Howzatt! Cricket Scorekeeper

Final Project Report(CS104)

Hindocha Avadh Dharmeshbhai
Roll Number: 24B1005
IIT Bombay, CSE Department

29th April 2025

Contents

1 Introduction **2**

2 Project Overview **2**
2.1 Motivation 2
2.2 Technology Stack 2
2.3 Directory Structure 2

3 Key Functionalities **3**
3.1 Setup Page 3
3.2 Live Page 3
3.3 Scorecard Page 5
3.4 Summary Page 6

4 Styling and UI Design **7**

5 Advanced Features Implemented **8**

6 Session Storage Design **9**
6.1 Persistent Match Data 9
6.2 How was the Data Stored 10
6.3 Handling Page Navigation 10

7 Challenges Faced **10**

8 How score.js Powers the Application **11**

9 Learnings **11**

10 Conclusion **11**

1 Introduction

Cricket, one of the most widely followed sports across the world, demands precise and dynamic scorekeeping to reflect the evolving state of a match. This project introduces **Howzatt! Cricket Scorekeeper** — a fully browser-based application designed to facilitate real-time ball-by-ball scoring for two-innings matches. The system not only captures runs, wickets, and overs but also updates individual player statistics and match summaries on the fly.

Built entirely using HTML, CSS, and JavaScript, this project emphasizes front-end development techniques to deliver a seamless and responsive scoring interface without any reliance on server-side or backend infrastructure. With an intuitive layout and structured game logic, the application mimics official scoring tables and commentary flow, making it both functional and user-friendly for casual games or practice sessions.

2 Project Overview

2.1 Motivation

Manual cricket scoring during casual or school-level matches is often prone to errors, delays, and lack of standard presentation. These issues affect match flow and can lead to disputes or loss of key statistics like player performance or partnerships.

Howzatt! Cricket Scorekeeper was developed to offer a simple, offline, digital alternative using just HTML, CSS, and JavaScript. It maintains live match data across multiple pages using `sessionStorage`, enabling features like real-time commentary, strike rate/run rate calculations, and automated innings transitions.

The focus on a frontend-only approach ensures wide accessibility and strengthens skills in DOM manipulation, session handling, and event-driven design — making it both educational and practical.

2.2 Technology Stack

- HTML for page structure
- CSS for styling
- JavaScript for logic and interactivity
- Session Storage for client-side data persistence

2.3 Directory Structure

```
score.js setup.html setup.css live.html live.css
scorecard.html scorecard.css summary.html summary.css
```

3 Key Functionalities

3.1 Setup Page

The setup page initializes the match configuration and prepares the web app for live scoring. It collects key inputs from the user and stores them for use throughout the match.

- Takes team names, toss winner, toss decision, and number of overs.
- Validates that all required fields are filled and overs is a positive integer.
- Determines which team bats first and sets up match logic accordingly.
- Stores the following in `sessionStorage`:
  - Team names, innings number, batting/bowling order.
  - Initial stats: runs, wickets, overs, extras.
  - Empty arrays for batting and bowling lineups for both innings.
  - Commentary templates for dot balls, runs (1 to 6), extras, and dismissals (bowled, caught, LBW).
  - Boolean flags such as `initialised`, `noBall`, `batterForm`, `bowlerForm`.
- All data is prepared for seamless handover to `live.html` using front-end-only logic (no backend or database).
- Commentary is randomized to make ball-by-ball updates dynamic and engaging.

3.2 Live Page

The `live.html` page is the central match engine where all scoring actions take place. It dynamically updates both the user interface and the stored match state using JavaScript, with full ball-by-ball interactivity.

- Contains buttons to log ball outcomes: 0–6 runs, wide, no-ball, and wicket.
- Captures and updates batter statistics per delivery: runs scored, balls faced, boundaries hit, and strike rate.
- Updates bowler statistics: overs bowled, runs conceded, wickets taken, maiden overs, and economy rate.
- Implements automatic strike changes on odd runs and end-of-over logic.
- Tracks innings progression: switches to second innings when all wickets fall or max overs are bowled.
- Automatically detects and displays match results:
  - By runs (if defending team wins).
  - By wickets and balls left (if chasing team wins).

