Busy Bee
Analysis and Design Document
Student: Carla-Maria Rusu
Group: 30431

Revision History

Date	Version	Description	Author
18/03/2020	1.0	Document inception.	Carla-Maria Rusu
04/04/2020	1.1	Architecture modification.	Carla-Maria Rusu
09/05/2020	1.2	Added Data Model & Design Model	Carla-Maria Rusu
02/06/2020	2.0	Revamped	Carla-Maria Rusu

Table of Contents

I. Project Specification

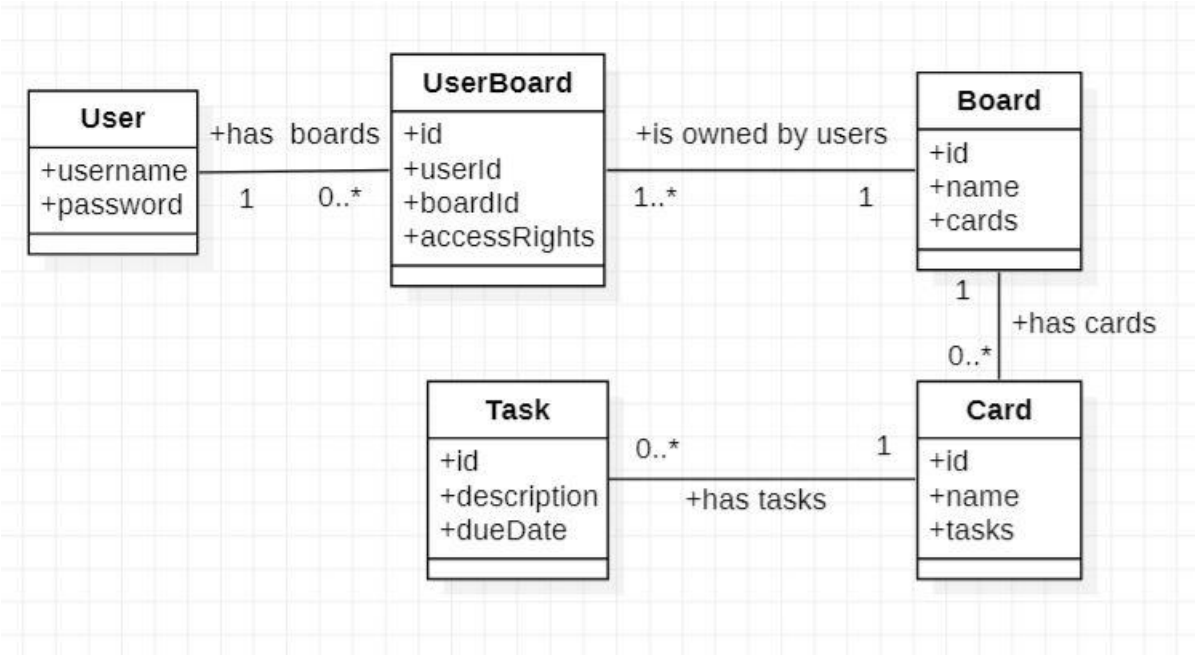
Busy Bee is an event planner and organizer which can be used by an individual or a group of people. It is meant to aid in planning events by giving users the ability to create new boards, add cards, add tasks and due dates, add contributors and set the contributors' access rights.

The application interface is user-friendly and provides drag and drop capabilities. The gives the user the ability to easily move tasks from one card to another. The event creator has the ability to set the access rights of members: view only and edit.

Busy Bee is a web application; thus it requires an Internet connection.

II. Elaboration – Iteration 1.1

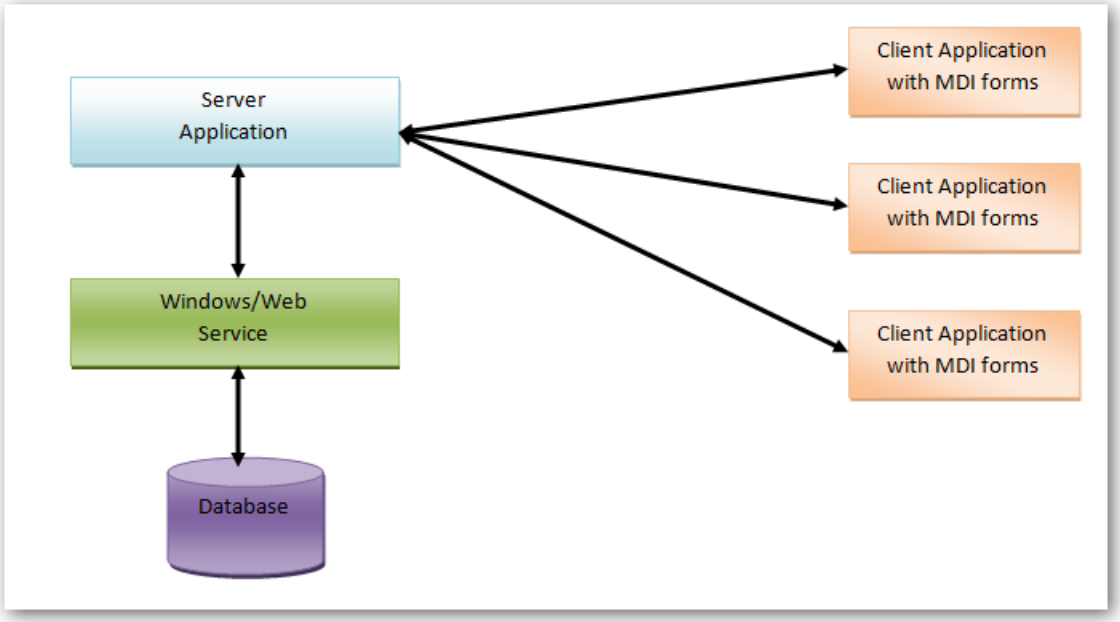
1. Domain Model



2. Architectural Design

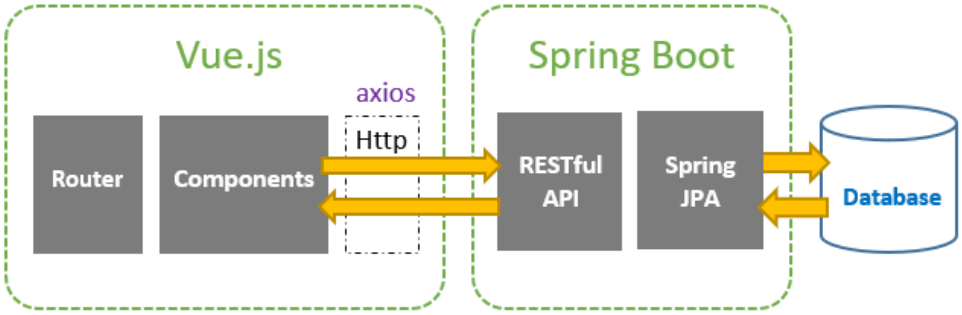
2.1 Conceptual Architecture

The conceptual architecture pattern chosen for this project is the **Client Server Architecture** as it is one of the most suitable architectures for web applications. This type of architecture has one or more client computers connected to a central server over a network or internet connection. This system shares computing resources.

