**High-Level Design Document**

Table of Contents

[1. Overview 2](#_Toc179294364)

[2. Architecture 2](#_Toc179294365)

[3. Components 2](#_Toc179294366)

[4. Game Flow 3](#_Toc179294367)

[5. API Endpoints 3](#_Toc179294368)

[6. Technology Stack 3](#_Toc179294369)

**High-Level Design Document**

# 1. Overview

The Battleship Game solution consists of a **backend** built using **C# with ASP.NET Core** and a **frontend** built using **React**. The game allows a single player to play against the computer on a 10x10 grid, where ships are placed randomly, and the player can fire shots to sink the ships.

# 2. Architecture

The application follows a **client-server architecture**, where the **backend** handles game logic, and the **frontend** provides an interface for the user to interact with the game.

* **Backend**:
  + Built using **ASP.NET Core** to expose APIs.
  + Handles the game setup, random ship placement, shot validation, and response.
  + Ensures feedback (hit, miss, or sunk) is sent to the frontend.
  + Manages the overall state of the game.
* **Frontend**:
  + Built using **React**.
  + Displays the game grid and allows the user to click on cells to fire shots.
  + Sends shot requests to the backend via API and displays results visually.
  + Handles user input and game state updates based on backend responses.

# 3. Components

1. **Backend Components**:
   * **GameGrid.cs**:
     + Handles the creation of the game grid, ship placement, and shot validation.
     + Manages the grid's state and tracks ship positions and hits.
   * **Ship.cs**:
     + Represents each ship (Battleship, Destroyers) and tracks its position and hit status.
     + Checks if the ship has been sunk based on its hit points.
   * **GameController.cs**:
     + API controller that exposes endpoints for setting up the game, firing shots, and retrieving game status.
     + Integrates with the GameGrid to perform necessary operations based on user actions.
2. **Frontend Components**:
   * **App.js**:
     + The main React component that renders the game grid and handles user interactions (clicking cells).
     + Communicates with the backend API to fire shots and updates the UI accordingly.
   * **App.css**:
     + Defines styles for the grid, cells, and overall layout of the game interface.
     + Provides visual feedback on cell states (e.g., hit, miss).

# 4. Game Flow

1. **Game Initialization**:
   * When the game starts, the backend API (/setup-game) is called to initialize the game grid and randomly place the ships.
   * The ships are hidden from the user.
2. **User Interaction**:
   * The user clicks a cell on the grid to fire a shot (in the format "A5", for example).
   * The frontend sends the shot request via a POST request to the backend (/fire).
3. **Shot Validation**:
   * The backend checks if the shot hits a ship or is a miss.
   * The response is sent back to the frontend (hit, miss, or sunk).
4. **Game Status**:
   * After each shot, the backend checks if all ships have been sunk.
   * The game continues until all ships are destroyed, at which point the user is notified.

# 5. API Endpoints

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /api/game/setup-game | Sets up a new game with random ship placement. |
| POST | /api/game/fire | Accepts a coordinate and returns whether it is a hit, miss, or sunk. |
| GET | /api/game/status | Returns the current status of the game (ongoing, won, etc.). |

# 6. Technology Stack

* **Backend**:
  + ASP.NET Core (C#)
  + REST API
  + Random ship placement logic
  + Unit testing (optional)
* **Frontend**:
  + React (JavaScript)
  + Fetch API for communication with the backend
  + CSS for styling the grid and user interface

**7. Assumptions and Constraints**

* The game is for a single player versus the computer.
* Ships are placed randomly by the backend.
* The grid size is fixed at 10x10, and the ships have predefined sizes:
  + 1x Battleship (5 squares)
  + 2x Destroyers (4 squares each)
* The player continues firing until all ships are destroyed.

**8. Conclusion**

The design ensures that the game logic is encapsulated within the backend API while the frontend focuses on user interaction and grid visualization. The separation of concerns allows for flexibility and easier maintenance. Additionally, the REST API makes it possible to extend the game with more advanced UI or AI features in the future.