

# SOFTWARE REQUIREMENT SPECIFICATION






---



Rome was Built  
in a Day



## HIIT GYM MANAGER GOOD X SOFTWARE

-  Vansh Sood (u23534402)
-  Denis Woolley (u23528860)
-  Jared Hürlimann (u23543932)
-  Jason Mayo (u23587572)
-  Amadeus Fidos (u22526162)

# Introduction

The HIIT Gym Management Software addresses a critical business need for HIIT-focused fitness studios: simplifying complex, error-prone administrative workflows so owners and coaches can concentrate on delivering high-quality training. Without specialized tools, gym operators juggle membership sign-ups, billing, class scheduling, attendance tracking, and member communications manually- tasks that consume time, introduce costly mistakes, and undermine member satisfaction.

This project will deliver a unified, user-friendly platform that centralizes every aspect of HIIT gym operations.

- **Members** will be able to register, manage their subscriptions, view and book classes, and track workout results from a mobile app.
- **Coaches** gain an intuitive interface for publishing daily workouts, overseeing assigned sessions, and reviewing performance data.
- **Managers** and administrators access a web-based dashboard to oversee pending and active members, configure weekly class schedules, assign coach roles, and monitor capacity and attendance trends.

By integrating payment processing, real-time leaderboards, and announcements, the system not only streamlines daily tasks but also drives member engagement and retention. Leveraging data insights -such as attendance reports, performance trends, and class popularity-gym operators can make informed decisions to grow their business sustainably. The scope includes user authentication and role management, membership credit handling, class management and booking, structured workout creation, live classes and leaderboards, administrative overviews, advanced analytics, gamification features, and more.

By adhering to agile practices and CI/CD guidelines, our team gained hands-on experience in modern web/mobile frameworks, PostgreSQL schema design, background job processing, and secure API development.

## **Deliverable:**

A deployed, fully functional HIIT Gym Manager system comprising

1. A background worker daemon for scheduled tasks and notifications
2. A RESTful API backend
3. A React Native mobile app for members and coaches
4. A Next.js web app for managers and administrators



# Functional Requirements Specification

**Client:** CrossFit Box

**Mentor:** Johan Bloem

**Delivery Deadline:** 24 October 2025

---

## R1: Authentication & Role-Based Access

### R1.1: User Registration & Login

- R1.1.1: Members register and login via mobile (email, password).
- R1.1.2: Coaches login via mobile (provisioned by admin/manager).
- R1.1.3: Managers & Admins login via web (single shared manager account + individual admin accounts).

### R1.2: Logout & Session Management

- R1.2.1: All roles can logout.

### R1.3: Role Management (Manager/Admin only)

- R1.3.1: Create, edit, or revoke Coach and Admin accounts.
  - R1.3.2: Assign roles to users.
- 

## R2: Member Onboarding & Profile

### R2.1: Member Profile

- R2.2.1: View personal details.
  - R2.2.2: Track personal progress, attendance history and PRs.
  - R2.2.3: Choose to opt-in or opt-out of public leaderboards
- 

## R3: Class Booking (High Priority)

### R3.1: Class Catalogue & Details

- R3.1.1: View upcoming classes (date, time, coach, capacity, workout type).
- R3.1.2: Join live classes when they start

### R3.2: Member Booking & Cancellation

- R3.2.1: Book classes against active subscription or class-credit balance (enforce pay-first model).
- R3.2.2: Cancel within window
- R3.2.3: Prevent double-booking (same time slot) and capacity overflow.



## **R4: Class Management & Setup (High Priority)**

### **R4.1: Class Setup & Scheduling (Manager)**

- R4.3.1: Create, edit, delete single or recurring classes (with presets) i.e. recurring weekly schedules or individually created classes
- R4.3.2: Override for holidays, coach swaps, special events.
- R4.3.3: Define coach assignment, capacity, duration, class credit cost.

### **R4.2: Coach Assignment & Notification**

- R4.4.1: Assign coaches to classes; view schedule in coach mobile dashboard.
- 

## **R5: Workout Design & Management (High Priority)**

### **R5.1: Workout Creation for scheduled classes (Coach)**

- R5.1.1: Free-text entry → structured templates (For Time, AMRAP, EMOM, Chipper, Intervals)
- R5.1.2: Tag workouts by category.

### **R5.2: Template Library (Load from history)**

- R5.2.1: Save common WOD templates for reuse.
  - R5.2.2: Create, delete and edit templates
- 

## **R6: Payments & Subscriptions (High Priority) (ADVISED AGAINST DOING)**

### **R6.1: Payment Processing**

- R6.1.1: Integrate recurring (monthly) and once-off payments
- R6.1.2: Enforce “pay-first” model: no booking without paid subscription/credit.
- R6.1.3: Generate and email receipts/invoices.

