LinkedHU_CENG
Architecture Notebook

1. Purpose

The main purpose of this document is to describe the general and abstract design of the entire system. Architectural design introduces the behavior of the system without details. It does not include the implementation of the system components, instead includes the behavior of them. This abstract view provides an overview about the understanding of the general architecture of the system for the stakeholders. When they have a general overview of the system, it is easier to communicate with each other. Furthermore, designing an abstract overview of the system at the earlier stages is useful for deciding the functional and non-functional system requirements. So, preparing this document makes it easy to do software requirements analysis.

This document describes the philosophy, decisions, constraints, justifications, significant elements, and any other overarching aspects of the system that shape the design and implementation in an abstract way. The rest of the document will describe the architectural decisions that will be used in the implementation of the system, non-functional requirements such as performance, security and maintenance. Also, there will be assumptions and constraints for the implementation. All the content will be visualized and supported with the diagrams.

2. Architectural goals and philosophy

Determining the architectural goals is required to evaluate the success of the implementation at the end of the project by considering the desired functional and non-functional requirements. One of the most important goal of designing the system architecture is to make the team members to understand the reasons behind the architectural decisions. This will make it easy to adapt the system to the legacy systems or adapt the system to an architecture which has performance issues, also make it easy to maintainability. Since the goals are already specified, developers can easily reach a conclusion about the satisfiability. A set of architectural goals for the LinkedHU_CENG system as follows:

- This program is a web application. It should be flexible under the constraints of the universal web applications.
- The system must be maintainable even after a long period of time. Because, the system will be used in a living environment. This means that as new people join the system (such as newly admitted students, graduates or academicians), the expectations from the system may change. Or some glitches may be detected after some time and need to be fixed. Therefore, developers have to design the implementation to modify it easily in the future.
- The system needs to work error-proof under unusual conditions. For instance, when there are 10,000 concurrent users in the system at the same time, the system shall work without any errors and give responses within 5 seconds. Also, there should be no problem when short-term internet problems occur.
- The system should be easy to learn how to use.
- The system should be independent from the operating system and the browser.
- Users must be logged in to perform any operations on the system (except for the registration). The system should provide proper authentication for performing this purpose.
- The architecture should separate the presentation and interactions of the system data as much as possible.
- User interface of the system should be English to provide universal maintainability.

3. Assumptions and dependencies

Assumption is a thing that is accepted as true or as certain to happen, without any proof. Whereas, in a project network, a dependency is a link among a project's terminal elements.

- Each team member should have some prior knowledge about databases, front-end and back-end technologies. Having some experience with them and this kind of project provides a big plus for the entire life-cycle of the project.
- Each team member should be available for this project at least 12 hours per week.
- There are no budget limitations.
- Passwords shall be stored in encrypted form.
- The private messages that are sent between users shall be stored in encrypted form.
- The system requires an internet connection.
- When we determine the performance or time requirements, we assume that the transactions between database and system are efficient and fast.
- The system should be reachable at least 99% of the 24 hours a day and 7 days a week.

4. Architecturally significant requirements

Architecturally Significant Requirements are a subset of the functional and non-functional requirements. Therefore, requirements that have various effects, point shot trade-offs and architecturally significant assumptions are being selected.

- The response time of the system to the requests should be as minimum and acceptable as possible. As we stated in the Software Requirements Specification (SRS) document, the system shall provide a maximum response time of 0.5 seconds for search, provide a maximum response time of 0.3 seconds for real-time communication, and provide a maximum response time of 0.4 seconds to populate the main page feed functionalities. Note that, these response times do not include the user's internet response time.[1]
- The system shall not store all of the passwords that belong to the user as what it is. Instead, all passwords should be encrypted using the SHA-256[2] algorithm. Because, SHA-256 algorithm is hard to break but easy to encrypt. Therefore, we have chosen the SHA-256 algorithm for this purpose.
- The system shall encrypt the private messages as well that are sent between users with a standard encryption technique and shall store the encrypted messages. We have chosen the SHA-256 algorithm again for this purpose. Again, because it is hard to break but easy to encrypt or decrypt.
- The back-end of our system will be handled with ASP.NET[3]. The first and the most important reason why we have chosen ASP.NET is that ASP.NET follows the MVC[4] architecture conveniently[5]. Another reason is that it reduces the coding time, so we can make enough time for documentation and implementation optimizations or maintainability. Also, it allows Cross-platform migration.
- As stated at SRS, the system is expected to provide at least 99% uptime for availability[6]. Furthermore, if the system is down, it should be recoverable as quickly as possible.
- The system should provide scalability[7]. Scalability is the measure of a system's ability to increase or decrease in performance and cost in response to changes in application and system processing demands.
- The system shall not activate or delete the user account without the administrator's approval.

