**Restaurant Reservation System - Project Documentation**

**1. Introduction**

This document details the development and functionality of a modern web application designed to streamline restaurant reservation management. The system provides features for both customers (making reservations) and restaurant staff (managing reservations, sending notifications).

**1.1. Purpose**

The purpose of this application is to:

* Simplify the reservation process for customers.
* Provide a centralized platform for restaurants to manage bookings efficiently.
* Reduce manual errors and improve communication.
* Enhance the overall customer experience.

**1.2. Scope**

The system includes the following features:

* **Reservation Creation:** Customers can book tables online by providing their details and desired date/time.
* **Reservation Management:** Restaurant staff can view, update, and manage reservations (e.g., confirm, cancel, modify).
* **Real-time Status Updates:** The system provides up-to-date information on reservation status.
* **Email Notifications:** Automated emails are sent to customers to confirm, update, or cancel reservations.
* **Responsive Design:** The application is accessible and usable across various devices (desktops, tablets, and mobile phones).
* **Secure Data Handling:** Customer data and reservation information are stored securely.

**2. System Overview**

**2.1. System Architecture**

The system follows a modern web application architecture:

* **Frontend:** React + TypeScript (for the user interface)
* **Backend (Database):** Supabase (for data storage and management)
* **External Services:** EmailJS (for email notifications)

**2.2. Data Flow**

1. **Reservation Creation:**
   * Customer submits reservation details through the React frontend.
   * Frontend sends a request to Supabase to store the reservation data.
   * Supabase stores the data and returns a confirmation.
   * Frontend triggers an email notification via EmailJS.
2. **Reservation Management:**
   * Restaurant staff accesses the management interface (React frontend).
   * Frontend fetches reservation data from Supabase.
   * Staff updates reservation details (status, notes) through the frontend.
   * Frontend sends an update request to Supabase.
   * Supabase updates the data.
   * If necessary, the frontend triggers an email notification to the customer via EmailJS.

**3. Functional Requirements**

**3.1. Customer-Facing Features**

* **Reservation Form:**
  + Customers can enter their name, email, phone number, party size, reservation date, and reservation time.
  + Form validation ensures accurate data entry.
* **Reservation Confirmation:**
  + Customers receive an email confirmation upon successful reservation.
  + Confirmation includes reservation details.
* **Reservation Cancellation (Optional):**
  + Customers may have the option to cancel their reservations (depending on the implementation).

**3.2. Restaurant Staff Features**

* **Reservation Dashboard:**
  + Displays a list of all reservations with relevant details.
  + Provides filtering and sorting options (e.g., by date, status).
* **Reservation Details:**
  + Allows staff to view the complete details of a specific reservation.
* **Reservation Status Management:**
  + Staff can update the status of a reservation (e.g., "Pending," "Confirmed," "Cancelled," "Completed").
* **Notes:**
  + Staff can add internal notes to a reservation (e.g., special requests, dietary restrictions).
* **Email Notifications:**
  + The system automatically sends email notifications to customers for reservation confirmations, updates, and cancellations.

**4. Non-Functional Requirements**

* **Performance:** The application should be responsive and provide a smooth user experience.
* **Scalability:** The system should be able to handle a growing number of reservations.
* **Security:**
  + Customer data should be stored securely.
  + Authentication and authorization should be implemented to protect access to the management interface.
* **Usability:** The application should be intuitive and easy to use for both customers and staff.
* **Reliability:** The system should be reliable and minimize downtime.
* **Maintainability:** The codebase should be well-organized and easy to maintain.
* **Accessibility:** The application should be accessible to users with disabilities, adhering to accessibility guidelines (WCAG).
* **Responsiveness:** The application should be fully responsive and work seamlessly on different screen sizes and devices (desktops, tablets, and mobile phones).

**5. Technology Stack**

* **Frontend:**
  + **React:** A JavaScript library for building user interfaces.
  + **TypeScript:** A superset of JavaScript that adds static typing.1
  + **Shadcn UI:** A collection of reusable UI components built with Tailwind CSS and Radix UI.
  + **React Hook Form:** A library for building performant and flexible forms in React.
  + **Zod:** A TypeScript-first schema declaration and validation library.
  + **date-fns:** A modern JavaScript date utility library.
* **Backend (Database):**
  + **Supabase:** A Firebase alternative that provides a PostgreSQL database, authentication, real-time subscriptions, and storage.
* **Email Service:**
  + **EmailJS:** A service that allows sending emails directly from client-side applications without needing a backend server for email.