### **R6.2: Subscription & Packages**

- R6.2.1: Define unlimited, fixed-credit, and custom-duration plans.
  - R6.2.2: Auto-renewal with opt-out; configurable cancellation policies.
  - R6.2.3: Member dashboard: view active plan, usage, expiry, and remaining credits.
  - R6.2.4: Manager can define new packages
- 



## **R7: Score Submission & Leaderboards (Medium Priority)**

### **R7.1: Score Recording**

- R7.1.1: Members submit their workout result (time, reps, rounds).
- R7.1.2: Coaches can review/edit scores post-class.
- R7.1.3: Coaches can fill in scores for members who cannot do so at the time.

### **R7.2: Leaderboards**

- R7.2.1: Live daily leaderboard by class.
  - R7.2.2: Overall (gym-wide) leaderboards.
  - R7.2.3: Format-specific ranking rules (e.g., low-time vs. high-reps).
  - R7.2.4: Privacy toggle: opt-out of public display.
- 

## **R8: Reports & Analytics (Medium Priority)**

### **R8.1: Operations & Utilization**

- R8.1.1: Class fill-rate, no-show rates, cancellation rate, gym utilization.

### **R8.2: Financial Metrics**

- R8.2.1: Monthly revenue breakdown (recurring vs. one-off)
- R8.2.2: Average Revenue Per User (ARPU) / Lifetime Value (LTV), churn rates.

### **R8.3: Retention & Acquisition**

- R8.3.1: New signups per day and conversion funnel
  - R8.3.2: Cohort retention.
- 

## **R9: Gamification & Engagement (Low Priority / Optional)**

### **R9.1: Streaks & Badges**

- R9.1.1: Track consecutive attendance streaks.
- R9.1.2: Points system or award badges for milestones (e.g., 50 WODs).

### **R9.2: Avatar Evolution (Experimental)**

- R9.2.1: Unlock avatar upgrades based on points; optional "avatar battle" mini-game.
- 

## **R10: Workout Data Input Subsystem**

### **R10.1: Common Input Framework**

- R10.1.1: Real-time submission ensures the leaderboard updates instantly.



- R10.1.2: Allow manual entry/edit of the fields.
- 

### **R11.1: "For Time" Workouts**

#### **R11.1.1: Timer Controls**

- R11.1.1.1: "Start" button begins a stopwatch; timestamp recorded as start\_time.
  - R11.1.1.2: "Stop" button halts the stopwatch; timestamp recorded as end\_time.
  - R11.1.1.3: Display elapsed time in mm:ss format during timing.
- 

### **R11.2: AMRAP (As Many Rounds/Reps As Possible)**

#### **R11.2.1: Time-cap and Countdown**

- R11.2.1.1: Display the workout's time cap (e.g., "12:00") and a countdown timer.

#### **R11.2.2: Rounds & Reps Entry**

- R11.2.2.1: Two numeric fields: rounds\_completed and extra\_reps.
  - R11.2.2.2: Real-time calculation of total\_reps = rounds\_completed × reps\_per\_round + extra\_reps.
- 

### **R11.3: EMOM (Every Minute on the Minute)**

#### **R11.3.1: Minute countdown**

- R11.3.1.1: Specify how many minutes (intervals) in that workout
- R11.3.1.2: Display the countdown of the minute followed by short data entering period

#### **R11.3.2: Number of intervals completed entry**

- R11.3.2.1: Indicate whether they completed the task for that minute or not
- 

### **R11.5: Interval Workouts (Fixed Work/Rest Blocks)**

#### **R11.5.1: Work/Rest Timer**

- R11.5.1.1: Built-in timer alternates Work and Rest phases (configurable durations).
  - R11.5.1.2: At end of each Work phase, prompt for reps\_this\_interval.
- 

### **R11.6: Weight & Scaling Inputs**

- R11.6.1: Reps and Load entry calculates score taking into account their body weight
- 



# User Stories

Title:	Priority: 1	Estimate:
Member Registration		

## User Story:

As a new member  
I want to register via mobile app using email and password  
So that I can access the gym services

## Acceptance Criteria:

Given I am a new user on the registration page  
When I enter valid email, password  
Then my account is created and I can login

Title:	Priority: 1	Estimate:
Member Login		

## User Story:

As a registered member  
I want to securely log in to the mobile app using email and password  
So that I can access my profile, book classes and track my workouts

## Acceptance Criteria:

Given my login credentials are correct  
When I submit the login credentials  
I am given access to the app

Title:	Priority: 1	Estimate:
Coach login		

As a coach  
I want to login via mobile app using manager-provisioned credentials  
So that I can access coach features

## Acceptance Criteria:

Given I have valid coach credentials  
When I enter them correctly in the login screen  
Then I am granted access to the coach sections of the app



Title:	Priority: 1	Estimate:
Booking		

As a member  
I want to book classes using my subscription or credits  
So that I can attend sessions

---

**Acceptance Criteria:**

Given I have an active subscription credits  
When I book a class  
Then my credits are deducted and I receive a booking confirmation

---

Title:	Priority: 2	Estimate:
Join class		

As a member  
I want to join a class that I have booked  
So that I can participate and record my workout details