[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://dotnet.microsoft.com/en-us/apps/aspnet/mvc
[6] https://en.wikipedia.org/wiki/Availability_(system)
[7] https://en.wikipedia.org/wiki/Scalability

5. Decisions, constraints, and justifications

Decisions and constraints of this project have been selected carefully. This section includes justifications and restrictions for our decisions and constraints.

We have chosen the response time for the search functionality as 0.5s, for the real-time communication functionality as 0.3s, and for the populating the main page feed as 0.4s. These numbers are coming from Doherty Threshold and other general acceptances. According to the Doherty Threshold, an immediate interface between the user and the computer will enhance the experience and make it easier, and the user experience will shift from painful to addicting if the system feedback time decreases below 400ms. As a result, the reaction time should not be more than 400 milliseconds. However, almost all of the members of our team will deal with searching problem for the first time; therefore, we decided to increase the response time of the search functionality to be able to reach our goals. On the other hand, we decided other response times with respect to general acceptances.

Storing user passwords is a critical component for any web or mobile application which has a registration system. An application is reliable and secure if and only if the system that users use stores a user's password in a secure way. The system must ensure that it is secured in such a way that if your data is compromised, a system shouldn't expose its user's password. We all have heard in our daily lives about the companies that compromise their users' personal information and details. Even if the system does not store the users' passwords as encrypted, it is worse. If you are developing an application that is either web or mobile and the system needs to store users personal information, it is important to prevent compromises. Hashing the passwords is one of the common approaches to storing passwords securely. Converting hashed values back to its original form is impossible, but you don't need to convert them back to break. Once you know that a certain string converts to a certain hash, you know that any instance of that hash represents that string. Hashing a password is good because it is quick and it is easy to store. Instead of storing the user's password as plain text, which is open for anyone to read, it is stored as a hash which is impossible for a human to understand. Here, we choose the SHA-256 algorithm to do hashing. SHA-256 algorithm generates an unique, fixed-size 256-bit (32-byte) hash. Furthermore, you also need to salt the passwords because a hacker can eliminate the advantages of just hashing the passwords by some techniques. Salting is where you add an extra bit of data to the password before you hash it, something like you would append every password with a string before hashing. This would mean the string before hashing would be longer and therefore harder to find a match.

Also, keeping users' private messages private is one of the key privacy problems. We have to again use a hash algorithm to encrypt the messages. Since encryption is easy and breaking is hard, we again use SHA-256 algorithm to do hashing. The same reasons and explanations are valid for encrypting the messages.

