# ICESHU4
# Architecture Notebook

## 1. Purpose

Software development is becoming more complex with the addition of new features and hardware advances. It is really difficult to develop a software system directly without an architectural view , goals and philosophy. A software architecture identifies major components of a system, their inputs, outputs, and interactions with each other, enabling developers to organize and consider scalability and reuse issues. It simply gives developers a map about what they are trying to build and what are the requirements. It also guides developers to build clean , reusable and portable software systems in an efficient manner. A software architecture document has a critical role to minimize the changes that project costs and deadlines will get out of control and lead to the project failure. Without it, developers will revise the whole project with its components and requirements repeatedly . It serves as a reference for developers, project managers, and other stakeholders involved in the software development process.

The purpose of this document is to present a thorough architectural overview of the Course- Evaluation system called *Iceshu4* by utilizing various architectural views to illustrate different aspects of the system. This notebook aims to document and communicate the important architectural decisions made for the software system.

This document explains significant design decisions, standards, constraints, justifications, and conventions used in the development of the system. It describes the architectural design pattern of the whole system with architectural goals, significant requirements, views , mechanisms. It also points out the assumptions and dependencies that drive architectural decisions, key abstractions of the system and layers of the architectural framework.

## 2. Architectural goals and philosophy

Architectural goals and philosophy for web-based software system guides for designing and building the system that satisfies functional and non-functional requirements and outlines the key goals and principles that system should provide. The goals and philosophy provide a framework for creating a system that is scalable, secure, modular, usable, interoperable, efficient in terms of performance, flexible, and maintainable. In that way, developers can ensure that the software system meets the user needs, can adapt to changing requirements, and be able to easily maintain and update. The architectural goals of Iceshu4 are as follows;

- The system should be scalable in case of changing requirements.
- The system should continue running without any trouble under unexpected conditions. (e.g. Too many users trying to access the system, short-term internet connection problems, data traffic on too many requests etc.)
- The system should be designed under efficient performance manners to minimize data latency and respond to user requests in a fast way.
- Security has an essential part for the system. User passwords should be stored in the database in a hashed form so that it will secure the system from hack operations. Also system data should be secured in a way that each user can view the data s/he is supposed to view according to his/her role (e.g. while admin can see all the data in detail, student can see only his/her own courses and evaluation form results) and unregistered users should not be able to view any system data.
- Graphical User Interface of the system should be easy to use and navigate and also should meet user needs.
- The system should provide an Open-Close Principle. It should be close for modification and open for extensions.
- The architecture should be modular, enabling the system to be easily maintained and updated.
- The architecture should be designed with interoperability so that the system can communicate with other systems and platforms (e.g. system can send email by connecting email services and platforms).
- The architecture of  system components should be easy to understand and implement.
- The system should have clean and efficient code principles to be flexible and maintainable.

## 3. Assumptions and dependencies

Assumptions are beliefs or expectations that are taken for granted during the design and implementation of a software system. These assumptions are generally about the users, the environment, the hardware and software platform. Dependencies can be defined as the relationships between system components and be separated as software dependencies, hardware dependencies and third-party dependencies. To ensure that the software system performs as expected, it is essential to identify and manage both assumptions and dependencies. The dependencies and assumptions of this system are listed below:

- Users should have reliable and stable internet connections to connect the system.
- The database of the system should respond to user requests in an efficient time.
- All user passwords should be stored encrypted.
- It is assumed that users have valid email addresses.
- It is assumed that there are no budget limitations.
- The system should operate continuously, 24 hours a day, 7 days a week.

## 4. Architecturally significant requirements

Architecturally significant requirements are the requirements that play an important role in software system design and architecture. These requirements are generally about system scalability, security, reliability, maintainability and overall system performance. In our system, there are bunch of significant requirements listed below:

- Angular Framework is used to develop the frontend of the project. For user interface design, Primeflex CSS library user which includes common CSS properties. Also Primeng Component library and Angular Material Library for Angular are used to develop system components.
  https://primeng.org/
  https://material.angular.io/
  https://www.primefaces.org/primeflex/
- All user passwords must be encrypted using the HS256 signing algorithm.
- Backend is implemented using Spring and Java programming language. All the user requests done from GUI are handled by RESTful web services via Spring Boot. Spring boot communicates with the database using Jpa repositories and Hibernate ORM for SQL queries.
  https://spring.io/
- All system data security and authentication operations such as login and register are handled by Spring Security. Spring Security is a framework that provides authentication, authorization, and protection against common attacks..
  https://docs.spring.io/spring-security/reference/index.html
- System should be able to respond to user requests in 5 seconds.
- Exception handling must be done on the backend to provide better maintenance.

## 5. Decisions, constraints, and justifications

In this part, we have presented our critical decisions that we have chosen carefully with their constraints and justifications because these decisions affect our all project structure. The possible changes that will be done in the future about these decisions can be costly for us.