---

**Acceptance Criteria:**

Given that I have booked the class  
When the time for class comes  
Then I can join and participate

---

Title:	Priority: 2	Estimate:
Record workout details (leaderboards)		

As a member  
I want my workout times and scores to be recorded  
So that the coach can see my performance and my scores are shown on a leaderboard

---

**Acceptance Criteria:**

Given that I am participating in a class  
When I complete a workout  
Then the score and time are shown to the coach

---





Title:	Priority: 2	Estimate:
Set workouts		

As a coach  
 I want to add workouts to my assigned classes  
 So that attending members have something to do in class

---

**Acceptance Criteria:**

Given that I have a class assigned to me  
 When I add workouts and their details to the class  
 Then the workouts are added to the class details and the users can view them

---

Title:	Priority: 2	Estimate:
View member scores		

As a coach  
 I want to view the scores and times for the members' workouts  
 So that I can better understand their performance

---

**Acceptance Criteria:**

Given that there are members attending my class  
 When a member completes a workout  
 Then their score is updated and the updates show on my screen

---

Title:	Priority: 1	Estimate:
View Schedule		

As an administrator  
 I want to view the weekly class schedule  
 So that I can plan accordingly

---

**Acceptance Criteria:**

Given that I have administrator permissions and am logged in on the dashboard  
 When I open the home page  
 Then I can see the class schedule

---



Title:	Priority: 2	Estimate:
View user roles		

As an administrator  
I want to be able to view users for each role  
So that I can plan user management

---

**Acceptance Criteria:**

Given that I am logged in on the administrator dashboard  
When I open the user management page  
Then I can see users for each role

---

Title:	Priority: 2	Estimate:
Approve membership		

As an administrator  
I want to be able to approve newly registered members  
So that they can make full use of the services

---

**Acceptance Criteria:**

Given that the new member is still not approved  
When I click on the approve button  
Then the member's status is set to approved

---

Title:	Priority: 1	Estimate:
Add roles		

As an Administrator  
I want to be able to assign roles to users  
So that I can manage user permissions

---

**Acceptance Criteria:**

Given that I have permissions to manage a user's role  
When I am on the user's edit page and select to add roles  
Then I can add roles that the user does not have

---



Title:	Priority: 1	Estimate:
Remove roles		

As an Administrator  
I want to be able to remove roles from users  
So that I can manage user permissions

---

**Acceptance Criteria:**

Given that I have permissions to manage a user's role and a user has at least one role assigned  
When I am on the user's edit page and click to remove a specific role  
Then the role is removed from the user's list of roles and the user loses the associated permissions

---

Title:	Priority: 2	Estimate:
Class Setup		

As a manager  
I want to create single or recurring classes  
So that I can manage the schedule

---

**Acceptance Criteria:**

Given I'm in the class management interface  
When I create a new class  
Then it appears in the schedule

---

Title:	Priority: 2	Estimate:
Assign Coach to Class		

As an administrator  
I want to be able to assign a different coach to a class  
So that coaches can be easily swapped

---

**Acceptance Criteria:**

Given that I am on the schedule page of the dashboard  
When I click on a class in the schedule  
Then I can edit the class details and assign a coach

---



Title:	Priority: 1	Estimate:
Register Coach		

As an administrator  
 I want to be able to register coaches on my dashboard  
 So that the coach assignment process becomes faster

---

**Acceptance Criteria:**

Given that the coach is not already a registered user and I have the coach's details  
 When I input the coaches details and register them  
 Then the coach becomes a user with the coach role and can log in as a coach

---

Title:	Priority: 3	Estimate:
Admin notifications		

As an administrator  
 I want to see important notifications on my dashboard  
 So that I am updated with any new events happening in the system

---

**Acceptance Criteria:**

Given that an event requiring notification happens  
 When I open the dashboard  
 Then the notification is shown

---

Title:	Priority: 3	Estimate:
Read notification		

As an administrator  
 I want to mark notifications as read  
 So that I do not see the same notification twice

---

**Acceptance Criteria:**

Given that I have read a notification  
 When I open my dashboard again  
 Then it will not pop up again

---



<b>Title:</b> Leaderboard opt-out	<b>Priority:4</b>	<b>Estimate:</b>
--------------------------------------	-------------------	------------------

As a member  
I want to opt-out of my scores being on the leaderboard  
So that others cannot see my score

---

**Acceptance Criteria:**

Given I have opted out of leaderboard visibility  
When others view the leaderboard  
Then my score does not appear in the rankings

---

<b>Title:</b> Member Profile	<b>Priority:2</b>	<b>Estimate:</b>
---------------------------------	-------------------	------------------

As a member  
I want to view and update my profile information  
So that my emergency contacts and details are current

---

**Acceptance Criteria:**

Given I am logged in  
When I navigate to the profile section  
Then I can view and edit my name, contact info, and emergency contacts

---

<b>Title:</b> Coach Schedule	<b>Priority:1</b>	<b>Estimate:</b>
---------------------------------	-------------------	------------------