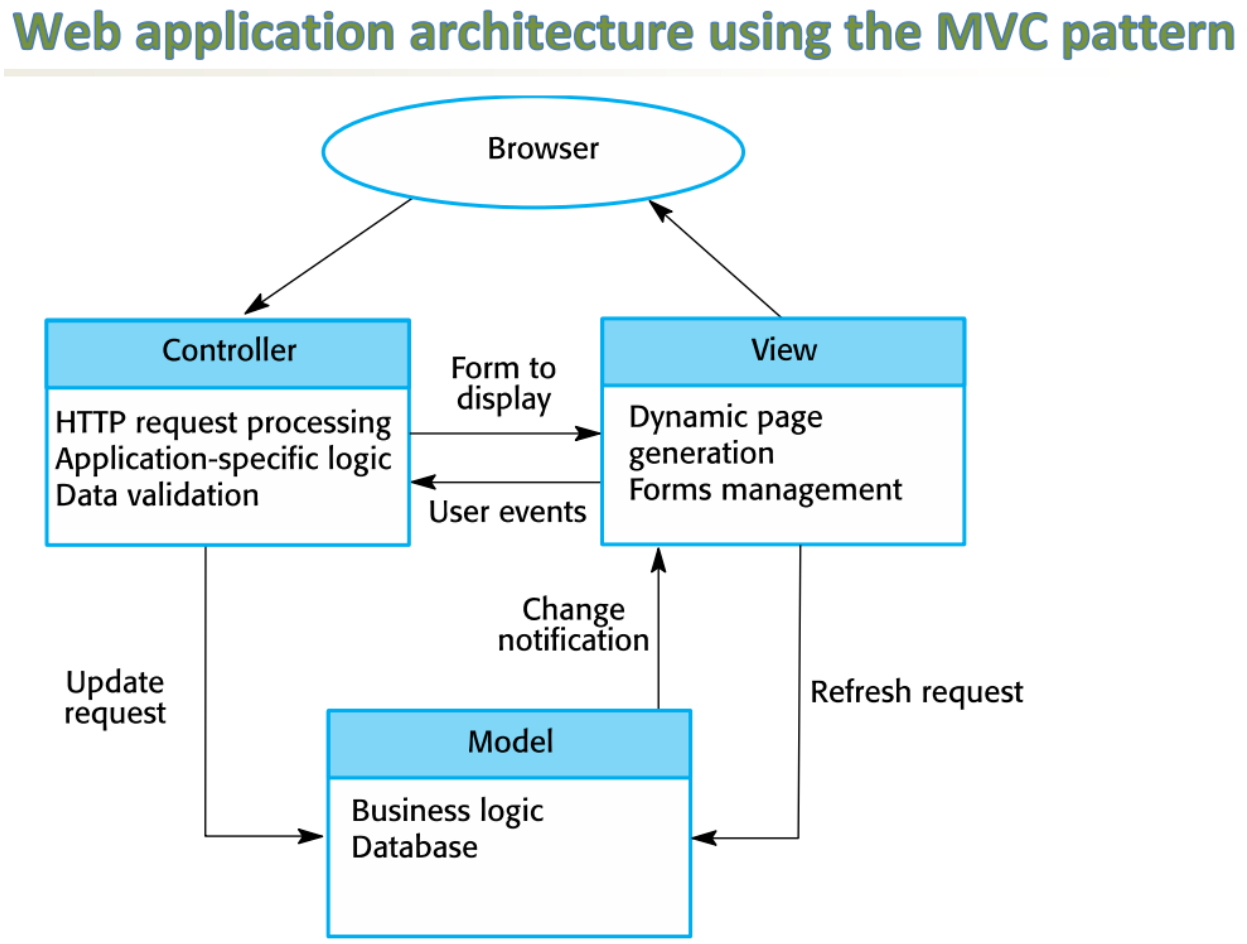
We want our system as available to the users and stakeholders as possible. It is a metric that measures the probability that a system is not failing or requiring a repair action when it needs to be used. For high availability, we choose 99% uptime as the most optimal for our case. We cannot choose any higher, because of higher cost.

In our back-end, ASP.NET architecture is used to eliminate redundancy over implementations. It allows us to build an application with minimal or zero configurations. So, the architecture provides quick implementations to us. ASP.NET follows MVC architecture in which each layer communicates with the layer directly below or above (hierarchical structure) it.

6. Architectural Mechanisms

MVC Pattern

The entire LinkedHU_CENG project is built on MVC Pattern. As you know, MVC separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. Model component manages data and associated operations on that data. View component defines and manages how data is presented to the user. Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to View and Model. We have separate the classes of the controllers for cases such as user, post, announcement, etc. in the same package. Controller passes the related information to the View or Model components, gets the results from these, and responds to the user. In that way, we have an architecture that is not cluttered.



Layered Architecture

Layered Architecture organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. In the Presentation Layer, there are Views and Controllers. All of the other layers as Business Layer and Persistence Layer are in the Model. In the Presentation Layer, the requests made by a user (it may be a login request, post request, create announcement request, comment request, etc.) are handled and converted for other layers (Model) if needed. In the Business Layer, the business logic is handled and all of the authorization and validation is performed. In the Persistence Layer, all of the storage logic is stored and handled. Also, CRUD operations are performed there. In that way, the code is going to be easier to maintain, compact, and easily movable.

7. Key abstractions

We abstract the components of the layered pattern. Presentation layer, business layer, persistence layer, and database layer are main parts of the layered pattern. We tried to satisfy high cohesion and low coupling with modularity of the system. The entire system has divided classes into parts instead of working with singular class structure. In the presentation layer, the controller classes handle the requests and forward them to the next layer. Only simple controls are done in this layer. We aimed to make an abstraction between this layer and business layer through the controller classes. In the business layer, we have service classes which don't have any access to databases without the next layer.

8. Layers or architectural framework

Presentation Layer

We have controller classes to handle requests and forward them to the next layer, business layer. This layer gets requests, checks them according to request validation, and forwards them to related service classes.

Business Layer

We have service classes to handle necessary operations for the requests coming from the presentation layer and forward them to the persistence layer. Also, the service classes communicate with repository classes in the persistence layer.

Persistence Layer

We have controller classes to make data operations without direct connection between the project and database. Also, repository classes communicate with the business layer and database. We implemented an interface to use essential database CRUD functions.

