Dissertation/Report submitted to the

UNIVERSITY OF MOHAMED BOUDIAF, M'SILA, ALGERIA



FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

in partial fulfillment of the requirements for the degree of

Master's in Computer Science

Specialization: Computer Science

Option: Information System and Software Engineering

Presented by

Zaoui Islam Mohammed Lamine
Nouar Abdelrahman

Entitled

Intelligent Coaching Application

Under the supervision of

Samir Akhrouf

Jury Members

Prof. Nasser Eddine Mouhoub	University of M'sila	President
Dr. Nawel Boukhari	University of M'sila	Examiner

June, 2026

الملخص

تقدّم هذه الأطروحة تصميم وتنفيذ تطبيق جوال للتدريب الرياضي مدعوم بوكيل ذكاء اصطناعي. يهدف النظام المقترح إلى حل مشكلة التشتت في تطبيقات اللياقة البدنية الحالية من خلال دمج مجموعة شاملة من الميزات في منصة واحدة متكاملة، تشمل: تخطيط التمارين، تسجيل التمارين، تتبع التغذية والسوائل، تصور التقدم، تحليل الصور الغذائية باستخدام الذكاء الاصطناعي، التواصل بين المدرب والمتدرب، والإرشاد الشخصي.
تم تطوير تطبيق العميل باستخدام Native React وExpo، بينما تم بناء الباك إند باستخدام Bun وElysiaJS. يعتمد النظام في حفظ البيانات على PostgreSQL مع Prisma، ORM مع استخدام امتداد pgvector لتمكين البحث الدلالي (Semantic Search) لدعم التوصيات الذكية. تدمج طبقة الذكاء الاصطناعي نموذج Gemini Google للتدريب الحواري وتحليل صور الطعام، بالإضافة إلى Ollama والنموذج nomic-embed-text لتقديم توصيات تمارين معززة بالاسترجاع (Retrieval-Augmented Generation). تم نشر الباك إند على Fly.io مع تفعيل أنظمة المراقبة والتسجيل لضمان التشغيل المستقر. يوضح التطبيق الناتج كيفية الاستفادة من سياق المستخدم، والبيانات الرياضية المنظمة، وسير عمل وكلاء الذكاء الاصطناعي لتقديم تجربة تدريبية أكثر تكيفًا وفعالية، مع تحسين استمرارية التتبع ودعم اتخاذ قرارات أفضل في التدريب والتغذية.

الكلمات المفتاحية: اللياقة البدنية، تطبيق جوال، وكيل ذكاء اصطناعي، نموذج لغة كبير، تحليل غذائي، استرجاع دلالي

Abstract

This thesis presents the design and implementation of a mobile fitness coaching application powered by an AI agent. The proposed system tackles the fragmentation of current fitness applications by integrating workout planning, exercise logging, nutrition and hydration tracking, progress visualization, AI-powered food image analysis, coach-athlete communication, and personalized guidance into a single unified platform.

The mobile client was developed using React Native and Expo, while the backend was built with Bun and ElysiaJS. Data persistence is managed through PostgreSQL with Prisma ORM, enhanced by the pgvector extension to enable semantic search for intelligent recommendations. The AI layer integrates Google Gemini for conversational coaching and food image analysis, alongside Ollama and the `nomic-embed-text` model to deliver retrieval-augmented workout recommendations.

The backend was deployed on Fly.io with monitoring and logging mechanisms to ensure operational reliability. The resulting application demonstrates how user context, structured fitness data, and AI agent workflows can be effectively combined to provide more adaptive

coaching, improve tracking consistency, and support better long-term training and nutrition decisions.

Keywords: fitness, mobile app, AI agent, large language model, food analysis, semantic retrieval

---

Résumé

Ce mémoire présente la conception et la mise en œuvre d'une application mobile de coaching sportif assistée par un agent d'intelligence artificielle. Le système proposé répond à la fragmentation des applications de fitness actuelles en intégrant, au sein d'une plateforme unique et cohérente, la planification des entraînements, l'enregistrement des exercices, le suivi de la nutrition et de l'hydratation, la visualisation des progrès, l'analyse d'images alimentaires par IA, la communication entre coach et athlète, ainsi que des recommandations personnalisées.

L'application mobile a été développée avec React Native et Expo, tandis que le backend repose sur Bun et ElysiaJS. La persistance des données est assurée par PostgreSQL avec Prisma ORM, complété par l'extension pgvector pour la recherche sémantique. La couche d'intelligence artificielle intègre Google Gemini pour le coaching conversationnel et l'analyse d'images alimentaires, ainsi qu'Ollama avec le modèle `nomic-embed-text` pour des recommandations d'entraînement augmentées par récupération.

Le backend a été déployé sur Fly.io avec des mécanismes de surveillance et de journalisation pour garantir sa fiabilité. L'application finale illustre comment le contexte utilisateur, les données structurées de fitness et les flux de travail des agents IA peuvent être combinés pour offrir un accompagnement plus adaptatif, améliorer la régularité du suivi et favoriser de meilleures décisions en matière d'entraînement et de nutrition à long terme.

Mots-clés : fitness, application mobile, agent IA, modèle de langage large, analyse alimentaire, recherche sémantique

Dedication

باسم الله الرحمن الرحيم

الحمد لله التي بنعمته تتم الصالحات

أحمده وأشكره على توفيقه وتسييره وإعانته

فما وصلنا اليه إلا بفضله وكرمه

أهدي ثمرة تخرجي هذه، عرفانًا بالجميل ومحبة لا تُوصف

إلى من كان دعاؤهما سندي وصبرهم نور طريقي

أبي الغالي وأمي الحبيبة

حفظهما الله وأدامهما تاجًا فوق رأسي

وإلى روح جدي الراحل رحمه الله رحمة واسعة

الذي ما زالت ذكراه تسكن قلبي وترافق خطواتي

وإلى جدي العزيز وجدّتي الكريمتين

أطال الله في أعمارهم وبارك في صحتهم

وإلى أستاذي الفاضل الذي كان له الأثر الكبير في توجيهي ودعمي

وإلى أصدقائي الأوفياء الذين شاركوني التعب والفرح

وكانوا خير رفقة في هذه الرحلة

زاوي إسلام

محمد لمين

Dedication

بسم الله الرحمن الرحيم

الحمد لله الذي بنعمته تتمّ الصالحات، والذي وفقنا وسدّد خطان
ا لإتمام هذا العمل

:أُهدي هذا العمل المتواضع

،إلى أمي الحبيبة
.التي ضحّت من أجلي ولم تدّخر جهدًا في سبيل إسعادي على الدوام
.شكراً لكِ على كل نزلة دمعة، وكل ليلة سهرتِها من أجلي

،إلى والدي العزيز
صاحب الوجه الطيب والأفعال الحسنة، الذي لم يبخل عليّ طوال
.حياته
.شكراً لك على كل كلمة تشجيع، وكل دعاء صدق وصلني من قلبك

،إلى إخوتي الأحبة
.الذين كان لهم بالغ الأثر في تجاوز العقبات والصعاب
.وجودكم كان داعمًا لي في كل مرحلة

،إلى أساتذتي الكرام
.ممن لم يتوانَوا في مدّ يد العون وتقديم التوجيهات القيمة
.شكرًا لكم على الصبر والتفاني في نقل المعرفة

،إلى أصدقائي المقربين
.الذين وقفوا بجواري وساعدوني بكل ما يملكون، في أصعدة كثيرة

وأسأل الله تعالى أن يجعل هذا العمل نافعًا ومباركًا
.وأن يوفقنا جميعًا لما فيه الخير والسداد ،

نوار
عبدالرحمان

Contents

Introduction **1**
0.1 General Introduction . . . 1
0.2 Proposed Solution and Objectives . . . 2
0.3 Manuscript Organization . . . 2

1 AI Agents **4**
1.1 Introduction . . . 4
1.2 Large Language Models (LLMs) . . . 4
1.2.1 Definition . . . 4
1.2.2 LLM Components . . . 4
1.2.3 LLM Usage . . . 5
1.3 What is an AI Agent? . . . 6
1.3.1 General Workflow . . . 6
1.3.2 Applications . . . 7
1.4 Examples . . . 7
1.5 LLMs vs. AI Agents: Key Differences . . . 8
1.6 How AI Agents Manage Context . . . 8
1.6.1 Retrieval-Augmented Generation (RAG) . . . 8
1.6.2 System Prompts . . . 9
1.6.3 Tool Calling . . . 9
1.6.4 Managed Context Providers (MCPs) . . . 9
1.7 The Importance of Rich Context for AI Agents . . . 9
1.8 Conclusion . . . 10

2 Gym Progress Tracking and Fitness Management **11**
2.1 Introduction . . . 11
2.2 Importance of Fitness Tracking . . . 11
2.2.1 Core Functions . . . 11
2.3 Core Principles of Gym Training . . . 12
2.3.1 Exercise Selection . . . 12
2.3.2 Training Volume and Intensity . . . 13
2.3.3 Rest and Recovery . . . 13
2.3.4 Progressive Overload . . . 13

2.3.5 Periodization . . . 13
2.3.6 Proper Form . . . 14
2.3.7 Nutrition Management . . . 14
2.3.8 Consistency . . . 15
2.4 AI in Fitness Coaching . . . 15
2.4.1 Human Coach vs. AI Coach . . . 15
2.4.2 Personalized Recommendations . . . 16
2.5 Context-Aware AI Coaching Example . . . 17
2.6 Conclusion . . . 17

3 Similar Fitness Tracking Applications 19
3.1 Introduction . . . 19
3.2 Mobile Applications . . . 19
3.2.1 Definition . . . 19
3.2.2 Importance of Mobile Fitness Applications . . . 19
3.3 Similar Fitness Applications . . . 20
3.3.1 In Algeria . . . 21
3.3.2 In Arab World . . . 22
3.3.3 Worldwide . . . 23
3.4 Conclusion . . . 25

4 The Design of the Proposed Application 26
4.1 Introduction . . . 26
4.2 Requirements Specification . . . 26
4.2.1 Functional Requirements . . . 26
4.2.2 Non-Functional Requirements . . . 29
4.3 UML . . . 29
4.3.1 UML Definition . . . 29
4.3.2 The Importance of Using UML . . . 30
4.3.3 UML Diagram Taxonomy . . . 30
4.3.4 Diagrams Employed in the Implementation . . . 31
4.4 Database Architecture . . . 40
4.5 AI Models and Services Used . . . 42
4.5.1 Large Language Model (Google Gemini) . . . 42
4.5.2 Embedding Model (Ollama / nomic-embed-text) . . . 43
4.5.3 Plan Manager Tool --- Approval Flow . . . 43
4.6 Application Architecture . . . 44
4.6.1 Layered Architecture . . . 44
4.6.2 Component Diagram . . . 45

4.7 Conclusion 46

5 Implementation of the Proposed Application 47
5.1 Introduction 47
5.2 Development Environment 47
5.2.1 Hardware 47
5.2.2 Used Technologies and Tools 47
5.3 The Application Architecture 50
5.3.1 Backend Server Architecture 50
5.3.2 Mobile Client Architecture 52
5.3.3 API Contract and Integration Layer 53
5.3.4 Development Workflow 54
5.3.5 Production Deployment and Observability 55
5.4 Application Interfaces 57
5.4.1 API Documentation Interface 57
5.4.2 Fly.io Deployment Dashboard 58
5.4.3 Production Monitoring Dashboard 59
5.4.4 Authentication and Onboarding 60
5.4.5 Dashboard and Home Screen 61
5.4.6 Workout Planning and Tracking 62
5.4.7 Nutrition and Diet Management 65
5.4.8 Progress Tracking 67
5.4.9 AI Coach and Messaging 68
5.4.10 Member-Coach Messaging 69
5.4.11 Equipment and Preferences 70
5.4.12 User Profile 71
5.5 API Integration and Backend Communication 71
5.6 Conclusion 72

General Conclusion 74
Summary of Achievements 74
Contributions 75
Future Work 75

References 77

Appendix A: Calculation Details 80
.1 Total Daily Energy Expenditure (TDEE) 80
.1.1 Basal Metabolic Rate (BMR) 80

.1.2 Activity Factors 80
.2 Training Volume 81
.3 One-Rep Max Estimation (Epley Formula) 81
.4 Body Mass Index (BMI) 81
.5 Macronutrient Distribution 81

