3311 System Design Document
Green Bean

Table of Contents

CRC Cards

Class Name: **Users**	
Responsibilities - Keeps track of user bookings - Keeps track of the comments and ratings of arenas	*Collaborators* - Bookings - Arena - Feedback

Class Name: **Bookings**	
Responsibilities: - Prompt User to make a booking for an arena - Keeps track of the user's bookings for one or more arenas - Let arenas keep a list of bookings made from user	*Collaborators* - Arena - Users

Class Name: **Events**	
Responsibilities: - Prompt User to making events through the bookings of the arena - Keeps track of the user's events for one or more arenas - Let arenas keep a list of events coming up	*Collaborators* - Arena - Users - Bookings

Class Name: **Arena**	
Responsibilities - Displays information of the arena - Keeps track of the bookings made by users - Displays comments of each arena	*Collaborators* - Users - Bookings - Feedback

Class Name: **Users**	
Responsibilities - Keeps track of user bookings - Keeps track of the comments and ratings of arenas	*Collaborators* - Bookings - Arena - Feedback

Class Name: **Feedback**	
Responsibilities - Keeps track of user comments and ratings - Each comments and ratings will be under one of the arenas	*Collaborators* - User - Arena

System Interaction

Our Web application will be made through a FERN stack. The FERN stack will be utilizing Firebase, Express, React, and Node.js. Here is a description of the System interaction, describing the dependencies and requirements made for the application.

Operating System

Server environment for the backend is can be cross-form, meaning that the web application Will be able to run on Linux, Windows and Mac

Programming Languages and Runtime

Node.js will be used as the Runtime, built along with the Express framework to work around the backend. Both will utilize JavaScript and Typescript as our main languages. Npm (Node Package Manager) is installed for managing the dependencies for both server and client Applications. React will be used for the Frontend of the web application.

Databases

A Firebase Project must be created to apply its services to the web application. The service for providing data and handling its functions is Firestore. Firestore will be online based so no local database is being used, and also it will be used in real-time. Firebase CLI needs to be installed on the local environment to enable deployment and local testing. It also requires authentication to the Firebase project.

Network Configuration

Requires ports 3000 for running the web application.

Environmental Variables

For the environment variables requires sensitive information like **Firebase API keys**, **Firebase Project ID**, **Service Account credentials** (for server-side Firebase Admin SDK if used), and any other third-party API keys (e.g., payment or analytics services) to be stored securely in environment variables. All the variables will be saved on an **.env** file.

Software Architecture: MVC

Our FERN stack application will be built by using the Model-View-Controller. This architecture well be helpful for organized code in logical layers, improved readability, maintainability, proper separation of components for single responsibilities.

Model

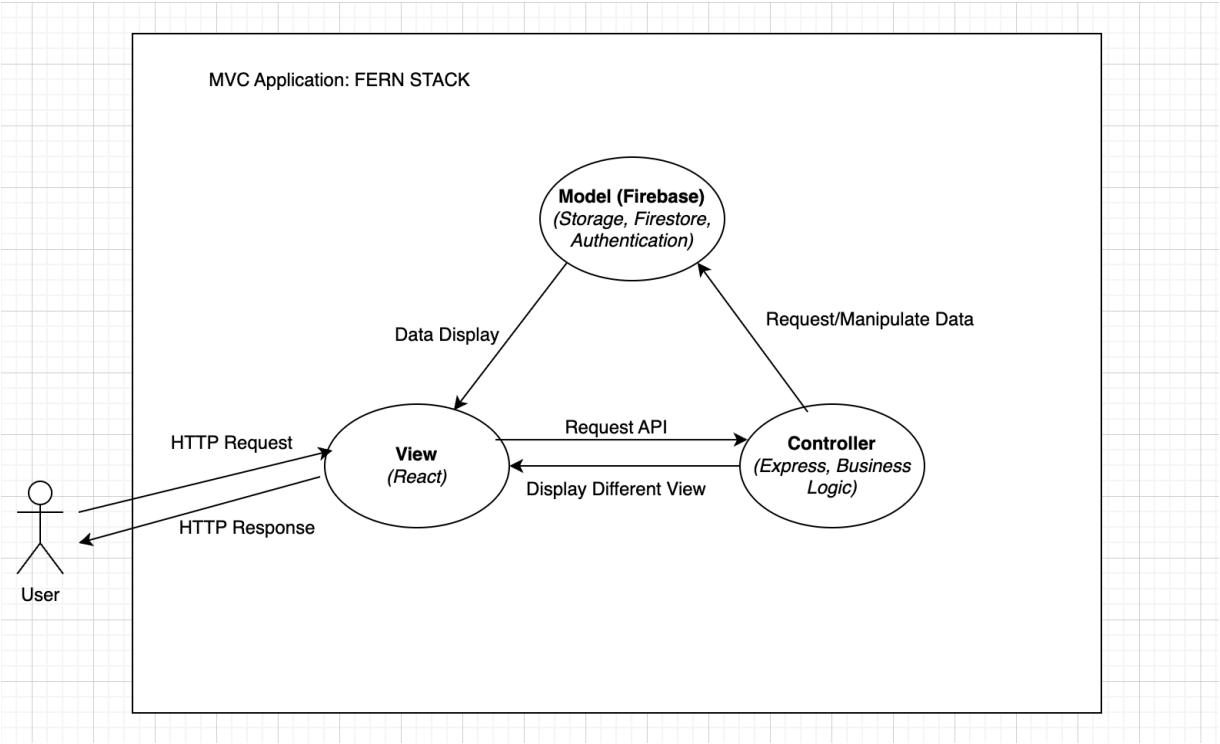
The model component represents the data layer of the application. Firebase services will be used to handle data storage. For the Model component, it is responsible for retrieving, storing and manipulating data in Firebase. It will define data structures and constraints. And lastly It will perform validation and other data-related logic

