

Room Rent Management System - Development Guide

This guide documents the development process, architecture, and key features of the Room Rent Management Application. It serves as a reference for understanding the codebase and for future development.

1. Project Overview

The **Room Rent Management System** is a modern, responsive web application designed to help landlords manage room rentals. It features a dark-themed UI, real-time data persistence using Firebase Firestore, and a dashboard for tracking room status.

Tech Stack

- **Frontend Framework:** React (v18)
- **Build Tool:** Vite
- **Styling:** Vanilla CSS (with CSS Variables for theming)
- **Icons:** Lucide React
- **Database:** Firebase Firestore
- **Hosting:** Firebase Hosting

2. Project Setup

The project was initialized using Vite for a fast development environment.

```
# Command used to create the project
npx create-vite@latest room_rent_app --template react

# Navigate to directory
cd room_rent_app

# Install dependencies
npm install
npm install lucide-react date-fns firebase
```

3. Project Structure

```

room_rent_app/
├── src/
|   ├── assets/      # Static assets (images, global css)
|   |   └── logo.png  # App logo
|   ├── components/ # Reusable UI components
|   |   ├── AddRoomForm.jsx # Modal form to add new rooms
|   |   ├── BookingModal.jsx # Modal to book a room
|   |   └── RoomCard.jsx  # Card component displaying room info
|   ├── App.jsx      # Main application logic and layout
|   ├── main.jsx     # Entry point
|   ├── index.css    # Global styles and theme variables
|   └── firebase.js  # Firebase configuration and initialization
├── firebase.json   # Firebase Hosting configuration
└── .firebaserc     # Firebase project alias
└── package.json    # Project dependencies

```

4. Key Features & Implementation

A. Global State Management

The application uses React's useState and useEffect hooks in App.jsx to manage the state of rooms and bookings.

- **rooms**: An array of room objects fetched from Firestore.
- **bookingRoom**: Tracks which room is currently being booked (triggers the modal).

B. Firebase Integration (Database)

Instead of local storage, we use **Firebase Firestore** for real-time data persistence.

Configuration (src/firebase.js):

Initializes the Firebase app and exports the Firestore database instance (db).

Real-time Updates (App.jsx):

We use onSnapshot to listen for changes in the rooms collection. This ensures that if multiple users are viewing the app, updates appear instantly without refreshing.

```

useEffect(() => {
  const q = query(collection(db, 'rooms'), orderBy('name'));
  const unsubscribe = onSnapshot(q, (snapshot) => {
    const roomData = snapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data()
    }));
    setRooms(roomData);
    setLoading(false);
  });
  return () => unsubscribe();
}, []);

```

CRUD Operations:

- **Create:** addDoc adds a new room document.
- **Update:** updateDoc modifies a room's status (e.g., 'available' -> 'occupied') and adds booking details.

C. Component Architecture

1. RoomCard.jsx:

- Displays room details (Name, Type, Price).
- Shows status badges (Available/Occupied).
- Calculates "Days Left" for occupied rooms using date-fns.
- Handles "Book Now" and "Check Out" actions.

2. BookingModal.jsx:

- A modal dialog for entering tenant details.
- Includes validation for dates (Check-out must be after Check-in).
- Uses a backdrop blur effect for a modern look.

3. AddRoomForm.jsx:

- A modal triggered from the header to add new rooms.
- Collects Room Name, Type (Standard/Deluxe/Suite), and Price.

5. Styling & Design System

The app features a custom **Dark Navy Theme** implemented via CSS Variables in src/index.css.

Color Palette

- **Surface:** #0f172a (Dark Navy)
- **Primary:** #06b6d4 (Cyan/Sky Blue)
- **Success:** #10b981 (Emerald Green)
- **Danger:** #ef4444 (Red)
- **Text:** #f8fafc (Off-white)

Utility Classes

We created custom utility classes (similar to Tailwind CSS) for rapid development:

- Layout: .flex, .grid, .gap-4, .justify-between
- Spacing: .p-4, .m-4, .mb-8
- Typography: .text-xl, .font-bold, .text-muted
- Visuals: .rounded-xl, .backdrop-blur-sm, .bg-gradient-to-r

6. Deployment (Firebase Hosting)

The application is deployed as a Single Page Application (SPA) on Firebase Hosting.

Deployment Steps:

1. **Build the Project:**

```
npm run build
```

This creates a dist folder with optimized static files.

2. Login to Firebase:

```
npx firebase-tools login
```

3. Initialize/Configure:

Created firebase.json to point to the dist folder and handle rewrites for SPA routing.

4. Deploy:

```
npx firebase-tools deploy
```

Live URL: <https://room-rent-management-app.web.app/> (<https://room-rent-management-app.web.app/>)

7. Future Improvements

Potential features to add in the future:

- **Authentication:** Allow only authorized admins to add/edit rooms.
- **Search & Filter:** Filter rooms by type, price, or availability.
- **Booking History:** Keep a log of past bookings.
- **Image Upload:** Allow uploading photos for each room.

8. Troubleshooting

Common Issues

1. "Missing or insufficient permissions" Error

- **Cause:** This happens when Firestore Security Rules block the request.
- **Fix:** Go to Firebase Console > Firestore Database > Rules and change allow read, write: if false; to allow read, write: if true; (for development only).

2. "firebase: command not found" or "could not determine executable"

- **Cause:** The Firebase CLI is not installed globally or npx is having trouble resolving the package.
- **Fix:** Use the full package name with npx:

```
npx firebase-tools login  
npx firebase-tools deploy
```

3. Changes not showing on the live site

- **Cause:** You might be seeing a cached version or forgot to build before deploying.
- **Fix:** Always run npm run build before running npx firebase-tools deploy.

9. Maintenance

Updating the App

1. Make changes to your code locally.
2. Test the changes: `npm run dev`
3. Build the production bundle: `npm run build`
4. Deploy the updates: `npx firebase-tools deploy`

Backup

- Firestore automatically handles data redundancy, but you can export your data from the Google Cloud Console if needed for manual backups.