**6. Installation and Setup**

**6.1. Prerequisites**

* Node.js (v14 or higher)
* npm or yarn package manager
* Supabase account
* EmailJS account

**6.2. Installation Steps**

1. **Clone the Repository:**  
   Bash  
   git clone <repository-url>  
   cd <repository-name>
2. **Install Dependencies:**  
   Bash  
   npm install # or yarn install
3. **Set Up Environment Variables:**
   * Create a .env.local file in the project's root directory.
   * Add the following environment variables, replacing the placeholders with your actual values:  
     VITE\_SUPABASE\_URL=your\_supabase\_url  
     VITE\_SUPABASE\_ANON\_KEY=your\_supabase\_key  
     VITE\_EMAILJS\_PUBLIC\_KEY=your\_emailjs\_public\_key  
     VITE\_EMAILJS\_SERVICE\_ID=your\_emailjs\_service\_id  
     VITE\_EMAILJS\_TEMPLATE\_ID=your\_emailjs\_template\_id
4. **Database Setup (Supabase):**
   * Create a new Supabase project on the Supabase website.
   * Obtain the SUPABASE\_URL and SUPABASE\_ANON\_KEY from your Supabase project settings.
   * Use the Supabase SQL editor or a tool like DBeaver to run the following SQL script to create the reservations table:  
     SQL  
     create table reservations (  
      id uuid default uuid\_generate\_v4() primary key,  
      customer\_name text not null,  
      customer\_email text not null,  
      customer\_phone text not null,  
      party\_size integer not null,  
      reservation\_date date not null,  
      reservation\_time time not null,  
      status text not null default 'pending',  
      notes text,  
      created\_at timestamp with time zone default timezone('utc'::text, now()) not null,  
      updated\_at timestamp with time zone default timezone('utc'::text, now()) not null  
     );  
       
     -- Enable Row Level Security  
     alter table reservations enable row level security;  
       
     -- Create policies  
     create policy "Enable read access for all users" on reservations  
      for select using (true);  
       
     create policy "Enable insert for authenticated users only" on reservations  
      for insert with check (auth.role() = 'authenticated');  
       
     create policy "Enable update for authenticated users only" on reservations  
      for update using (auth.role() = 'authenticated');
5. **Email Setup (EmailJS):**
   * Create an EmailJS account.
   * Set up an email service (e.g., Gmail).
   * Create an email template with the following variables:
     + to\_email (recipient's email)
     + to\_name (recipient's name)
     + reservation\_status (status of the reservation)
     + notes (any additional notes)
   * Obtain the EMAILJS\_PUBLIC\_KEY, EMAILJS\_SERVICE\_ID, and EMAILJS\_TEMPLATE\_ID from your EmailJS dashboard.
6. **Start the Development Server:**  
   Bash  
   npm run dev # or yarn dev  
   * The application should now be running in your browser (usually at http://localhost:5173).

**7. Project Structure**

src/  
├── components/ # Reusable UI components  
│ ├── ui/ # Shadcn UI components (styled components)  
│ └── forms/ # Form components (e.g., reservation form)  
├── lib/ # Utility functions and libraries  
│ ├── supabase.ts # Supabase client setup  
│ └── utils.ts # General utility functions  
├── pages/ # Application pages  
│ ├── Reservations.tsx # Page for making reservations  
│ └── ManageReservations.tsx # Page for managing reservations (admin)  
├── types/ # TypeScript types  
│ └── index.ts # Type definitions for data models  
└── App.tsx # Main application component  
└── main.tsx # Entry point of the application  
└── index.html # HTML template  
└── vite-env.d.ts # TypeScript environment declarations

**8. API Reference**

**8.1. Supabase**

* **Create Reservation:**  
  TypeScript  
  import { supabase } from "@/lib/supabase";  
  import { Reservation } from "@/types"; // Assuming you have a Reservation type  
    
  async function createReservation(reservationData: Omit<Reservation, 'id' | 'status' | 'created\_at' | 'updated\_at'>) {  
   const { data, error } = await supabase  
   .from("reservations")  
   .insert([  
   {  
   ...reservationData,  
   status: "pending", // Default status  
   },  
   ])  
   .select();  
    
   if (error) {  
   console.error("Error creating reservation:", error);  
   throw error;  
   }  
    
   return data;  
  }
* **Update Reservation:**  
  TypeScript  
  import { supabase } from "@/lib/supabase";  
    
  async function updateReservation(id: string, updates: Partial<Pick<Reservation, 'status' | 'notes'>>) {  
   const { data, error } = await supabase  
   .from("reservations")  
   .update({  
   ...updates,  
   updated\_at: new Date().toISOString(),  
   })  
   .eq("id", id)  
   .select();  
    