To create the backend of our system faster and easier, we have used Spring Boot and benefited from its generic architecture. Spring Boot's out-of-the-box features, such as auto-configuration and embedded servers, make it easy to get started with backend development without having to write a lot of boilerplate code. Its adherence to best practices and industry standards ensures that the resulting application is safe, reliable, and scalable. Its integration with other Spring projects, such as Spring Security and Spring Data, makes it easy to build a complete and cohesive application without having to spend time integrating different components. By using Spring Boot, we can create a robust and efficient backend for our system.

The system requires an authentication mechanism to ensure secure user logins and logouts. We have utilized Spring Security to provide authentication and role-based authorization for our system's login and logout mechanisms. On the one hand, Spring Security is a dependency that adds complexity to the project, and it requires well configuration to work correctly. However, the benefits of using Spring Security far outweigh the constraints. Spring Security is a widely used and well-established authentication and authorization framework that has been thoroughly tested and reviewed by the community. Its comprehensive and customizable security model allows developers to implement a variety of security requirements, such as user authentication, authorization, and session management while providing a high level of security. Also, it has a lot of built-in methods that eliminate the need to write custom code for authentication and authorization. Therefore we can provide security in our project with less effort by using Spring Security.

The secure storage of user passwords is essential in web application development. To prevent password exposure, hashing is a popular technique that converts passwords into unreadable hashes. Also, salting is recommended to add an extra layer of security to the hash. BCrypt provides all these functionalities to store passwords in the database securely. For this reason, we have selected to use BCrypt which has been provided by Spring Security in our system.

# 6. Architectural Mechanisms

## MVC Pattern

In our project, we have implemented the MVC (Model-View-Controller) pattern to keep our code organized and maintainable. This pattern separates the three components: Controller, Model, and View. When a user sends a request, it first goes to the Controller, which then communicates with the Model or View as needed. We have created separate Controller classes for different functions, such as AuthController, UserController, and AccountController, etc. to avoid cluttering the code. After the work has been done in the controller part, if we need to make some changes in the database we call our Model classes like UserRepository. This allows us to easily handle user requests and provide appropriate responses by using related classes in the MVC structure.

## Layered Architecture

We have used Spring Boot for our backend, which led us to adopt a layered architecture. The Presentation Layer contains the View and Controller, while the Model houses all the other layers such as Business and Persistence. The Presentation Layer handles user requests such as login, post, club creation, and logout. If necessary, these requests are converted for other layers in the Model. In the Business Layer, all the business logic is handled, including authorization and validation. In the Persistence Layer, storage logic is stored and executed, and CRUD operations are performed. This approach makes the code more maintainable, compact, and portable.

# 7. Key abstractions

We organized our project components into layers, namely the presentation layer, business layer, persistence layer, and database layer. Our goal was to achieve high cohesion and low coupling by creating a modular design. Instead of a single class structure, we divided our classes into parts.

In the presentation layer, we have controller classes that handle requests and pass them to the next layer if the requests meet the correct structure. Only simple controls are performed in this layer. Our intention was to create an abstraction between this layer and the business layer through the controller classes.

In the business layer, we have service classes that do not have direct access to databases without the next layer. This access limitation creates an abstraction between the business layer and the persistence layer through the repository classes. The repository classes facilitate the connection between the business layer and the database, and we used interfaces as repositories, utilizing the JpaRepository interface.

# 8. Layers or architectural framework

## Presentation Layer

In the Presentation Layer of our project, we have specialized controller classes that handle user requests and pass them on to the next layer, the Business Layer. These controllers are responsible for receiving requests and validating them before forwarding them to the appropriate service classes.

## Business Layer

In the business layer, we have service classes that perform operations on the requests received from the presentation layer and pass them on to the persistence layer. Additionally, these service classes interact with the repository classes located in the persistence layer.

## Persistence Layer

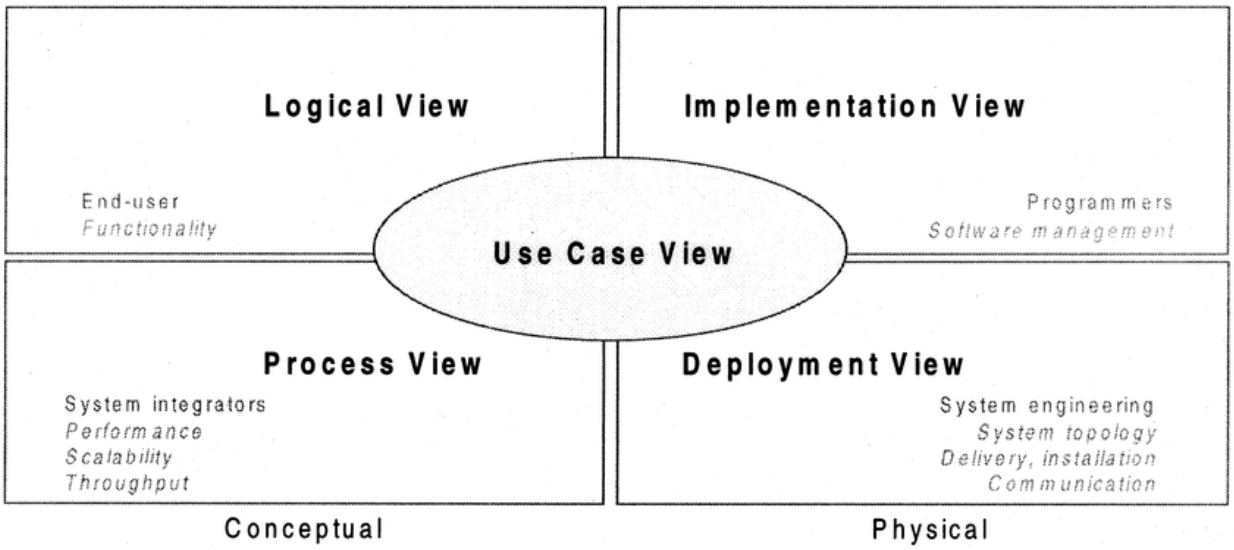
In the Persistence Layer, we have repository classes that handle data operations without directly connecting the project to the database. These classes communicate with the Business Layer and the database. To facilitate this, we used the SpringDataJpa interface which provides essential CRUD functions like find, save, findAll, findBy, etc.