Appendix B: List of Acronyms **82**

List of Tables

1.1 Comparison between standalone LLMs and AI agents 8

2.1 The three core functions of gym progress tracking 11
2.2 Comparison between human coach and AI coach in fitness management . . 16

3.1 Comparative summary of similar fitness applications by region 25

4.1 Functional requirements 28
4.2 Non-functional requirements 29
4.3 Roles and their use cases 34
4.4 Class descriptions for the domain model 38

5.1 Hardware used during development 48
5.2 Mobile API classes and their corresponding backend route prefixes 54
5.3 API classes and their responsibilities 72
4 Activity level multipliers for TDEE calculation 80
5 Macronutrient calorie values and recommended ranges 81

List of Figures

1.1 LLM components pipeline: from data and tokenization through architecture and training to inference, evaluation, and deployment. 5
1.2 General AI agent workflow: task reception, analysis and planning, context gathering, action execution, and response synthesis. 7
1.3 Context-driven agent reasoning loop: goals and constraints feed context state, which drives agent reasoning, actions, and feedback. 10

2.1 A visual diagram showing three interconnected circles representing Performance Tracking, Health Optimization, and Motivation & Accountability with icons and key metrics beneath each. 12
2.2 A grid showing different exercise types with visual icons: compound movements (squat, deadlift, bench press), isolation exercises (bicep curl, leg extension), functional training, and cardio HIIT workouts. Source adapted from the exercises dataset [11]. 12
2.3 A line graph showing the relationship between training volume (X-axis) and intensity (Y-axis) over weeks/months of training. 13
2.4 Comparison of three periodization models showing Linear (gradual progression), Undulating (wave-like intensity), and Block (distinct phases). Source adapted from the referenced periodization model image [14]. 14
2.5 Before/after form comparison for squat showing correct vs. incorrect technique with key form annotations. 14
2.6 Macronutrient breakdown pie chart showing optimal ratios for muscle gain, fat loss, maintenance. Source adapted from Beehive Meals' macro-balanced diet guide [16]. 15
2.7 App interface showing AI recommendations with explanations, alternatives, confidence scores. 17

3.1 Power Fitness Club Dz mobile interface. 21
3.2 BGR SMART GYM interface with integrated gym operations. 21
3.3 MayStrong application interface combining workout, nutrition, and local ecosystem features. 22
3.4 Kam Calorie interface emphasizing Arabic voice-based nutrition tracking. . 22
3.5 Rashaqa interface integrating calorie tracking, exercise guidance, and hydration reminders in Arabic. 23

3.6 Soraate interface for Arabic calorie logging and community motivation. . . 23
3.7 Strava interface centered on route and activity tracking with performance metrics. 24
3.8 MyFitnessPal interface focused on meal planning and nutrition logging. . . 24
3.9 Fitbit ecosystem view highlighting dashboard metrics and wearable-linked progress tracking. 24

4.1 Hierarchy of UML Diagrams --- static (class, component, object, package) and behavioral (use case, sequence, activity, state). 30
4.2 Use Case Diagram: Member --- registration, onboarding, workout management, diet tracking, AI coaching, and messaging. 32
4.3 Use Case Diagram: Coach --- registration, plan creation, conversation management, and proposal sending. 33
4.4 Use Case Diagram: Admin/System --- seed data, manage predefined plans, index RAG knowledge base. 34
4.5 Sequence Diagram: User Registration --- client sends credentials, AuthLogic validates, creates user, returns tokens. 35
4.6 Sequence Diagram: Email OTP Verification --- client requests OTP, AuthLogic generates and sends via MailRepo, client verifies. 35
4.7 Sequence Diagram: AI Chat with Plan Generation --- message flows through ChatAgent, AIProvider calls context and plan tools, returns streamed response. 36
4.8 Sequence Diagram: Coach Sends Proposal --- coach sends workout proposal, DirectMessageLogic persists and broadcasts to member via SSE. 36
4.9 Sequence Diagram: Member Accepts Proposal --- member accepts, plan is forked for member, coach notified via SSE. 36
4.10 Class Diagram: Full Domain Model --- User, MemberInfo, CoachInfo, Plan, PlanSession, PlanBlock, BlockItem, Exercise, Equipment, DietPlan, Meal, DirectConversation, DirectMessage, ChatConversation. 37
4.11 Activity Diagram: Registration & Onboarding --- register, verify OTP, complete onboarding, role-based routing to Member or Coach flow. 39
4.12 Activity Diagram: Member Daily Flow --- log workouts, log nutrition with AI food analysis, chat with AI coach, message coach, manage proposals. . . 40
4.13 Activity Diagram: Coach Daily Flow --- create workout/diet plans, message members, send proposals, mute conversations. 40
4.14 Entity-Relationship Diagram: Full Database Schema --- User, MemberInfo, CoachInfo, Plan, PlanSession, PlanBlock, BlockItem, Exercise, Equipment, DietPlan, Meal, DirectConversation, DirectMessage, SessionLog, DailyNutritionLog. 41

4.15 AI Chat Agent Tool Flow --- member message processed by Gemini LLM, which calls context, plan_manager, and rag_search tools, all backed by PostgreSQL. 42
4.16 RAG Embedding & Search Flow --- member query embedded via nomic-embed-text, vector similarity search via pgvector, top-K results summarized by Gemini LLM. 43
4.17 Plan Manager Approval Flow --- LLM drafts plan, creates approval token, member confirms via chat, token verified, plan persisted. 43
4.18 Layered Backend Architecture --- Routes → Logic → Repositories → Infrastructure, with AI Agents and Tools interacting with Logic and Repos. . 45
4.19 Full Component Diagram --- Mobile/Web clients connect to ElysiaJS API server, which orchestrates AI Layer (Chat, FoodVision, Title agents, RAG Search, Plan Manager), Data Layer (PostgreSQL + pgvector, S3), and External Services (Gemini API, Ollama, SMTP). 46

5.1 Backend layered architecture: Routes → Logic → Repos → Infra. 50
5.2 Mobile client layered architecture: Presentation → Application Logic → State Management → API Integration. 52
5.3 Development pipeline from business logic to mobile consumption --- Elysia route definitions generate OpenAPI spec, Scalar UI, MCP tools, and typed mobile API classes. 54
5.4 OpenAPI Documentation --- interactive Scalar UI interface at the `/docs` endpoint. 57
5.5 Fly.io App Dashboard --- overview of deployed application operational status, VM instances, logs, and environment configuration. 58
5.6 Grafana Dashboard --- real-time visualizations of CPU, memory, network traffic, and request latency for the deployed application. 59
5.7 Login Screen --- email and password authentication with registration option. 60
5.8 Onboarding Screens --- step-by-step profile creation and initial setup for personalized recommendations. 60
5.9 Dashboard/Home Screen --- key metrics, daily streaks, and quick-access features. 61
5.10 Workout Plans List --- browsing pre-designed and custom workout plans with duration, intensity, and goal indicators. 62
5.11 Create Workout Plan --- defining sessions, blocks, and exercises with sets, reps, rest periods, and target weights. 63
5.12 Active Workout Session --- live tracking interface with current exercise, target metrics, and set logging controls. 64

5.13 Diet Plan Overview --- daily nutritional goals for calories, protein, carbs, fats, and water intake. 65
5.14 Meal Logging --- manual entry and AI-powered food image analysis for automated nutritional estimation. 66
5.15 Progress Charts --- line charts for weight trends, bar charts for workout frequency, and personal record tracking. 67
5.16 AI Coach Chat --- natural language conversations with personalized guidance on workouts, nutrition, and form. 68
5.17 Messaging Interface --- real-time communication between members and human coaches with persisted conversation history. 69
5.18 Equipment Selection --- specifying available equipment for filtering and recommending compatible exercises. 70
5.19 Profile Screen --- personal information, fitness goals, notification settings, and account management. 71

Introduction

0.1 General Introduction

Physical fitness and balanced nutrition are essential components of a healthy lifestyle. Regular exercise helps improve strength, endurance, body composition, cardiovascular health, and mental well-being, while a well-managed diet supports energy balance, recovery, muscle growth, and long-term health. In recent years, interest in gym training, home workouts, calorie control, and personalized diet planning has increased significantly, especially among people who want to improve their physical appearance, prevent health problems, or build sustainable habits. However, achieving visible and lasting progress requires more than simply exercising or following a temporary diet; it requires continuous monitoring, structured planning, and the ability to understand how training and nutrition affect the body over time.

Traditional workout and diet tracking methods often make this process difficult. Many users still rely on handwritten notebooks, scattered notes, generic workout plans, manual calorie calculations, or advice that is not adapted to their personal goals, fitness level, equipment, schedule, or physical limitations. These methods can be inconsistent and incomplete, making it hard to evaluate progress accurately, identify training plateaus, adjust exercise intensity, or understand whether nutrition habits support the desired objective. In addition, beginners may struggle to choose suitable exercises, organize weekly sessions, estimate food intake, or interpret changes in weight and performance. As a result, users may lose motivation, follow unsuitable programs, or fail to improve despite regular effort.

The development of mobile technologies and artificial intelligence offers an opportunity to improve this situation. Smartphones make it possible to record workouts, meals, body measurements, and progress indicators in a simple and accessible way, while AI agents can analyze user context and provide more personalized guidance. In this work, we propose an intelligent mobile fitness coaching application powered by an AI agent. The application aims to help users track their workout and diet progress, visualize their development, receive personalized recommendations, and improve their training and nutrition decisions over time. By combining structured fitness logging, progress monitoring, nutrition support, and context-aware AI coaching, the proposed solution seeks to provide a coherent platform that supports users in reaching their fitness goals more safely and effectively.

0.2 Proposed Solution and Objectives

The proposed solution is a mobile fitness coaching application powered by an AI agent. It is designed to centralize workout tracking, diet monitoring, body measurements, progress visualization, and personalized guidance in a single platform. Instead of treating training and nutrition as separate activities, the application connects user goals, fitness level, available equipment, workout history, food intake, and progress indicators so that users can better understand their development and make improved decisions over time. The AI agent supports this process by analyzing the user's context and generating recommendations related to workouts, nutrition, recovery, and habit improvement.

The main objectives of this work are as follows:

1. To study the main concepts related to AI agents, large language models, context management, retrieval, and tool use in order to understand how an intelligent coaching assistant can be designed.

2. To analyze the essential factors that influence gym progress and fitness outcomes, including training volume, intensity, exercise selection, progressive overload, consistency, recovery, nutrition, and user goals.

3. To compare existing fitness tracking applications in order to identify their strengths, limitations, and gaps, especially regarding personalization, workout tracking, nutrition management, and integrated coaching support.

4. To design a complete mobile fitness coaching system that includes user profiles, workout plans, diet tracking, progress monitoring, coach-member communication, database architecture, and AI services.

5. To implement a functional mobile application powered by an AI agent that helps users log workouts and meals, track progress, receive personalized guidance, and improve their fitness decisions.

0.3 Manuscript Organization

This manuscript is organized into five main chapters:

1. **Chapter 1: AI Agents.** This chapter introduces Large Language Models, AI agents, agent workflows, the difference between standalone LLMs and AI agents, and the context-management techniques needed to build a reliable AI coaching assistant.

2. **Chapter 2: Gym Progress Tracking and Fitness Management.** This chapter presents the fitness and nutrition concepts required by the application, including workout monitoring, training principles, recovery, nutrition management, consistency, and context-aware AI coaching.

3. **Chapter 3: Similar Fitness Tracking Applications.** This chapter reviews and compares existing mobile fitness applications in Algeria, the Arab world, and the global market in order to identify useful features and limitations that guide the proposed solution.

4. **Chapter 4: The Design of the Proposed Application.** This chapter describes the system requirements, UML diagrams, database architecture, AI models and services, approval flow, and layered application architecture used to prepare the implementation.

5. **Chapter 5: Implementation of the Proposed Application.** This chapter explains the development environment, technologies, backend and mobile architectures, application interfaces, API integration, deployment, and the implemented AI-powered fitness tracking features.

Chapter 1

AI Agents

1.1 Introduction

Integrating Large Language Models (LLMs) with retrieval mechanisms, external tools, and persistent memory significantly improves the capability of intelligent systems to reason, plan, and act in dynamic environments. LLMs provide natural-language understanding and generation, whereas AI agents organize these abilities within structured workflows that can gather context, invoke tools, and maintain continuity across interactions. This chapter examines LLMs, AI agents, their key differences, and context-management techniques, illustrating how these concepts form the technical foundation for the proposed AI fitness coach before the next chapter defines the fitness knowledge that the coach must use.

