

# Project Report: Custom FizzBuzz Online Game Implementation

## 1. Introduction

The project involved designing and implementing a custom FizzBuzz online game. The game allows users to create their own versions of the FizzBuzz game by defining rules that replace numbers with specific words. The application follows a 3-tier architecture comprising a ReactJS front end, a .NET 8 Web API back end, and a PostgreSQL database, all containerized with Docker.

## 2. Requirements

The requirements for the project included:

- Users should be able to create custom FizzBuzz-like games with unique names.
- The game should replace numbers based on the rules defined by the user.
- A game session should serve random numbers to the player, and the player should respond with the correct word(s) based on the rules.
- The game should last for a user-defined duration, and no duplicate random numbers should be served within a session.
- The application should display the score at the end of the session.
- The solution should be containerized using Docker and include unit tests for both the front end and back end.

## 3. System Design

The system was designed with a 3-tier architecture, separating concerns into the following layers:

1. **Presentation Layer (Front End):** Implemented with ReactJS and TypeScript, responsible for user interaction, displaying games, and managing game sessions.
2. **Business Logic Layer (Back End):** A .NET 8 Web API responsible for managing game logic, validating rules, generating random numbers, and verifying player answers. This backend implementation was completely built based on repository design pattern to ensure an improvement for maintainability, testability, and flexibility of the codebase.
3. **Data Layer (Database):** PostgreSQL stores game data, including game rules, sessions, and player scores.

## 4. Implementation

### 4.1 Back-End Implementation

The back end was implemented using .NET 8 Web API. Key components included:

- **Database Models:**
  - **Game:** Stores the game name, author, and rules.
  - **Session:** Tracks active game sessions, player information, and scores.
  - **SessionNumber:** Logs numbers served in a session to ensure no duplicates.
- **API Endpoints:**
  - **POST /api/games:** Create a new game.
  - **GET /api/games:** Retrieve all games.
  - **GET /api/games/{gameId}:** Get a specific game by its ID.
  - **POST /api/sessions:** Start a new game session.
  - **GET /api/sessions/{sessionId}:** Get a specific session by its ID.
  - **GET /api/sessions/{sessionId}/next-number:** Get the next random number in a session.
  - **POST /api/sessions/{sessionId}/submit-answer:** Submit an answer and validate it against the game rules.
  - **POST /api/sessions/{sessionId}/end:** End a specific session by its ID.
  - **GET /api/sessions/{sessionId}/results:** Get the result of a specific session by its session ID.

### Games

POST /api/Games

GET /api/Games

GET /api/Games/{id}

### Sessions

POST /api/Sessions

GET /api/Sessions/{id}

GET /api/Sessions/{sessionId}/next-number

POST /api/Sessions/{sessionId}/submit-answer

POST /api/Sessions/{sessionId}/end

GET /api/Sessions/{sessionId}/results

- **Business Logic:**
  - Random number generation to avoid duplicates within a session.
  - Answer validation based on the custom rules defined by the user.
  - Persistence of game and session data in the PostgreSQL database.
- **Containerization:**
  - Dockerfile and `docker-compose.yml` were created to containerize the .NET API and PostgreSQL database.

## 4.2 Front-End Implementation

The front end was implemented using ReactJS with TypeScript. Key components included:

- **Home Component:**
  - Displays the home screen to the users where they can see the form to create a new game and the list of available games that they can choose to play.

# FizzBuzz Game

## Create a New Game

Game Name

Author

Min  Max

**Rules**

Number  Word

**ADD RULE**

**CREATE GAME**


## Available Games


- foobooloo
- fizzbuzz
- sitstand
- walkrun
- runhit
- joinleave
- punchkick
- eatdrink
- drinkeat

- **Game Component:**
  - Allows users to begin a session by entering their names and duration that they want to play the game.



