LinkedHU_CENG
Architecture Notebook

1. Purpose

Software Architecture is the high-level description of design and implementation. Elements are assembled to compose the system and satisfy requirements. This document lays out our philosophy, assumptions, decisions, specification, and design regarding the software architecture of LinkedHU_CENG.

2. Architectural goals and Philosophy

Development Process. Our design process is Scenario-driven. Since use-case scenarios describe the most significant functionalities of the system, architectural design is driven from these scenarios. Then, architecture is implemented, tested, evaluated. Our Iterative development provides more chance to prototype, test, re-evaluate, evolve, refine the architectural design, re-assess and mitigate risks.

Our prototypes will be minimum-viable-products, successfully executing its just enough features. New features will be extended on top of a working system. Without perfectionism, if time permits, additional quality will be polished on top.

In general, we will follow the design philosophy we have been taught, throughout our Hacettepe CS courses, especially Software Engineering and its lab.

Usability & Supportability & Maintainability. We aim to deliver a smooth experience to our users, using reliable and generic technologies. Non-processor-intensiveness and cross-browser compatibility of LinkedHU_CENG will enable it to reach all our target users. Our design will also have maintainability and extensibility in mind, for further evolution and integration with other academic systems.

Performance. LinkedHU_CENG is based on small textual data transactions, with a minimal UI. Thus it will be easy on the processors on the user-side, probably less resource intensive than 90s games. And on the server side, even though the design aims for later scalability, in the first few demo releases, user count will not exceed 100. Thus, performance/availability will not be major concerns for us in our development process.

3. Assumptions and dependencies

Below factors drive our architectural decisions:

- Besides having no official monetary budget, Time is our limited resource. Furthermore, Average Java Enjoyers members are developing such Web application for the first time, learning some technologies from scratch. Thus, sometimes our design decisions might not be based on the best industry-standard practices, but whatever just works and makes sense to us.
- Correctness is important for our system, as there will be important announcements shared on it. The commands should be delivered to Database successfully and the expectancies must be met with minimum error possible.
- The tools/languages/technologies LinkedHU_CENG uses are hardware independent. Thus, hardware dependency is not a concern since it is already abstracted from both users and the development team.

4. Architecturally significant requirements

- LinkedHU_CENG should be a dynamic website, available on popular desktop browsers, used without installation on the user-side.
- MVC Architectural Pattern is implemented since this was a requirement, but also for its advantages. More details on our implementation of this pattern is explained in next chapters.
- LinkedHU_CENG uses MySQL Database on the backend for information storage and retrieval. This database resides in the same server as LinkedHU_CENG backend.
- On the Backend, LinkedHU_CENG is a SpringBoot application, created using Java language. Firstly tested on localhost, then deployed and hosted on Firebase server.
- On the Frontend, HTML creates the website skeleton, CSS and Bootstrap defines visual theme and quality, JavaScript and jQuery defines the processes triggered by a user's interaction with the website.
- Frontend and Backend have to communicate via HTTP protocol. Since LinkedHU_CENG is a critical academic network, HTTPS can be implemented for better security and privacy.
- System must not let a non-logged-in user change any data on the system.

5. Decisions and constraints

Backend Framework

As Java Enjoyers, we enjoy Java very much. We were certain we wanted to use Java as our Backend language. There was only one question left to be answered: which backend framework? Our search has led us to SpringBoot, which is very well integrated with Java and MVC pattern.

Communication / Integration

LinkedHU_CENG is a Dynamic Webpage. Contents of a page are rendered into HTML files at the server-side. CSS and JS files are sent and run at the user-side for further functionality. A new HTTP request from Frontend to Backend results in a new HTML page sent from Backend to Frontend. For handling HTML templating and parsing HTTP requests, we use Thymeleaf. Thymeleaf is a server-side template engine for SpringBoot.

Therefore, LinkedHU_CENG is NOT a Single-Page-Application where a Frontend Framework is used to communicate with Backend via an API using JSON packages and so on. That would require us to implement an extra layer into the system, which we do not need.

