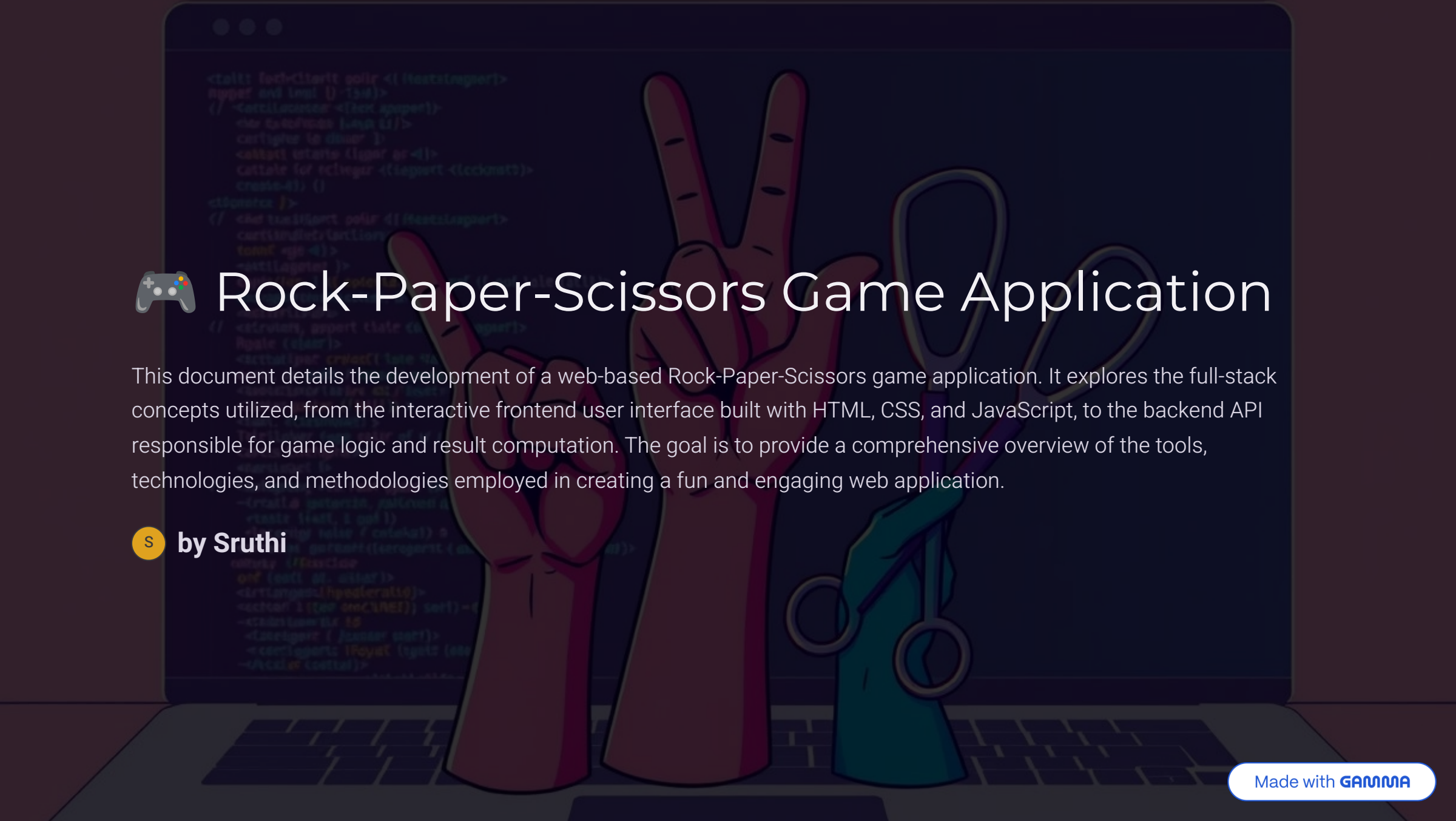


🎮 Rock-Paper-Scissors Game Application

This document details the development of a web-based Rock-Paper-Scissors game application. It explores the full-stack concepts utilized, from the interactive frontend user interface built with HTML, CSS, and JavaScript, to the backend API responsible for game logic and result computation. The goal is to provide a comprehensive overview of the tools, technologies, and methodologies employed in creating a fun and engaging web application.

