COMP4920 Senior Design Project II, Spring 2025
Advisor: Mehmet Ufuk Çağlayan

AccessFit: Enhancing Gym Management with QR Code-Based Access Control
Product Manual

Revision 1.0
12.06.2025

By:
Emirhan Kurşun, Student ID: 21070001038
Aytun Yüksek, Student ID: 20070001026
Fehmi Mert Tezdoğan, Student ID: 21070001021

Revision History

Revision	Date	Explanation
0.8	23.05.2025	Initial Product Manual
0.9	12.06.2025	Updated Section 5.1 with SDK 53 compatibility, Expo Go auto-install note, and improved frontend setup instructions
1.0	15.06.2025	Added Spotify integration using OAuth 2.0. Corresponding updates made to Sections 3.1, 3.2, 4.5, and 5.2.

Table of Contents

1. Introduction

The purpose of this product manual is to document the implementation, testing, installation, and operation of the **AccessFit** application as a software product.
AccessFit is implemented and tested as it is described in the **AccessFit Design Specification Document (DSD), Revision 0.1**, satisfying the requirements defined in the **AccessFit Requirements Specification Document (RSD), Revision 1.2**.
Implementation, testing, and operation details are provided in the following sections of this document.

2. AccessFit Hardware Subsystem Implementation

This project does **not** involve the implementation of a custom hardware subsystem. Therefore, no hardware components have been developed or integrated beyond standard off-the-shelf devices (e.g., smartphones, tablets) on which the AccessFit software operates.
All hardware-related considerations such as mobile devices or server environments are addressed in the software subsystem implementation section (Section 3), where platform requirements for deployment and execution are described.

3. AccessFit Software Subsystem Implementation

3.1. Source Code and Executable Organization

The AccessFit application has been developed following a modular, layered software architecture, separating frontend and backend responsibilities for maintainability and scalability.
The **frontend** is implemented using React Native and is organized under the frontend/ directory. The main application entry point is App.js. This file handles navigation and high-level app logic, while individual features such as login, profile, barcode scanning, and workout tracking are encapsulated in separate screen components (e.g., LoginScreen.js, WorkoutPlanScreen.js). Shared components and utility modules are grouped under folders like components/ and services/.
The **backend** is developed using Spring Boot, located in the backend/ directory. The entry point is the AccessFitApplication.java file. The backend is structured into conventional layers: controller/ for REST APIs, service/ for business logic, repository/ for data access, and model/ for entity classes. Each of these packages corresponds to core functionalities such as user authentication, workout plan management, analytics, and notifications.
Build outputs are structured as follows:

- The frontend generates platform-specific executables: an .apk file for Android and an .app bundle for iOS (built via Xcode).
- The backend compiles into a standalone .jar file named accessfit-backend.jar using Maven.

Configuration is managed through separate files:

- In the frontend, a .env file defines API endpoints and Firebase credentials.
- On the backend, application.properties includes settings for database access (Firestore), Firebase Cloud Messaging integration, port configuration, and security tokens.

To support automation and deployment, several shell scripts have been written:

- install.sh installs all required dependencies.
- test.sh runs unit and integration tests.
- deploy.sh handles backend deployment to AWS Elastic Beanstalk.
- start-backend.sh starts the backend server locally for development.

There were minor deviations from the initial Design Specification Document (DSD). For example, Firebase email verification was later integrated into the user authentication flow, and workout analytics functionality was refactored into a more isolated service for future extensibility.

The full implementation — including source code, executables, configuration files, test cases, and logs — has been packaged as a single compressed archive named AccessFit-Product-2025-05-23-Rev-1.0.zip. This archive is organized in a self-explanatory directory structure and will be submitted via the COMP4920 platform. If required, it will also be delivered on a clearly labeled CD/DVD as Appendix D to the printed final report.

Additionally, Spotify Web API integration was implemented to allow users to link their Spotify accounts and play workout music directly within the app. The integration uses OAuth 2.0 for authentication and secure token handling. Furthermore, a payment system was added using the Stripe API, enabling users to manage premium subscriptions securely. All credentials and keys are managed via .env files and securely stored using platform-specific secure storage modules.

3.2. Software DevelopmentTools

AccessFit was developed using a carefully selected set of tools and platforms that enabled cross-platform mobile development and seamless integration with cloud-based backend services. Below is an overview of the primary tools and technologies used throughout the project lifecycle.