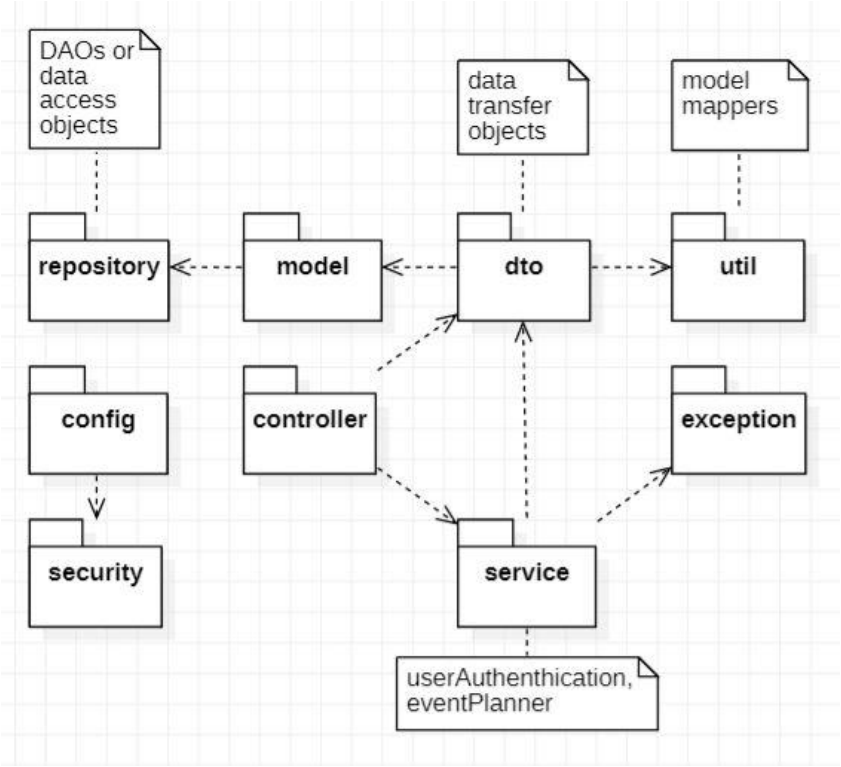


2-1 Client Server Architecture

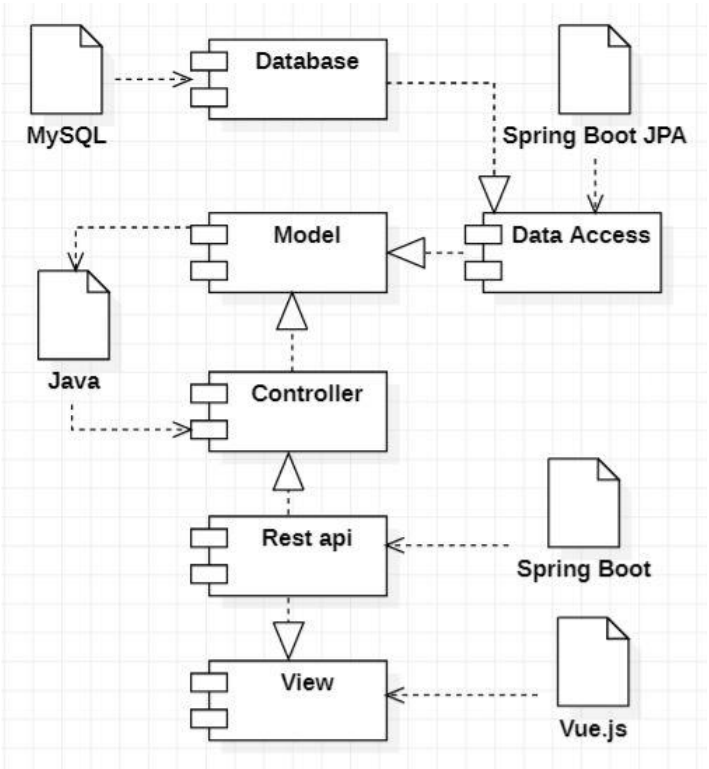
Domain-driven design (DDD) is the selected architectural style; it is an approach to software development for complex needs by connecting the implementation to an evolving model. The word domain refers to the subject area on which the application is intended to be apply. The reason why this style was chosen is that it enables the creation of complex designs based on the models of the domain. Another feature of DDD is the Building Blocks, namely several defined high-level concepts used to create and modify domain models (such as entity, service, repositories, factories, etc).