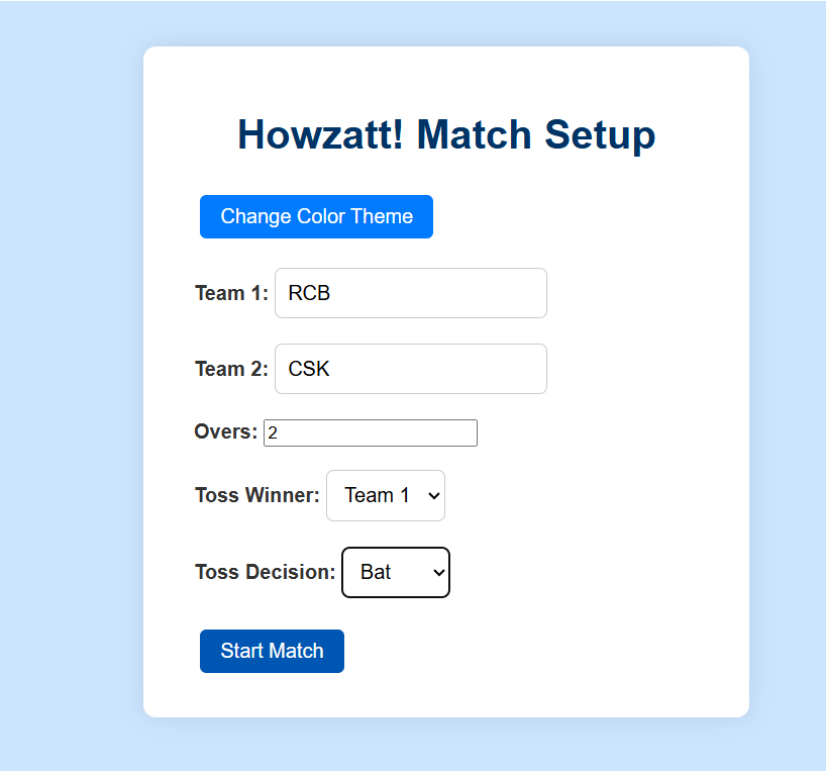

Figure 1: Setup Page Interface: Configuring the match before it begins.

  - Match tied if scores are level.

- Prompts for:
  - Initial players at start of each innings: Strike Batter, Non-Strike Batter, and Bowler.
  - New batter when a wicket falls.
  - New bowler at the end of each over.
- Maintains dynamic ball-by-ball commentary, randomized using pre-defined text arrays (dot balls, runs, boundaries, extras, and dismissals).
- Uses `sessionStorage` extensively to retain game state and allow seamless transition between pages.
- Displays real-time tables of active batter and bowler stats using HTML table updates.
- Shows current and required run rates dynamically as match progresses.

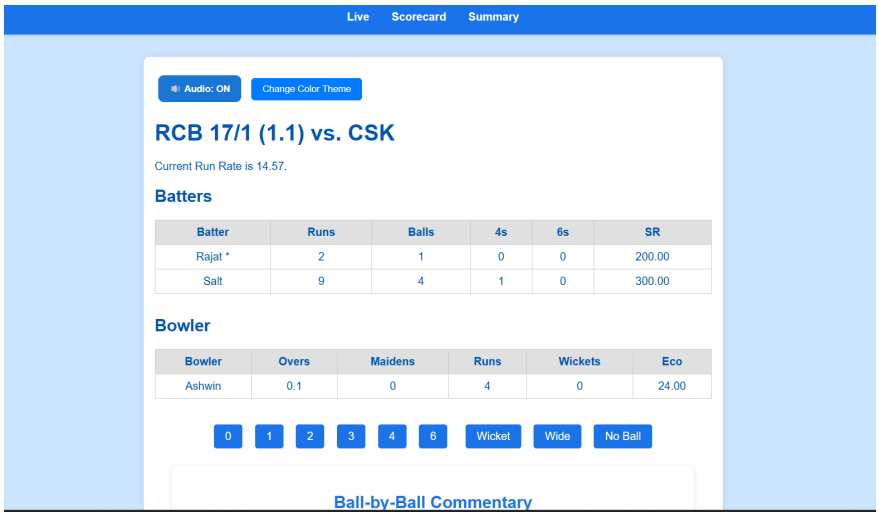

Figure 2: Live Scoring Interface: Ball-by-ball scoring and statistics update engine.

