🏏**Capstone Project** **Documentation – Cricket Trivia & Quiz App**

# **Introduction**

## **Purpose**

**Problem/Opportunity**: Cricket fans lack dynamic platforms that challenge their knowledge across diverse topics like player stats, rules, and match history. Existing apps are often static, generic, or lack competitive features.

**Value**: Trivia experiences with real-time timers, fair scoring, and category-based challenges boost engagement, learning, and replayability.

**Current State**: Static quizzes, little leaderboard competition, poor scoring logic, and no category-level segmentation.

**Desired State**: A fast, visually polished, user-friendly cricket trivia platform with timed questions and competitive leaderboards.

**Past Projects**: Many basic quiz platforms exist (e.g. Kahoot) but most aren’t cricket specific and lack dynamic scoring or leaderboards.

## **Industry/ domain**

**Domain**: Sports Education & Fan Engagement

**Current State**: Cricket audiences demand interactivity. Fantasy sports and live stat apps are booming. Trivia is underused in this space.

**Value Chain**: The app creates value across content, delivery, engagement, and retention – which is everything a modern gamified learning platform aims for.

**Key Concepts**: Gamification, timed engagement, fan personalisation, score fairness

**Cross-industry Potential**: This model applies to football, corporate onboarding, academic testing, music trivia, and general edutainment.

## **Stakeholders**

|  |  |  |  |
| --- | --- | --- | --- |
| **Stakeholder Type** | **Access Level** | **Why They Care** | **Expectations** |
| Cricket Fans | Full | Compete, learn, engage | Timed quizzes, fair scoring, ranking |
| Admins/Moderators | Backend | Maintain data and questions | Easy question control and access |
| Developers | Full | Build and maintain features | Clean API contracts and docs |

# **Product Description**

## **Architecture Diagram**

A diagram of a software application

AI-generated content may be incorrect.

Frontend (React.js + MUI):

* Renders views (/, /quiz, /score, /leaderboard)
* Axios for API calls
* Countdown timer, answer logic

Backend (Node.js + Express.js):

* RESTful API routes
* Validates scores, enforces rank logic
* Swagger at /api-docs

Database (MySQL):

* users, questions, user\_scores
* Optimised for category filter and leaderboard lookups

## User Stories

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **User Story Title** | **Description** | **Priority** | **Notes** |
| 1 | Choose Quiz Category | As a user, I can pick a category before the quiz begins. | High | History, Rules, Miscellaneous, Records, Players. |
| 2 | Timed Questions | As a user, I can answer each question within a time limit. | High | Uses Timer component |
| 3 | Score Summary | I want to see my score, time taken, and review answers. | High | Uses ScoreScreen page |
| 4 | Compare with Leaderboard | See how many scores rank compared to others in real time. | High | Leaderboard ranks |
| 5 | Replay or Switch Category | I can return to homepage and play a different quiz. | Medium | Encourages reengagement |
| 6 | Submit Score Persistently | Score should store in DB and rank even with ties. | High | Shared-rank logic used. |

## **User Flow**

A diagram of a complex system

AI-generated content may be incorrect.

## **Figma Mockup Design (Desired Outcome)**

Start Screen

A screenshot of a quiz

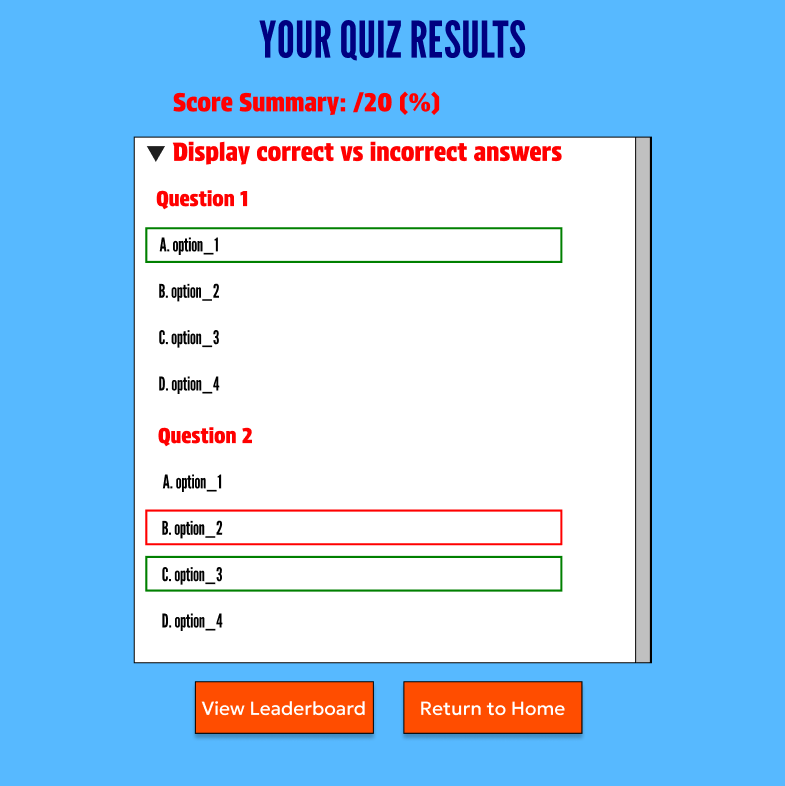
AI-generated content may be incorrect.

