Analysis and Design Document
Student: Şinca Mădălina
Group: 30433

Revision History

Date	Version	Description	Author
10/04/2018	1.0	Initial iteration of document	Șinca Mădălina
22/04/2018	1.2	Second iteration of document	Șinca Mădălina

Table of Contents

I. Project Specification

The purpose of this document is to collect, analyze, and define high-level needs and features of the Arch Climbing Wall (ACW) Web Application. It focuses on the capabilities needed by the stakeholders and the target users, and why these needs exist. The details of how the ACW Web Application fulfills these needs are detailed in the use-case and supplementary specifications.

II. Elaboration – Iteration 1.1

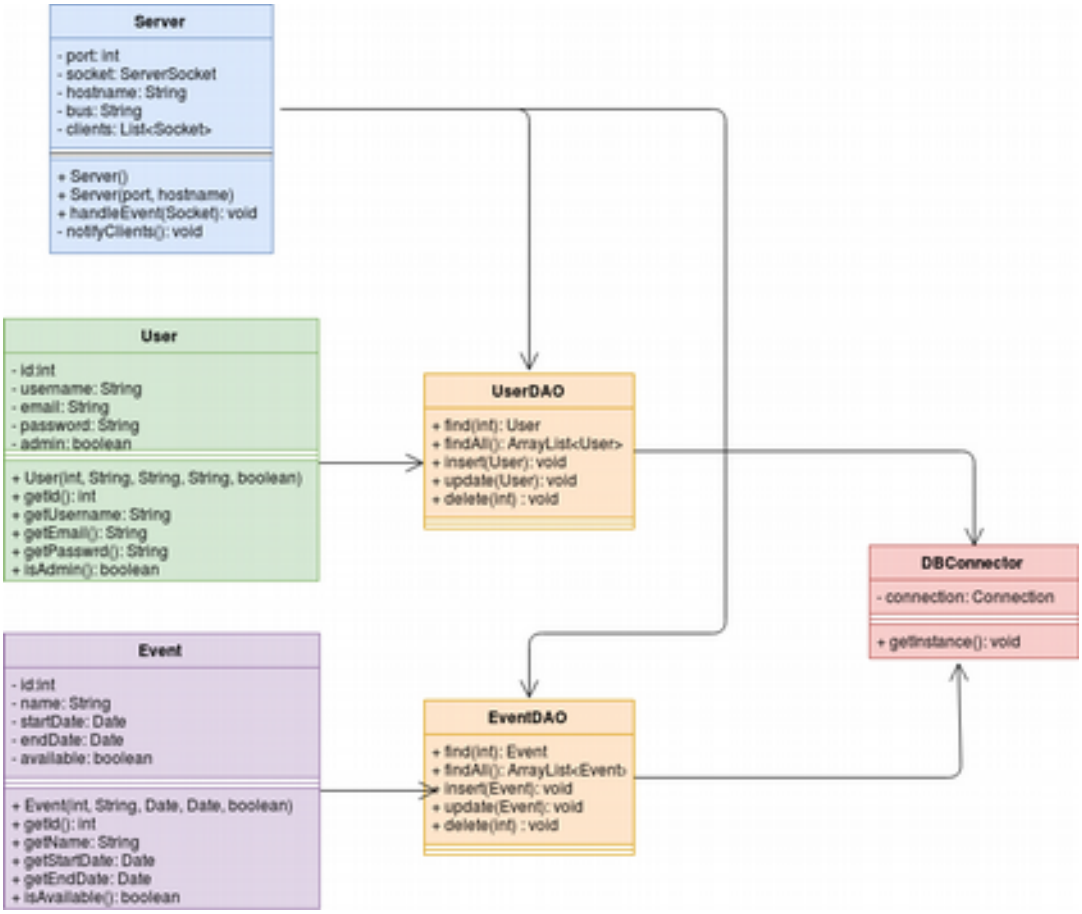
1. Domain Model

The Client will contain 1 model:

- *User – represents a user, either the active one or inactive one.*

The Server consists of the following models:

- *User – same as above;*
- *Event;*
- *Course;*



2. Architectural Design

2.1 Conceptual Architecture

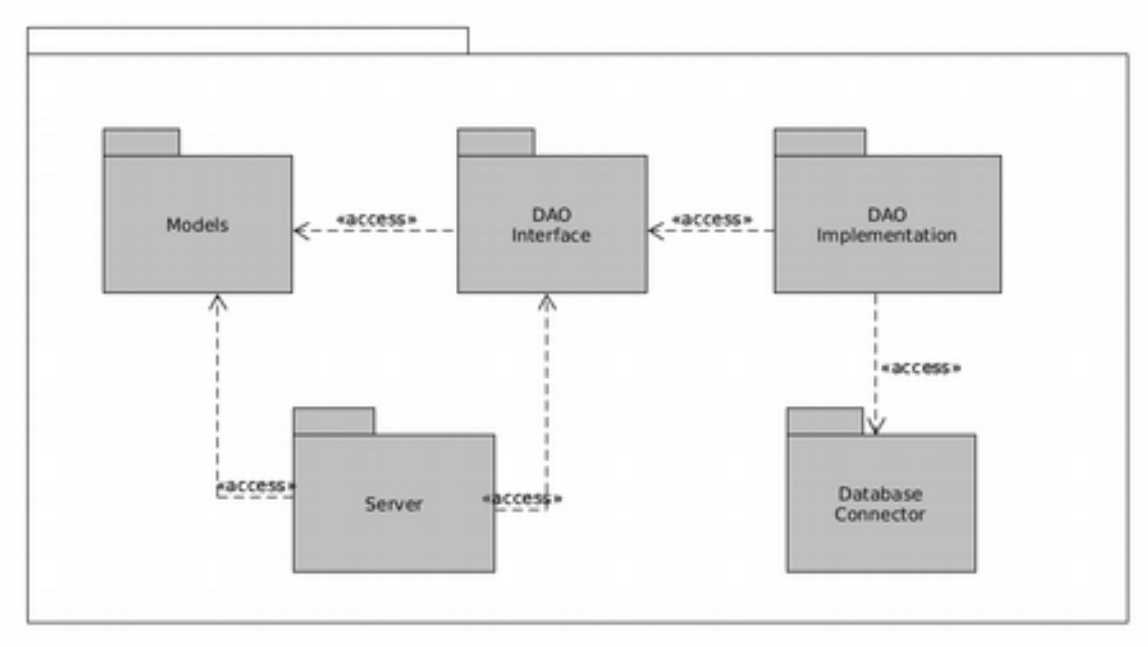
The system consists of two endpoints : the users and the server. The used architectural patterns are:

- Client – Server : The server sends request / response messages to each client at a given point in time and vice-versa. While communicating with all its clients, the server also listens for new connections;
- Event Bus : The server acts as a virtual bus. Every client listens for events while performing playback. When a client wants to perform an action (eg. pause, stop), it sends a message to the server telling it to place the message on the bus, and all the other clients are then notified.

