HACKATHON PHASE 1

COLLEGE CODE: 9616

COLLEGE NAME: MARTHANDAM COLLEGE OF ENGG & TECHNOLOGY,
KUTTAKUZHI

DEPARTMENT : COMPUTER SCIENCE AND ENGINEERING

STUDENT NM ID: 4A292CE454FD89980DA46C60A07C0C29

ROLL NO: 23CSE43

DATE: 06.10.2025

TECHNOLOGY : FRONT END TECHNOLOGY

PROJECT NAME: RESTAURANT TABLE BOOKING APP

SUBMITTED BY.

TL: KIRUBHA.J.WILSON(9442055847)

TEAM MEMBERS: RAMYA R.R(8220785941)
MONISHA.J (
MEGHA SHINY(6385362694)

TABLE OF CONTENTS

PROJECT OVERVIEW AND OBJECTIVES

PROBLEM STATEMENT

The modern dining experience is often hampered by inefficiencies such as long waiting times for tables, difficulty in making reservations, and a lack of real-time information for both customers and restaurant staff.

Traditional phone-based reservation systems are cumbersome and do not provide dynamic updates on table availability or wait times. Customers need a seamless, digital solution to discover restaurants, check real-time availability, and book their dining experience effortlessly.

KEY FEATURES

Smart Restaurant Discovery & Search:

Users can search for restaurants based on cuisine, location, and ratings.

- Real-Time Table Reservation:

A dynamic booking system that shows live table availability.

- Estimated Time of Arrival (ETA) Integration:

Utilizes Google Maps API to provide accurate travel time, helping to manage table turnover.

- Digital Queue Management:

For walk-in customers, a virtual waiting list eliminates the need for physical waiting.

- Customer Relationship Management (CRM) & Loyalty Features:

Inspired by systems like Eat App, this will help restaurants manage customer data and offer loyalty rewards.

EXPECTED OUTCOMES/ OBJECTIVES

SDINE aims to become a comprehensive platform that bridges the gap between diners and restaurants.

The expected outcome is a fully functional web application that reduces customer wait times, increases restaurant operational efficiency and table turnover, and enhances the overall dining experience through technology. The system will be scalable, user-friendly, and provide valuable analytics to restaurant partners, similar to the tools offered by OpenTable and Tock.

TECHNOLOGY STACK & ENVIRONMENT SETUP

1.Backend

- Runtime Environment: Node.js
- Web Framework: Express.js. This will be used to create a robust and scalable server, handle HTTP requests, and define our RESTful API endpoints.
- Authentication: Firebase Auth for secure user and restaurant staff login/signup.

2. Frontend

- Framework: React.js. It will be used to build a dynamic, component-based, and responsive user interface.
- State Management: Context API and/or Redux for managing global application state (e.g., user authentication status, shopping cart for pre-orders).

3. Database

- Primary Database: Cloud Firestore from Firebase. This NoSQL database is chosen for its real-time synchronization capabilities, which are crucial for live table availability and queue updates. It also offers seamless integration with other Firebase services.

4. APIs & External Services

- Maps & Routing: Google Maps API will be integrated for location services, displaying restaurants on a map, and calculating ETA for users.

- Payment Gateway: (Planned for future phases) Stripe or Razorpay API for handling prepaid reservations, inspired by Tock's model.

5. Development Tools & Environment

- Version Control: Git with a remote repository on GitHub.

- Package Manager: npm or Yarn.

- Code Editor: VS Code.

- Environment Setup: Detailed instructions will be documented for setting up a local development environment, including cloning the repo, installing dependencies, and configuring environment variables for API keys.

API DESIGN & DATA MODEL(1)

1. Planned REST Endpoints

User-Facing Endpoints:

- POST /api/auth/signup - Register a new user.

- POST /api/auth/login - Authenticate a user.

- GET /api/restaurants - Fetch a list of restaurants (with filters).

- GET /api/restaurants/:id - Get details of a specific restaurant.
- GET /api/restaurants/:id/availability - Check real-time table availability.
- POST /api/reservations - Create a new reservation.
- GET /api/users/:userId/reservations - Get a user's reservation history.

Restaurant Management Endpoints:

- POST /api/restaurant/login - Authenticate restaurant staff.
- GET /api/restaurant/reservations - View upcoming reservations.
- PUT /api/reservations/:id/status - Update reservation status (e.g., confirmed, seated, completed).

2. Request/Response Format

All API responses will follow a standardized JSON format.

