Problem Statement:

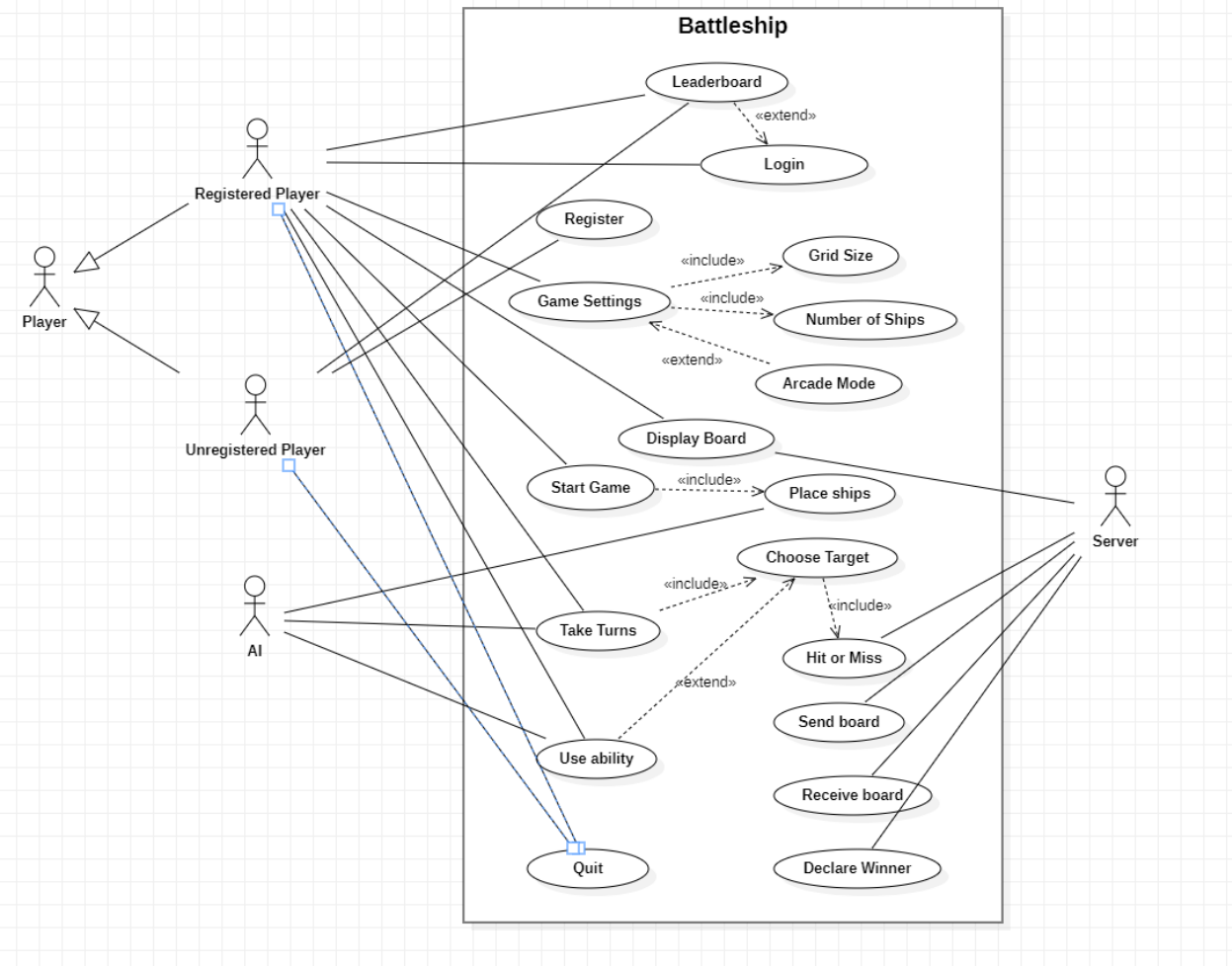
Our project involves creating a game inspired by the popular board game Battleship. This battleship with a login/ register page is created with a user-friendly and engaging user interface that allows the players to choose their ships and place them on a default board. The login / register page is connected to a MongoDB backend which can be monitored with MongoCompass.

