Gambling web application

Tobias Adrian,
Jacob Bergfalk,
Erik Ekström,
Hugo Liss-Daniels

Web applications / DAT076

Chalmers tekniska högskola

Introduction

The application is a gambling website meant to allow users to invest and gamble their virtual currency in the featured games available on the site. For now the games included are a coin flip game and a stock market predictor game. The site is in its early development state and does not offer any real money gambling. However, the user can invest virtual money to test all the games on the site.

The purpose of this website is for educational purposes and although there are currently no plans for the site to be a full on working gambling website, with hundreds of games for the users to play there is still room for that to be the case in the future with further development. We would like to allow users to experience gambling without the need to invest their money to the site, while also keeping them notified on how dangerous gambling is and to only gamble the money you can afford to lose.

https://github.com/JacobBergfalk/PointsAndLeaderboard

User Manual

Installation

To install and run the program you need to download the project file, as well to make sure you have the following programs installed:
@testing-library/dom
@testing-library/jest-dom
@testing-library/react
@types/express
@types/jest
@types/mocha
@types/node
@types/react-dom
@types/react
@types/supertest
bcrypt
chart.js
dotenv
express-session
express
jest
pg-hstore
pg
react-chartjs-2
react-dom
react-router-dom
sequelize
supertest
ts-jest
ts-node-dev
typescript

npm install @testing-library/dom @testing-library/jest-dom @testing-library/react @types/express @types/jest @types/mocha @types/node @types/react-dom @types/react @types/supertest bcrypt chart.js dotenv express-session express jest pg-hstore pg react-chartjs-2 react-dom react-router-dom sequelize supertest ts-jest ts-node-dev typescript

Additionally, to run correctly with a database you need to create a database with the docker desktop application. In the database you also need to create the following table:

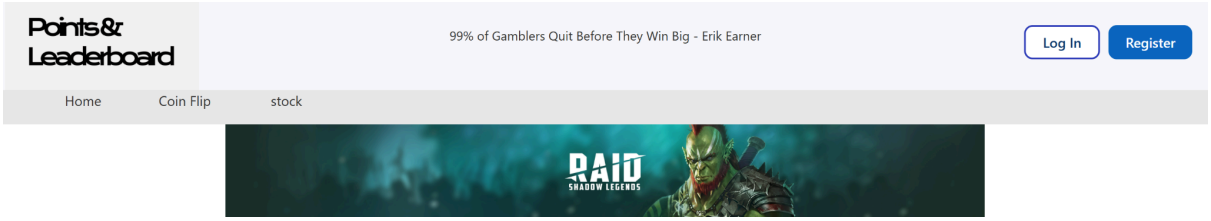
CREATE TABLE users
username TEXT PRIMARY KEY,
password TEXT NOT NULL,
balance INT NOT NULL

How to run the program

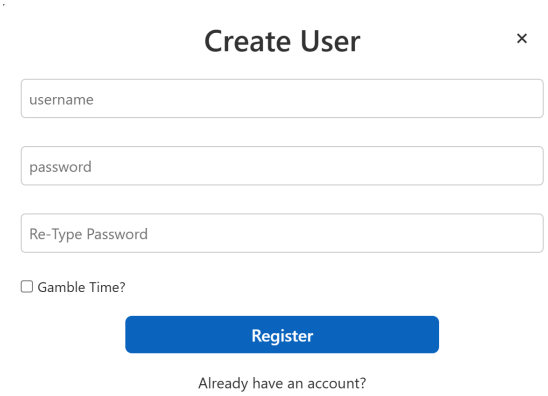
1. Start the docker database server
2. Open a terminal in the PointsAndLeaderboard/Server directory and run the command: npm run dev
3. Open a terminal in the PointsAndLeaderboard/Client directory and run the command: npm run dev
4. Click the localhost link in the client terminal and a window in your browser should appear.

Navigation of the site

When you are on the site there is a navigation bar where you can navigate through the home screen, the Coin Flip game and the stock predictor Game.