3.3 Scorecard Page

The `scorecard.html` page is designed to present a comprehensive overview of individual and team performances after (or during) the match. It fetches all player data from `sessionStorage` and renders it into structured tables representing both innings.

- The page is divided into four main sections:
  - Innings 1 Batting Scorecard
  - Innings 1 Bowling Summary
  - Innings 2 Batting Scorecard
  - Innings 2 Bowling Summary
- Batting scorecards list:
  - Player name, dismissal type (e.g., `b BowlerName`, `c Fielder b Bowler`), runs scored, balls faced, number of 4s/6s hit, and strike rate.
  - Highlights the dismissal string stored in the player's 11th stat field (e.g., `playerName + "stats"[10]`).
- Bowling summaries include:
  - Bowler name, overs bowled, maiden overs, runs conceded, wickets taken, and economy rate.
  - Stats are calculated dynamically by tracking overs and total balls delivered.
- Uses `getStats()` function to extract session-stored player statistics and populate HTML tables.

- Dynamically updates tables using the JavaScript DOM without refreshing the page.
- Responsive table layout improves readability across devices.
- Allows access mid-match without disrupting scoring or live data.

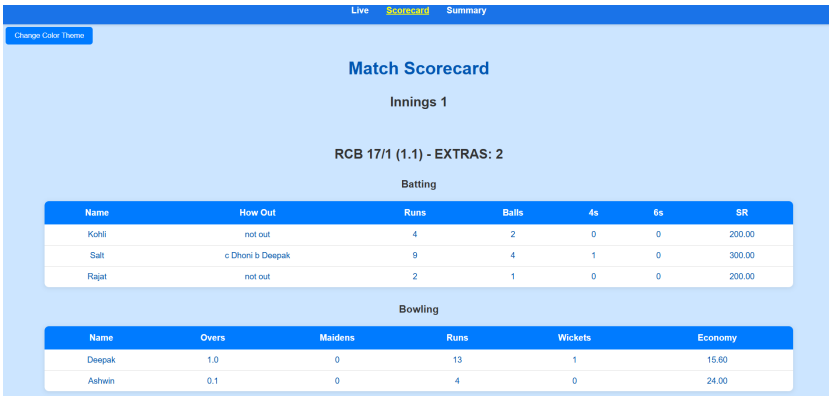

Figure 3: Scorecard Page: Full statistical breakdown of both innings.

3.4 Summary Page

The `summary.html` page displays the final match result once the second innings ends due to a win, loss, or tie.

- Fetches total runs, wickets, and overs from `sessionStorage`.
- Calculates result in three formats: win by runs, win by wickets (with balls left), or tie.
- Uses overs-to-balls logic to compute remaining deliveries.
- Includes a "Reset" button to clear session data and restart the match setup.
- The layout is responsive, with result text prominently centered for readability.

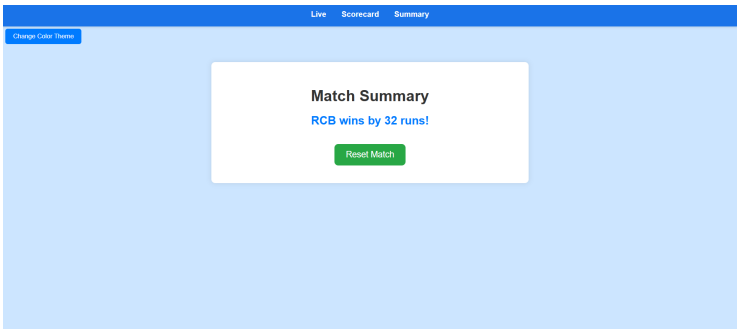

Figure 4: Summary Page: Displays the final match outcome and resets the application state.

