DreamCatch – Assignment 3

Student: Lupascu Liana

**Group: 30238**

Table of Contents

1. Requirements Analysis 3

1.1 Assignment Specification 3

1.2 Functional Requirements 3

1.3 Non-functional Requirements 3

2. Use-Case Model 3

3. System Architectural Design 4

4. UML Sequence Diagrams 6

5. Class Design 7

6. Data Model 8

7. Bibliography 8

1. Requirements Analysis

# Assignment Specification

For this assignment, I have designed and implemented an application that monitors the various topics we dream of and the quality levels of sleep over time. The application allows users to input a short description together with a set of tags to add an entry to the user’s specific list of dreams. Once an entry is created the user is able to evaluate the quality of his/her sleep with the following app metrics: duration, energy level, stress. These measurements are recorded on a scale of 1-5 for easy tracking. Given existing data the application is capable of aggregating these metrics across various tag categories and is able to present a daily chart for each quality metric given the user inputs a requested category.

# Functional Requirements

The application is required to :

* Use an ORM and a DI Container
* Use any OOP language ( in this case, Java)
* Use a client-server architecture
* Use a decorator pattern for changing the colour of the daily charts
* Use a CQRS architecture, use a mediator pattern to handle requests
* Have the data stored in a database
* All the inputs of the application will be validated against invalid data before submitting

the data and saving it in the database.

# Non-functional Requirements

The user should be able to input dreams and generate graphs based on the dreams that they had in the past, observing how their stress, energy and duration level have evolved over time.

2. Use-Case Model

Use-Case Description:

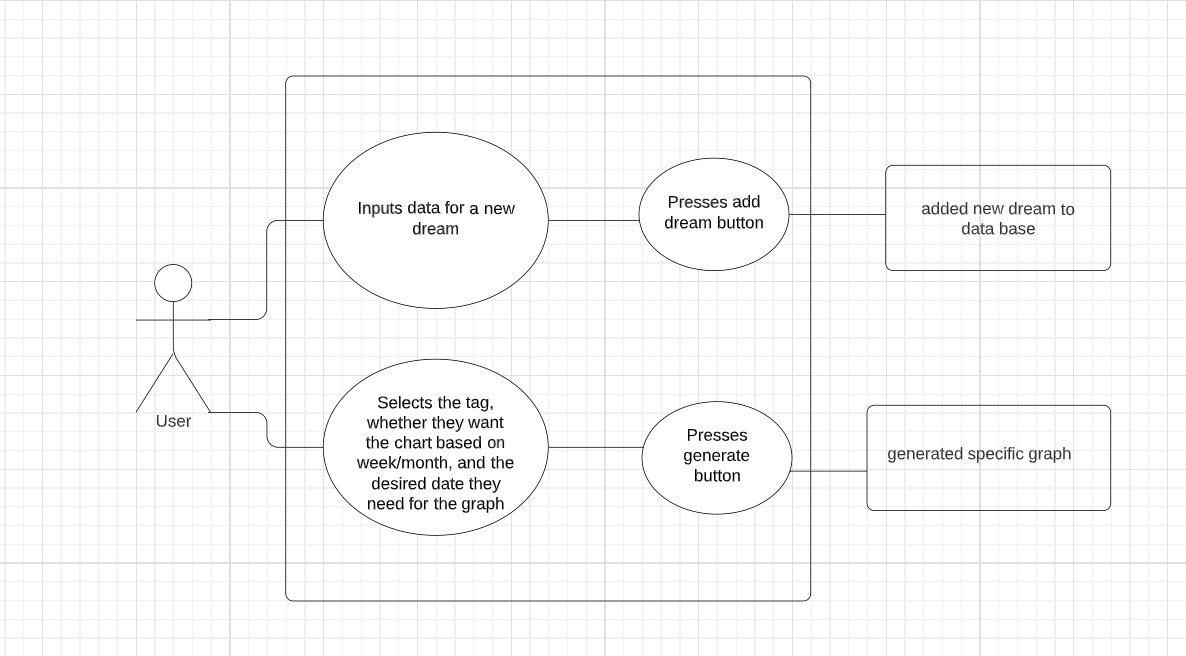
Use case: Add Dream

Level: User-goal level

Primary actor: User

Main success scenario:

* 1. System presents a form for the user to input a short description and set of tags for the dream.
  2. User fills out the form and submits it.
  3. User selects "Add Dream" from the Input Dream Data screen.
  4. System validates the data and creates a new entry for the dream in the user's or application-defined category.