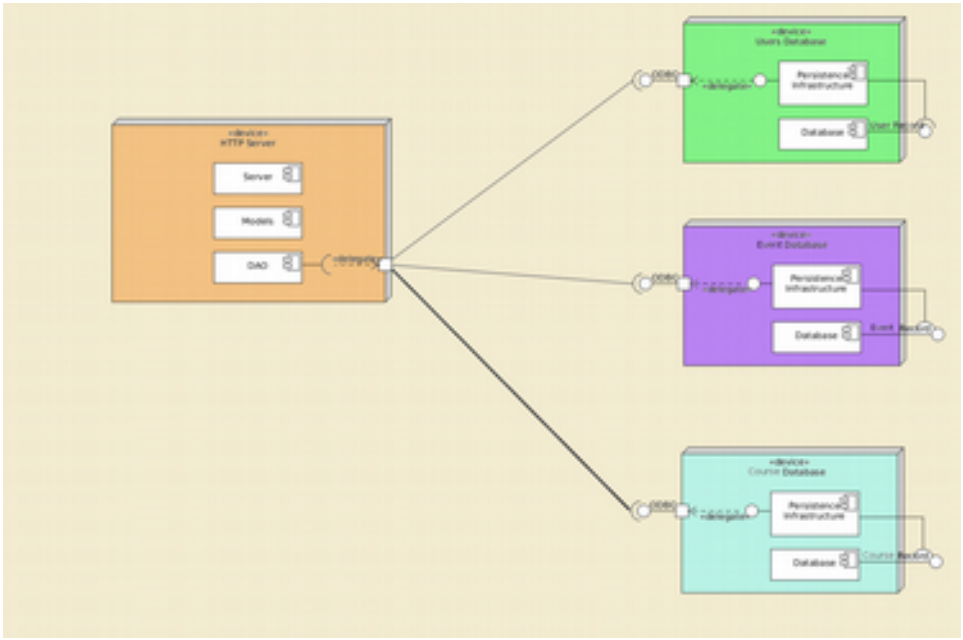
MVC + C-S diagram here

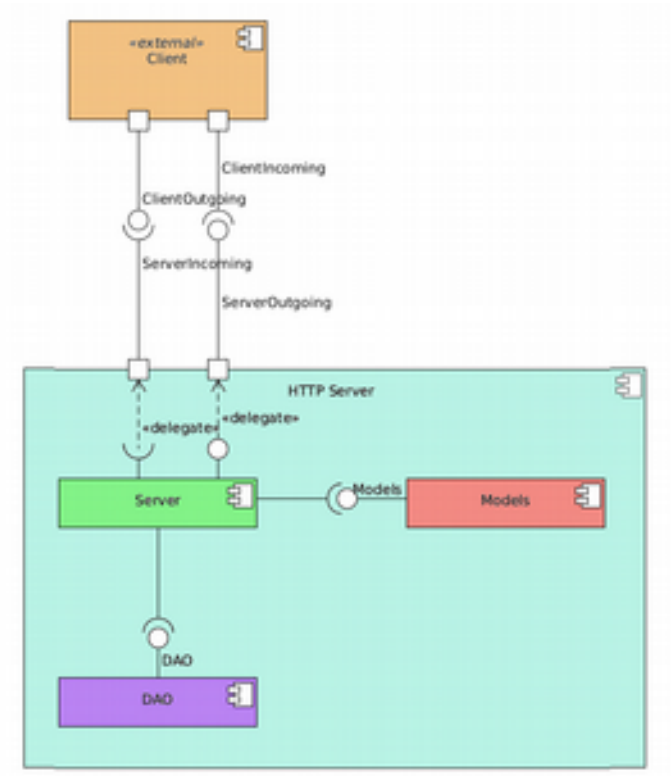
The described approach has been selected due to its simplicity and performance.

2.2 Package Design



2.3 Component and Deployment Diagrams




«external»
Client
ClientIncoming
ClientOutgoing
ServerIncoming
ServerOutgoing
HTTP Server
«delegate»
«delegate»
Server
Models
Models
DAO
DAO

