HOWZATT! Cricket Score Keeper

Uma Kumari
Roll No: 24B1026

Overview

This document explains the functionality, data handling, and logic behind the JavaScript implementation of the cricket scoring web application developed by Uma Kumari (24B1026). The application supports live scoring, strike rotation, innings handling, and bowler statistics tracking.

1 Application Structure

- HTML Pages: setup.html, live.html, scorecard.html, match_summary.html
- CSS Pages: setup.css, live.css, scorecard.css, match_summary.css
- JavaScript File: score.js

2 Instruction

- In setup, user enters the name of the teams playing and selects the team that wins the toss and their decision.
- In live, user enters the number of runs and clicks on the submit button for each ball unless it is a wide, or no ball, or a wicket has fallen.
- If the ball is wide, the user clicks on the button 'wide'.
- If it is a no ball, the user selects the number of runs taken and then clicks on the button 'no ball'.
- If a wicket has fallen, then the button 'wicket' should be clicked then, the user will receive a prompt for the new striker.
- To go to the scorecard to see the data for batsmen and bowlers in the ongoing innings, user must click on the button 'Go To the Scorecard'.
- When the match is over, the live page will not take more inputs and a button to go to match summary will appear where the result for the match will be displayed.

3 In Setup

3.1 Functions and Data Flow

3.1.1 start_match()

Initializes local/session storage variables for a new match. It saves team names, toss winner, and decision. It is used in live.html to decide which team will bat first.

3.2 CSS

- Button: On hovering, the button darkens.
- Margin: The form wrapper has a margin with a bordered white box.
- Opacity: Reduced opacity with a stadium background image.

3.3 Screenshot

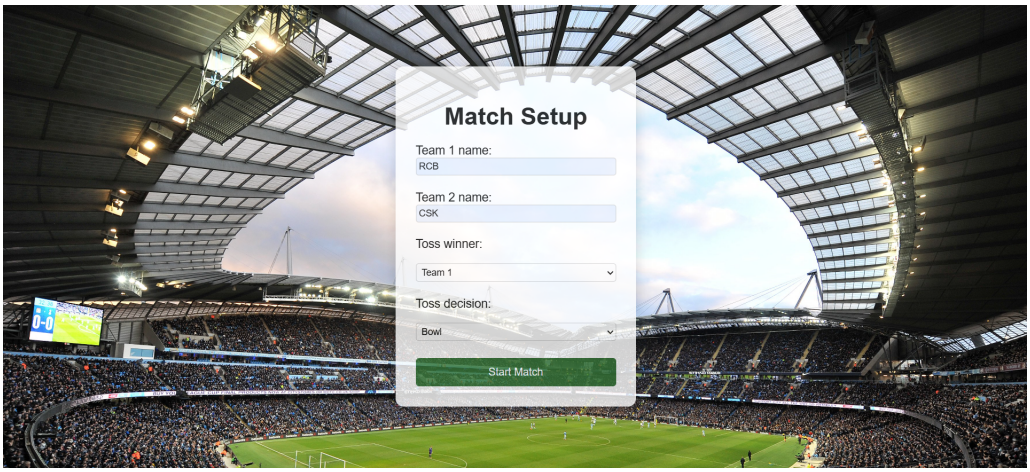

Figure 1: Setup Page Interface

4 In Live

4.1 Functions and Data Flow

Handles runs, wickets, strike rotation, and bowler changes after every 6 balls.

4.2 CSS

Styled inputs, radio buttons, tables, and a translucent scoring container over a stadium background.

Figure 2: Live Scoring - Input Panel

Figure 3: Live Scoring - Scoreboard Display

4.3 Screenshots

5 In Scorecard

5.1 Functions and Data Flow

Displays batter and bowler stats saved in sessionStorage. Fetches from storage and formats in a table.

5.2 CSS

Table formatting, use of borders and paddings for score visibility.

5.3 Screenshots

Figure 4: Scorecard - Batting Stats

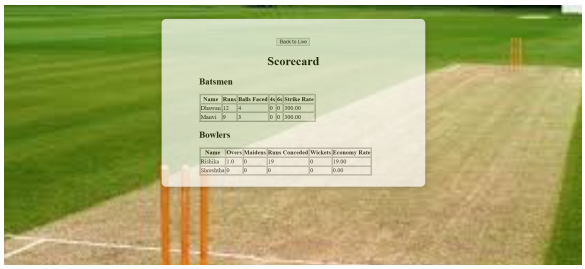

Figure 5: Scorecard - Bowling Stats

6 In Match Summary

6.1 Functions and Data Flow

Loads and displays the match result from sessionStorage. Depending on which team's runs are more and whether the batted or bowled in the first innings, the result is displayed.

6.2 Screenshot

Figure 6: Match Summary Page

7 JavaScript Logic and Data Flow

7.1 Live Page Logic

Ball Submission and Score Update

- Handles input from the user like runs, wicket, extras.
- Updates:
  - Batter statistics: runs scored, balls faced.
  - Bowler statistics: runs conceded, balls bowled, wickets taken.
  - Team totals: runs, wickets, overs.
- All updates are stored in `sessionStorage` to maintain persistence across navigation. persistState() function is used for this. This so that whenever scorecard page is opened from live then all the data is updated and there is no lag between live and scorecard.
- updateScoreboard() function is used to update the data in the batters and bowlers tables present in the Live page.

