ICES4HU
Architecture Notebook

1. Purpose

This document outlines the system architecture and provides a reference point for stakeholders, architects, developers, and other team members to ensure alignment and consistency with project goals and objectives. It provides a platform for collaboration and knowledge sharing among architects, enabling them to leverage each other's expertise and experience to make informed decisions and achieve successful results. The architecture notebook gives an overview of how the system functions without delving into the specifics of its implementation. Its main emphasis is on the conduct of individual components within the system.

Creating an abstract summary for a system in its early stages is beneficial as it aids in identifying the functional and non-functional components of system requirements. This document streamlines software requirements analysis by breaking down the system's fundamental aspects into the clear and concise language to make it easily recognizable. This document facilitates software requirements analysis by outlining the system's philosophy, decisions, constraints, justifications, significant elements, and any other aspects of the system that shape the design and implementation.

The architectural notebook is a crucial tool in designing and developing a system that meets the organization's and its stakeholders' needs. It ensures that all team members are on the same page and that the project is aligned with long-term goals. Ultimately, it supports the project's success and is essential for effective communication and collaboration throughout the development process. Additionally, this document visualizes all content with diagrams.

2. Architectural goals and philosophy

The architectural goals and philosophy passages provide important guidance for designing a robust and maintainable system. By setting clear goals and identifying critical issues, developers can ensure that the system will meet the needs of its users and operate effectively under a variety of conditions. In addition, by emphasizing the importance of understanding the motivations behind architectural decisions, developers can ensure that the system is implemented in a consistent and coherent manner.
The architecture of the ICES4HU system should prioritize the needs of its users by providing a user-friendly interface that is easy to learn and navigate. The system should also be flexible enough to adapt to the specific needs of Hacettepe University, including legacy systems and performance requirements. Additionally, the system should be designed with maintainability in mind, allowing for easy modification and updating as needed.

Architectural Goals:

- **User-Friendliness:** ICES4HU should have a clear and intuitive user interface that supports a variety of user roles and permissions, making it easy to use and navigate.
- **Scalability:** The system should be designed to be scalable, allowing it to handle a large number of courses and instructors, as well as a growing number of students.
- **Maintainability:** The system should be maintainable even after a long period of time, and it should be easy to modify in the future as new instructors or courses are added, or changes in evaluation criteria occur.
- **Integration:** The system should be able to integrate with legacy systems and other external services, allowing for seamless data exchange and cross-platform functionality.
- **Robustness:** The system should be able to function effectively under a variety of conditions, including heavy usage and short-term internet connection problems, without compromising performance or reliability.
- **Modifiability:** The architecture should support the separation of presentation and interaction from the system data, allowing for greater flexibility and modifiability in the future.

- **Learnability:** The system should be easy to learn how to use, with clear and intuitive interfaces.
- **Authentication and Authorization:** Users must be logged in to access and perform any operations on the system, including viewing evaluations or submitting evaluations. The system should provide proper authentication and authorization for this purpose.
- **Reporting:** The system should provide reporting capabilities to allow for the analysis of evaluation results and feedback.
- **Integration with Existing Systems:** The system should be able to integrate with existing university systems, such as student information systems and learning management systems, to streamline processes and improve efficiency.

3. Assumptions and dependencies

Assumptions:

- Students will be required to evaluate their instructors and courses using the ICES4HU system.
- The system will be used by a large number of students and instructors.
- The system will be integrated with Hacettepe University's existing student information system.
- The evaluation results will be used to inform decisions related to the course and instructor improvements.
- The system has no budget limitations, allowing the project team to allocate the necessary resources to ensure the system meets desired requirements. This may impact architectural decisions such as technology stack and infrastructure selection.
- The team members possess sufficient knowledge of databases, front-end, and back-end technologies. The system's architecture should leverage this knowledge to ensure optimal performance, and scalability and handle large volumes of data, traffic, and user activity.
- The team members are available for at least 9 hours per week, impacting project planning, scheduling, team size, and resource allocation.

Dependencies:

- The system will need to be accessible via the internet and have reliable network connectivity.
- The system will need to be secure to ensure confidentiality and data integrity.
- The system will need to be scalable to handle a large number of evaluations and users.
- The system will need to be compatible with Hacettepe University's existing technology infrastructure.
- The development team will need to have a thorough understanding of the evaluation process and the needs of students and instructors.
- The system will need to comply with any relevant regulations or policies related to data privacy and security.
- The system will need to be tested for usability and user acceptance.
- The system will need to be able to handle high traffic loads during peak evaluation periods.
- The system will need to have a user-friendly interface that is accessible to all users, including those with disabilities.
- The development team will need to have access to appropriate development tools and software licenses.
- The system will need to have appropriate data storage and retrieval mechanisms to ensure efficient data management.