As a coach  
I want to see my assigned classes  
So that I can manage my time and prepare accordingly

---

**Acceptance Criteria:**

Given I am logged in as a coach  
When I open the schedule in my dashboard  
Then I see all my assigned classes with times



Title:	Priority:2	Estimate:
Search classes		

As a member  
I want to search for classes and filter the search  
So that I can find classes that I want to attend

---

**Acceptance Criteria:**

Given there are classes that meet my criteria  
When I click search  
Then I can see all the classes that meet search parameters

---

Title:	Priority:3	Estimate:
Subscription management		

As a manager  
I want to create new subscription packages  
So that I can create more offers for members

---

**Acceptance Criteria:**

Given I'm in the package management dashboard  
When I define a new package with unique parameters  
Then it becomes available for purchase

---

Title:	Priority:3	Estimate:
Subscription renewal		

As a member  
I want to automatically receive payment receipts via email  
So that I have records for my purchases

---

**Acceptance Criteria:**

Given I've completed a payment  
When the payment processes successfully  
Then I receive a confirmation receipt via email

---



Title:	Priority:4	Estimate:
Member class payments		

As a member  
I want to pay for a subscription  
So I can start booking classes

---

**Acceptance Criteria:**

Given I selected a plan  
When I complete a payment  
Then my access activates immediately

---

Title:	Priority:4	Estimate:
Opt-out of auto-renewal		

As a member  
I want to be able to cancel auto-renewal of subscriptions  
So I can have full control of my payments

---

**Acceptance Criteria:**

Given I disable auto-renewal  
When I confirm the cancellation  
Then my plan expires on the end date

