options & Requirements: Struggling to find internships because we don't know which companies are available or what they're looking for.
- ✓ Data Problems: Without one system to keep track of everything, we end up with mixed-up and wrong information, messing up reports and evaluations.
- ✓ Old-School Evaluations: Advisors have to physically visit companies to see how students are doing, which eats up time and doesn't give updates as they happen.
- ✓ Reporting Hassles: The current setup makes it tough to pull together reports and analytics, so it's hard to see if the internship program is working well or not.

1.3 Objectives of the Project

1.3.1 General Objective

The general objective of this project is to develop a web-based Internship Management System (IMS), with the aim of enhancing the efficiency and effectiveness of its internship management process.

1.3.2 Specific Objectives

- ✓ To enable student signup with their details, background, and skills.
- ✓ To provide portal for posting internship and selecting applicants.
- ✓ To streamline applications and approvals for internships.
- ✓ To track student progress and provide regular feedback.
- ✓ To evaluate student performance and offer feedback.
- ✓ To create a user-friendly interface for all users.

1.4. Feasibility Analysis

1.4.1. Technical Feasibility

From a technical standpoint, the proposed system is feasible for development. Our team possesses the capability to accomplish this by leveraging existing technologies such as Next.js, MongoDB, and Nest.js. We are equipped with the necessary hardware, including laptops, and have access to reliable internet connections, alongside the requisite software tools for programming.

1.4.2. Operational Feasibility

Operational feasibility for the web-based internship management system is promising, as it addresses genuine user needs by providing an intuitive interface that simplifies the internship placement process. Support from management is evident, and users are actively seeking a more streamlined approach to managing internships. The system is expected to significantly reduce the time and effort required for internship operations, making it a practical and valuable addition to the educational landscape.

1.4.3. Economic Feasibility

The Web-Based Internship Management System demonstrates strong economic viability through cost savings, efficiency gains, increased placements, enhanced collaboration, scalability, and improved student outcomes. By automating processes, reducing administrative costs, and fostering better student-employer matches, the system ensures a positive return on investment for educational institutions. Its adaptability and long-term sustainability make it a financially sound

choice, contributing to a thriving educational environment and boosting the economic prospects of all stakeholders.

1.5 Scope

Our project is focused on the development of a Web-Based Internship Management System. The primary goal is to establish an intuitive and efficient platform that transforms the internship management process. This system will facilitate:

- ✓ Seamless registration for users and the management of their profiles.
- ✓ An employer portal for the posting of opportunities and the review of applications.
- ✓ Comprehensive reporting and analytics capabilities.
- ✓ Evaluation and tracking of student progress.
- ✓ Robust user account management.

Limitations of IMS

- ✓ The system may not be fully accessible to people with disabilities, such as those with visual impairments.
- ✓ The system does not automatically monitor students' attendance during their internship.

1.6 Significance of the project

Internship management systems are significant because they can help universities, employers, and students to improve the internship experience for everyone involved.

IMS can help university to:

- ✓ Streamline the internship application and selection process
- ✓ Track student progress and provide support
- ✓ Generate reports to assess the success of the internship program

For employers:

- ✓ Post internships and manage applications
- ✓ Review and select student applicants
- ✓ Track student progress and provide feedback
- ✓ Generate reports to assess the success of the internship program

For students:

- ✓ Find and apply to internships
- ✓ Track their internship progress and manage their workload

1.7 Beneficiaries of the Project

Students: The platform primarily benefits students by providing them with convenient access to a diverse range of internship opportunities. It empowers them to:

- ✓ Search for internships.
- ✓ Apply for internships.
- ✓ Track their application status.
- ✓ Find internships aligned with their skills, interests, and career aspirations.

Companies:

- ✓ Post internship opportunities.
- ✓ Interact with a pool of talented students.
- ✓ Select qualified candidates for internships.

Internship Coordinators and Faculty Members: The platform assists internship coordinators and faculty members by providing essential tools for:

- ✓ Monitoring student progress.
- ✓ Evaluating student performance.
- ✓ Streamlining administrative tasks related to internships.

1.8 Methodology of the project

1.8.1 Data collection tools and techniques

Techniques

- Interview
- Survey

Tools

- Recorder
- Paper
- Pen

1.8.2 System Analysis and Design

In the system analysis and design phase, we adopt the Object-Oriented System Analysis and Design (OOSAD) approach, which offers an effective framework for constructing, managing, and integrating objects within our system. This approach encompasses several key phases, including:

- ✓ **Object-Oriented Analysis (OOA):** This phase involves modeling the system's functionality through use case modeling, identifying and organizing business objects, establishing their relationships, and detailing their behaviors.
- ✓ **Object-Oriented Design (OOD):** Utilizing tools such as E-draw, Visio, and Rational Rose, we refine use case models and design sequence and activity diagrams to support use case scenarios and model object interactions.

We chose the OOSAD approach, particularly the Unified Modeling Language (UML) model, for its significant benefits, including enhancing design reusability, reducing software maintenance costs, lightening the maintenance load, ensuring consistency across analysis, design, and programming phases, and improving communication among stakeholders.

1.8.3. System Development Model

In our project, we utilized the iterative model for system development due to several compelling reasons:

- ✓ The project's modular nature, including user management, application processing, and reporting, is well-suited to incremental development, allowing for sequential development, testing, and deployment that enhances the system's value progressively.
- ✓ It facilitates active collaboration with stakeholders such as administrators, coordinators, employers, and interns, accommodating regular feedback and adjustments to meet evolving needs.
- ✓ The model offers flexibility to accommodate changes in requirements and adapt to the dynamic nature of internship programs, ensuring alignment with organizational goals.
- ✓ Early iterations enable risk identification and mitigation, enhancing project stability and managing potential issues proactively.
- ✓ Prototyping during initial phases helps in visualizing system features and refining them based on stakeholder feedback, ensuring the final system meets user expectations effectively.

1.8.4. System Testing Methodology

Unit Testing with Jest Framework

Unit testing focuses on evaluating individual components of the system in isolation to ensure their correctness.

Reasons for Choosing Jest:

- ✓ **Simplicity and Ease of Use:** Jest provides a straightforward and user-friendly testing experience.
- ✓ **Widespread Adoption:** Widely accepted within the React and Node.js ecosystems, aligning with our technology stack.
- ✓ **Focus on Code Quality:** Enables us to concentrate on assessing individual components for code quality.
- ✓ **Early Error Detection:** Facilitates the early identification of errors in the development cycle.

Integration Testing with Jest Framework

Integration testing involves automating tests for integrated system components to ensure smooth collaboration between different procedures.

Reasons for Choosing Jest:

- ✓ **Seamless Integration:** Jest seamlessly integrates into our Next.js and Nest.js development stack.
- ✓ **Collaborative Development:** Aligns well with our collaborative development approach within the Next.js and Nest.js environment.
- ✓ **Comprehensive Feature Set:** Jest's feature set, including built-in assertions and support for asynchronous testing, makes it ideal for integration testing.
- ✓ **Behavior-Driven Testing Philosophy:** Integration with React Testing Library supports our philosophy of testing behavior rather than implementation details.

1.8.5 Development tools and technologies

Frontend technologies

- Next js
- Zustand
- Figma UI design
- Tailwind CSS
- Typescript

Backend technologies

- Nest js
- Node js
- MongoDB

CHAPTER TWO

DESCRIPTION OF THE EXISTING SYSTEM

2.1. Introduction of Existing System

In current college internship systems, internships vary from mandatory to optional. Students apply to companies and, once accepted, seek departmental approval. Advisors guide students, mainly through emails and occasional site visits. If issues arise, the college or Industrial Linkage office helps students find suitable internships, including those with government-affiliated companies.

The process involves application, approval, and ongoing advisor support, with email as the main communication tool and sealed envelopes for confidential evaluations. Advisors' visits provide extra support. Students conclude their internships by submitting detailed reports.

This system combines mandatory and optional internships, streamlining approvals and enhancing communication. A new finance flow has been added, where Department Heads relay students' financial details for stipend processing.

Overall, this system integrates academic and practical learning, aiding in holistic student development.

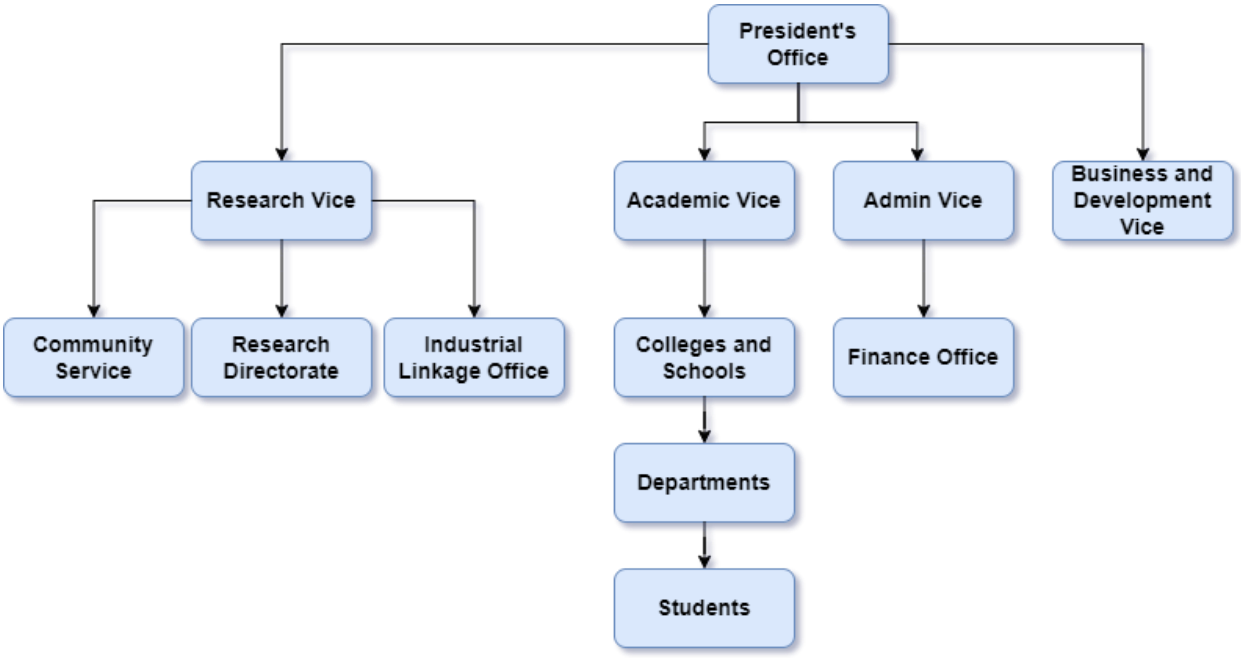


Figure 2. 1 Structure of Existing System

When referring to the "**existing system**" or "**system**" in the subsequent sections, we specifically denote the manual system currently in use by the university.

2.2. Users of Existing System

1. **College Dean**: The college dean oversees the overall internship process within the college. They approve the assignment of advisors as designated by the department heads, ensuring alignment with the college's academic objectives.

2. **Department Heads**: Department heads approve internship placements, ensuring they align with the department's academic goals. They assign advisors to students based on company placements.

3. **Advisors**: Assigned advisors guide and support students throughout their internship tenure. They offer advice, ensure academic relevance, and provide feedback on students' progress and experiences.

4. **Students**: Students initiate the process by applying to companies using provided application forms. They seek departmental approval upon securing acceptance,

communicate with assigned advisors, report work progress via email, and prepare presentations summarizing their internship experiences.

5. **Companies**: Companies accept students for internships, use sealed envelopes provided by the departments for attendance tracking and performance evaluation, and engage in the evaluation process in collaboration with the academic institution.

6. **Industrial Linkage Office**: The Industrial Linkage office intervenes when students face challenges securing placements. They ensure students are assigned to government companies for valuable internship experiences, providing support in placement processes.

2.3. Major Functions of the Existing System

- **Application Management**: Facilitating student applications to companies. Consistent process across departments or colleges for application submission.
- **Approval and Assignment**: Approving applications and assigning advisors.
- **Communication and Reporting**: Enabling communication between stakeholders. Uniform communication channels but varying depth based on departmental policies.
- **Evaluation and Performance Tracking:** The internship evaluation process includes varied weightages: company assessments range from 60% to 70%, while advisor evaluations vary between 30% and 40%. Presentations and assessments also play a key role in summarizing internship experiences.
- **Advisory Support**: Providing guidance to students. Consistent advisory support with differing impact on assessments based on departmental emphasis.
- **Intervention and Placement Support**: Assisting students facing placement challenges. Uniform intervention process for equitable opportunities.

2.4. Sample Forms and Other Documents of the Existing Systems

Sample Forms and Other Documents of the Existing Systems are in the Appendix II part.

2.5. Drawbacks of the Existing System

Challenges in the Existing System:

- ✓ Limited internship choices due to industry focus, competition, and network gaps.
- ✓ Ineffective oversight of student, advisor, and company interactions.
- ✓ Subjective and inconsistent evaluation processes.
- ✓ Advisor support varies with department resources.
- ✓ Risk of misuse with sealed evaluation envelopes.
- ✓ Unstructured internships lead to uneven experiences.
- ✓ Post-internship career guidance is lacking.

The outlined challenges represent significant drawbacks within the current system, condensed here for clarity and professional context.

2.6 Business Rules of the Existing System

- ✓ Department heads approve student placements; college deans oversee advisor assignments.
- ✓ Sealed envelopes for assessments must remain unopened by students to keep evaluations confidential.
- ✓ Students should use designated platforms like email for advisor communication.
- ✓ Standardize evaluation criteria for consistency and fairness in assessments.
- ✓ The Industrial Linkage office assists students with placement challenges.
- ✓ Advisors guide students, linking academic learning with practical experience.
- ✓ Students are required to present summaries of their internships for evaluation.
- ✓ A robust monitoring system tracks engagement among students, advisors, and companies.

CHAPTER THREE

PROPOSED SYSTEM

3.1. Functional Requirements

- ✓ The system should enable students to create accounts by providing essential details and to edit their profiles with personal, educational, skills, and preference information.
- ✓ The system should enable students to search for internships using criteria like industry, location, duration, and skills, and to apply for internships by submitting necessary documents.
- ✓ The system should enable students to track their application.
- ✓ The system should allow company employers to register their companies and post new internship opportunities with detailed job descriptions and application instructions.
- ✓ The system should enable employers to view, shortlist, and manage applications, communicate with universities and students about application statuses, interviews, and selections, and provide feedback and evaluations on intern performance.
- ✓ The system should facilitate employer registration, internship postings, application management.
- ✓ The system should enable mentors to evaluate interns.
- ✓ The system should enable university administration to register the institution and add colleges.
- ✓ The system should enable department heads assign advisors, approve internships and student accounts.
- ✓ The system should authorize system admins to approve universities.

3.2. Non-functional Requirements

3.2.1. User Interface and Human Factors

- ❖ User Interface Design: The system will provide a clean, intuitive, and responsive user interface, utilizing Next.js and Tailwind CSS. The design will be user-friendly, ensuring

ease of navigation and accessibility. The interface will be adaptive, offering a seamless experience across various devices, including desktops, laptops, tablets, and smartphones.

- User Expertise: The system will be designed for a diverse user base, including students seeking internships, administrative staff managing internship programs, and business representatives posting opportunities.
- User Interaction: The system will offer interactive elements like dropdown menus, search bars, and filter options to facilitate easy access to information and features. Feedback mechanisms, such as success messages and error alerts, will be incorporated to guide users through their interactions with the system.
- Consistency and Familiarity: The design will maintain consistency in layout, color schemes, and typography to foster a sense of familiarity and ease of use across different sections of the system. Familiar design patterns and icons will be used to enhance user understanding and reduce the learning curve.

3.2.2. Hardware Consideration

- **Client-Side Hardware:** Since the system is accessed through web browsers, it will be designed to be compatible with a wide range of devices, including desktop computers, laptops, tablets, and smartphones. The key requirement is that these devices should support modern web browsers (like Chrome, Firefox, Safari, etc.). Adequate processing power and memory for a smooth browsing experience. However, since most processing is done server-side, the client hardware doesn't need to be high-end.

- **Server-Side Hardware:** The server hosting the application (Nest.js backend and MongoDB database) should have sufficient processing power, memory, and storage to handle the expected load. This includes managing multiple concurrent user connections and large volumes of data storage and processing. The hardware should be scalable or use cloud-based services that can dynamically allocate resources based on demand.

- The web-based system is not directly involved with other hardware systems. Its interaction is primarily through the internet or network with client devices and potentially other web services.

3.2.3. Security Issues

Security is our foremost concern, right after ensuring optimal performance. The Web-Based Internship Management System places a high priority on implementing robust security measures to protect against internal and external intrusions. Our security protocols will be carefully designed to stop unauthorized access, prevent data breaches, and uphold the confidentiality, integrity, and availability of sensitive information.

Authorized User Security

Security measures are implemented to protect against actions that go beyond the user's designated roles and permissions. This includes monitoring and access control mechanisms to prevent unauthorized users from performing sensitive operations.

Security Levels

The security levels for the Web-Based Internship Management System are defined as follows:

- **User Authentication and Authorization:** JWT (JSON Web Token) is employed for user authentication and authorization. Security tokens are generated upon successful login and carry user role information for subsequent authorization checks.
- **Role-based access control (RBAC):** is implemented to ensure that users can only access functionalities and data relevant to their roles.

Key Features of JWT Implementation

- **Token Generation:** Upon successful user authentication, a JWT is generated and sent to the client as a secure means of identifying the user.
- **Token Payload:** The JWT payload includes information such as user ID, username, and user roles, providing the necessary details for authorization checks.
- **Token Validation:** The system validates incoming JWTs to ensure their integrity and authenticity before processing requests.
- **Token Expiry and Refresh:** JWTs have a configurable expiration period, and a refresh mechanism is implemented to renew authentication when needed.

3.2.4. Performance Consideration:

- **System Responsiveness:** The system shall promptly respond to all user actions, including loading and processing, within a reasonable timeframe. This responsiveness is a fundamental aspect of user satisfaction, and we will continuously assess and monitor it as a key indicator of the system's success.

- **Concurrent User Support:** The system shall support multiple concurrent users without compromising performance or responsiveness. The exact number of concurrent users supported may vary depending on system configuration and infrastructure.

- **Load Capacity:** The system shall be capable of handling typical user loads without significant performance degradation. Under extreme load conditions, the system may experience increased response times. Scalability considerations should be implemented to address potential heavy usage scenarios.

3.2.5. Error Handling and Validation:

Exception Handling:

- **Exception Handling Coverage:** The system shall prioritize handling critical exceptions, including server errors, client errors, validation errors, database connectivity issues, and authentication/authorization failures.

- **Error Response Consistency:** The system shall provide clear, descriptive, and consistent error messages or codes for each exception type, enabling developers and users to quickly identify and resolve issues.

- **Error Resolution Guidance:** The system shall provide appropriate guidance or links to documentation for resolving common error scenarios.

Environment Considerations:

- **Basic Functionality Maintenance:** Under adverse conditions, the system shall maintain its core functionalities, such as user authentication, data access, and basic application operations.

- **Error Resilience:** The system shall demonstrate error resilience by gracefully handling exceptions and providing appropriate error messages or notifications to users and administrators.

Authorization Consideration:

- **JWT Integration:** The system shall integrate JWTs into its authentication and authorization processes to provide secure user identification and access control.

- **Authorization Granularity:** The system shall enforce granular access control based on user roles to restrict user access to specific resources and functionalities.

Validation Strategies:

- **Frontend Validation:** The system shall incorporate frontend validation libraries or frameworks compatible with Next.js to validate user inputs in real-time, providing immediate feedback and preventing invalid data submission.

- **Backend Validation:** The system shall implement backend validation mechanisms using Nest.js validation of user inputs, ensuring data consistency and integrity before storage or processing.

- **Database Schema Validation:** The system shall leverage MongoDB's schema validation capabilities to enforce data structure and integrity constraints at the database level, preventing invalid data from being stored in the database.

3.2.6. Quality Issues:

System Reliability, Availability, and Robustness:

- **Reliability:**

- **Mean Time Between Failures (MTBF):** The system shall strive for an MTBF of at least 80.%, minimizing the occurrence of unplanned outages and service disruptions.

- **Mean Time to Repair (MTTR):** The system shall aim for an MTTR of no more than 30 minutes for critical failures, ensuring prompt recovery and service restoration.

❖ **Availability:**

- **System Uptime:** The system shall target an uptime of at least 85%, maximizing the time during which users can access and utilize system functionalities.

- **Downtime Minimization:** Downtime shall be minimized through proactive maintenance, redundancy measures, and effective incident management procedures.

❖ **Robustness:**

- **Error Handling: Resource Management:** The system shall efficiently manage system resources, such as CPU, memory, and network bandwidth, to prevent performance degradation under The system shall implement comprehensive error handling mechanisms to gracefully handle unexpected exceptions and errors, preventing cascading failures and maintaining system stability.

- **Load Balancing:** The system shall employ load balancing techniques to distribute incoming requests across multiple servers, ensuring optimal performance and responsiveness under high traffic conditions.

Client Involvement in Quality Assessment:

❖ **Assessment Plan:** A structured assessment plan shall be established to outline the client's role in evaluating system quality attributes, including usability, functionality, performance, and adherence to business objectives.

- **Quality Evaluation Participation:** Clients shall be actively involved in evaluating the system's quality attributes, providing feedback on usability testing, functionality demonstrations, performance benchmarks, and adherence to business requirements.

