SportsHub

🏗️ System Architecture

1. Overview

This document outlines the architectural structure of the Community Sports Facility Management System. It defines the system's layers, components, and technologies, along with justifications for architectural decisions. The objective is to ensure a scalable, secure, and maintainable web application that facilitates facility bookings, issue reporting, and role-based access for different users.

2. Key Stakeholders

Role	Responsibility
Resident	Book and review facilities
Facility Staff	Manage maintenance and issue reports
Admin	Oversee operations, approve bookings

3. Architectural Approach: Multi-Tier Architecture

To ensure clear separation of concerns and scalability, the system adopts a **Multi-Tier Architecture** consisting of the following layers:

a. Presentation Layer (Frontend)

- **Technology:** React.js + TailwindCSS
- **Responsibilities:**
    - Render dynamic interfaces
    - Handle client-side logic and routing
    - Display data retrieved from backend APIs
    - Responsive design for multiple user roles
- **Key Features:**

    - Role-based dashboards
    - Booking calendar and forms
    - Real-time updates with Firestore

---

b. Application Logic Layer (API Services)

- **Technology:** Firebase Cloud Functions
- **Responsibilities:**
    - Act as the business logic and data processing layer
    - Serve secure APIs to frontend
    - Integrate with Firebase Auth and Firestore
    - Manage role-based access control

---

c. Authentication & Authorization

- **Technology:** Firebase Auth + OAuth 2.0
- **Responsibilities:**
    - Provide secure login (Google OAuth)
    - Enforce role-based access
    - Manage sessions and identity lifecycle

---

d. Data Layer (Database)

- **Technology:** Firebase Firestore (NoSQL)
- **Responsibilities:**
    - Store user data, bookings, facility info, and feedback
    - Sync data in real-time with frontend
    - Serve as the source of truth for the application

---

e. Hosting & CI/CD

- **Technology:** Microsoft Azure + GitHub Actions
- **Responsibilities:**
    - Host frontend as static site via Azure Static Web Apps
    - Run automated deployment pipelines
    - Integrate testing before every production push
    - Maintain compliance with South African data residency laws

---

4. Data Flow & Integration

User Requests Flow:

1. A user (Resident, Staff, Admin) interacts via React frontend.
2. Authentication is verified via Firebase Auth.
3. Valid requests hit Cloud Function APIs for processing.
4. Business logic is executed and data is read/written to Firestore.
5. Response is returned and rendered in the UI.

---

5. Security Considerations

- All API endpoints are protected using Firebase security rules.
- OAuth 2.0 enables secure sign-ins and token-based access control.
- Real-time database rules prevent unauthorized data access.
- Continuous vulnerability scanning via GitHub code scanning.

---

6. Scalability & Maintainability

- **Serverless Architecture:** Reduces operational overhead and scales automatically.
- **Modular Codebase:** React components and Firebase functions are reusable and independently deployable.
- **CI/CD Pipeline:** Enables faster, safer deployments.

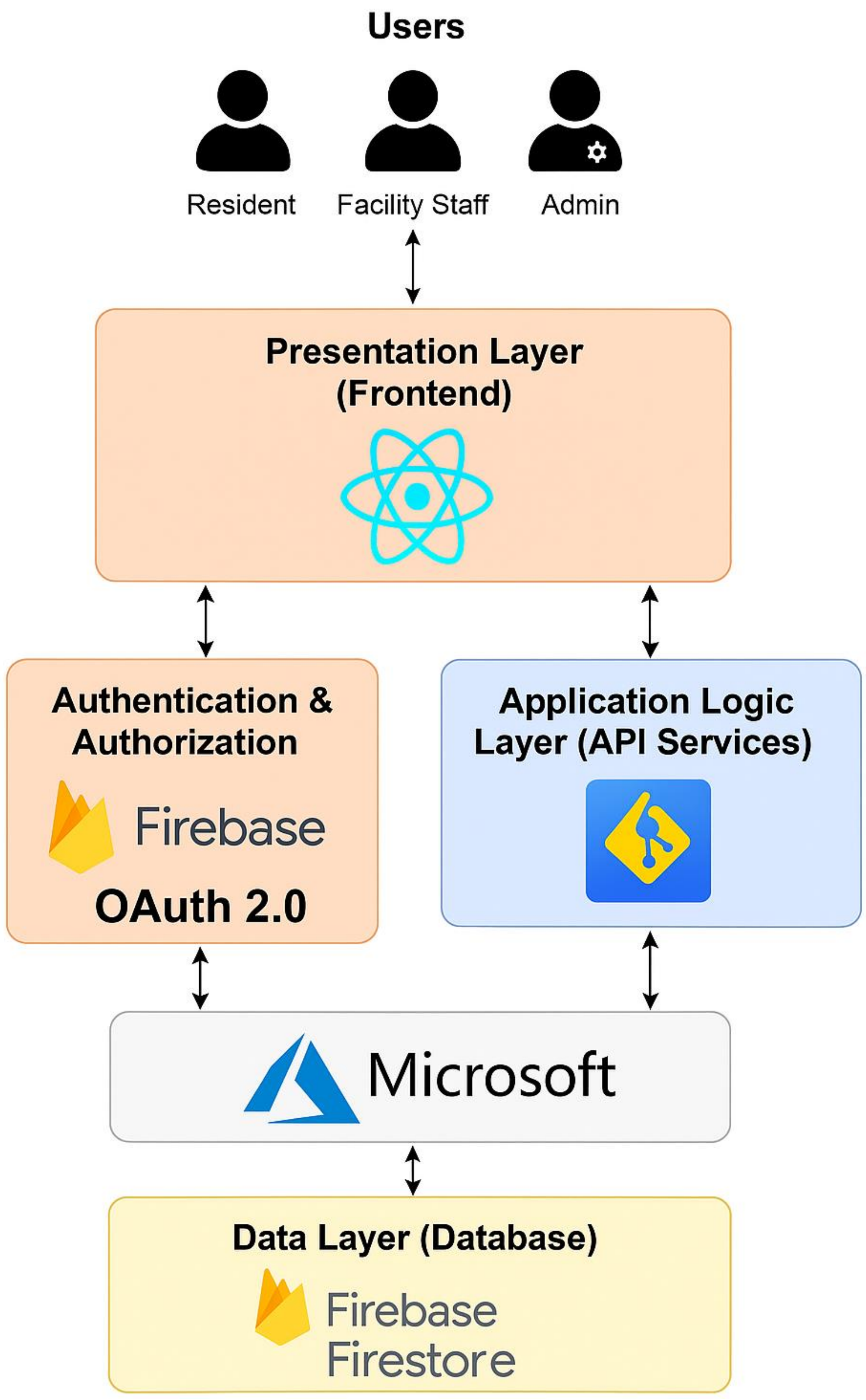
---

7. Technology Stack Summary

Layer	Technology
Frontend	React.js,
Auth	Firebase Authentication, OAuth 2.0
Backend API	Firebase Cloud Functions
Database	Firebase Firestore
Hosting	Microsoft Azure Static Web Apps
DevOps	GitHub Actions, Notion

---

8. Diagram


Users
Resident
Facility Staff
Admin
Presentation Layer (Frontend)
Authentication & Authorization
Firebase
OAuth 2.0
Application Logic Layer (API Services)
Microsoft
Data Layer (Database)
Firebase Firestore

9. Future Considerations

- Introduce caching mechanisms with Redis for performance optimization.
- Expand analytics capabilities using BigQuery or Google Analytics.
- Explore Progressive Web App (PWA) features for offline support.