## MVC Architecture

- **Model:**
  We will use Spring Data JPA and PostgreSQL to implement this layer in a correct manner. We will handle all business logic and data processes in this layer. Also, we will use the H2 database for testing operations.
- **View:**
  We chose to use the Angular framework for our front-end architecture, as we have prior experience working with it. This decision helped us to reduce the implementation time due to our familiarity with the framework.
- **Controller:**
  To establish a REST API infrastructure, we utilized the Spring RestController annotation. Each controller has distinct security configurations. By leveraging Spring Framework annotations, we created handlers for GET, POST, PUT, and DELETE requests.

## 9. Architectural views

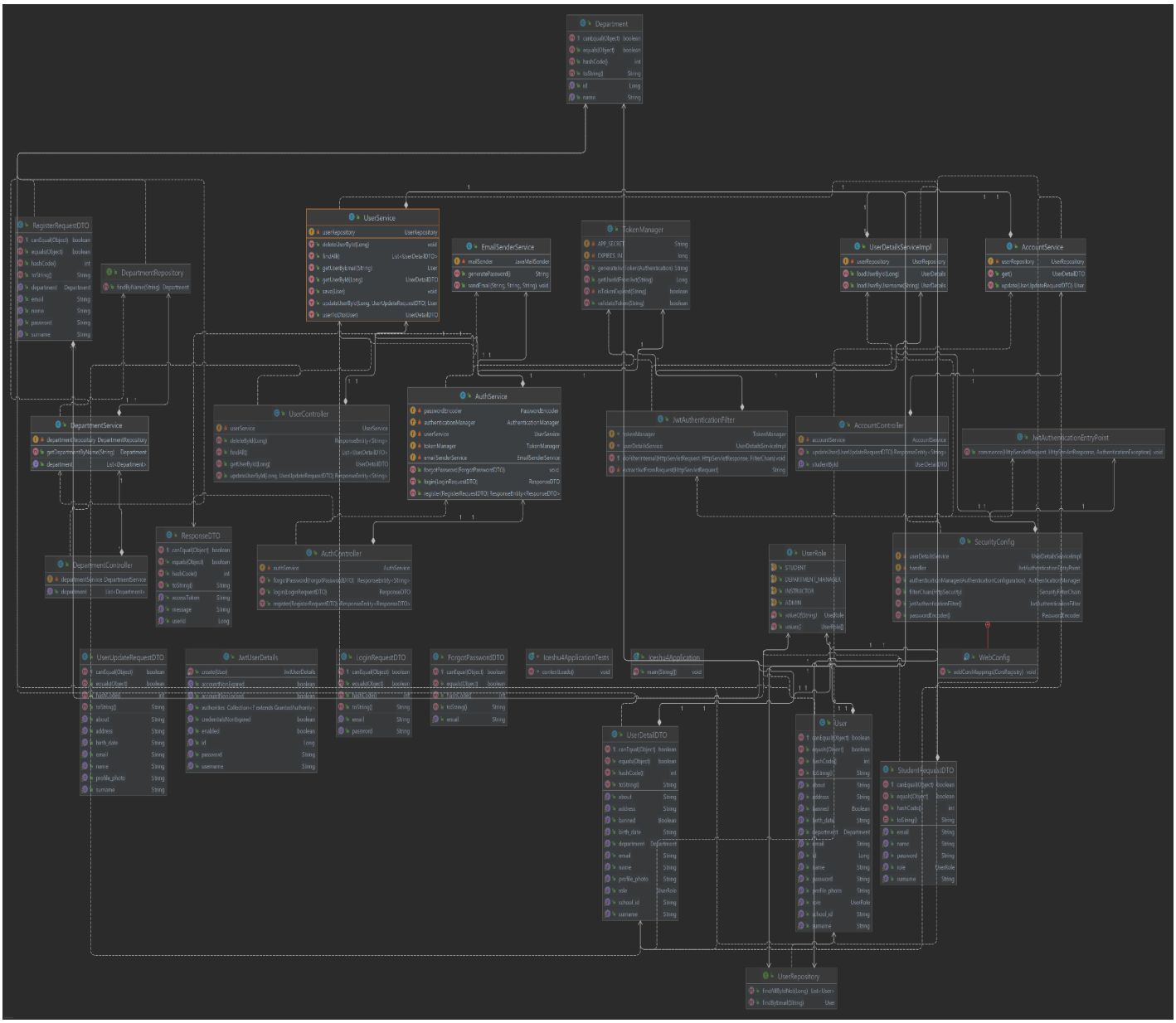
An architectural view is a representation of a software system's architecture that focuses on specific aspects or concerns of the system. It provides a high-level description of the system's structure and behavior, and helps stakeholders to understand and communicate the key features and design decisions of the system.
We can use a 4+1 view model which provides a comprehensive and holistic view of a software system.