MVC Architecture

Model

We have selected PostgreSQL as the database of the project which is deployed in AWS servers. We selected this database, because some of our team members have experience with PostgreSQL. Therefore, this selection makes the implementation duration sorter because of our previous knowledge. Also, we saw the harmony between PostgreSQL and ASP.NET when we searched the internet.

View

We have selected JavaScript for the front-end implementation because of its benefits such as modularity, efficiency, etc.

Controller

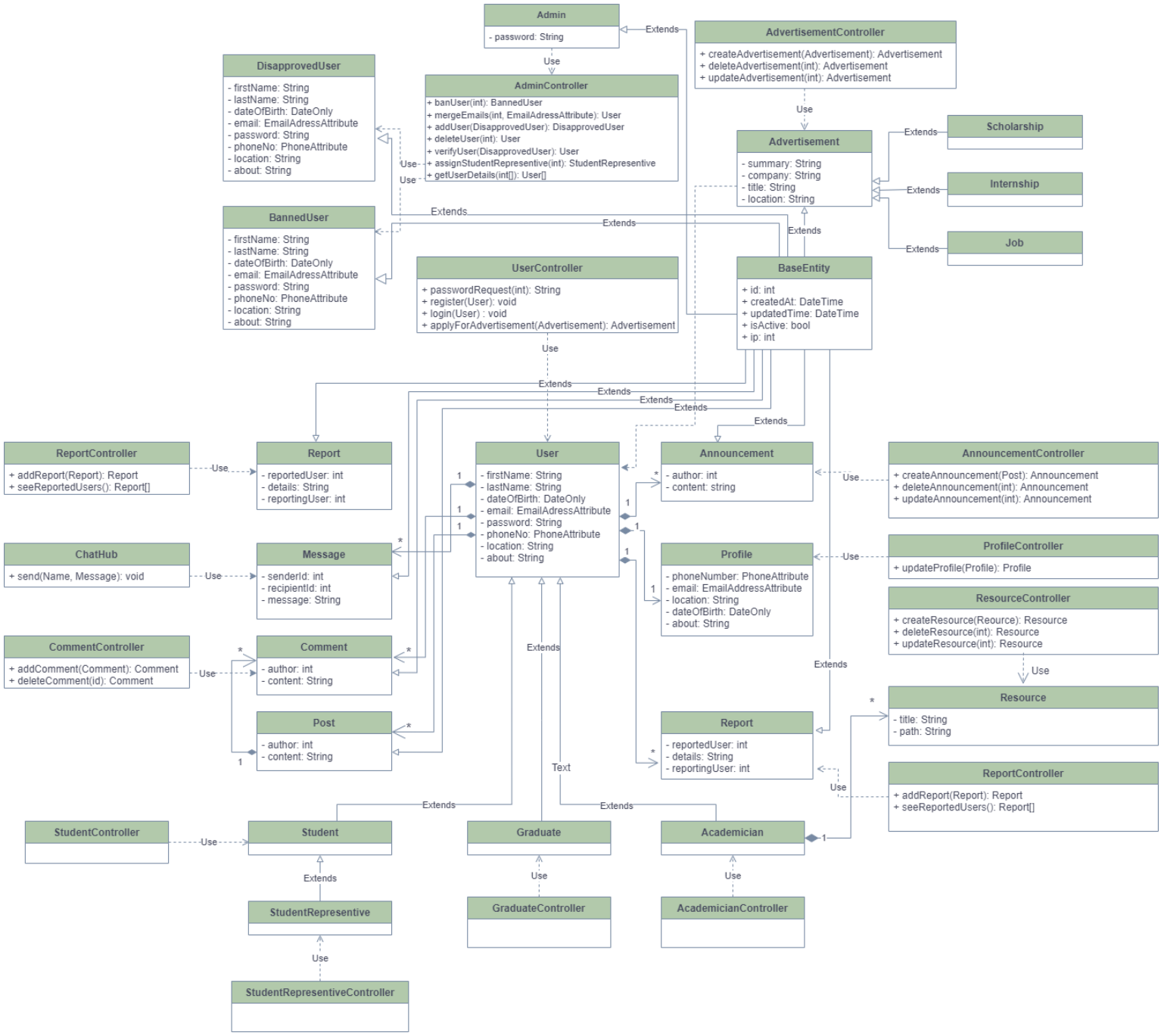
We used ASP.NET Route("api/[controller]") annotation to build an API infrastructure. Also, we have separate security configurations for each controller.