S **by Sruthi**

Objective

The primary objective of this project was to develop a fully interactive Rock-Paper-Scissors game accessible via a web browser. Beyond just creating a playable game, a significant focus was placed on practical application of fundamental full-stack web development concepts. This involved:

- Frontend UI (HTML, CSS): Designing and implementing a visually appealing and intuitive user interface that allows players to make their choices and view game outcomes clearly.
- Game Logic and Interaction (JavaScript): Developing the client-side JavaScript code to manage user interactions, handle game states, and communicate with the backend.
- Server-Side Response (API Backend): Building a lightweight backend API to process game requests, determine the computer's move, and compute the game result, ensuring fair and consistent gameplay.

Tools & Technologies

The development of the Rock-Paper-Scissors game leveraged a combination of industry-standard web technologies, ensuring a robust and interactive application:

HTML5

Used for structuring the game's core elements, including player choice buttons, display areas for choices and results, and the "Play Again" button. HTML5 ensures semantic markup and accessibility.

CSS3

Applied for styling the entire application, creating a modern and engaging user experience. This includes layout, color schemes, responsive design, and interactive button effects.

JavaScript (Frontend)

The backbone of the client-side interactivity. JavaScript handles all game logic, including event listeners for button clicks, sending data to the backend, receiving results, and dynamically updating the UI.

Node.js (Backend)

Utilized for building a simple, yet effective, API server. Node.js processes incoming game requests, generates the computer's move, determines the winner, and sends the game results back to the frontend.

This combination allowed for a clear separation of concerns, with the frontend focusing on presentation and user interaction, and the backend handling the core game mechanics.

CSS Styling

The CSS styling for the Rock-Paper-Scissors game was designed to create a modern, engaging, and user-friendly experience. A dark theme was chosen to provide a sleek and contemporary look, enhancing readability and visual appeal. The interactive elements, particularly the choice buttons, feature vibrant colors and subtle hover effects to provide immediate visual feedback to the user, making the game more dynamic.

Highlights of the CSS implementation include:

- The `.game-container` is styled with rounded borders and a subtle shadow, giving it a soft, three-dimensional appearance that stands out on the page.
- The `.choice` buttons incorporate CSS transitions and hover animations, ensuring smooth visual changes when the user interacts with them.
- The `#game-result` element utilizes a standout yellow color to prominently display the outcome of each round, drawing the user's attention to the most critical information.
- The layout is also responsive, adapting gracefully to different screen sizes to ensure a consistent experience across desktops, tablets, and mobile devices.

Code snippet:

```
.choice:hover { background: #21a1f3; }
```

JavaScript Logic

The core game logic resides within the client-side JavaScript, responsible for managing user interactions and communicating with the backend. Upon page load, event listeners are attached to each of the choice buttons (Rock, Paper, Scissors). When a user clicks a button, the JavaScript captures their choice and constructs a POST request to the backend API. This request includes the user's selected move, serialized as JSON.

After sending the request, the frontend awaits a response from the backend. The expected response contains three key pieces of information: **userChoice** (confirming the player's move), **computerChoice** (the random move generated by the server), and **result** (declaring the winner or a tie). Once received, this data is dynamically displayed on the user interface, updating the respective elements to show both players' moves and the outcome of the round. To prevent multiple clicks during a single round, the choice buttons are temporarily disabled until the result is displayed, and then re-enabled as part of the "Play Again" functionality.

Code logic:

```
const response = await fetch(backendUrl, { method: 'POST', headers: {'Content-Type': 'application/json'}, body: JSON.stringify({ choice: userChoice }) });
```

Play Again Function

To enhance user experience and allow for continuous gameplay, a "Play Again" function is implemented, triggered by a dedicated button. When the user clicks this button, the game state is reset to its initial condition, preparing for a new round. Specifically, this function performs the following actions:

- Resets the displayed choices: The areas showing the player's and computer's previous moves are cleared or set to a default placeholder.
- Clears the result text: The message indicating "You win!", "You lose!", or "It's a tie!" is removed from the display.
- Re-enables choice buttons: If the choice buttons were disabled after a round, they are re-enabled to allow the player to make a new selection.

This functionality ensures a seamless flow, enabling players to enjoy multiple rounds without requiring a full page refresh, contributing to a more interactive and enjoyable gaming experience.

Functionality:

```
document.getElementById('play-again').addEventListener('click', function() { // Reset game state });
```

Backend Integration

The backend integration for the Rock-Paper-Scissors game is handled by a simple Node.js API, responsible for determining the game outcome. This API is accessible at the URL **http://localhost:3000/play**.

The backend listens for POST requests, which should contain a JSON payload specifying the user's choice. For example:

```
{ "choice": "rock" }
```

Upon receiving a request, the Node.js server performs the following actions:

- Validates the incoming user choice.
- Generates a random move for the computer (rock, paper, or scissors).
- Compares the user's choice with the computer's choice to determine the game result based on Rock-Paper-Scissors rules.