- **Development Process Feedback:** Clients shall have the opportunity to provide feedback on the development process, including adherence to timelines, requirements fulfillment, and overall project alignment with their expectations.

- **Actionable Feedback:** Client feedback shall be treated with the utmost importance, and actionable steps shall be taken to address valid concerns, incorporate improvements, and maintain client satisfaction.

3.2.7. Backup and Recovery:

- Backup Execution: We employ MongoDB Atlas's built-in backup feature for continuous, point-in-time backups.
- Recovery Time Objective: Our Recovery Time Objective (RTO) is clearly defined to ensure minimal downtime and disruption to system availability.

3.2.8. Physical Environment:

Deployment Location:

- The system must be hosted on a cloud platform.

External Factors:

- **Device Compatibility:** Users must be able to access the system from different devices (desktops, laptops, tablets, smartphones), and the system should be compatible with a range of screen sizes and resolutions.

- **Browser Compatibility:** The system must function correctly across various web browsers (Chrome, Firefox, Safari, Edge) and their different versions.

- **Browser and Technology Updates:** The system must remain compatible with the latest browsers and frameworks if there are updates and changes in web technologies.

- **Policy Changes:** If there are any changes in government policies related to internships, education, or employment the system must be updated to comply with the new regulations.

3.2.9. Resource Issues:

Resource Constraints: Since the system is accessible through web browsers, it places minimal constraints on the system's resource consumption, as most processing occurs server-side rather than on the client hardware.

3.2.10. Documentation:

Document Level:

At the system documentation level, we aim to include user documentation for end-users and development process documentation for the development team.

User Documentation:

User Manuals:

- **Interns:**

  - **Onboarding Guide:** The system shall provide a comprehensive guide for interns to onboard onto the system which includes information on account creation, profile setup, and an overview of available features.

  - **Task Management:** The system shall explain how interns can use the system to manage their tasks and submit reports.

- **Supervisors:**

- ➢ **Supervisor Dashboard:** The system should specify the features available in the supervisor dashboard, including task assignment, progress tracking, and feedback submission.

- ➢ **Evaluation Process:** It should provide instructions on how supervisors can evaluate interns, submit feedback, and generate reports.

- ❖ **Administrators:**

  - ➢ **Data Management:** The user documentation should explain procedures for data management, including adding and updating internship programs, managing user accounts, and handling system-wide configurations.

Development Process Documentation:

- ❖ **Code Documentation:** We aim to establish and document a code documentation standard to ensure that all code includes clear and concise comments.

- ❖ **Design Principles:** Our objective is to document design principles that will guide the architectural decisions and overall design approach of the system.

- ❖ **Version Control Practices:** We will aim to cover procedures for creating branches, merging code, handling conflicts, and ensuring a stable and traceable version history in our documentation.

CHAPTER 4

SYSTEM ANALYSIS

In this chapter, we tried to discuss use case models, object models, and dynamic models.

4.1. System Model

In this project, we developed abstract system models using use case modeling to detail user interactions and system functionalities.

4.1.1. Use Case Model

4.1.1.1. Use Case Diagram

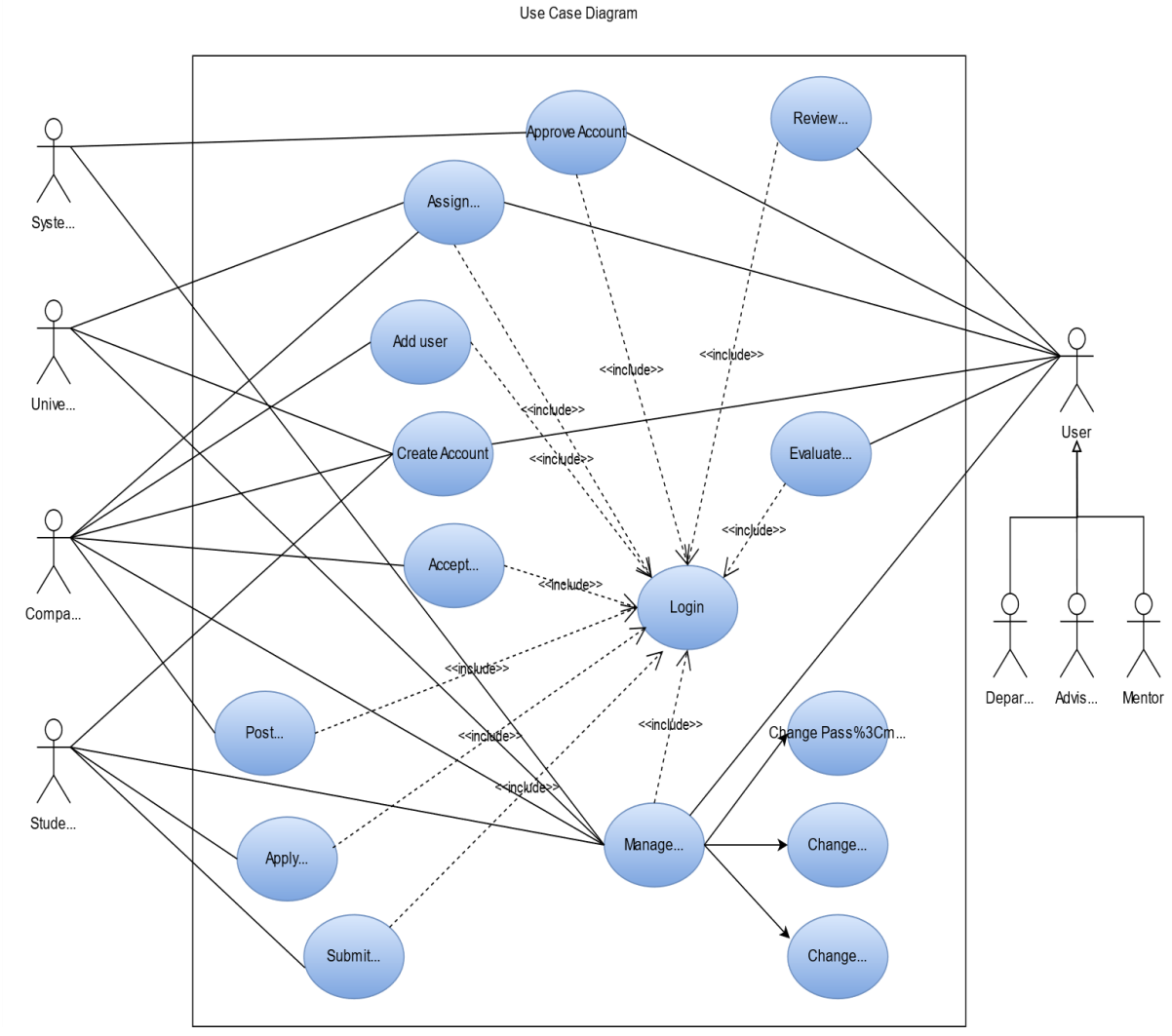


Figure 4. 1 Use case Diagram 1

4.1.1.2 Use Case Description

The following tables show use case descriptions based on the system business or context of our system

Student Registration

Table 4. 1 Student Registration

Use Case Name	User Registration
Use Case Identifier	UC001
Actor(s)	Student
Description	This use case describes the process by which a student registers a new user account within the system
Pre-condition	The student has access to the system but is not yet registered with a user account.
Post-condition	The student successfully creates a new user account and gains access to the system.

1. The student navigates to the User Registration page on the Internship Management System.
2. The system presents a registration form with fields for the necessary details, such as username, password, personal information, education, skills, and preferences.
3. The student fills in the required information in the registration form.
4. The system validates the entered data, checking for completeness and accuracy.
5. If the validation is successful, the system proceeds to the next step. If there are validation errors, the system prompts the student to correct the information.

6. The student reviews the entered information to ensure accuracy and completeness.
7. The student submits the registration form.
8. The system processes the registration request and creates a new user account for the student.
9. The system generates a unique identifier for the new user account.
10. The student receives a confirmation message indicating successful registration and provides instructions for further actions, such as account activation.
11. Upon successful activation, the student gains access to their profile and the Internship Management System.

Profile Management

Table 4. 2 Profile Management

Use Case Name	**Profile Management**
Use Case Identifier	UC002
Actor(s)	Student
Description	This use case describes the process by which a student manages and updates their user profile within the system.
Pre-condition	The student is logged into the system and has an existing user account.
Post-condition	The student successfully updates their profile information, and the changes are reflected in the system.

1. Student logs in and accesses the dashboard.
2. Chooses "Profile Management."
3. Views current profile details.
4. Selects sections to edit.
5. Edits profile information in the presented form.

6. System validates entered data.
7. Submits edited profile.
8. Receives confirmation of successful update.
9. Optionally reviews the updated profile.
10. Navigates back or explores other features.

Internship Search

Table 4. 3 Profile Internship Search

Use Case Name	**Internship Search**
Use Case Identifier	UC003
Actor(s)	Student
Description	This use case describes the process by which a student searches for internships based on various criteria within the Internship Management System.
Pre-condition	The student is logged into the system and has an active user account.
Post-condition	The student views a list of relevant internships based on their search criteria.

1. Student logs in and accesses the dashboard.
2. Selects "Internship Search."
3. Defines search criteria.
4. System retrieves relevant internships.
5. Reviews internship details.
6. Refines search or selects a specific opportunity.
7. Views detailed information about the chosen internship.
8. Completes the application process if interested.
9. Receives confirmation of application submission.
10. Optional: Tracks application status.
11. Navigates back or explores other features.

Application Submission

Table 4. 4 Application Submission

Use Case Name	Application Submission
Use Case Identifier	UC004
Actor(s)	Student
Description	This use case describes the process by which a student submits an application for a specific internship within the Internship Management System.
Pre-condition	The student has successfully searched and selected an internship from the system.
Post-condition	The student receives confirmation of the successful submission of the internship application.

1. Student logs in and accesses the dashboard.
2. Selects "Application Submission."
3. Views eligible internships.
4. Chooses a specific opportunity.
5. Fills out a tailored application form.
6. Validates entered data and documents.
7. Reviews and submits the application.
8. System updates application status.
9. Receives confirmation of submission.
10. Optional: Tracks application status.
11. Navigates back or explores other features.

Company Registration

Table 4. 5 Company Registration

Use Case Name	Company Registration
Use Case Identifier	UC005
Actor(s)	Company HR
Description	This use case outlines the process by which a company's HR registers the company on the Internship Management System.
Pre-condition	The Company HR has access to the system but is not yet registered with a user account.
Post-condition	The company is successfully registered in the system.

1. The Company HR initiates the company registration process.
2. The system prompts the Company HR to provide necessary details, including company name, industry, and contact information.
3. The Company HR enters the required information and submits the registration form.
4. The system validates the entered data.
5. If the validation is successful, the system registers the company and assigns a unique identifier.
6. The Company HR receives a confirmation message indicating successful registration.

Internship Posting

Table 4.6 Internship Posting

Use Case Name	Internship Posting
Use Case Identifier	UC006
Actor(s)	Company HR
Description	This use case outlines the process by which a Company HR posts new internship opportunities within the Internship Management System.
Pre-condition	The Company HR is authenticated and the company is registered in the

	system.
Post-condition	A new internship opportunity is successfully posted in the system.

1. The Company HR initiates the internship posting process.
2. The system presents a form for the Company HR to input details about the internship, including job descriptions, requirements, duration, and application instructions.
3. The Company HR completes the form and submits the internship posting.
4. The system validates the entered data.
5. If the validation is successful, the system posts the internship opportunity.
6. The Company HR receives a confirmation message indicating successful posting.

Performance Evaluation

Table 4. 7 Performance Evaluation

Use Case Name	**Performance Evaluation**
Use Case Identifier	UC007
Actor(s)	Company Mentor
Description	This use case outlines the process by which a Company Mentor evaluates the performance of interns during and after the internship within the Internship Management System.
Pre-condition	The Company Mentor is authenticated, the company is registered, and there are interns who have completed or are undergoing internships.
Post-condition	The performance of interns is successfully evaluated and recorded in the system.

1. The Company Mentor initiates the performance evaluation process.
2. The system presents a form or platform for the Company Mentor to assess various aspects of the intern's performance, such as skills, work quality, and professionalism.
3. The Company Mentor completes the evaluation form and submits it.

4. The system records the evaluation in the intern's profile.
5. Optionally, the system may notify the intern of the evaluation results.

The Company Mentor receives confirmation messages after successfully evaluating intern performance.

University Registration

Table 4. 8 University Registration

Use Case Name	**University Registration**
Use Case Identifier	UC008
Actor(s)	University admin
Description	This use case outlines the process by which a University Admin registers their university on the Internship Management System.
Pre-condition	The University Admin has access to the system but is not yet registered with a user account.
Post-condition	The university is successfully registered in the system.

1. The University Admin initiates the university registration process.
2. The system presents a form for the University Admin to input details about the university, including name, location, and contact information.
3. The University Admin completes the form and submits the registration.
4. The system validates the entered data.
5. If the validation is successful, the system registers the university and assigns a unique identifier.
6. The University Admin receives a confirmation message indicating successful registration.

Adding Department

Table 4. 9 Adding Department

Use Case Name	Adding Department
Use Case Identifier	UC009
Actor(s)	College Dean
Description	This use case outlines the process by which a College Dean adds a new department within the Internship Management System.
Pre-condition	The College Dean is authenticated, the college is registered, and there is a need to add a new department.
Post-condition	A new department is successfully added to the system under the college.

1. The College Dean initiates the process of adding a new department.
2. The system presents a form for the College Dean to input details about the new department, such as name, specialization, and contact information.
3. The College Dean completes the form and submits the addition of the new department.
4. The system validates the entered data.
5. If the validation is successful, the system adds the new department under the college and assigns a unique identifier.
6. The College Dean receives a confirmation message indicating a successful addition.

Advisor Assignment

Table 4. 10 Advisor Assignment

Use Case Name	Advisor Assignment
Use Case Identifier	UC0010

Actor(s)	Department Head
Description	This use case outlines the process by which a Department Head assigns advisors to students within the Internship Management System.
Pre-condition	The Department Head is authenticated, the department is registered, and there are students requiring advisor assignment.
Post-condition	Advisors are successfully assigned to students in the department.

1. The Department Head initiates the advisor assignment process.
2. The system presents a list of students in need of advisor assignment within the department.
3. The Department Head selects advisors for each student or approves advisor assignments made by advisors themselves.
4. The system updates the advisor assignment for each student.
5. Optionally, the system may notify students and advisors of the assignment results.
6. The Department Head receives confirmation messages after successfully managing advisor assignments.
7. The Department Head accesses the approval section in the system.

4.1.1.3. Use case Scenario

Use Case Scenario: User Registration (UC001)

Title: Student Registration

Actor: Abel (Student)

Goal: To create a new user account and gain access to the Internship Management System.

Scenario:

1. Abel Visits the Registration Page: Abel accesses the Internship Management System's website and navigates to the "Register" or "Sign Up" section.

2. Filling Registration Form: Abel enters required details like username, password, personal information (name, email, etc.), academic details, skills, and preferences into the registration form.

3. Validation and Submission: The system validates the entered data, ensuring completeness and accuracy. Abel reviews the details and corrects any errors indicated by the system.

4. Confirmation and Account Creation: Once all details are accurate and complete, Abel submits the registration form. The system processes the request, generates a unique identifier, and creates Abel's account.

5. Confirmation Message: Abel receives an on-screen confirmation message notifying successful registration and providing instructions for further steps, including account activation.

6. Account Activation (Optional): Abel may receive a verification email to activate the account. Following the instructions in the email, Abel activates the account, gaining access to the Internship Management System.

Title: Profile Management **(UC002)**

Actor: Abel (Student)

Goal: Update personal details in the Internship Management System.

Preconditions:

1. Abel is logged into the Internship Management System.

2. Abel intends to modify existing profile information.

Main Flow:

1. Login and Access Profile: Abel logs into the system using valid credentials and accesses profile-related options.

2. Profile Management: From the dashboard, Abel navigates to the "Profile Management" section.

3. View and Edit Profile: Abel sees the current profile details, including personal information, education, skills, and preferences. Abel selects the sections to modify.

4. Editing Profile: Abel chooses the sections requiring updates and makes necessary changes to the profile information.

5. Validation and Submission: The system validates the modified data for completeness and accuracy. If successful, the system proceeds to process Abel's submission.

6. Processing Updates: Upon successful validation, the system saves the modifications to Abel's profile.

7. Confirmation (Optional): Abel receives an optional on-screen confirmation message indicating the successful update of the profile.

8. Navigation: Abel, having completed the profile update, can navigate back to the dashboard or explore other functionalities within the system.

Use Case Scenario: Internship Search (UC003)

Title: Internship Search

Actor: Abel (Student)

Goal: To discover and potentially apply for suitable internships within the Internship Management System.

Preconditions:

1. Abel is logged into the Internship Management System with an active user account.

2. Abel intends to explore available internships to find suitable options.

Main Flow:

1. Login and Access Dashboard: Abel logs into their account on the Internship Management System. The system presents Abel's dashboard, featuring an option for "Internship Search."

2. Navigating to Internship Search: Abel selects the "Internship Search" functionality from the dashboard.

3. Search Criteria Input: The system displays a comprehensive search interface with various filters (industry, location, duration, skills, etc.). Abel inputs desired search criteria that best align with their internship preferences and career goals.

4. Search Results: The system processes Abel's entered criteria and retrieves a list of matching internships. Abel reviews the list comprising detailed job descriptions, requirements, and company information.

5. Refining Search and Review: Abel has the option to further refine search parameters or sort the results based on personal preferences. Abel selects a specific internship to explore more detailed information.

6. Detailed Internship Review: The system displays comprehensive details about the selected internship. Abel thoroughly reviews the specifics to ensure alignment with personal preferences and career aspirations.

7. Application Process: If interested, Abel chooses to apply for the internship. The system prompts Abel to submit necessary documents and information for the application.

8. Application Submission: Abel completes the application process by providing required details as prompted by the system. The system confirms the successful submission of Abel's application.

9. Optional Tracking (Post-Application): Optionally, the system may offer an application tracking feature for Abel to monitor the status of their application.

10. Navigation Back: Abel has the flexibility to return to the dashboard or explore other functionalities within the system after completing the internship search and application process.

Use Case Scenario: Application Submission (UC004)

Title: Application Submission

Actor: Abel (Student)

Goal: To successfully submit an application for a specific internship within the Internship Management System.

Preconditions:

1. Abel is logged into the Internship Management System with an active user account.
2. Abel has navigated to the application section after selecting a desired internship.

Main Flow:

1. Login and Access Dashboard: Abel logs into their account on the Internship Management System. The system presents Abel's dashboard, which includes the "Application Submission" option.
2. Navigating to Application Submission: Abel selects the "Application Submission" section from the dashboard.
3. Selecting Internship and Application Form: The system displays a list of available internship opportunities. Abel chooses a specific internship they intend to apply for.
4. Application Form and Data Entry: The system presents an application form with required fields and options for document uploads. Abel fills in all necessary details within the form and uploads essential documents as required.

5. Validation and Review: The system conducts validation checks on the entered information and uploaded documents. If validation is successful, Abel reviews the entire application to ensure accuracy and completeness.

6. Application Submission: Abel submits the completed application form through the system.

7. Confirmation and Status Update: The system records the submitted application and updates its status to "Submitted" for the selected internship. Abel receives a confirmation message indicating the successful submission of their application.

8. Optional Application Tracking: Optionally, the system provides Abel with an application tracking feature to monitor the status of their application.

9. Navigation Back: After completing the application submission process, Abel has the flexibility to return to the dashboard or explore other functionalities within the system.

Use Case Scenario: Company Registration (UC005)

Title: Company Registration & Account Creation

Actor: Mr. Abebe (Company HR)

Goal: To register the company while creating a personal account on the Internship Management System.

Preconditions:

1. Mr. Abebe aims to create a personal account.

2. The Company Mr. Abebe represents needs to be registered on the system.

Main Flow:

1. Initiating Account Creation and Company Registration: Mr. Abebe begins the account creation process, intending to register the company simultaneously.

2. Filling Personal Details: The system prompts Mr. Abebe to provide personal details for account creation. Simultaneously, the system requests essential company details for registration.

3. Entering Account Information: Mr. Abebe enters his personal details, including name, email, and password, to create his account.

4. Providing Company Details: Alongside personal information, Mr. Abebe inputs the necessary company details:
    a. Company name
    b. Industry type
    c. Contact information (address, phone, email)
    d. Optional additional details if required by the system.

5. Submitting Account and Company Information: Mr. Abebe submits both the personal account and company registration information.

6. Validation of Entered Data: The system validates the entered data for completeness, accuracy, and adherence to required formats.

7. Successful Validation:

    a. Upon successful validation:
        - The system creates Mr. Abebe's personal account.
        - Registers the company within the system database.
        - Generates a unique identifier for the company.
        - Builds a comprehensive company profile based on the provided details.
        - Mr. Abebe receives confirmation of successful account creation and company registration.

Use Case Scenario: Company Mentor Evaluation (UC006)

Actor: Company Mentor (Mr. Nahom)

Goal: To evaluate the performance of interns during or after their internship within the Internship Management System.

Preconditions:

1. Mr. Nahom, the Company Mentor, is authenticated and logged into the system.
2. The company is registered in the system.
3. There are interns who have completed or are currently undergoing internships.

