ICES4HU
Architecture Notebook

1. Purpose

Prior to beginning the construction of a system, making sound architectural selections is crucial in avoiding future catastrophic blunders. The goals and requirements of the client are mostly satisfied by this procedure, which is carried out prior to the system's creation.

The system's architectural components, framework components, system philosophy, limits, and decisions are all established as a consequence of the creation of this document.

The first stage of the software development process is architectural design. It displays the primary structural elements as well as their connections. An architectural model of the system is generated as a result of this design.

2. Architectural goals and philosophy

ICES4HU is a web application. It does not require high processing power. It works on most devices that can access the web.

In order for users to use all functions of the application easily, it should have a clean, easy-to-understand interface.

Programmers should write clean and understandable code in order to minimize the possible problems that may be encountered during the coding phase of the application and to be comfortable in coding the updates to be made after the application's release.

The maximum load that the application can take instantly should be calculated considering the potential users, and the system should be designed to be robust enough to handle this traffic.

3. Assumptions and dependencies

The system that emerges as a result of the coding should meet the user requests and comply with the plans made at the beginning with the revisions.

The system should be accessible at a rate of 99.9%. Errors that may crash the system should be detected beforehand, and maintenance should be done before shutting down the system, if possible.

Changing the page in the system, using the functions, reading and writing the database should be fast and secure.

Identity transactions in the system will be handled with Spring Security inside Spring. This framework contains authentication, authorization and other important security operations.

Important information such as the password entered into the system should be stored in an encrypted form.

4. Architecturally significant requirements

Bootstrap will be used with common CSS templates in our user interfaces

The system will have the standard RESTful API.

Logging operations will be done in detail so that the errors that occur in the system can be easily noticed.

The system should respond to the request resulting from any transaction within 7 seconds at most.

Important passwords such as the password in the system will be encrypted by BCrypt with a random salt process. Even if this data is captured as a result of a possible attack, security weakness will not occur because the actual passwords are not revealed.

5. Decisions, constraints, and justifications

Using Spring Boot framework in Java language is very helpful in making such an application. If it is easy to include the features and extra add-ons it contains, it is a great convenience. On the backend side, the application will stand up in this way.

The application must be used by logging into the system. And there are multiple users in the system with different authorizations. Therefore, the issue of authentication and authorization is important for the application. Spring Security in Spring meets these needs of the system and makes the programmer's job easier during the coding phase. We will actively use Spring Security in our project.

Storing the passwords to be used while logging into the system in the database is one of the most important issues. Although we take the necessary precautions to prevent possible attacks, it is not impossible to steal passwords. Therefore, by storing the passwords in our system in an encrypted manner, we prevent the passwords from being revealed even as a result of a possible theft. Here we will use BCrypt for encryption. Thanks to the random salting in it, even if it is a difficult possibility with the Rainbow attack, when the password is cracked and the password is revealed, it will be even more difficult to access the real password from the password created by the salting process.

6. Architectural Mechanisms

MVC(Model-View-Controller) divides the system into 3 in general. Model layer includes entities (database), View part contains the parts seen by the user (frontend), the controller part includes functions in the system (doing necessary operations in database with commands from view). Detailed information can be found in the “Layers or Architectural Network” section.

The tiered structure is the result of the developers dividing the application into tiers with similar features together. In this way, it becomes an efficient process as a result of performing the necessary operations to the relevant layer instead of the entire application. It is possible to add, remove or change layers as needed. Detailed information can be found in the “Layers or Architectural Network” section.

7. Key abstractions

The phrase "key abstractions" describes the attention and abstraction of ideas, features, or functions that are essential for solving a particular issue or problem. These abstractions are employed to simplify a more intricate structure. It is crucial while creating and comprehending complicated systems. These abstractions aid in seeing the broad picture by breaking complicated systems down into simpler, easier-to-understand components.

Four distinct layers were employed in this project.

Controller: By designing the application's entry point, controller classes control user requests. This ensures communication across the project's many layers and facilitates more orderly management of the code.

Service: Service classes act as the levels on which the business logic is located, making the code more understandable and modular. Additionally, using the Service classes makes it simpler to test and maintain business logic.

Repository: Repository classes are the levels in a database where operations are done. Here, data is read, written, and deleted. As a result, database activities are better organized in the code, making the project simpler to manage.