We came to this decision since it will be a more efficient development process, considering the overall lack of Web development experience the team has. Also, we do not have a use-case where we asynchronously need to process data and update some content on the page without reloading and so on.

Our approach also comes with easier testing and debugging. With our approach, we can directly see if for example our CRUD operations work correctly, since we reload some section of data again from the Database, when Frontend requests the updated version of same page again.

6. Architectural Mechanisms

Single Responsibility & Open-Close Principles

A single element in our system should have a single responsibility. This way, required later modifications are minimized, and reuse is possible. Elements should be designed as encapsulated, closed for modification, and open for extension. This allows maintainability, extensibility, scalability.

High Cohesion & Low Coupling Principle

Cohesion is bringing logically related elements together in a module, which are contributing to a single task. High cohesion means the system is designed and structured well, leading to high performance. Coupling is the complexity of the communication/dependency links on a network of modules. High coupling leads to a chaotic spider web, a developer's nightmare. Whereas, low coupling implies that modules are realizing their tasks independently and only interacting when needed, leading to low dependency, high maintainability.

Model-View-Controller Pattern

An architectural pattern is a tried-and-tested general method of composing a well-designed software system on the high-level. MVC is a pattern commonly used in desktop GUIs and Web applications to separate User Interface, Control, and Data systems, and abstract them from each other.

This modularity allows:

- having multiple ways to view and interact with the same data.
- security of the Data layer, since it cannot be directly accessed from the Interface.
- any of the layers to be changed/extended in future, without affecting others.
- parallelization and independence of development processes for different layers.

As Average Java Enjoyers, implementing this pattern, we would like to benefit from all these advantages.

7. Key abstractions

Our MVC architectural pattern having multiple layers, Controller being the intermediary between View and Model,

- provides abstraction of layers from each other, encapsulation. For example: View sends data and requests to Controller, yet does not worry how they will be processed.
- provides low coupling. Dependencies are kept minimal. For example: View does not directly interact with Model. View interacts with Controller, Controller interacts with Model.
- Therefore; brings flexibility, maintainability, and better extendibility.
- Also, developers of different layers can work in parallel, only having to know the interfaces between layers.

8. Layers or MVC architectural framework

View

View is the abstract name of Frontend, user-side, UI. View sends HTTP requests and data to Controller, based on user's interaction with the Website. Then, receives HTTP responses, HTML files and other data from Controller, to process and display on the user-side.

Controller

Controller layer controls the system and resides in the Backend. It processes HTTP requests and data received from View, calls the needed services to take action. Services interact with Models and perform functions.

Then, Controller updates the View by sending HTTP responses, HTML (new page to display) files and other data.

Model

Model layer consists of model classes that represent data, such as Student and Post. These classes are manipulated in the Controller layer according to business logic. Model layer interacts with the Database and performs CRUD operations.