---



# User Characteristics

## 1. Members

**Technical skill:** Mostly average smartphone users; may not be tech-savvy.

**Needs:** Quick and intuitive class booking, easy navigation, reminders, and performance tracking.

**Constraints:** Limited time, low patience for long forms or slow apps.

**Devices:** Mobile-first (Android/iOS).

## 2. Coaches

**Technical skill:** Moderate—may be familiar with fitness apps or gym software.

**Needs:** View schedules, upload workouts quickly (text/media), access class rosters.

**Constraints:** Often working on tight schedules, need fast, responsive tools.

**Devices:** Mobile-first, some may use tablets.

## 3. Managers/Admins

**Technical skill:** Moderate to high—may be familiar with spreadsheets, dashboards, and basic IT systems.

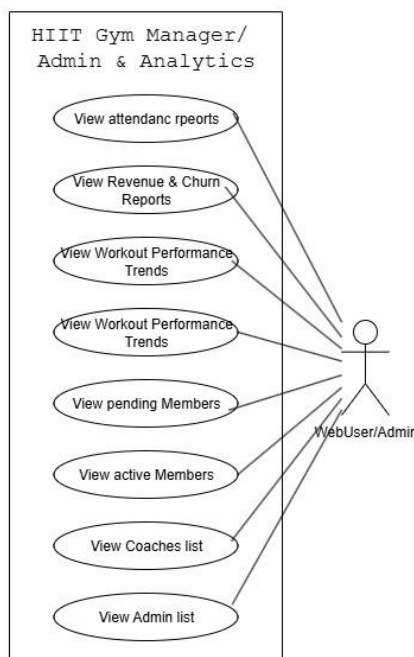
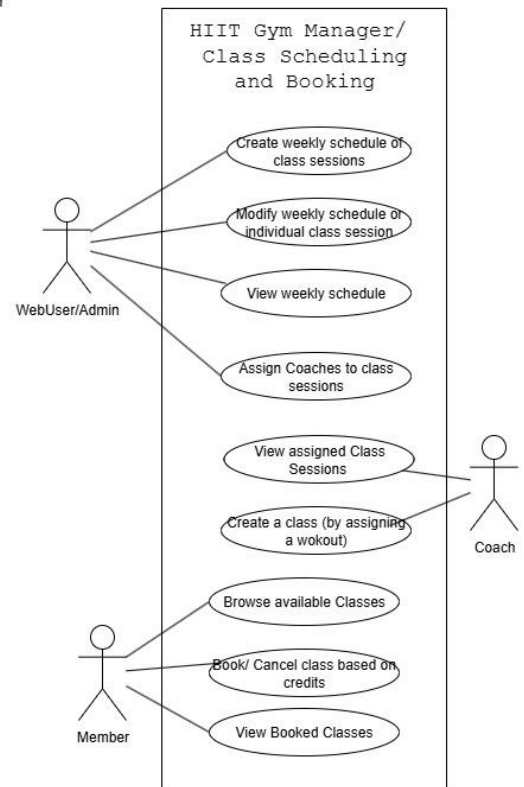
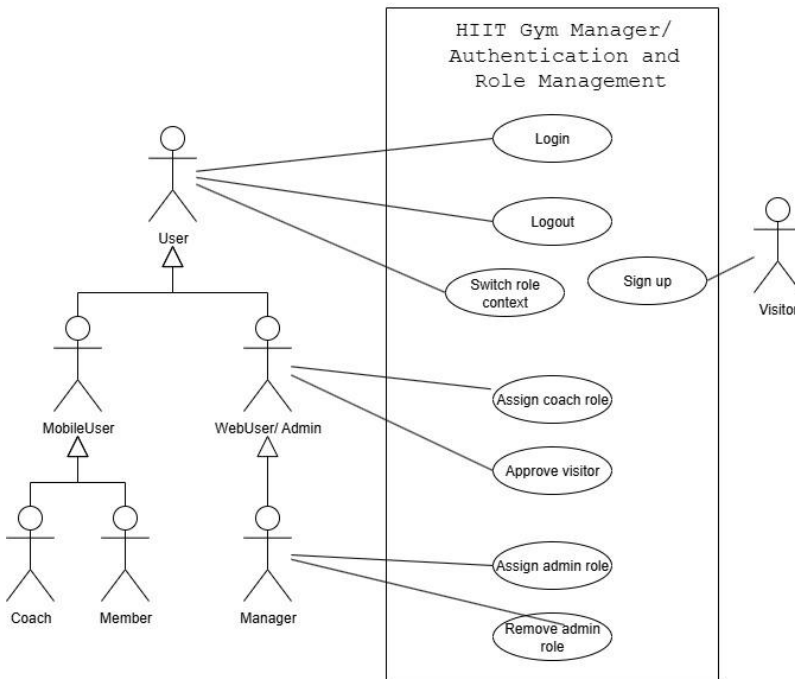
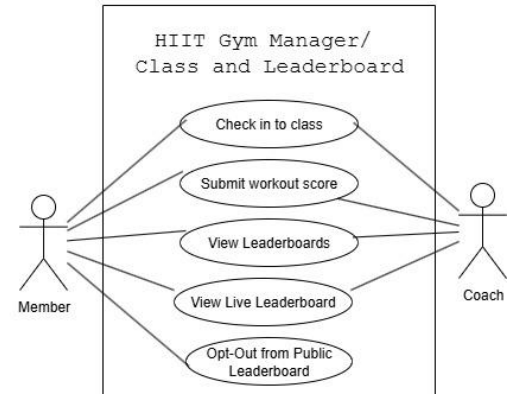
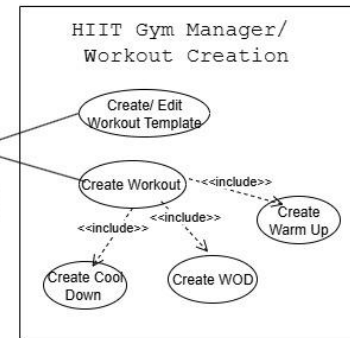
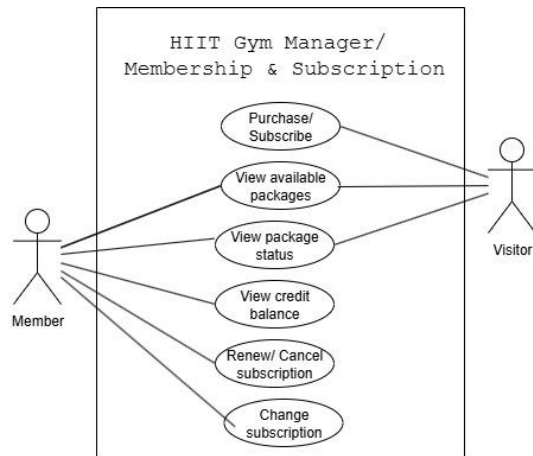
**Needs:** Oversee members and coaches, manage schedules, packages, reports.