Category Selection Pop Up  
A screenshot of a quiz

AI-generated content may be incorrect.

Quiz Interface (Question Display)  
A screenshot of a computer

AI-generated content may be incorrect.

Score Display (After Quiz)  


Leaderboard Screen

A screenshot of a computer screen

AI-generated content may be incorrect.

## **Database Design**

A screenshot of a computer screen

AI-generated content may be incorrect.

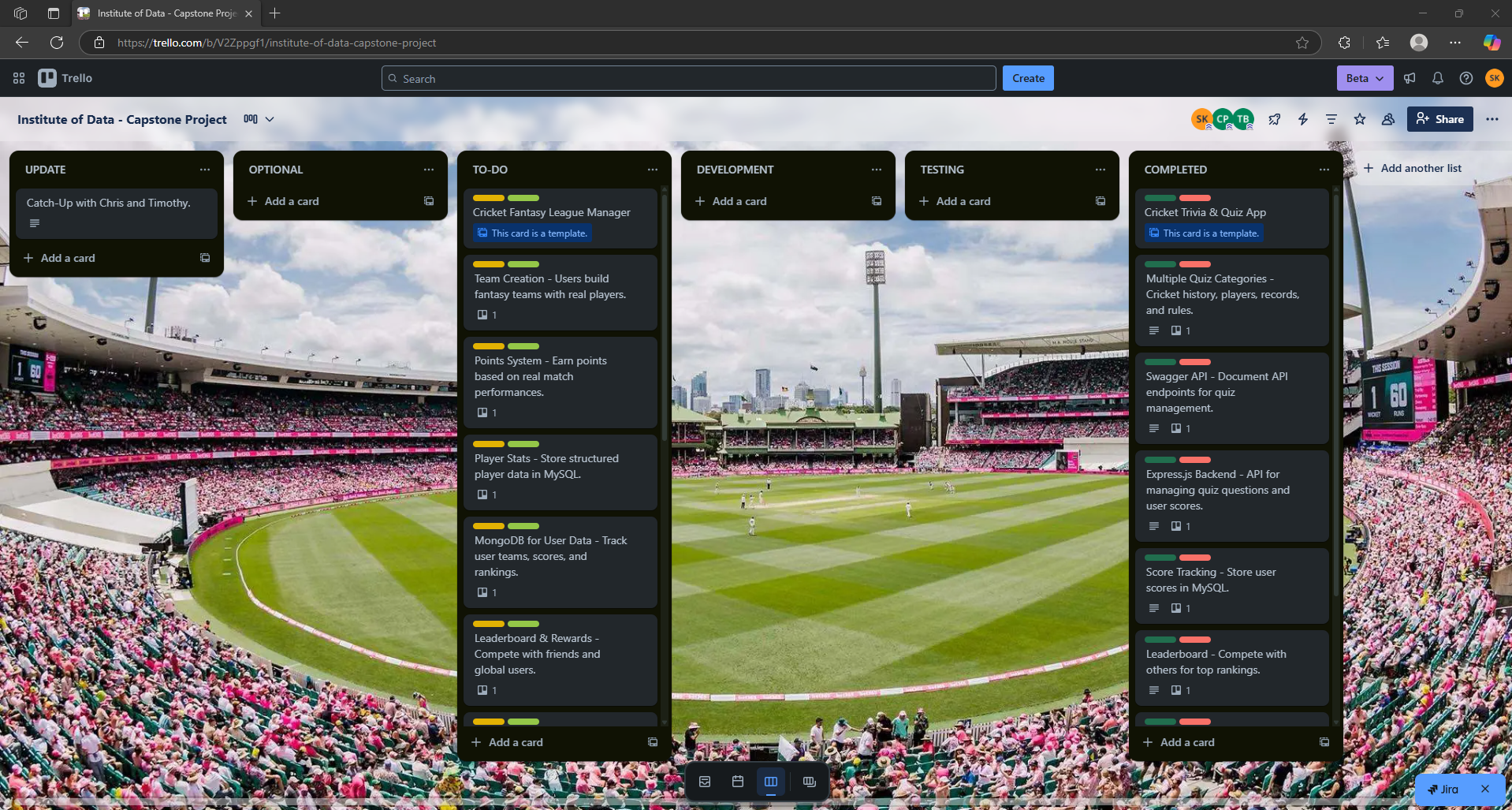
## **Open Questions/Out of Scope**

* Email login or authentication system
* Admin dashboard for adding questions
* Real-time multiplayer or PVP quiz battles
* Push notifications

## **Non-functional Requirements**

|  |  |
| --- | --- |
| **Requirement** | **Description** |
| Security | No user credentials stored; only name used during quiz session. |
| Performance | 1000 sessions/hr; lazy-loaded question set, optimised SQL. |
| Usability | Simple UI with MUI, responsive views, large clickable buttons. |
| Speed | Load time less than 1s, answer transition less than 300ms. |
| Reliability | Recoverable quiz state (handled via props/state on refresh). |
| Maintainability | Swagger API, modular backend routes, MySQL schema, dotenv usage. |

# **Project Planning**

**Trello Board**:

Cricket Trivia & Quiz App (Priority 1)

* Multiple Quiz Categories – Cricket history, players, records, and rules.
* Timed Quizzes – Users answer within a set time limit.
* Score Tracking – Store user scores in MySQL.
* Leaderboard – Compete with others for top rankings.
* React.js UI – Interactive quiz interface with Bootstrap & Material UI.
* Express.js Backend – API for managing quiz questions and user scores.
* Swagger API – Document API endpoints for quiz management.