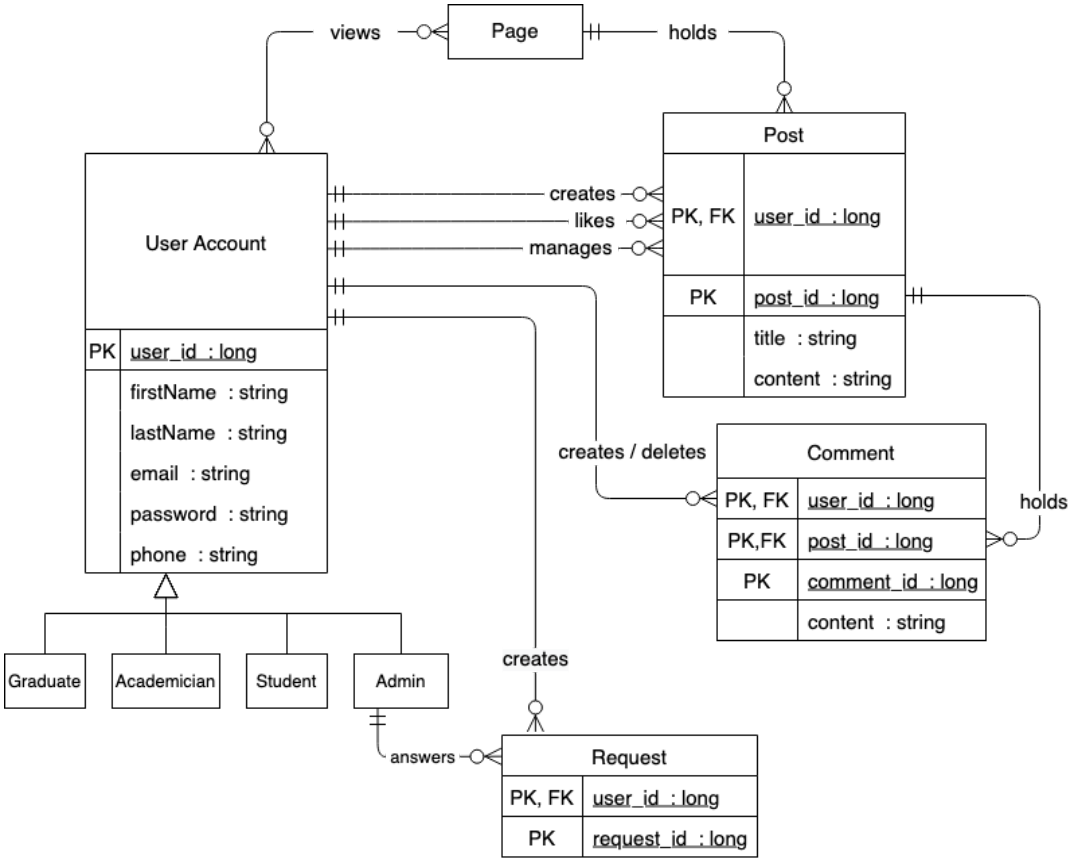
9. Architectural views

We would like to use the 4+1 View Model to describe our software architecture, as proposed by Philippe Kruchten in 1995. To address different concerns of various stakeholders, 5 different views are used, depicting the system from different perspectives. Stakeholders only have to care about what they want to know, from their own perspective.