- **Logical View:** This view describes the functionality of the system and its components with OOP concepts such as classes and relationships.
- **Development View:** This view describes how the system is developed by including details about tools, processes and methodologies.
- **Process View:** This view describes the system processes and instructions and gives details about data and information flow of the system.
- **Physical View:** This view describes the physical components of the system like hardware, servers, networks and details about connections between them.
- **Scenarios View:** This view provides scenarios that illustrate how the system is used by different types of users such as administrators, students, department managers and instructors.

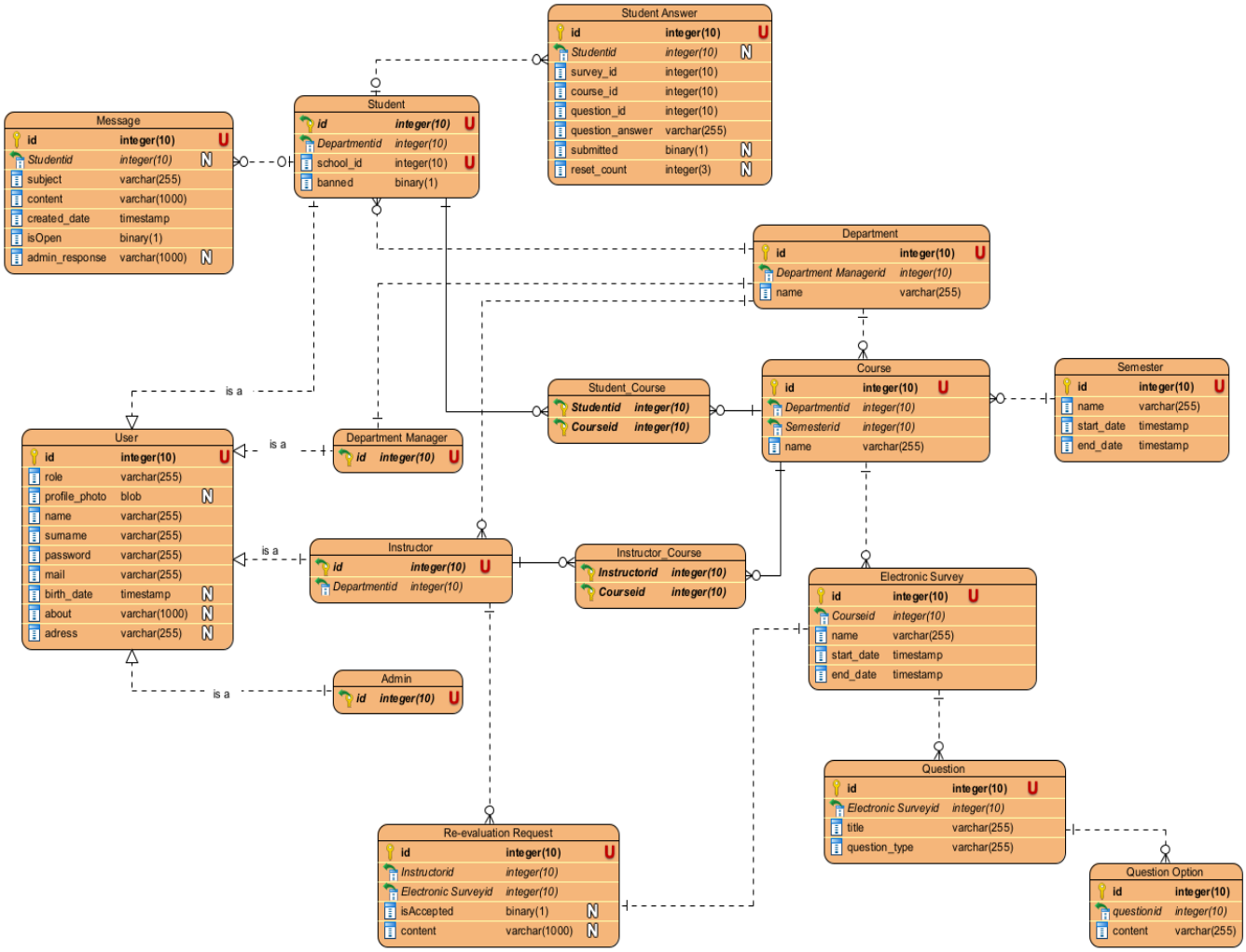
# 1. Logical View

- **Class Diagram**



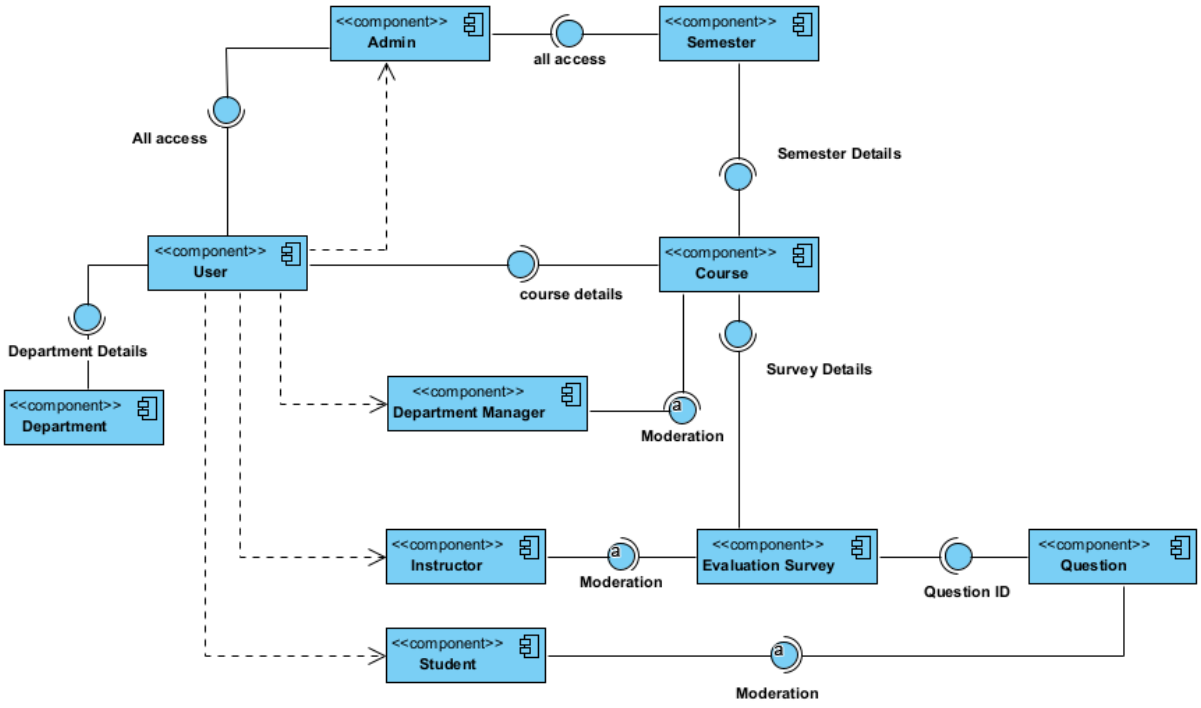

***Note: Since it is too large to view, it is also uploaded as an image***

- **ER Diagram**

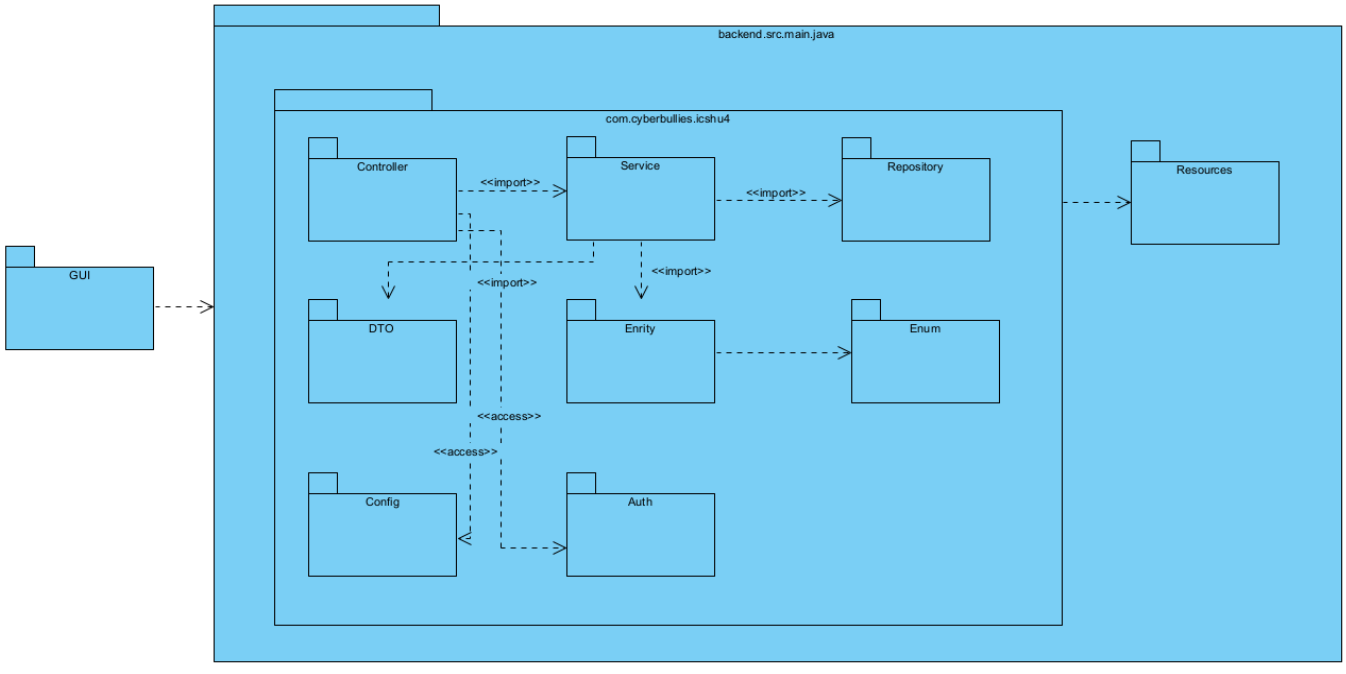

Student Answer
id integer(10) U
Studentid integer(10) N
survey_id integer(10)
course_id integer(10)
question_id integer(10)
question_answer varchar(255)
submitted binary(1) N
reset_count integer(3) N
Student
id integer(10) U
Departmentid integer(10)
school_id integer(10) U