3. System Architectural Design

**3.1 Architectural Pattern Description**

CQRS (Command Query Responsibility Segregation) is an architectural pattern that separates the responsibility for handling read (query) and write (command) operations on a data model. In a CQRS architecture, the read and write models are kept separate, and each model is optimized for its respective use case. The pattern aims to provide a more scalable and flexible architecture by allowing different models to be optimized for different needs.

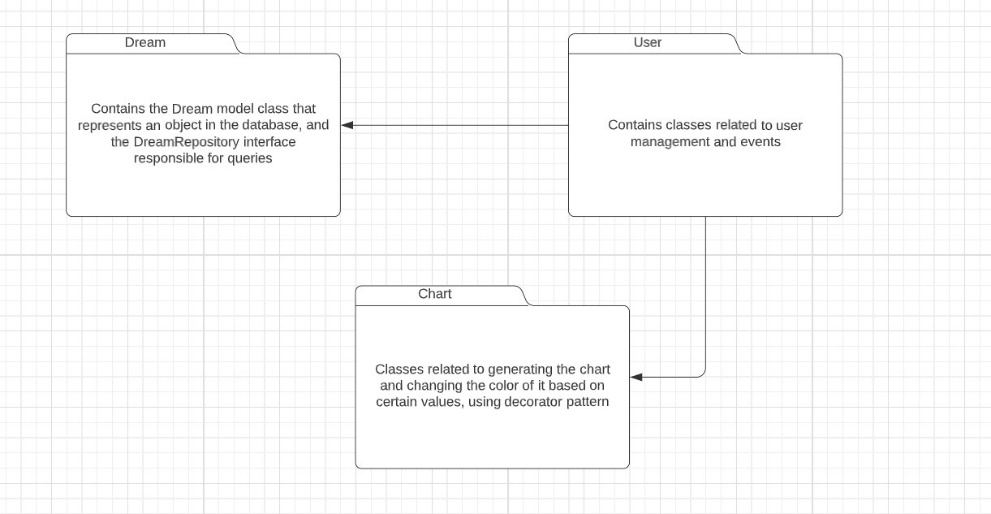
For my project, I have used the CQRS pattern on backend when adding a new dream, using Java. Specifically, I have implemented the write side of the pattern using the classes CreateDreamCommand, DreamCreatedEvent, DreamEventHandler, and DreamAggregate.

The CreateDreamCommand class represents a command to create a new dream. When a user submits a dream through the frontend, a new CreateDreamCommand instance is created and sent to the backend. This command is handled by the DreamAggregate class.

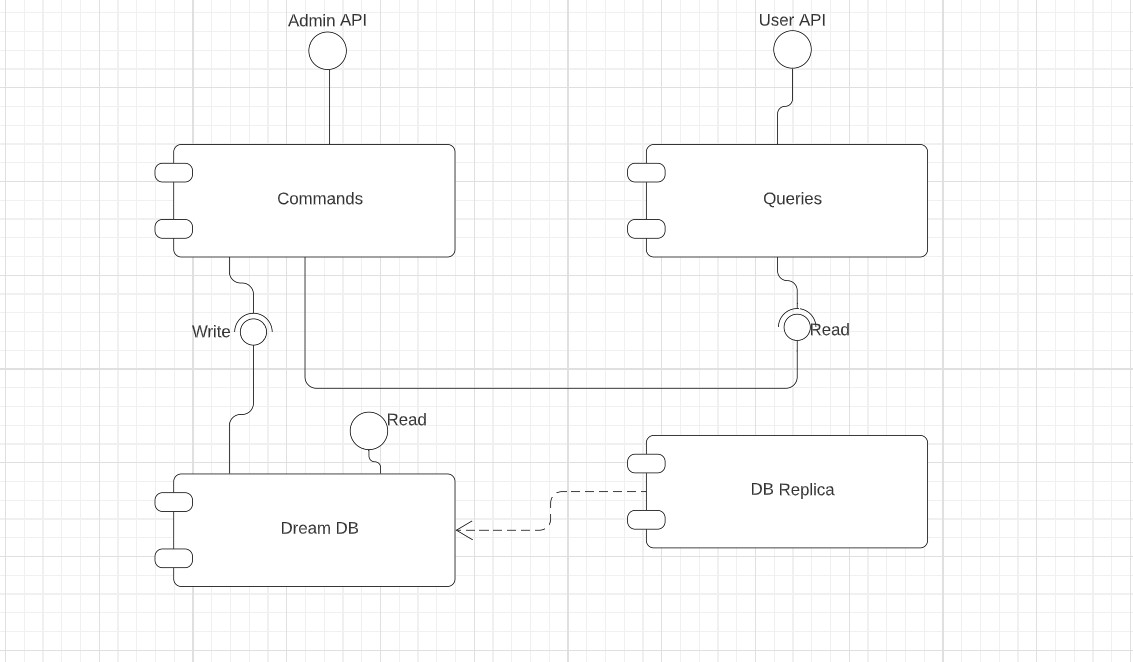
The DreamAggregate class is responsible for handling the CreateDreamCommand and creating a new DreamCreatedEvent. This event contains the data for the new dream that was created. Once the event is created, it is sent to the DreamEventHandler.