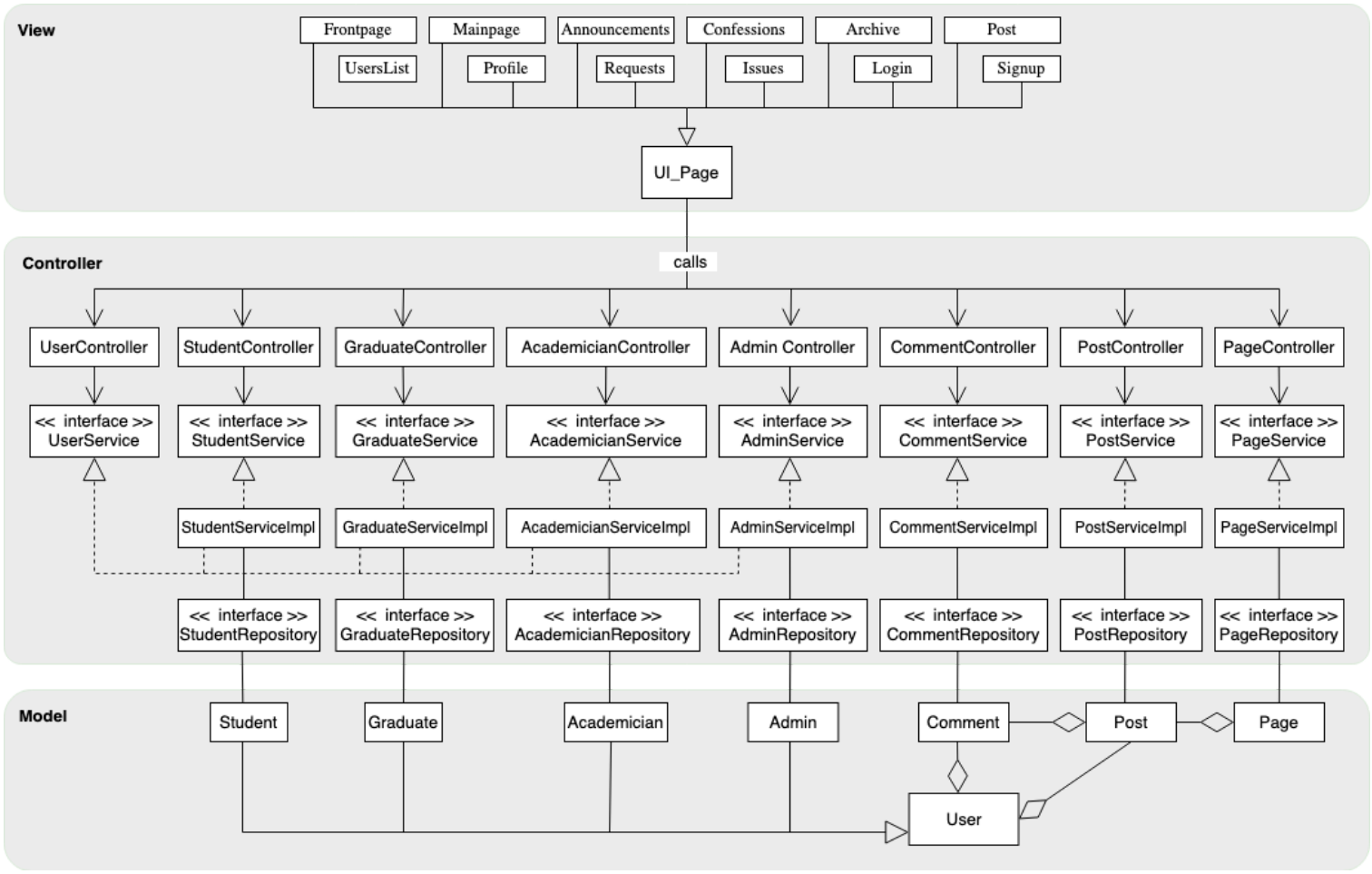
- **Logical View**

  Describes the structure of Object Oriented design, as classes and objects, and their logical relationships within OOP paradigm principles such as inheritance, association and so on. Since class methods are depicted too, system functionality (functional requirements) can be identified from this view.

A. Entity-Relationship Diagram for LinkedHU_CENG



B. Class Diagram for LinkedHU_CENG



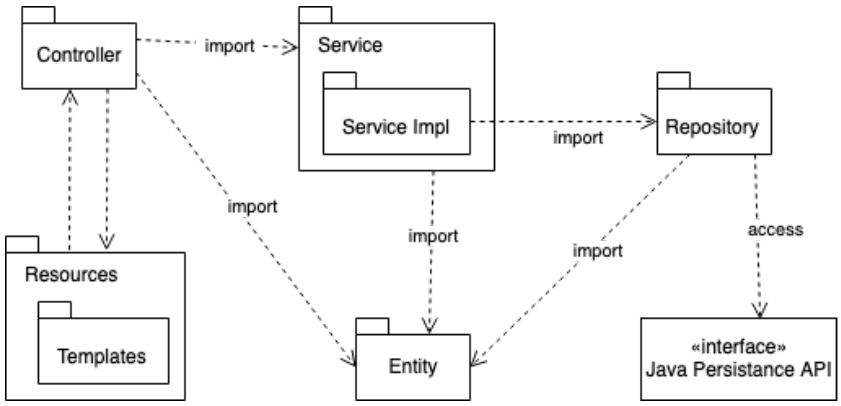
- **Process View**

  Describes run-time behavior and synchronization of concurrent processes. Sequences of data exchange, function invocations, triggers, interactions between software elements are depicted. In this view, system engineers can care for non-functional requirements such as performance, concurrency, and fault-tolerance. For example, we can depict run-time interactions between our M V C components on a sequence diagram.