## Start a Session

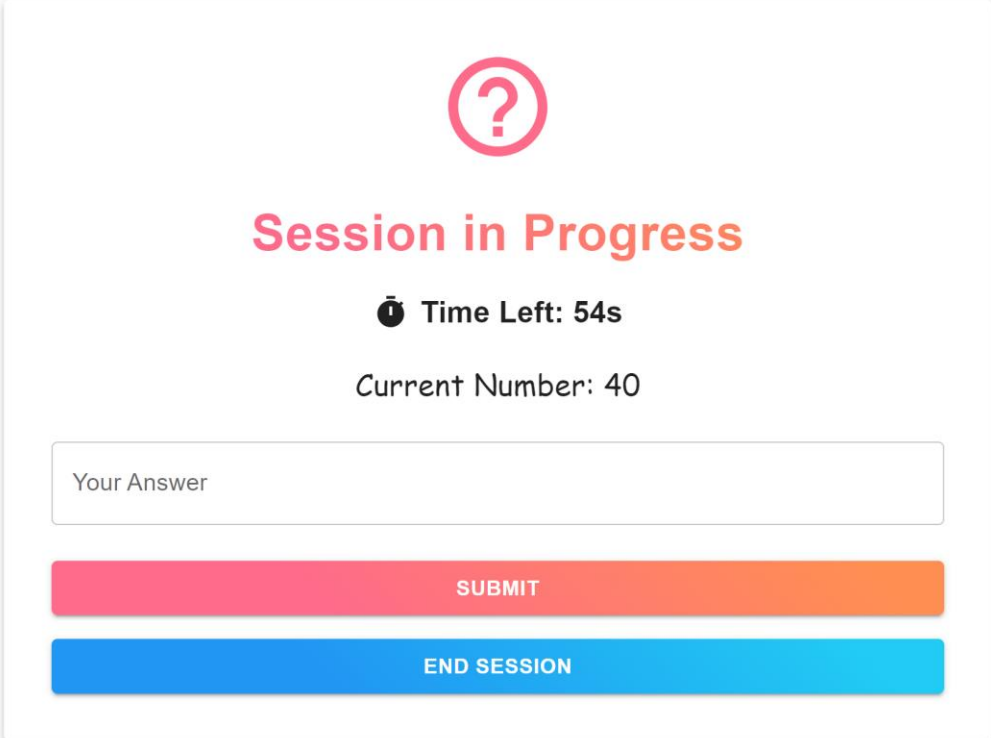




**START SESSION**

- **Session Component:**

- Showing up an in-progress session where the user answers the given question within the pre-defined duration. Alternatively, the user can choose to end the session early by clicking the button “END SESSION”.



The image shows a user interface for a 'Session in Progress'. At the top, there is a pink circular icon with a white question mark. Below this, the text 'Session in Progress' is displayed in a bold, pink font. Underneath, a timer icon (a clock face) is followed by the text 'Time Left: 54s'. Below the timer, the text 'Current Number: 40' is shown. A text input field with the placeholder 'Your Answer' is positioned below the current number. At the bottom, there are two prominent buttons: a pink button labeled 'SUBMIT' and a blue button labeled 'END SESSION'.

- **Result Component:**

- Generates the result of the most recent session after it is ended.

# Game Over!



Congratulations! You did a great job!



Your Score: 2/2

[BACK TO HOME](#)

- **GameForm Component:**
  - Allows users to create a new game by specifying the game name, author, range of numbers, and custom rules.

# Create a New Game

Game Name

fizzbuzz

Author

Andy

Min

0

Max

100

## Rules

Number

3

Word

fizz

ADD RULE

CREATE GAME

- Validates users' inputs to prevent invalid or inappropriate requests and align with the requirements.

# Create a New Game

Game Name

Game Name is required

Author

Author is required

Min

Min is required

Max

Max is required

## Rules

Number

Number is required

Word

Word is required

ADD RULE

CREATE GAME



# Create a New Game

Game Name

FizzBuzz

Author

Andy

Min

20

Max

15

Rules

Number

3

Word

Fizz

!

Value must be greater than or equal to 20.

ADD RULE

CREATE GAME

- **GameList Component:**
  - Displays a list of all available games, allowing users to select and start a game session.