banned binary(1)
Message
id integer(10) U
Studentid integer(10) N
subject varchar(255)
content varchar(1000)
created_date timestamp
isOpen binary(1)
admin_response varchar(1000) N
Department
id integer(10) U
Department Managerid integer(10)
name varchar(255)
Course
id integer(10) U
Departmentid integer(10)
Semesterid integer(10)
name varchar(255)
Semester
id integer(10) U
name varchar(255)
start_date timestamp
end_date timestamp
Student_Course
Studentid integer(10)
Courseid integer(10)
is a
User
id integer(10) U
role varchar(255)
profile_photo blob N
name varchar(255)
sumame varchar(255)
password varchar(255)
mail varchar(255)
birth_date timestamp N
about varchar(1000) N
adress varchar(255) N
Department Manager
id integer(10) U
Instructor
id integer(10) U
Departmentid integer(10)
Instructor_Course
Instructorid integer(10)
Courseid integer(10)
Electronic Survey
id integer(10) U
Courseid integer(10)
name varchar(255)
start_date timestamp
end_date timestamp
Admin
id integer(10) U
Question
id integer(10) U
Electronic Surveyid integer(10)
title varchar(255)
question_type varchar(255)
Question Option
id integer(10)
questionid integer(10)
content varchar(255)
Re-evaluation Request
id integer(10) U
Instructorid integer(10)
Electronic Surveyid integer(10)
isAccepted binary(1) N
content varchar(1000) N