1.2 Large Language Models (LLMs)

1.2.1 Definition

A Large Language Model (LLM) is a deep neural network, typically based on the transformer architecture, that has been trained on massive text corpora to capture patterns of language, semantics, and general world knowledge [1]. LLMs can generate fluent text, answer questions in natural language, summarize documents, translate between languages, and support a variety of text-based tasks. Their knowledge is encoded in billions of learned parameters, which are fixed once training is complete and cannot be updated without retraining or fine-tuning [1].

1.2.2 LLM Components

The operation of an LLM depends on several interconnected components (adapted from Milvus AI Quick Reference):

1. **Data**: Large and diverse datasets used for pre-training and, optionally, fine-tuning the model.

2. **Tokenizer**: A component that maps raw text into discrete tokens (subwords, words, or characters) that the model can process.

3. **Model Architecture**: The neural network structure (for example, a transformer) that processes tokenized inputs through multiple layers of computation [1].

4. **Parameters**: The internal weights and biases learned during training, often numbering in the billions.

5. **Training Infrastructure**: High-performance hardware (such as GPUs or TPUs), distributed training frameworks, and specialized software that enable efficient training of large models.

6. **Inference Engine**: The runtime system that loads the trained model and generates outputs for new inputs, optimized for latency and scalability.

7. **Evaluation**: Metrics and benchmarks (for example, task accuracy, perplexity, or domain-specific test suites) used to assess model performance [2].

8. **Deployment**: The processes and tooling for serving the model in production, including APIs, load balancing, monitoring, and security controls.

These components form a pipeline from raw data through training and evaluation to operational deployment, enabling LLMs to be integrated into real-world applications.

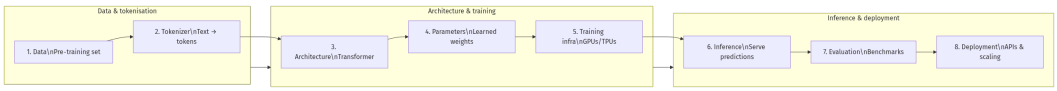


Figure 1.1: LLM components pipeline: from data and tokenization through architecture and training to inference, evaluation, and deployment.

1.2.3 LLM Usage

LLMs are widely used for tasks such as:

- Text generation and completion
- Question answering and dialogue
- Summarization of long documents
- Translation between languages

- Code generation and code-related assistance
- Conversational agents and chat interfaces

Despite their versatility, LLMs have important limitations. Their knowledge is static and cannot be updated online, which may lead to outdated or inaccurate responses [2]. Moreover, they can generate convincing but incorrect information (hallucinations) when asked to reason about facts beyond their training distribution [2].

1.3 What is an AI Agent?

An AI agent is an autonomous software system that leverages one or more LLMs together with external tools, retrieval mechanisms, and persistent memory to solve complex, multi-step tasks. Unlike standalone LLMs, agents can reason about objectives, plan actions, and interact with dynamic environments by invoking APIs, executing code, and querying external data sources on behalf of users [3].

1.3.1 General Workflow

A typical AI agent follows a structured workflow:

1. **Receive Task/Query**: The agent receives a user request or high-level objective, often expressed in natural language.
2. **Analyze and Plan**: It parses the intent, decomposes the task into subtasks, and determines an action plan.
3. **Context Gathering**: The agent retrieves relevant information from documents, databases, or external systems, combines it with system instructions, and may consult prior interactions stored in memory.
4. **Action Execution**: Based on the plan, the agent executes actions such as API calls, code execution, database queries, or web searches.
5. **Synthesize and Respond**: After collecting and verifying results, the agent integrates them into a coherent response, checks for consistency, and returns a final answer to the user.

This workflow enables agents to act as orchestrators that combine LLM reasoning with external capabilities rather than relying solely on in-context knowledge [3].

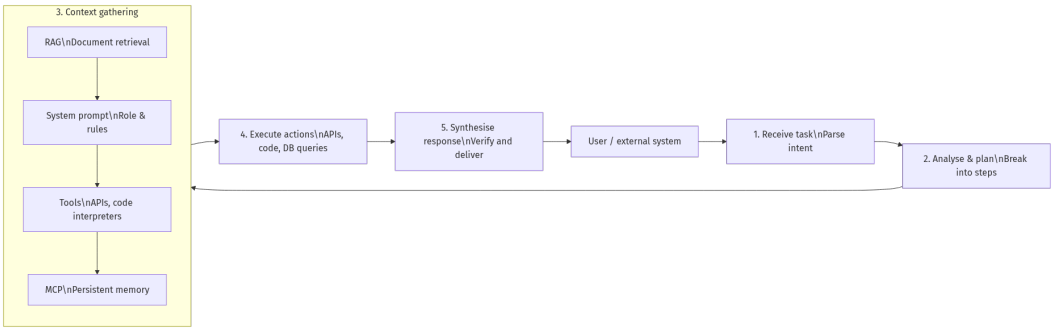


Figure 1.2: General AI agent workflow: task reception, analysis and planning, context gathering, action execution, and response synthesis.

1.3.2 Applications

AI agents are deployed in many domains, including:

- Research and information-gathering assistants
- Personal productivity agents (scheduling, email drafting, task management)
- Code generation, refactoring, and debugging tools
- Decision support systems for analysts and executives
- Enterprise knowledge-management systems that surface internal documents
- Real-time data-analysis and monitoring agents

These applications exploit the agent's ability to reason over structured and unstructured data, invoke external services, and maintain state across multiple interactions [3].

1.4 Examples

Several modern systems illustrate how AI-agent concepts are applied in practice. These examples show that agents are not limited to chat interfaces: they can coordinate tools, interact with files and external services, execute actions, and maintain task context across several steps.

OpenClaw is an example of a general-purpose autonomous AI agent platform. It is designed to connect language models with tools and communication channels so that users can request practical tasks in natural language. Instead of only generating advice, an OpenClaw-style agent can be configured to read context, call services, manage workflows, and execute actions under user-defined permissions. This makes it representative of personal and productivity agents that combine reasoning, memory, and tool use.

OpenCode is an example of a specialized coding agent. It focuses on software-development workflows such as understanding a codebase, modifying files, generating code, debugging errors, and running development commands. This type of agent demonstrates how LLM reasoning becomes more useful when it is combined with repository context, terminal tools, test execution, and iterative feedback from build or lint results.

Together, OpenClaw and OpenCode demonstrate two important categories of AI agents: broad automation agents that operate across many everyday tasks, and domain-specific agents optimized for a particular professional workflow. Both examples highlight the central idea of agentic AI: the model does not only answer questions, but also plans, uses tools, observes results, and refines its actions until the task is completed.

1.5 LLMs vs. AI Agents: Key Differences

Table 1.1: Comparison between standalone LLMs and AI agents

Feature	**LLMs Alone**	**AI Agents**
Core Functionality	Text generation, Q&A, summarization	Autonomous task-solving, planning, and acting [3]
Context Management	Limited to the current prompt window	Dynamic: retrieval, tools, memory, and prompts
Tool Integration	None (integration must be done externally)	Native via API calls, code execution, search
Memory	Stateless; no cross-session persistence	Persistent via user profiles and history
Trustworthiness	Higher risk of hallucinations and outdated info	Lower risk due to external verification
Typical Applications	Chatbots, translation, simple Q&A [1], [2]	Research, coding, decision support, automation

This table highlights that AI agents extend LLMs by adding structured workflows, external grounding, and memory, at the cost of greater architectural complexity and orchestration overhead.

1.6 How AI Agents Manage Context

AI agents overcome the limitations of LLMs by combining multiple context-management techniques into a coherent pipeline.

1.6.1 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) allows agents to query external knowledge bases --- such as document repositories, databases, or web indexes --- before generating an an-

swer. Given a user query, the agent retrieves a set of relevant documents or passages and then conditions the LLM's response on both the original query and these retrieved materials. This approach grounds the model's output in verifiable data sources, thereby reducing the likelihood of hallucinations and improving factual accuracy [4].

1.6.2 System Prompts

System prompts define the agent's role, behavior, and output format. They encode instructions, constraints, and safety guardrails that are injected into every reasoning step, ensuring consistent behavior and alignment with organizational policies. Well-designed system prompts help guide the agent toward reliable, domain-appropriate responses and reduce undesired or unsafe outputs [5].

1.6.3 Tool Calling

Agents can invoke external tools --- such as APIs, code interpreters, calculators, or web search engines --- to obtain live data, perform computations, or interact with other systems. Tool calling extends the agent's capabilities beyond static knowledge, enabling it to respond to queries that depend on current information (for example, stock prices, weather, or inventory levels) or on programmatic processing [3].

1.6.4 Managed Context Providers (MCPs)

Managed Context Providers (MCPs) store persistent information about users, including profiles, preferences, and interaction histories. By integrating MCPs into the agent's context, the system can personalize responses, maintain continuity across sessions, and adapt to evolving user needs. This persistent memory layer is critical for long-running applications such as personal assistants or enterprise-support agents [6].

1.7 The Importance of Rich Context for AI Agents

Providing AI agents with detailed and accurate context about user goals, preferences, and constraints is essential for several reasons:

- It reduces hallucinations and factual errors by grounding reasoning in retrievable and verifiable information [4], [7].
- It enables personalization, allowing agents to tailor responses to specific users, roles, or domains [6], [7].

- It supports safety and compliance by ensuring that outputs conform to organizational rules and regulatory requirements [5], [7].
- It improves the reliability of real-world outcomes, especially in complex, multi-step workflows [4], [6], [7].

In practice, the richer and more structured the context available to an agent, the more effectively it can reason, plan, and act in ways that align with user expectations and operational constraints.

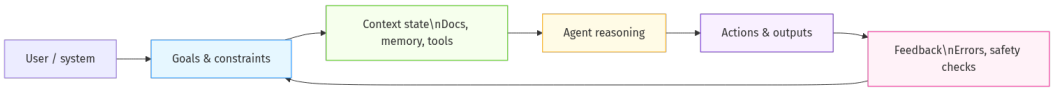


Figure 1.3: Context-driven agent reasoning loop: goals and constraints feed context state, which drives agent reasoning, actions, and feedback.

1.8 Conclusion

Integrating LLMs with retrieval, tool calling, system prompts, and managed context provides a robust framework for building reliable AI agents. This approach reduces the limitations of standalone language models by grounding responses in relevant information and adapting them to user goals and constraints [3], [4], [5], [6], [7]. Incorporating this framework into the proposed fitness application enables the AI component to operate as a context-aware coach, making the fitness variables discussed in the next chapter essential for safe and personalized guidance.

Chapter 2

Gym Progress Tracking and Fitness Management

2.1 Introduction

Combining systematic workout monitoring, nutrition management, and AI-driven personalization significantly improves the way users pursue sustainable fitness goals. Workout tracking provides measurable performance data, whereas nutrition and recovery indicators help explain progress and guide safer adjustments. This chapter examines the main principles of gym progress tracking, including exercise selection, training volume, progressive overload, recovery, nutrition, consistency, and AI coaching, showing how the agent foundation introduced in Chapter 1 must be supplied with accurate fitness context.

2.2 Importance of Fitness Tracking

2.2.1 Core Functions

Effective gym progress tracking applications deliver three essential functions [8].

Table 2.1: The three core functions of gym progress tracking

Function	Benefits	Key Metrics
Performance Tracking	Monitor strength gains, endurance improvements, workout consistency	Weight lifted, reps, sets, exercise duration, max capacity
Health Optimization	Maintain nutrition, body composition, injury prevention	Calories consumed, macronutrients, body weight, body fat percentage, injury logs
Motivation & Accountability	Visual progress, goal achievement, behavioral reinforcement	Workout streaks, milestone achievements, progress graphs, personal records

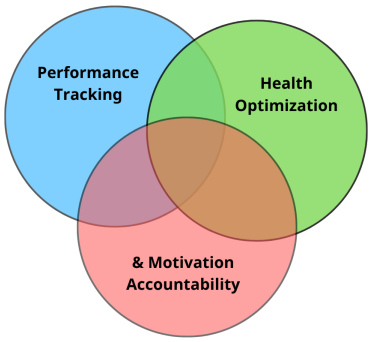


Figure 2.1: A visual diagram showing three interconnected circles representing Performance Tracking, Health Optimization, and Motivation & Accountability with icons and key metrics beneath each.