My Roadmap

Phase 1: Planning & Setup (Week 1)

* Define Core Features:
  + Multiple quiz categories (e.g., History, Players, Rules, Records, Miscellaneous)
  + Timed quizzes, scoring, and leaderboard
* UI Wireframes & Flow Design:
  + Homepage → Category selection → Quiz → Score → Leaderboard
* Tech Stack Decisions:
  + Frontend: React.js, Bootstrap, Material UI
  + Backend: Node.js + Express.js
  + Database: MySQL for questions, users, and scores
* Initialize GitHub Repository and project structure

Phase 2: Backend Development (Week 2–3)

* Set Up Express Server and RESTful routes
* Create API Endpoints:
  + GET /api/quiz/categories
  + GET /api/quiz/questions
  + POST /api/quiz/submit
  + GET /api/quiz/leaderboard
* Database Design:
  + users: player info
  + questions: trivia content
  + user\_scores: submitted scores per category
* Swagger Integration: Document all routes at /api-docs

Phase 3: Frontend Development (Week 4–5)

* Build UI Components:
  + CategoryPopup modal
  + QuizInterface with countdown timer
  + ScoreScreen for results and review
  + Leaderboard for rankings
* Implement Routing & State Management
* Styling & Responsiveness: Apply Material UI + Bootstrap

Phase 4: API Integration & Gameplay Logic (Week 5–6)

* Connect Frontend to Backend via Axios
* Timed Quiz Logic: Disable input on timeout
* Score Submission: POST to backend and reflect in leaderboard
* Rank Logic: Shared rank for tied scores, animated transitions

Phase 5: Testing & Optimization (Week 6–7)

* Unit Testing with Vitest: Timer, Score calculation, UI routing
* API Testing: Swagger + Axios responses
* Edge Case Handling:
  + Empty user name
  + Incomplete quiz submission
  + Duplicate/tied scores
* Performance Optimization: Mobile layout, animation speed, button feedback

Phase 6: Documentation + GitHub Deployment

# **Testing Strategy**

Steps to Ensure Product Quality:

* Implemented unit tests using Vitest and @testing-library/react.
* Used modular testing for each feature (Timer, CategoryPopup, ScoreScreen).
* Integrated Axios and React Router mocks to isolate component logic.
* Manual exploratory testing for layout responsiveness, route navigation, and user interaction flow.
* Backend API tested via Swagger and Thunder Client.

Feature-Level Testing:

* Timer Component: Verified countdown, input lock on timeout.
* Category Popup: Tested modal open/close, category selection, validations.
* ScoreScreen: Ensured score, time, and answer review display accurately.
* Leaderboard: Validated shared-rank logic and API data rendering.
* Routing & UX: Checked flow across homepage, quiz, score, leaderboard
* API Endpoints: Swagger tests for /categories, /questions, /submit, /leaderboard.

Edge Case Handling:

* Empty Username: Triggers alert and prevents quiz start.
* Timeout Scenarios: Auto-disables options after time runs out.
* Incomplete Quiz Submission: Prevented partial scores from being stored.
* Tied Scores: Shared rank logic implemented to ensure fairness.
* Repeated Renders: Fixed leaderboard duplicate rendering bugs via unique keys and state guards.

# **Implementation**

* Hosted frontend on http://localhost:5173/
* Express.js backend at http://localhost:5000/
* MySQL connection managed via .env
* Data validated on backend before submission
* Swagger served at /api-docs for dev/test team

# **End-to-end solution**

|  |  |
| --- | --- |
| **Goal** | **Status** |
| Testing coverage and mocks | ✓ |
| Category-driven question randomiser | ✓ |
| Responsive frontend with routing | ✓ |
| Accurate leaderboard ranking | ✓ |
| Score saving with category and user | ✓ |

# **References**

**GitHub**: <https://github.com/saishk98/IoD-CapstoneProject>

**Figma**: <https://www.figma.com/design/MYwO4mNtgHfJv63vHSFrj1/Institute-of-Data--IOD----Capstone-Project?m=auto&t=TDuznfvC5VnRUZWe-6>

**Trello**: <https://trello.com/b/V2Zppgf1/institute-of-data-capstone-project>

**Front End**: CSS. BootStrap.css, JavaScript, React.js, Material UI, Axios, Vitest

**Back End**: Node.js, Express.js, MySQL, Swagger API, Thunder Client