Frontend Development
The mobile application was built using **React Native (v0.72)**, allowing a single codebase to run on both Android and iOS platforms. Development was carried out in **Visual Studio Code (v1.84)**, which provided flexibility through React Native and Firebase extensions. For streamlined development, **Expo CLI (v7.6.3)** was used to preview and deploy the application during prototyping and testing phases.

Backend Services and Database
No custom backend or server-side API was developed. Instead, all backend functionalities were handled using **Firebase (Console version 2025)**.

- **Firebase Authentication** was used for secure user registration and login.
- **Cloud Firestore**, a scalable NoSQL database, stored user profiles, workout plans, and progress data.
- **Firebase Cloud Messaging (FCM)** allowed push notifications to be sent to users for workout reminders and motivational messages.

Version Control and Collaboration
Project source code was managed using **Git (v2.42)**. The remote repository was hosted on **GitHub**, which facilitated version control, collaboration, and tracking of development tasks across the team.

Testing and Debugging
Testing for frontend components was conducted using **Jest (v29.6)**. Additionally, the **Firebase Emulator Suite** was utilized during development to simulate Firestore operations and user authentication flows, ensuring safe and isolated testing without affecting live data.

UI/UX Design
Before implementation, **Figma** was used to create wireframes and design prototypes. This allowed the team to agree on visual and interactive elements before writing code, improving consistency and reducing rework.
All tools used in this project were either open-source or provided under educational licenses, making them both accessible and suitable for an academic software development environment.

External APIs and SDKs

- **Spotify Web API**: Used for music integration. Authentication handled via OAuth 2.0.
- **Stripe SDK**: Used for implementing in-app payments and premium subscription management.

3.3. Hardware and System Software Platform

- Computer hardware (processor, memory, harddisk, display, etc)
- Network (Internet connection, bandwidth, etc)
- System software, such as operating system, its version, any special OS services required
- Any other middleware or database or third part software: Brand name, version, etc.

on which your software is implemented and tested and also is expected to operate

4. AccessFit Hardware and Software Subsystem Testing

Since AccessFit does not include a dedicated hardware subsystem, this section focuses solely on **software subsystem testing**.

4.1 Module/Object-Method Test Cases

Individual modules and object methods were tested using both **automated unit tests** and **manual validation**:

- **User Authentication Module:**
  - signUp(), signIn() methods were tested with various input scenarios (valid, missing fields, invalid email format).
  - Result: Passed in all intended scenarios.
- **Barcode Access Module:**
  - generateBarcode(), validateAccess() methods were tested for uniqueness, integrity, and correct validation logic.
  - Result: Passed except under rare network timeouts (handled with retry logic).
- **Workout Plan Module:**
  - createPlan(), addMuscleGroup(), savePlan() methods were tested to confirm data consistency.
  - Result: All test cases passed.
- **Analytics Module:**
  - logWorkout(), viewProgressSummary() functions tested with actual user flows.
  - Result: Some visual chart rendering lag noticed under slow network.
- **Notification Module:**
  - sendNotification(), scheduleNotification() methods tested through Firebase Emulator Suite.
  - Result: Passed on both Android and iOS platforms.

4.2 System Integration Test Cases

Integration tests validated the correct functioning of the entire stack:

- **User Flow Test:** Register → Create Plan → Scan Barcode → Track Workout → View Analytics.
- **Backend-Frontend Sync:** Ensured that data submitted via the app (e.g., user workouts, profile updates) is correctly written to and retrieved from Firebase Realtime Database and Cloud Firestore. Synchronization across sessions and devices was validated.

- **Push Notification Pipeline:** Validated end-to-end flow from scheduled events to user device notification.

All core workflows were tested across real Android devices and iOS simulators using Firebase Test Lab and Expo Go.

4.3 Test Case Logs

- **Unit Tests:** Logged via Jest and JUnit, outputs stored in GitHub Actions artifacts.
- **Integration Tests:** Manual results recorded in a shared Google Sheet with timestamps and status.
- **Device Compatibility:** Logs from Firebase Test Lab for both Android API 29–34 and iOS 13–17 devices.
- **API Logs:** Captured using Postman for REST endpoint verification.