Example: Successful Reservation Creation (POST /api/reservations)

Request Body:

```
{ "restaurantId": "RST123", "userId": "USER456", "partySize": 4, "reservationTime": "2023-11-15T19:30:00Z" }
```

Response Body (201 Created):

{ "success": true, "message": "Reservation confirmed", "data": { "reservationId": "RES789", "confirmationCode": "SDINE-7A8B9C", "status": "confirmed" } }

Error Response (400 Bad Request):

{ "success": false, "message": "No tables available for the selected time.", "errorCode": "NO_AVAILABILITY" }

API DESIGN & DATA MODEL (2)

3. Database Schema (Firestore Collections):

Users Collection:

userId (string, unique)

name (string)

email (string)

phoneNumber (string)

preferences (map)

Restaurants Collection:

restaurantId (string, unique)

name (string)

address (map)

cuisineType (array)

tableConfigurations (array of maps: {tableId, size})

operatingHours (map)

<u>Reservations Collection:</u>

reservationId (string, unique)

restaurantId (string)

userId (string)

partySize (number)

scheduledTime (timestamp)

status (string: e.g., "booked", "seated", "completed", "cancelled")

confirmationCode (string)

createdAt (timestamp)

<u>Waitlist Collection (for walk-ins):</u>

waitlistId (string, unique)

restaurantId (string)

customerName (string)

partySize (number)

phoneNumber (string)

estimatedWaitTime (number) // in minutes status (string: "waiting", "notified", "seated")

FRONT END UI/UX DESIGN PLAN

1. Wireframes & Navigation Flow

The application will follow a single-page application (SPA) architecture with a clear hierarchical flow.

Key Screens:

- Landing Page:

  Search bar hero section, featured restaurants.

- Search Results Page:

  Filterable list/grid of restaurants with key info (name, rating, ETA).

- Restaurant Detail Page:

  High-resolution images, menu (future), reviews, and the reservation booking widget showing live availability.

- User Dashboard:

  For users to view upcoming reservations, history, and profile settings.

- Restaurant Admin Dashboard:

For staff to manage reservations, view the waitlist, and update table statuses.

Navigation:

A persistent navigation bar will provide links to Home, Search, and User Profile.

2. State Management Approach

Given the dynamic nature of the app (live availability, user auth, shopping cart), a centralized state management solution is essential. We will use Redux or the Context API to manage:

- User State: Authentication status, user profile data.
- Restaurant State: Currently viewed restaurant, search results.
- Reservation State: Current booking process data (selected time, party size).
- UI State: Loading indicators, modal open/close states, notifications.

This ensures that the state is predictable and accessible across different components, leading to a more maintainable codebase.

DEVELOPMENT & DEPLOYMENT PLAN

1.Team Roles & Responsibilities

- Frontend Lead:

Responsible for developing the React.js user interface, ensuring responsiveness, and integrating with backend APIs.

- Backend Lead:

Responsible for designing and building the Node.js/Express server, defining API endpoints, and integrating with Firebase.

- Database & DevOps Lead:

Manages the Firestore database schema, security rules, and the deployment pipeline. Also handles integration of external APIs (Google Maps).

- UI/UX Designer (Shared Role):

All team members will contribute to wireframing and design decisions, guided by research on references like OpenTable and Eat App.

2. Git Workflow

We will adopt a Feature Branch Workflow to maintain a clean and stable main branch.

- main branch: Always contains production-ready code.
- develop branch: Integration branch for completed features.
- Feature Branches: Created from develop for each new feature (e.g., feature/user-authentication, feature/booking-system).

Process:

1. Create a feature branch.
2. Work on the feature and commit regularly.
3. Push the branch and create a Pull Request (PR) to develop.
4. Code review by at least one other team member is mandatory before merge.
5. After testing, develop is merged into main for a release.

3. Testing Approach

A multi-layered testing strategy will be implemented to ensure reliability.

- Unit Testing:

  Test individual functions and components in isolation. (Jest, React Testing Library).

- Integration Testing:

Test the interaction between the backend API and the database.

- End-to-End (E2E) Testing:

Automate critical user flows like the reservation process from search to confirmation. (Cypress or Playwright).

4. Hosting & Deployment Strategy

The application will be deployed as a full-stack project on a cloud platform for scalability and reliability.

- Frontend Hosting:

Vercel or Netlify.

These platforms offer excellent support for React applications with continuous deployment from the main branch.

- Backend Hosting:

Heroku or Railway.

These Platform-as-a-Service (PaaS) providers simplify the deployment of Node.js applications.

- Database & Infrastructure:

Google Firebase.

As our primary database and auth service, it is a fully managed, serverless solution, eliminating the need for server management.

The deployment pipeline will be automated: a push to the main branch will trigger a build and deployment process on the respective hosting platforms.

REFERENCES:

Eat App - Cloud-based reservation system with CRM and loyalty features (https://restaurant.eatapp.co)
OpenTable – Global leader in restaurant booking (https://www.opentable.com)
Tock – Prepaid reservation system (https://www.exploretock.com)
Firebase Documentation (https://firebase.google.com/docs)
Google Maps API (https://developers.google.com/maps/documentation)