Main Flow:

1. Initiating Evaluation: Mr. Nahom initiates the performance evaluation process for a specific intern.

2. Evaluation Criteria: The system presents a comprehensive evaluation form/platform covering various aspects of intern performance.

3. Providing Detailed Evaluation: Mr. Nahom carefully assesses each criterion, providing detailed feedback, ratings, and comments.

4. Submitting Evaluation: Mr. Nahom completes the evaluation form entirely and submits it to the system. The system securely records the evaluation in the intern's profile.

5. Communication of Results (Optional): The system may notify the intern of evaluation results, granting access to feedback and ratings.

6. Confirmation of Evaluation: Mr. Nahom receives confirmation messages from the system, indicating successful evaluation completion.

Use Case Scenario: University Registration (UC007)

Actor: University Admin (Ebisa)

Goal: To register the university on the Internship Management System.

Preconditions: Ebisa, the University Admin, is authenticated and logged into the system.

Main Flow:

1. Initiating Registration: Ebisa initiates the university registration process within the system.

2. Completing Registration Form: The system presents a registration form requiring
    a. University name
    b. Location details (address, city, state, country)
    c. Contact information (phone number, email address)
    d. Official website URL
    e. Accreditation details (optional)
    f. Other pertinent information (e.g., student population, degree programs)

    Ebisa fills out the form, ensuring accuracy and completeness of the provided information.

3. Submitting Registration Form: After completing the form, Ebisa submits the registration form.

4. Validating Entered Data: The system performs validation checks on the entered data for consistency and correctness.

5. Upon successful validation: The system registers the university in its database. A unique identifier is assigned to the university. A university profile is created containing the provided information. Ebisa receives a confirmation message indicating the successful registration.

Use Case Scenario: Adding Department (UC008)

Actor: College Dean (Melaku)

Goal: To add a new department within the Internship Management System, under the existing college.

Preconditions:

1. Melaku, the College Dean, is authenticated and logged into the system.

2. The college is already registered in the system.

Main Flow:

1. Initiating Department Addition: Melaku accesses the relevant section in the system to add a new department.

2. Entering Department Details:

    a. The system presents a form prompting for department details, including:
        - Department name
        - Head or chair of the department (optional)
        - Specialization or area of focus
        - Contact information (phone number, email address)
        - List of faculty members (optional)
        - List of offered courses (optional)

    b. Melaku carefully completes the form, ensuring accuracy and completeness.

3. Submitting Form: Melaku submits the form to initiate the department addition process.

4. Validation: The system validates entered data for consistency and correctness.

5. Successful Addition: If validation is successful:
    a. The system adds the new department under the associated college.
    b. Assigns a unique identifier to the department.
    c. Creates a profile with provided information.
    d. Melaku receives a confirmation message indicating successful addition.

Use Case Scenario: Advisor Assignment (UC009)

Actor: Department Head (John)

Goal: To efficiently assign advisors to students within the department in the Internship Management System.

Preconditions:

1. John, the Department Head, is authenticated and logged into the system.
2. The department is registered within the system.
3. There are students within the department requiring advisor assignment.

Main Flow:

1. Initiating Advisor Assignment: John, as the Department Head, accesses the system's advisor assignment section specific to their department.

2. Reviewing Students for Assignment: The system displays a comprehensive list of students needing advisor assignment, presenting:
    a. Student name
    b. Year of study
    c. Major/specialization
    d. Current advisor status (if applicable)

3. Advisor Assignment Process: For each student, John proceeds with either of the following approaches:

    a. Direct Assignment: John selects an available advisor from a list of faculty members within the department. The system verifies the chosen advisor's availability and qualifications for the assignment.

    b. Approving Self-Assignment: The system identifies students with a preferred advisor choice. John reviews student preferences and advisor availability. John approves or suggests an alternative advisor based on availability and qualifications.

4. Assignment Confirmation: The system records the advisor assignment for each student, ensuring proper documentation. Optionally, the system notifies students and assigned advisors regarding the assignment outcome through email or in-system notifications. John, as the Department Head, receives confirmation messages signaling the successful completion of the advisor assignment process.

4.2. Object Model

Within this section, we present a comprehensive overview of the object model.

4.2.1. Class Diagram

In this subsection, our aim is to visually represent the conceptual connections among objects and classes. This includes excluding details such as data and member visibility, associations, generalizations, cardinality, and other related aspects in order to provide a simplified yet informative diagrammatic illustration.

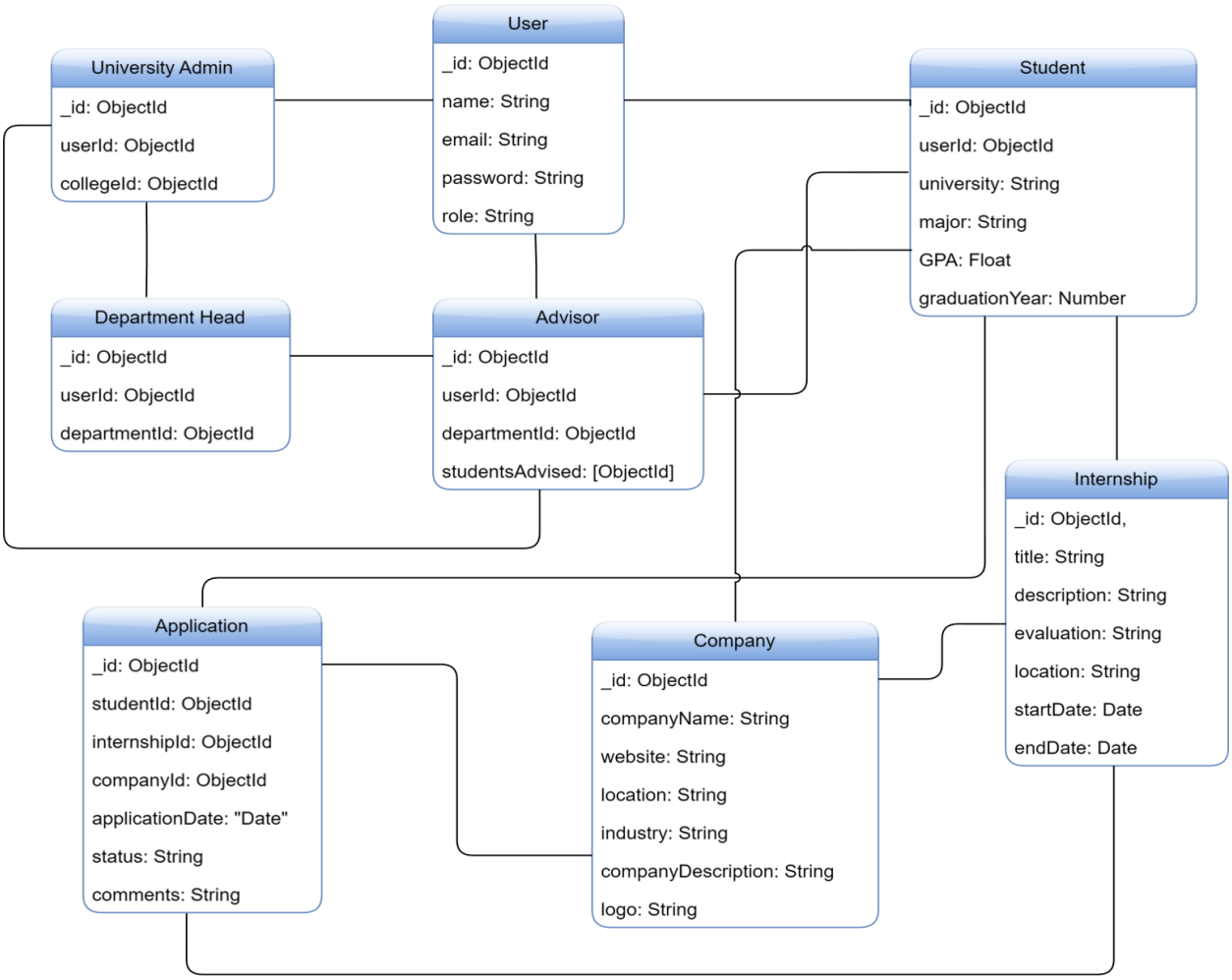


Figure 4. 2 Class Diagram

4.2.2. Data Dictionary

Here's the data dictionary for our internship management system, broken down into multiple tables for optimized storage and querying in MongoDB:

Table: Users

Table 4. 5 Users

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier	User ID index
Name	String	-	-	Name index
Email	String	unique	Valid email address	Email index
Password	String	hashed and salted	Securely stored password	-
Role	Enum: Student, Employer, Mentor, UniversityAdmin	-	User role	Role index

Table: Students (inherits from Users)

Table 4. 6 Students

Attribute	Data Type	Key Constraints	Constraints	Indexes
University	ObjectId	foreign key references Universities._id	Reference to university	University and student ID compound index
Major	String	-	-	Major index
Skills	Array of Objects	-	-	List of skill objects with name and proficiency level

Preferences	Array of Objects	-	-	List of preference objects with industry, location, etc.
Documents	Array of Objects	-	-	List of document objects related to the student

Table: Document

Table 4. 7 Document

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier	User ID index
Document Name	String	-	-	Document name/title
Type	Enum: CV, Portfolio, Transcript, Certificate, Letter of Recommendation, Other	-	-	-
Submission Date	Datetime	-	Date of document submission	-
Content	Binary	-	Binary data of the document content	-
Size	Integer	-	File size in bytes	-
Format	String	-	File format or extension (e.g., PDF, DOCX, etc.)	-
Description	String	-	Brief description or notes about the document	-

Table: Company

Table 4. 8 Company

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier	User ID index
Company Name	String	-	-	Company name index
Description	String	-	-	Brief company description

Table: HR (inherits from Users, Company)

Table 4. 9 Human Resource

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier	User ID index
Company Name	String	-	-	Company name index
Description	String	-	-	Brief company description

Table: Mentors (inherits from Users, Company)

Table 4. 10 Mentor

Attribute	Data Type	Key Constraints	Constraints	Indexes
Title	String	-	-	Mentor title/position

Company ID	ObjectId	foreign key references Employers._id	Reference to the associated company	Company and mentor ID compound index
Assigned Intern	ObjectId (optional)	foreign key references Students._id	Reference to the assigned intern (optional)	Assigned intern and mentor ID compound index

Table: Internships

Table 4. 28Internships

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier	Internship ID index
Title	String	-	-	Internship title index
Description	String	-	-	Internship description
Industry	String	-	-	Industry category
Location	String	-	-	Internship location
Duration	Number	-	-	Internship duration in months
Start Date	Date	-	-	Start date range index
Skills Required	Array of Objects	-	-	List of required skill objects with name and proficiency level
Company ID	ObjectId	foreign key references Employers._id	Reference to the company	Company and internship ID compound index

Table: Applications

Table 4. 11 Application

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier	Application ID index
Student ID	ObjectId	foreign key references Students._id	Reference to the student	Student and application ID compound index
Internship ID	ObjectId	foreign key references Internships._id	Reference to the internship	Internship and application ID compound index
Status	Enum: Submitted, Shortlisted, Interviewed, Offered, Rejected	String	-	Application status
Date Applied	Datetime	-	-	Submission date range index
Cover Letter	String	-	-	Text of submitted cover letter
Resume	String	-	-	Text of submitted resume
Feedback	String	-	-	Student feedback after internship completion

Table: University

Table 4. 30 University

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	Primary Key	Unique identifier	University ID index
Name	String			University name index

Table: College

Table 4. 31 Collage

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	Primary Key	Unique identifier	College ID index
Name	String			College name index
University	ObjectId	Foreign Key	References University._id	University and college ID compound index

Table: Departments

Table 4. 32 Department

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier for department	Department ID index
Name	String	-	Department name	Department name index
College	ObjectId	Foreign key references Colleges._id	Reference to college	College and department ID compound index

Table: Department Heads (inherits from Users)

Table 4. 33 Department Head

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier for head	Department head ID index
Name	String	-	Head's name	Head name index

Department	ObjectId	Foreign key references Departments._id	Reference to department	Department and head ID compound index

Table: Advisors (inherits from Users)

Table 4. 34 Advisors

Attribute	Data Type	Key Constraints	Constraints	Indexes
ID	ObjectId	_id (Primary Key)	Unique identifier for advisor	Advisor ID index
Name	String	-	Advisor's name	Advisor name index
Department	ObjectId	Foreign key references Departments._id	Reference to department	Department and advisor ID compound index

4.3 Dynamic Model

The dynamic model captures the system's time-sensitive elements, focusing on how the states of system objects evolve over time. This section aims to record the actions and interactions of the entity model using sequence, activity, and state chart diagrams.

4.3.1. Sequence Diagram

4.3.2. Activity Diagram

Internship process:

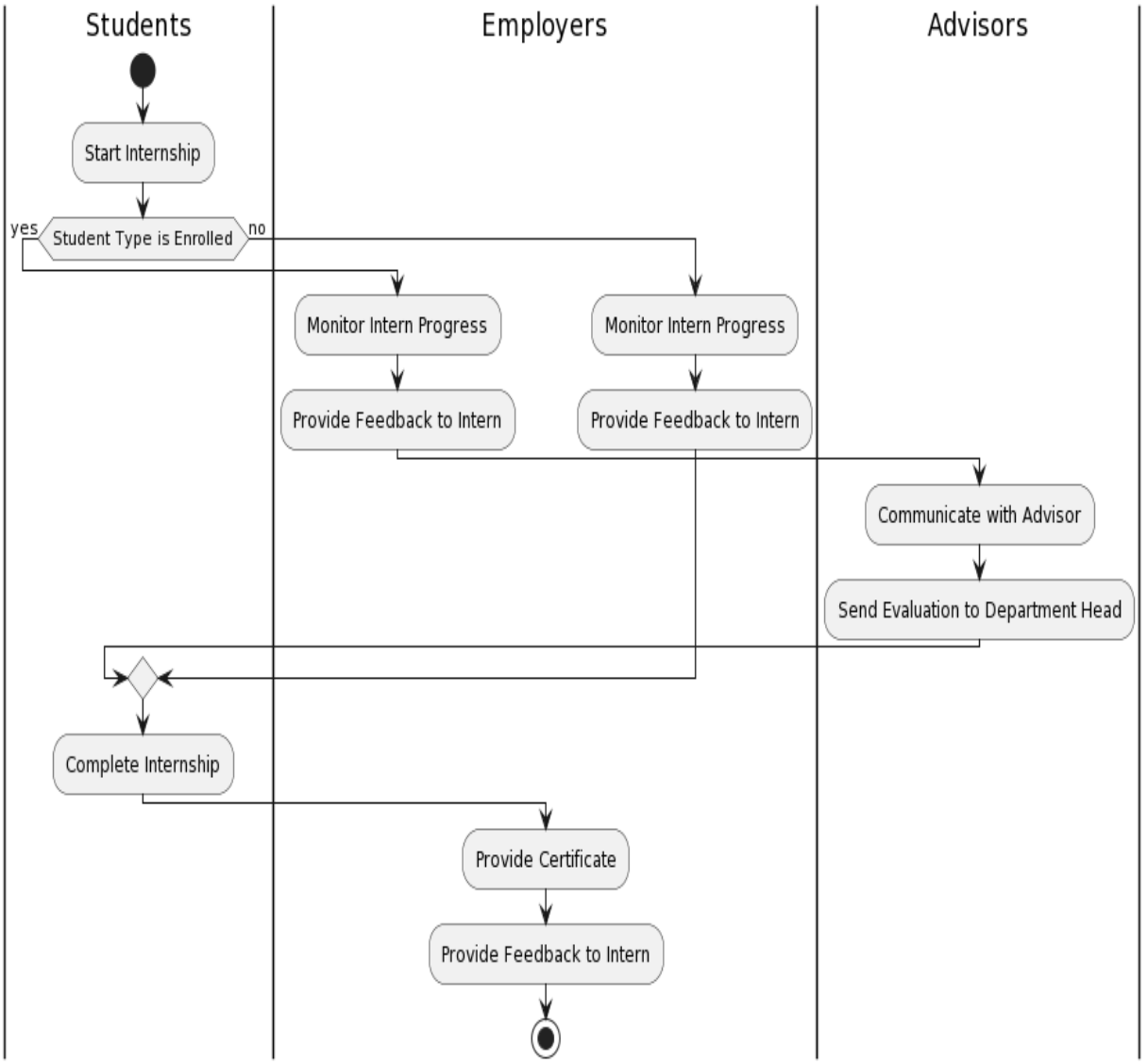


Figure 4. 3 Activity Diagram (Internship Process)

Student Validation:

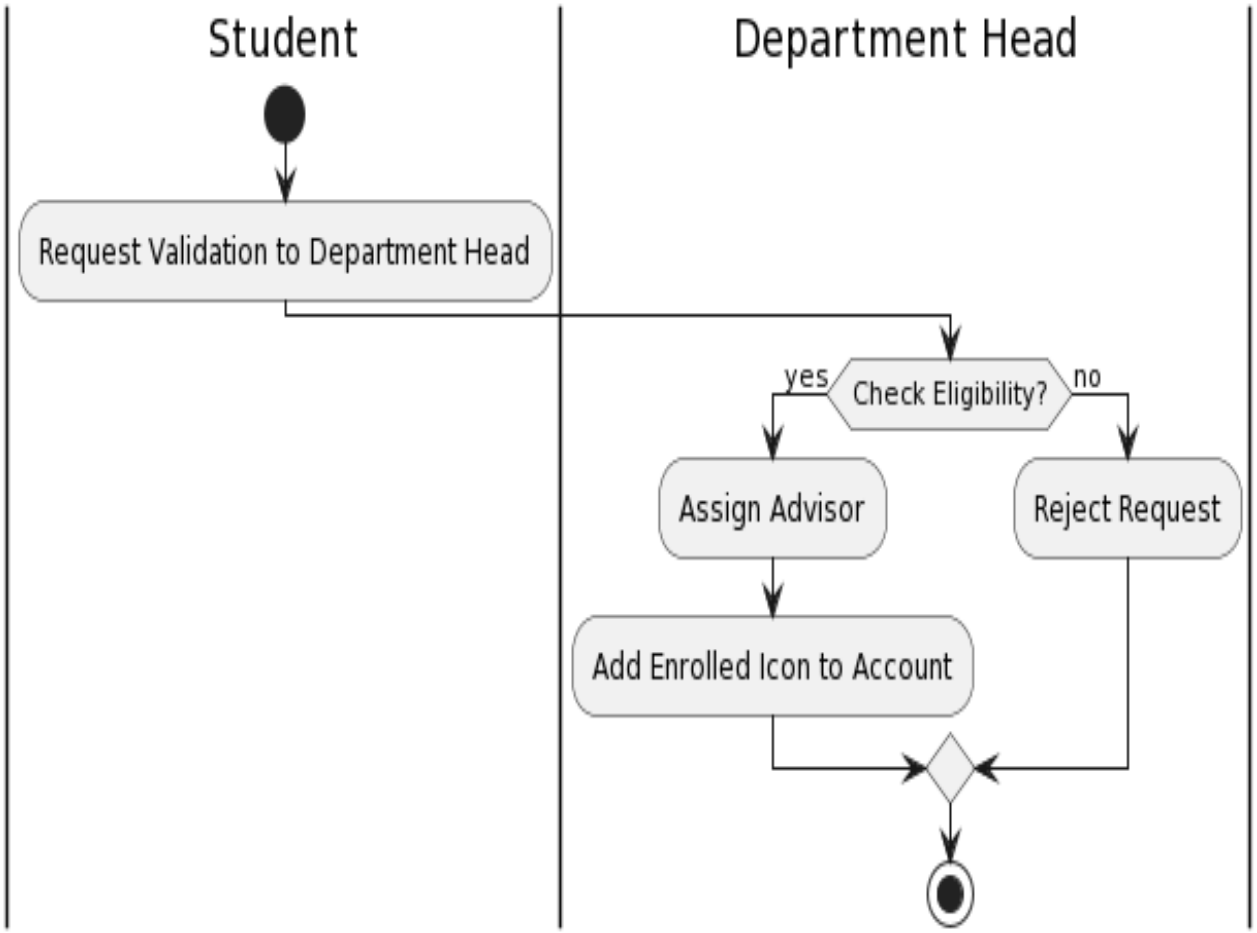


Figure 4. 4 Activity Diagram (Student Validation)

Apply Application:

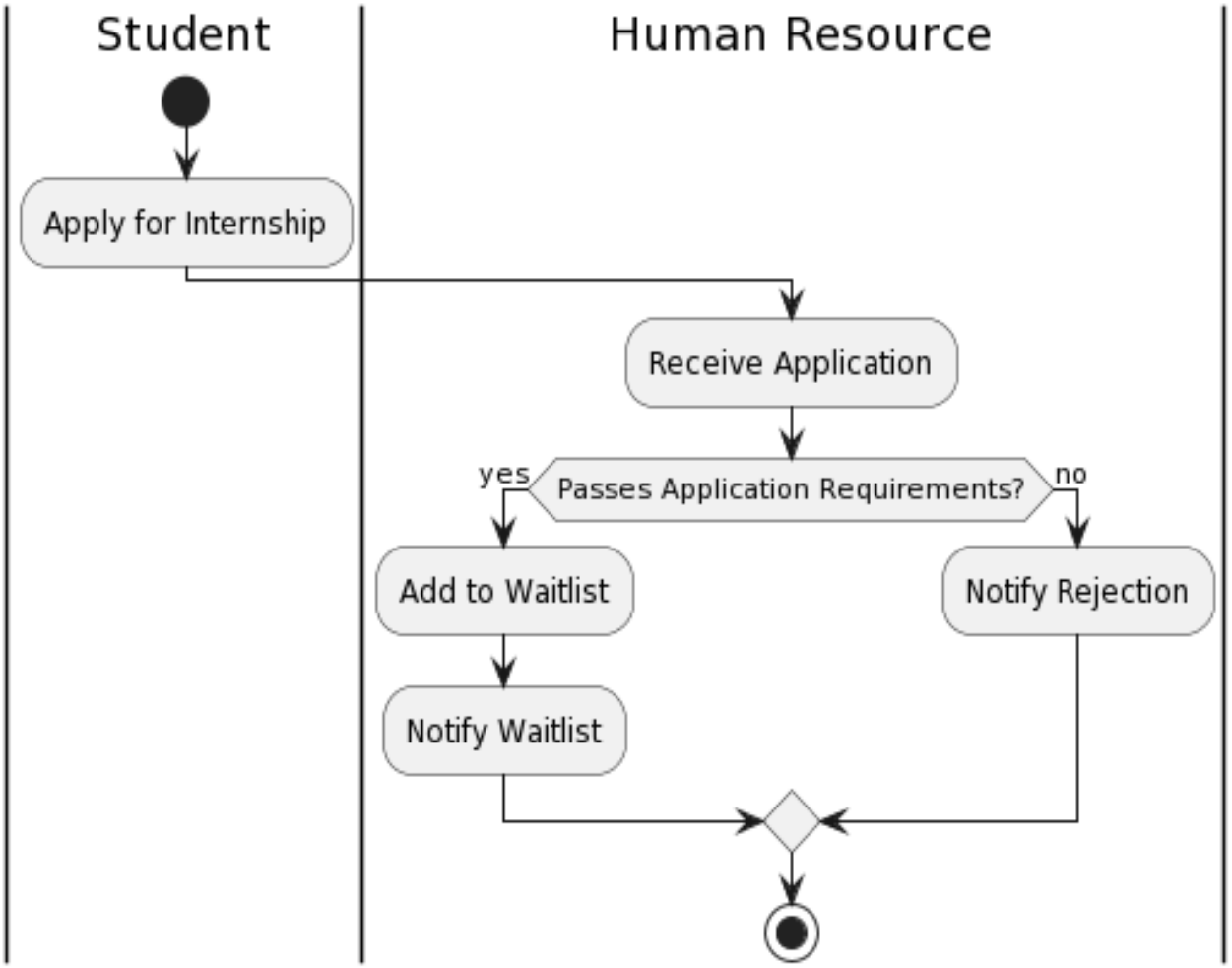


Figure 4. 5 Activity Diagram (Apply Application)

Interview process:

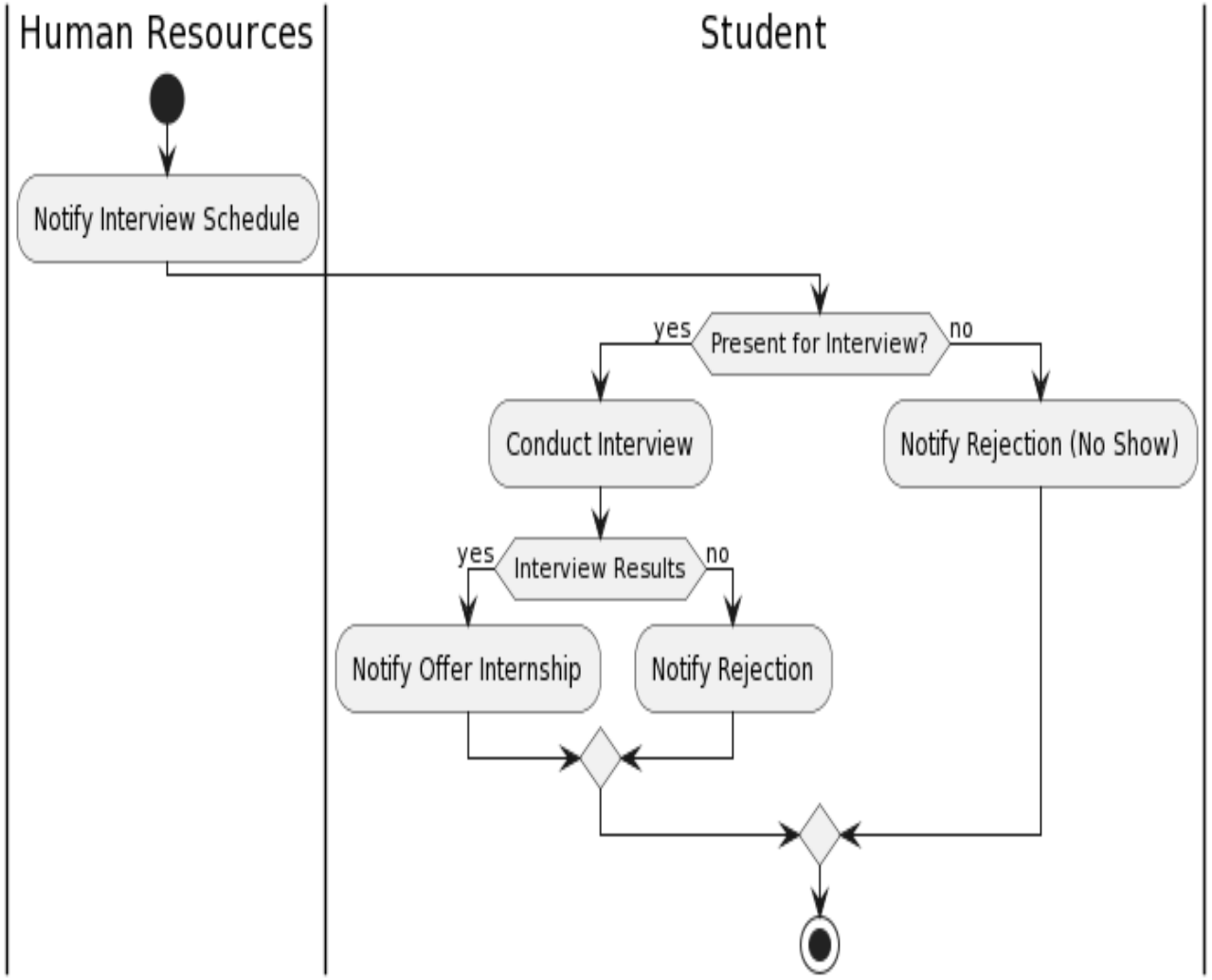
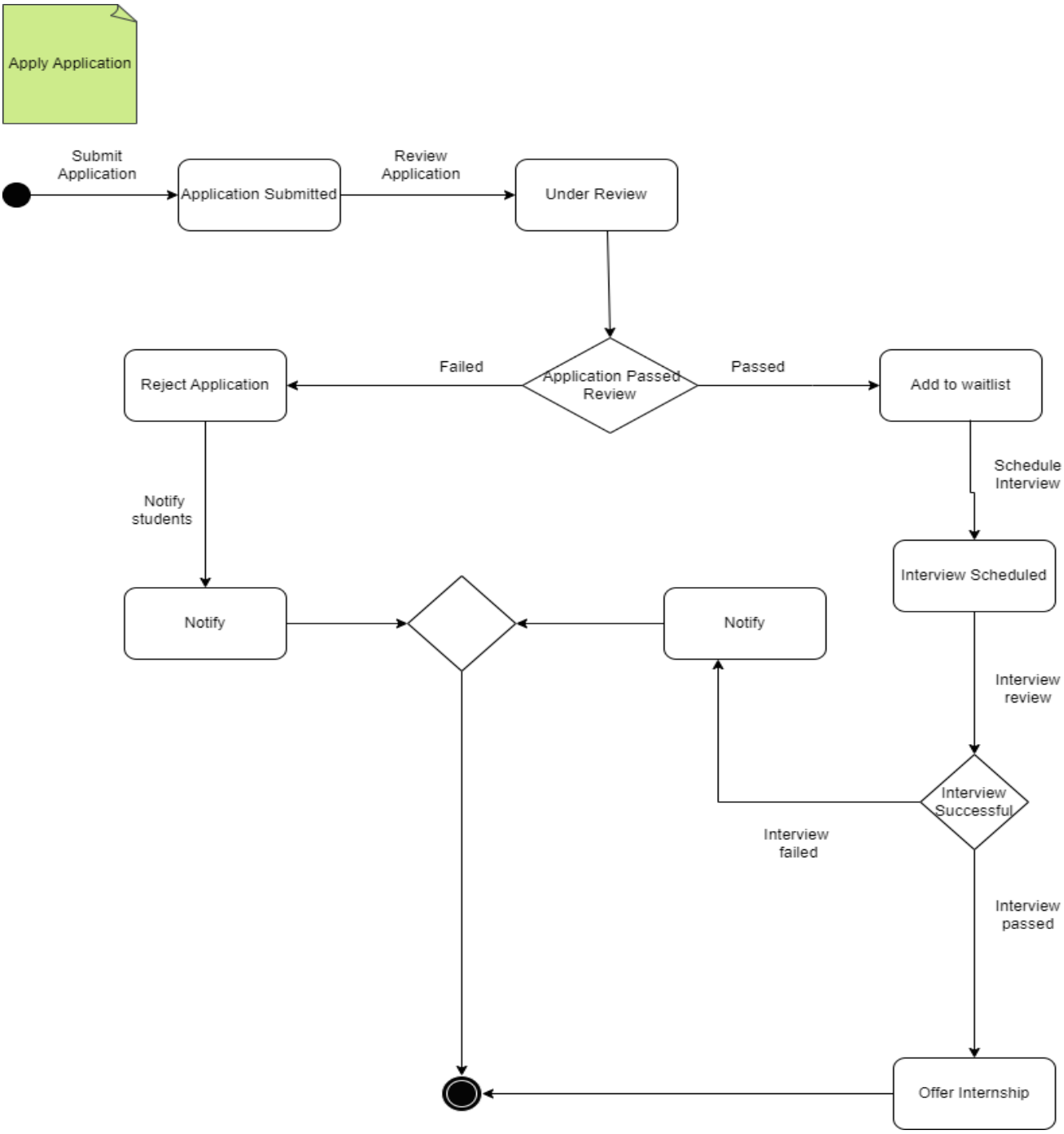


Figure 4. 6 Activity Diagram (Interview Process)

4.3.3 State Chart Diagram



Internship progress tracking process:

Figure 4. 7 State Diagram(Internship Progress Tracking)

Internship Process:

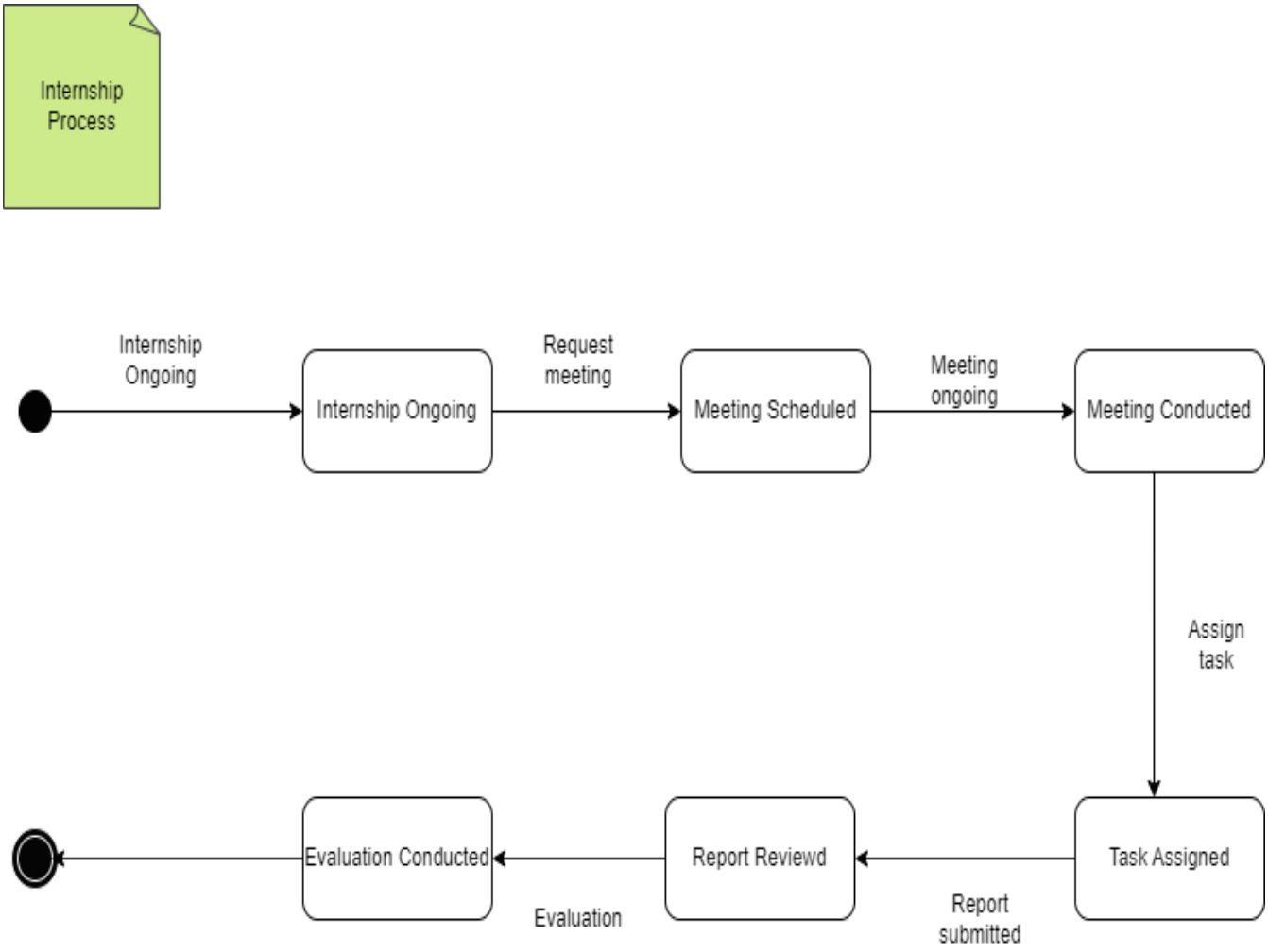


Figure 4. 8 State Diagram(Internship Process)

Approve Student:

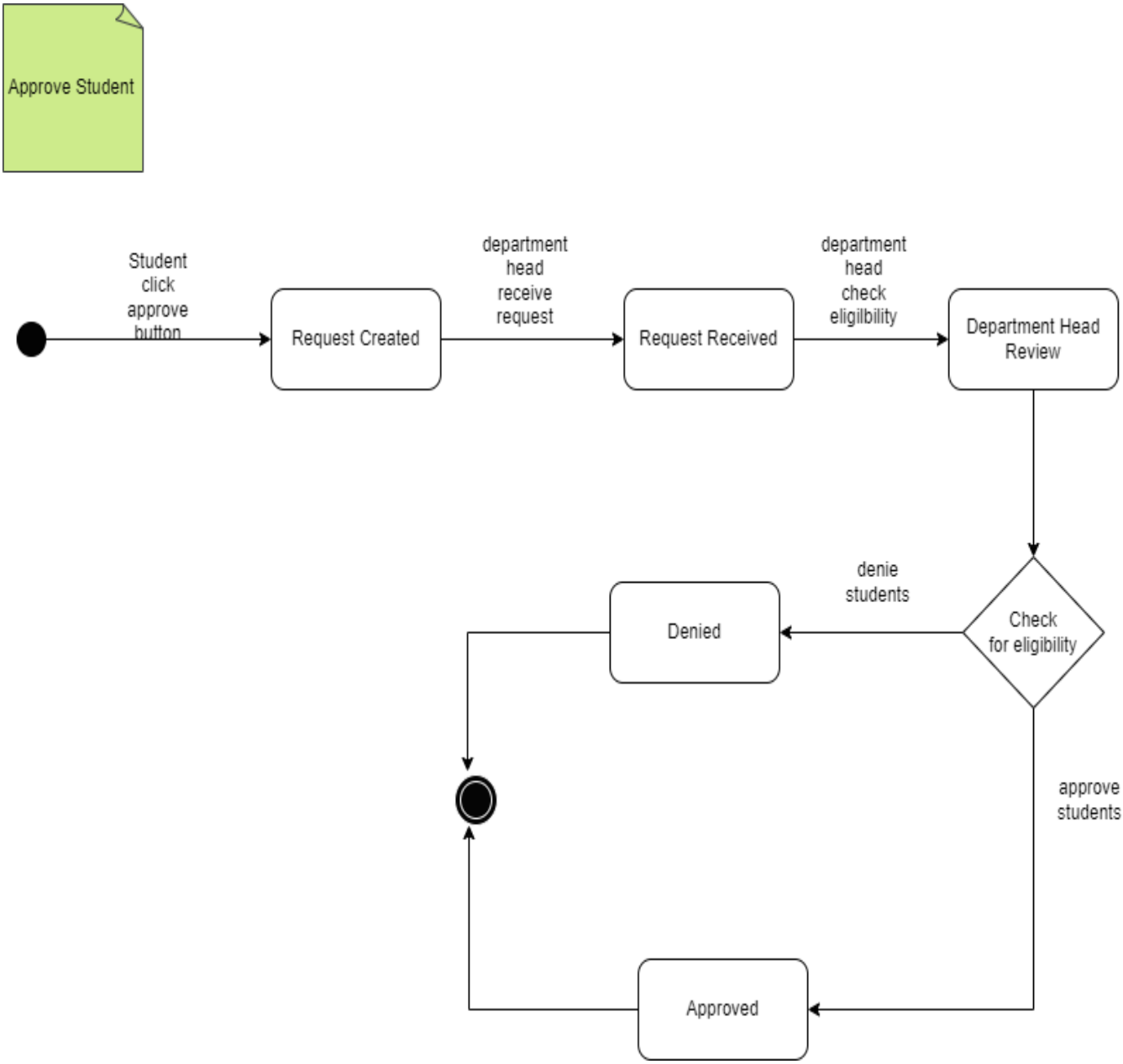


Figure 4. 9 State Diagram (Approve Student)

CHAPTER FIVE

SYSTEM DESIGN

5.1. Design Goals

The Design Goals specify the qualities of the Web-based Internship Management System that should be achieved and addressed during the design and implementation phase.

1. User Interface and Human Factors

Goals:

- **User-Friendly Interface:** We're committed to creating a seamless experience. We'll collaborate with potential users, seeking their feedback to refine and improve the interface.
- **Responsive Design:** We'll adopt a mobile-first approach, making the system adaptive and responsive through media queries and flexible layouts.

2. Hardware Consideration

Goals:

- **Scalability:** We're designing with growth in mind, using a client-server architecture to scale horizontally. Regular checks will help us adjust resources as needed.

3. Security Issues

Goals:

- **Presentation Tier:**

  JWT for Authentication:

  - We implement JWT-based authentication to ensure secure user login sessions.
  - User roles and permissions are included in JWT claims for effective authorization.

- **Application Tier:**

Argon2 for Password Hashing:

- We use Argon2 for robust and adaptive password hashing, prioritizing the protection of user credentials.
- Secure password policies are implemented to further enhance our security measures.

JWT Validation:

- We validate incoming JWTs to guarantee the integrity and authenticity of user requests.
- Short-lived expiration times are set in JWTs to limit the duration of user sessions.

- **Data Tier:**
  - **Securely Store Hashed Passwords:**
    - Hashed passwords using Argon2 are securely stored in our database, bolstering protection against unauthorized access.
    - Additionally, we apply encryption to safeguard sensitive data stored in the database.

4. Performance Consideration

Goals:

- **Optimizing MongoDB:** Regularly analyze the usage of indexes in MongoDB using tools like explain (). Ensure that queries utilize indexes effectively.
- **Asynchronous Processing**: we'll use asynchronous processing for non-blocking operations to enhance system responsiveness.
- **Horizontal Scaling:** we'll consider horizontal scaling by distributing the application across multiple servers or containers to handle increased load.
- **Lazy Loading:** We implement lazy loading for non-essential resources, loading them only when needed, which speeds up initial page loading.

5. Error Handling and Validation

Goals:

- **Client-Side Validation:** We implement client-side validation zod form validation. This provides immediate feedback to users and reduces the likelihood of invalid data being submitted.
- **Server-Side Validation:** We implement server-side validation in Nest.js controllers or services using validation pipes. This ensures that only valid data is processed on the server
- **Database Constraints:** Enforce data integrity at the database level using constraints (e.g., unique constraints, foreign key constraints) to prevent invalid data from being stored.

6. Quality Issues

Goals:

- **Code Quality:** Through code reviews, linting tools, and adhering to coding style guides, we'll maintain high-quality code. Continuous integration (CI) will automate code quality checks.
- **Testing and QA:** Our robust testing suite will cover unit, integration, and end-to-end tests, all supported by reliable automated testing tools.

7. Backup and Recovery

Goals:

- We'll try to create a detailed plan, outlining steps to take during system failures and assigning responsibilities clearly.
- **Database Restore:** we'll use the mongorestore command-line tool provided by MongoDB.

8. Physical Environment

Goals: For the hosting infrastructure, we'll opt for a reliable and cost-effective cloud provider, implementing load balancing strategies to handle traffic effectively.

9. Documentation

Goals: If time permits, we will explore adding a help desk feature to better assist users with any questions or issues they may have.

