Process of Project Development

🏛️ System Architecture & Technology Planning

🎯 Project Foundation and Technological Direction

At the outset of the project, our focus centredaround selecting a robust and scalable architecture that could support the core requirements of a community sports facility management system. This entailed carefully choosing the **technology stack**, defining the **system architecture**, and ensuring the **UI/UX design** would support an intuitive and responsive user experience.

▯ Adopted System Architecture: Multi-Tier Design

We opted for a **multi-tiered architecture**, which cleanly separates concerns across three distinct layers:

- **Presentation Layer (Frontend)**
- **Application Logic Layer (API Services)**
- **Data Layer (Database)**

This structure not only improves maintainability but also supports scalability and independent development across different system components.

⚛️ Frontend: React.js

Our frontend is developed using **React.js**, a widely adopted JavaScript library for building responsive and performant **Website.**

- **Benefits of React for this project:**
  - Component reusability and modularity
  - Virtual DOM for fast UI rendering
  - Strong ecosystem and integration with state management libraries
  - Ideal for dynamic UIs such as booking calendars and real-time updates

---

🔐 Authentication & Authorization

User identity and access management are critical for this application. We integrated **Firebase Authentication** with **OAuth 2.0** to enable secure, role-based access control across three roles:

- Resident
- Facility Staff
- Admin

This approach provides a robust, scalable solution without the need to self-host complex identity infrastructure.

🔧 Backend: API Layer via Firebase Cloud Functions

Rather than using Node.js and Express.js, our backend logic is abstracted into **Firebase Cloud Functions**, which allow us to create secure, scalable RESTful APIs without managing dedicated servers.

- **Why Cloud Functions?**
  - Serverless: No backend infrastructure to manage
  - Auto-scalable: Grows with traffic without manual intervention
  - Real-time triggers integrated with Firestore
  - Tight integration with Firebase Auth and Firestore

🗄️ Database: Firebase Firestore (NoSQL)

Our data is stored in **Google Firebase Firestore**, a NoSQL document-based database chosen for its scalability, high availability, and real-time data synchronization.

- **Key advantages:**
  - Enables real-time UI updates (e.g., booking confirmations or event changes)
  - Scales well with high concurrent users
  - Seamlessly integrates with Firebase Auth and Cloud Functions

Firestore's ability to sync data across clients instantly enhances the responsiveness of the system, especially for time-sensitive actions like facility bookings and live maintenance updates.

☁ Hosting & Deployment: Azure Web Services

The web application is deployed via **Microsoft Azure**, selected based on our team's familiarity and the platform's robust support for:

- Hosting static React.js apps using Azure Blob Storage or Static Web Apps
- Integrations with **GitHub Actions** for CI/CD pipelines
- Compliance with local data residency requirements (hosted within South Africa)

This setup ensures low-latency access for users affiliated with the University of the Witwatersrand and offers easy scalability and managed security updates.

▯ Finalized Technology Stack

Layer	Technology
Frontend	React.js, HTML5, CSS3
Authentication	Firebase Auth, OAuth 2.0
Backend API	Firebase Cloud Functions
Database	Firebase Firestore (NoSQL)
Hosting & CI/CD	Azure Static Web Apps, GitHub Actions

☑ Summary

This architecture enables a **fast, scalable, and maintainable** system tailored for modern web applications. It supports our goals of real-time updates, modular deployment, and agile feature delivery—perfectly suited for our community-focused solution.

Here's a **rephrased, enriched, and technically aligned** version of the "Initial Development Phase" and "Tools and Technologies Used" sections, updated to better reflect best practices in Agile/Scrum, React development, and modern tooling:

🚀 Development Phases

🏢 Groundwork with Agile & Scrum

Our project adopted the **Agile methodology**, specifically leveraging the **Scrum framework**, to enable incremental development, continuous feedback, and iterative delivery. Scrum allowed us to break down the project into manageable sprints, each culminating in a working software increment and a retrospective for improvement.

At the start, the team collaborated to define the **scope and vision** of the application. We established a shared understanding of the functional goals and user expectations. Once aligned, we began preparing our development environment by completing several key setup activities:

- 📄 **Version Control**: A centralized GitHub repository was created for code management and team collaboration.
- 🗄️ **Initial Deployment**: A basic React.js web app was deployed to Microsoft Azure to establish our hosting environment and test continuous deployment flows.
- □ **Testing Framework Initialization**: The **Jest** testing environment was configured early to ensure we could begin development using a **Test-Driven Development (TDD)** approach.
- 🗂️ **Project File Structure**: We designed a modular folder structure suitable for scalable React applications.
- 🎨 **UI/UX Design Foundations**: Wireframes and UI mockups were developed collaboratively using feedback from our stakeholders. These visual references guided early frontend development and helped identify key interaction flows.

Early stakeholder interaction was a cornerstone of this phase. Our client provided input on aesthetics and usability—suggesting, for example, a shift in UI colour palette for a fast, responsive interface development.

🛠️ Tools & Technologies Utilization

To support our development process, we are utilizing a modern toolset designed for seamless collaboration, automated deployment, and transparent progress tracking:

Tool	Purpose
GitHub	Codebase hosting, pull request reviews, version control
GitHub Actions	Automated CI/CD pipeline to test, build, and deploy our application after every commit
Notion	Organizes projects, notes, and more in a single, unified platform. It also excels at real-time collaboration and offers a user-friendly interface

Tool	Purpose
WhatsApp	Lightweight communication for quick coordination, reminders, and asynchronous daily standups
Discord	Primary platform for all **Scrum ceremonies**, including Sprint Planning, Daily Standups, Reviews, and Retrospectives

---

□ Benefits Realized

- **Clear team alignment** through early planning and backlog creation
- **Faster feedback loops** by using CI/CD and test automation from day one
- **Increased stakeholder engagement** via collaborative UI prototyping
- **Adaptability to change**, such as refining designs based on client feedback before development began

---