## 2. Development View

- **Component Diagram**

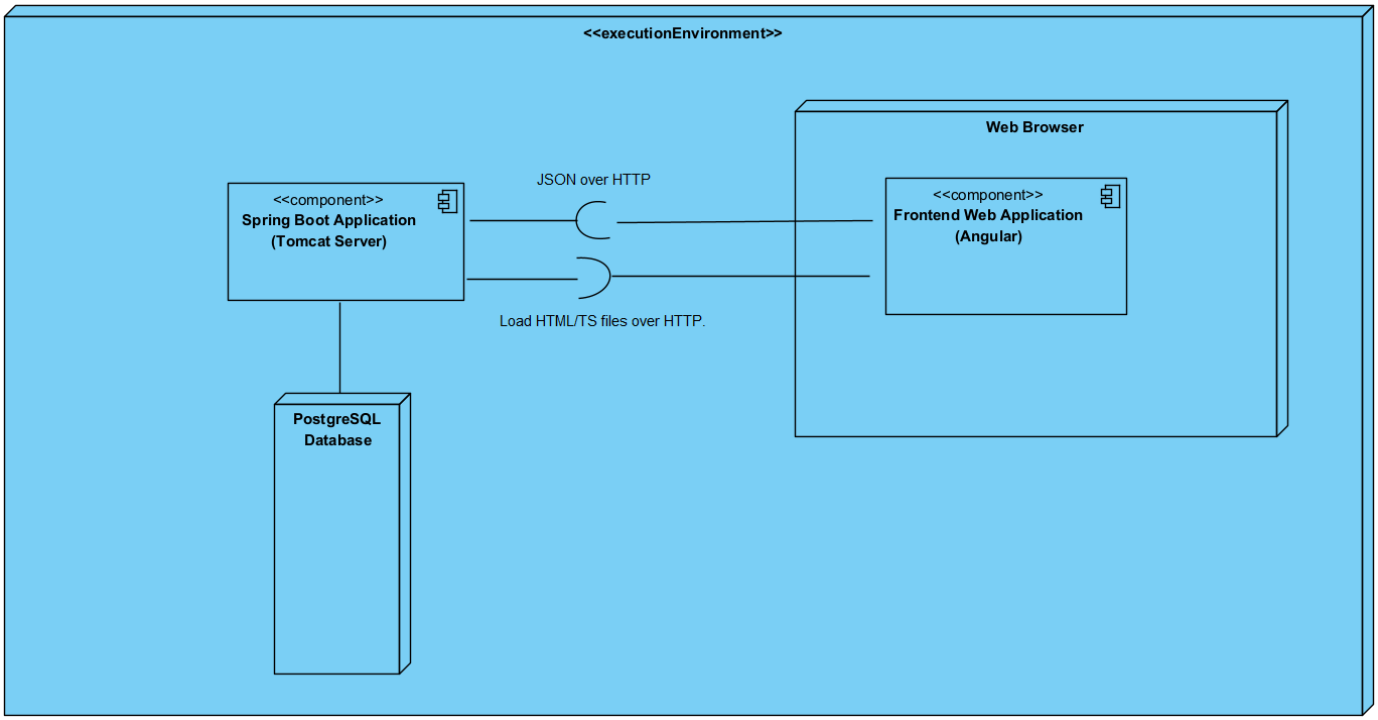



- **Package Diagram**



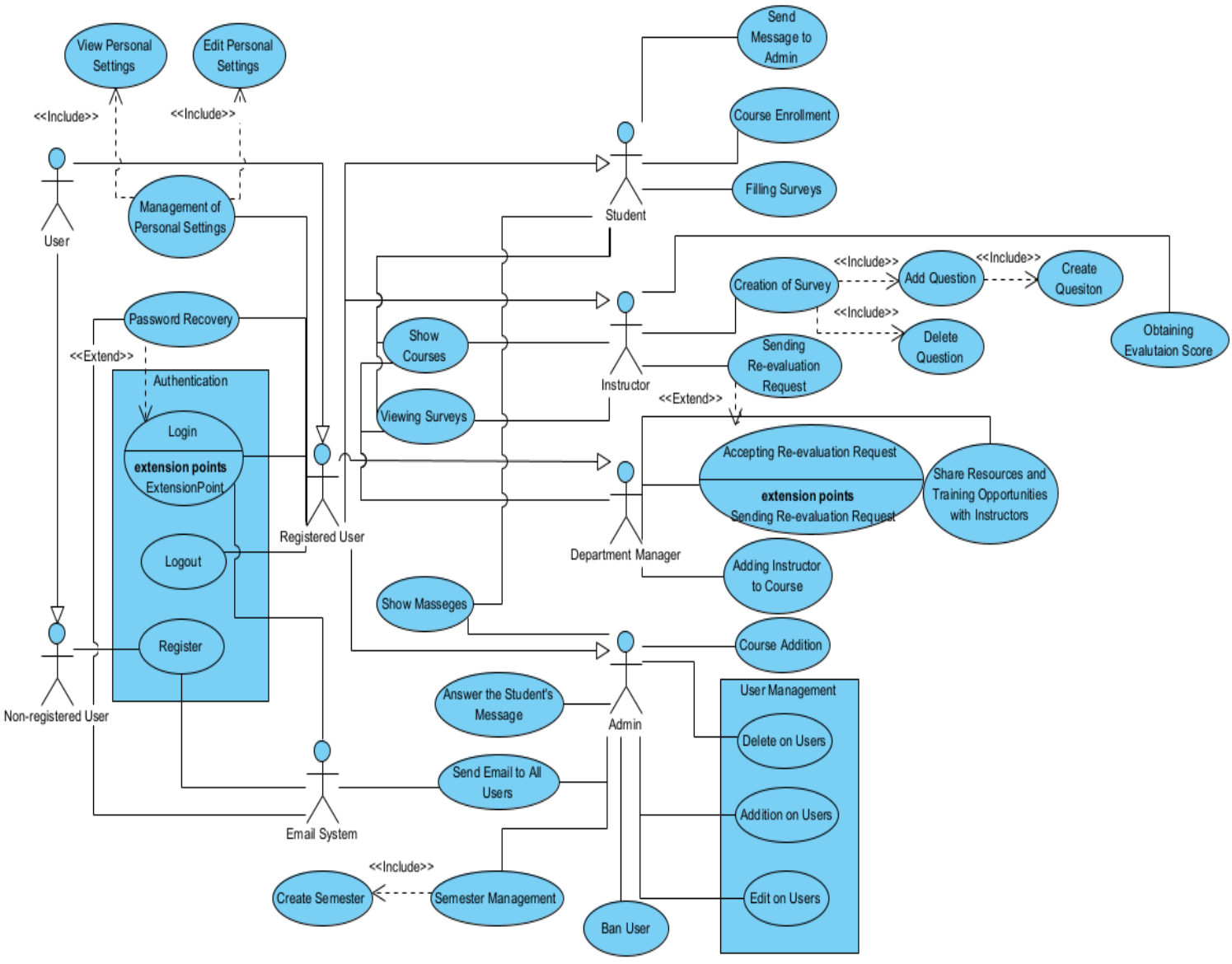
## 3. Physical View

- **Deployment Diagram**



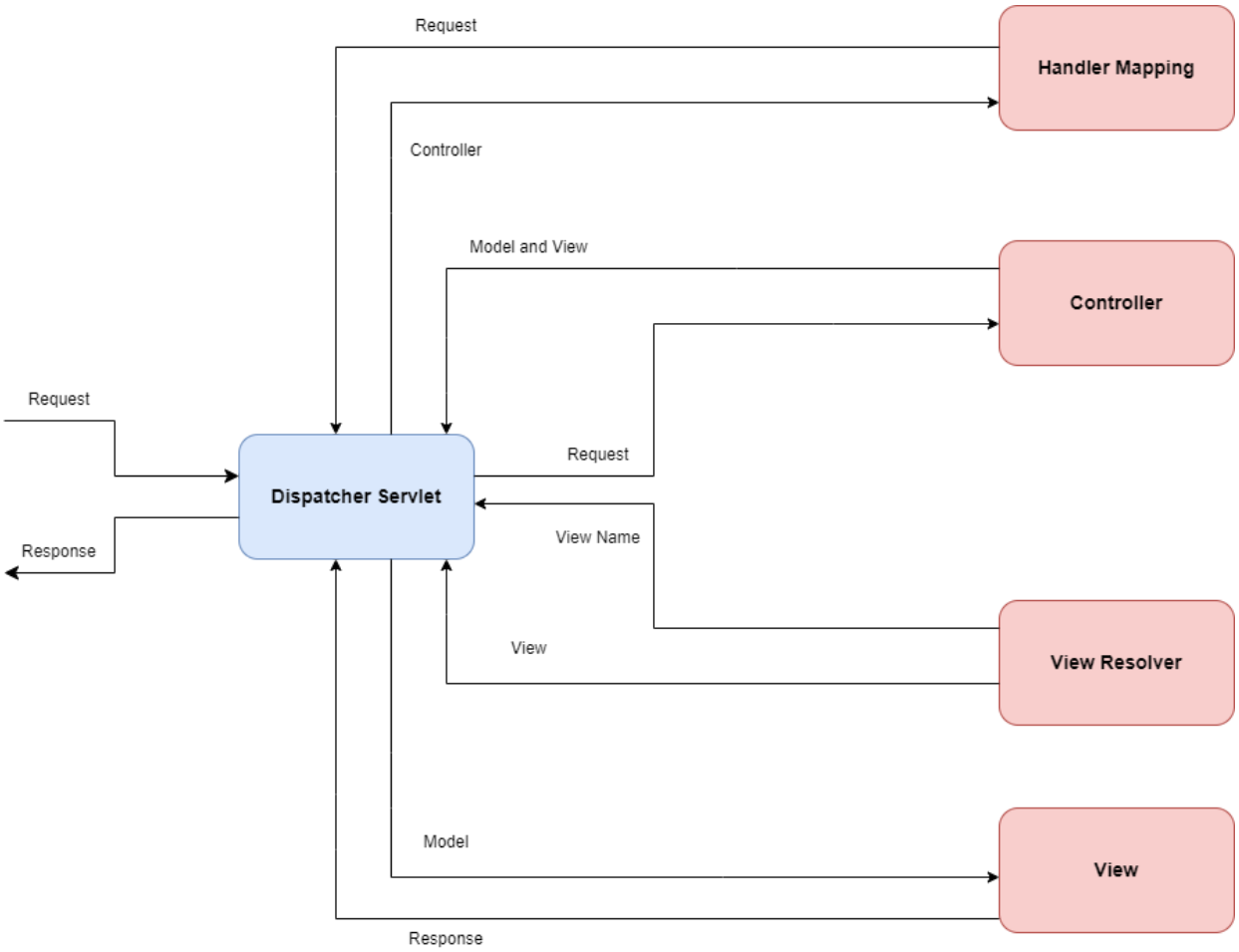
## 4. Scenario View

- **Use Case Diagram**



## 5. Conceptual View

- **Block Diagram**




## 6. Process View

Will be given in the next delivery.

# 10. Traceability Table

| Traceability Table | Numan Kafadar | Mustafa Çağrı Korkmaz | Yunus Emre Terzi | Umut Güngör | Osman Faruk Derdiyok |
|---|---|---|---|---|---|
| Purpose | X | | | | |
| Architectural Goals and Philosophy | X | | | | |
| Assumptions and Dependencies | X | | | | |
| Architecturally Significant Requirements | X | | | | |
| Decisions, Constraints and Justifications | | X | | | |
| Architectural Mechanisms | | X | | | |
| Key Abstractions | | X | | | |
| Layers or architectural framework | | X | | | |
| Architecturally Views | X | X | | | |
| Class Diagram | X | | | | |
| ER Diagram | X | X | | | |
| Component Diagram | X | | | | |
| Package Diagram | X | | | | |
| Deployment Diagram | | X | | | |
| Use Case Diagram | | | X | | |
| Block Diagram | | X | | | |