Entity: By modeling the data stored in the database, entity classes are utilized to carry out database operations. By guaranteeing the consistency and correctness of the data, the project's quality is improved and database operations occur in the code more often.

8. Layers or architectural framework

We used Spring Boot framework in our application. Three layers make up the typical architecture of Spring Boot:

Presentation Layer: Access to the user interface is made possible via the presentation layer. These classes, known as controller classes, are found in this layer and handle user requests. Controller classes take in user requests, carry out the required tasks, and then provide the outcome. Classes that process this data in our web application, for instance, are found in this layer when the user fills out a form.

Service Layer: This layer is where business logic is put into practice. The business logic utilized in the Controller classes is performed on this layer, which also houses the Service classes. Service classes, which are frequently used in Controller classes, prepare and provide results by modifying data, utilizing databases, or carrying out other actions. Classes that carry out tasks like adding items to the basket, processing payments, or confirming orders, for instance, are part of this layer of an e-commerce application.

Persistence Layer: Database activities are carried out at the layer known as the persistence layer. In this layer, repository classes are located, and it is through these classes that database operations are carried out. Repository classes read, write, or delete data by accessing the database. In this way, it is ensured that the data used by the Service classes are kept and managed in the database.

9. Architectural views

The architectural view is the manner in which a software system is structured and organized, which includes its various components, how they interact with one another, and how they are put into use. This viewpoint is crucial for comprehending how the system is designed and how it functions, and it may be split into multiple classifications, such as logical, operational, and use cases, to help better grasp the system's different elements and how they work together.

One vital phase of the software development process is providing comprehensive specifications for all project features, including diagrams. Yet, presenting all elements, like equipment and structural features, in a single diagram may not be feasible. Hence, the 4+1 architectural view model is utilized to depict and explain every detail of a project in all dimensions.

The 4+1 architectural view is a model of software architecture comprising five distinct perspectives, with each view reflecting a unique aspect of the system under construction. The four primary views, including logical, development, process, and physical, are employed to define the functional and non-functional needs of the system, its development process, and the manner in which it is physically deployed. In contrast, the "+1" perspective, also known as the use case view, clarifies the system's purpose, revealing how end-users will utilize it and the various scenarios in which it will be employed.

Logical View: The logical view of a software system focuses on the system's functions and how they are used by end-users. UML diagrams are utilized to represent this view, which describes the structure and behavior of important components of the system, such as critical interfaces, significant classes, subsystems, and the relationships between them. This view typically employs Class and ER Diagrams as structural diagrams, with the Class Diagram showing the system's classes, their attributes, operations or methods, and their relationships with other classes.

Process View: The process view focuses on the system’s run-time behavior and deals with the system’s dynamic elements. It explains the system processes and how they communicate. Concurrency, distribution, integrator, performance, and scalability are all addressed in the process view. The sequence diagram, communication diagram, and activity diagram are all UML diagrams that can be used to describe a process view.