9. Architectural views

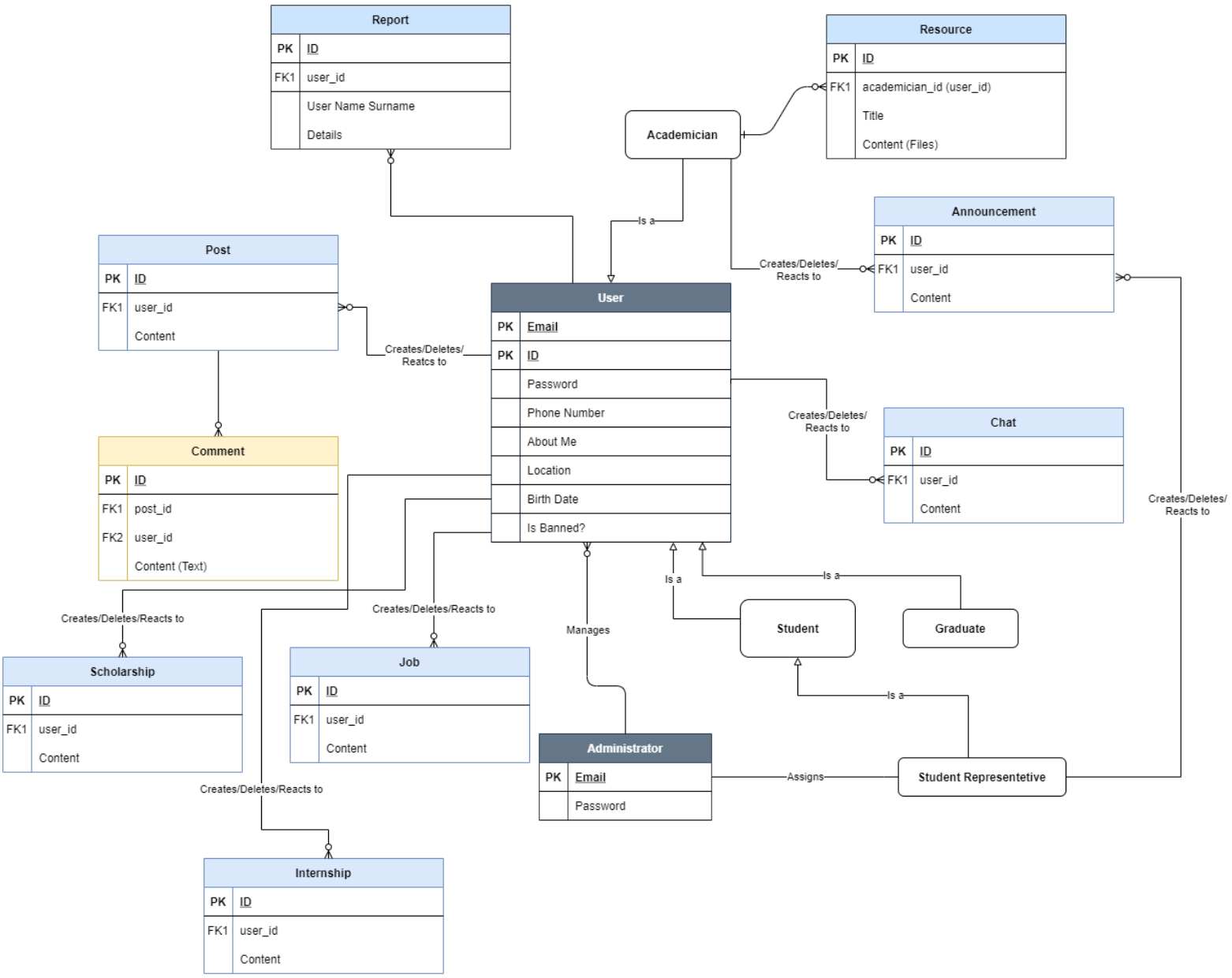
Recommended views

- **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.