4 Styling and UI Design

The application's user interface has been carefully crafted using CSS to ensure a visually appealing and consistent experience across all four major pages. Each page includes its own dedicated CSS file to isolate styles and improve maintainability.

- **Modular CSS Files**: Every HTML page is linked with a specific stylesheet:
  - `setup.html` → `setup.css`
  - `live.html` → `live.css`
  - `scorecard.html` → `scorecard.css`
  - `summary.html` → `summary.css`
- **Modern Layouts using Flex and Grid**:
  - Used `display: flex` and `justify-content/align-items` extensively to arrange forms and sections.
  - Centered containers ensure alignment across screen sizes.
  - Match dashboard uses flexible wrappers to dynamically organize buttons and stats.
- **Consistent Theme and Palette**:
  - A clean sky blue background paired with blue and black fonts for readability.
  - Section headers are styled to stand out using increased font weight and padding.
  - Button color transitions and box shadows enhance interactivity.
- **Button Design**:
  - Run buttons (0–6, extras, wicket) styled with margins, border-radius, and hover feedback.
  - Hovering changes color and opacity to guide user input.

  - Disabled buttons appear faded with `cursor: not-allowed`.

- **Forms and Placeholders**:
  - Input fields (e.g., batter/bowler entry) use placeholders with spacing and outlines.
  - Forms are padded, centered, and respond gracefully to page resizing.
- **Tables for Stats Display**:
  - Alternate row colors applied using `nth-child(even)` selector for readability.
  - Headers are bold and uppercased; columns are auto-sized for responsiveness.
  - Tables update dynamically using DOM manipulation.
- **Navigation and Responsiveness**:
  - A minimalist horizontal navigation bar appears on all major pages.
  - Links styled for color change on hover to indicate interactivity.
  - Layouts remain functional on varying screen widths.
- **Commentary Box Styling**:
  - Commentary entries rendered in a scrollable box with custom font and spacing.
  - Past 5 commentary lines maintained using dynamic rendering logic.
  - Uses subtle background shading to distinguish this section.
- **Custom Tab Icon:**
  - A custom favicon (tab logo) was added to enhance the application's branding and professionalism. This small cricket-themed icon appears on the browser tab for all pages, making the web app visually identifiable and polished.

Each CSS file was purpose-built to align visually with the core logic of the corresponding page, enhancing the intuitive experience for users. The uniformity of structure, clarity of button layouts, and table formatting are key contributors to the usability of **Howzatt!**.

5 Advanced Features Implemented

- **Ball-by-ball Commentary:** Each ball event triggers a dynamic commentary line selected randomly from pre-defined arrays such as dotBallCommentary, oneRunCommentary, fourRunCommentary, bowledCommentary, etc.
- **Rolling Commentary Feed:** The latest five commentary entries are displayed in a live-updating feed, enhancing the realism and immersion.

- **Extras Handling:** Fully supports logic for wide balls and no-balls, including automatic extra run addition and free-hit handling on no-balls.
- **Audio Commentary Integration:** To enhance the realism of the live scoring experience, an optional audio commentary feature was added. Prerecorded commentary lines (generated using text-to-speech synthesis in Python) were saved as individual MP3 files. When enabled, each scoring event (dot ball, runs, boundary, wicket, etc.) plays a corresponding commentary audio clip dynamically through JavaScript. A toggle button was added to the live scoring interface to allow users to start or stop audio commentary during the match. The audio system is fully browser- based and works offline once the audio files are available locally.
- **Theme Toggle Functionality:** A color theme toggle button was added to the live scoring interface, allowing users to switch between a blue and green theme dynamically. This was implemented using CSS variables and JavaScript to modify the root class on-the-fly. While the color override is not yet fully comprehensive across all elements, the feature demonstrates flexibility in UI customization and lays the groundwork for a fully theme-responsive design in future versions. (I got the idea for this on the last day, hence it has not been implemented properly.)
- **Free Hit Logic:** If a ball is declared a no-ball, wickets cannot be taken on the immediate next delivery (free hit enforcement).
- **Maiden Over Detection:** Overs without runs from the bat are flagged as maiden overs at their end, and the bowler's stat is updated accordingly.
- **Prevent Double Triggers:** Catch dismissal and bowled confirmation buttons use one-time event listeners (e.g., { `once: true` }) to avoid unintended logic stacking.
- **Interactive Player Setup:** New batter and new bowler forms appear contextually at wickets or over changes, ensuring input validation and smooth gameplay.