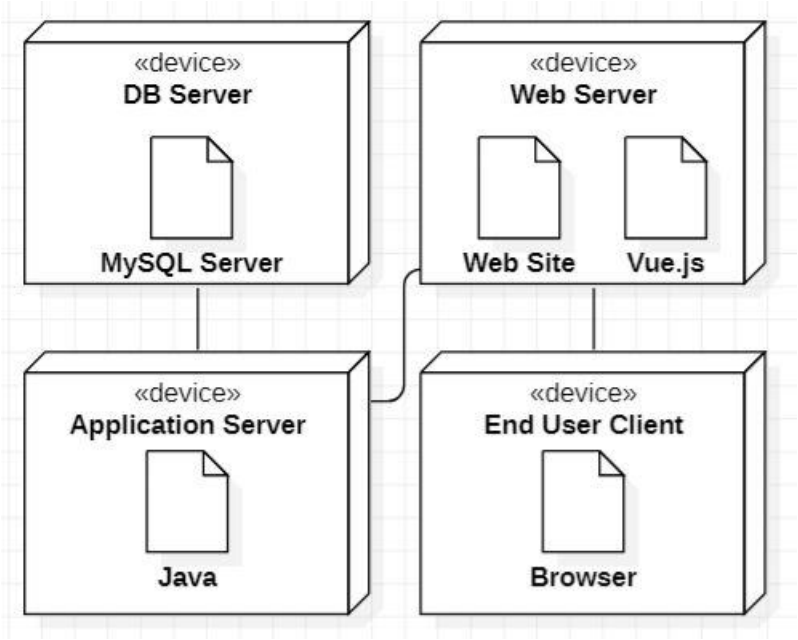
2.2 Package Design



2.3 Component and Deployment Diagrams



2-2 Component Diagram



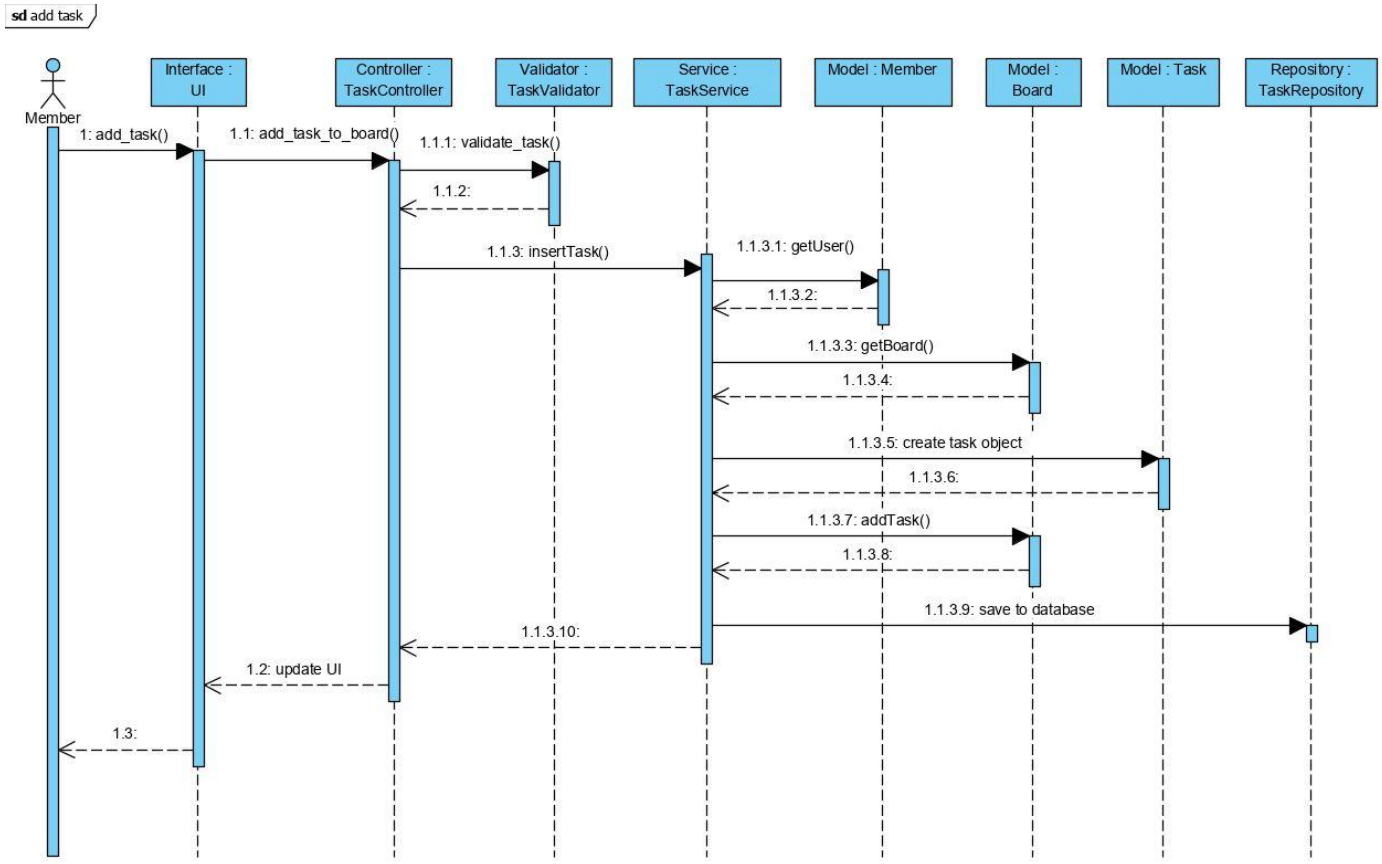
2-3 Deployment Diagram

III. Elaboration – Iteration 1.2

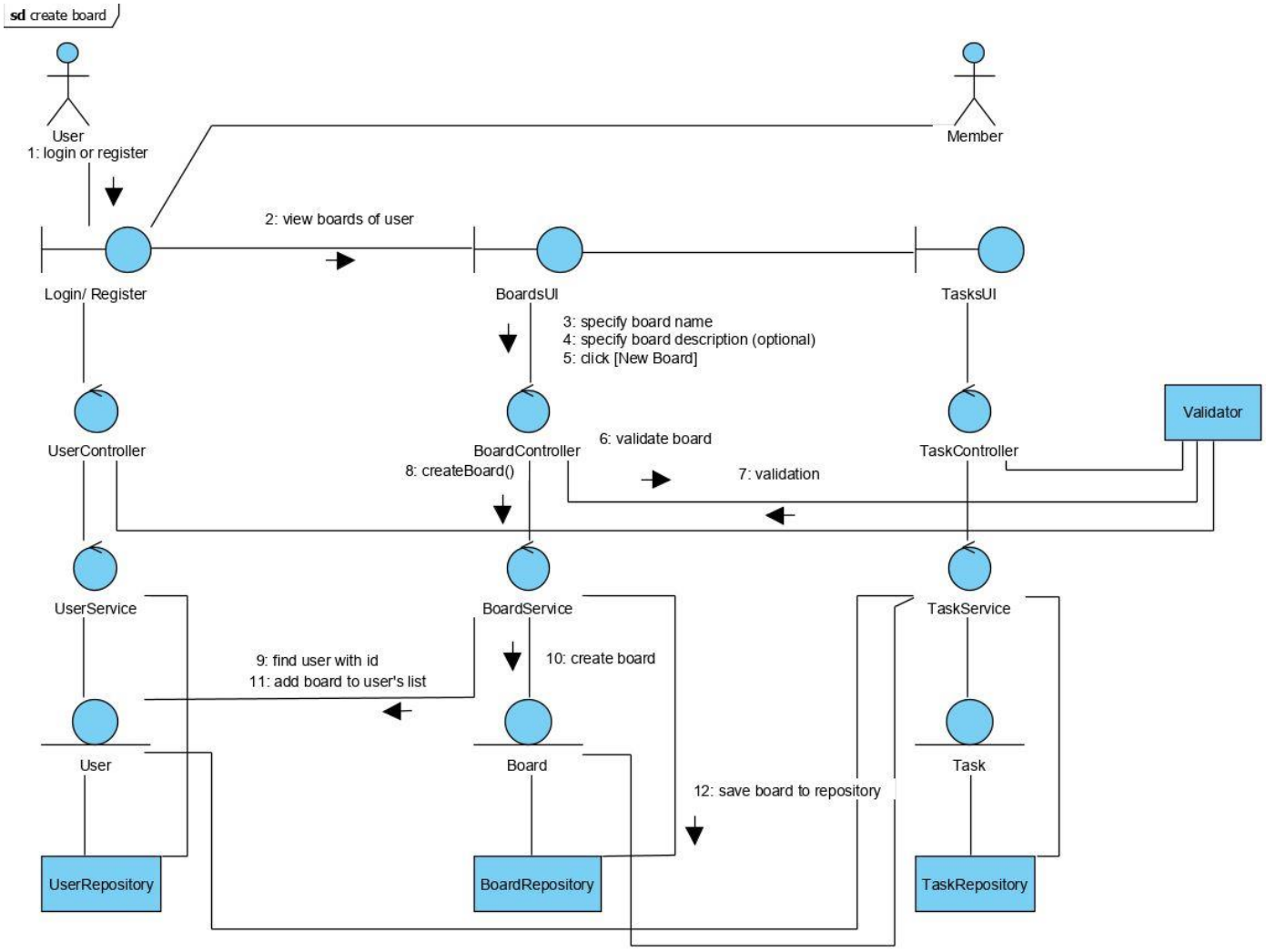
1. Design Model

1.1 Dynamic Behavior

1.1.1 Sequence Diagram – add new task to an existing board

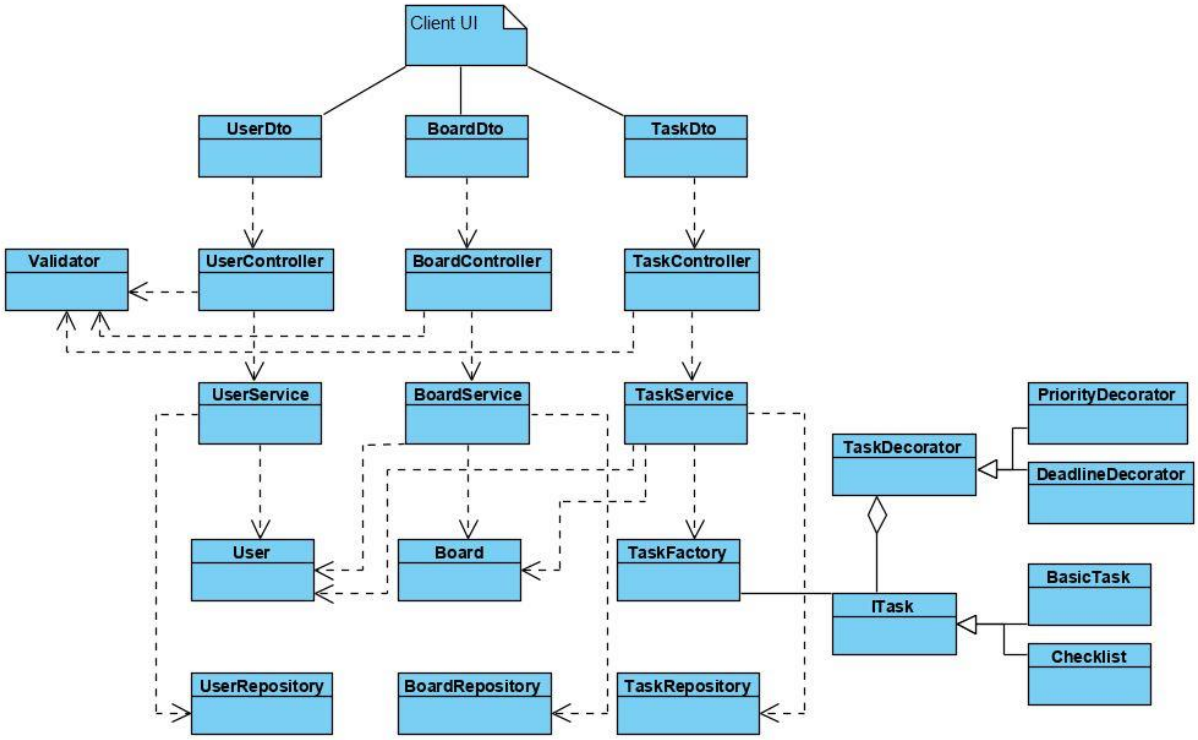


1.1.2 Communication Diagram – create new board

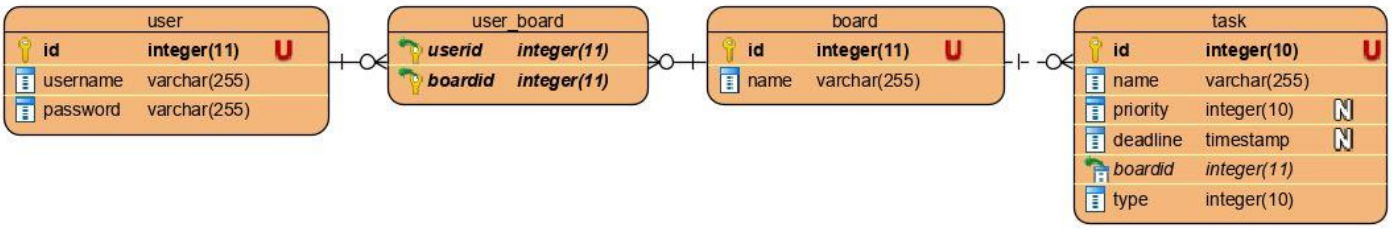


1.2 Class Design

- Might add Factory Pattern according to needs to create tasks of different types
- Might add Decorator Pattern to personalize tasks with priority level or deadline
- Might add Mediator Pattern at Controller/Service level
- Might add Observer Pattern to notify board members of changes