In order to play the games and gamble, you first need to create an account by clicking the register button in the top right corner of the page. You will get this pop up panel:



Enter a username and password. Make sure to check the “Gamble Time?” checkmark and press the Register button

After creating your account you will see that the Login and Register buttons have been replaced by a Profile button. By pressing the profile button you will go to the user page where the user can earn the virtual currency free of charge as well as log out the user.

The website has 2 games in which the user can partake by using the navigation bar.

1. The coin flip game. Enter the amount of currency to wager then press the vinn pengar knappen and if you win you get double the amount. However if you lose the coinflip you will lose all the money you wagered.
2. The Stock chart game. The user will see a chart that is moving. The user can then choose the amount to wager and predict if the stocks will rise with the Long button or decrease with the Short button. This game is purposely rigged against the player.

Design

Frontend components

Components

Header: This is the website's main header and it includes the navigation and the user authentication controls. It displays the site logo with a link to the homepage and rotates quotes in the center. To the right it shows a profile button if the user is logged in and if the user is not logged in it shows a log in and a register button. It manages registrationOpen and loginOpen which controls the visibility of the registration and login button.

Navbar: The navigation bar is the bar under the header where there are three buttons to navigate tha application: Home, Coin Flip and Game.

Sidebar: This is an area on the right side of the screen where there is an ad to another gambling site.

QuoteRotator: This is a method implemented by the header to show different “motivational quotes” in the middle of the header. It manages the states “index” that determines which quote should be displayed and “fade” which controls the fade-in and fade-out of the quotes.

Salespitch: This is an element at the bottom of the application where we have three reasons why this is a good site.

TopLeaderboard: This displays the top five players with the highest balance in the game. It fetches the users data from the database and sorts the players in descending order, the house is a player that has a minimum of 1200 coins but if someone has more than that it adds so that the house always has 100 coins more than the highest user balance. It manages two states and those are Player[] which stores the list of players both the one fetched from the database and local players. And balance which is the logged in users balance.

Assets

AuthContext: This component manages user authentication by storing and providing authentication state and related functions. It keeps track of their username and whether a user is logged in or not. When the app loads it automatically checks the login status by making a request to the backend. It provides functions for logging in, registering, and logging out and updates the authentication state accordingly. The component manages two states: loggedIn which indicates if a user is authenticated, and username which stores the username. These states are updated based on responses from the backend to ensure accurate authentication status across the application.

Pages

Coinflip: This component allows users to play a simple coin flip game by betting coins. It requires the user to be logged in and fetches their balance from the backend. The game sends a request to the backend with the chosen bet amount and updates the balance based on the outcome. If the user wins a “thumbs up” image is displayed and if they lose a “laughing cat” image is displayed. After 0.5 seconds the game resets to its default state. The component manages three states: balance which stores

the user's current coin balance, betAmount which tracks the amount the user wants to bet and imageSrc, which updates based on the game result.

Index: This is the home page for this application and it contains a section for the most popular games, recently played games and the leaderboard for all players. It also maintains two arrays, games which lists all games with an individual image and latestGames which shows the latest games you played.

Profile: This component displays the logged-in user's username and balance while providing options to manage their account. It fetches the user's balance from the backend and updates it. The user can "Invest" to add 100 coins to their balance or choose to log out, which redirects them to the home page. It manages two state variables: balance which tracks the user's current balance and error which stores error messages if any requests fail. The component also includes a placeholder for account deletion functionality.

StockChart: This component simulates a stock trading game by displaying a "live" stock price. It has two options, short which makes the price go up and long which makes the price go down. This basically simulates the worst thing that could happen when doing a trade so you lose money either way. The component manages several states, including: prices which is an array that holds the simulated stock prices over time, timestamps, which is an array tracking the time for each stock price update, tradePrice and tradeType which tracks the price at which the user entered a trade and whether the trade is "long" or "short", betAmount that is the amount the user bets on the trade, currentProfitLoss which calculates the current profit or loss based on the difference between the entry price and the latest price.

