# **Introduction**

This documentation covers the Micro Web Application (MWA) developed using the Ionic Framework. The MWA is designed as a modular component to be seamlessly integrated into the main GymNation app, which is managed by our third-party partner, eGym. This microservice-based web application enhances the overall user experience by offering convenient self-service features directly from the app.

Our architecture follows the Ionic Portal approach, enabling the integration of independent micro-apps into a single host application. You can learn more about this model through [eGym's Micro Web App documentation](https://egym.stoplight.io/docs/micro-web-apps/k13uzbeg9y803-reference-web-application).

**Product Overview**

The GymNation Portal App is a customer-centric micro web application that empowers GymNation members with more control and flexibility over their accounts. Instead of visiting kiosks or gym reception areas, members can now manage key aspects of their membership directly through the app.

This application will be embedded within the main GymNation app using the Ionic Portal system, ensuring smooth interoperability and consistent user experience across all services offered by GymNation and its partners.

**Purpose & Scope**

The primary goal of this portal app is to improve customer engagement and satisfaction by enabling members to perform essential account-related actions independently and effortlessly through their mobile device.

The key features covered by the application include:

* **Payment Handling** – Manage and complete membership-related transactions using Telr Secure Payments.
* **Upgrade Plans** – Change or enhance membership plans with ease.
* **Update Member Details** – Modify personal information without the need for staff assistance.
* **Account Freezing** – Temporarily suspend membership as per user preference.
* **Account Cancellation** – Allow users to cancel their subscription conveniently within the app.

# **System Architecture**

The **Member Area App** is designed using a microservices-inspired architecture, enabling scalability, modularity, and ease of maintenance. Each module focuses on a specific business capability, allowing seamless integration and future extensibility. Below is an overview of the system’s core components:

**1. Web Service Layer**

* Built with NestJS (Node.js framework), this layer serves as the main entry point of the application.
* It manages routing of incoming HTTP requests to the appropriate internal modules and microservices.
* Provides a modular structure for organizing features and services cleanly and efficiently.

**2. Secrets Manager**

* Utilizes AWS Secrets Manager to securely store and manage sensitive credentials (e.g., database credentials, API keys).
* Ensures that secrets are accessed securely and rotated when necessary.

**3. Database Layer**

* Powered by PostgreSQL, this layer stores all persistent data, including member profiles, transactions, and activity logs.
* Optimized for relational queries and scalability.

**4. Auth Module**

* Manages user authentication using JWT tokens.
* Supports token validation, and secure access to protected routes.

**5. OTP Module**

* Handles One-Time Password (OTP) generation and verification.
* Used for verifying sensitive actions like account updates, plan changes, and cancellations.

**6. Member Module**

* Manages core member information including profile data, contact details, and preferences.

**7. Member Logs Module**

* Tracks all member activities and system interactions.
* Useful for audit trails, support inquiries, and historical actions.

**8. Contract Module**

* Retrieves and manages all contracts associated with each member.
* Supports multiple contracts per member if applicable.

**9. Plans Module**

* Lists available plans and their metadata (e.g., features, durations, eligibility).

**10. Upgrade Module**

* Processes member plan upgrade transactions.
* Calculates proration, fees, and updates the current plan upon confirmation.

**11. Payment Plans Module**

* Handles dynamic payment calculation logic based on plan types, upgrades, and billing cycles.

**12. Cancellation Module**

* Manages account cancellation workflows, including eligibility checks and reason capture.
* Logs cancellation details for future analysis.

**13. Freeze Module**

* Allows members to freeze their account temporarily, according to GymNation’s freeze policy.
* Handles duration, validation, and reactivation logic.

**14. Club Module**

* Lists all GymNation club locations along with metadata such as address, contact, and availability.

**15. External Integrations**

* Integrates with third-party services including:
  + **Telr** – for secure payment processing.
  + **Perfect Gym** – for member and facility data synchronization.
  + **Salesforce** – for CRM and support data integration.
  + Other systems to ensure seamless cross-platform functionality.

# **Backend API**

Our backend API is built using NestJS and serves as the central interface for all business logic and data operations. It provides a structured, scalable, and secure way for the frontend and other systems to interact with application data. Designed with modularity in mind, the API exposes all necessary endpoints for user management, transaction handling, integration services, and more. This layer ensures a clean separation of concerns and acts as the business logic engine of the platform.

## **Technology Stack**

* **Backend Framework**: Nest js
* **Database**: PostgreSQL, Prisma (db toolkit)
* **Deployment**: Docker
* **Payment Gateway**: Telr
* **SQL ORM**: Prisma Client
* **Authentication**: Nest Js Auth
* **Tunneling**: Ngrok
* **Secret Managers**: AWS Client Secrets Manager
* **Email Notification**: Send Grid
* **Sms Notification**: SMS Global / Saudi Connect

# **Frontend App**

The frontend of our system is developed using the **Ionic Framework**, structured as a **micro web app** to promote modularity, scalability, and seamless integration within the broader **eGym ecosystem**. Designed as a feature-rich submodule, it serves as the main user interface for delivering our custom app functionalities.

Leveraging Ionic's **cross-platform capabilities**, the app is fully optimized for deployment on **both Android and iOS devices**, ensuring a consistent and intuitive user experience across mobile platforms. It utilizes **Capacitor** to bridge native device APIs, allowing smooth access to platform-specific features like storage, notifications, and biometric authentication.

## **Technology stack**

* **App Framework**: Ionic Framework
* **Frontend**: React Vite
* **Tunneling**: Ionic App Flow
* **Programming Language**: Typescript
* **Deployment:** Docker
* **Native SDK**: Ionic Capacitor
* **State Management**: Redux
* **Internalization**: i18next
* **CSS**: Tailwind CSS
* **Forms:** React hook form
* **Unit Testing:** React Testing Library, Jest and Vitest

## **Others**

* **Figma**: <https://www.figma.com/design/SKRFsOvW7QM7Oga3e37n02/final-members-module?node-id=0-1&p=f&t=HuuFMa1uqJrFjzNA-0>
* **Telr Documentation**: <https://docs.telr.com/reference/get-started>
* **Perfect Gym Integration**: <https://gymnation.perfectgym.pl/Api/Docs/ApiReference/>
* **Appflow dashboard:** <https://ionic.io/login?source=appflow&product=appflow>

## **Environment Variables**

For security reasons, certain data variables are intentionally excluded from this configuration. Please refer to the DevOps documentation or contact the DevOps team directly to obtain the necessary environment variables required to run this project in your development environment.

VITE\_APP\_NODE\_ENV=development

VITE\_APP\_API\_URL\_DEV=http://localhost:3005

VITE\_APP\_API\_URL\_STAGING=

VITE\_APP\_API\_URL\_PRODUCTION=

VITE\_AUTH\_ENDPOINT=auth

VITE\_MEMBERS\_ENDPOINT=members

VITE\_OTP\_ENDPOINT=otp

VITE\_PAYMENTPLANS\_ENDPOINT=paymentplans

VITE\_CLUBS\_ENDPOINT=clubs

VITE\_CONTRACTS\_ENDPOINT=contracts

VITE\_FREEZE\_ENDPOINT=freezes

VITE\_PAYMENT\_ENDPOINT=payments

VITE\_CANCELLATION\_ENDPOINT=cancellations

VITE\_PLAN\_ENDPOINT=plans

VITE\_UPGRADES\_ENDPOINT=upgrades

**Test Data:**

Sample test data that we used for testing the app.

{

  "email": "test@gymnation.com",

  "password": "Karl",

  "firstName" : "Karl",

  "lastName" : "Test"

}