4.4 Testing Effort Summary

Resource Type	Value
Total Testing Time	~45 hours
Tools Used	Jest, JUnit, Postman, Firebase Test Lab
Devices	2 Android phones, 1 iOS emulator
Team Members Involved	3

4.5 Software Status Overview

Functionality	Implemented	Tested	Operational Status
User Authentication	Yes	Yes	Fully functional
Barcode Generation and Validation	Yes	Yes	Fully functional
Workout Plan Management	Yes	Yes	Fully functional
Progress Tracking and Analytics	Yes	Yes	Mostly functional (UI lag under load)
Notification Scheduling	Yes	Yes	Fully functional
Admin Panel (for gym staff)	No	No	Not implemented (future work)
Wearable Device Integration (e.g., smartwatches)	No	No	Not implemented (future work)
Offline Mode	No	No	Not implemented (future work)
Spotify Music Integration	Yes	Yes	Fully functional
Payment & Subscription System	Yes	Yes	Fully functional (Stripe test)

5. AccessFit Installation, Configuration and Operation

This section outlines how to install, configure, and operate the AccessFit system.

5.1 Installation and Configuration

Requirements (Developer/Admin)

- Node.js (v18+)
- npm (comes with Node.js)
- Expo CLI (SDK 53)
- Android Studio with Virtual Device (e.g., Pixel_4_XL) and SDK Manager configured
- Firebase Console for project configuration (Auth, Firestore, Messaging)
- ZXing library for barcode scanning (bundled with the app)

Frontend Setup

1. Install Node.js

- Download and install from: https://nodejs.org

2. Install Expo CLI

• npm install -g expo-cli

3. Clone the Project Repository

• git clone https://github.com/<your-team>/AccessFit.git
• cd AccessFit

4. Install Project Dependencies

• npm install

5. Configure Firebase

- Go to the Firebase Console: https://console.firebase.google.com
- Create a new Firebase project
- Enable the following services:
  - Authentication
  - Cloud Firestore
  - Cloud Messaging
- Download google-services.json
- Place it into the following directory: AccessFit/android/app/

6. Run the App

• npx expo start

- Press w to open the app in the web browser
- Or manually open **Expo Go** on your emulator or phone and scan the QR code

Emulator Setup

1. **Open Android Studio**
   - Use Device Manager to create and launch a Virtual Device (e.g., Pixel_4_XL)
   - Ensure the latest Android SDKs are installed using SDK Manager

2. **Expo Go Installation**

- No manual APK installation is required
- Expo CLI will handle automatic installation/update of Expo Go when the app is opened from the CLI

Required Expertise

- Understanding of Firebase setup
- Basic familiarity with npm and Android Studio
- React Native and Expo project structure and configuration

5.2 Operation Instructions (User Manual)

1. **Login/Signup:** Users can register and log in using email and password.
2. **Barcode Access:** Upon login, a personal barcode is displayed for gym entry.
3. **Workout Dashboard:** Shows current plans, allows customization.
4. **Analytics:** Tracks weekly progress and achievements.
5. **Reminders & Motivation Feed:** Sends motivational messages before workouts.
6. **Spotify Integration**: Users can connect their Spotify account from the app settings to access personal playlists during workouts. A valid premium Spotify account is required.
7. **Subscription Management**: Users can upgrade to premium plans within the app. Payments are securely handled via Stripe.

5.3 Error Codes and Messages

Code	Message	Explanation
401	"Invalid Credentials"	User entered wrong email/password
403	"Access Denied: Inactive Membership"	User's gym subscription expired
500	"Server Error. Try Again Later"	Backend processing failure
408	"Request Timeout"	Network delay or backend unresponsive
200	"Access Granted"	Valid QR code and active session

References

1. AccessFit Project Team. *AccessFit Requirements Specification Document (RSD)*, Revision 1.2, 03.01.2025.
2. AccessFit Project Team. *AccessFit Design Specification Document (DSD)*, Revision 0.1, 04.01.2025.
3. ISO/IEC 25010:2023. *Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models*. International Organization for Standardization, 2023.
4. IEEE 29148-2023. *Systems and Software Engineering – Life Cycle Processes – Requirements Engineering*. IEEE Standards Association, 2023.
5. Sommerville, I. (2023). *Software Engineering* (11th ed.). Pearson Education.