The DreamEventHandler class is responsible for handling the DreamCreatedEvent and updating the write model of the dream data. In your project, this includes adding the new dream to the user's list of dreams.

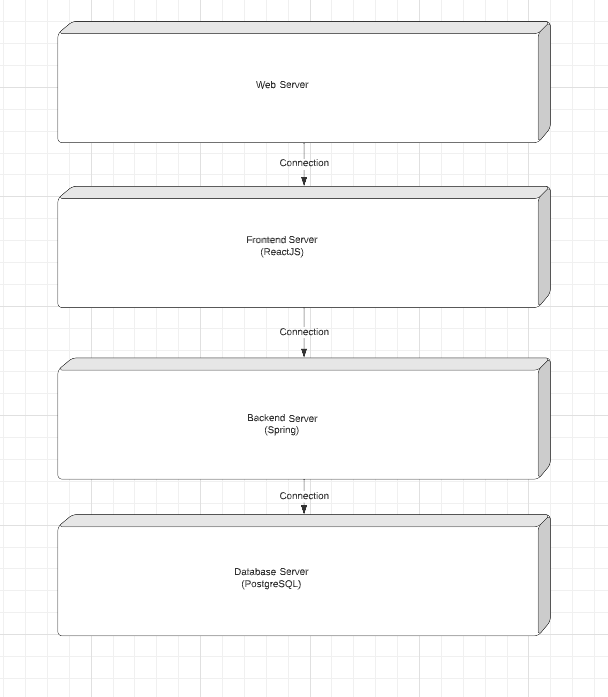
**3.2 Diagrams**



* *package diagram -*

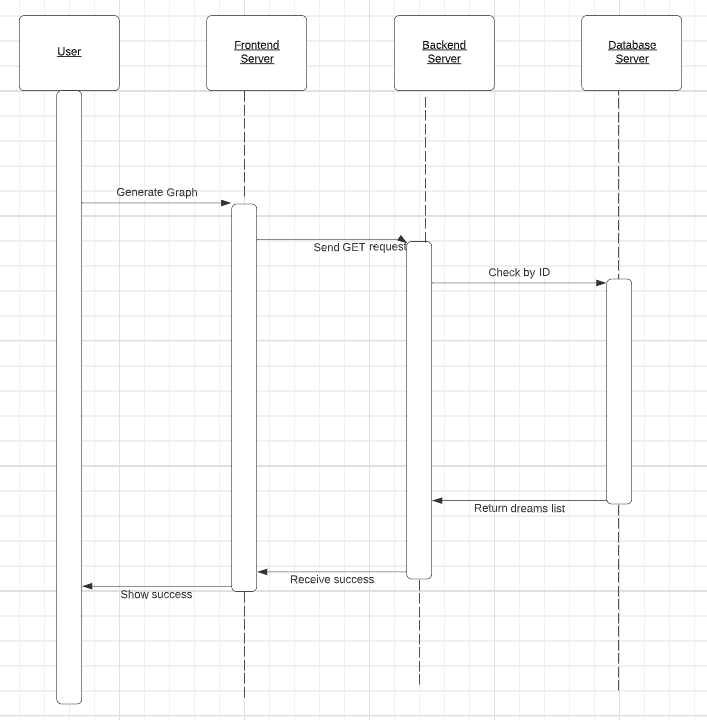


* *component diagram -*



* *deployment diagram -*

4. UML Sequence Diagrams



In this sequence diagram, the user starts by selecting a specific tag, whether or not they want the graph based on week/month, and introducing their ID and wanted week/month. They press the Generate Graph button from the application through the frontend server. The frontend server then sends a GET request to the backend server, which returns the user’s list of dreams from the database server that matches the user’s specifications. Once the list is returned successfully, the backend server sends a success response back to the frontend server, which uses the received data to generate a graph and display it to the user.