The game will then send the coordinates of the ships to the backend where an AI player will place their ships on another board. The objective of the game is to correctly guess where the opponent's ships are located on the default board by attacking positions, with each attack resulting in either a red or blue highlight on the board. Blue indicating a miss and red indicating a hit.

The game ends with a victory to the player if they manage to destroy all five ships of the AI. The game ends with a defeat if the AI manages to destroy your ships fire and a winner is declared.

The key challenges in designing this game includes creating an intuitive and responsive interface for placing the ships, implementing an efficient algorithm for the AI player's ship placement and developing an accurate and fair system for determining the winner.

I. Use Case Diagram:



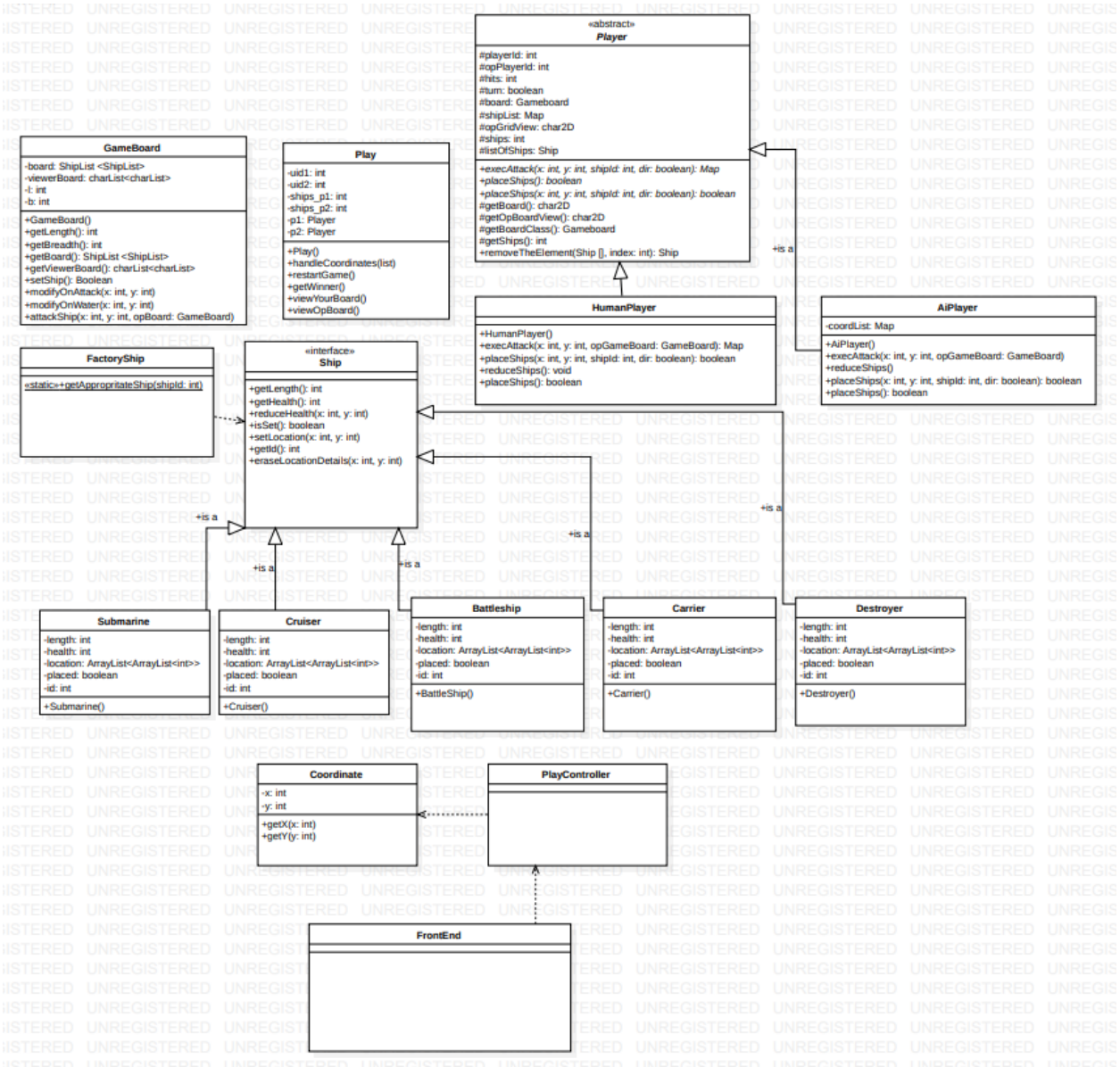
Use Cases (major):