View

React will serve as the View component. The View will be responsible for displaying data and responding to user actions. It has the responsibility to communicate with the Controller component to retrieve and modify data. And lastly it will take on Updating UI components to be optimized with any changes with the data model or application state.

Controller

The Controller component acts as the connection between View and Controller. The Express server will be implemented to the stack, acting as its Controller component. The controller component is responsible for handling HTTP requests from the client (React frontend). It should invoke the appropriate model actions to retrieve or manipulate data. The processed data should be returned in the View component. It should also apply application-specific business logic and routing logic.


MVC Application: FERN STACK
Model (Firebase)
(Storage, Firestore, Authentication)
Data Display
Request/Manipulate Data
HTTP Request
View
(React)
Request API
Controller
(Express, Business Logic)
Display Different View
HTTP Response
User

System Decomposition

Model (Firebase, Authentication)

Role - Handles data storage and retrieving it, and other backend services like cloud storage.

Detailed Design - Data is structured by using collections provided by Firestore. Constraints will be applied to ensure data integrity. Authentication rules define access permissions based on user roles.

Error Strategies

- *Invalid Data Input:* Client-side input will be validated to ensure proper data submissions and reducing invalidity. For the backend, Firebase functions will further validate the data to decide whether to reject the data or accept it.
- *Network or External System Failure:* If Firebase services are temporarily down, the system will retry the data requests for a short interval time. If that does not work out, then Firebase will send a response message to notify the system that the services are Unavailable. The Controller will notify the View to display the error message.
- *Authentication Errors:* Errors during login will prompt user to retry typing their correct credentials. Specific errors will not be disclosed for security purposes

View (React)

Role - Provides the user interface, handling data display and capturing user interactions

Detailed Design - React components are designed to make up the UI and handle specific elements and user action. Each component handles its own state and uses props to interact with child components, updating based on backend data and Controller responses.

Error Strategies

- *Invalid User Input:* The View includes client-side validation for forms(eg. email, sign in, payment, format check, required fields) to catch basic input errors before submission.
- *Display of Network Errors:* When the Controller detects network or data issues, it relays error messages to the View.
- *Fallbacks for Critical Failures:* For prolonged service issues, error pages will be shown to give users context.

Controller (Express)

Role - Acts as the intermediary, handling requests from the View controller, apply business logic, and interacting with the Model.

Detailed Design - The Controller component routes each HTTP request to the correct function, validates input, performs actions on the Model, and formats data for the View. Express middleware like body-parser and cors manage request data and security.

Error Strategy

- *Invalid User Input:* The Controller performs server-side validation to ensure user inputs meet required standards. Invalidation will trigger error messages with specific status codes (eg. 400) and messages.
- *Error Responses for External Failures:* If Firebase fails to respond, the Controller will respond by sending an HTTP 503 error to the View, along with a “try again later” message.
- *Fallbacks for Logic Failures:* When the application will encounter unexpected errors, the Controller will catch it using the error-handling functions provided by Express. The error will be logged, and it returns a generic error response to avoid exposing sensitive details