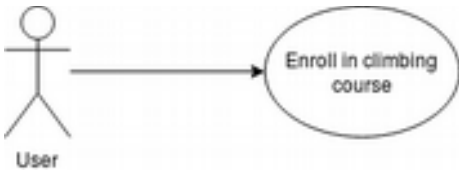
III. Elaboration – Iteration 1.2

1. Design Model

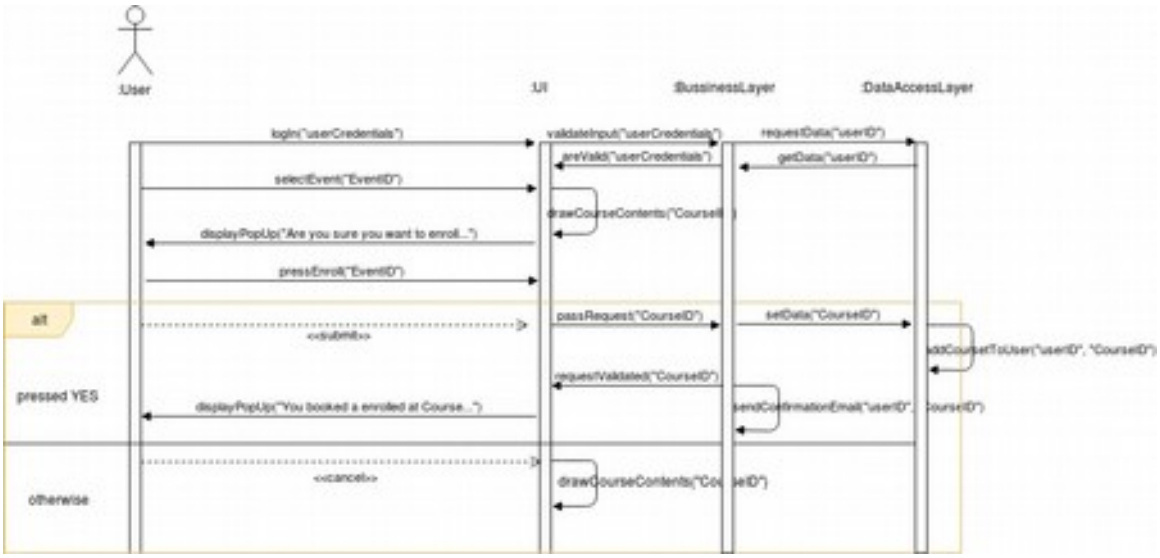
1.1 Dynamic Behavior

Course Enrollment Steps:

- The user logs in;
- The user queries through the available course options;
- The user selects the course he/she desires to enroll in (or wants to enroll their children in);
- The user presses the “enroll me” button at the bottom of the page, below the course description;
- Pop-up window displaying “Are you sure you want to enroll to Course X?” - No, Yes;
- The user presses Yes;
- Pop-up displaying “You have just enrolled to Course X. You must have received an e-mail confirmation and Course X should now appear under the <<my courses>> section in your profile”;
- The user will receive an e-mail confirmation and the course they just enrolled in will appear under the “my courses” section in their profile;

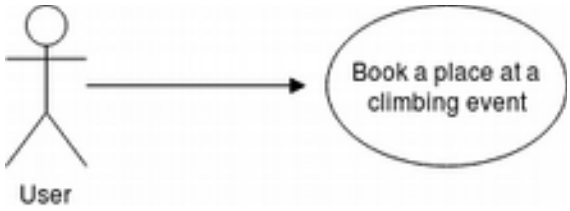


- ***Extensions:*** If the user presses the No button then the application cancels the enrollment and return to Course X main page.