5.2. Proposed System Architecture

We've chosen a three-tier architecture for our web-based internship management system for the following reasons:

1. **Scalability:** Enables independent scaling of presentation, application, and data tiers to handle growing user loads.
2. **Modularity and Maintainability:** Separation of presentation, business logic, and data layers ensures easier maintenance and updates.
3. **Flexibility and Adaptability:** Allows for updating or replacing components without impacting other tiers, supporting agility in technology adoption.
4. **Clear Division of Responsibilities:** Facilitates organized development, testing, and troubleshooting with distinct roles for each tier.
5. **Improved Security:** Implements security measures at each tier, providing defense in depth.
6. **Interoperability:** Supports seamless interaction with various application servers and databases, enhancing system integration.
7. **Enhanced Performance:** Distribution across multiple servers improves overall performance with load balancing strategies.
8. **Support for Different Client Types:** Accommodates various client types, ensuring optimal user experiences on different devices.
9. **Centralized Management and Updates:** Simplifies updates, maintenance, and version control with centralized application logic and data.
10. **Better Resilience and Fault Isolation:** Enhances system reliability by isolating faults or issues to specific tiers.

Presentation Tier

The Presentation Tier displays information related to the services or operations within our system. It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network

Application Tier

The Application Tier (middle tier) controls the business logic and processes based on Service calls from the Presentation Tier (User Interface). The Application Tier is separate from the presentation tier and, as its own layer, controls an application's functionality by performing detailed processing.

Data Access Tier

This tier consists of database servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers. Giving data its own tier also improves scalability and performance.

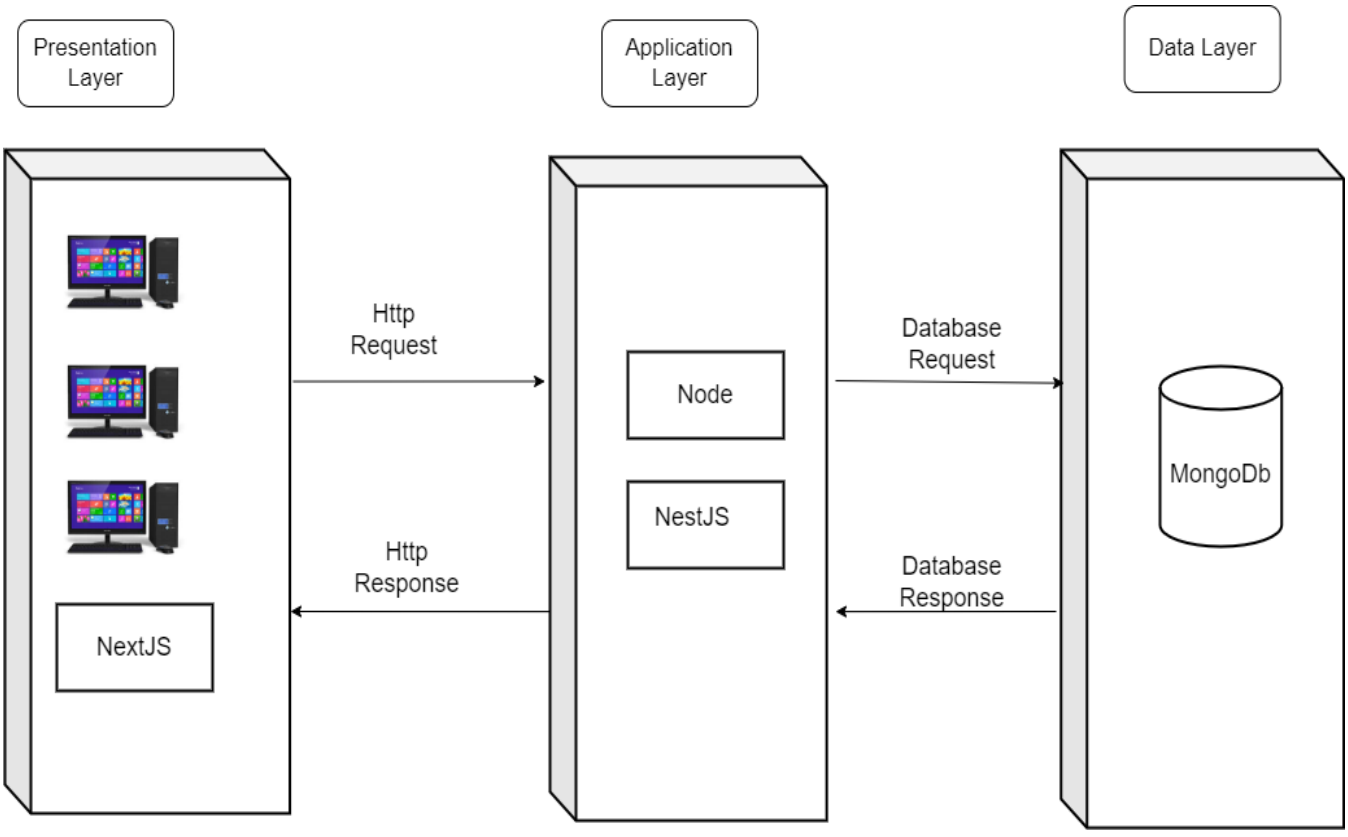


Figure 5. 1 Proposed System Architecture

5.2.1. Subsystem Decomposition and Description

The component diagram's main purpose is to show the structural relationships between the components of a system. UML component diagrams are great for identifying the architectural landscape for Our system as they enable us to model the high-level software components, and more importantly the interfaces to small components. we decomposed in to different components such as: -

1. User Management Subsystem:

- **Description:** Handles user authentication, registration, and profile management for administrators, employers, coordinators, and interns.
- **Components:**
  - Authentication Module
  - User Registration Module
  - Profile Management Module

2. Internship Posting Subsystem:

- **Description:** Manages the process of posting internship opportunities by employers.
- **Components:**
  - Internship Posting Interface
  - Application Processing Module
  - Notifications Module

3. Application Management Subsystem:

- **Description:** Facilitates the submission, review, and processing of internship applications.
- **Components:**
  - Application Submission Module
  - Application Review Module
  - Status Notification Module

4. Progress Tracking Subsystem:

- **Description:** Monitors and tracks the progress of interns during their internship.
- **Components:**

- Progress Tracking Interface
- Evaluation and Feedback Module

5. Reporting and Analytics Subsystem:

- **Description:** Generates reports and analytics for administrators and coordinators.
- **Components:**
  - Report Generation Module
  - Analytics Module
  - Data Visualization Interface

6. Document Management Subsystem:

- **Description:** Handles the storage and retrieval of internship-related documents.
- **Components:**
  - Document Repository Module
  - Document Upload and Download Interface

7. Security Subsystem:

- **Description**: Ensures the security and integrity of the system.
- **Components:**
  - Authentication and Authorization Module
  - Data Encryption Module

8. System Administration Subsystem:

- **Description:** Provides tools and functionalities for system administrators to manage the overall system.
- **Components:**
  - User Management Interface
  - System Configuration Module

❖ Maintenance and Updates Module

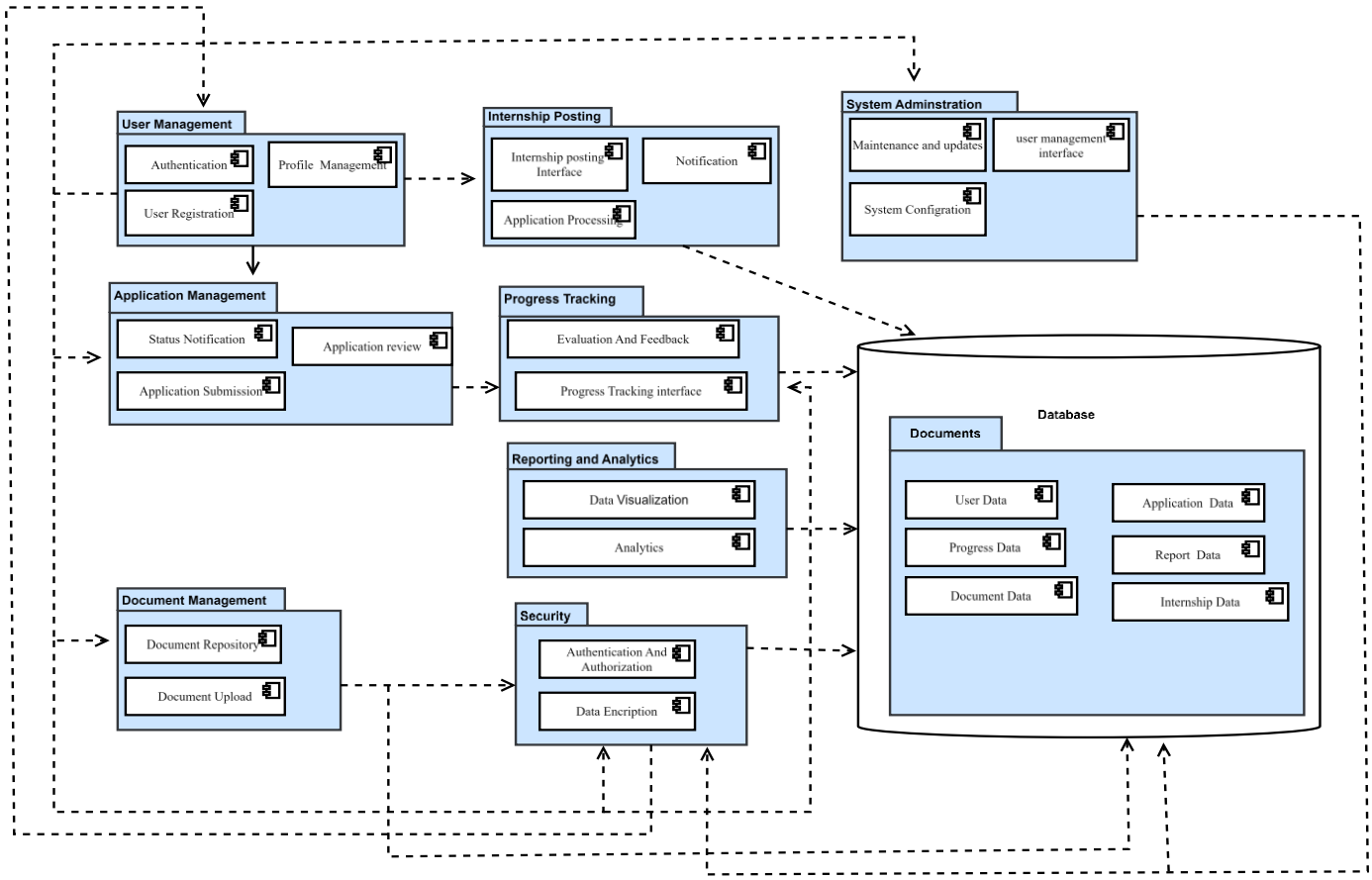


Figure 5. 2 Component Diagram

5.2.2. Hardware/Software Mapping

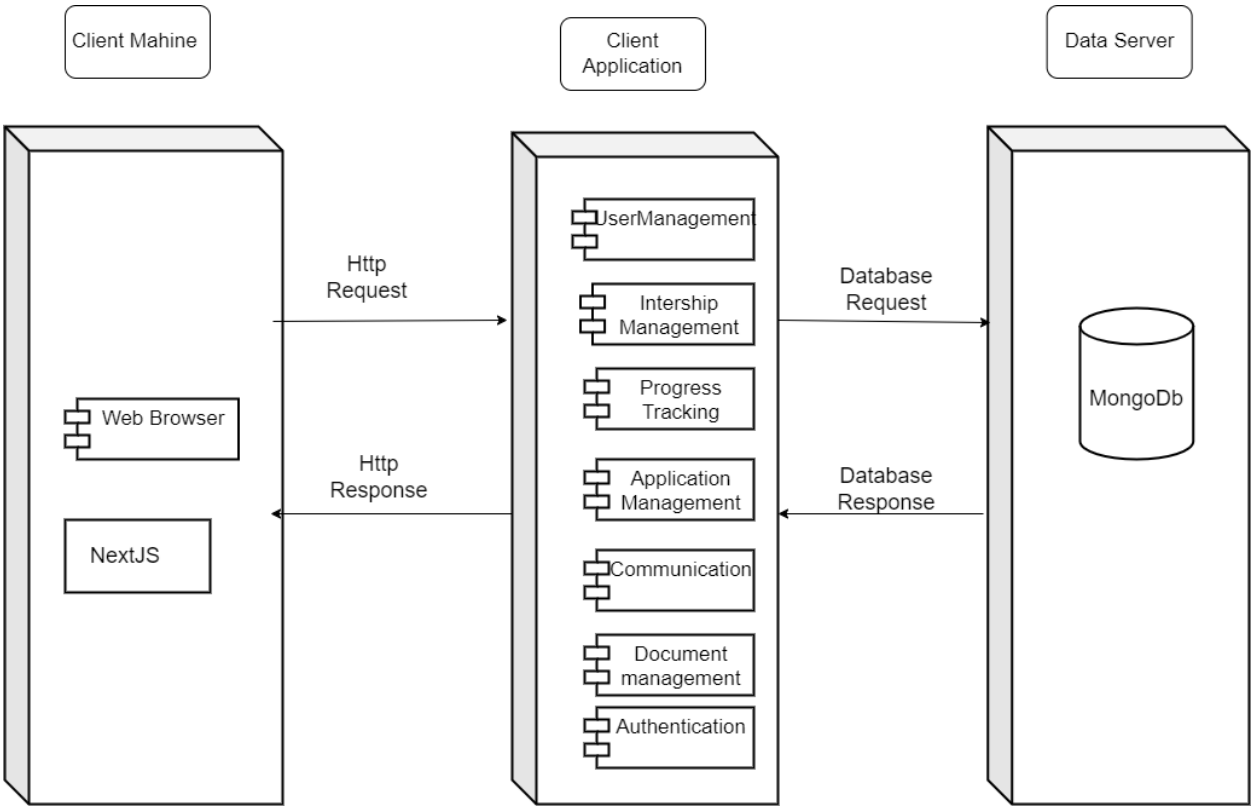


Figure 5. 3 Hardware /Software mapping

5.2.3 Detailed Class Diagram

The diagram captures three main classes: Users, Students, and Documents. Users hold basic identification and registration data, while Students encapsulate academic specifics like university, GPA, skills, and associated documents. Documents store details of uploaded files, such as type and content. Relationships are depicted as Users connected to Students in a one-to-one association, while Students can have multiple Documents in a one-to-many correlation, reflecting their various uploaded files.

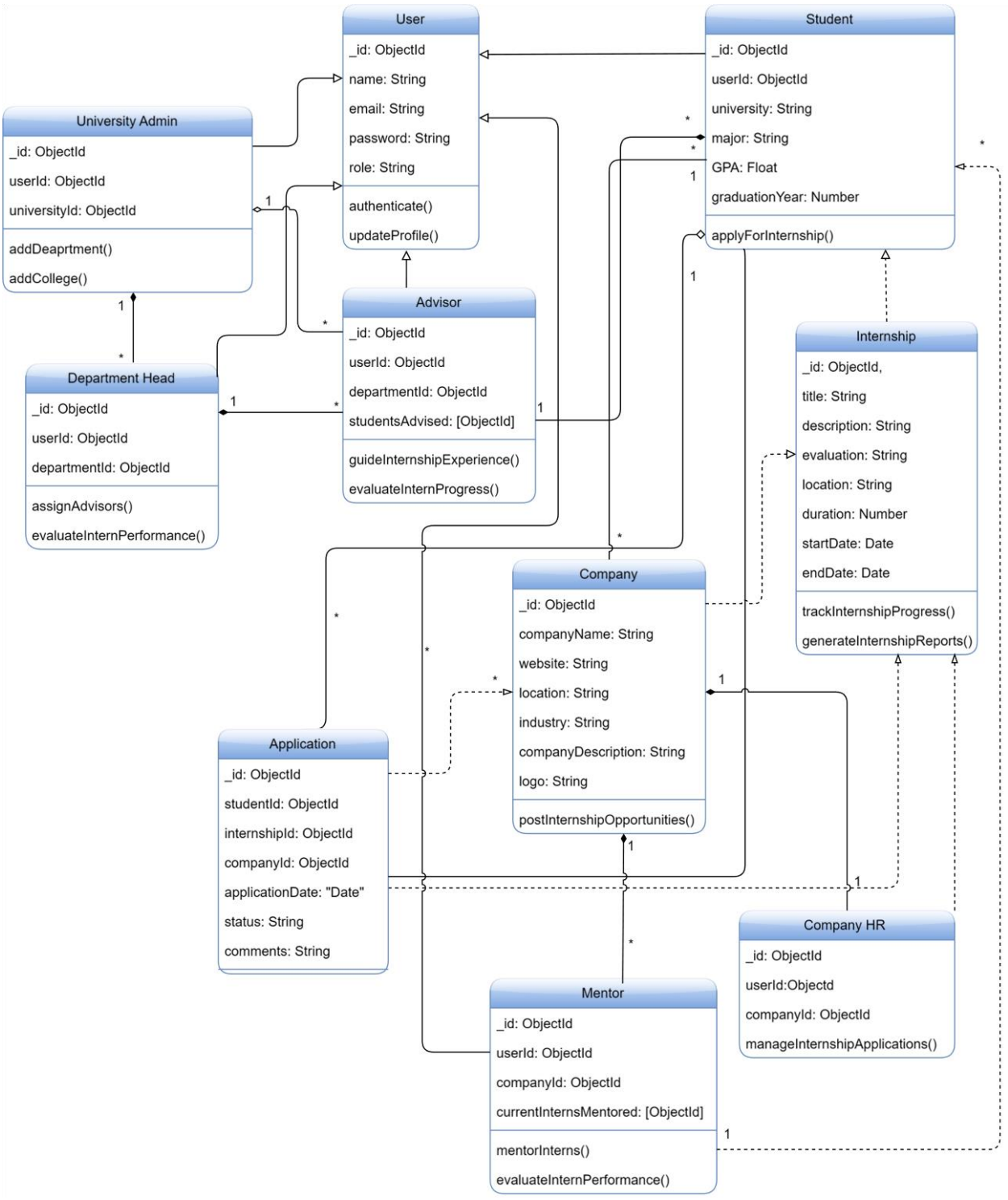


Figure 5. 4 Detailed Class Diagram

5.2.4 Persistent Data Management

The Persistent Data Management system in the web-based internship platform structures and handles critical data like user details, internship records, and document information. It's powered by MongoDB, a flexible NoSQL database, organizing data through distinct schemas for users, companies, internships, and more. We have tried to depict it below.

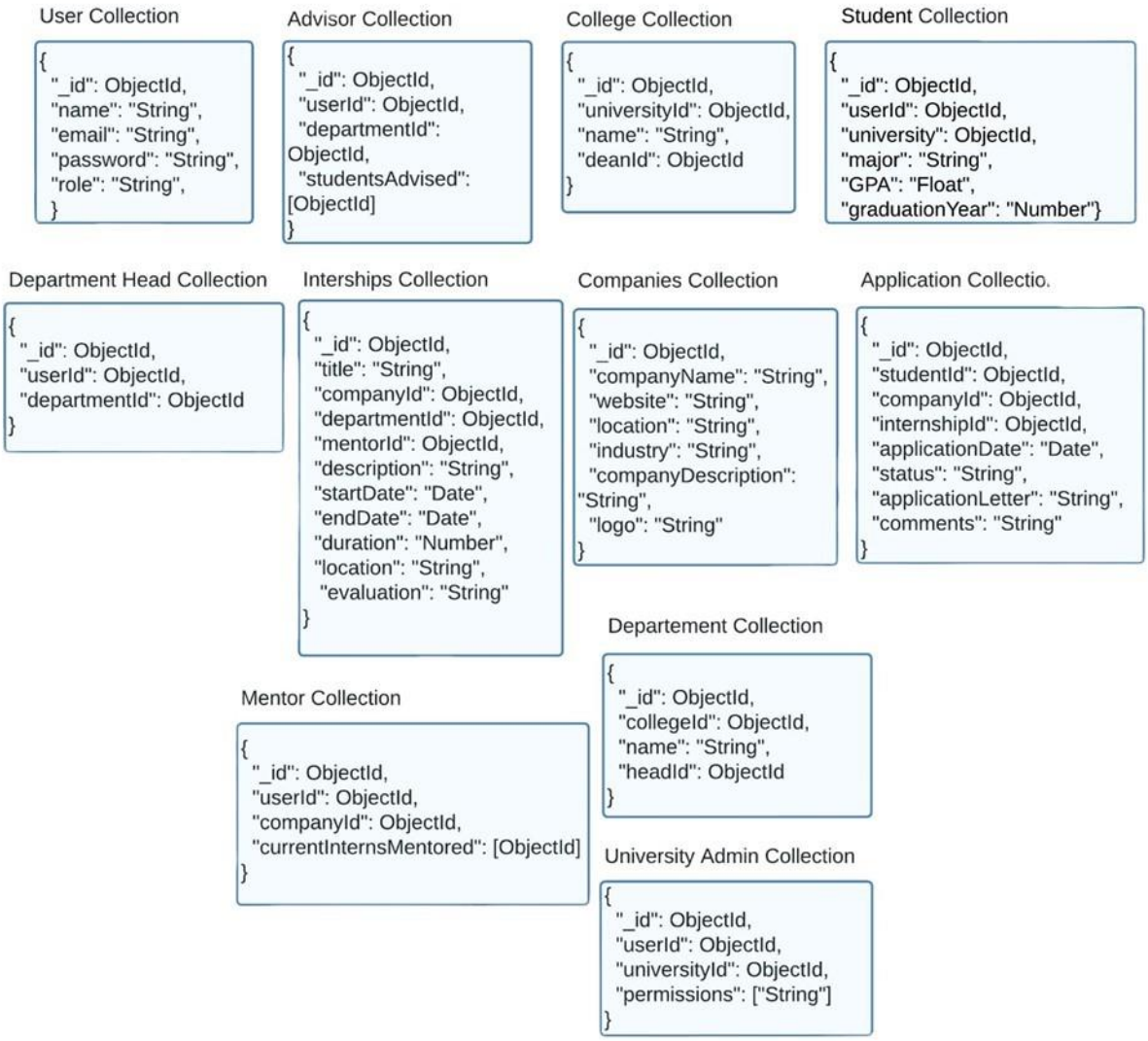


Figure 5. 5 Persistent Data Diagram

5.2.5. Access control and Security

Table 5. 1 Access Control Privileges

Actor	Access Control
Student	View available internship opportunities. Apply application View application status and feedback. Access personal profile information.
Company HR	Post new internship opportunities. Update details of posted internship opportunities. View applications submitted by students.
Company Mentor	View progress reports of interns. Evaluate students
Advisor	View progress reports of interns.
Department Head	View overall internship data for the department. View progress reports of interns within the department. Approve interns students.
University Admin	Access to all functionalities and data about the university. Manage user accounts and roles in the university. Generate comprehensive reports and analytics. Initiate communication to companies
System Admin	Manage system configurations and settings. Ensure system integrity and security. Manage all accounts

5.3. Packages

Package diagram shows the decomposition of subsystems into packages and overview of each package dependencies with other packages.

