Online Betting Platform

**-Project-**

**SDI, 244**

**Bendea Paula-Alexandra**

**Stefanie Andrei-Petru**

# Business Definition

Platform for a sports betting business. There will be two types of users.

One is the end-user who can perform the following actions:

* Visit the website (visitor or registered user)
* Register (create a new account)
* Deposit or withdraw money
* Make bets on the available sports events (create tickets)
* Receive notifications based on the outcome of his tickets

And the other type is the administrator/manager who will be able to:

* See end-user history
* CRUD on sports events

The application will be created using the following technologies:

* MySQL for the database (might change)
* Spring for back-end (DAO and service dispatcher) – Tomcat server
* AngularJS for front-end (one-page application)
  + Supports data binding
  + One-page application
  + AJAX calls
  + Modularization

# Use-Cases

## Manage betting offer

Level: summary level

Primary actor: Website manager

Main success scenario: The manager has to login and perform a set of actions (add, remove or change the existing offer data. If everything is performed right, the end-user should be able to make bets using the new offer

Extensions: In case the data introduced by the manager is not valid it simply will not be displayed to the end-user

## Make a bet

Level: user-goal level

Primary actor: Registered User

Main success scenario: After deciding the bets he wants to make and allocating the desired amount of money, the user can send the request to the platform receiving a ticket containing the bets and the financial details

Extensions: The user might wait too long to make a bet and the matches he chose might become unavailable. In this case he will simply receive an error message.

## Register

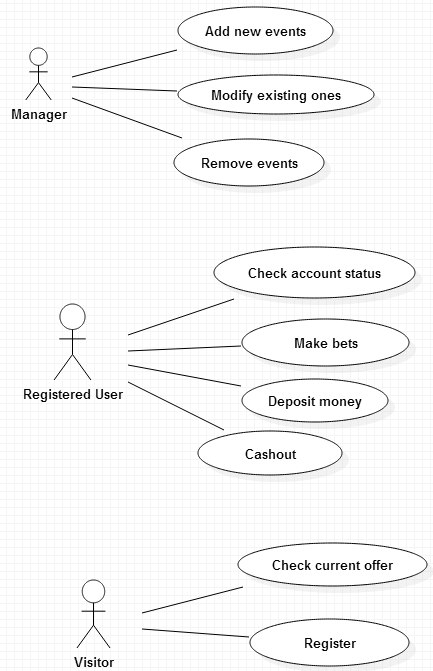
Level: user-goal level

Primary actor: Visitor

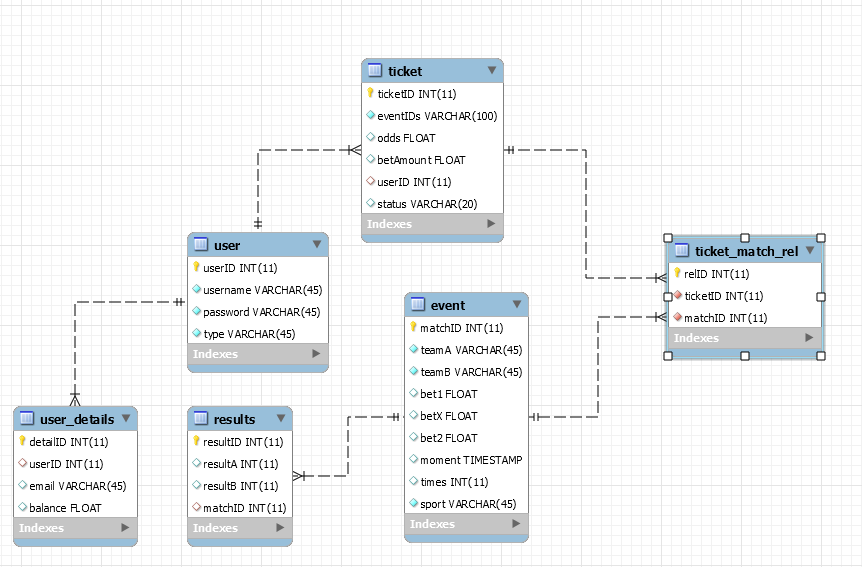
Main success scenario: If a visitor (person that is not logged in) enters the site, he will still be able to see the current offer, but he will not be able to make any bets. However, he can access the registration page where he can create an account.

Extensions: The data user provided must be valid and he must be 18 years or older otherwise he will not be able to create an account.