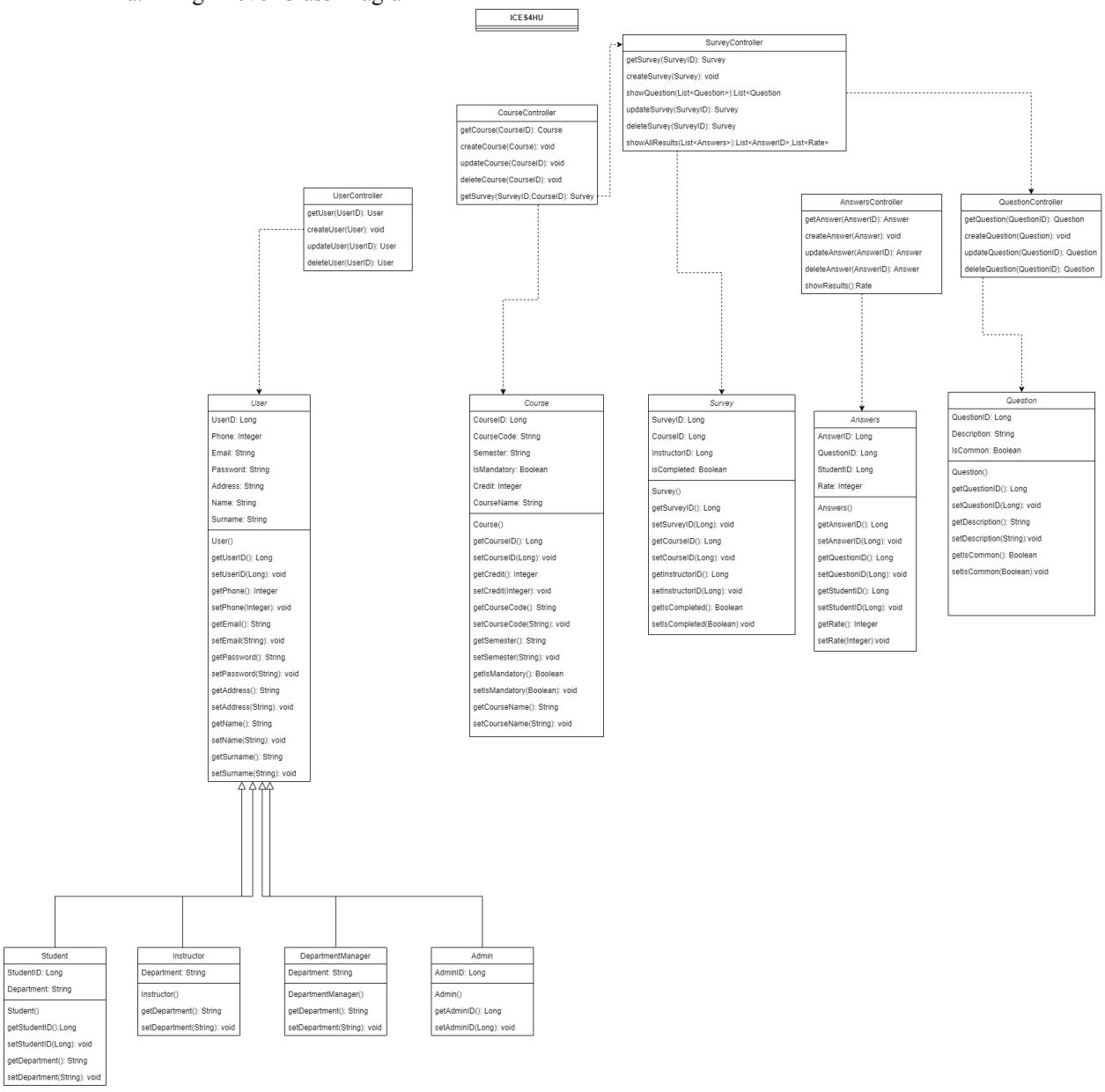
Development View: The development view depicts a system from the standpoint of a programmer and is concerned with software administration. The implementation view is another name for this view. It describes system components using the UML Component diagram. The Package diagram is one of the UML diagrams used to depict the development view.

Physical View: The physical view portrays the system from the perspective of a system engineer. The physical layer, it is concerned with the topology of software components as well as the physical connections between these components. The deployment view is another name for this view. The deployment diagram is one of the UML diagrams used to depict the physical perspective.