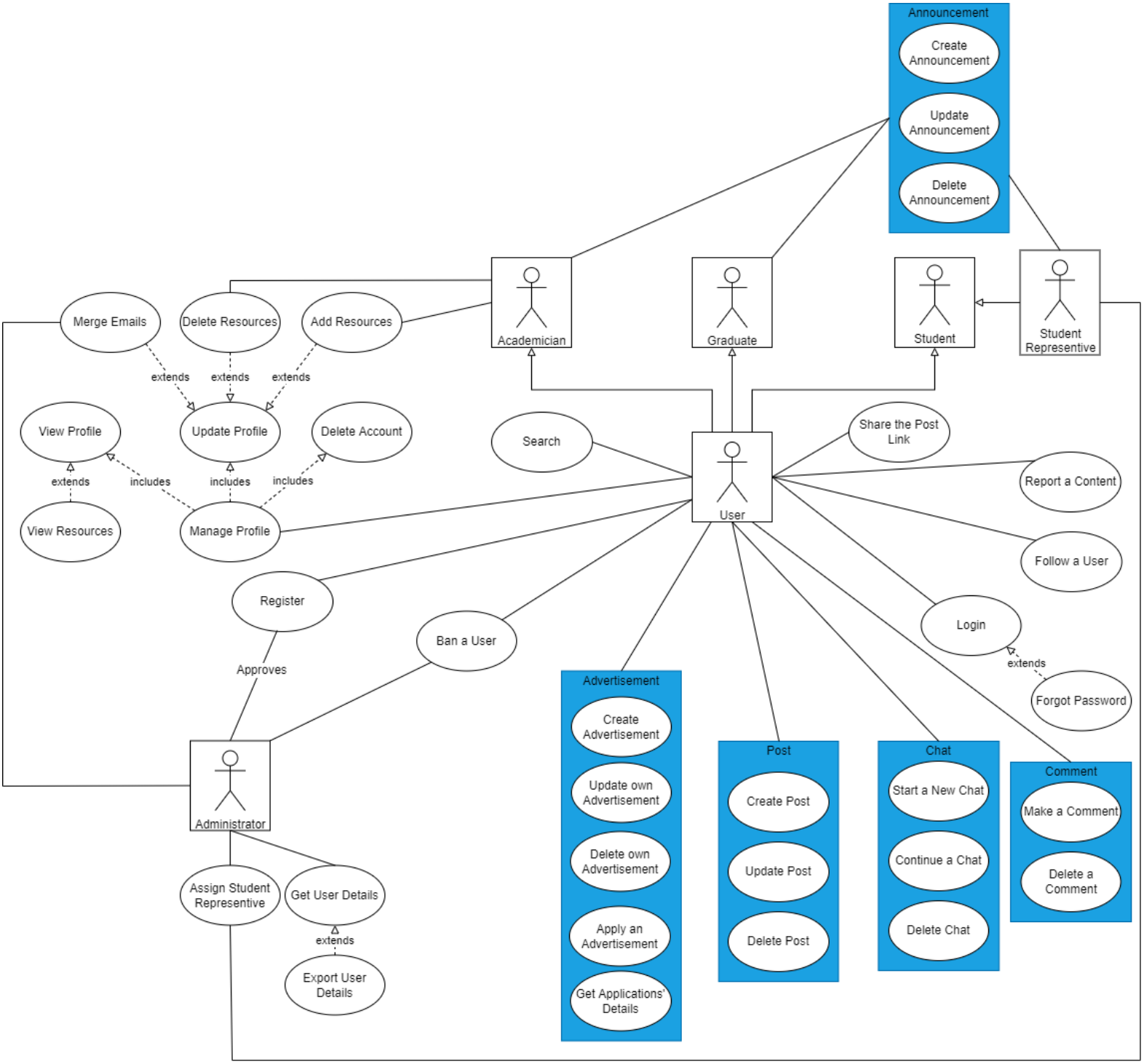
Class Diagram



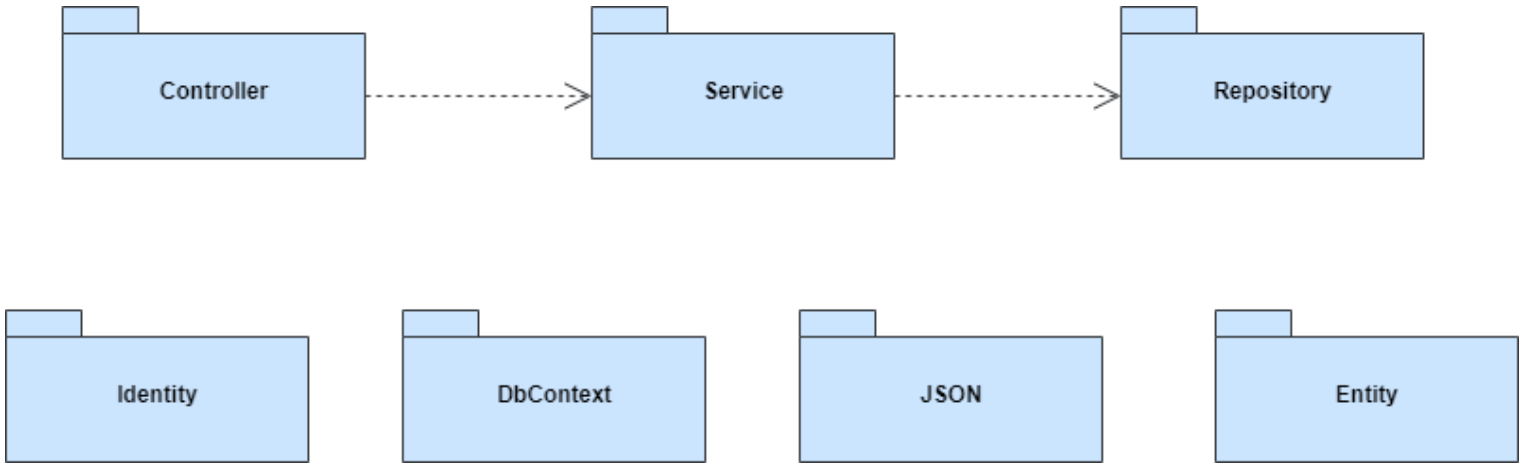
Updated ER Diagram



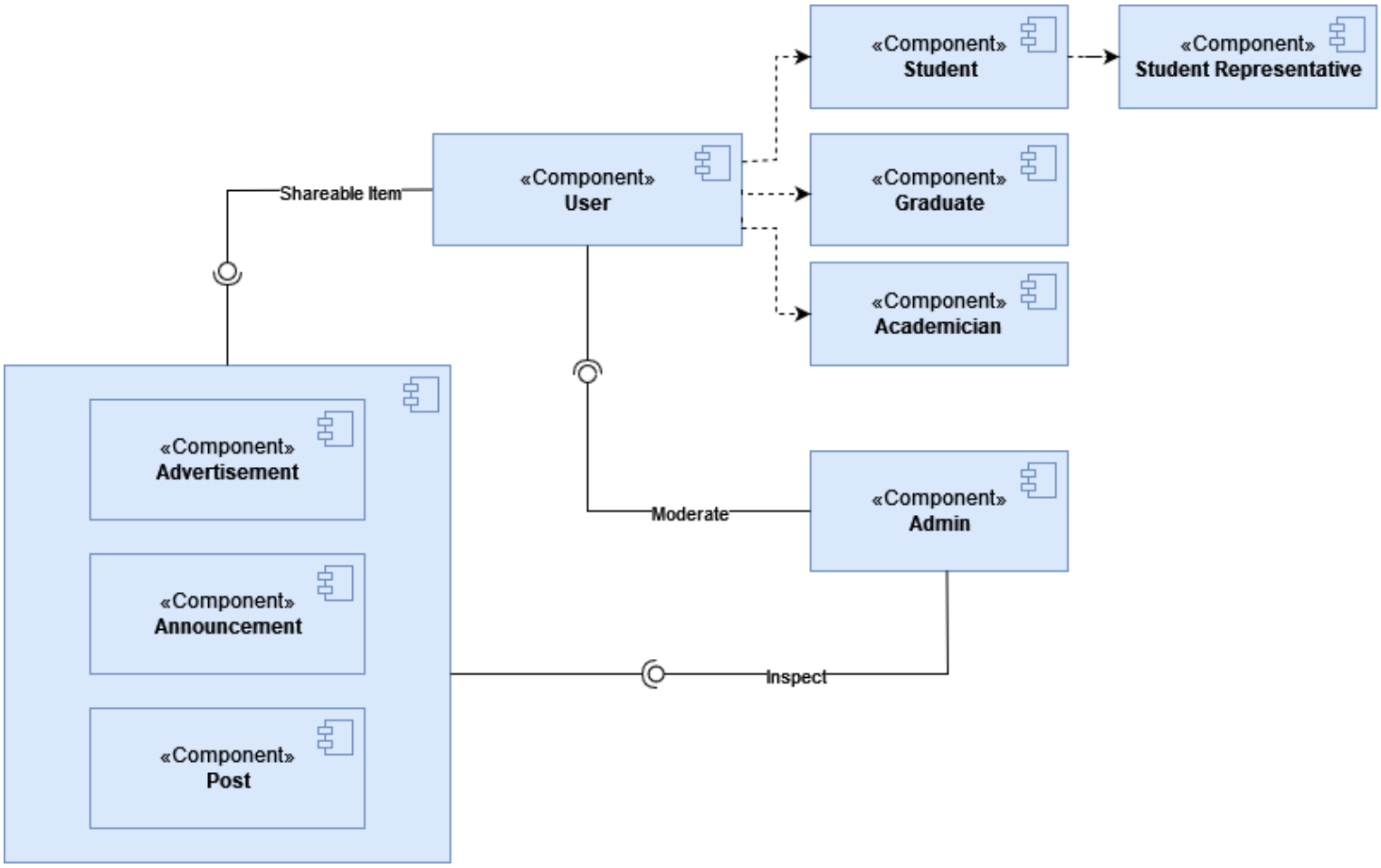