# Available Games

foobooloo

fizzbuzz

- **State Management:**
  - Redux was used to manage the state across components, including the list of games, current session data, and player scores.
- **API Integration:**
  - Axios was used for making HTTP requests to the back-end API to create games, start sessions, and submit answers.
- **Routing:**
  - React Router was used to manage navigation between different pages, including the home page, game creation page, and session page.
- **Containerization:**
  - A Dockerfile was created to containerize the React application.

## 5. Testing

Unit tests were written for both the front end and back end to ensure the reliability of the application:

- **Back-End Testing:**
  - xUnit and Moq were used to test the .NET API's business logic and repository methods.
  - Tests covered scenarios such as game creation, session management, random number generation, and answer validation.

▲ Name	↕ Covered	↕ Uncovered	↕ Coverable	↕ Total	↕ Percentage	
— FooBooLooGameAPI	326	47	373	562	87.3%	<div><div></div></div>
DictionaryToJsonConverter	5	0	5	11	100%	<div><div></div></div>
FooBooLooGameAPI.Controllers.GamesController	34	0	34	50	100%	<div><div></div></div>
FooBooLooGameAPI.Controllers.SessionsController	59	0	59	81	100%	<div><div></div></div>
FooBooLooGameAPI.DTOs.CreateGameDto	5	0	5	12	100%	<div><div></div></div>
FooBooLooGameAPI.DTOs.SessionResultDto	6	0	6	14	100%	<div><div></div></div>
FooBooLooGameAPI.DTOs.StartSessionDto	3	0	3	11	100%	<div><div></div></div>
FooBooLooGameAPI.DTOs.SubmitAnswerDto	2	0	2	9	100%	<div><div></div></div>
FooBooLooGameAPI.Data.GameDbContext	40	0	40	33	100%	<div><div></div></div>
FooBooLooGameAPI.Entities.Game	7	0	7	14	100%	<div><div></div></div>
FooBooLooGameAPI.Entities.Session	9	0	9	16	100%	<div><div></div></div>
FooBooLooGameAPI.Entities.SessionNumber	5	0	5	12	100%	<div><div></div></div>
FooBooLooGameAPI.Helpers.DictionaryValueComparer	6	0	6	14	100%	<div><div></div></div>
FooBooLooGameAPI.Repositories.Implementations.GameRepository	21	0	21	40	100%	<div><div></div></div>
FooBooLooGameAPI.Repositories.Implementations.SessionRepository	82	0	82	109	100%	<div><div></div></div>
FooBooLooGameAPI.Services.GameService	14	0	14	31	100%	<div><div></div></div>
FooBooLooGameAPI.Services.SessionService	28	0	28	46	100%	<div><div></div></div>

- **Front-End Testing:**

- Jest and React Testing Library were used to test React components and their integration with Redux.
- Tests ensured that components correctly displayed data, handled user input, and made appropriate API calls.

```

Test Suites: 11 passed, 11 total
Tests:       55 passed, 55 total
Snapshots:   5 passed, 5 total
Time:        8.307 s
Ran all test suites.

```

## 6. Deployment

The application was fully containerized and can be deployed using Docker:

- **Docker Compose:** A `docker-compose.yml` file was provided to deploy the PostgreSQL database, .NET API, and React front-end application together.
- **Port Configuration:** The back-end API was configured to run on port 5000, and the front-end application was served on port 3000.

## 7. Conclusion

The project successfully met the requirements by providing a customizable FizzBuzz game that users can play online. The solution was implemented using modern web technologies and best practices, including a clear separation of concerns, state management with Redux, and containerization with Docker. The application is fully tested and can be easily deployed, making it a scalable and maintainable solution.

## 8. Future Enhancements

Possible future enhancements could include:

- **User Authentication:** Implementing user authentication to allow players to save their game progress and view their past scores.
- **Leaderboard:** Adding a leaderboard feature to display top scores globally or within specific games.
- **Enhanced Game Rules:** Allowing more complex game rules, such as combining multiple conditions with logical operators.

This project demonstrates a comprehensive approach to full-stack web application development, showcasing skills in both front-end and back-end technologies, as well as system design, testing, and deployment.