2. Data Model



3. Unit Testing

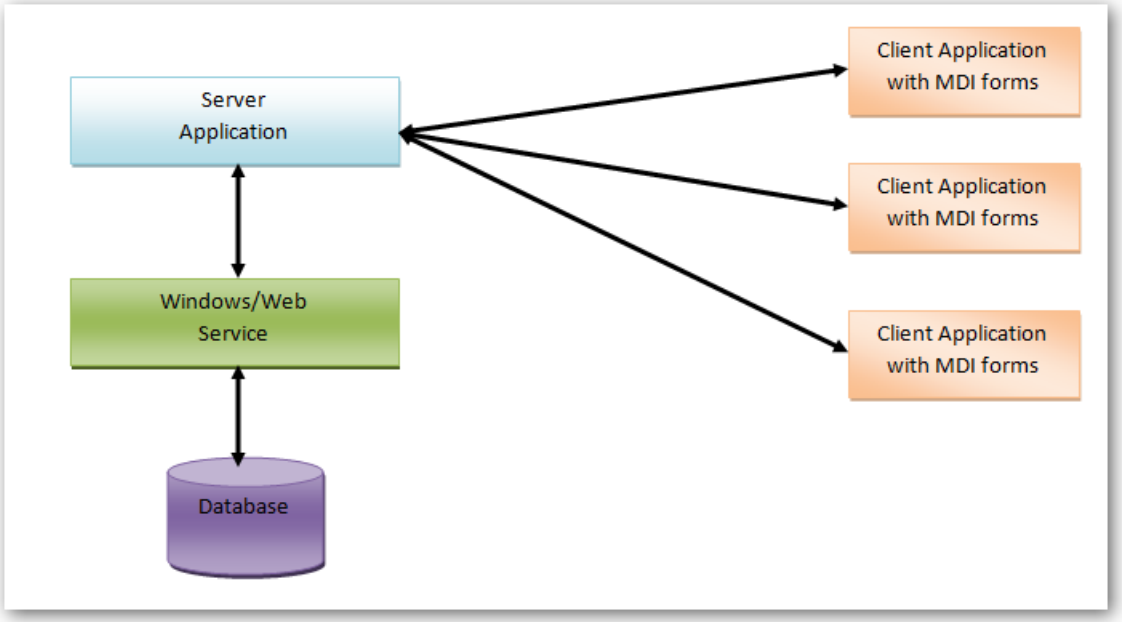
Unit testing was used to test the functionality of the application. This is elaborated at System testing section.

IV. Elaboration – Iteration 2

1. Architectural Design Refinement

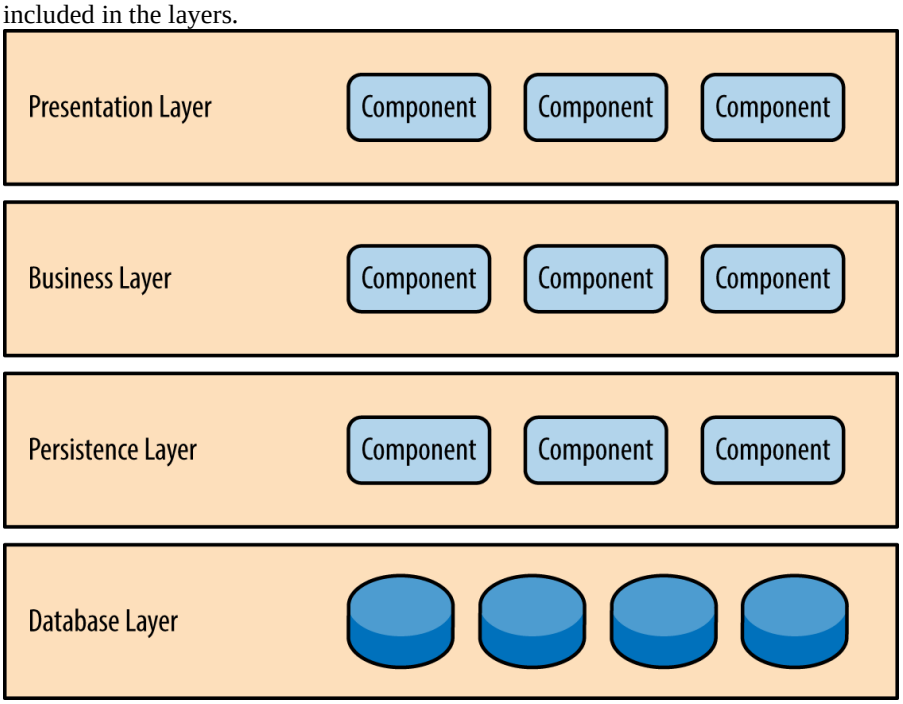
1.1. Conceptual Architecture

The conceptual architecture pattern chosen for this project is the **Client Server Architecture** as it is one of the most suitable architectures for web applications. This type of architecture has one or more client computers connected to a central server over a network or internet connection. This system shares computing resources.



1-1 Client Server Architecture

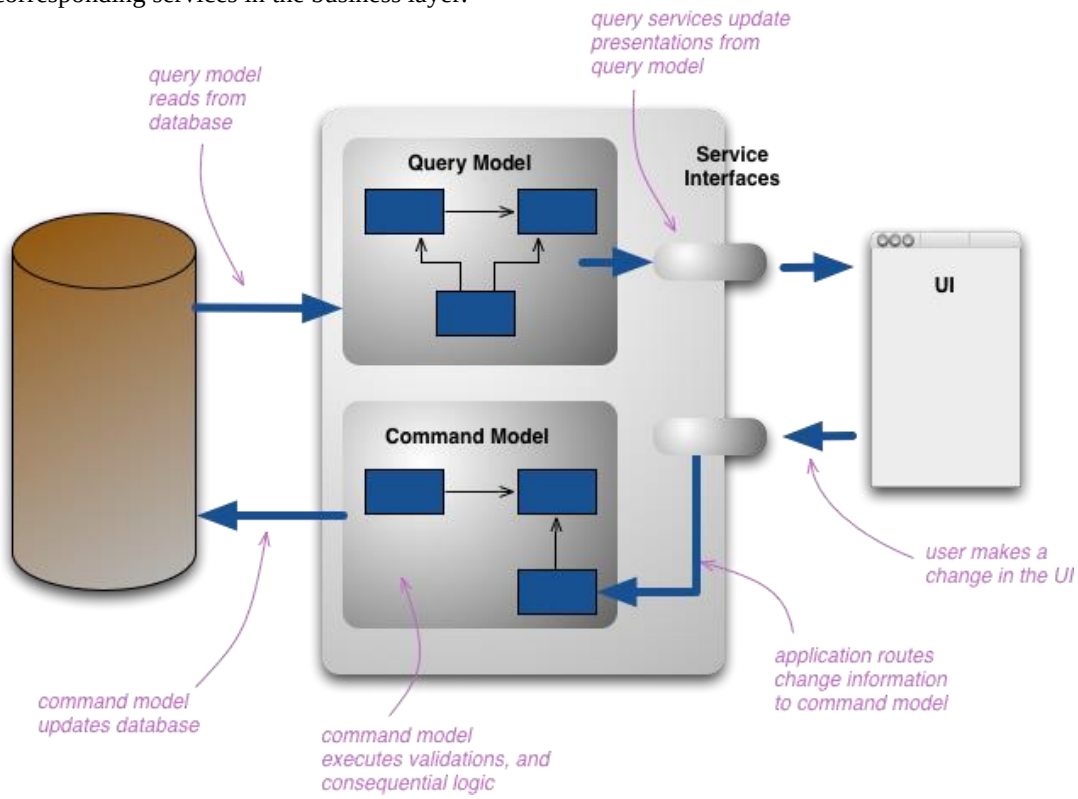
The backend is structured using the **Layered Architecture** pattern. This architecture separates the layers such that packages are loosely coupled: a layer may interact only with the layer beneath it. The layers are: Presentation layer, Business layer, Persistence layer and Database layer. The utilities package is not included in the layers.