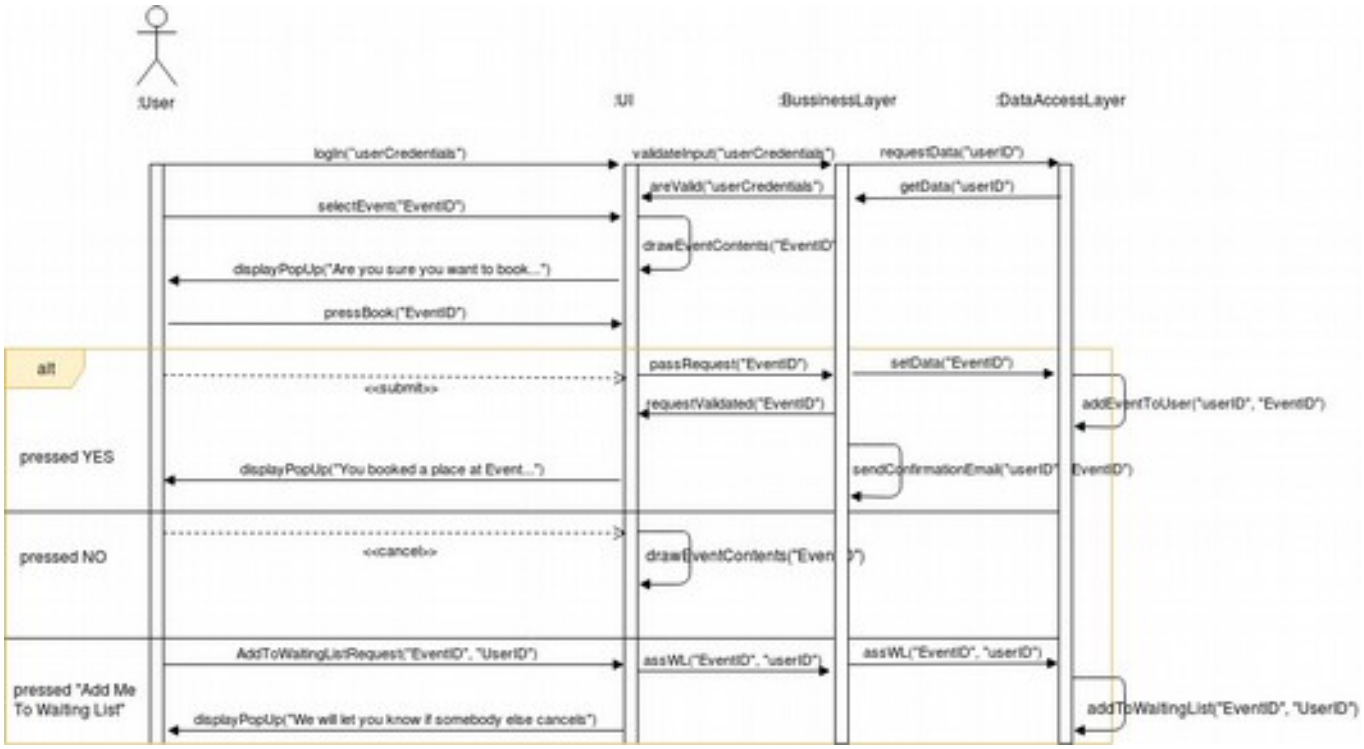
Event Booking Steps:

- User logs in;
- The user queries through the available events options;
- The user selects the event he/she desires to attend;
- The user presses the “Book A Place” button at the bottom of the page, below the event description;
- Pop-up window displaying “Are you sure you want to book a place at Event X?” - No, Yes;
- The user presses Yes;
- Pop-up displaying “You have just booked yourself a place for Event X. You must have received an e-mail confirmation and Event X should now appear under the <<my events>> section in your profile”;
- The user will receive an e-mail confirmation and the event they just booked will appear under the “my events” section in their profile;



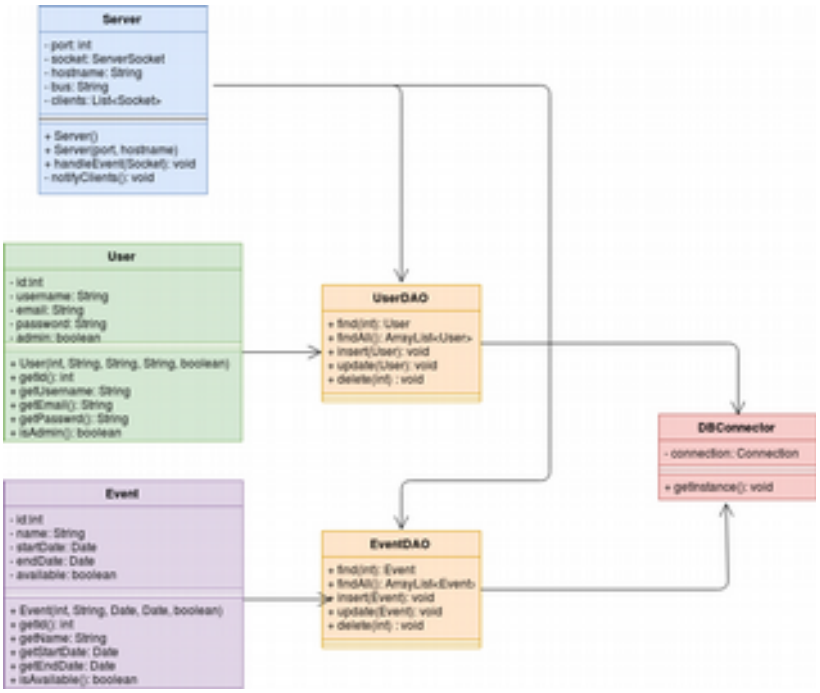
- ***Extensions:*** If there are no more places available at said event, then the user has the option o pressing “Let Me Know If a Place Becomes Available” button.