**Constraints:** Need accuracy, stability, and some level of customization (e.g., setting packages or policies).

**Devices:** Desktop-focused, possibly using laptops or tablets.

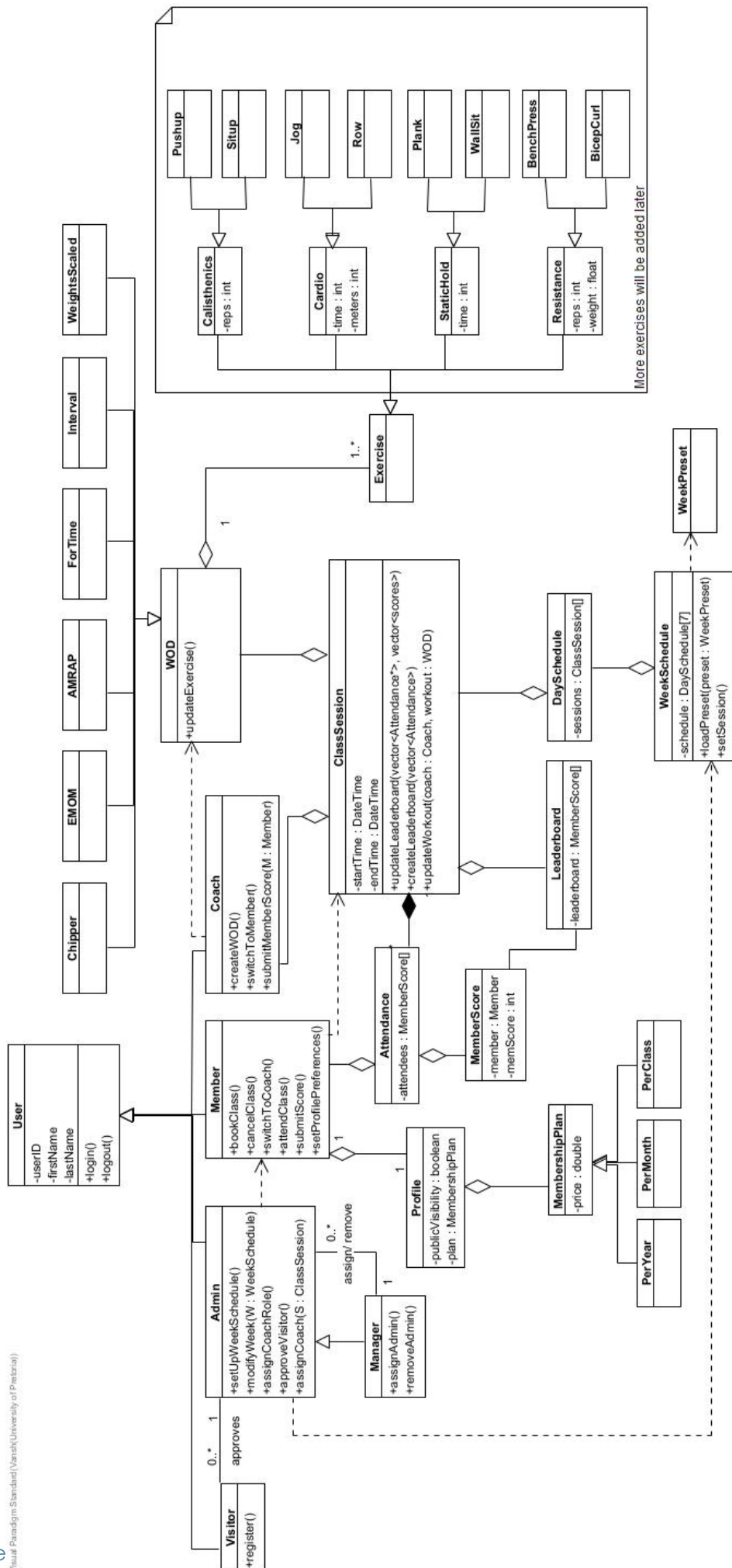






NOTE: Class Sessions are just time slots; Classes are the actual class with workout assigned that members can then book





## Quality Requirements (Non-Functional)

- **QR1: Usability**
- **QR2: Security**
- **QR3: Reliability/ Availability**
- **QR4: Portability**
- **QR5: Performance**

**SEE Architectural Requirements Document**

For more details regarding the Quality Requirements and Architectures

## Architectural Patterns

1. **Clean/ Layered Architecture - Overall**
2. **Event-Driven / Pub-Sub – Live & Notification System**
3. **Component-Based - Frontend**

## Design Patterns

- **Singleton** Shared DB connection pool and configuration loader
- **Factory / Builder** Create workout or notification objects based on format/type at runtime
- **Strategy** Scoring algorithms per workout format (For Time, AMRAP, EMOM, etc.)
- **Observer** Notification subsystem: subscribe to domain events to send emails/SMS
- **Adapter** Integrate external APIs (Stripe, Twilio, Setmore) behind uniform interface
- **Command** Encapsulate scheduled tasks (send reminders, generate reports)

## Constraints

### Technology Stack

- Frontend: Next.js (Web), React Native (Mobile)
- Backend: Node.js (API, workers, real-time)
- Database: PostgreSQL (with Drizzle ORM)
- Cache/Queue: Redis (or RabbitMQ)

### Infrastructure & Deployment

- Monorepo managed by Turborepo with Yarn Workspaces or pnpm
- Containerization: Docker & Docker-Compose for local dev; Kubernetes for production
- CI/CD: GitHub Actions with lint/test/build pipelines

### Compliance

- PCI-DSS for payment data (via Stripe's tokenization)
- GDPR-style data handling and digital consent storage

### Operational

- Must support South Africa (Africa/Johannesburg) timezone scheduling
- Limit use of third-party vendor lock-in (use open standards where possible)

### Resource

- Mobile app must operate on iOS 13+ and Android 9+
- Web UI optimized for modern browsers (Chrome, Safari, Edge)



# Technology Requirements

---

## 1. Mobile App & Web Interface

- **Mobile Framework:** React Native (with Expo)
- **Web Framework:** Next.js
- **Language:** TypeScript
- **Rationale:**
  - Shared codebase → consistency and speed across platforms
  - Seamless state management using Redux
  - Cross-platform UI flexibility for roles-based interfaces
- **Alternatives:**