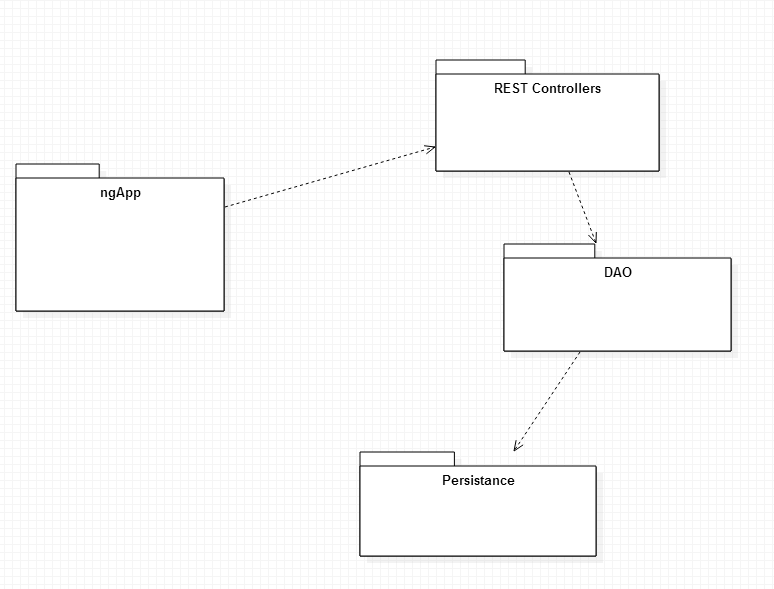
## Diagrams



# Data Model



# Architecture



## Web App

Written in Angular, deployed as a static website (only HTML, JS, and CSS files). It is the client application that consumes the backend REST API through HTTP.

### Main Pages

* Client Login/register - <http://localhost/LunarBet/#/login>

A picture containing background pattern

Description automatically generated

* Manager/admin login - <http://localhost/LunarBet/#/loginManager>
* Current offer per sport

Graphical user interface, application

Description automatically generated

* Results page

Table

Description automatically generated

* Admin dashboard

Chart, bar chart

Description automatically generated

## Backend

The backend in built with Spring (MVC, Security) and exposes a RESTful interface. This is composed of a set of controllers (REST controllers) that handle incoming requests. It also leverages Hibernate as an ORM for persisting the data into a MySQL RDMS.

### Directory Structure

Text

Description automatically generated

### Controllers

**Controllers** are responsible for intercepting client request (coming from the web app) and processing them. They are Spring MVC Rest Controllers. The most notable controllers are:

* **AuthenticationController** – responsible for registration (/api/register) and authentication (/api/authenticate). It works based on JWT tokens.
* **EventController** – responsible for handling event-related requests such as adding new events (/api/events/generate) by an admin or fetching the current offer/events (/events?sport=<sport>) for the clients.
* **BetController** – responsible for storing client-created tickets (/api/bet/add)
* **ResultController** – responsible for updating the results of the finished events (/api/result/generate) and fetching them when requested by the clients (/api/result/all)

### ORM

The persistence is handled through **Hibernate**. The entity classes together with the relationships between them are defined in config files in *src/main/resources/hbm*.

Text

Description automatically generated

The project makes use multiple relationship types such as **one-to-many** for storing the user details and bets, and **many-to-many** for storing the bet results – one event result affects multiple tickets, while one ticket might contain multiple events.

The ORM session is handled by the Spring IoC container by registering the factory method, *getSession*, as a @Bean.

Text

Description automatically generated

The session is then injected by Spring in all classes such as controllers that need the Hibernate session through the @Autowired annotation.

Text

Description automatically generated

### Build

The application is built using Maven which also manages its dependencies. To build, package, and deploy the application, run *mvn clean install* – the deployment location is configured in *pom.xml*.

### Tests

The system is covered by the unit tests defined in *src/test/java*. These ensure that the main flows of the application work as expected.

# Deployment

## Requirements

* Static website hosting such as Apache or nginx
  + XAMPP or WAMP can be used on Windows
* MySQL 5.7
  + XAMPP, WAMP, or Docker can be used
* Servlet container
  + E.g. Tomcat

# Contributions

* Andrei Stefanie –Hibernate setup, deployment
* Paula Bendea – User interface
* Pair-programming – REST Controllers, communication between applications, architecture, data model, documentation.

# References

* Spring Documentation <https://docs.spring.io/spring-framework/docs/current/reference/html/>
* Hibernate Documentation <https://hibernate.org/orm/documentation/5.6/>
* Angular Documentation <https://angular.io/docs>
* Spring Response Handling <https://www.baeldung.com/spring-response-entity>
* Maven <https://maven.apache.org/guides/>