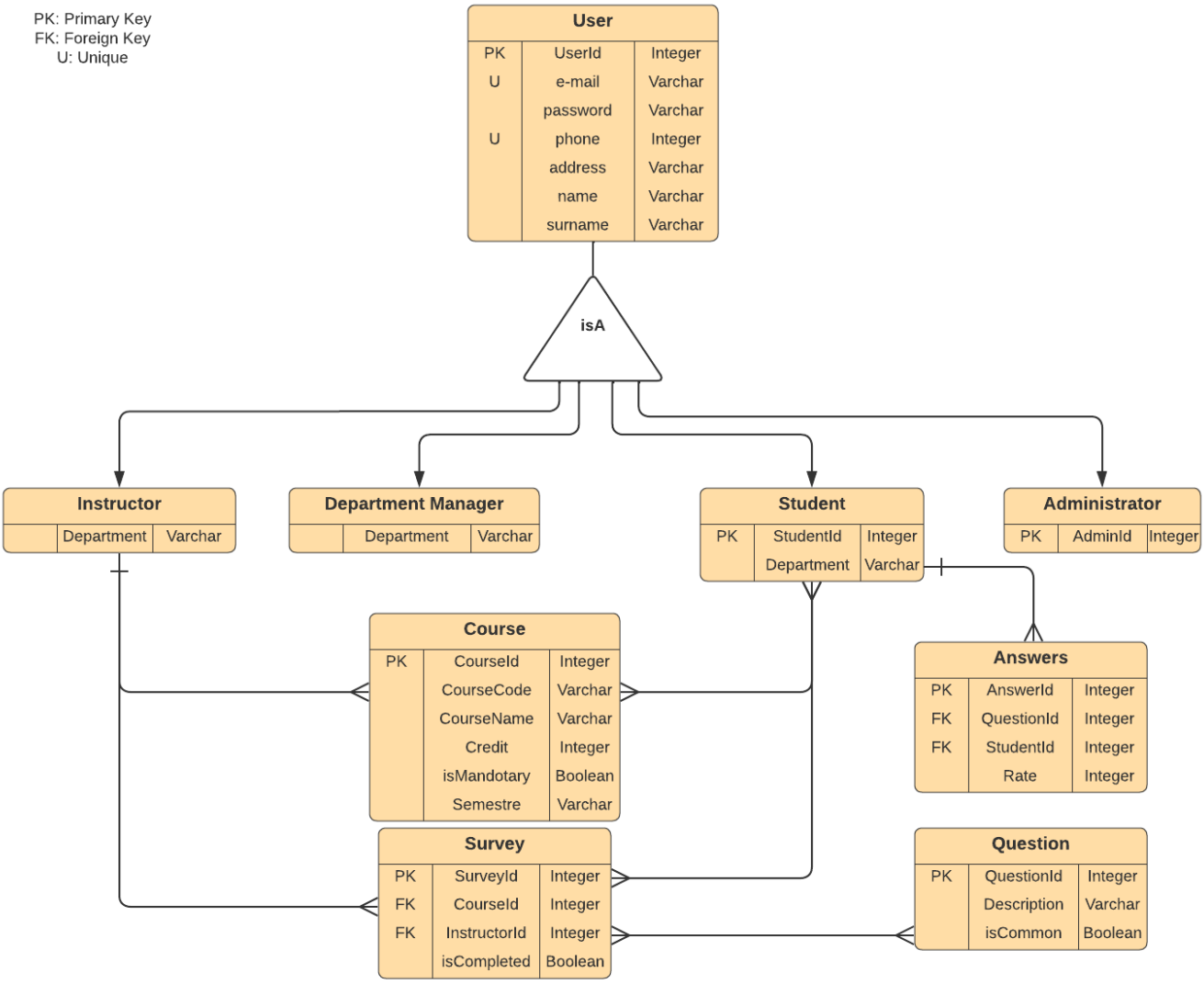
Scenarios: A small number of use cases, or scenarios, that become the fifth view, are used to illustrate the description of architecture. Sequences of interactions between objects and processes are described in the scenarios. They are used to identify architectural aspects as well as to demonstrate and assess the design of the architecture. They can also be used as a starting point for architecture prototype testing. The use case view is another name for this view.

In the following sections, those diagrams are given. Full size Class, Component, Package, Deployment and Block Diagrams can be found.