1-2 Layered Architecture

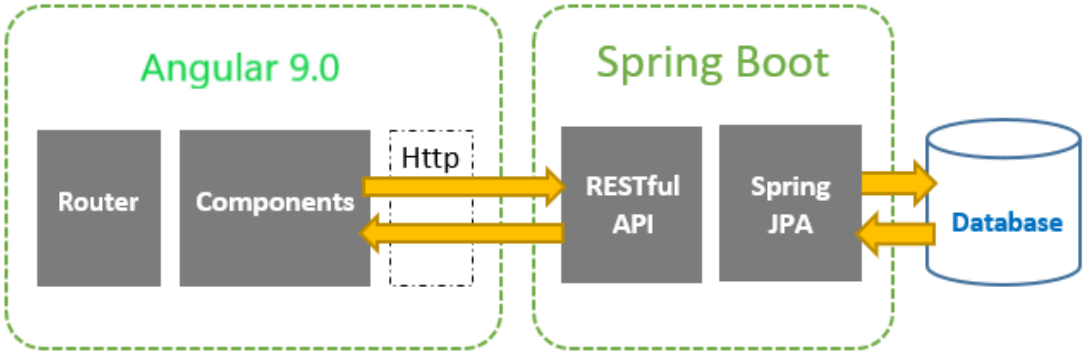
The backend is also built using a **CQRS Architecture**. CQRS stands for Command Query Responsibility Segregation. CQRS is an architectural pattern that separates command and query requests. Commands encompass create, update, and delete actions, while queries represent read actions. This pattern adds a layer of complexity, whilst de-coupling the code. The architecture was designed using the **Mediator Design**

Pattern. The Mediator pattern promotes code reusability and enables loosely coupled components. The mediator represents the link between requests and responses, i.e. a handler. The handler applies an action to the request in order to obtain the required response. A mapping is created between each request and its handler. As such, the controller's in the presentation layer communicate through the mediator with the corresponding services in the business layer.

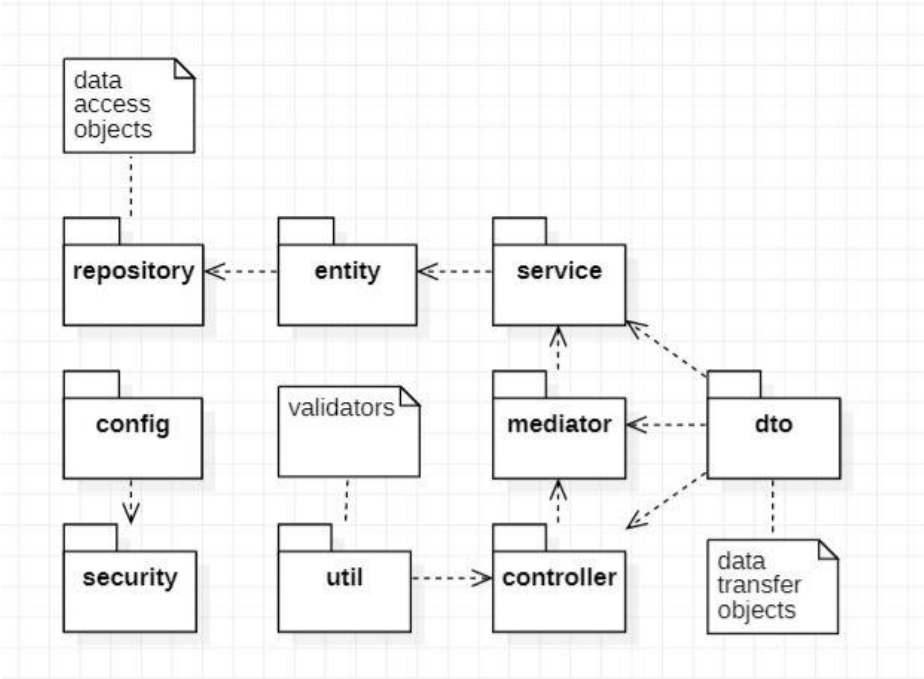


1-3 CQRS Architecture

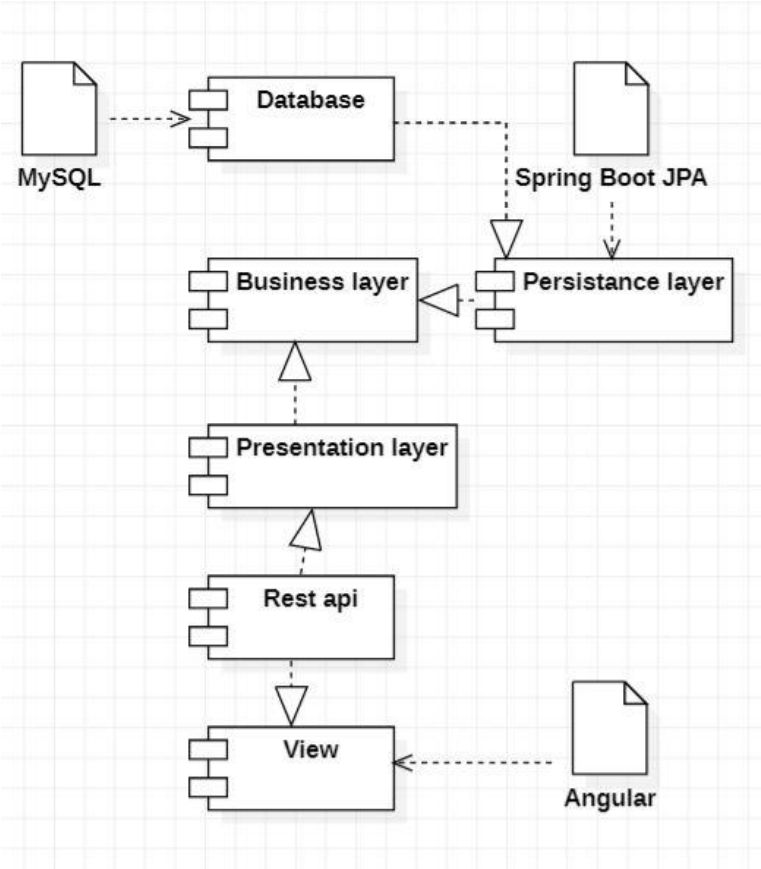
Domain-driven design (DDD) is the selected architectural style; it is an approach to software development for complex needs by connecting the implementation to an evolving model. The word domain refers to the subject area on which the application is intended to be apply. The reason why this style was chosen is that it enables the creation of complex designs based on the models of the domain. Another feature of DDD is the Building Blocks, namely several defined high-level concepts used to create and modify domain models (such as entity, service, repositories, factories, etc).



