Design patterns

Assignment 1 - Task 2

This document provides an exhaustive overview of the Design Patterns used and implemented by our group (Group 34a) for the Rowing Association project. From the broad variety of Design Patterns available and compatible with our requirements and problems, we decided on choosing and applying two of those: the Builder Pattern and the Strategy Pattern.

The builder pattern was chosen to create big objects with a lot of attributes in an easily readable way.

Strategy Pattern was chosen to have in mind the event-microservice from our implementation when users want to get the matching events based on their preferences/availability.

In the upcoming part of this document, deeper explanations of those two Patterns can be found, followed up by the corresponding diagrams.

Strategy pattern

The Strategy pattern was implemented in the system (Event microservice) to allow users to view a list of valid events sorted by different criteria. There are two sort strategies available: sorting by event ID (time they have been added to the database) and sorting by time (start time of the event).

The event ID sort strategy sorts events based on the time they were added to the database, with events added earlier appearing first. This is useful for users who want to see the events in the order they were added.

The time sort strategy sorts events based on the start time of the event, with events starting sooner appearing first. This is useful for users who want to see the events that are happening soonest first.

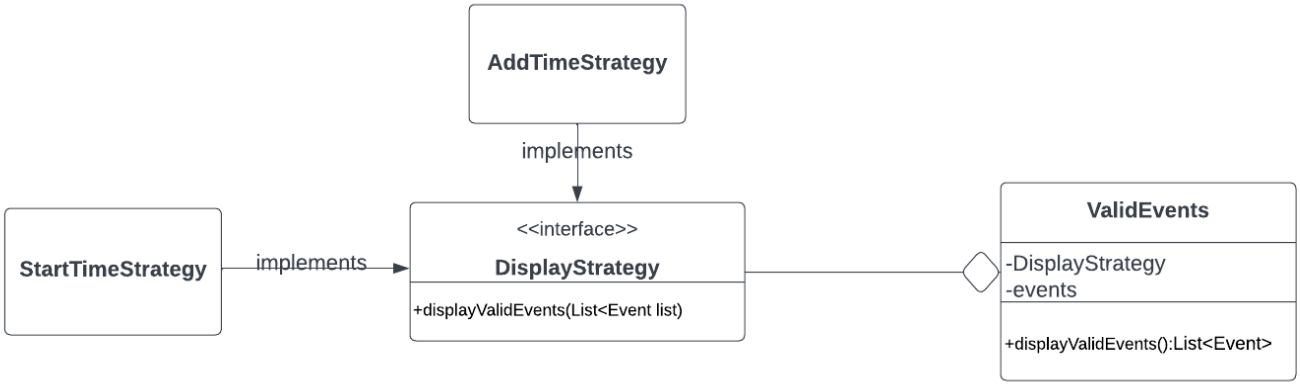
The Strategy design pattern was implemented by creating two Concrete Strategies(AddTimeStrategy and StartTimeStrategy) classes which implement the Display Strategy interface and define the logic for comparing events based on their IDs and dates/times, respectively. These two Concrete Strategy classes can be used to sort the list of events in different ways depending on the desired criteria.

The ValidEvents (the Context) class uses the Strategy pattern to allow users to display a list of events sorted by different strategies. The class has a field for a DisplayStrategy object, which

can be set to an instance of either the AddTimeStrategy or the StartTimeStrategy class. The displayValidEvents() method of the ValidEvents class uses the displayValidEvents() method of the DisplayStrategy object to sort and display the list of events. This allows the sorting logic to be encapsulated in the comparator classes. We use the class after we already obtain all valid events for the user and just before showing them to him.

This design pattern allows a very easy extension of sorting strategies by simply creating your specific classes, which also makes scalability easier.

Class diagram:



Builder Pattern

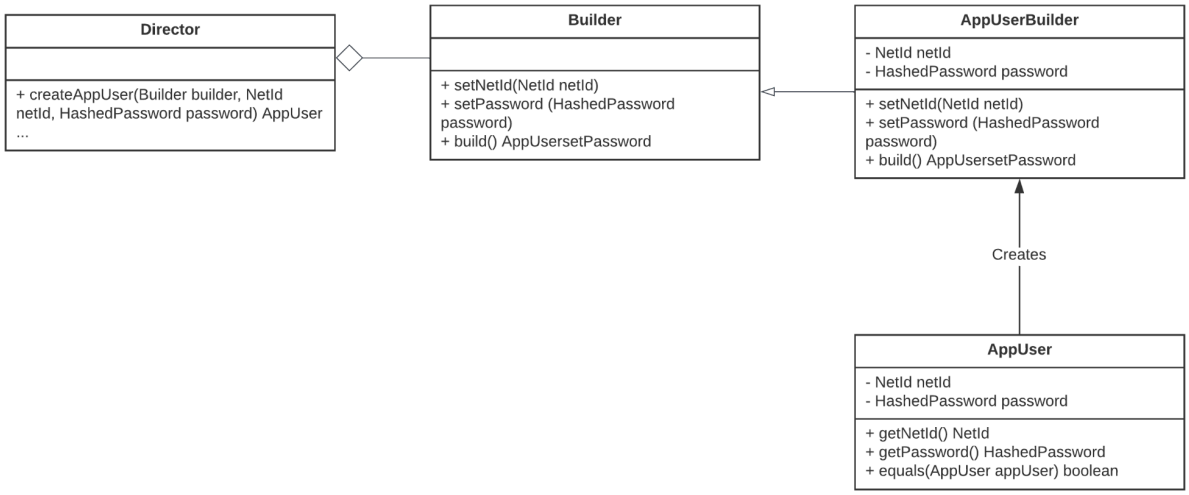
In our code, we implemented the Builder pattern to create objects in a controlled and flexible way. We defined a Builder interface with methods for building the various parts of the object and then created a concrete implementation of the Builder interface for each object that we wanted to build.

For example, to build events, we created an EventBuilder class that implemented the Builder interface. This EventBuilder class had methods for setting the values of the event object, such as the event type, admin, and date-time.

We also created a Director class to oversee the object construction process. The Director class directed the builder to set up the values of the object being built, and then called the build method of the Builder to return the completed object.

In our controllers, we used the Builder pattern to save objects to the database. This allowed us to have more control over the construction process and to easily extend the pattern to handle

other objects in the future. We used the Builder pattern to build the AppUser in the user microservice, the Event in the event microservice, and the AppUser in the authentication microservice. Overall, the Builder pattern proved to be a useful tool for creating complex objects in a flexible and extensible way.



The class diagram above shows the classes that were used to implement the Builder Pattern. Similarly, the construction for the Event and another user used the same principles.