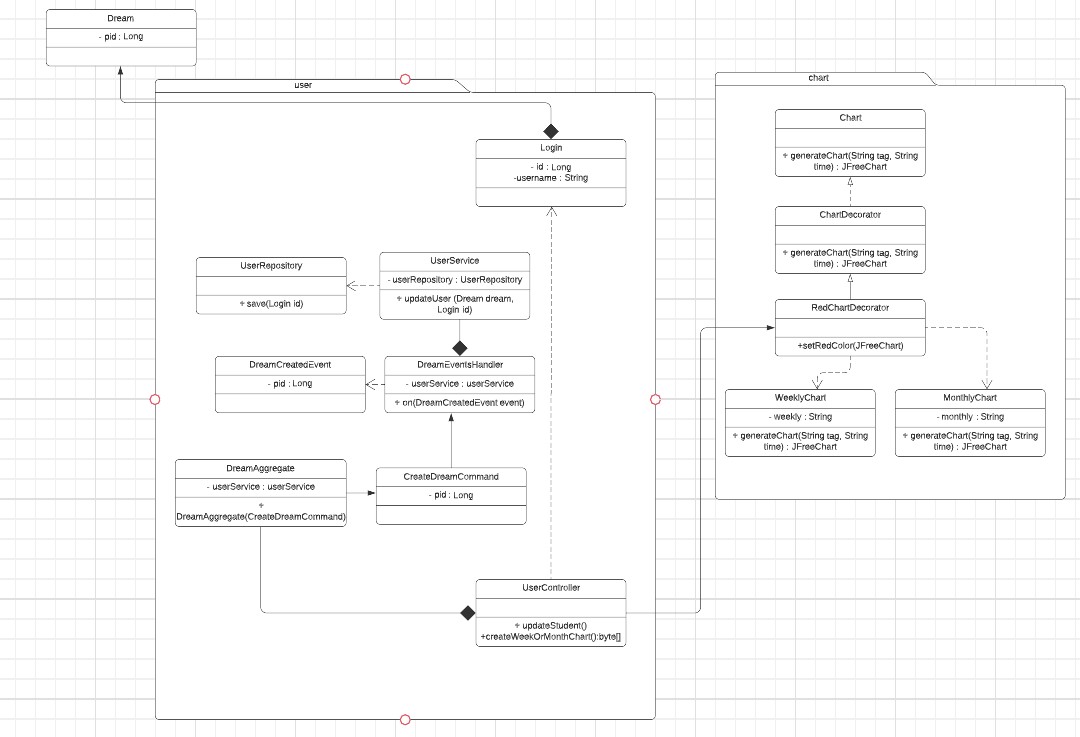
5. Class Design

**5.1 Design Patterns Description**

The decorator pattern is a design pattern that allows behavior to be added to an individual object, either statically or dynamically, without affecting the behavior of other objects from the same class. In other words, it adds functionality to an object without changing its original structure.

In my project, I used the decorator pattern to change the color scheme to the bar charts generated by JFreeChart. The JFreeChart library provides a basic set of color schemes for bar charts, but I wanted to change the color of the bars based on their value, specific for each metric. To achieve this, I created a custom renderer class named RedChartDecorator that extends the ChartDecorator interface. Using the values retreived from the plot of the chart, I use the library to render the chart.

**5.2 UML Class Diagram**



6. Data Model

1. User model: This model represents a user of the application. It contains fields such as the user's id (autogenerated with each input in the database), username, password, and a list of dreams, represented by Dream objects.

2. Dream model: This model represents a dream entry in the system. It contains fields such as the dream description, tag (Happy, Nightmare or Weird), energy, stress and duration levels, and. Additionally, it may contain fields for quality metrics associated with the dream, such as duration, energy level, and stress and an id (also autogenerated).

Both of these models are essential for the functioning of the system. The User model allows users to input and track information related to their dreams, while the Dream model allows users to know in-depth information about their sleep quality metrics.

7. Bibliography

1. <https://www.javatpoint.com/spring-boot-architecture>

2. <https://www.tutorialspoint.com/design_pattern/decorator_pattern.htm>

3. <https://www.baeldung.com/cqrs-event-sourcing-java>

4. <https://www.tutorialspoint.com/hibernate/orm_overview.htm>