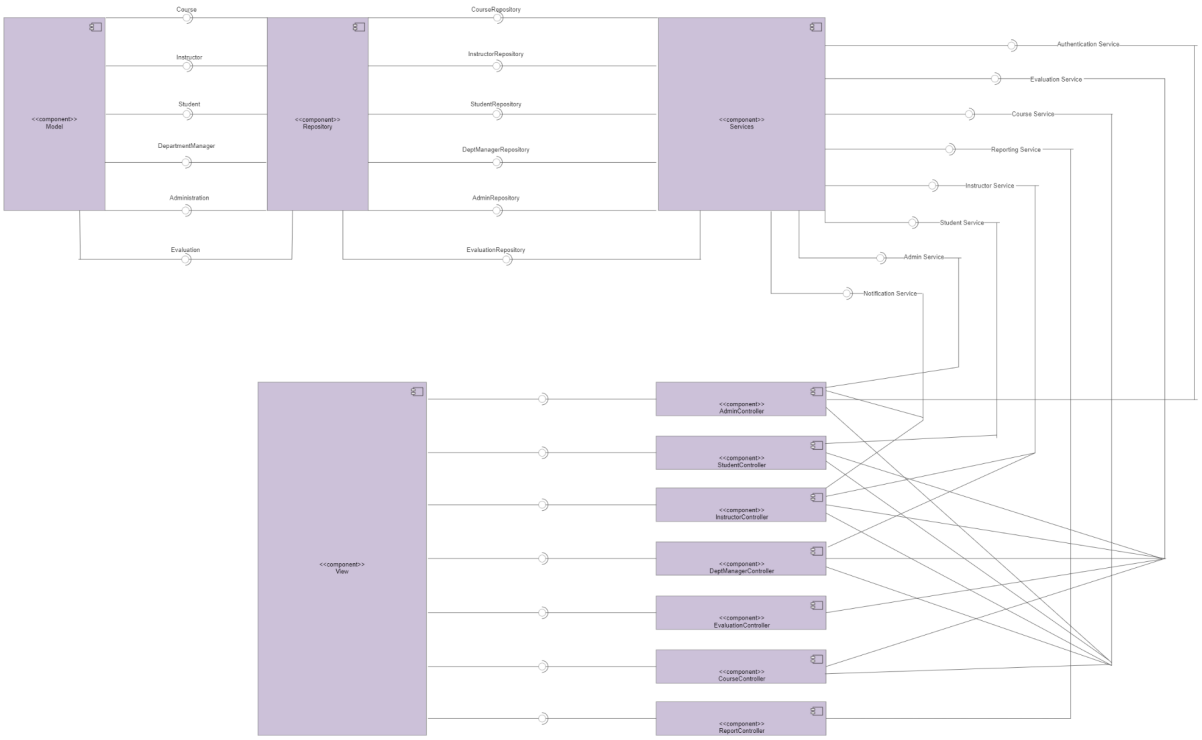
1. Logical View
    a. High Level Class Diagram