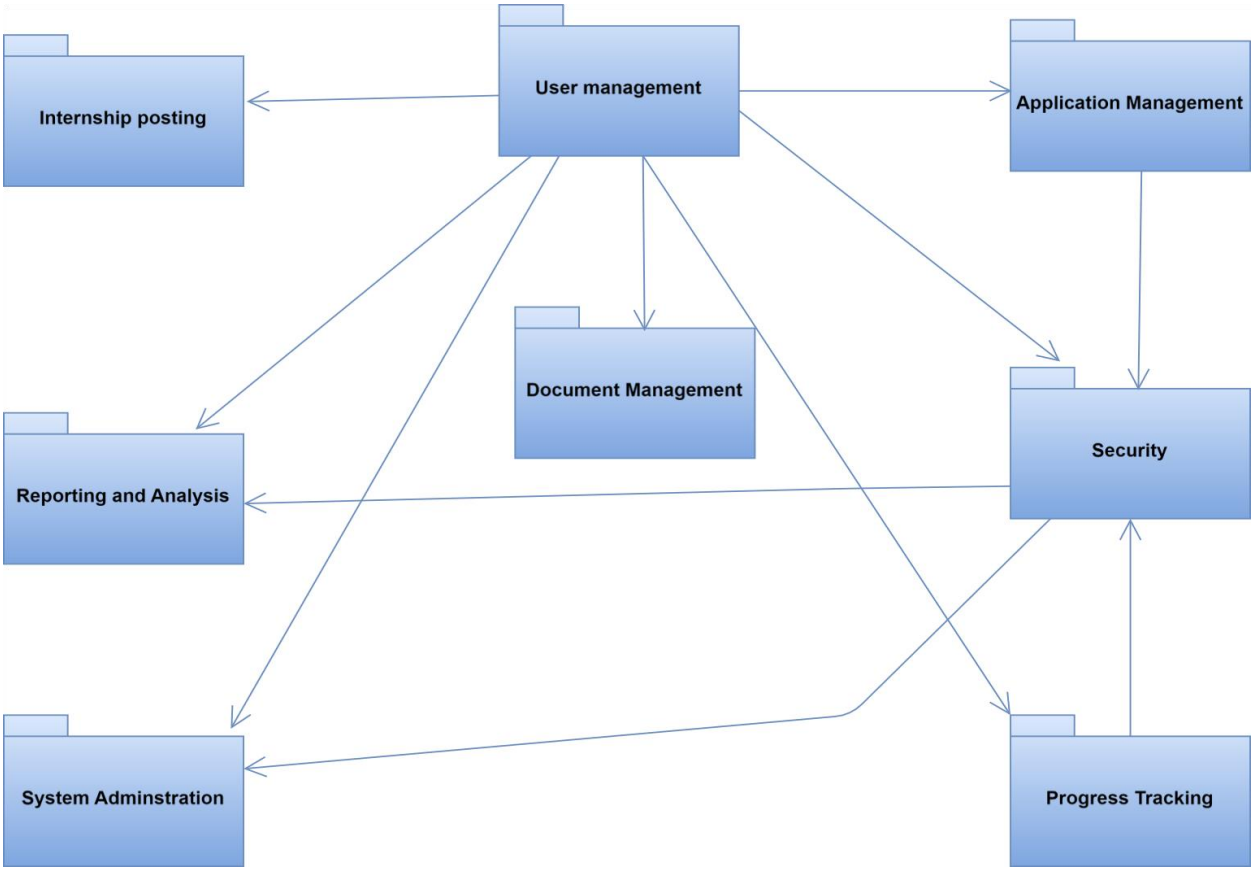


Figure 5. 6 Package Diagram

5.4. Algorithm Design

- **Pseudocode for applying an application**

```
Begin
    userAuthenticated = authenticateUser()
    IF userAuthenticated THEN
        {
            Display "Navigate to application submission page"
        }
        {
            internshipOpportunities = retrieveInternshipOpportunities()
```

```
        }
        {
            selectedOpportunity =
userSelectsOpportunity(internshipOpportunities)


 }
        IF selectedOpportunity in internshipOpportunities THEN
            {
                applicationForm =
retrieveApplicationForm(selectedOpportunity)
            }
            IF applicationForm is not null THEN
                {
                    applicationData = fillOutApplicationForm()
                }
                {
                    supportingDocuments = uploadSupportingDocuments()
                }
                {
                    DisplayApplicationSummary(applicationData,
supportingDocuments)
                }

                userConfirmation = userConfirmsSubmission()
                IF userConfirmation THEN
                    {
                        applicationId = submitApplication(applicationData,
supportingDocuments)
                        sendConfirmationNotification(applicationId)
                    }
                ELSE
                    {
                        Display "Application submission canceled"
                    }
                END IF
            ELSE
                {
                    DisplayError "Application form not available"
                }
            END IF
        ELSE
            {
```

```
                DisplayError "Selected internship opportunity not
available"
            }
        END IF
    ELSE
        {
            DisplayError "User authentication failed"
        }
    END IF
End
```

❖ **Pseudocode for approving student accounts**

```
Begin
      departmentHeadAuthenticated = authenticateDepartmentHead()

       IF departmentHeadAuthenticated THEN
              {
            pendingStudentAccounts = getPendingStudentAccounts()
            DisplayPendingStudentAccountsList(pendingStudentAccounts)
        }

             {
            selectedStudentAccount =
departmentHeadSelectsStudentAccount(pendingStudentAccounts)
        }

              IF selectedStudentAccount in pendingStudentAccounts THEN
                      {
                studentAccountDetails =
getStudentAccountDetails(selectedStudentAccount)
                DisplayStudentAccountDetails(studentAccountDetails)
            }

                      {
                reviewDecision =
departmentHeadReviewsStudentAccount(studentAccountDetails)
            }

                       IF reviewDecision == "Approve" THEN
                              {
                    markStudentAccountAsApproved(selectedStudentAccount)
                    sendApprovalNotification(studentAccountDetails.student)
                    Display "Student account approved successfully"
```

```
                }
            ELSE
                                IF reviewDecision == "Reject" THEN
                                        {

markStudentAccountAsRejected(selectedStudentAccount)

sendRejectionNotification(studentAccountDetails.student)
                            Display "Student account rejected"
                        }
                ELSE
                                        {
                            DisplayError "Invalid review decision"
                        }
                END IF
            END IF
        ELSE
                            {
                DisplayError "Invalid student account selection"
            }
        END IF
    ELSE
              {
            DisplayError "Department head authentication failed"
        }
    END IF
End
```

5.5. Interface Design

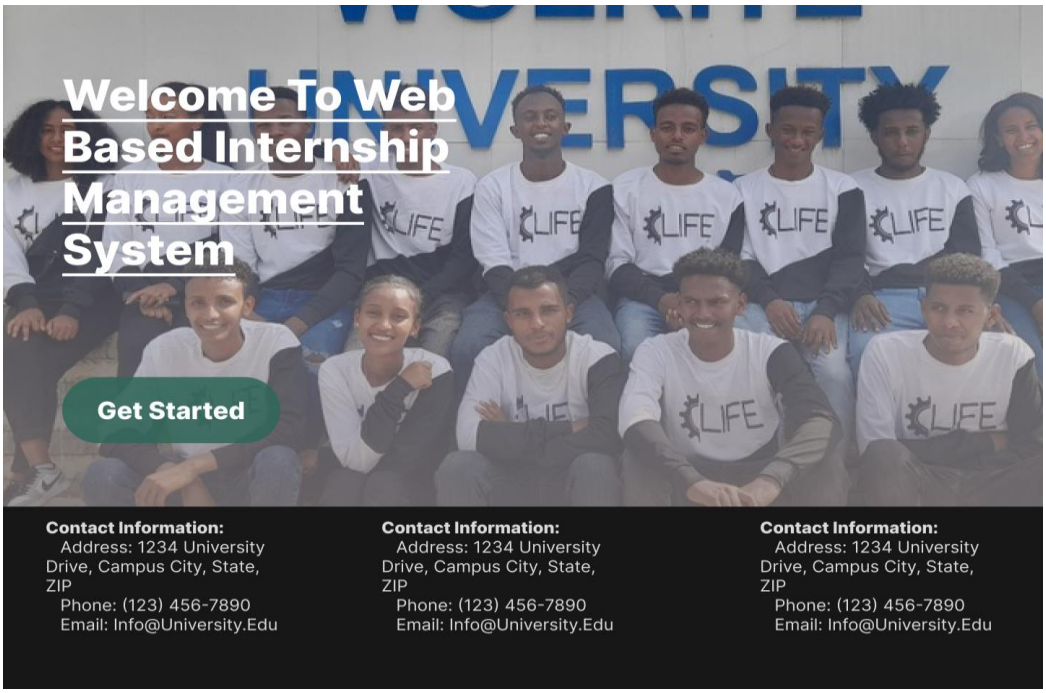


Figure 5. 7 Landing Page

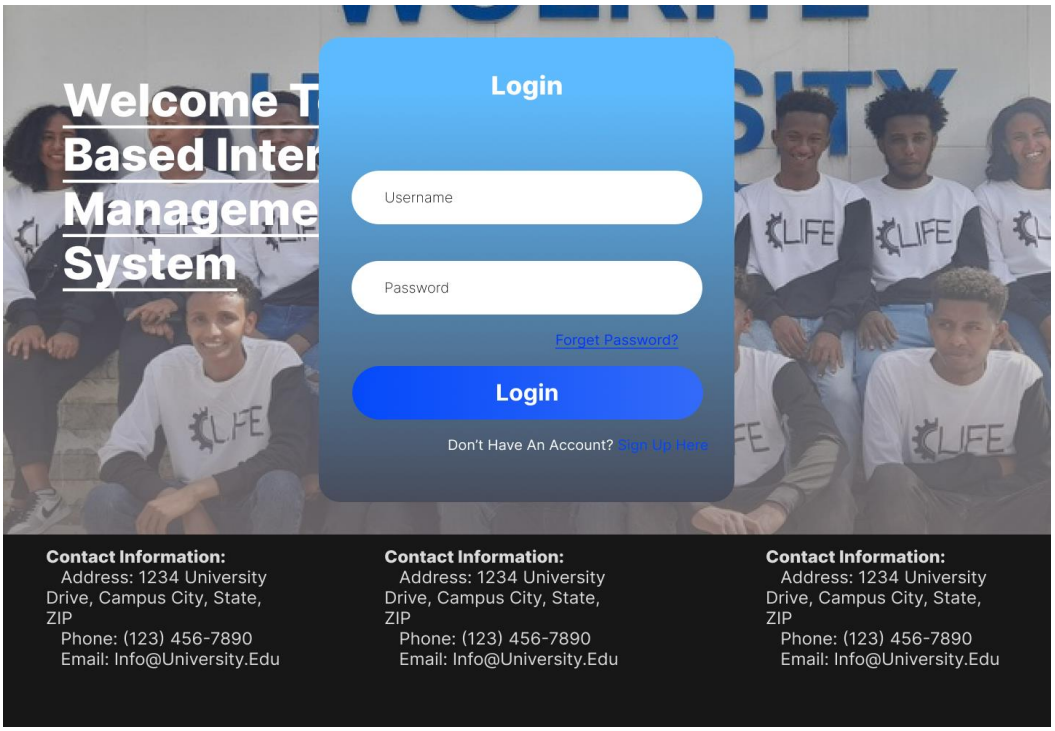


Figure 5. 8 Login Page

CHAPTER SIX

IMPLEMENTATION AND TESTING

6.1. Implementation of the Database

MongoDB is a highly suitable choice for our Web-Based Internship Management System due to its several advantageous features:

- Flexible Schema Design: MongoDB's document-oriented structure is ideal for managing diverse and evolving internship data such as student profiles and mentor feedback, which often vary in structure. This flexibility eliminates the need for complex joins or schema migrations.
- Scalability: MongoDB's sharding capability allows for horizontal scalability, distributing data across multiple servers to maintain performance as our user base and data volume grow, supporting the expansion of our internship program.
- Efficient Data Retrieval: The robust query language and indexing support in MongoDB enable efficient execution of complex queries, facilitating quick data access which is essential for a seamless user experience.
- Compatibility with Modern Technologies: MongoDB's support for JSON and BSON formats integrates smoothly with modern web development frameworks, easing the development process and allowing our team to focus on building interactive features.
- Community and Documentation: MongoDB's active community and extensive documentation provide a valuable resource for our development team, helping to apply best practices and troubleshoot challenges effectively.

Overall, MongoDB enhances our system with its flexibility, scalability, and performance, ensuring a robust and responsive platform for interns, mentors, and administrators.

Tables as Persistent Models:

MongoDB does not use tables like traditional relational databases, but rather collections to store documents. Each collection can be considered analogous to a table, and documents within

collections serve as records. With Prisma ORM, we defined models that map to MongoDB collections as shown below:

```
model Role {
  id    String @id @default(auto()) @map("_id") @db.ObjectId
  name  String @unique
  users User[]
}

model User {
  id               String          @id @default(auto()) @map("_id")
@db.ObjectId
  email            String          @unique
  password         String
  firstName        String
  middleName       String
  userName         String          @unique
  role             Role?           @relation(fields: [roleName],
references: [name])
  roleName         String?
  profilePic       String
  imagePublicId    String
  phoneNum         String
  hashedRt         String?
  verified         Boolean?
  notifications    Notification[]
  createdAt        DateTime        @default(now())
  updatedAt        DateTime        @updatedAt
  conversationIds  String[]        @db.ObjectId
  conversations    Conversation[]   @relation(fields:
[conversationIds], references: [id])
```

```
  Student          Student[]
  Company          Company[]
  universityUsers  UniversityUser[]
  colleges         College[]
  departments      Department[]
  Advisor          Advisor[]
  Mentor           Mentor[]
  messagesSent     Message[]        @relation("Sender")
  messagesReceived Message[]        @relation("Receiver")
  University       University[]
}

model Student {
  id             String        @id @default(auto()) @map("_id")
@db.ObjectId
  userId         String        @unique @db.ObjectId
  user           User          @relation(fields: [userId], references:
[id], onDelete: Cascade)
  universityId   String?       @unique @db.ObjectId
  University     University?   @relation(fields: [universityId],
references: [id])
  departmentId   String?       @unique @db.ObjectId
  department     Department?   @relation(fields: [departmentId],
references: [id])
  year           Int
  gpa            Float
  skills         String[]
  resumeUrl      String?
  resumePublicId String?
  isOnInternship Boolean?
```

```
  advisorId      String?       @db.ObjectId
  advisor        Advisor?      @relation(fields: [advisorId],
references: [id])
  mentorId       String?       @db.ObjectId
  mentor         Mentor?       @relation(fields: [mentorId],
references: [id])
  internshipId   String?       @db.ObjectId
  internship     Internship?   @relation(fields: [internshipId],
references: [id])
  createdAt      DateTime      @default(now())
  updatedAt      DateTime      @updatedAt
  Report         Report[]
  Feedback       Feedback[]
  Evaluation     Evaluation[]
  Application    Application[]
}

model Internship {
  id                      String                @id @default(auto())
@map("_id") @db.ObjectId
  title                   String
  companyId               String                @db.ObjectId
  company                 Company               @relation(fields:
[companyId], references: [id])
  startDate               DateTime
  endDate                 DateTime
  schedule                Schedule
  compensations           Compensations
  description             InternshipDescription @relation(fields:
[internshipDescriptionId], references: [id])
```

```
  internshipDescriptionId String                @unique @db.ObjectId
  createdAt               DateTime              @default(now())
  updatedAt               DateTime              @updatedAt
  Report                  Report[]
  Application             Application[]
  Student                 Student[]
}

model Company {
  id                 String        @id @default(auto()) @map("_id")
@db.ObjectId
  name               String        @unique
  industry           String
  companyHRId        String        @db.ObjectId
  companyHR          User          @relation(fields: [companyHRId],
references: [id])
  logoUrl            String
  logoPublicId       String
  website            String?
  email              String
  phoneNum           String
  address            Address?
  internshiprOferred Internship[]
  createdAt          DateTime      @default(now())
  updatedAt          DateTime      @updatedAt
  Mentor             Mentor[]
  Feedback           Feedback[]
  Application        Application[]
}
```

```
model Application {
  id           String           @id @default(auto()) @map("_id")
@db.ObjectId
  studentId    String           @db.ObjectId
  student      Student          @relation(fields: [studentId],
references: [id])
  internshipId String           @db.ObjectId
  internship   Internship       @relation(fields: [internshipId],
references: [id])
  companyId    String           @db.ObjectId
  company      Company          @relation(fields: [companyId],
references: [id])
  status       ApplicationStatus
  createdAt    DateTime         @default(now())
  updatedAt    DateTime         @updatedAt
}

model University {
  id                String          @id @default(auto()) @map("_id")
@db.ObjectId
  name              String          @unique
  logoUrl           String
  logoPublicId      String
  websiteUrl        String?
  college           College[]
  departments       Department[]
  universityAdminId String          @db.ObjectId
  verified          Boolean         @default(false)
  universityAdmin   User?           @relation(fields: [userId],
references: [id])
```

```
  email             String           @unique
  phoneNum          String
  address           Address?
  createdAt         DateTime         @default(now())
  updatedAt         DateTime         @updatedAt
  Student           Student[]
  universityUsers   UniversityUser[]
  userId            String?          @db.ObjectId
}

model UniversityUser {
  id           String     @id @default(auto()) @map("_id")
@db.ObjectId
  university   University @relation(fields: [universityId],
references: [id])
  universityId String
  user         User       @relation(fields: [userId], references:
[id])
  userId       String
}

model Department {
  id               String     @id @default(auto()) @map("_id")
@db.ObjectId
  name             String     @unique
  departmentHeadId String     @db.ObjectId
  departmentHead   User       @relation(fields: [departmentHeadId],
references: [id])
  collegeId        String     @db.ObjectId
  college          College    @relation(fields: [collegeId],
```

```
references: [id])
  email            String
  phoneNum         String
  createdAt        DateTime   @default(now())
  updatedAt        DateTime   @updatedAt
  Student          Student[]
  University       University @relation(fields: [universityId],
references: [id])
  universityId     String     @db.ObjectId
  Advisor          Advisor[]
}

model Advisor {
  id           String       @id @default(auto()) @map("_id")
@db.ObjectId
  userId       String       @db.ObjectId
  user         User         @relation(fields: [userId], references:
[id])
  departmentId String       @db.ObjectId
  department   Department   @relation(fields: [departmentId],
references: [id])
  createdAt    DateTime     @default(now())
  updatedAt    DateTime     @updatedAt
  Student      Student[]
  Report       Report[]
  Feedback     Feedback[]
  Evaluation   Evaluation[]
}

model Mentor {
```

```
  id         String       @id @default(auto()) @map("_id")
@db.ObjectId
  userId     String       @db.ObjectId
  user       User         @relation(fields: [userId], references:
[id])
  companyId  String       @db.ObjectId
  company    Company      @relation(fields: [companyId], references:
[id])
  createdAt  DateTime     @default(now())
  updatedAt  DateTime     @updatedAt
  Student    Student[]
  Report     Report[]
  Feedback   Feedback[]
  Evaluation Evaluation[]
}
```

Defining indexes

```
model Role {
  id    String @id @default(auto()) @map("_id") @db.ObjectId
  name  String @unique
  users User[]
  @@index([name]) // Index on the 'name' field for faster lookups
}

model User {
  id               String           @id @default(auto()) @map("_id")
@db.ObjectId
  email            String           @unique
  password         String
  firstName        String
```

```
  middleName       String
  userName         String          @unique
  role             Role?           @relation(fields: [roleName],
references: [name])
  roleName         String?
  profilePic       String
  imagePublicId    String
  phoneNum         String
  hashedRt         String?
  verified         Boolean?
  notifications    Notification[]
  createdAt        DateTime        @default(now())
  updatedAt        DateTime        @updatedAt
  conversationIds  String[]        @db.ObjectId
  conversations    Conversation[]   @relation(fields:
[conversationIds], references: [id])
  Student          Student[]
  Company          Company[]
  universityUsers  UniversityUser[]
  colleges         College[]
  departments      Department[]
  Advisor          Advisor[]
  Mentor           Mentor[]
  messagesSent     Message[]        @relation("Sender")
  messagesReceived Message[]        @relation("Receiver")
  University       University[]
  @@index([email]) // Index on the 'email' field for faster lookups
  @@index([roleName]) // Index on the 'roleName' field for faster
lookups
}
```

```
model Student {
  id             String        @id @default(auto()) @map("_id")
@db.ObjectId
  userId         String        @unique @db.ObjectId
  user           User          @relation(fields: [userId], references:
[id], onDelete: Cascade)
  universityId   String?       @unique @db.ObjectId
  University     University?   @relation(fields: [universityId],
references: [id])
  departmentId   String?       @unique @db.ObjectId
  department     Department?   @relation(fields: [departmentId],
references: [id])
  year           Int
  gpa            Float
  skills         String[]
  resumeUrl      String?
  resumePublicId String?
  isOnInternship Boolean?
  advisorId      String?       @db.ObjectId
  advisor        Advisor?      @relation(fields: [advisorId],
references: [id])
  mentorId       String?       @db.ObjectId
  mentor         Mentor?       @relation(fields: [mentorId],
references: [id])
  internshipId   String?       @db.ObjectId
  internship     Internship?   @relation(fields: [internshipId],
references: [id])
  createdAt      DateTime      @default(now())
  updatedAt      DateTime      @updatedAt
```

```
  Report          Report[]
  Feedback        Feedback[]
  Evaluation      Evaluation[]
  Application     Application[]
  @@index([userId]) // Index on the 'userId' field for faster lookups
}

model Company {
  id                  String        @id @default(auto()) @map("_id")
@db.ObjectId
  name                String        @unique
  industry            String
  companyHRId         String        @db.ObjectId
  companyHR           User          @relation(fields: [companyHRId],
references: [id])
  logoUrl             String
  logoPublicId        String
  website             String?
  email               String
  phoneNum            String
  address             Address?
  internshipsOferred Internship[]
  createdAt           DateTime      @default(now())
  updatedAt           DateTime      @updatedAt
  Mentor              Mentor[]
  Feedback            Feedback[]
  Application         Application[]
  @@index([companyHRId]) // Index on the 'companyHRId' field for
faster lookups
}
```

Configuring a Schedule of Database Backup

- **Backup Frequency:** For our internship management system, where data like student profiles, internship details, and communication logs are essential, we opt for daily backups.
- **Backup Method:** we chose Snapshot backups as it offers a cost-effective solution while still providing essential data protection capabilities. By capturing the state of the database at specific intervals, typically daily, snapshot backups enable us to restore data to the last snapshot in case of data loss or system failure.