   if (error) {  
   console.error("Error updating reservation:", error);  
   throw error;  
   }  
    
   return data;  
  }
* **Get All Reservations:**  
  TypeScript  
  import { supabase } from "@/lib/supabase";  
    
  async function getAllReservations() {  
   const { data, error } = await supabase  
   .from("reservations")  
   .select("\*");  
    
   if (error) {  
   console.error("Error fetching reservations:", error);  
   throw error;  
   }  
    
   return data;  
  }

**8.2. EmailJS**

* **Send Email:**  
  TypeScript  
  import emailjs from '@emailjs/browser';  
    
  const serviceId = import.meta.env.VITE\_EMAILJS\_SERVICE\_ID;  
  const templateId = import.meta.env.VITE\_EMAILJS\_TEMPLATE\_ID;  
  const publicKey = import.meta.env.VITE\_EMAILJS\_PUBLIC\_KEY;  
    
  async function sendEmail(params: { to\_email: string; to\_name: string; reservation\_status: string; notes: string; }) {  
   try {  
   const response = await emailjs.send(serviceId, templateId, params, publicKey);  
   console.log('Email sent successfully!', response);  
   } catch (error) {  
   console.error('Error sending email:', error);  
   }  
  }

**9. Error Handling**

The application implements error handling in several areas:

* **Form Validation:** React Hook Form and Zod are used to validate user input, preventing invalid data from being submitted.
* **Database Operations:** Supabase client calls are wrapped in try...catch blocks to handle potential errors during database interactions. Error messages are logged to the console, and appropriate feedback is provided to the user.
* **Email Sending:** EmailJS send calls are also handled with try...catch to manage potential email sending failures.
* **Network Requests:** Fetch API calls are handled with error checking to manage network connectivity issues.
* **User Feedback:** The UI provides informative error messages to the user when errors occur, helping them understand and resolve issues.

**10. Contributing**

1. **Fork the repository** on GitHub.
2. **Create your feature branch:**  
   Bash  
   git checkout -b feature/amazing-feature
3. **Commit your changes:**  
   Bash  
   git commit -m 'Add some amazing feature'
4. **Push to the branch:**  
   Bash  
   git push origin feature/amazing-feature
5. **Open a Pull Request** on the original repository.

**11. License**

This project is licensed under the MIT License. See the LICENSE file for details.

**12. Support**

For support, please:

* Open an issue in the GitHub repository.
* (Optional: Provide an email address for direct contact)

**13. Developer Profile: Sai Kumar Thota**

* **Name:** Sai Kumar Thota
* **Role:** Full-Stack Developer
* **Skills:**
  + **Frontend:** React, TypeScript, JavaScript (ES6+), HTML, CSS, Tailwind CSS, Shadcn UI, React Hook Form, Zod, Redux (or similar state management), Responsive Design, Accessibility (WCAG), Testing (Jest, React Testing Library)
  + **Backend:** Node.js, Express.js (or similar framework), RESTful APIs
  + **Database:** PostgreSQL, Supabase, SQL
  + **Version Control:** Git, GitHub
  + **Build Tools/Package Managers:** npm, yarn, Vite
  + **Other:** Agile methodologies, Software Design Patterns, Problem Solving, Communication
* **Experience:** (Provide a brief summary of relevant work experience, projects, or contributions. Include any links to portfolio, GitHub, etc.)
* **Education:** (Mention your educational background)
* **Projects:** (List key projects you've worked on, highlighting your contributions and the technologies used)
* **GitHub:** (Link to your GitHub profile)
* **Portfolio:** (Link to your online portfolio, if available)
* **Summary:** A highly motivated and skilled Full-Stack Developer with a passion for building robust and user-friendly web applications. Proven ability to work effectively in team environments and deliver high-quality solutions. Eager to contribute to challenging projects and continue learning and growing as a developer.

**Important Notes:**

* Replace the placeholder values (e.g., <repository-url>, your\_supabase\_url) with your actual information.
* This documentation provides a comprehensive overview. You may need to add more specific details depending on the complexity of your project.
* Remember to keep your documentation up-to-date as you make changes to your application.
* This is a template, please fill in the specific details of Sai Kumar Thota's profile.

**Sources**

1. <https://github.com/san-script/financial-dashboard-assignment>

2. <https://github.com/alvarorom3/mockup-atlas-gym>

3. <https://github.com/AlbertKhai/f8-fullstack-offline>