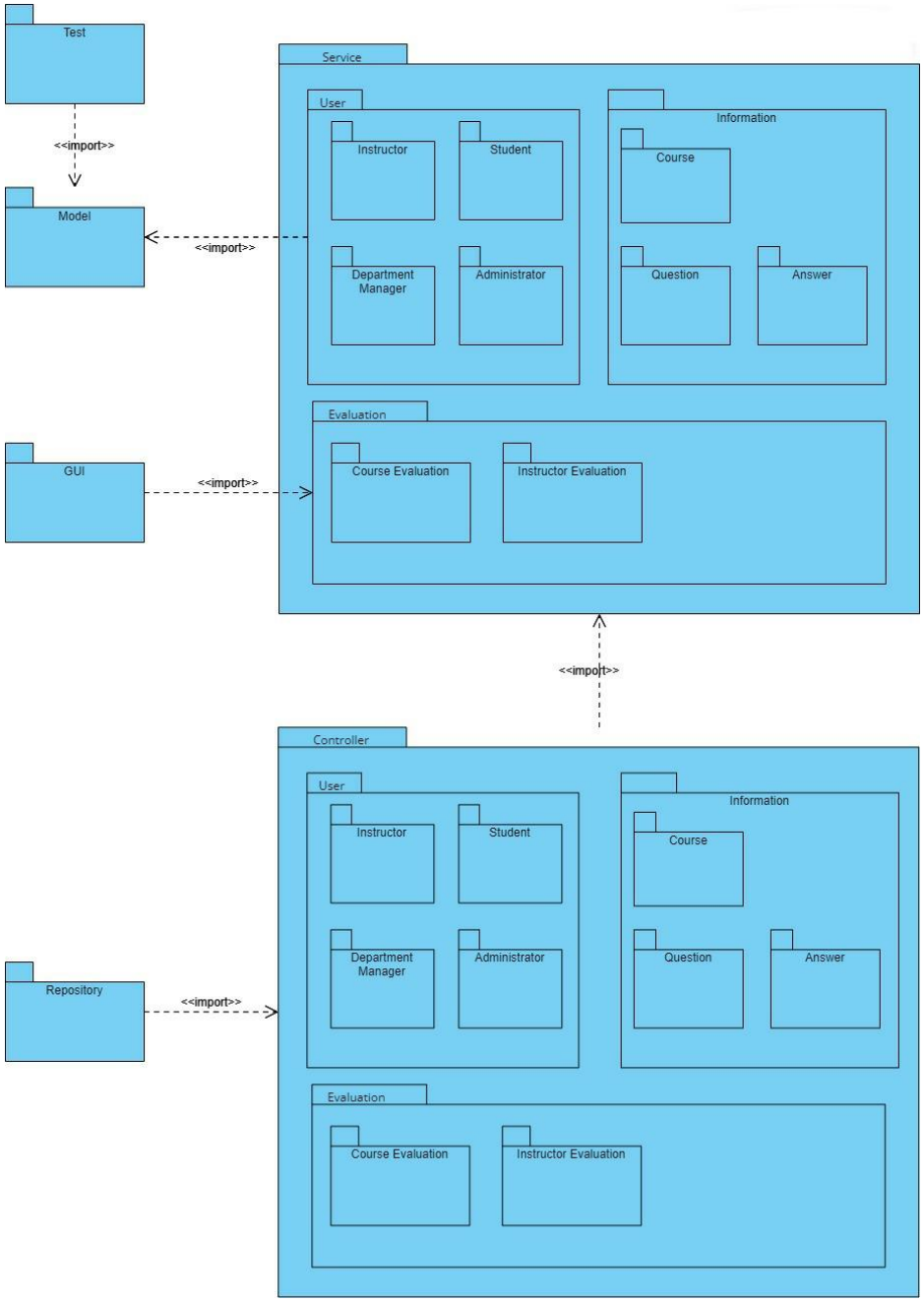
b. ER Diagram


PK: Primary Key
FK: Foreign Key
U: Unique
User
PK UserId Integer
U e-mail Varchar
password Varchar
U phone Integer
address Varchar
name Varchar
surname Varchar
isA
Instructor
Department Varchar
Department Manager
Department Varchar
Student
PK StudentId Integer
Department Varchar
Administrator
PK AdminId Integer
Course
PK CourseId Integer
CourseCode Varchar
CourseName Varchar
Credit Integer
isMandotary Boolean
Semestre Varchar
Answers
PK AnswerId Integer
FK QuestionId Integer
FK StudentId Integer
Rate Integer
Survey
PK SurveyId Integer
FK CourseId Integer
FK InstructorId Integer
isCompleted Boolean
Question
PK QuestionId Integer
Description Varchar
isCommon Boolean

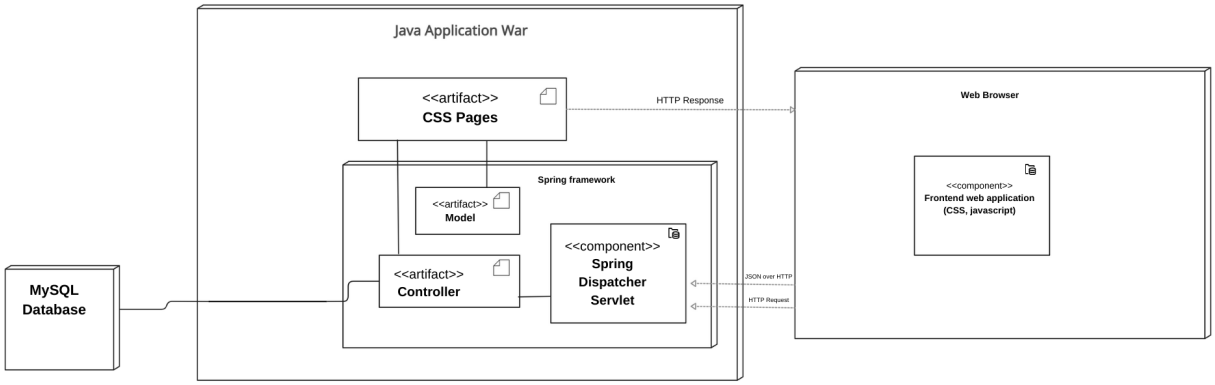
2. Implementation View
    a. Component Diagram