Research shows consistent tracking significantly improves fitness goal achievement through data-driven insights and behavioral reinforcement [9].

2.3 Core Principles of Gym Training

2.3.1 Exercise Selection

Effective programs include compound movements (squats, deadlifts, bench press), isolation exercises (bicep curls, leg extensions), functional training, and cardiovascular work [10].

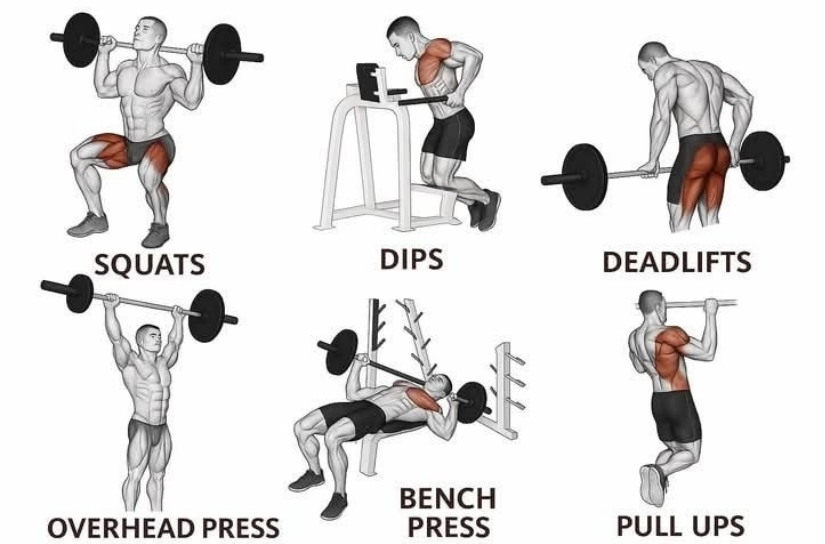


Figure 2.2: A grid showing different exercise types with visual icons: compound movements (squat, deadlift, bench press), isolation exercises (bicep curl, leg extension), functional training, and cardio HIIT workouts. Source adapted from the exercises dataset [11].

2.3.2 Training Volume and Intensity

Training volume (sets × reps × weight) and intensity (% of maximum effort) drive adaptation. Track sets, reps, weight, rest periods, RPE, and frequency [10].



Figure 2.3: A line graph showing the relationship between training volume (X-axis) and intensity (Y-axis) over weeks/months of training.

2.3.3 Rest and Recovery

Recovery includes 7--9 hours sleep, rest days, active recovery, and post-workout nutrition [12].

2.3.4 Progressive Overload

Gradually increase demands through weight, reps, sets, reduced rest, or improved form [10].

2.3.5 Periodization

Strategic training phases prevent plateaus: linear (gradual intensity increase), undulating (weekly variations), block (focused adaptation phases) [13].

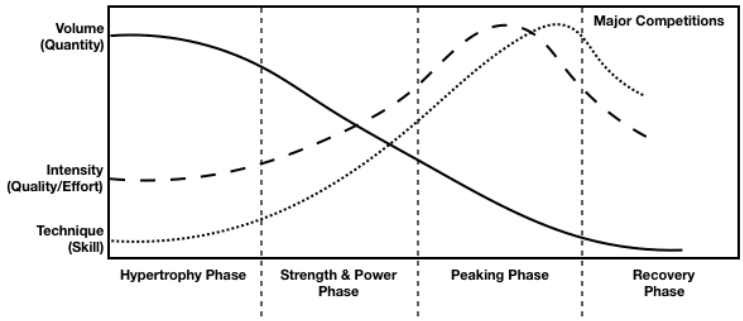


Figure 2.4: Comparison of three periodization models showing Linear (gradual progression), Undulating (wave-like intensity), and Block (distinct phases). Source adapted from the referenced periodization model image [14].

2.3.6 Proper Form

Correct technique maximizes effectiveness and prevents injury [12].



Figure 2.5: Before/after form comparison for squat showing correct vs. incorrect technique with key form annotations.

2.3.7 Nutrition Management

Track macronutrients, micronutrients, hydration, meal timing, and caloric balance [15].

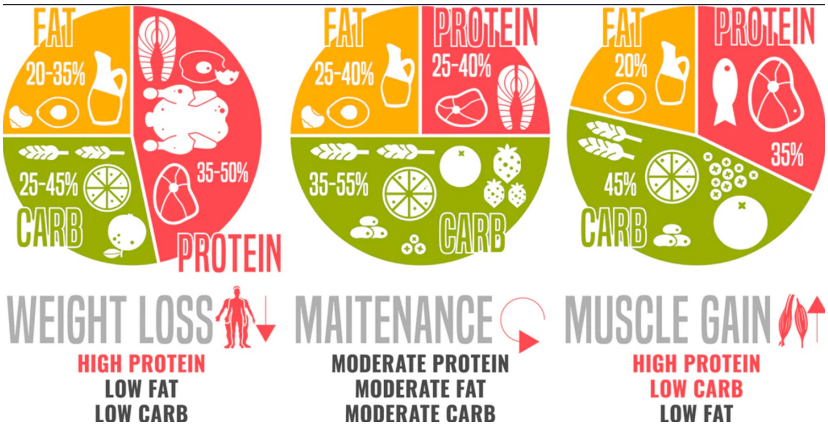


Figure 2.6: Macronutrient breakdown pie chart showing optimal ratios for muscle gain, fat loss, maintenance. Source adapted from Beehive Meals' macro-balanced diet guide [16].

2.3.8 Consistency

Promote adherence through habit tracking, reminders, progress visualization, and gamification [9].

2.4 AI in Fitness Coaching

2.4.1 Human Coach vs. AI Coach

A gym progress tracking system should not treat human coaching and AI coaching as direct replacements for one another. A human coach remains important for motivation, technique correction, emotional support, and safety supervision, while an AI coach is useful for fast analysis, continuous availability, and data-driven personalization. Table 2.2 summarizes the recommended balance between both coaching approaches.

Table 2.2: Comparison between human coach and AI coach in fitness management

Criterion	**Human Coach**	**AI Coach**
Personalization	Adapts plans using observation, experience, and direct communication	Adapts recommendations using logged workouts, nutrition records, goals, and user context
Technique and safety	Can observe posture, correct form, and stop unsafe execution immediately	Can provide guidance and warnings, but cannot fully verify movement quality without sensors or video analysis
Availability	Limited by schedule, location, and session cost	Available at any time for questions, reminders, summaries, and quick plan adjustments
Motivation	Provides emotional support, accountability, and encouragement through human interaction	Provides reminders, progress feedback, streaks, and automated motivational messages
Cost and scalability	More expensive and harder to scale for many users	Lower cost per user and easier to scale across a large user base
Best use case	Complex cases, beginners needing supervision, injury risk, and advanced coaching decisions	Daily tracking, routine recommendations, progress summaries, and first-level personalized guidance

In general, a hybrid coaching model is often stronger than relying on only one approach: the AI coach can handle continuous tracking, reminders, nutrition support, and first-level recommendations, while the human coach can provide validation, advanced decisions, and direct supervision when needed.

2.4.2 Personalized Recommendations

AI analyzes performance data to recommend optimal exercises aligned with goals and limitations [17].

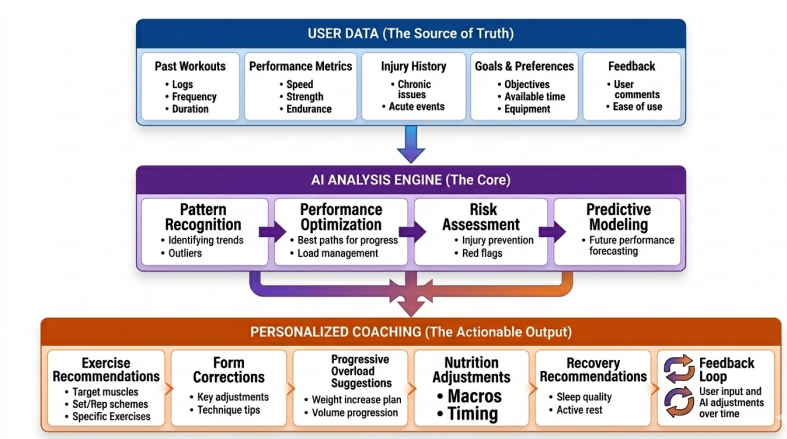


Figure 2.7: App interface showing AI recommendations with explanations, alternatives, confidence scores.

2.5 Context-Aware AI Coaching Example

User: 32-year-old male, 85 kg, 178 cm, Blida Algeria. Goal: muscle gain + fat loss. Mild lower-back pain. Home gym (dumbbells 5--30 kg, pull-up bar, bands, mat). Budget: 25,000 DZD/month food. 4 days/week, 45--60 min sessions. High-protein preference, no fish/broccoli, simple recipes < 20 min [18].

AI Workflow:

1. **System Prompt**: Certified trainer role, evidence-based, back-safe modifications, DZD costs [12].
2. **Retrieval**: Home hypertrophy programs with lower-back modifications [10], [12].
3. **Nutrition**: TDEE $\sim$ 2,850 kcal, target 2,500 kcal deficit, 180 g protein using local foods [15].

Program Summary:

- **Workouts**: Push/Pull/Legs/Upper split, back-safe (neutral spine, controlled RDLs).
- **Meals**: $\sim$ 850 DZD/day --- oats+whey breakfast, chicken+rice lunch, beef/couscous dinner, yogurt+almond snacks.

2.6 Conclusion

Integrating workout monitoring, nutrition awareness, recovery management, and AI personalization provides a practical framework for sustainable gym progress. This approach helps transform raw user activity into safer recommendations and clearer performance feedback [9],

[17], [18]. Incorporating these principles into the proposed application requires structured workout data, personal context, and progress indicators, which the following chapter uses as criteria for evaluating existing mobile fitness applications.

Chapter 3

Similar Fitness Tracking Applications

3.1 Introduction

Combining mobile technologies with fitness tracking has made workout logging, nutrition monitoring, progress visualization, and coaching support more accessible to everyday users. Mobile applications provide convenient interfaces for daily input, whereas wearables and dashboards improve feedback through continuous data and visual summaries. This chapter examines representative fitness applications in Algerian, Arab, and international contexts, comparing their strengths and limitations against the tracking and personalization requirements defined in Chapter 2.

3.2 Mobile Applications

3.2.1 Definition

A mobile fitness application is a digital tool for smartphones, tablets, or wearables that helps users track workouts, nutrition, and health metrics. It can be native, web-based, or hybrid, and usually includes logging tools, progress charts, and device integration.

In gym progress tracking, it serves as a central hub that combines manual entries and wearable data to generate clear insights, reminders, and personalized guidance.

3.2.2 Importance of Mobile Fitness Applications

Mobile fitness apps have become essential tools for people who want to manage their training and health efficiently. Their importance can be summarized as follows:

1. **Unified Training Hub**: Mobile applications consolidate workouts, nutrition, body measurements, and health data into a single platform. This omni-channel approach

ensures that information collected from the gym, home workouts, and outdoor activities is synchronized and easy to review.

2. **Improved Accessibility**: Users can access their training plans, historical records, and performance analytics at any time. Real-time notifications about scheduled workouts, milestones, or deviations from plan help maintain accountability and prevent missed sessions.

3. **Enhanced User Experience**: Modern fitness applications present data through intuitive dashboards, charts, and summaries. This visual feedback makes complex information --- such as volume trends, recovery status, or macro distribution --- clear and actionable, thereby improving the overall training experience.

4. **Increased Adherence and Motivation**: Built-in reminders, streak counters, badges, and goal-tracking features encourage users to stay consistent. Push notifications about upcoming workouts or progress milestones help sustain engagement and long-term adherence to training programs.

5. **Brand and Community Building**: For gyms, coaches, and fitness brands, mobile apps strengthen brand recognition and customer loyalty. Personalized programs, branded interfaces, and community challenges make it easier for users to remain connected to a particular training ecosystem.

6. **Better Communication and Coaching**: In-app messaging, video feedback, and progress reports enable remote coaches and personal trainers to interact efficiently with clients. This two-way communication supports program adjustments, form corrections, and timely encouragement.