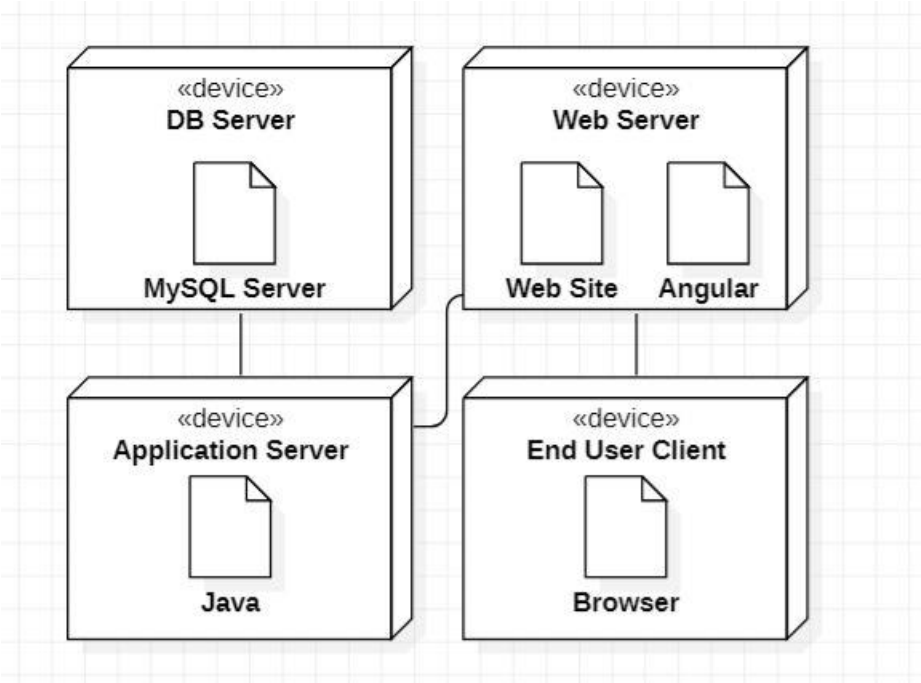
1.2. Package Design



1.3. Component and Deployment Diagrams



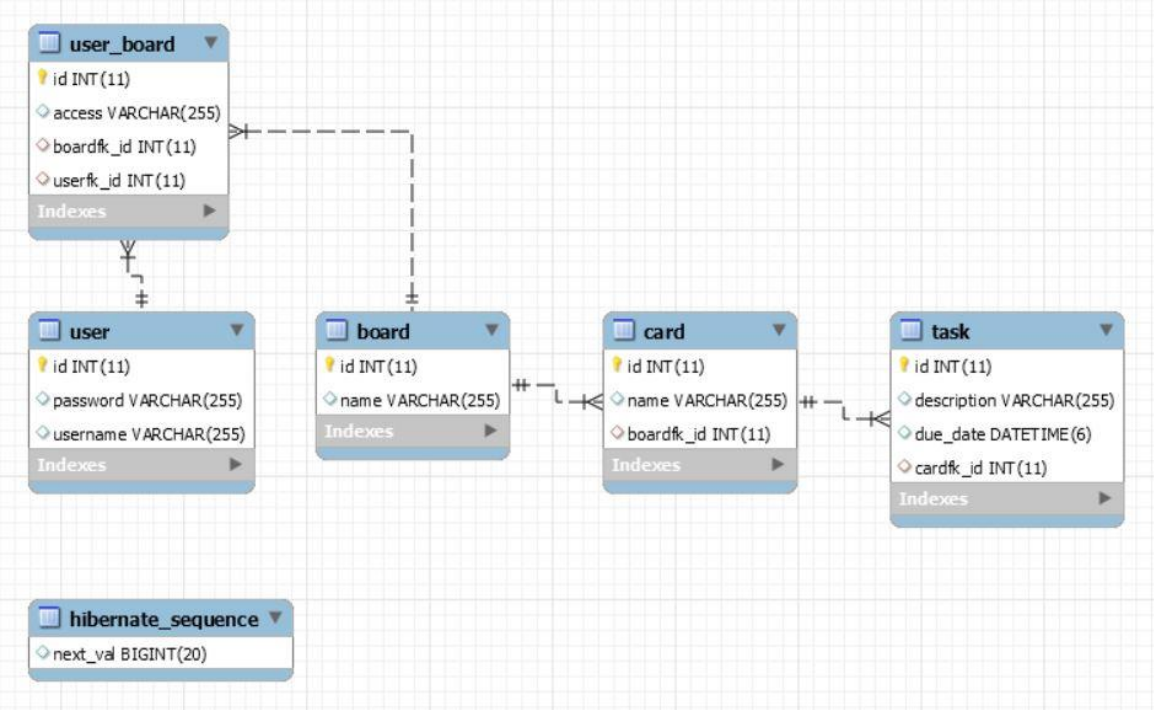
1-4 Component Diagram



1-5 Deployment Diagram

2. Design Model Refinement

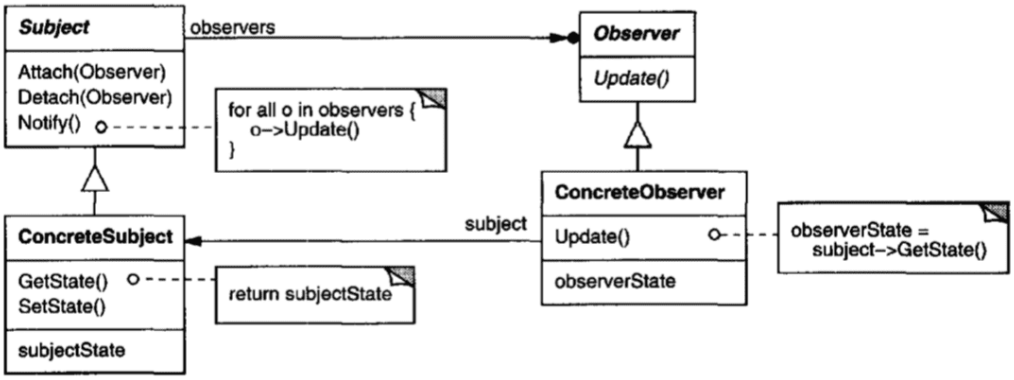
2.1. Data Model



2.2. Design Patterns

2.2.1. Observer Pattern

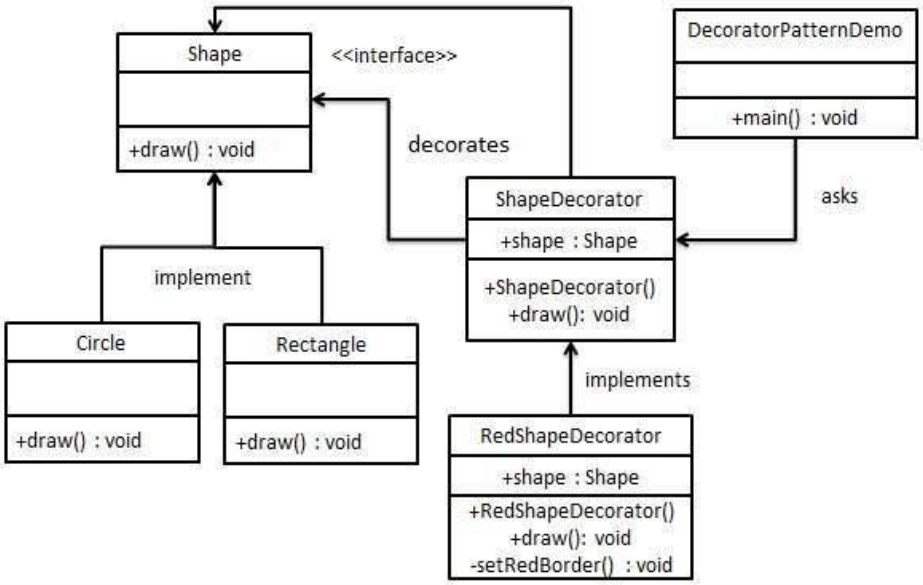
- Used in the frontend side to change the colour of the due dates in the tasks as such:
  - Orange for overdue tasks
  - Golden for almost due (less than 3 days left)
  - Green for not yet due (more than 3 days left)



- A subject computes the amount of days left until due date then sets the state to an integer and notifies observers
- An observer gets updated (notified) and checks the subjects state, then sets a variable accordingly
- In the Html, the colour of the due dates are chosen based on the observer's variable