- If the user presses the No button then the application cancels the enrollment and return to Event X main page.

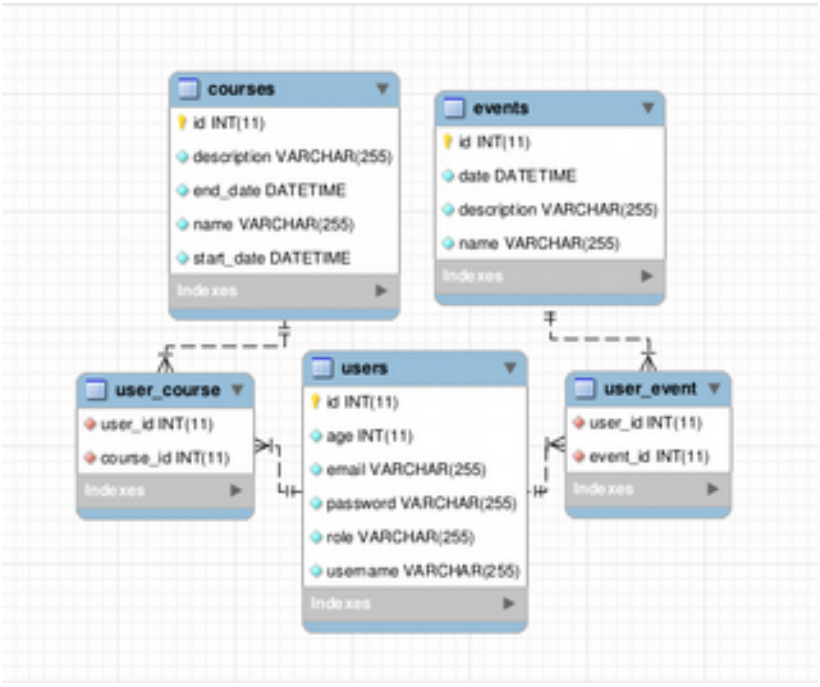


1.2 Class Design

The most important patterns used in this project's creation are the Command, Observer and Factory. The Command pattern is used to add functionality to the UI's buttons, the Observer pattern is useful for displaying data onto the UI, while the Factory pattern is used to create a connection of the back-end to the database. A conceptual class diagram can be seen below:



2. Data Model



The presented system relies on 3 data models: the User data, Course data and the Event data. (+ the

auxiliary connector tables). The User data encapsulates a user's username, email address and password, while the Event data model does so about the Event Credentials.

3. Unit Testing

The unit testing should consists of gradually testing small parts of the application such as **user connectivity** (log in, sing in), **uploading events** such as courses or **offers.** Each Service class will be tested individually (ideally by the use of Junit test & the dependency injections will be created using a mocking framework). A Junit test case is characterized by a known input and an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

Example:

step 1: create a user with specific credentials e.g. id = 34;
step 2: create a climbing course with id = 18;
step 3: test the enrollment feature by making the new user enroll to the new course;
step 4: check whether the new user has been truly enrolled to the course and whether the course now contains the new user as a participant;

IV. Bibliography

https://www.thymeleaf.org/doc/tutorials/2.1/thymeleafspring.html

http://www.baeldung.com/spring-email

https://spring.io/guides/gs/serving-web-content/#scratch