Configuring Database-Level Security

To enhance the security of our Web-Based Internship Management System, we focus on several key areas:

- Authentication and Authorization: We use strong authentication methods supported by MongoDB Atlas, such as username/password and LDAP. Additionally, we deploy role-based access control (RBAC) and JSON Web Tokens (JWT) to manage user permissions effectively, ensuring users access only data relevant to their roles.
- Access Control: We utilize MongoDB's access control features to restrict user access to specific databases, collections, or operations, preventing unauthorized data access.
- Encryption: We employ SSL/TLS encryption to protect data in transit between our server and the database. For data at rest, we use field-level encryption within MongoDB to secure sensitive fields in documents.
- Regular Auditing and Monitoring: We continuously audit and monitor database activities using MongoDB's auditing features. This helps us detect and respond to security threats promptly, maintaining the system's integrity and confidentiality.

By implementing these measures, we ensure robust security to protect against both internal and external threats, safeguarding sensitive information within our system.

6.2. Implementation of the Class Diagram

```
// Role enum definition
```

```
export enum Role {
    SYSTEM_ADMIN = 'SYSTEM_ADMIN',
    STUDENT = 'STUDENT',
    UNIVERSITY_ADMIN = 'UNIVERSITY_ADMIN',
    COLLEGE_DEAN = 'COLLEGE_DEAN',
    DEPARTMENT_HEAD = 'DEPARTMENT_HEAD',
    COMPANY_HR = 'COMPANY_HR',
    ADVISOR = 'ADVISOR',
    MENTOR = 'MENTOR',
}
```

```
// StudentService class implementation

import { Injectable } from '@nestjs/common';
import { PrismaService } from 'src/common/prisma/prisma.service';

@Injectable()
export class StudentService {
    constructor(private prismaService :PrismaService){}

    async getStudentInUniversity(id: string) {

        const student = await this.prismaService.student.findMany({
            where: {
                universityId: id
            },
        include: {
            advisor: true,
            department:true,
```

```
            user:true
        },
        });

        return student;
    }
}
```

```
// UserService class implementation

import { Injectable } from '@nestjs/common';
import { CreateUserDto, UpdateUserDto } from 'src/common/dtos';
import { PrismaService } from 'src/common/prisma/prisma.service';
import * as argon from 'argon2';

@Injectable()
export class UsersService {
  constructor(private prismaService: PrismaService) {}

  async createStaffUser(
    createUserDto: CreateUserDto,
    universityId: any,
  ): Promise<any> {
    const hashedPassword = await
argon.hash(createUserDto.userPassword);

    const newUser = await this.prismaService.user.create({
      data: {
        firstName: createUserDto.firstName,
```

```
      middleName: createUserDto.middleName,
      userName: createUserDto.userName,
      profilePic: createUserDto.profilePic,
      imagePublicId: createUserDto.profilePicPublicId,
      phoneNum: createUserDto.phoneNum,
      verified: false,
      email: createUserDto.email,
      password: hashedPassword,
      roleName: createUserDto.roleName,
    },
  });

  // Create the association between the user and the university
  await this.prismaService.universityUser.create({
    data: {
      universityId: universityId,
      userId: newUser.id,
    },
  });

  return newUser;
}

async getAllUsers(): Promise<any> {
  const allUsers = await this.prismaService.user.findMany();
  return allUsers;
}

async getUserById(id: string): Promise<any> {
  const user = await this.prismaService.user.findUnique({
```

```
      where: {
        id: id,
      },
    });
    return user;
  }

  async getAllUniversityUsers(id: string): Promise<any> {
    const allUsers = await
this.prismaService.universityUser.findMany({
      where: {
        universityId: id,
      },
      include: {
        user: true,
      },
    });
    return allUsers;
  }

  async getAllUniversityStudents(id: string): Promise<any> {
    const allUniversity = await this.prismaService.student.findMany({
      where: {
        universityId: id,
      },
    });

    return allUniversity;
  }
```

```
async updateUser(updateUserDto, id: string): Promise<any> {
  const updatedUser = await this.prismaService.user.update({
    where: {
      id: id,
    },
    data: updateUserDto,
  });
  return updatedUser;
}

async verifyUser(id: string) {
  const verifiedUser = await this.prismaService.user.update({
    where: {
      id: id,
    },
    data: {
      verified: true,
    },
  });
  return verifiedUser;
}

async deleteUser(id: string) {
  const deletedUser = await this.prismaService.user.delete({
    where: {
      id: id,
    },
  });
  return deletedUser;
}
```

```
  async user(id: string): Promise<any> {
    try {
      const usersWithoutRole = await
this.prismaService.universityUser.findMany(
        {
          where: {
            universityId: id,
          },
          include: {
            user: true,
          },
        },
      );

      return usersWithoutRole;
    } catch (error) {
      console.error('Error fetching users without role:', error);
      throw error;
    }
  }

  async getNormalUser(id: string): Promise<any> {
    const usersWithoutRole = await this.prismaService.user.findMany({
      where: {
        id: this.user.toString(),
      },
    });
    return usersWithoutRole;
  }
```

```
async getCOLLEGE_DEAN(): Promise<any> {
  const usersWithoutRole = await this.prismaService.user.findMany({
    where: {
      roleName: 'COLLEGE_DEAN',
    },
  });
  return usersWithoutRole;
}
async getDEPARTMENT_HEAD(): Promise<any> {
  const usersWithoutRole = await this.prismaService.user.findMany({
    where: {
      roleName: 'DEPARTMENT_HEAD',
    },
  });
  return usersWithoutRole;
}

async getNotDeanandHeadUser(): Promise<any> {
  const usersWithoutRole = await this.prismaService.user.findMany({
    // where: {
    //     AND: [
    //         { College: null },
    //         { Department: null },
    //     ]
    // }
  });
  return usersWithoutRole;
}
```

```
  async assignRoleToUser(userId: string, roleName: string):
Promise<any> {
    return this.prismaService.user.update({
      where: { id: userId },
      data: { roleName: roleName },
    });
  }
}
```

```
// HeadService class implementation
import { Injectable } from '@nestjs/common';
import * as argon from 'argon2';
import { PrismaService } from 'src/common/prisma/prisma.service';
@Injectable()
export class HeadService {
  constructor(private prismaService: PrismaService) {}

  async assignAdvisorToStudent(
    studentId: string,
    advisorId: string,
  ): Promise<void> {
    await this.prismaService.student.update({
      where: { id: studentId },
      data: { advisorId },
    });
  }

  async createAdvisor(dto: any): Promise<void> {
    try {
```

```
    const hashedPassword = await argon.hash(dto.adminPassword);
    const user = await this.prismaService.user.create({
      data: {
        userName: dto.adminUserName,
        email: dto.adminEmail,
        password: hashedPassword,
        firstName: dto.adminFirstName,
        middleName: dto.adminMiddleName,
        profilePic: dto.adminProfilePicture,
        imagePublicId: dto.adminImagePublicId,
        phoneNum: dto.adminPhoneNumber,
        roleName: 'ADVISOR',
        notifications: dto.adminNotifications,
        conversationIds: dto.adminConversationIds,
      },
    });

    await this.prismaService.advisor.create({
      data: {
        department: {
          connect: { id: user.id },
        },
        user: {
          connect: { id: user.id },
        },
      },
    });
  } catch (error) {
    console.log(error);
  }
```

```
  }
}
```

6.3. Configuration of the Application Server

In our development, we have opted for using the Nest.js framework for our backend server, which provides a robust and scalable foundation for building modern web applications. Here's why we chose Nest.js and how we configured our application server:

Justification for Application Server Selection:

- Express.js Underlying Framework: Nest.js is built on top of Express.js, a widely used and highly performant Node.js web framework. This provides us with the flexibility and familiarity of Express while adding additional features and structure through Nest.js.
- Modular and Scalable Architecture: Nest.js promotes a modular architecture that allows us to organize our codebase into reusable modules, controllers, and services. This makes it easier to maintain and scale our application as it grows in complexity.
- Dependency Injection and Testing Support: Nest.js supports dependency injection, which facilitates writing clean and testable code. This enables us to easily mock dependencies for unit testing, ensuring the reliability and stability of our application.
- Typescript Support: Nest.js is built with TypeScript, a statically typed superset of JavaScript. This provides us with compile-time type checking, better IDE support, and enhanced code readability and maintainability.

Configuration Activities:

- Start and Shutdown Procedures: We use Nest.js lifecycle hooks for graceful startup and shutdown, ensuring resources are properly managed.

- Folder and File Organization: Our project structure adheres to Nest.js recommendations, with separate directories for controllers, services, modules, and configuration files, improving organization and ease of maintenance.
- Local and Remote Access: Our server is configured to allow both local access via localhost and remote access through network configurations, ensuring flexibility in connectivity.
- Separation of Server and Database: We utilize cloud services like MongoDB Atlas for database management, keeping our application server and database infrastructure separate for better scalability and security.
- Port Number Configuration: Our server's port number is customizable, supporting multiple instances on one machine or distribution across multiple machines.

Overall, our choice of Nest.js as the application server for our project, coupled with proper configuration and organization, provides us with a solid foundation for developing and deploying our web-based Internship Management System.

6.4. Configuration of Application Security

To bolster the security of our Web-Based Internship Management System, we've implemented several robust measures:

- Input Validation: We use Zod for client-side validation and NestJS's class validator for server-side validation to ensure all incoming data is accurate and secure.
- Encryption/Decryption: Passwords are protected using the Argon hashing algorithm, and JSON Web Tokens (JWT) secure our session management, ensuring user credentials and session tokens are safe from unauthorized access.
- Role Definition and Access Privileges: Different user roles (students, advisors, mentors, etc.) are clearly defined with specific privileges, ensuring users can only access what is relevant to their role.
- Session Management: Sessions are managed securely using JWTs and local storage, with mechanisms in place for token generation and expiration handling.

- Compliance and Implementation: We adhere to strict non-functional security requirements throughout development and testing, incorporating secure coding practices, regular audits, and updates on security patches and vulnerabilities.

By integrating these security features, we ensure a well-protected system that maintains the confidentiality, integrity, and availability of user data and system operations.

6.5. Testing

6.5.1. Test Case

- User Authentication:
    - Test Case: Verify that users can log in with valid credentials.
    - Expected Outcome: Successful authentication and redirection to the user dashboard.
- Internship Application Submission:
    - Test Case: Submit an internship application with valid details.
    - Expected Outcome: Application saved in the system with correct information.
- Internship Application Validation:
    - Test Case: Submit an internship application with missing required fields.
    - Expected Outcome: Display validation errors for missing fields.
- Session Timeout Handling:
    - Test Case: Leave the system idle for the session timeout duration.
    - Expected Outcome: User logged out automatically and prompted to log in again upon activity.
- Internship Application Status Update:
    - Test Case: Update the status of an internship application (e.g., accepted, rejected).
    - Expected Outcome: Application status updated in the system and reflected in the user dashboard.
- Successful Internship Posting:
    - Test Case: Create a new internship posting with all required fields filled correctly.
    - Expected Outcome: Internship posting is successfully created and displayed in the internship listings.

6.5.2 Testing Tools and Environment

For testing purposes, we utilize Jest as our primary testing framework, renowned for its simplicity, flexibility, and robustness in testing JavaScript applications. Jest provides a suite of powerful features, including test runners, assertion libraries, and mocking capabilities, enabling us to conduct unit testing seamlessly.

6.5.3 Unit Testing

Testcase 1: User Authentication

```
// login.controller.spec.ts

import { Test, TestingModule } from '@nestjs/testing';
import { LoginController } from './login.controller';
import { LoginService } from './login.service';

// Mocking the LoginService
const mockLoginService = {
    login: jest.fn(dto => Promise.resolve({
        access_token: 'mockAccessToken',
        refresh_token: 'mockRefreshToken'
    })),
};

describe('LoginController', () => {
    let controller: LoginController;

    beforeEach(async () => {
        const module: TestingModule = await Test.createTestingModule({
            controllers: [LoginController],
            providers: [
                {
                    provide: LoginService,
                    useValue: mockLoginService
```

```
                },
            ],
        }).compile();

        controller = module.get<LoginController>(LoginController);
    });

    afterEach(() => {
        jest.clearAllMocks();
    });

    it('should call loginService with expected params and return
tokens', async () => {
        const dto = { email: 'rover@gmail.com', password: '12345678'
};
        const result = await controller.login(dto);

        expect(mockLoginService.login).toHaveBeenCalledWith(dto);
        expect(result).toEqual({
            access_token: 'mockAccessToken',
            refresh_token: 'mockRefreshToken'
        });
    });

    it('should handle and return a HttpStatus.OK on successful login',
async () => {
        const dto = { email: 'rover@gmail.com', password: '12345678'
};
        const spy = jest.spyOn(mockLoginService,
'login').mockImplementation(() => Promise.resolve({
            access_token: 'mockAccessToken',
```

```
            refresh_token: 'mockRefreshToken'
        }));

        const result = await controller.login(dto);

        expect(spy).toHaveBeenCalledWith(dto);
        expect(result).toBeDefined();
        expect(result.access_token).toBeDefined();
        expect(result.refresh_token).toBeDefined();
    });

});
```

Testcase 2: Internship Application Submission, Internship Application Validation and Internship Application Status Update

```
// apply.service.spec.ts

import { Test, TestingModule } from '@nestjs/testing';
import { PrismaService } from 'src/common/prisma/prisma.service';
import { ApplyService } from './apply.service';
import { Prisma } from '.prisma/client';
import { DefaultArgs } from '@prisma/client/runtime/library';
import { CreateApplicationDto, UpdateApplicationDto } from
'src/common/dtos';



describe('ApplyService', () => {
    let service: ApplyService;
    let prismaService: Partial<PrismaService>;

    beforeEach(async () => {
        // Mock PrismaService
```

```
        prismaService = {
            application: {
                create: jest.fn().mockResolvedValue({ id: '123',
status: 'PENDING' }),
                update: jest.fn().mockResolvedValue({ id: '123',
status: 'ACCEPTED' }),
                findMany: jest.fn().mockResolvedValue([{ id: '123',
status: 'PENDING' }]),
                findUnique: jest.fn().mockResolvedValue({ id: '123',
status: 'PENDING' }),
                delete: jest.fn().mockResolvedValue({ id: '123' }),
            } as Partial<Prisma.ApplicationDelegate<DefaultArgs>>,
        } as Partial<PrismaService>;

        const module: TestingModule = await Test.createTestingModule({
            providers: [
                ApplyService,
                { provide: PrismaService, useValue: prismaService }
            ],
        }).compile();

        service = module.get<ApplyService>(ApplyService);
    });

    describe('createApplication', () => {
        it('should create an application', async () => {
            const dto: CreateApplicationDto = { studentId: '1',
companyId: '1', internshipId: '1', status: 'PENDING' };
            const result = await service.createApplication(dto);
            expect(result).toEqual({ id: '123', status: 'PENDING' });
```

```
expect(prismaService.application.create).toHaveBeenCalledWith({
                data: dto
            });
        });
    });

    describe('updateApplication', () => {
        it('should update an application', async () => {
            const dto: UpdateApplicationDto = { status: 'ACCEPTED' };
            const result = await service.updateApplication('123',
dto);
            expect(result).toEqual({ id: '123', status: 'ACCEPTED' });

expect(prismaService.application.update).toHaveBeenCalledWith({
                where: { id: '123' },
                data: dto
            });
        });
    });

    describe('getApplications', () => {
        it('should return all applications', async () => {
            const result = await service.getApplications();
            expect(result).toEqual([{ id: '123', status: 'PENDING'
}]);

expect(prismaService.application.findMany).toHaveBeenCalled();
        });
    });
```

```
    describe('getApplicationById', () => {
        it('should return a single application', async () => {
            const result = await service.getApplicationById('123');
            expect(result).toEqual({ id: '123', status: 'PENDING' });

expect(prismaService.application.findUnique).toHaveBeenCalledWith({
                where: { id: '123' }
            });
        });
    });

    describe('deleteApplication', () => {
        it('should delete an application', async () => {
            const result = await service.deleteApplication('123');
            expect(result).toEqual({ id: '123' });

expect(prismaService.application.delete).toHaveBeenCalledWith({
                where: { id: '123' }
            });
        });
    });

});
```

Testcase 3: Session Timeout Handling

```
// refresh.service.spec.ts

import { Test, TestingModule } from '@nestjs/testing';
import { RefreshService } from './refresh.service';
import { PrismaService } from 'src/common/prisma/prisma.service';
import { GenerateJwtService } from '../jwt/generate.jwt.service';
```

```
import * as argon from 'argon2';
import { ForbiddenException } from '@nestjs/common';
import { Prisma } from '.prisma/client';
import { DefaultArgs } from '@prisma/client/runtime/library';


jest.mock('argon2', () => ({
    verify: jest.fn(),
    hash: jest.fn()
}));

describe('RefreshService', () => {
    let service: RefreshService;
    let prismaService: Partial<PrismaService>;
    let jwtService: Partial<GenerateJwtService>;

    beforeEach(async () => {
        prismaService = {
            user: {
                findUnique: jest.fn(),
                update: jest.fn()
            } as Partial<Prisma.UserDelegate<DefaultArgs>>,
        } as Partial<PrismaService>;
        jwtService = {
            getToken: jest.fn()
        };

        const module: TestingModule = await Test.createTestingModule({
            providers: [
                RefreshService,
                {
```

```
                    provide: PrismaService,
                    useValue: prismaService
                },
                {
                    provide: GenerateJwtService,
                    useValue: jwtService
                },
            ],
        }).compile();

        service = module.get<RefreshService>(RefreshService);
    });

    it('should be defined', () => {
        expect(service).toBeDefined();
    });

    describe('refreshTokens', () => {
        it('should throw ForbiddenException if user does not exist',
async () => {
            prismaService.user.findUnique =
jest.fn().mockResolvedValue(null) as any;
            await expect(service.refreshTokens('user-id', 'token'))
                .rejects.toThrow(ForbiddenException);
        });

        it('should throw ForbiddenException if refresh token does not
match', async () => {
            prismaService.user.findUnique =
jest.fn().mockResolvedValue({
                id: 'user-id',
```

```
                email: 'email@test.com',
                roleName: 'User',
                hashedRt: 'hashed-refresh-token'
            });
            (argon.verify as jest.Mock).mockResolvedValue(false);

            await expect(service.refreshTokens('user-id', 'invalid-
token'))
                .rejects.toThrow(ForbiddenException);
        });

        it('should successfully refresh tokens when session is valid',
async () => {
            prismaService.user.findUnique =
jest.fn().mockResolvedValue({
                id: 'user-id',
                email: 'email@test.com',
                roleName: 'User',
                hashedRt: 'hashed-refresh-token'
            });
            (argon.verify as jest.Mock).mockResolvedValue(true); //
Mock to return true
            jwtService.getToken = jest.fn().mockResolvedValue({
                access_token: 'new-access-token',
                refresh_token: 'new-refresh-token'
            });
            (argon.hash as jest.Mock).mockResolvedValue('new-hashed-
rt');

            const result = await service.refreshTokens('user-id',
'valid-token');
```

```
            expect(result).toEqual({
                access_token: 'new-access-token',
                refresh_token: 'new-refresh-token'
            });
            expect(jwtService.getToken).toHaveBeenCalledWith('user-
id', 'email@test.com', 'User');
            expect(prismaService.user.update).toHaveBeenCalledWith({
                where: { id: 'user-id' },
                data: { hashedRt: 'new-hashed-rt' }
            });
        });
    });
});
```