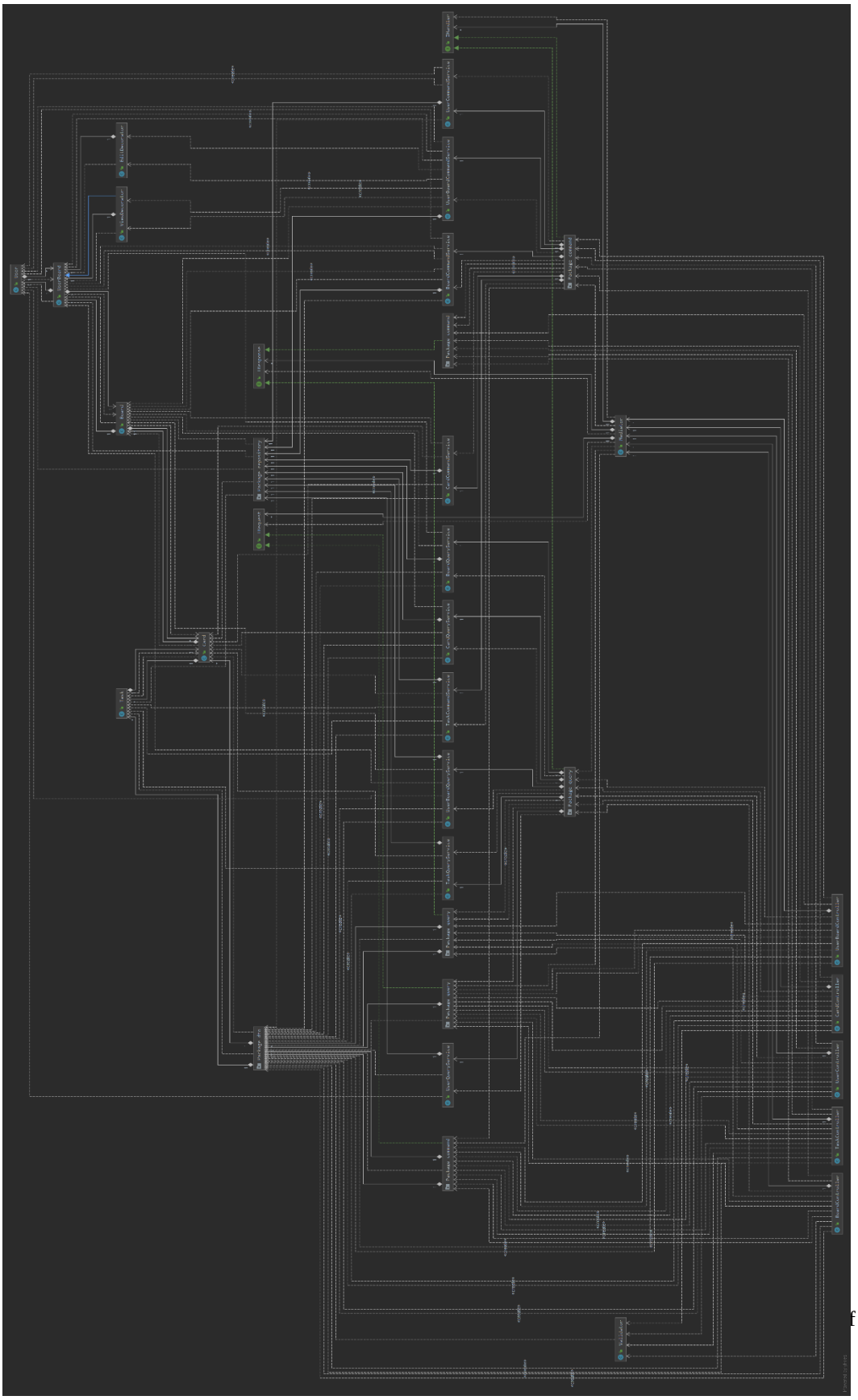
2.2.2. Decorator Pattern

- Used in the backend to automatically set access rights of new member based on user's choice (edit or view buttons)
- Ensures correct values for the access field (only edit and view)
- Implemented with the creation of two decorators (EditDecorator and ViewDecorator) which encapsulate a UserBoard object which contains the user to board mapping and the access rights.
- The decorators each contain methods which set the UserBoard's object access attribute to the corresponding value



2.2.3. **Mediator Pattern** – detailed in the architecture section

3. **UML Class Diagram**



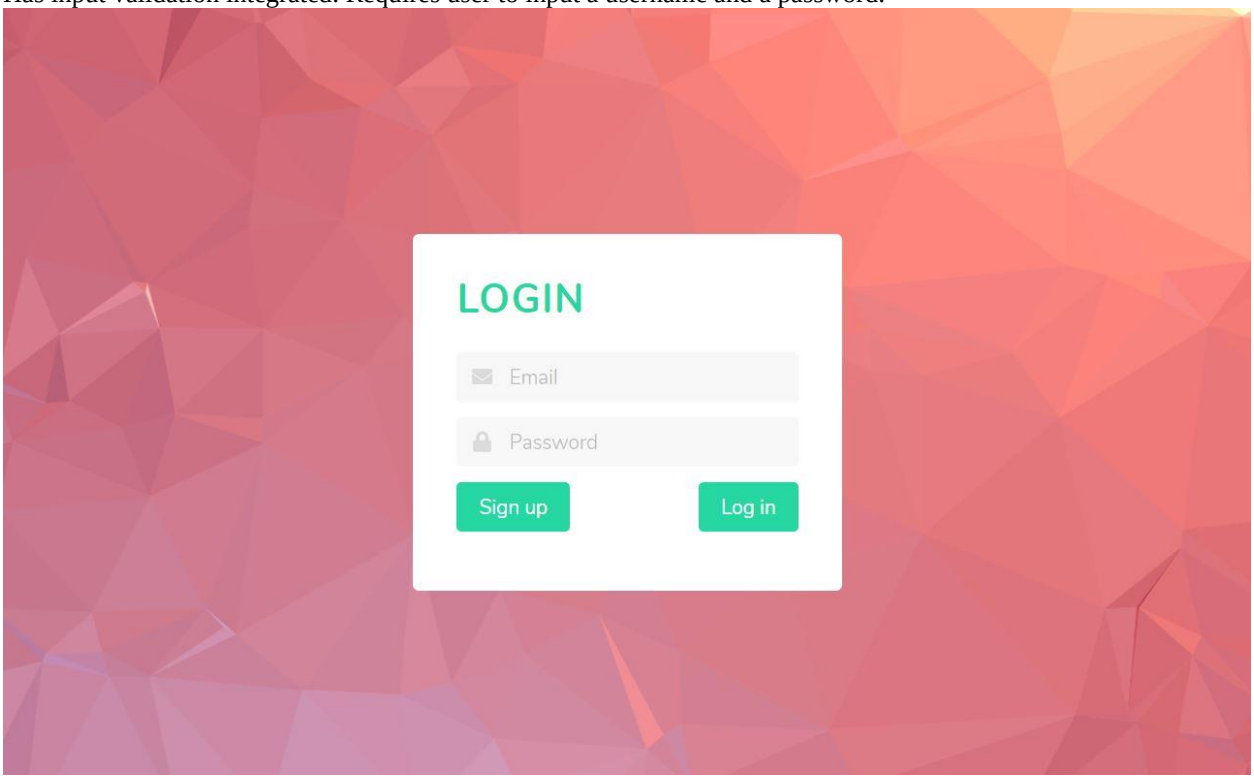
V. Construction and Transition

1. System Testing

The system is tested according to the use cases present in the Use Case Model document.