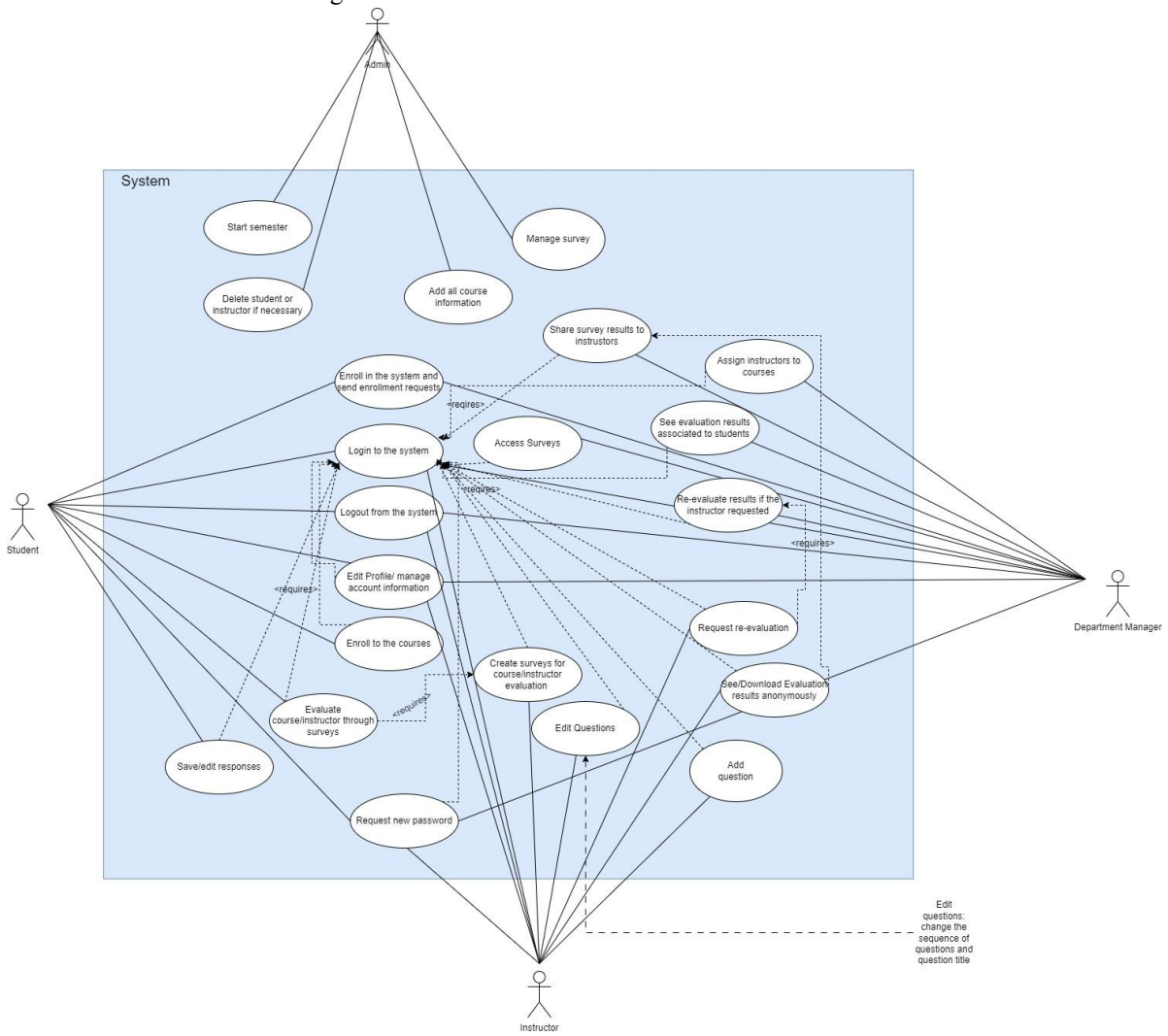
b. Package Diagram



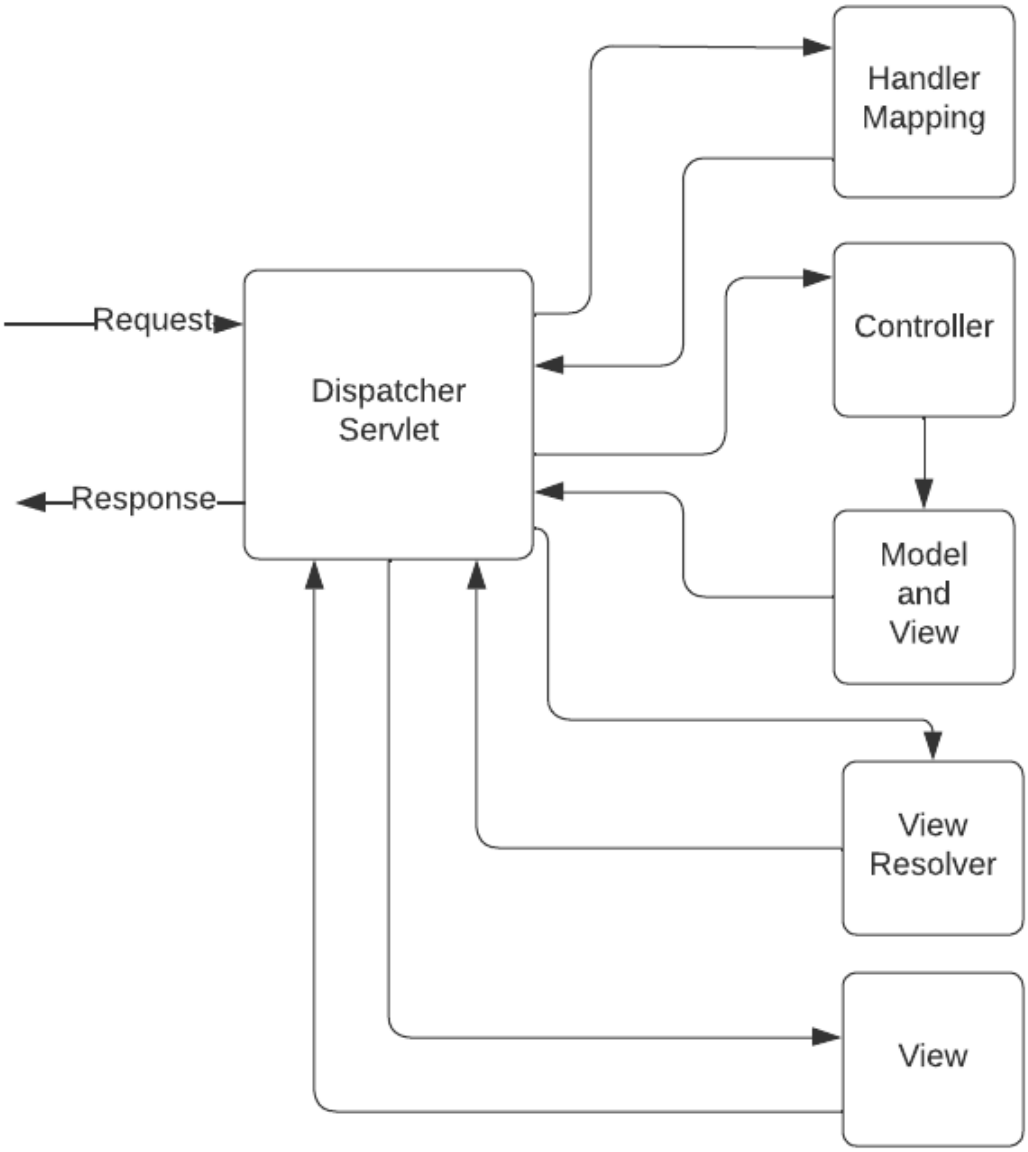
3. Physical View
    a. Deployment Diagram



4. Scenario View
    a. Use Case Diagram



5. Conceptual View
    a. Block Diagram



6. Process View
   This view has a Sequence Diagram and it will be given in the next delivery

	Deniz Erkin Kasaplı (Software Project Manager)	Berkay Barulay (Software Analyst)	Ahmet Karaca (Software Architect)	Gizem Aleyna Tuzcu (Software Configuration Manager)	Hasan Malkoç (Software Tester)
Architecture Notebook	***X***	***X***	***X***	***X***	***X***
Risk List	***X***	***X***		***X***	***X***
Risk Management Report	***X***	***X***			
Configuration and Change Management Report				***X***	***X***
Block Diagram			***X***		
Component Diagram				***X***	
Deployment Diagram		***X***			

High Level Class Diagram	***X***				
Package Diagram					***X***