Strike Rotation Logic

- Triggered after each ball submission.
- Logic:
  - If runs scored are odd, striker and non-striker are swapped.
  - After every 6 legal balls, an over is completed:
    - Strikers are rotated.
    - A new bowler is selected from the input.
- swapStrikers() function is used to switch the striker and non-striker.

Bowler Rotation and Economy Update

- Occurs after every completed over (6 legal deliveries).
- The economy rate is computed as:

$$Economy = \frac{RunsConceded}{OversBowled}$$

- Updates bowler stats in `sessionStorage` for consistency.

Innings Transition and Match Completion Logic

The code uses a ball-by-ball tracking system to manage innings transitions and determine when the match concludes. Each innings is limited to 12 balls, simulating a short-format match. The logic maintains a running count of balls bowled, and upon reaching 12 deliveries or 10 wickets, the first innings concludes automatically. At this point, all relevant data—including batter and bowler statistics, total runs, and wickets—is saved to `sessionStorage` under a separate key to isolate innings data. When the second innings begins, a new set of striker, non-striker, and bowler stats is initialized, and the run chase begins based on the target set in the first innings. The match ends automatically once the second innings reaches 12 balls, loses 10 wickets, or surpasses the target score. A summary result is generated using simple conditional logic that compares the total runs and computes either a winning margin by runs or wickets, or declares a tie if both teams have the same score.

7.2 Scorecard Page Logic

Clicking on 'Go To Scorecard' button on live calls the function go_to_scorecard() which calls persistState() to save all the data to sessionStorage() and then on loading scorecard.html all the data is brought by getItem() function from sessionStorage and updated in the tables present in the scorecard.

7.3 Match Summary Page Logic

- When Match Summary button is clicked in the live page, then go_to_matchsummary() is called which redirects to match_summary.html and saves the required data to decide the winner to the sessionStorage.
- Loads final match data from `sessionStorage`:
  - Runs, wickets, balls, team names.
  - Toss winner and decision (bat or bowl).
- Based on batting order and final scores, the result is computed as:
  - Win by runs.
  - Win by wickets (with balls remaining).
  - Tie if scores match.

Example logic:

```
if (team1_runs > team2_runs) {
    result.innerHTML = `${team1_name} wins by ${team1_runs -
       ↪ team2_runs} runs`;
} else if (team2_runs > team1_runs) {
    result.innerHTML = `${team2_name} wins by ${10 -
       ↪ team2_wickets} wickets`;
} else {
    result.innerHTML = `The match was a tie.`;
}
```

7.4 Customizations

I implemented three customizations.

7.4.1 Live commentary

With every ball there are different cases.

- If only runs are scored, then the comment is updated before the strikers are swapped(which happens if the number of runs is odd). The comment is like- (striker name) took (runs on this ball) runs.
- If it is a wide or a no ball, then the comment is displayed by calling wide() function or no_ball() function. The comment is- 'It is a wide' or 'It is a no ball. Runs conceded on this ball are (nruns on this ball).
- If a wicket has fallen. Then the batsman who got out along with the number of runs he/she took in the number of balls he/she played is the comment.

7.4.2 Wide

- Number of runs of the batting team in that innings is increased by 1. Number of balls is not changed.
- Number of runs conceded by the current bowler is increased by 1. Consequently, economy rate changes.

7.4.3 No Ball

- Number of runs of the batting team in that innings is increased by 1 more run than the number of runs taken on that ball. Number of balls is not changed.
- Number of runs taken by the striker increases by the number of runs taken on that ball. Number of runs conceded increases by 1 more than the runs scored on that ball.
- Consequently, economy rate and strikerate of striker changes.

7.5 Overall Data Flow Summary

- **Setup Page (`setup.html`):**
  - Stores team names, toss winner, and toss decision in `localStorage`.
- **Live Page (`live.html`):**
  - Handles ball-by-ball inputs and updates player stats.
  - Maintains match state in `sessionStorage`, including striker/nonstriker, current bowler, ball count, and score.
- **Scorecard Page:**
  - Fetches and formats batter and bowler stats stored in `sessionStorage`.
- **Match Summary Page:**
  - Reads stored values and determines the final match result using conditional logic.

8 Data Storage Structure

8.1 LocalStorage

Stores:

- `team1`, `team2`
- `tossWinner`, `tossDecision`

8.2 SessionStorage

Stores:

- `striker`, `nonstriker`
- `bowlerStats` (economy, overs, wickets)
- `ballCount`, `runs`, `wickets`
- `firstInningsData`, `secondInningsData`

9 Future Improvements

- Add player profiles and match history
- Optimize for mobile devices
- Add more innings (e.g., Test format)
- Store complete matches in a backend database
- Increase number of overs in each innings.

10 References

1. W3Schools. “For JavaScript Syntax.”https://www.w3schools.com/
2. International Cricket Council (ICC). “Cricket Rules.” https://www.icc-cricket.com/about/cricket/rules-and-regulations
3. Cricbuzz. “To see an example for cricket score keeping sites.” https://www.cricbuzz.com/
4. GeeksforGeeks. “JS (localStorage and JSON.stringify).” https://www.geeksforgeeks.org/
5. Lecture Slides and Codes. “coding syntax.”