4. Architecturally significant requirements

Architecturally Significant Requirements refer to a specific set of requirements that have significant effects, point to trade-offs and architecturally significant assumptions. These requirements are a subset of both functional and non-functional requirements. Organizations typically focus on identifying and selecting these requirements during their software development process. It allows them to understand better the interactions between different systems and how they impact the overall architecture. Ultimately, identifying and addressing these requirements before the development process ensures overall compliance with architectural standards and reduces the risk of costly rework.

- The system should respond to user requests (e.g. filling out surveys, viewing survey results) in a timely manner to ensure a good user experience. The system should aim to provide a maximum response time of 1 second for most user interactions[1], such as submitting a survey response or viewing survey results.
- The system will encrypt all user passwords using the SHA-256 algorithm[2], as it is difficult to break but easy to encrypt. This means that the system will not store passwords in their original form, providing an additional layer of security. By using SHA-256, the system is able to safeguard user information and minimize the risk of data breaches.
- The back-end of the system should be handled with ASP.NET[3], which follows the MVC architecture[4] and allows for cross-platform migration. This reduces coding time and allows for the optimization of implementation and maintainability.
- The system should provide at least 99% uptime for availability[5] and should be able to recover quickly if it goes down.
- The system should be scalable[6], allowing for increased or decreased performance and cost in response to changes in demand.
- The system should not activate or delete user accounts without administrator approval to ensure the security[7] and integrity of user data.
- The system should allow instructors to view evaluation results without compromising the anonymity of the students who provided them.
- The system should allow students to enroll in courses and evaluate them, but not access evaluation results for other courses they did not enroll in.

5. Decisions, constraints, and justifications

- Decisions have been made regarding architectural approaches and technological tools. The system will be developed using the Model-View-Controller (MVC) design pattern. React and ASP.NET will be used on the client and server side respectively, with ASP.NET providing server-side rendering and React handling the client-side rendering.
- Database tables will be designed in accordance with the principles of relational database design. This decision has been made to ensure data integrity, consistency, and ease of maintenance.
- Various constraints have been imposed on security-related issues. Passwords will be salted and hashed with the SHA256 algorithm and stored in the database to prevent techniques such as Rainbow Table attacks. In addition, data transmission during user login will be encrypted using the HTTPS protocol. User authentication will be performed for operations performed while the session is open, and session time limits will be set.

[1] https://www.sciencedirect.com/topics/computer-science/system-response-time
[2] https://en.wikipedia.org/wiki/SHA-2
[3] https://dotnet.microsoft.com/en-us/apps/aspnet
[4] https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
[5] https://en.wikipedia.org/wiki/High_availability_software
[6] https://www.sentinelone.com/blog/scalable-architecture/
[7] https://vedcraft.com/architecture/top-10-things-to-know-about-security-as-a-software-architect/

- Taking into account the scalability of the system, it will be designed to be compatible with a distributed architecture. Various performance optimizations will be made to cope with high user traffic.
    - Caching: Caching frequently accessed data can reduce the amount of time and resources required to retrieve it from the database.
    - Load Balancing: Distributing the incoming traffic across multiple servers can help to evenly distribute the load and prevent any one server from becoming overwhelmed.
    - Database Optimization: Optimizing the database queries and indexes can improve the overall performance of the system.
- Constraints have been imposed on database design and data access. Database tables will be designed in accordance with the principles of relational database design. DAO (Data Access Object) design pattern will be used for data access.
- Priority will be given to the ease of use and accessibility of the system. The user interface will have an intuitive structure and enable users to easily perform their desired actions.

Dos:

- ➔ Give importance to user experience and make the user interface simple and user-friendly.
- ➔ Be sensitive to the privacy and security of user data.
- ➔ Use encryption and authentication methods for system security.
- ➔ Make regular backups to prevent data loss.
- ➔ Avoid unnecessary database queries for system performance and usability.
- ➔ Allow students to fill out surveys for multiple courses.
- ➔ Create logs to track and report errors in the system.
- ➔ Use existing technologies and techniques while developing the system.

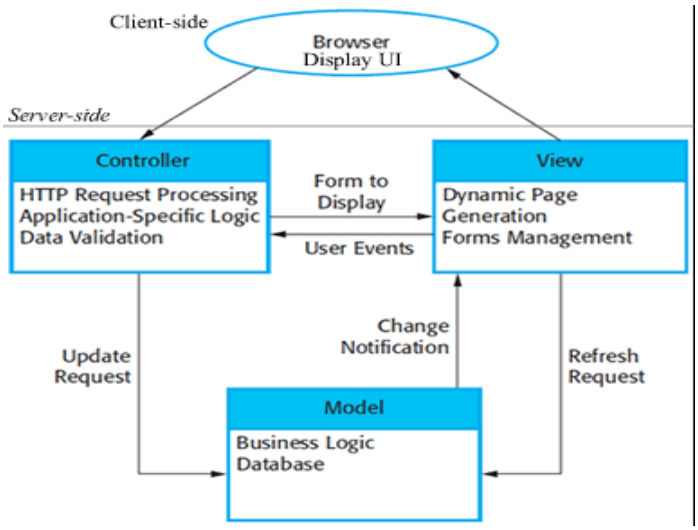
Don'ts:

- ➔ Do not sell or share user data with third parties.
- ➔ Do not access or use user data without permission.
- ➔ Avoid unnecessary processes that can cause system overload.
- ➔ Do not request unnecessary information or resources from users to complete surveys.
- ➔ Do not use any method to distort or manipulate survey results.
- ➔ Avoid giving users false or misleading information.
- ➔ Do not add unnecessary features or make the system too complex.

6. Architectural Mechanisms

Model View Controller

Model: This component handles the data storage and management of the system. It includes a database that stores course and instructor evaluations, student feedback, and other related information. The model component also includes the business logic layer that performs the necessary calculations and validations for the evaluations.
View: The view component is responsible for displaying the data and information from the model to the user. In ICE4HU, this includes the UI screens where students can submit evaluations and instructors can view the results. The views are created using React framework for front-end development.
Controller: The controller component manages the communication between the model and the view. In ICE4HU, the controller handles the user input from the view and sends it to the model to process. The controller also retrieves the data from the model and sends it to the view to display. The controller is developed using the ASP.NET framework for back-end development.



Layered Architecture

- *Presentation Layer:* The presentation layer will consist of UI screens developed using the React framework. It handles user requests such as submitting course and instructor evaluations and displaying the results.
- *Entity Layer:* The entity layer will consist of a controller component developed using ASP.NET. This layer processes user requests from the presentation layer and communicates with the business layer.
- *Business Layer:* This layer includes the business logic of the system that processes the requests received from the application layer. It performs the necessary calculations and validations for the evaluations and communicates with the data access layer.
- *Data Access Layer:* This layer handles the database interactions of the system. It stores and retrieves data from the database and communicates with the business layer.
- *Data Layer:* The data layer will consist of an MS SQL Server database. We will utilize MS SQL Server to store all data related to the application. To ensure high availability and scalability, we will deploy the database to a cloud service such as Azure SQL Database.

By separating the different components into distinct layers, the code can be organized and maintained more easily. Each layer has a specific responsibility, and changes made to one layer will not impact the other layers. This allows for greater flexibility and scalability of the system.

7. Key abstractions

Abstraction	Description
Survey	Represents a course survey. Contains survey questions, the instructor being evaluated, and the participating students.
Question	Represents a single question in the survey. Contains question text, response options, and any constraints or rules for answering.
Instructor	Represents the instructor being evaluated in the survey. Contains the instructor's name, department, and course being taught.
Student	Represents a student participating in the survey. Contains the student's name, email, and responses to the survey questions.
Evaluation	Represents the evaluation of an instructor or a course based on the survey responses. Contains the overall score, individual question scores, and any comments or feedback provided by students.
Report	Represents a report generated from the survey data. Contains summary statistics, charts, and tables to help visualize and interpret the survey results.
User	Represents a user of the system, such as an administrator, instructor, or student. Contains login credentials, access permissions, and user profile data.

By keeping cohesion at the maximum level, we aimed to strongly correlate the content of each module with each other. High cohesion means that the subcomponents really belong together.
We minimized the coupling between modules because the modules don't need to know much about each other. Low connectivity facilitates future exchange.

8. Layers or architectural framework

Layers

Presentation Layer:

The presentation layer of ICE4HU will be implemented using React. This layer will be responsible for presenting the user interface (UI) to the user and handling user input, such as submitting course and instructor evaluations and displaying the evaluation results. The React framework will be used to create interactive and responsive UIs.

Entity Layer:

The entity layer will be implemented using ASP.NET. It will serve as the middle layer between the presentation layer and the data layer. The application layer will handle user requests received from the presentation layer and communicate with the business layer. It will also be responsible for routing requests to the appropriate controller and providing the necessary data to the presentation layer.

Business Layer:

The business layer of ICE4HU will be responsible for implementing the business logic of the system. It will perform the necessary calculations and validations for the evaluations and communicate with the data access layer. The business layer will consist of multiple sub-layers, including a service layer and a validation layer. The service layer will handle the business logic, while the validation layer will ensure that the data being processed by the service layer is valid.

Data Access Layer:

The data access layer of ICE4HU will handle the database interactions of the system. It will store and retrieve data from the MS SQL Server database and communicate with the business layer. The data access layer will be further divided into two sub-layers: the data context layer and the repository layer. The data context layer will provide the necessary context for database operations, while the repository layer will handle the actual database operations. The repository layer will use Entity Framework to provide an object-relational mapping layer for interacting with the database.