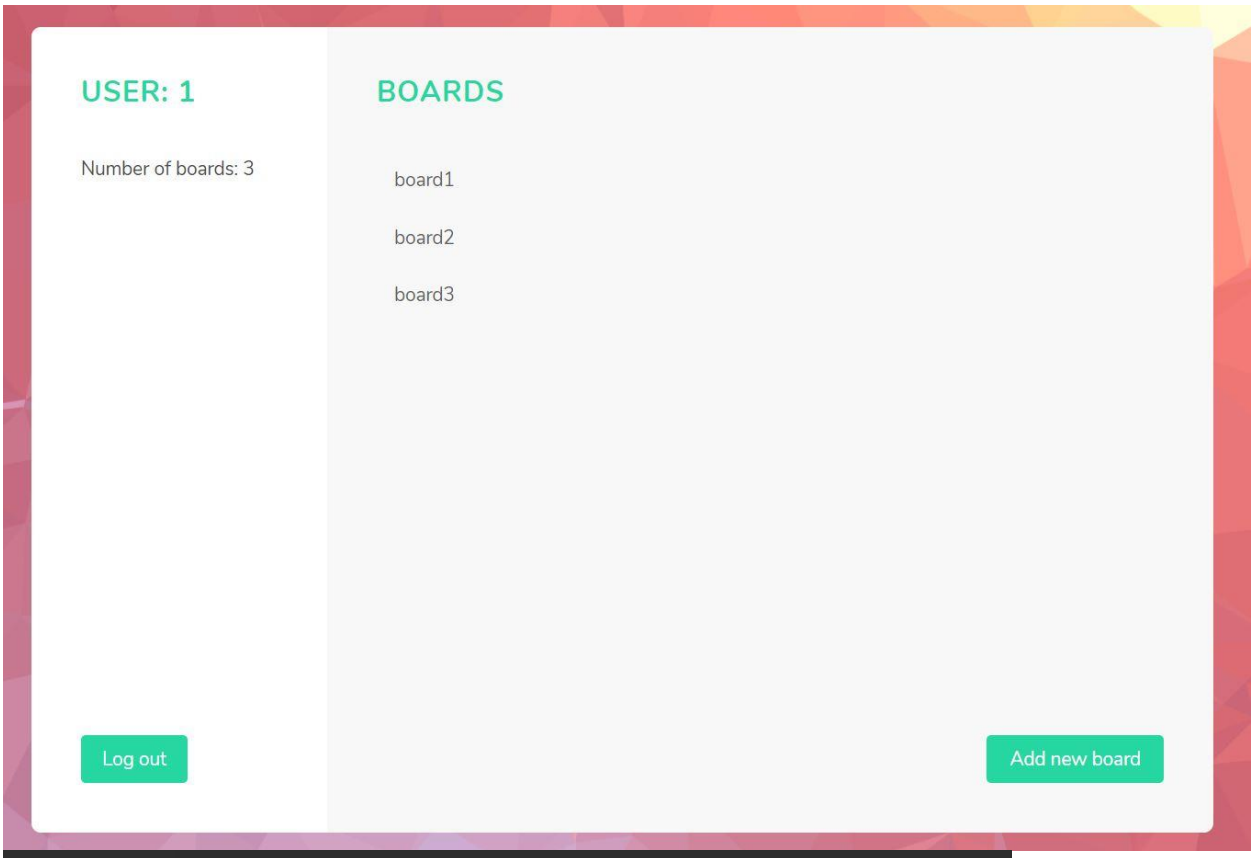
- Log in/Register page

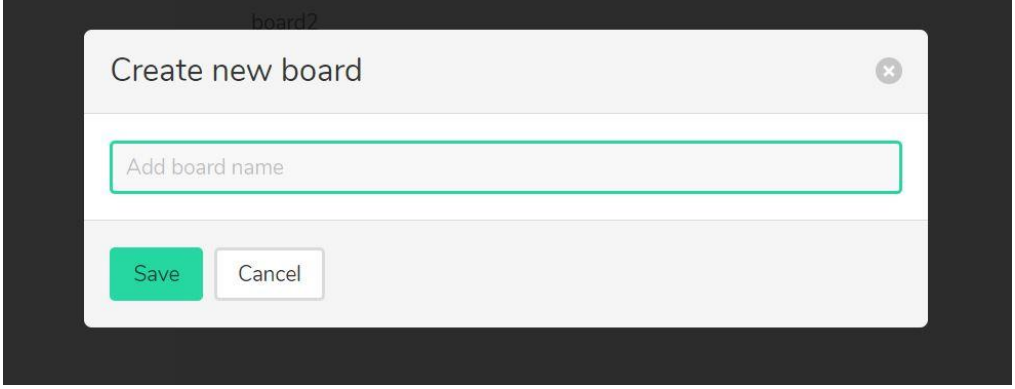
Has input validation integrated. Requires user to input a username and a password.



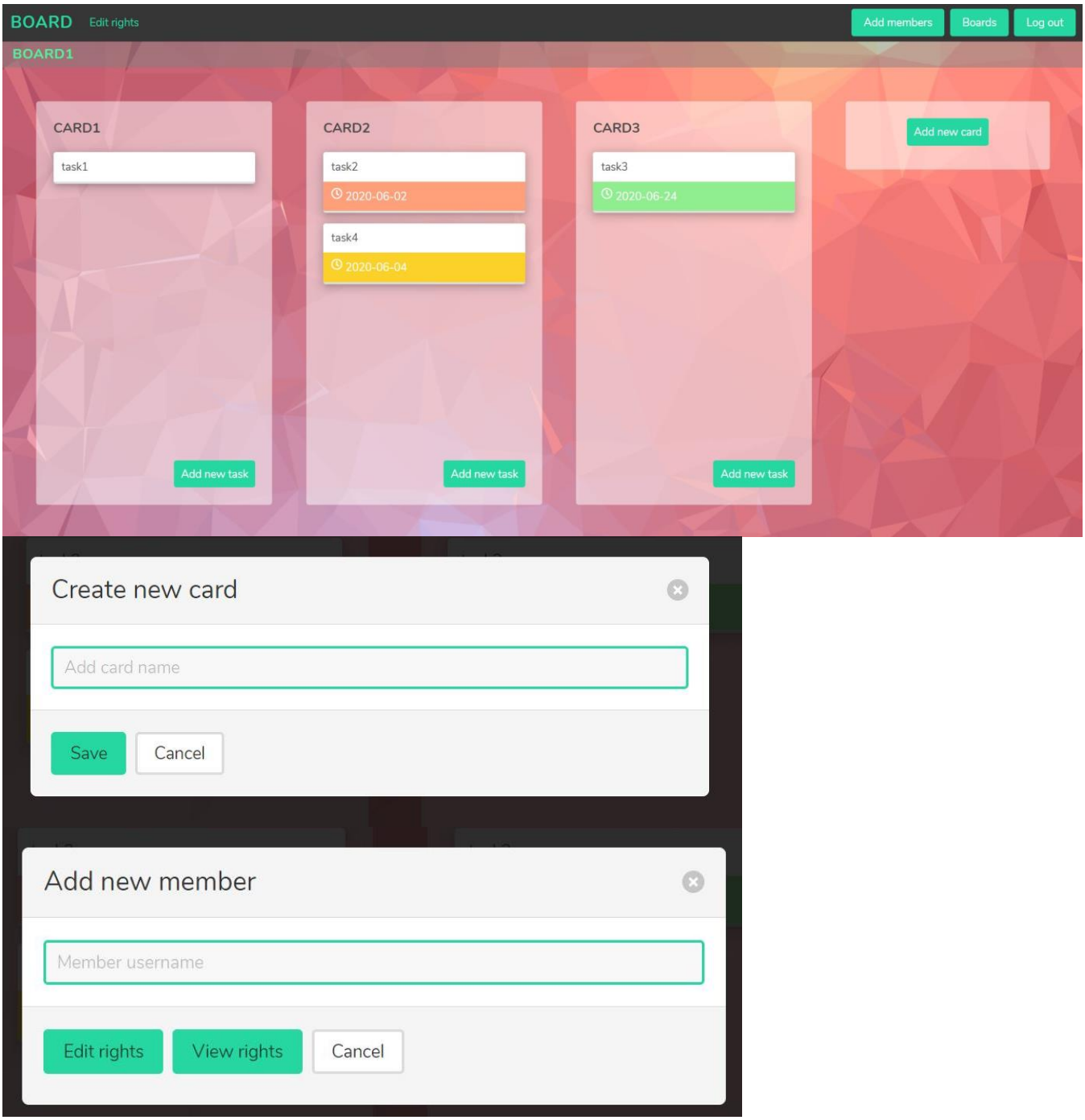
- View boards page

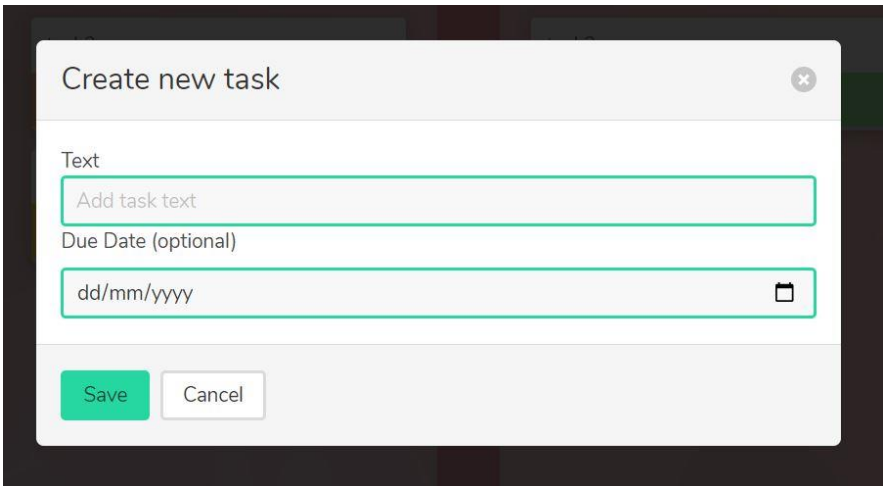
  Has:
  - Input validation
  - Modal window for adding board
  - Add board button
  - User info
  - List of boards





- Board page
  Has:
  - Add members button with modal window form with input validation
  - Log out button
  - Boards button
  - Add cards button with modal window form with input validation
  - Add tasks button with modal window form with input validation
  - Board name
  - Access rights
  - Cards with tasks that contain descriptions and optionally due dates


BOARD
Edit rights
Add members
Boards
Log out
BOARD1
CARD1
task1
Add new task
CARD2
task2
2020-06-02
task4
2020-06-04
Add new task
CARD3
task3
2020-06-24
Add new task
Add new card
Create new card
Add card name
Save
Cancel
Add new member
Member username
Edit rights
View rights
Cancel



2. Future improvements

A future improvement would be the integration of a bill splitter system such that the users can benefit from an economic management viewpoint as well.

VI. Bibliography

- https://martinfowler.com/
- https://material.angular.io/
- https://stackoverflow.com/