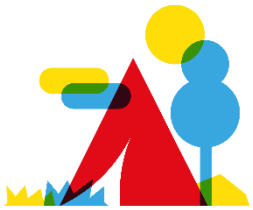

DES @ CSW Summer Internship

Client Application for Stock Market Simulator Game

Introduction

We're going to create a fun and interactive Stock Market Simulator game. You'll be using your favorite frontend framework to build this. Exciting, isn't it?

Team Formation

First things first, let's form teams. Pair up with 3 or 4 of your classmates. Remember, teamwork makes the dream work!

Understanding and Planning

Now, let's understand what we're building. We'll need a cool user interface and some neat features. Let's brainstorm and jot down our ideas.

Setting Up

Next, we'll set up our development environment. This is where we'll be building our awesome application.

Developing the User Interface

Time to bring our ideas to life! Let's start building the user interface. Remember, a good user interface is key to a great user experience.

Integrating with the Backend

Once we have our user interface, we'll connect it with the backend server. This will allow our application to perform actions like viewing portfolio, buying/selling stocks.

Remember, the goal of this project is not only to create a functional application but also to learn about teamwork, planning, and the software development process. Let's have fun and learn together! 😊

Game rules:

This is a round-based game where the main objective is to buy or sell the stock options that are available to the players to increase the player's net worth.

When the game starts, the players have all the same amount of available money (10000$) and are presented with a set of stock options that they can buy in each round.

where in each round, the player has a limited time (30 seconds) to register transactions.

When the round ends, all transactions are executed, and the player's net worth is updated with the values that were calculated.

The game ends after 10 rounds.

The winner of the game is the player that has the highest net worth of all.

Functional requirements:

Create a client application (web or other), that connects to a server where the game runs.

The connection to the server is made by two methods:

- **REST API** – Where the client application registers the players and gets some of the information.
- **WebSocket** – The main communication channel where the client handles real time bidirectional communication with the server to play the game.

The player should launch the Client application (or website) and should be presented with an input field to insert its name and register to play.

If a game is ongoing, he must wait for the game to end.

When the game starts, the client should allow the player to play the game by the rules above.

The players can use the chat to communicate between them or with all the players in the game.

The group may choose the programming language that they are more comfortable with.

Note: we suggest that it should be a web-based language (React, Angular, Vue, Astro ... etc) because we as frontend developers may help you more with those 😊

Planning:

Day 1 - Team Formation, Understanding, and Planning

In the morning, form your teams and spend some time understanding the project. Discuss your ideas and what you want to achieve with your application.

In the afternoon, plan out the features and user interface of your application. Sketch out how the game will look and how the user will interact with it.

Days 2 and 3 - Setting Up and Starting Development

In the morning, set up your development environment and start building the user interface based on your sketches.

In the afternoon, continue developing the user interface. Start integrating with the backend server to perform actions like viewing portfolio, buying/selling stocks.

Day 4 - Testing and Finalizing

Spend the morning testing your application. Look for any bugs and make improvements based on feedback and finalize the application.

API Specifications

REST API

For the REST API, the existent endpoints are:

Method	Endpoint	Payload	Response	Description
POST	**game/register-player**	PlayerDTO with only the name property filled.	PlayerDTO with the name and ID	It accepts a PlayerDTO object type with the name of the player and returns the unique identifier of the player that will be used for all other operations
GET	**game/player/{id}/wallet**	The ID of the player	PlayerDTO with the wallet and netWorth filled	Gets the current wallet of the player for the provided unique identifier.
POST	**game/leave**	PlayerDTO with the ID property filled.	N/A	Allows the player to leave the game. It accepts a PlayerDTO with the ID on registration.

WebSockets

Upon establishing connection, the following messages can be exchanged:

Direction	Message	Payload	Response	Description
server > client	**connected**	PlayerDTO	N/A	when a player joins the game.
server > client	**disconnected**	PlayerDTO	N/A	when a player leaves the game.
server > client	**game-started**	**Null**	N/A	When a game starts.
server > client	**game-ended**	PlayerDTO[]	N/A	
server > client	**round-started**	StockDTO[]	N/A	
server > client	**round-ended**	**null**	N/A	
client > server	**Transaction**	TransactionDTO	A string with the transaction operation result. (success or an error message)	When the player sends a transaction within a round.

client > server	**connect-confirm**	PlayerDTO	N/A	Required to link the playerID to the socket. Send after login
client > server	**enter-chat**	PlayerDTO	N/A	When a player wants to join the chat
client > server	**leave-chat**	PlayerDTO	N/A	When a player wants to join the chat
client > server	**send-message**	ChatMessageDTO	N/A	When a player wants to send a message to the chat group or another player
server > client	**chat-message**	ChatMessageDTO	N/A	A chat message sent by another player

DTO Specifications

PlayerDTO

- **id<string>:** The GUID of the player
- **name<string>:** The name of the player
- **netWorth<number>:** The calculated sum of the wallet
- **wallet<StockDTO[]>:** An array of the stocks that are in the user's wallet
- **chatMessages<ChatMessageDTO[]>:** An array of chat messages

StockDTO

- **id<string>:** The GUID of the stock option
- **company<string>:** The name of the company
- **description<string>:** The description of the company
- **symbol<string>:** The symbol of the stock option
- **price<string>:** The price of the stock

TransactionDTO

- **playerId<string>:** The GUID of the player that is transacting
- **stockId<string>:** The GUID of the stock that is being transacted
- **quantity<number>:** The quantity of the transacted stock

ChatMessageDTO

- **senderId<string>:** The GUID of the player that is sending the message
- **receiverId<string>:** The GUID of the stock that is receiving the message (should be "all" when sending to all players)
- **message<string>:** The quantity of the transacted stock
- **datetime<datetime>:** The date and time of the message (filled by the server)