Data Layer

The data layer will contain the MS SQL Server database. We will use MS SQL Server to store all the data related to the application. We will deploy the database to a cloud service such as Azure SQL Database to ensure high availability and scalability.

MVC

Model

We have selected MS SQL Server as the database of the project which is deployed in Azure cloud servers. We selected this database due to its high scalability and performance, as well as our familiarity with it. We have implemented the necessary database entities and relationships using Entity Framework, a popular object-relational mapping (ORM) tool for .NET applications.

View

We have used React for the front-end implementation due to its ease of use and versatility. We have implemented various UI components such as forms, tables, and charts to enable users to submit course and instructor evaluations, as well as view the results. We have used Axios, a popular JavaScript library, to establish a connection between the view and the controller layer.

Controller

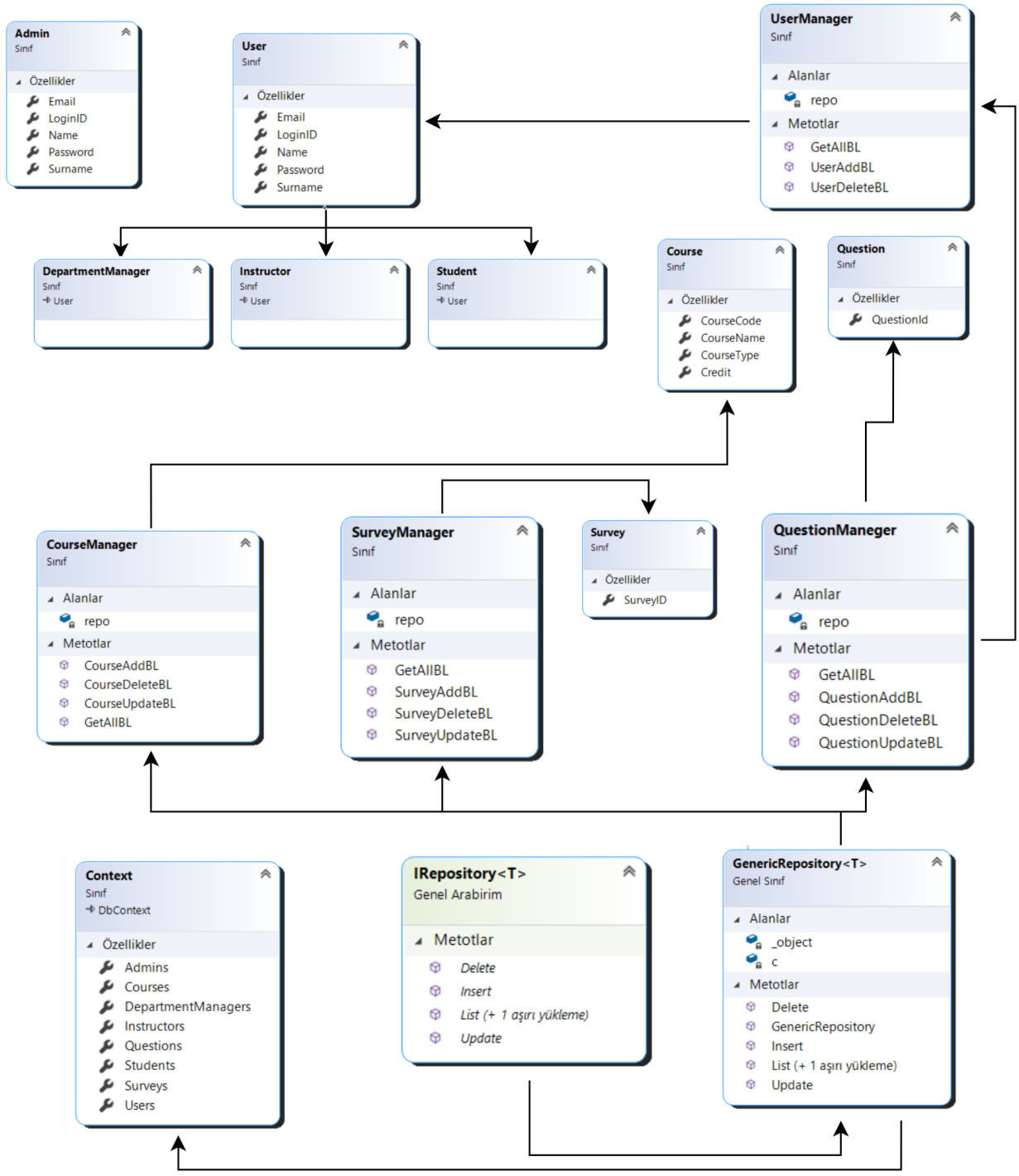
We have implemented a RESTful API infrastructure using ASP.NET Web API, which enables us to handle HTTP requests and responses in a consistent and standardized manner. We have created separate controllers for handling course and instructor evaluations, as well as user authentication and authorization. We have implemented various HTTP request methods such as GET, POST, PUT, and DELETE to handle CRUD (Create, Read, Update, Delete) operations. We have also implemented middleware for handling authentication and authorization using JSON Web Tokens (JWTs).
Overall, the MVC pattern helps us to separate the concerns of the project into distinct layers and components, which enables us to maintain and modify the project more easily over time.