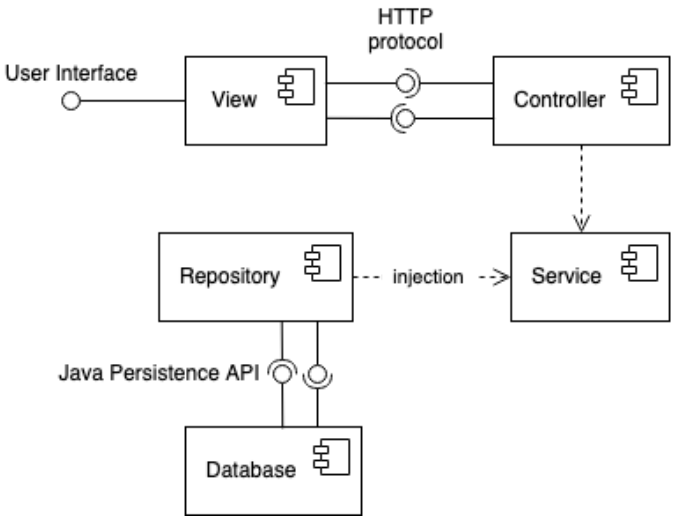
- **Development View**

  Describes how our system is statically composed of components/packages/subsystems that can be developed separately by developers. This view will mostly help for project management in terms of allocation of team members and resources, planning, progress evaluation.

C. Package Diagram for LinkedHU_CENG



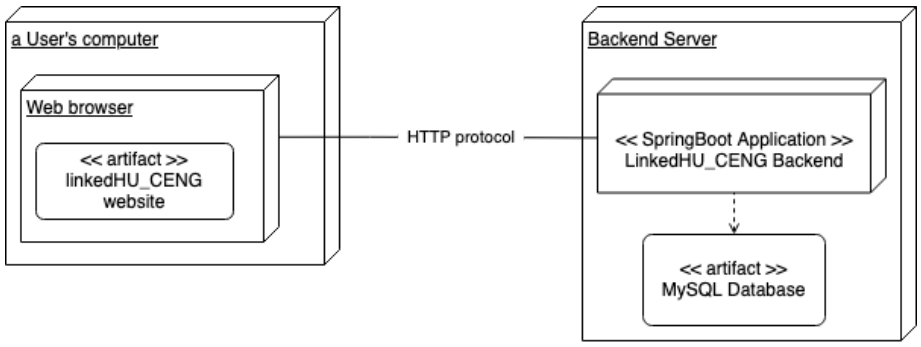
D. Component Diagram for LinkedHU_CENG



- **Physical View**

  Describes how our system is distributed on a network of collaborating processing nodes (hardware units). This view describes how the components of our system are mapped to and running on these nodes. The physical configuration will have impacts on non-functional requirements such as availability, performance and scalability.

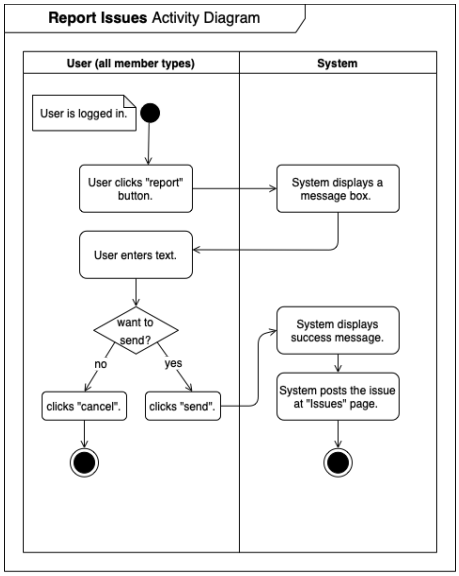
E. Deployment Diagram for LinkedHU_CENG



- **Scenarios View**

  Provides a high-level overview of the system, especially for our end-users, since it captures their interaction with the system. Use-case scenarios imply how all other four views are integrated into working together, to provide functionalities of the system. This view is significant since it is the starting point for system design and test-case preparation.

F. Example Activity Diagram for Report Issues (u-15) Use-Case



G. Use-case Diagram for LinkedHU_CENG