1. Player Registration: this use case involves allowing new players to register for the game. The player provides their email address and creates a unique username and password.
2. Player Login: this use case involves allowing registered players to log in to the game. The player provides their username and password.
3. Ship Placement: this use case involves allowing the player to select and place their ships on the default board. The player selects the type of ship they want to place and then clicks on the desired location on the board to place it.
4. AI Ship Placement: this use case involves generating a random and fair placement of the AI's ships on the other board.

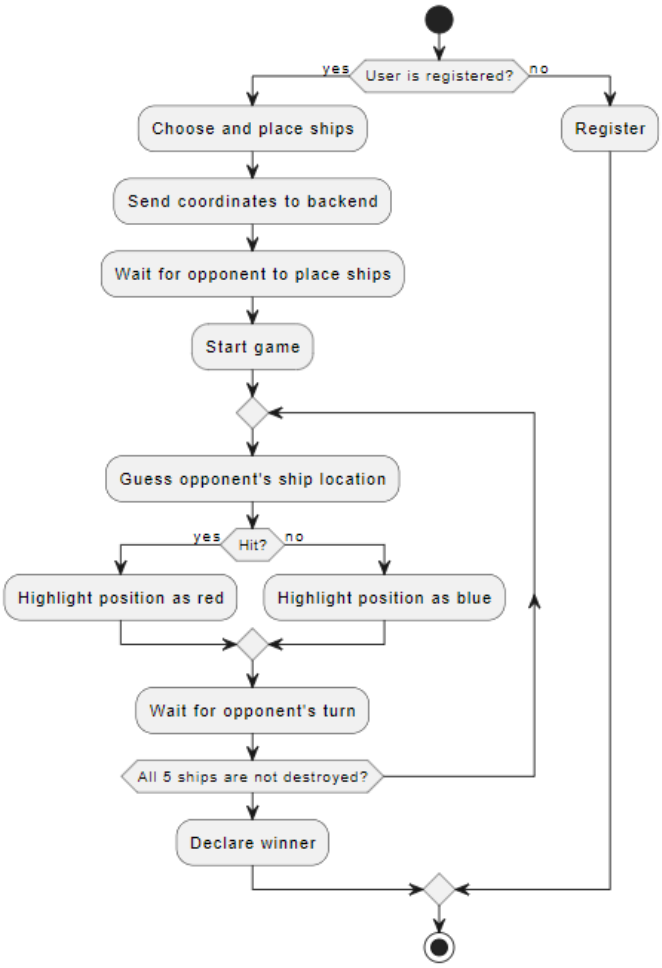
Use Cases (minor):

1. Reset Game: this uses case involves the game resetting to its initial state once the game has ended.
2. Attack: the use case involves allowing the player to select a position on the board to attack. The game checks whether there is a ship on that position, and if so, marks it as a hit. Otherwise, the game marks it as a miss.
3. Winning Condition: this use case involves checking whether the game has ended and determining the winner. If the player destroys all of the AI's ships, the player wins. If the AI destroys all of the player's ships, the player loses.
4. Backend Integration: connect the login / registration page to a MongoDB backend and monitor it using MongoCompass, which involves setting up the necessary APIs and database schemas.