9. Architectural views

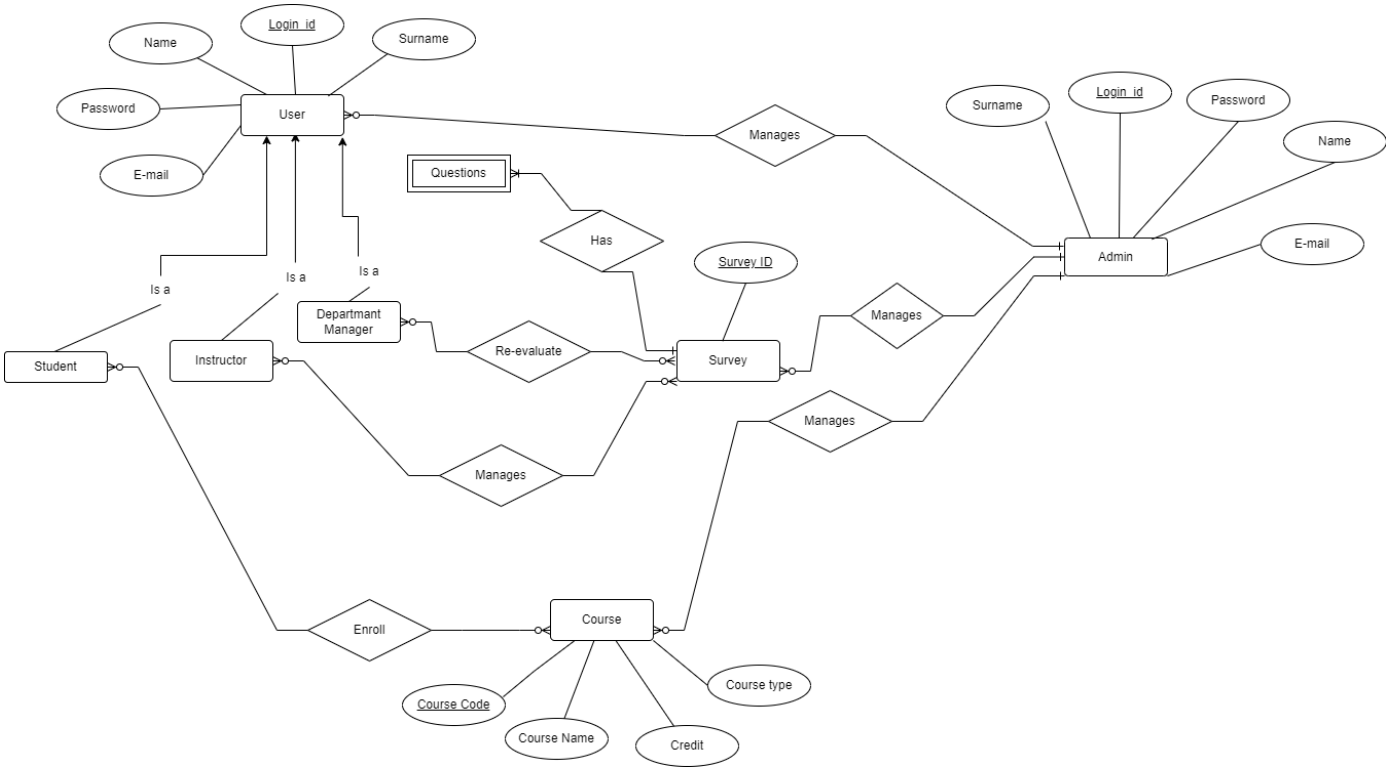
Recommended views

- **Logical:** Describes the structure and behavior of architecturally significant portions of the system. This might include the package structure, critical interfaces, important classes and subsystems, and the relationships between these elements. It also includes physical and logical views of persistent data, if persistence will be built into the system. This is a documented subset of the design.
- **Operational:** Describes the physical nodes of the system and the processes, threads, and components that run on those physical nodes. This view isn't necessary if the system runs in a single process and thread.
- **Use case:** A list or diagram of the use cases that contain architecturally significant requirements.