7. **Data-Driven Coaching and Marketing**: Mobile applications facilitate the collection of anonymized training and engagement data, which can be used to improve workout templates, refine recommendation algorithms, and tailor marketing campaigns to user preferences and behavior patterns.

8. **Integrated Services and Payments**: Many fitness apps integrate subscriptions, coaching packages, class bookings, and digital products directly into the interface. Users can purchase customized programs, book sessions, or upgrade plans without leaving the app, simplifying the entire service pipeline.

3.3 Similar Fitness Applications

This section presents representative examples of mobile fitness applications that demonstrate different approaches to workout tracking, nutrition management, and holistic health monitor-

ing. While the specific brand names and feature sets may vary, the goal is to highlight core concepts that inform the design of a comprehensive gym progress tracking system.

3.3.1 In Algeria

The Algerian fitness app landscape includes both gym-specific tools and local digital fitness platforms. Based on publicly available app pages and platform descriptions, the following examples are relevant:

1. **Power Fitness Club Dz**. Official app for a multi-branch gym network in Algeria. It focuses on club discovery, membership management, and access to personalized programs [19].



Figure 3.1: Power Fitness Club Dz mobile interface.

2. **BGR SMART GYM**. Described as an Algerian smart gym management solution with mobile access for gym clients and integration with access-control workflows [20].

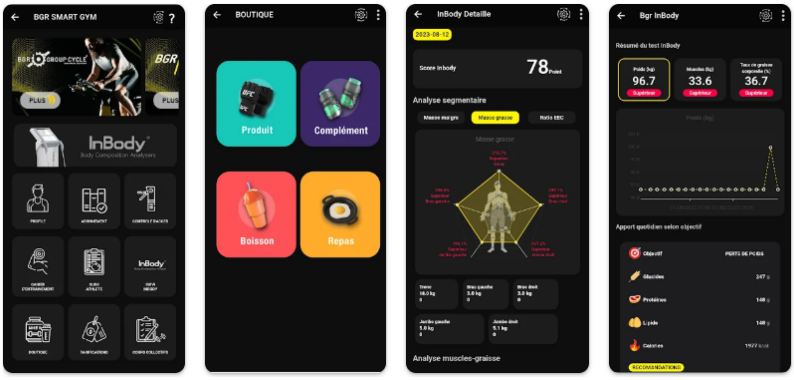


Figure 3.2: BGR SMART GYM interface with integrated gym operations.

3. **MayStrong**. Algeria-focused fitness and nutrition platform offering training programs, exercise resources, calorie tools, and gym/wellness directory features [21].

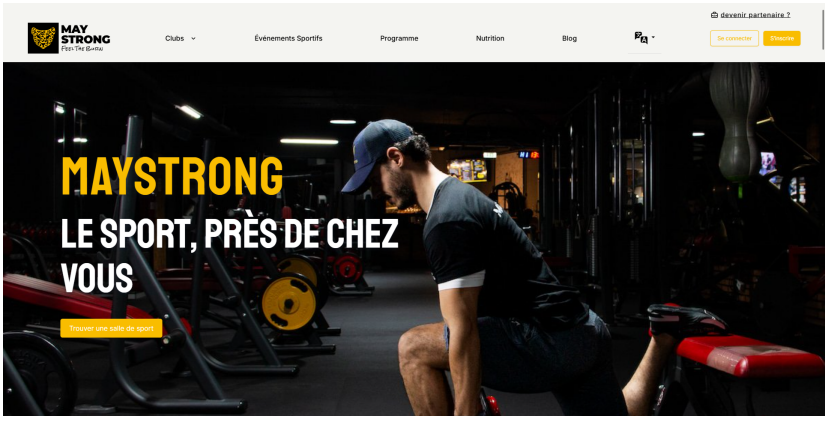


Figure 3.3: MayStrong application interface combining workout, nutrition, and local ecosystem features.

These examples show that Algeria has practical app usage centered on membership operations, coaching content, and localized French/Arabic-friendly fitness support.

3.3.2 In Arab World

Across Arab countries, successful fitness apps are usually Arabic-first and tailored to regional food culture and behavior patterns.

1. **Kam Calorie**. AI voice-first calorie tracking app built for Arabic speakers, with support for MENA cuisine and dialect-aware food logging [22].

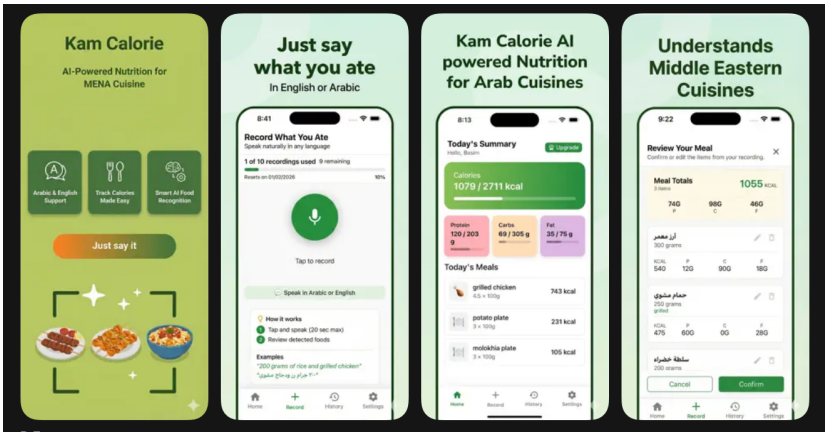


Figure 3.4: Kam Calorie interface emphasizing Arabic voice-based nutrition tracking.

2. **Rashaqa**. Arabic health and fitness app that combines calorie counting, step tracking, home workouts, diet plans, and hydration reminders [23].



Figure 3.5: Rashaqa interface integrating calorie tracking, exercise guidance, and hydration reminders in Arabic.

3. **Soraate**. Arabic calorie and fitness app with traditional meal tracking, activity logging (steps, water, weight), and community motivation features [24].



Figure 3.6: Soraate interface for Arabic calorie logging and community motivation.

These apps demonstrate a strong regional direction: localized language, culturally specific nutrition databases, and all-in-one daily habit tracking.

3.3.3 Worldwide

At the global level, fitness ecosystems are usually built around one app per major need: activity tracking, nutrition tracking, and wearable-based health monitoring.

1. **Strava**. Widely used for activity tracking and performance analysis in running, cycling, and outdoor training [25].

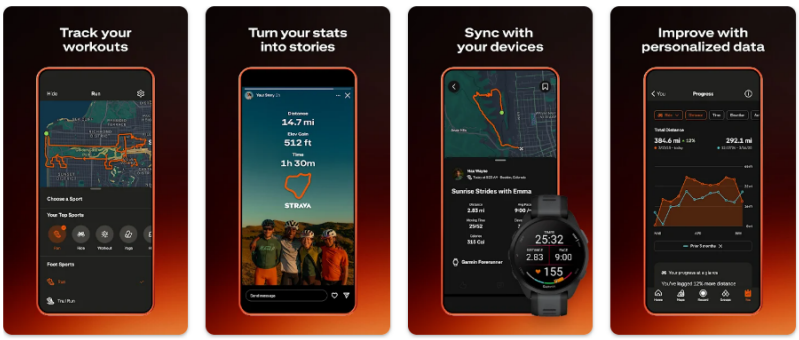


Figure 3.7: Strava interface centered on route and activity tracking with performance metrics.

2. **MyFitnessPal**. Global calorie and macronutrient tracking app with a large food database and meal logging tools [26].

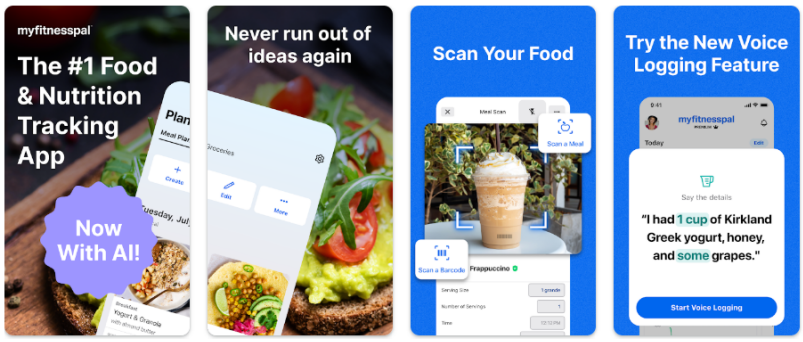


Figure 3.8: MyFitnessPal interface focused on meal planning and nutrition logging.

3. **Fitbit**. Wearable-centered platform focused on steps, heart-rate trends, sleep metrics, recovery indicators, and long-term health habits [27].

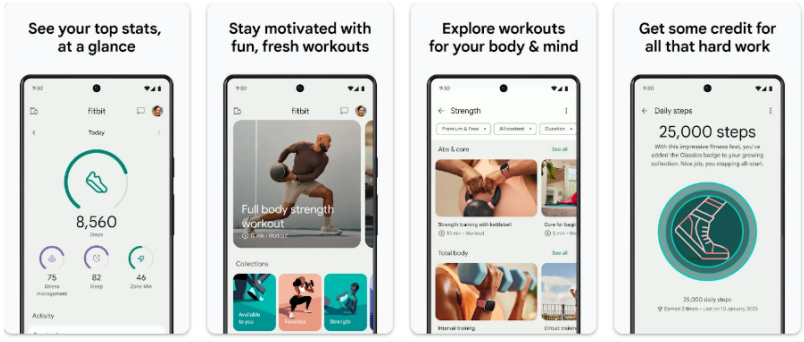


Figure 3.9: Fitbit ecosystem view highlighting dashboard metrics and wearable-linked progress tracking.

Across regions, one common insight remains clear: users get better outcomes when training, nutrition, and recovery data are connected in one coherent experience.

Table 3.1: Comparative summary of similar fitness applications by region

Application	Main Focus	Key Strength	Main Limitation
Power Fitness Club Dz	Gym membership and club engagement	Strong connection with local gym operations	Less emphasis on advanced analytics
BGR SMART GYM	Smart gym management and access workflows	Integration with gym administration and access control	Primarily oriented to gyms using the same ecosystem
MayStrong	Training + nutrition + local ecosystem	Combines coaching content with local fitness directory	Platform depth may vary by feature usage
Kam Calorie	Arabic voice nutrition tracking	Fast, dialect-aware meal logging for MENA cuisine	More nutrition-centric than full workout programming
Rashaqa	All-in-one daily fitness habits	Covers calories, steps, workouts, and reminders in Arabic	Can feel broad rather than specialized for athletes
Soraate	Arabic calorie and lifestyle tracking	Strong cultural fit and community-style motivation	Advanced training periodization is limited
Strava	Outdoor activity performance	Excellent activity analytics and social training network	Limited native nutrition tracking
MyFitnessPal	Calorie and macro management	Massive food database and flexible meal logging	Manual logging can be time-consuming
Fitbit	Wearable health and recovery monitoring	Strong passive tracking for sleep, heart rate, and steps	Best experience depends on compatible wearable devices

The comparison above clarifies why integrated systems are more effective: each app category solves a different part of the fitness journey, but no single standalone tool fully covers all needs for every user profile.

3.4 Conclusion

Integrating workout tracking, nutrition logging, progress visualization, and coaching support in one mobile platform provides a stronger user experience than separating these functions across different tools. The comparative study shows that many existing applications remain specialized, creating gaps in personalization, local adaptation, or data integration. Incorporating these findings into the proposed system leads directly to the design chapter, where the AI-agent principles of Chapter 1 and the fitness requirements of Chapter 2 are translated into requirements, UML models, database structures, and AI services.

Chapter 4

The Design of the Proposed Application

4.1 Introduction

Integrating UML modeling, database architecture, and AI service design provides a structured foundation for transforming the research findings into a functional fitness coaching application. UML clarifies system behavior and stakeholder interactions, whereas the database and architecture define how workout, nutrition, recovery, and user-context data will be stored and processed. This chapter examines the requirements, use case, sequence, class, activity, database, AI-service, and architecture designs, showing how the agent principles of Chapter 1, the fitness requirements of Chapter 2, and the application gaps identified in Chapter 3 are organized into an implementable system blueprint.

4.2 Requirements Specification

The design of the proposed application is guided by a clearly defined set of functional and non-functional requirements, distilled from the problem domain, the needs of the intended stakeholders, the agent capabilities needed for personalized coaching, the practical principles of gym progress tracking, and the limitations observed in comparable applications. These requirements serve as a contractual reference for system construction and validation while ensuring that the proposed system supports both intelligent recommendations and everyday usability.