Updated Use-case Diagram



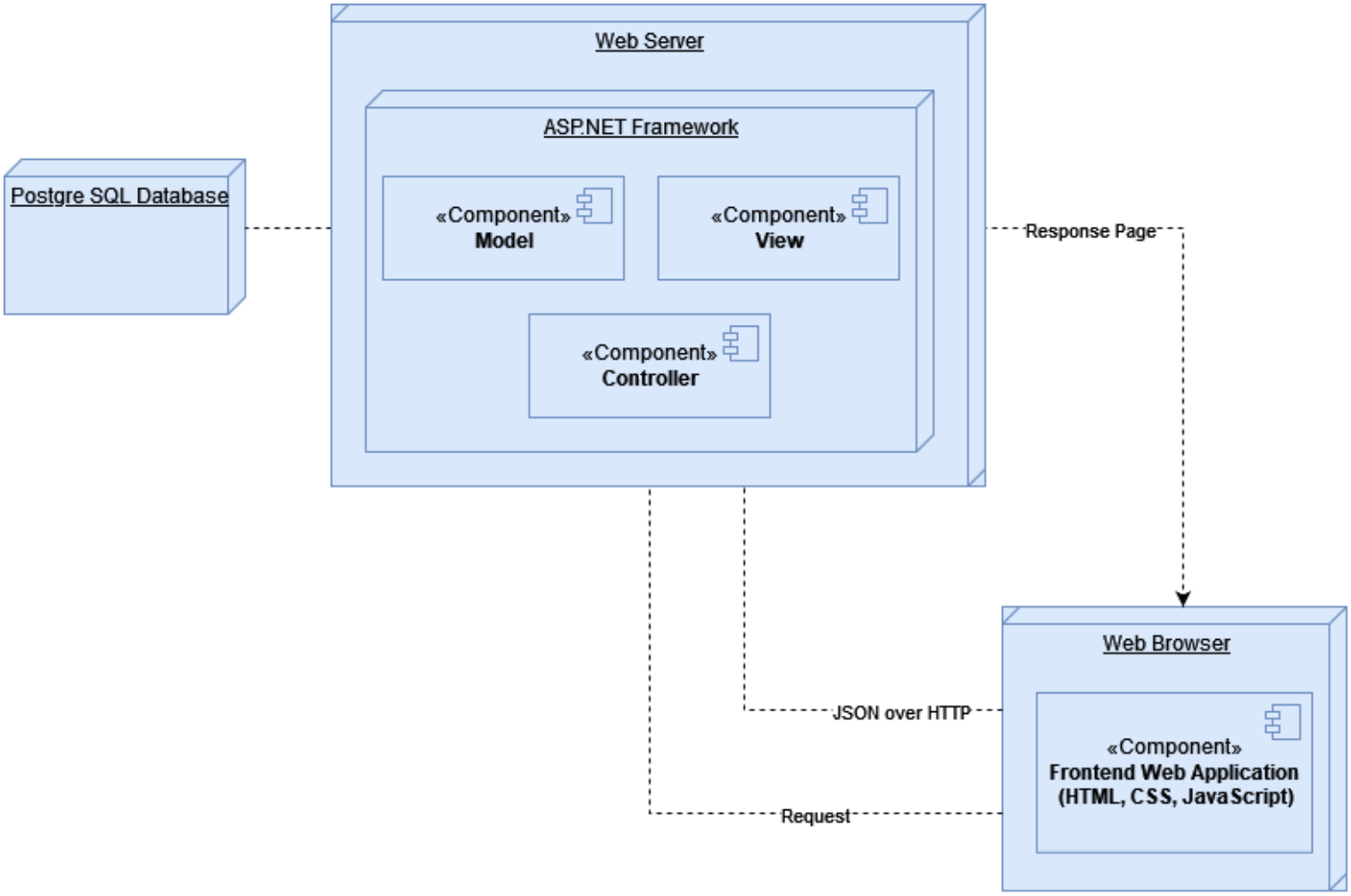
Package Diagram



Component Diagram



Deployment Diagram



Distribution of Tasks

This document was prepared primarily by Software Architecture (Mert Doğramacı) with contributions from the entire team.