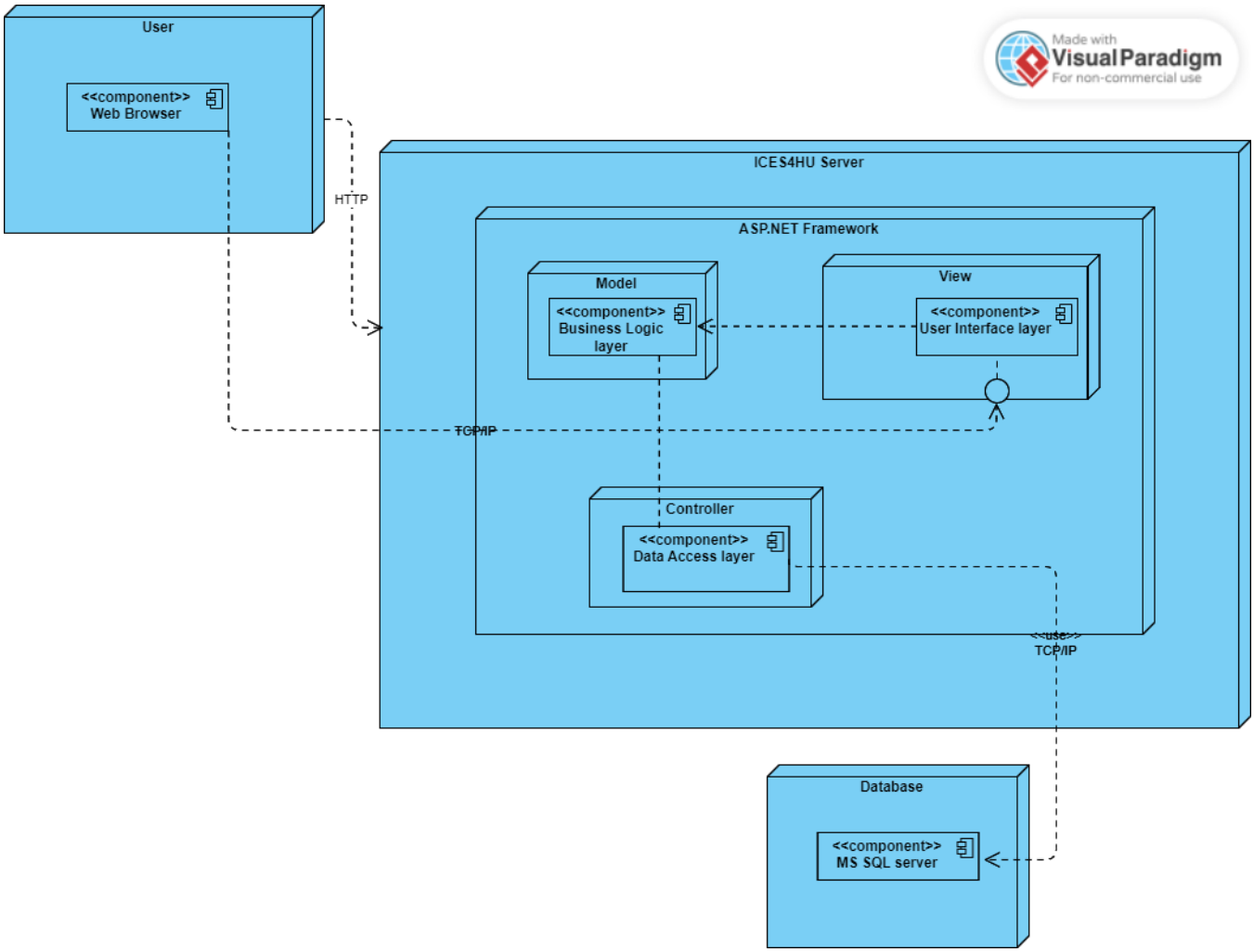
Class Diagram



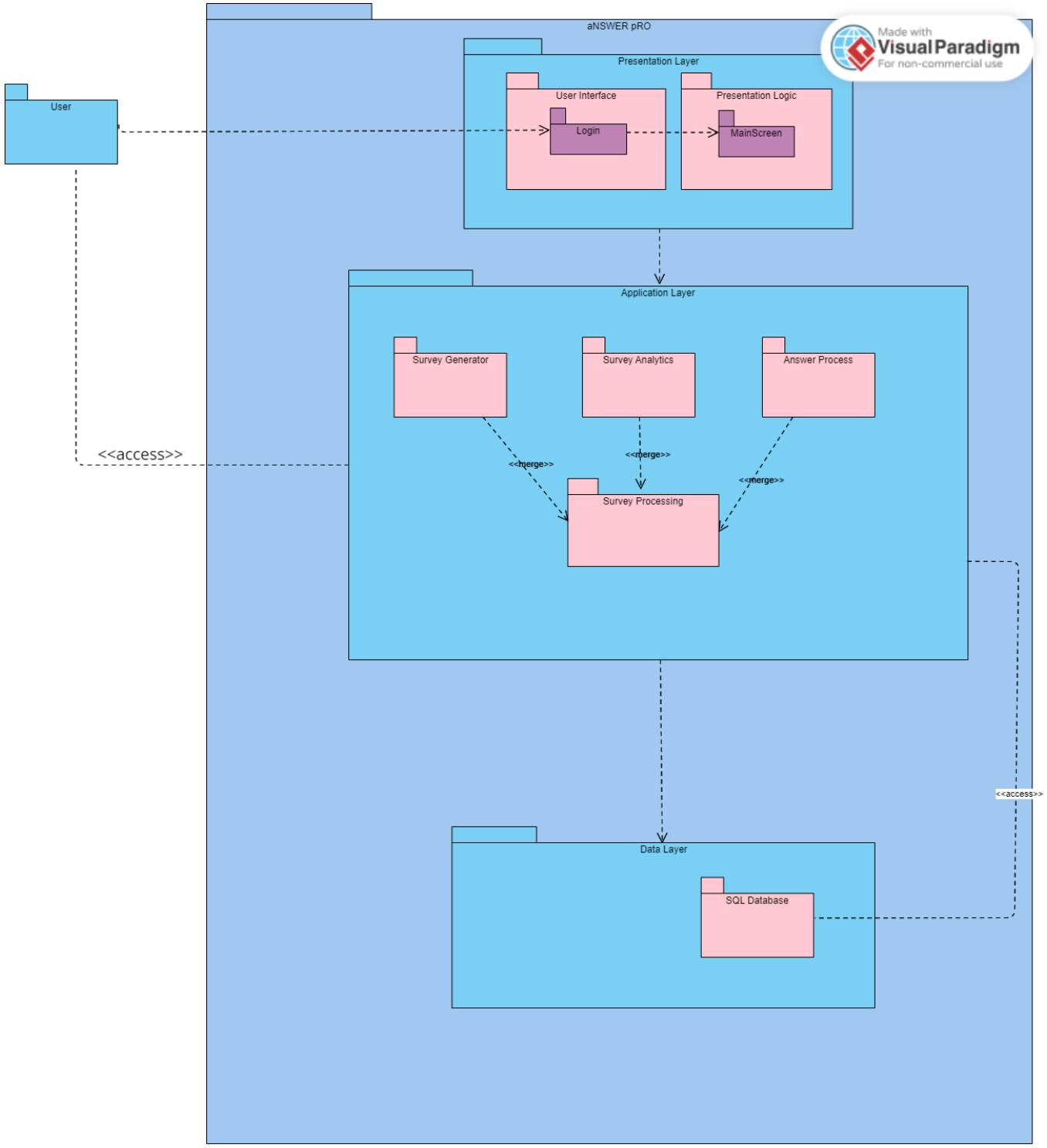
ER Diagram


Login_id
Name
Surname
Password
User
E-mail
Manages
Surname
Login_id
Password
Name
E-mail
Admin
Questions
Has
Survey ID
Is a
Is a
Is a
Department Manager
Re-evaluate
Survey
Manages
Student
Instructor
Manages
Manages
Enroll
Course
Course type
Course Code
Course Name
Credit