4.2.1 Functional Requirements

The functional requirements enumerate the specific behaviors and capabilities that the system must deliver. Onboarding and profile management reflect the rich-context requirement identified in Chapter 1; workout planning, session logging, diet tracking, and progress monitoring reflect the training and nutrition principles discussed in Chapter 2; and coach-member

communication, dashboards, and unified API access respond to the fragmentation observed in Chapter 3.

Table 4.1: Functional requirements

ID	Requirement Description	Category
FR1	Members and coaches shall be able to register accounts and authenticate via email and password.	Authentication
FR2	The system shall verify user email addresses by transmitting a one-time password (OTP).	Authentication
FR3	Upon first access, users shall complete an onboarding process to establish their fitness profile (goals, experience, available equipment, etc.).	Onboarding
FR4	Members shall be able to browse, search, and activate predefined or custom workout plans.	Workout Management
FR5	The platform shall automatically generate personalized workout plans using AI, conditioned on the member's profile and available equipment.	AI Workout Planning
FR6	Members shall log individual workout sessions, recording performance metrics at the level of sessions, blocks, and exercise items.	Progress Tracking
FR7	A nutrition module shall support the formulation of diet plans with explicit macronutrient and caloric targets, adaptable to user goals.	Diet Management
FR8	Members shall be able to record meals and water intake, with optional AI-powered food image analysis for automated nutritional estimation.	Nutrition Logging
FR9	A conversational AI agent shall deliver real-time fitness and nutrition advice, leveraging a retrieval-augmented generation (RAG) knowledge base of workout plans.	AI Coaching
FR10	Coaches shall create, manage, and share workout and diet plans with assigned members.	Coach Tools
FR11	Coaches and members shall communicate through a direct messaging system that supports workout and diet plan proposals, which members may accept or decline.	Direct Messaging
FR12	The system shall expose a fully documented REST API to support client applications and a Model Context Protocol (MCP) server for AI agent integration.	Integration
FR13	Administrative services shall initialize the exercise and equipment catalogues, curate predefined plans, and manage the RAG embedding index.	System Administration

4.2.2 Non-Functional Requirements

Non-functional requirements define quality attributes and constraints that the system must satisfy.

Table 4.2: Non-functional requirements

ID	Requirement Description	Attribute
NFR1	The REST API shall respond to typical requests within 300 milliseconds under normal load.	Performance
NFR2	Authentication and authorization shall be enforced via short-lived JWT access tokens and refresh tokens, with all credentials transmitted over HTTPS.	Security
NFR3	The system shall remain available and responsive even when external AI providers experience temporary delays, employing asynchronous stream processing where appropriate.	Reliability
NFR4	The database schema shall be designed in third normal form to ensure data consistency and avoid redundancy.	Data Integrity
NFR5	The application architecture shall support independent deployment of components (API server, AI agents, Ollama service) to facilitate maintenance and scaling.	Maintainability
NFR6	All external integrations (email, AI APIs, object storage) shall be abstracted behind repository interfaces to allow substitution without affecting business logic.	Modifiability
NFR7	The source code shall be structured as a monorepo with shared packages, managed by TurboRepo, to promote code reuse and streamline builds.	Development Efficiency
NFR8	User-facing operations must be intuitive: AI-generated plans require explicit user approval before activation, and sensitive actions are confirmed through secure, auto-expiring tokens.	Usability

4.3 UML

4.3.1 UML Definition

The Unified Modeling Language (UML) is a general-purpose, standardized modelling language extensively employed in software engineering. It provides a collection of graphical

notations that facilitate the specification, construction, visualization, and documentation of software-intensive systems. In this project, UML serves as the principal instrument for expressing the system's structural and behavioral facets in a manner that is both rigorous and accessible to diverse stakeholders.

4.3.2 The Importance of Using UML

The adoption of UML yields several significant benefits for the design and subsequent evolution of the application:

- It furnishes a professional mechanism for capturing design decisions that can be referenced throughout the development lifecycle.
- UML diagrams reduce the cognitive burden associated with understanding complex systems, thereby accelerating onboarding and communication among team members.
- As the project grows, the models can be modified with minimal cost, serving as a living documentation of the system architecture.
- The notation is independent of any particular programming language or development methodology, ensuring that the design remains valid even if implementation technologies change.
- By establishing a common visual vocabulary, UML bridges the gap between analysts, designers, and developers, mitigating the risk of misinterpretation.

4.3.3 UML Diagram Taxonomy

UML diagrams are categorized into two broad families: **static** diagrams, which depict the structural elements of a system, and **behavioral** diagrams, which illustrate its dynamic aspects.

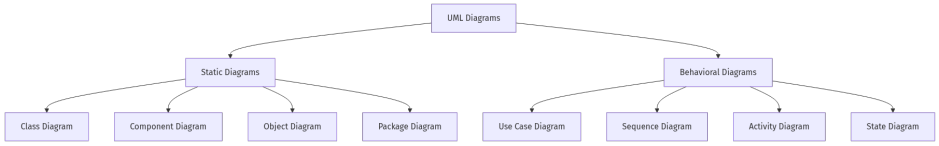


Figure 4.1: Hierarchy of UML Diagrams --- static (class, component, object, package) and behavioral (use case, sequence, activity, state).

The subsequent subsections elaborate on the specific diagram types harnessed in the design of the proposed application.

4.3.4 Diagrams Employed in the Implementation

Use Case Diagrams

Use case diagrams capture the interactions between actors and the system, delineating the functional scope from an end-user perspective. The present application identifies two principal human actors --- *Member* and *Coach* --- alongside a system-level actor responsible for administrative tasks.

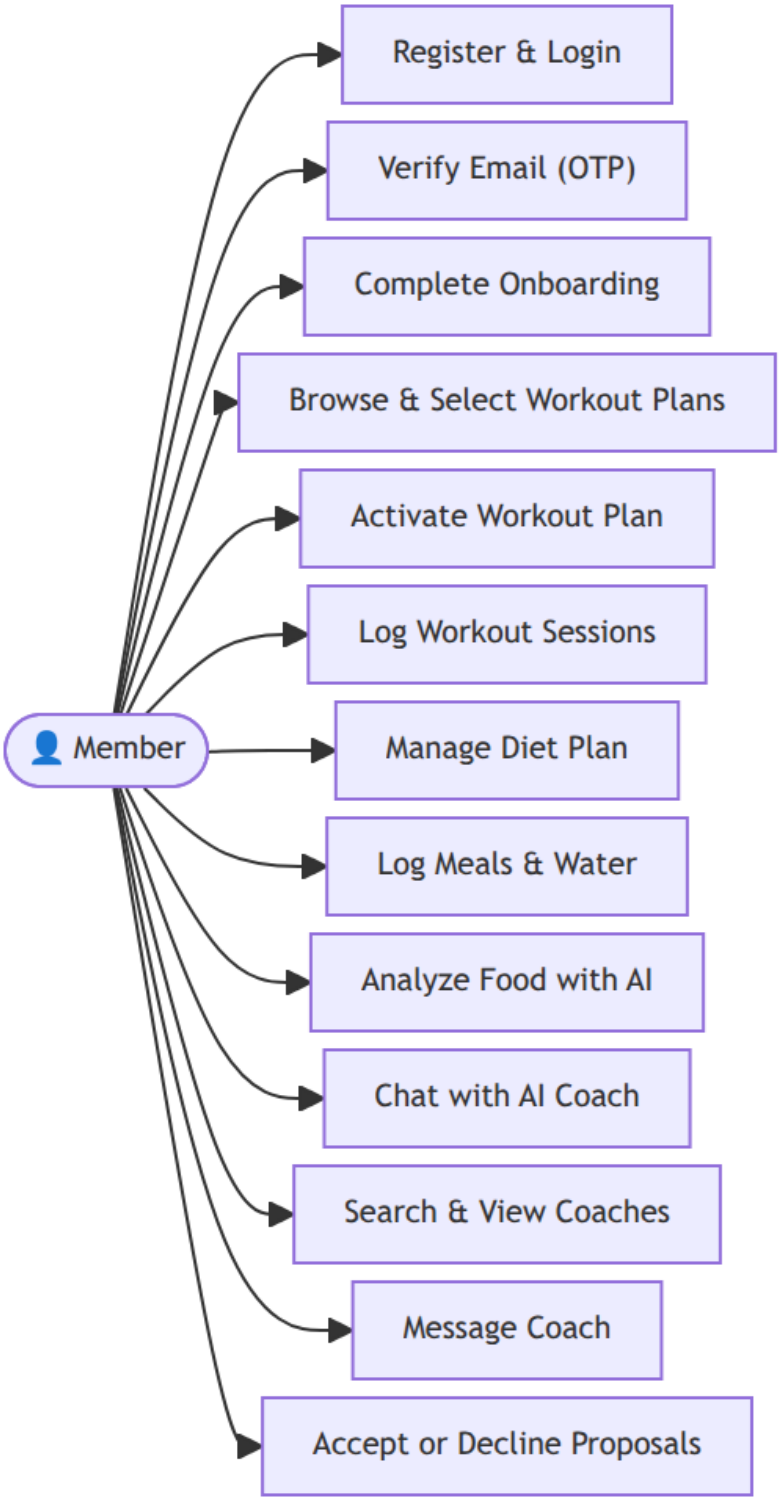


Figure 4.2: Use Case Diagram: Member --- registration, onboarding, workout management, diet tracking, AI coaching, and messaging.

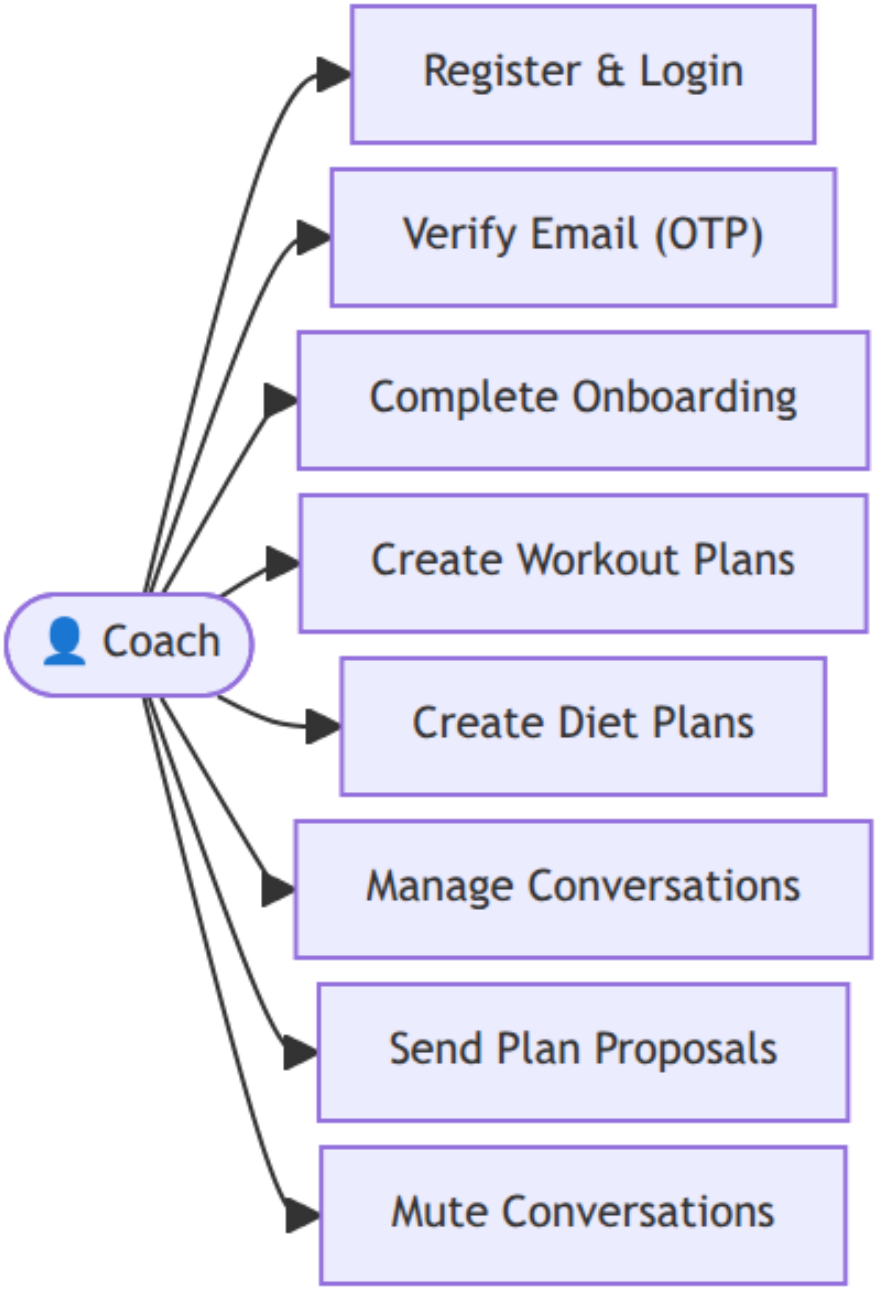


Figure 4.3: Use Case Diagram: Coach --- registration, plan creation, conversation management, and proposal sending.