Option	Pros	Cons
Flutter	<ul style="list-style-type: none"><li>• Very fast UI</li><li>• Single codebase like React Native</li></ul>	<ul style="list-style-type: none"><li>• Dart unfamiliar to team</li><li>• Heavier APKs</li></ul>
Native Android + iOS	<ul style="list-style-type: none"><li>• Best performance</li><li>• Full platform APIs</li></ul>	<ul style="list-style-type: none"><li>• Two codebases → doubles dev effort</li><li>• Small team can't staff both</li></ul>
React Native CLI (bare)	<ul style="list-style-type: none"><li>• Same libs as Expo</li><li>• More native control</li></ul>	<ul style="list-style-type: none"><li>• Build toolchain per OS → slower CI</li><li>• OTA updates need extra infra</li></ul>

- **Decision:** React Native + Expo selected for its single-codebase maintainability and OTA availability gains within time/budget constraints.
- **Supports:**
  - QR – Maintainability (Single shared codebase)
  - QR – Availability (OTA hot-fix deploys)
- **Offline support:** Redux Persist (stores queued requests in SQLite)

---

## 2. Backend API & Business Logic

- **Runtime:** Node.js (LTS)
- **Framework:** Express.js or NestJS
- **Database:** PostgreSQL and Drizzle ORM
- **Rationale:**
  - Robust and scalable
  - Strong TypeScript support
  - Clear relational modelling for users, schedules, and tracking data



- **Alternatives:**

Option	Pros	Cons
<b>NestJS</b>	<ul style="list-style-type: none"> <li>• Opinionated structure</li> <li>• Built-in DI, testing</li> </ul>	<ul style="list-style-type: none"> <li>• Heavier learning curve</li> <li>• Extra decorators/boilerplate</li> </ul>
<b>Django + DRF</b>	<ul style="list-style-type: none"> <li>• Admin site</li> <li>• Mature auth stack</li> </ul>	<ul style="list-style-type: none"> <li>• Python stack (new language)</li> <li>• Slower JSON throughput</li> </ul>
<b>Spring Boot</b>	<ul style="list-style-type: none"> <li>• Strong type-safety</li> <li>• Rich enterprise ecosystem</li> </ul>	<ul style="list-style-type: none"> <li>• Java verbosity</li> <li>• Higher memory footprint</li> </ul>

- **Decision:** Express chosen for fastest feature velocity and direct alignment with the team's TypeScript expertise, while still meeting QR-Scalability (stateless, horizontal) and QR-Maintainability (simple, flat structure).
- **Supports:**
  - QR – Scalability (non-blocking I/O)
  - QR - Maintainability (lightweight, minimal layers)

---

### 3. Real-Time Features

- **Clock Sync & Live Leaderboards: WebSocket (Socket.IO)**
- **Live Workout Scoring & Judge Input: Socket.IO**
- **Rationale:**
  - Supports responsive, low-latency communication
  - Enables real-time scoring, notifications, and updates
- **Alternatives:**

Option	Pros	Cons
<b>AWS Api Gateway (WebSocket)</b>	<ul style="list-style-type: none"> <li>• Fully managed, auto-scales</li> <li>• Native IAM auth possible</li> </ul>	<ul style="list-style-type: none"> <li>• Extra latency (region hop)</li> <li>• Monetary costs</li> </ul>
<b>Firebase Realtime Database</b>	<ul style="list-style-type: none"> <li>• Zero-config pub/sub</li> <li>• Built-in offline caching</li> </ul>	<ul style="list-style-type: none"> <li>• Data model = JSON tree - poor fit for relational scores</li> <li>• Vendor lock-in</li> </ul>
<b>pusher.com Channels</b>	<ul style="list-style-type: none"> <li>• Easy client SDKs</li> <li>• Webhooks for persistence</li> </ul>	<ul style="list-style-type: none"> <li>• Usage-based pricing</li> <li>• External service dependency</li> </ul>

- **Decision:** Socket.IO aligns with the current TypeScript/Node skill set and stays entirely inside the project's code-base.



- **Supports:**
  - QR – Responsiveness (immediate push updates)
  - QR - Maintainability (single language and repo)

---

#### 4. Data & Analytics

- **Analytics Dashboard:** Custom dashboards with charting libraries (e.g., Recharts, D3.js)
- **Usage Trends & Class Performance Metrics:** Server-side aggregation via Node.js
- **Alternatives:**

Option	Pros	Cons
<b>Chart.js</b>	<ul style="list-style-type: none"> <li>• Simple API,</li> <li>• Good defaults</li> </ul>	<ul style="list-style-type: none"> <li>• Vanilla JS → custom React wrappers needed,</li> <li>• Less flexible for complex layouts</li> </ul>
<b>Apache ECharts</b>	<ul style="list-style-type: none"> <li>• Large chart library,</li> <li>• Handles huge datasets</li> </ul>	<ul style="list-style-type: none"> <li>• Heavier bundle size,</li> <li>• TypeScript typings less mature</li> </ul>
<b>Metabase (external BI)</b>	<ul style="list-style-type: none"> <li>• Auto-generated dashboards,</li> <li>• Non-dev users can explore</li> </ul>	<ul style="list-style-type: none"> <li>• Requires separate service and auth,</li> <li>• Limited custom styling</li> </ul>