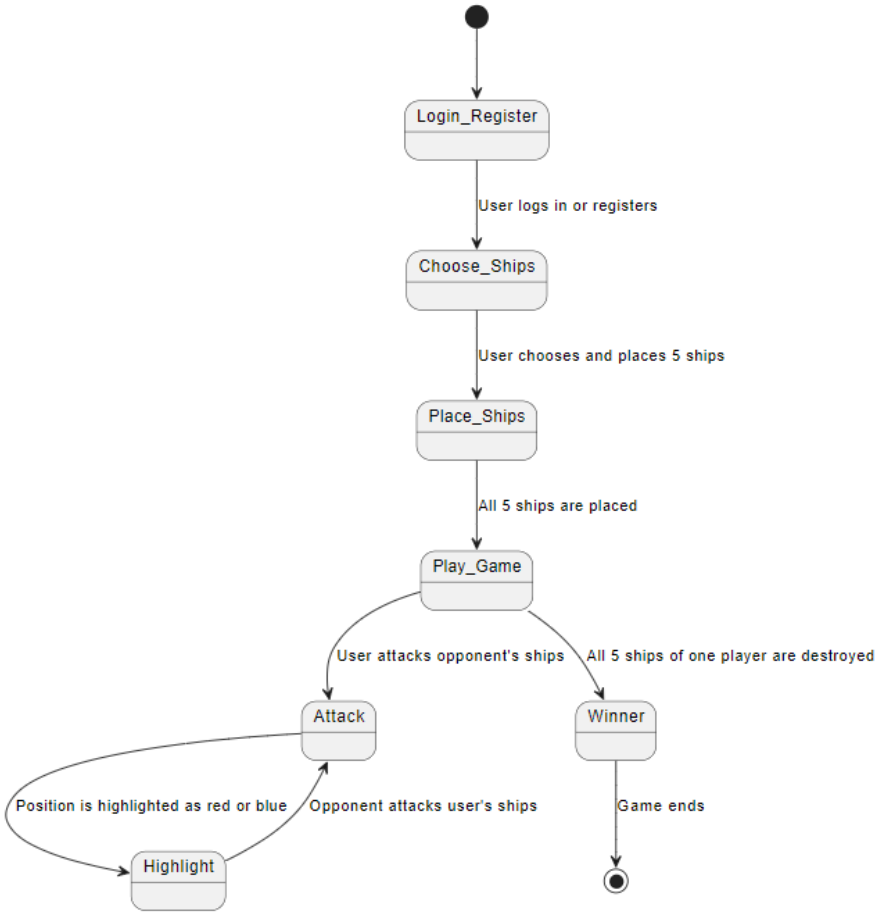
II. Class Diagram:



III. Activity Diagram:



IV. State Diagram:



V. Design Patterns used:

- Factory Pattern: for instantiation of ship. There is a FactoryShip which has a static method to return a ship object based on ID
- Command Pattern: passing data from frontend to backend. Once the ships are placed, the coordinates and ID of ships are packages in an object and sent to the backend (java) for further processing
- Chain of Responsibility: for accurate message passing and SRP based execution of different operations. The request to attack a coordinate comes from the frontend and is passed to the Controller, then to the HumanPlayer class then to Gameboard class finally
- Templates and Iterators: used for storing the coordinates, locations of ships and other similar details. To iterate over them, since they are associative; non linear data structures, iterators are used.
- Singleton: springboot application class is a singleton class by default.

VI. Screenshots:


localhost:3000
BATTLESHIP GAME
Start Playing
BATTLESHIP GAME
Ships
Destroyer (2)
Submarine (3)
Cruiser (3)