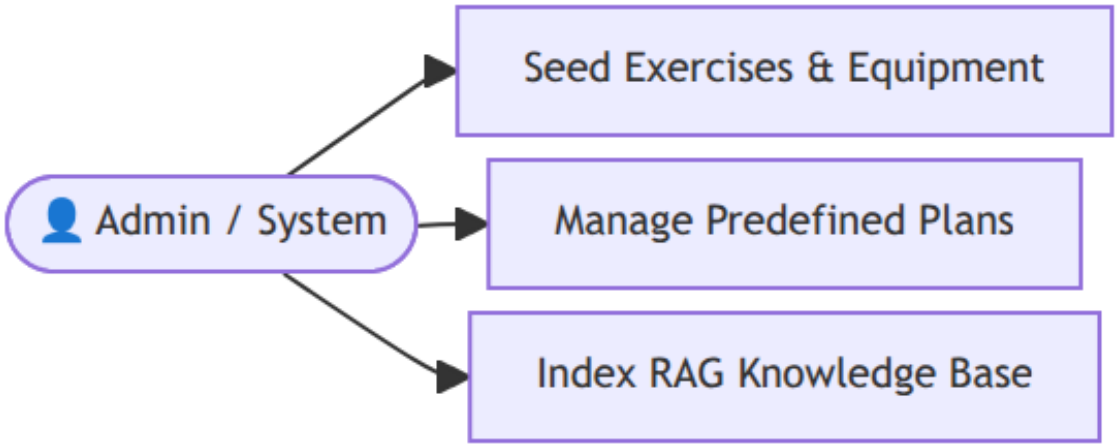


Figure 4.4: Use Case Diagram: Admin/System --- seed data, manage predefined plans, index RAG knowledge base.

Table 4.3: Roles and their use cases

Role	Use Cases
Member	Register/Login, Email Verification, Onboarding, Browse & Activate Plans, Log Workouts, Manage Diet, AI Food Analysis, AI Chat, Find Coaches, Direct Messaging, Accept/Decline Proposals
Coach	Register/Login, Onboarding, Create Workout/Diet Plans, Manage Conversations, Send Proposals, Mute Conversations
System/Admin	Seed data, Manage predefined plans, Index RAG embeddings

Sequence Diagrams

Sequence diagrams model the temporal exchange of messages among objects for a particular scenario. They are instrumental in validating that the designed components collaborate correctly to realize a use case.

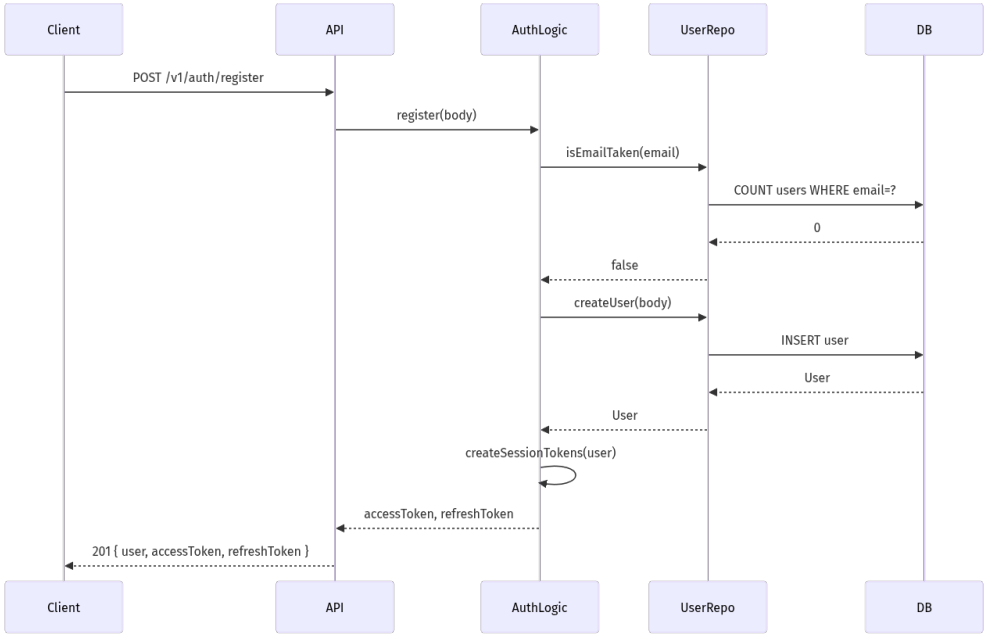


Figure 4.5: Sequence Diagram: User Registration --- client sends credentials, AuthLogic validates, creates user, returns tokens.

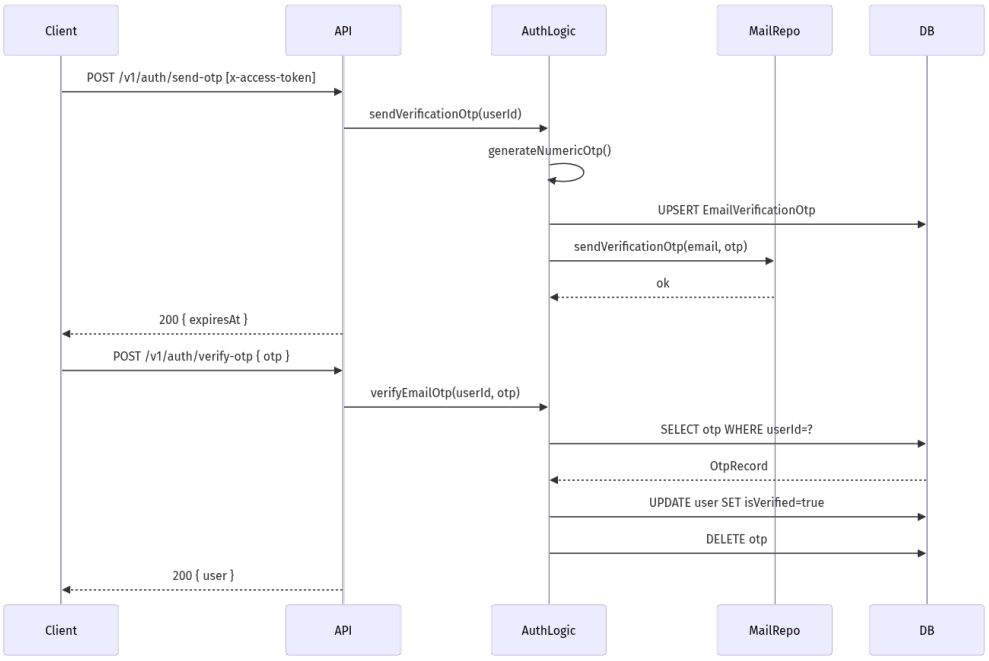


Figure 4.6: Sequence Diagram: Email OTP Verification --- client requests OTP, AuthLogic generates and sends via MailRepo, client verifies.

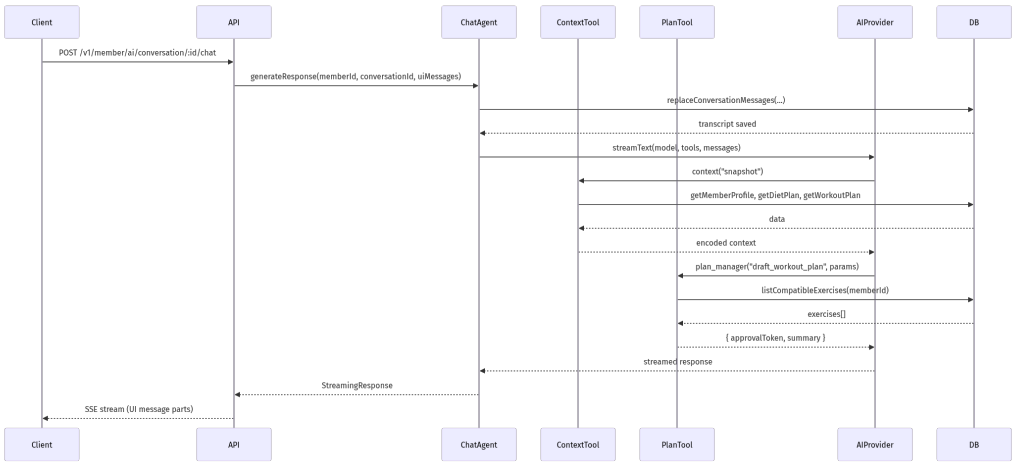


Figure 4.7: Sequence Diagram: AI Chat with Plan Generation --- message flows through ChatAgent, AIProvider calls context and plan tools, returns streamed response.

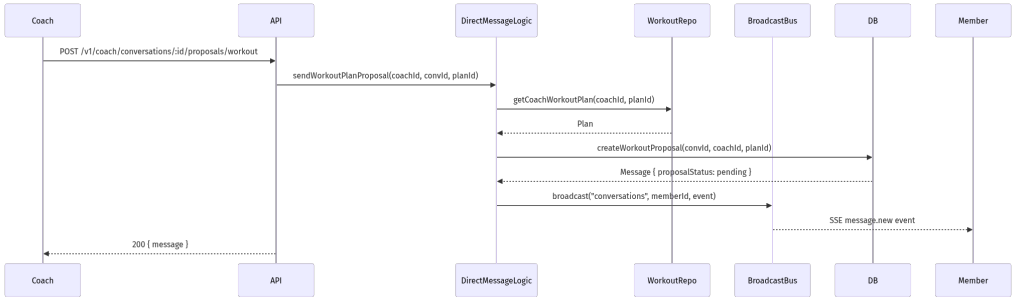


Figure 4.8: Sequence Diagram: Coach Sends Proposal --- coach sends workout proposal, DirectMessageLogic persists and broadcasts to member via SSE.

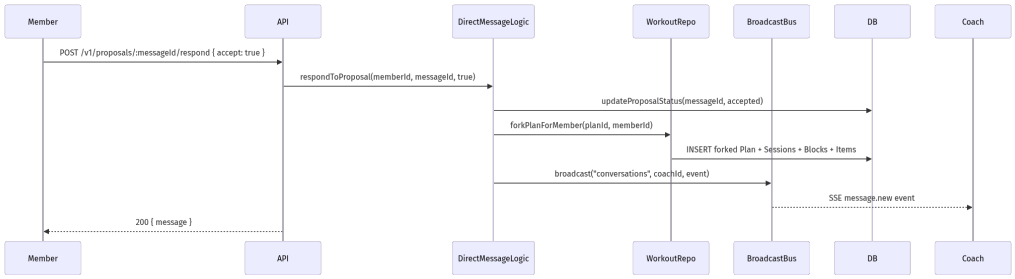


Figure 4.9: Sequence Diagram: Member Accepts Proposal --- member accepts, plan is forked for member, coach notified via SSE.

Class Diagram

Class diagrams define the static structure of the system by specifying the key domain entities, their attributes, operations, and the relationships that connect them. The following diagram provides a holistic view of the core domain model.