Backend API

Design

The backend API is designed using a layered architecture, ensuring a clear separation of responsibility and maintainability. Each Layer has a well defined responsibility and consists of three layers:

1. Router Layer - Handles the incoming HTTP requests, validates input and forwards the "computing" of the request to the Service Layer.
2. Service Layer - Implements and processes the logic of the requests, ensures security measures and interacts with the model layer.
3. Model Layer - Directly interacts with the database to retrieve and update information regarding the user.

Router Layer

The router layer acts as a gateway to the backend, forwarding authenticated and validated requests to the service layer. It is only responsible for defining API endpoints and handling HTTP requests and does not perform any logic nor does it have any direct contact with the database.

Another security measure is to not allow the router layer to have any access to the session data, but instead forwards the calls to the service layer after the requests have been authenticated and validated.

Service Layer

The service layer encapsulates the operations that we can perform upon the entities in the model and acts like a middleman between the router and model layer. This setup ensures that the application remains modular and secure. In our case the Service layers handle logic such as user authentication and session handling.

Model Layer

The model layer is responsible for the objects and database interaction. It retrieves and updates data based on the requests it retrieves from the service layer. In our case the Model layer is responsible for the Coin Flip game, creating a new user and handling database interactions.

User Authentication

The user authentication manages user access by handling registration, login, session management and authentication verification. It follows a session-based model, allowing users to remain logged in until they manually log out or their session expires.

When a user attempts to log in or register, the Router layer processes the request from the frontend and forwards the request to the Service Layer after validating the input data. The Service layer checks whether a user is already logged in. When the user is logging in, the entered password is compared against the hashed version which is stored in the database. If valid, a session is created. For registrations, the Service layer ensures that the username does not already exist in the database before hashing the password and storing it in the database. Once the user is successfully registered, a session is created, allowing the user to access the website.

All subsequent requests to protected routes undergo authentication, ensuring only logged in users can access certain functionalities. After the user is logged-in going through protected routes are verified to make sure the request is authenticated.

Coin Flip

The coin flip game is responsible for processing bets, determining the outcome and updating the users balance accordingly. Users cannot place bets exceeding their balance and once a bet is placed the outcome is final and quick. The game logic is independent from external factors and the only state operation is the balance. This guarantees that the results are determined instantly, providing immediate feedback for the user. The only persistent change is the adjustment of the users credits.

The game mostly uses the API endpoint the “/coinflip” POST request which handles the request.

Trading Simulator

The simulator allows the users to engage in simulated financial trading, where it is possible to open and close both long and short positions based on the stock price movements. Users place a trade at a specific entry price, and their profit or loss is determined when the user closes the position. The game

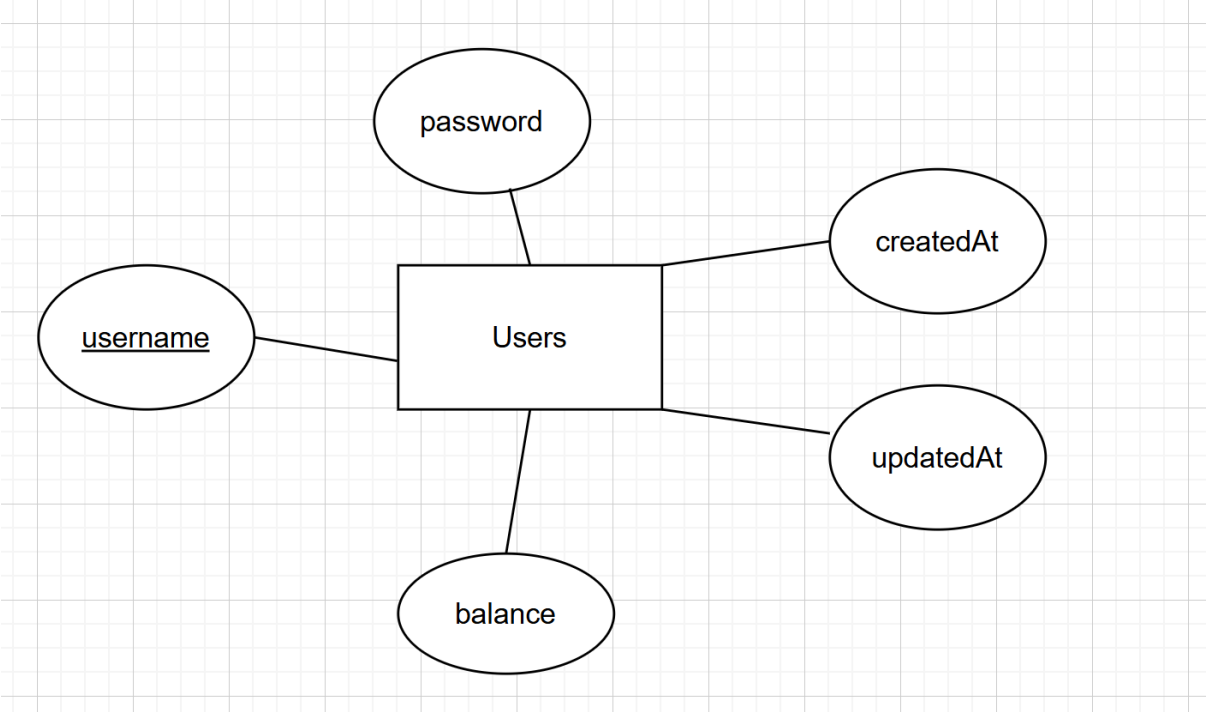
ensures that trades do not exceed the current balance of the user. The system mostly uses two POST requests - **“/stock/trade”** for opening a trade and **“/stock/close”** for closing the trade.