6 Session Storage Design

6.1 Persistent Match Data

To maintain game state across multiple pages, sessionStorage is leveraged to retain:

- Names of current striker, non-striker, and bowler.
- Score counters: runs, wickets, overs, extras for both innings.
- Arrays of player statistics (runs, balls, fours, sixes, etc.).
- Player initialization flags and commentary indexes.
- Lists of all batters/bowlers used per innings.
- Commentary history array to populate the rolling display.

6.2 How was the Data Stored

There are 4 arrays created in the JavaScript while initialising the match:

- Players of batting first for batting
- Players of bowling first for bowling
- Players of batting first for bowling
- Players of bowling first for batting

Players are added to the array according to need, and they are initialised by the initialisePlayer function, which basically makes an array for all individual players. The size of the array of each player is 11 elements, and it represents runs, balls, fours, sixes, strike rate, overs, maidens, runs, wickets, economy, and how out for each player, and these stats are dynamically updated as the match goes on. By this method, it's really easy to write the script for showing scorecard.

6.3 Handling Page Navigation

As the application spans across setup, live scoring, scorecard, and summary pages, sessionStorage ensures a persistent game state without server reliance. For instance:

- Live page resumes where it left off without data loss.
- Scorecard page renders statistics live from session data.
- Summary page computes winner based on session-based totals.

7 Challenges Faced

- **Overs and Ball Tracking:** Representing overs as x.y and converting accurately to balls and vice versa was essential to ensure logical consistency.
- **Bowler/Batter Switching:** Correctly rotating strike and tracking bowler changes after full overs without missing edge cases.
- **Simultaneous Input Race Conditions:** Preventing bugs from rapid user actions (e.g., wicket + no-ball conflict) using event lockouts and booleans.
- **Dynamic DOM Rendering:** Ensuring correct table updates and data overwrites, especially when new players entered the match.
- **Single-use Event Handlers:** Avoiding double submission of new batter/bowler entries via once-true flags and visibility toggles.

8 How score.js Powers the Application

`score.js` is the main JavaScript logic file that drives the core interactivity and state management across all pages. Its key responsibilities include:

- **Setup Initialization:** Handles input validation, toss logic, innings configuration, and commentary array generation on the `setup.html` page.
- **Scoring Engine:** Updates batter and bowler stats for every ball, registers wickets, rotates strike, and manages innings transitions.
- **Match Progress Tracking:** Detects end of innings, enforces match result logic, and calls transition routines.
- **DOM Manipulation:** Dynamically updates score display, commentary feed, tables for batters and bowlers.
- **Session Persistence:** Reads from and writes to sessionStorage at every logical point to keep state synchronized.
- **Special Inputs:** Manages the forms for new batter and new bowler with context-based visibility and validation.
- **Commentary Handling:** Ensures varied commentary lines per event using cycling index and randomization.

9 Learnings

- Deep understanding of event-driven programming with JavaScript.
- Creative use of sessionStorage for frontend-only state persistence.
- DOM traversal and manipulation for dynamic table rendering.
- Importance of UX flow and managing visibility of interactive elements.
- Debugging asynchronous issues with DOM updates and event stacking.

10 Conclusion

The **Howzatt!** cricket scoring web application is a full-fledged frontend system capable of managing a complete cricket match digitally. It not only replicates traditional scorekeeping but enhances it with real-time commentary, accurate player metrics, and a clean user interface. The modularity and stateless server architecture make it highly portable and extendable.

Potential future directions include adding persistent storage (using localStorage or backend), support for multi-device sync, or downloadable match reports.

References

- https://openai.com/index/chatgpt/
- https://freecodecamp.org/news/web-storage-localstorage-vs-sessionstorage-in-javascrip
- https://www.w3schools.com/