Testcase 4: Successful Internship Posting

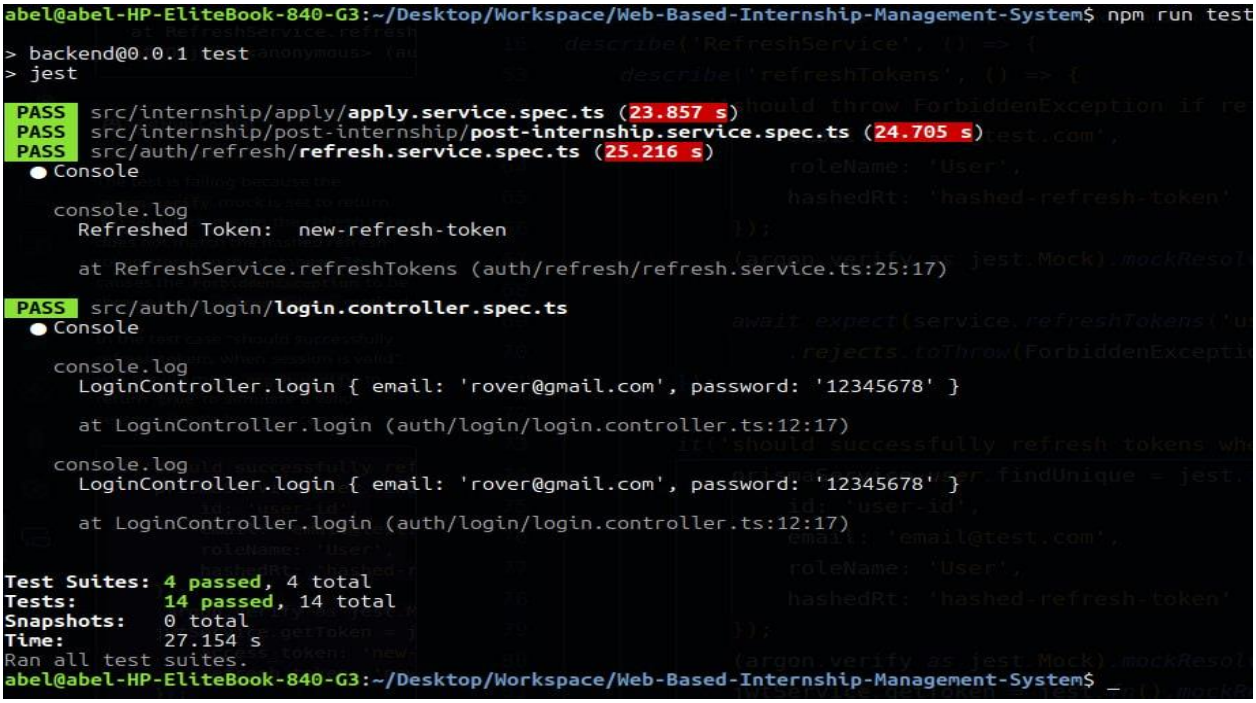
```
// post-internship.service.spec.ts

import { Test, TestingModule } from '@nestjs/testing';
import { PostInternshipService } from './post-internship.service';
import { PrismaService } from 'src/common/prisma/prisma.service';
import { Schedule, Compensations, CreateInternship } from
'src/common/dtos';
import { Prisma } from '.prisma/client';
import { DefaultArgs } from '@prisma/client/runtime/library';

describe('PostInternshipService', () => {
    let service: PostInternshipService;
    let prismaService: Partial<PrismaService>;

    beforeEach(async () => {
```

```
        prismaService = {
            internship: {
                findMany: jest.fn().mockResolvedValue([{ id: '1',
title: 'Test Internship' }]),
                create: jest.fn().mockImplementation((dto) =>
Promise.resolve({ id: '2', ...dto.data })),
                findUnique: jest.fn().mockResolvedValue(null),
                update: jest.fn().mockResolvedValue({}),
                delete: jest.fn().mockResolvedValue({}),
                // other methods are not mocked but are no longer
required by TypeScript to be present
            } as Partial<Prisma.InternshipDelegate<DefaultArgs>>,
            internshipDescription: {
                create: jest.fn().mockResolvedValue({ id: 'desc1',
description: 'Description' }),
                update: jest.fn().mockResolvedValue({}),
                delete: jest.fn().mockResolvedValue({}),
            } as
Partial<Prisma.InternshipDescriptionDelegate<DefaultArgs>>,
        } as Partial<PrismaService>;

        const module: TestingModule = await Test.createTestingModule({
            providers: [
                PostInternshipService,
                {
                    provide: PrismaService,
                    useValue: prismaService
                },
            ],
        }).compile();
```

```
        service =
module.get<PostInternshipService>(PostInternshipService);
    });

    it('should be defined', () => {
        expect(service).toBeDefined();
    });

    describe('getAllInternship', () => {
        it('should return an array of internships', async () => {
            const internships = await service.getAllInternship();
            expect(internships).toEqual([{ id: '1', title: 'Test
Internship' }]);

expect(prismaService.internship.findMany).toHaveBeenCalled();
        });
    });

    describe('createInternship', () => {
        it('should successfully create an internship', async () => {
            const dto: CreateInternship = {
                title: "New Internship",
                companyId: "comp1",
                responsibilities: ["Manage tasks"],
                qualifications: ["Bachelor's Degree"],
                applicationInstructions: "Apply here",
                description: "Full internship details",
                deadline: new Date(),
                startDate: new Date(),
                endDate: new Date(),
                schedule: Schedule.FULL_TIME,
```

```
                compensations: Compensations.PAID,
            };
            const result = await service.createInternship(dto);
            expect(result).toHaveProperty('id', '2');

expect(prismaService.internship.create).toHaveBeenCalledWith({
                data: expect.anything()
            });
        });
    });
});
```

Output



CHAPTER SEVEN

CONCLUSION AND RECOMMENDATION

7.1. Conclusion

Our project has worked hard to create the Web-Based Internship Management System (IMS) for Wolkite University with the primary goal of improving the effectiveness and efficiency of internship management procedures. We've addressed the precise goals outlined using a thorough approach, making sure that every aspect of the internship experience is well-taken care of.

Our system includes a comprehensive solution specifically designed to meet the demands of Wolkite University, from enabling smooth student registration to easing employer contact and digitizing the application and selection process. In addition, the incorporation of a communication platform, performance tracking tools, and assessment processes guarantees easy stakeholder engagement, promoting cooperation and feedback sharing.

In addition to achieving the specified goals, our system is aware of its limitations. The omission of the attendance and financial management functionalities indicates a purposeful focus on key elements necessary for enhancing the internship process, even in light of the experience's complexity. Moreover, the lack of instructor management features encourages additional thought for upcoming improvements, which could increase the system's potential reach.

In conclusion, our dedication to efficiency and innovation in educational administration is demonstrated by our Web-Based Internship Management System. Our study presents a strong solution that has the potential to transform internship administration at Wolkite University by combining important ideas and goals. Opportunities for future improvements and expansions still exist, indicating a commitment to meeting changing demands and promoting quality in internship programs.

References

Library, J. (2004). CSUSB ScholarWorks CSUSB ScholarWorks Theses Digitization Project. https://scholarworks.lib.csusb.edu/cgi/viewcontent.cgi?article=3592&context=etd-project

Gustiana, ilham P. (n.d.). Web-Based Internship Information System. IOP Conference Series: Materials Science and Engineering. Retrieved May 8, 2024, from https://www.academia.edu/69467399/Web_Based_Internship_Information_System?uc-g-sw=34291111

Intern Management System powered by GradLeaders for employers. (2023, February 7). GradLeaders. https://www.gradleaders.com/internship-management-system/

Wikipedia Contributors. (2019, May 13). Entity–relationship model. Wikipedia; Wikimedia Foundation. https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

Draw.io. (2024). Flowchart Maker & Online Diagram Software. App.diagrams.net. https://app.diagrams.net/

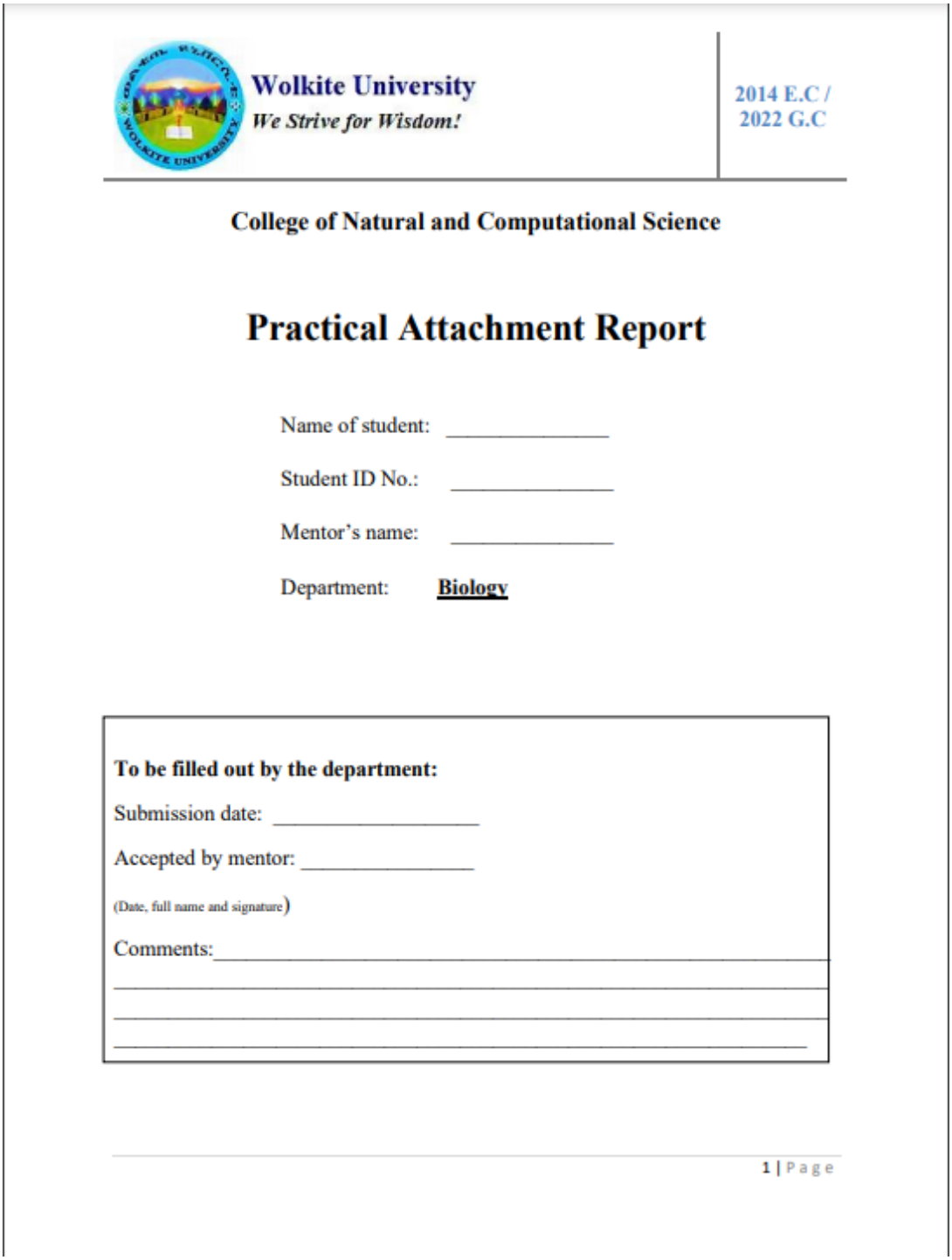
LucidChart. (n.d.). Lucid visual collaboration suite: Log in. Lucid.app. https://lucid.app/documents#/documents?folder_id=recent

Visual Paradigm. (2019). What is Class Diagram? Visual-Paradigm.com. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

Appendix I: Interview Questions

1. Can you describe the current internship management system in place at our college? How does it work from your perspective?
2. What are the key components of this system, and how do they support the management of internships?
3. What are the main challenges or limitations you have encountered with the current system?
4. How do these challenges impact the administration, students, and faculty involved in the internship process?
5. How effective is the current system in facilitating communication and coordination between students, faculty, and internship providers?
6. Are there any specific aspects of communication or coordination that need improvement?
7. How does the current system handle the tracking and evaluation of student performance during internships?
8. Are there any difficulties in monitoring student progress or providing feedback through the system?
9. Are there any technological enhancements you believe could improve the system?
10. Based on your experience, what specific changes or improvements would you recommend for the internship management system?
11. Are there any best practices or models from other institutions that you think could be beneficial for us to consider?
12. How do you envision the ideal system supporting our students and faculty in the context of internships?
13. Is there anything else you would like to share about your experience with the current internship management system?
14. Do you have any additional suggestions or ideas that could contribute to the development of a more effective system?

Appendix II: Existing System Forms and Reports



Wolkite University
We Strive for Wisdom!

2014 E.C / 2022 G.C

College of Natural and Computational Science

Practical Attachment Report

Name of student: _____

Student ID No.: _____

Mentor's name: _____

Department: **Biology**

To be filled out by the department:

Submission date: _____

Accepted by mentor: _____

(Date, full name and signature)

Comments:_____

_____

_____

_____

1 | Page

Practical Attachment Attendance Sheet

Student Full Name: _____

Organization/Office Name: _____

Month:_____

	Monday	Tuesday	Wednesday	Thursday	Friday
Week 1					
Week 2					
Week 3					
Week 4					

Total absent day in a month: _____

Supervisor Name: _____

Signature:_____ Organization/Office Stamp

Date:_____

Wolkite University
We Strive for Wisdom!

2014 E.C / 2022 G.C

Performance Evaluation Sheet to Be Filled by Organization/Office Focal Person

Monthly performance evaluation		Month	
Organization/Office Name			
Organization/Office Supervisor Name			
Student Name			
Student Department	**Biology**		

Please give appropriate percentage value in the box provided out of the total value given for each evaluation criteria

General Performance (25%)		
Punctuality	[5%]	
Reliability	[5%]	
Independence in work	[5%]	
Communication skills	[5%]	
Professionalism	[5%]	

Personal skills (25%)		
Speed of work	[5%]	
Accuracy	[5%]	
Engagement	[5%]	
Do you need him for your work	[5%]	
Cooperation with colleagues	[5%]	

Professional skills (50%)		
NB: The attachments are just 3rd year and they are not graduates		
Technical Skills	[5%]	
Organizational skills	[5%]	
Support of the organization tasks	[5%]	
Responsibility in the task fulfillment	[15%]	
Quality as a team member	[20%]	

Result	**Total percentage**	

Supervisor Signature ***Organization/ Office Stamp***

---------- ----------

Final Report Document Evaluation Sheet Filled by Examiners

1. Student name:_____
2. Programme:_____
3. ID No:_____
4. Place of attachment (Host Organization/office):_____
5. Period of practical attachment:_____

	ASPECTS	Percentage (allocated)	Percentage (obtained)
1	Executive Summary	5%	
2	Acknowledgment and tables of content	5%	
3	Introduction	10%	
4	Practical attachment Main Activities (Ability to use practical application of biological procedures, Problem solving, Originality)	35%	
5	Lesson learned & Experience Gained	20%	
6	Discussion & Recommendations	15%	
7	Conclusion & References	10%	
	Total	100	

Content

The contents to be covered during internship will be decided by interns, companies and department head. Internees are advised to go through major topics to increase their skill and knowledge in the time of their stay at companies.

Evaluation of Interns

In-Company Evaluation of Student Internship

Intern's Name_____ **Date**_____

Intern Supervisor_____

Department / Division_____/_____

This internship started on (date)_____**and was completed on (date)**_____

Do you permit the student to receive a copy of this evaluation? Yes_____ **No** _____

Excellent (Always demonstrates this ability/ consistently exceeds expectations weighting 10-9)

Good (Usually demonstrates this ability/ sometimes exceeds expectations weighting 8-7)

Average (Sometimes demonstrates this ability/meets expectations weighting 6-4)

Poor (Seldomly demonstrates this ability/rarely meets expectations weighing 3-1)

Not applicable/N/A (Not applicable to this internship experience)

Evaluation of personal qualities of the intern observed during internship. Select one evaluation level for each area by marking an "X" under the level that represents the intern's performance. The overall points will be 310 which means the intern have 100% got al points which is 40.

Example: - if the intern gets 250 from 310 points he/ she will get 250*40/310= 32.25.

5

Figure 1: In-Company evaluation of Students Internship for Engineering College

	Excellent	Good	Average	Below Average	N/A	Points earned
Ability to Learn						
Observes and/or pays attention to others	10 9	8 7	6 5 4	3 2 1		
Ask pertinent and purposeful questions	10 9	8 7	6 5 4	3 2 1		
Seek out and utilizes appropriate resources	10 9	8 7	6 5 4	3 2 1		
Accepts responsibility for mistakes and learns from experiences	10 9	8 7	6 5 4	3 2 1		
Open to new experiences; take appropiate risk	10 9	8 7	6 5 4	3 2 1		
Reading/ writing/ computational skills						
Reads/comprehends/follows written materials	10 9	8 7	6 5 4	3 2 1		
Communicates ideas and concepts clearly in writing	10 9	8 7	6 5 4	3 2 1		
Works with mathematical procedures appropriate to the job	10 9	8 7	6 5 4	3 2 1		
Attention to accuracy and detail	10 9	8 7	6 5 4	3 2 1		
Listening and oral communication skill						
Listen to others in an active and attentive manner	10 9	8 7	6 5 4	3 2 1		
Comprehends and follows verbal instructions	10 9	8 7	6 5 4	3 2 1		
Effectivelly participates in meetings or group settings	10 9	8 7	6 5 4	3 2 1		
demonstrate effective verbal communication skills	10 9	8 7	6 5 4	3 2 1		
Creative thinking and problem solving skills						
Seeks to comprehend and understand the "big picture"	10 9	8 7	6 5 4	3 2 1		
Breaks down compex taskes to manageable pieces	10 9	8 7	6 5 4	3 2 1		
Brainstorms/ develops options and ideas	10 9	8 7	6 5 4	3 2 1		
Respects input and ideas from other sources and people	10 9	8 7	6 5 4	3 2 1		
Interpersonal and Teamwork skill						
Relates to co-workers effectively	10 9	8 7	6 5 4	3 2 1		
Manages and resolves conflic to a geam atmosphere	10 9	8 7	6 5 4	3 2 1		
Supports and contributes a team atmosphere	10 9	8 7	6 5 4	3 2 1		
Controls emotions in a manner appropriate for work	10 9	8 7	6 5 4	3 2 1		
Demonstrate assertive but appropriate behaviour	10 9	8 7	6 5 4	3 2 1		
Basic work habits						
Reports to work as scheduled	10 9	8 7	6 5 4	3 2 1		
Is prompt in showing up to work and meeting	10 9	8 7	6 5 4	3 2 1		
Exhibits a postive and constructive attitude	10 9	8 7	6 5 4	3 2 1		
Dress and appearance are appropriate for this organization	10 9	8 7	6 5 4	3 2 1		
Character Attributes						
Bringing a sense of value and integrity to the job	10 9	8 7	6 5 4	3 2 1		
Seeks to serve others	10 9	8 7	6 5 4	3 2 1		
Refrain from gossip/respects the privacy of others	10 9	8 7	6 5 4	3 2 1		
Behaves in an ethical manner	10 9	8 7	6 5 4	3 2 1		
Respects the diversity of co-workers	10 9	8 7	6 5 4	3 2 1		
Overall performance of the intern (circle one)	Excellent	Good	Average	Poor	N/A	

Figure 2: Evaluation Look up table

Comments:

Functional Goal Evaluation (60%)

Describe the student's responsibilities and tasks: -

How was the students' academic and technical background adequate for the job?

☐ Excellent (4) ☐ Very good (3) ☐ Average (2) ☐ Below Average (1) ☐ N/A

Comments: -

How would you rate the quality of the student's work?

☐ Excellent (4) ☐ Very good (3) ☐ Average ☐ Below Average (1) ☐ N/A

How would you rate the quantity of the student's work?

☐ Excellent (4) ☐ Very good (3) ☐ Average ☐ Below Average (1) ☐ N/A

How would you rate the student's productivity?

☐ Excellent (4) ☐ Very good (3) ☐ Average ☐ Below Average (1) ☐ N/A

☐ I have ☐ I have not discussed this assessment with interns.

Evaluator's Signature: _____

Date: _____

Title/ Position: _____

Telephone_____

(Annex I)

Figure 3: Evaluation Form