Before opening a trade, the Service layer verifies that the user does not currently have an active trade and confirms that the user's balance is sufficient to cover the bet amount. If these conditions are met, the trade is executed and the users balance is adjusted accordingly. When closing a trade, the Service layer retrieves the trade details, calculates a profit or loss, and updates the users balanse to reflect the final outcome.

List of accepted requests

Request	Body	Valid response	Invalid response
POST /balance/add	Amount: Int	Sends back the new balance	“Invalid Amount” or “Failed to add Credits”
POST /coinflip	Choice: “Heads” Bet Amount: Int	Bet amount, Credits won or lost, win, choice	“Invalid request parameters”, “invalid choice”, “database error”
POST /login	username: string, password: string	“Login successful”	“Requires Username and password”, “Invalid credentials”, “Login error”
POST /logout	req: Request	“Logout successful”	“Logout error”: error, “Server Error”
POST /register	username: string, password: string	“User registered”	“Requires Username and password”, “Username already exists”, “Database error”, “Registration Error”
POST /stock/trade	tradeType: “long” or “short”, betAmount: int, entryPrice: int	res.json(result)	“Invalid Request”, “Stock trade Error”, “Server Error”
POST /stock/close	tradeType: “long” or “short”, betAmount: Int, entryPrice: Int, exitPrice: Int	res.json(result)	“Invalid Request”,”Server Error”
GET /session		loggedIn: True or False, username: currentUser	“User not logged in”, “Server error”, “Session check error:” error
GET /users		returns the user in the database	“Error fetching users”

ER-Diagram



Responsibilities

We used a flexible approach throughout development. This is not the first time we have worked on course projects together and are good friends outside of school. Therefore we are comfortable with being less strict with each person's responsibilities due to our previous experiences with each other. We used Git to manage and merge our contributions and everyone pitched in where needed and helped each other learn.

Typically Erik, Hugo and Jacob worked in a group of three starting each of the labs. This is because they work on the same bachelor thesis which made collaborations in person the main way of working together. Tobias, would either later the same day or the following day get informed of the progress of the project and make his own contributions. In retrospect certain parts of the project may not have needed three different people due to its linearity but for certain other parts working in a group of three was beneficial due to the complex aspects of the project.

Nearing the end of the course Jacob took a bigger responsibility in ensuring that the project reached its final state. Throughout the course Jacob, due to some previous basic experiences with website development and willingness to spend more hours than the rest of the group made the most contributions. Tobias also made an impressive amount of contributions considering he for the most part worked unaccompanied. That is not to say that Erik or Hugo did not contribute to the development of the project but a natural consequence of differences in proficiencies.