Finally, the backend constructs and sends a JSON response back to the frontend. This response includes all the necessary information for the client to update the UI:

```
{ "userChoice": "rock", "computerChoice": "scissors", "result": "You win!" }
```

This clear API contract ensures a smooth and efficient communication flow between the frontend and backend, allowing the server to focus solely on game logic without being burdened by UI rendering.

Features Completed

The development of the Rock-Paper-Scissors game successfully implemented several core features, providing a complete and interactive gaming experience:

User can choose a move

Interactive buttons allow the player to select Rock, Paper, or Scissors, serving as the primary input mechanism.

Random computer move generated

The backend intelligently generates a random move for the computer, ensuring fairness and unpredictability in each round.

Result displayed dynamically

After each round, the game dynamically updates the user interface to clearly show both players' choices and the outcome (win, lose, or tie).

Reset button for new game

A "Play Again" button allows users to quickly reset the game state and initiate new rounds without requiring a page refresh, enhancing continuous play.

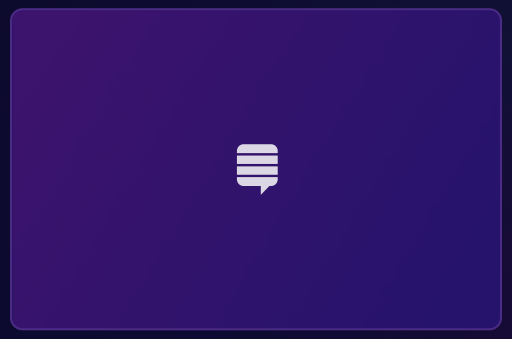
These completed features collectively deliver a functional and engaging Rock-Paper-Scissors web application.

What I Learned

Developing the Rock-Paper-Scissors game provided invaluable practical experience across various full-stack web development domains:

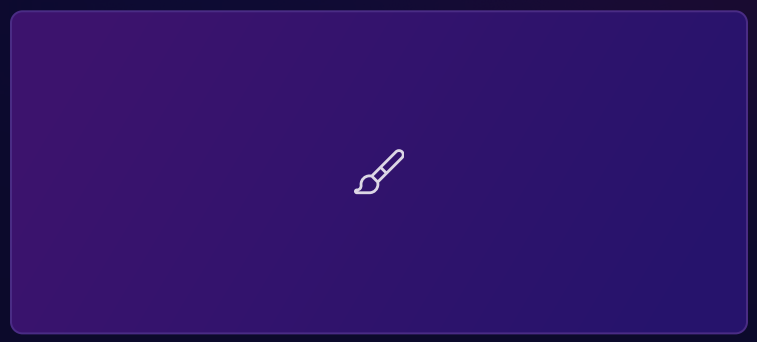
DOM manipulation and event handling in JavaScript

Gained proficiency in dynamically updating web content and responding to user interactions, which is fundamental for any interactive web application.



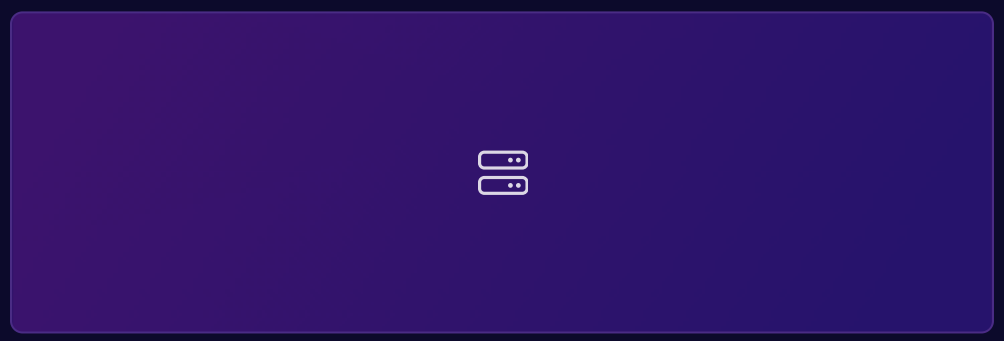
Sending and handling asynchronous API requests

Understood the importance of asynchronous operations for smooth user experiences, specifically how to send data to a server and process its response without freezing the UI.



Designing a clean and interactive UI

Applied principles of user experience design to create an intuitive and visually appealing interface, focusing on responsiveness and visual feedback.



Backend integration with game logic

Learned how to create a simple server-side API to handle core game mechanics, ensuring separation of concerns and robust game result computation.

This project reinforced the interconnectedness of frontend and backend technologies and provided a solid foundation for tackling more complex web development challenges in the future.