- **Decision:** Recharts selected: smallest learning curve for a React-native team and enough chart types for MVP dashboards, while server-side SQL keeps data logic in a single place.
- **Supports:**
  - QR – Maintainability (unified React codebase)
  - QR - Performance (push only aggregated rows to client)

---

#### 5. Infrastructure & CI/CD

- **CI/CD:** GitHub Actions + Terraform
- **Rationale:**
  - GitHub Actions lives in the same repo → zero extra accounts, automatic PR pipelines.
  - Terraform keeps infra version-controlled and repeatable across dev machines.
- **Decision:** GitHub Actions + Terraform chosen for seamless GitHub integration, no vendor lock-in, and zero cost at MVP scale, while meeting QR-Maintainability (one-file pipeline) and QR-Availability (blue-green deploy jobs with instant rollback).
- **Supports:**
  - QR – Maintainability (pipeline-as-code in YAML)
  - QR - Traceability (IaC state versioned in repo)



- **Alternatives:**

Option	Pros	Cons
<b>AWS Amplify (+ CloudFormation)</b>	<ul style="list-style-type: none"> <li>• Tight React support</li> <li>• Edge functions available</li> </ul>	<ul style="list-style-type: none"> <li>• Steeper AWS IAM setup</li> <li>• Costs can spike if mis-configured</li> </ul>
<b>Heroku Pipelines</b>	<ul style="list-style-type: none"> <li>• One-click deploys</li> <li>• Built-in rollback</li> </ul>	<ul style="list-style-type: none"> <li>• Free tier deprecated</li> <li>• Limited regions</li> </ul>
<b>GitLab CI + GitLab Runners</b>	<ul style="list-style-type: none"> <li>• Integrated container registry</li> <li>• Rich pipeline UI</li> </ul>	<ul style="list-style-type: none"> <li>• Entire repo must live on GitLab</li> <li>• Self-hosted runner setup for private projects</li> </ul>

---

## 6. Testing & Quality

- Unit Tests: Jest (TS)
  - Integration Tests: Jest (+ supertest)
  - E2E (Mobile): Detox
  - Linting: ESLint + Prettier
  - **Rationale:**
    - Jest runs in the same Node/TypeScript runtime as the code → zero context-switch.
    - supertest lets us hit Express routes without a real network port.
    - Detox drives React-Native apps on device/simulator; fits our Expo stack.
    - ESLint + Prettier enforce consistent style automatically in GitHub Actions.
  - **Decision:** We keep Jest + Detox because they align with the team's TypeScript/React-Native skillset, integrate smoothly with GitHub Actions, and incur no extra cloud cost or learning curve.
  - **Alternatives:**
    - Mocha + Chai + Sinon or Vitest (Unit and Integration)
    - Appium or Expo E2E
    - StandardJS (Linting)
  - **Supports:**
    - QR – Reliability (automated test suite on every PR)
    - QR – Maintainability (consistent code style gate in CI)
- 



## Why This Stack?

- **Developer Efficiency:** TypeScript-first stack enables rapid development with consistency across frontend and backend.
  - **User-Centric Design:** Offline support and real-time updates cater to mobile-first gym users and staff.
  - **Cost-Effective Scalability:** Uses free-tier and low-overhead tools while supporting rapid scale-up.
  - **Fitness-Tailored Features:** Built-in leaderboard logic, rep counters, and workout tracking modules are tailored to HIIT routines.
- 





## Deployment Model

(SEE [ARCH REQUIREMENTS DOC](#))

The HIIT Gym Management System will use a cloud-based deployment model to ensure accessibility, scalability, and ease of maintenance.

- The **frontend web app** (for managers/admins) is deployed using **Vercel**.
- The **mobile app** (for coaches and members) is distributed via **Expo**, accessible on both iOS and Android via TestFlight and an apk respectively and then later via store deployment.
- The **backend API** will be hosted on **Vercel** while the database will be hosted on **Supabase**
- Environment-specific configurations (e.g. dev, staging, production) will be managed using .env files and platform-level secrets.

## Live Deployed System

(SEE [ARCH REQUIREMENTS DOC](#))

A fully deployed and accessible version of the system will be available during the final demo:

- The **web admin interface** will be publicly accessible through a **Vercel-hosted URL**.
- The **mobile app** (for coaches and members) is distributed via **Expo**, accessible on both iOS and Android via TestFlight and an apk respectively and then later via store deployment.
- A **live backend API** connected to a hosted PostgreSQL database will support real-time operations like booking, workout uploads, and leaderboard updates.
- All key user flows, including sign-up, login, class booking, and workout submission, will be testable on the live system.
- The deployment will include seeded demo data and distinct roles (e.g., coach, member, manager) to showcase end-to-end functionality during the live evaluation.