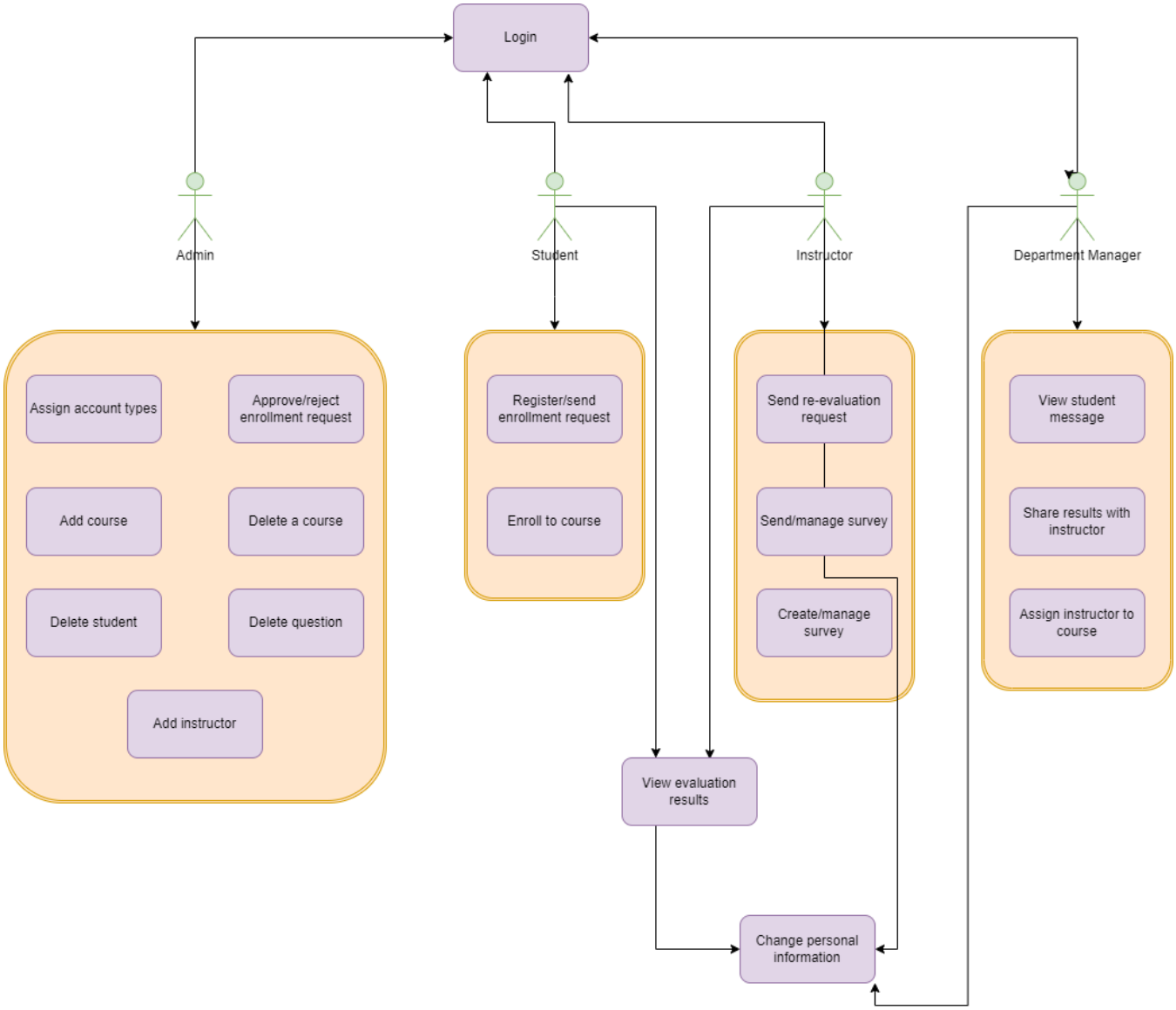
Deployment Diagram



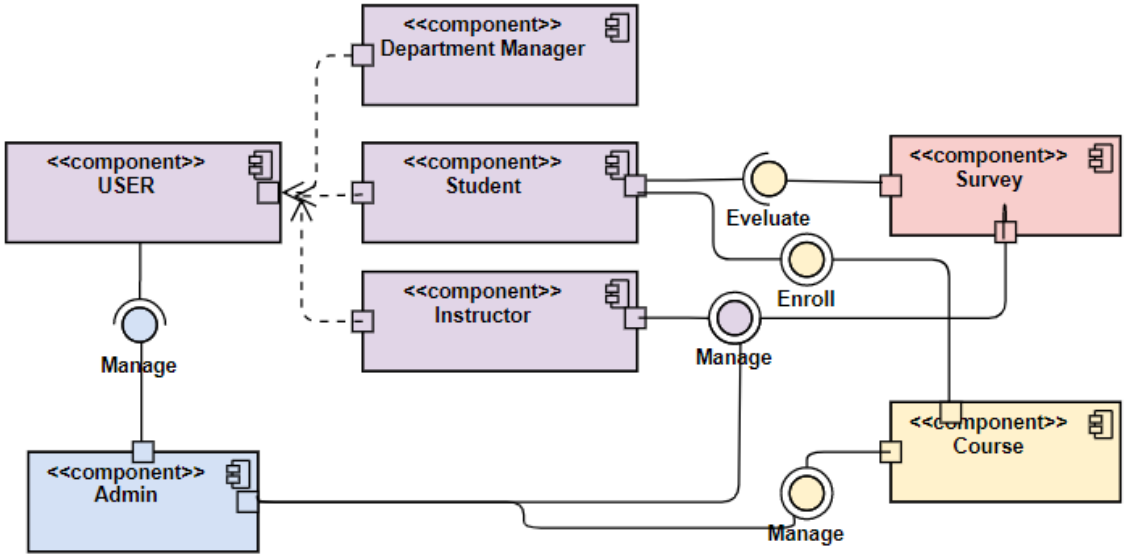
Package Diagram



Use Case Diagram



Component Diagram



10. Distribution of Tasks

- This notebook written by Şura Nur Ertürkmen and Ayşe İrem Yalçın.
- The newly created tables are shared between these two team members.
- The whole team has checked the report.