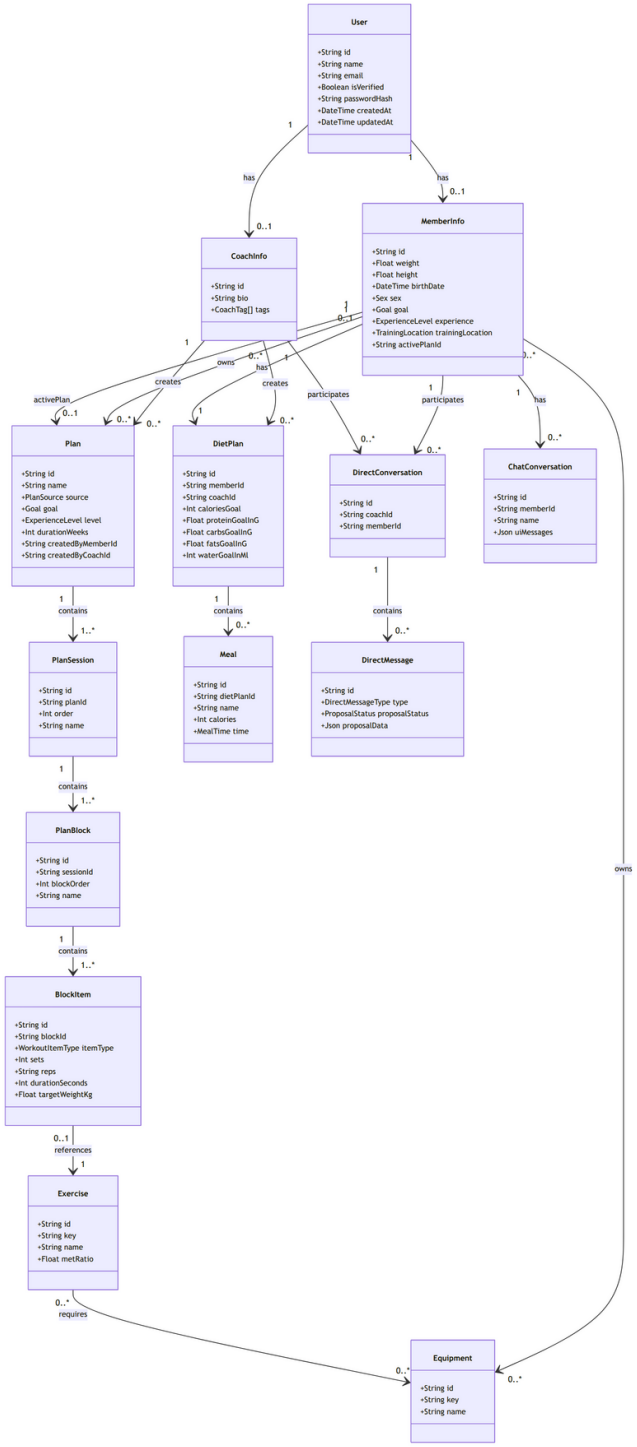


Figure 4.10: Class Diagram: Full Domain Model --- User, MemberInfo, CoachInfo, Plan, PlanSession, PlanBlock, BlockItem, Exercise, Equipment, DietPlan, Meal, DirectConversation, DirectMessage, ChatConversation.

Table 4.4: Class descriptions for the domain model

Class	Description
User	Core identity entity storing authentication credentials, verification status, and system timestamps.
MemberInfo	Extended profile for users who onboard as a Member, capturing anthropometric data, fitness goal, experience level, and training location.
CoachInfo	Extended profile for users who onboard as a Coach, encompassing biography, specialization tags, and contact links.
Plan	A structured workout plan, which may originate from the system, a coach, or a member's own design.
PlanSession	A single training day within a plan, holding an ordered sequence of blocks.
PlanBlock	A logically grouped set of exercises within a session, such as warm-up, main set, or cool-down.
BlockItem	A concrete exercise entry inside a block, specifying sets, repetitions, duration, and target intensity.
Exercise	A catalogue entry for an exercise, annotated with a MET ratio for energy expenditure estimation.
Equipment	A piece of gym apparatus; members declare available equipment; exercises state required equipment.
DietPlan	A nutritional blueprint prescribing daily calorie and macronutrient targets, coupled with a water consumption goal.
Meal	An individual food entry categorized by time slot (breakfast, lunch, etc.) within a diet plan.
Direct-Conversation	A private message channel linking one coach and one member.
Direct-Message	A single message in a direct conversation, capable of embedding workout/diet proposals with acceptance status.
Chat-Conversation	An AI-driven dialogue thread owned by a member, preserving the full message history in UI-oriented format.

Activity Diagrams

Activity diagrams model the procedural flow of tasks within the system, highlighting decision points and parallel branches. The following diagrams capture the principal user journeys.

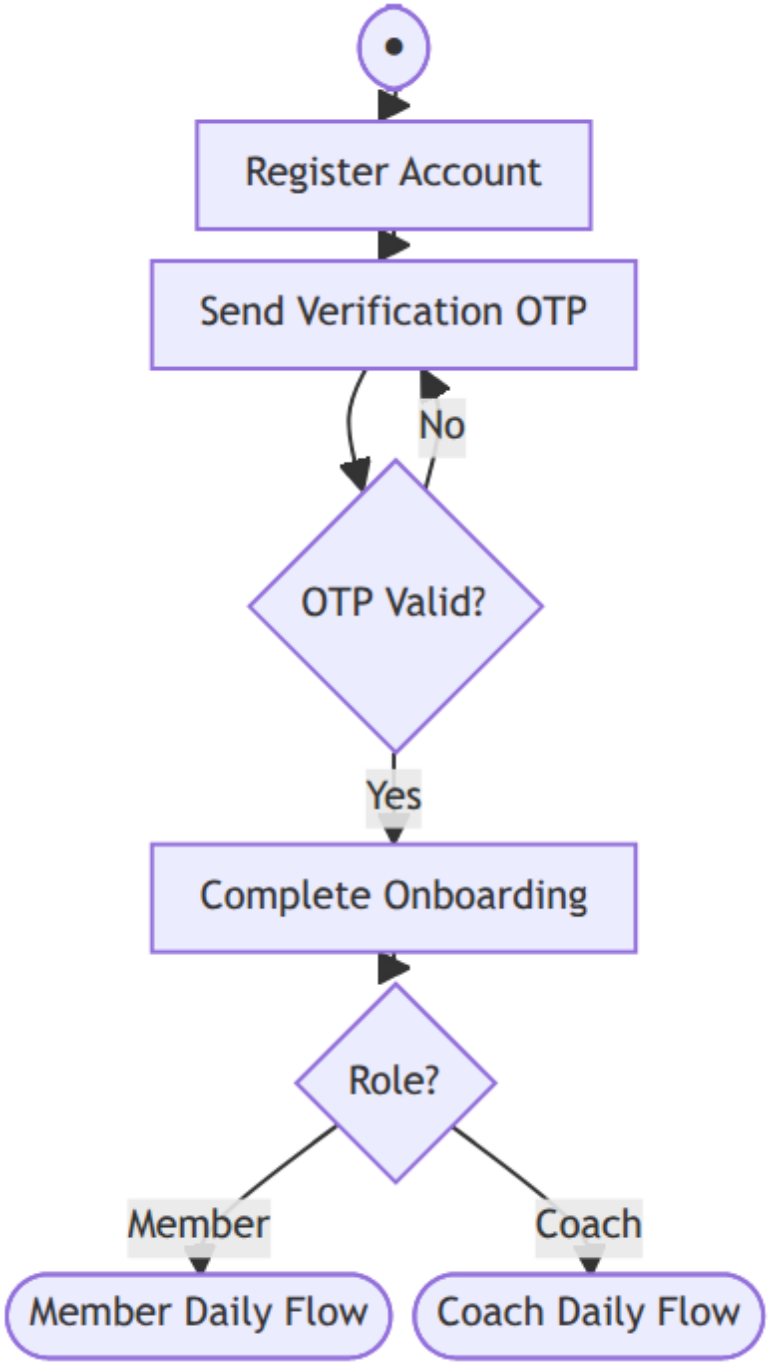


Figure 4.11: Activity Diagram: Registration & Onboarding --- register, verify OTP, complete onboarding, role-based routing to Member or Coach flow.

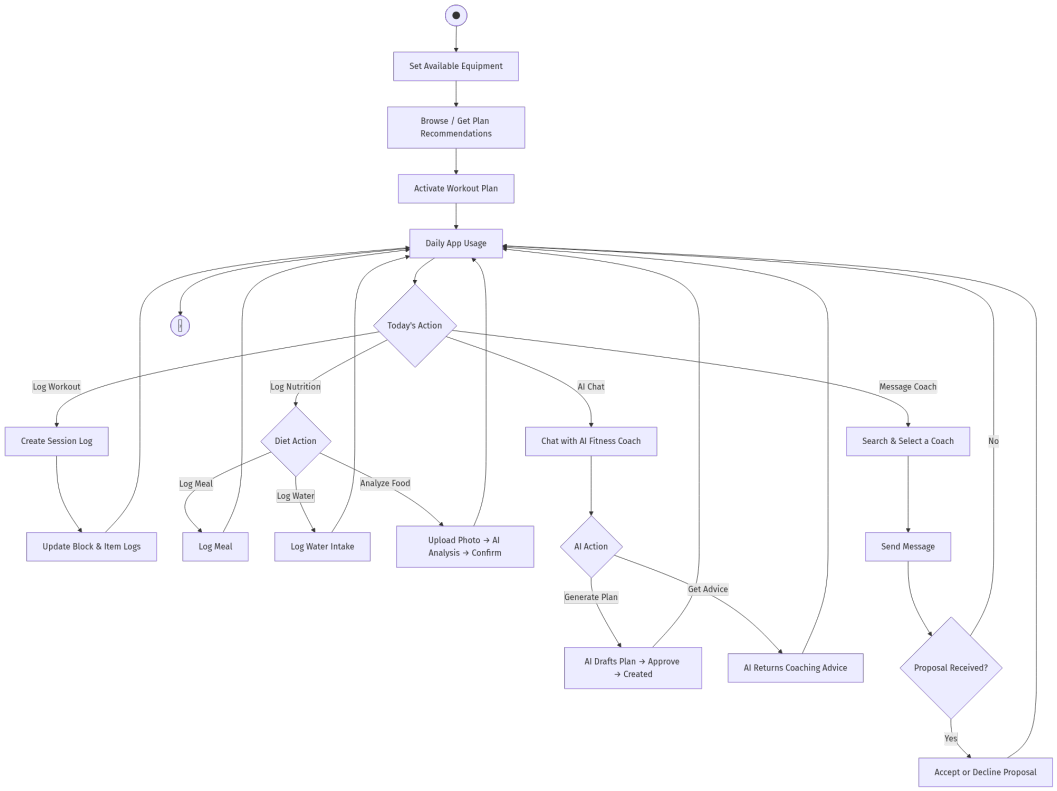


Figure 4.12: Activity Diagram: Member Daily Flow --- log workouts, log nutrition with AI food analysis, chat with AI coach, message coach, manage proposals.

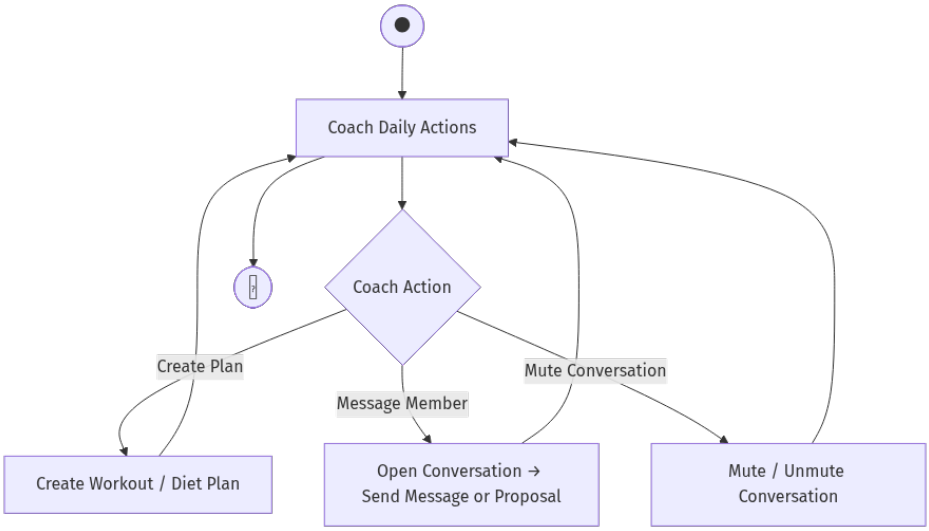


Figure 4.13: Activity Diagram: Coach Daily Flow --- create workout/diet plans, message members, send proposals, mute conversations.

4.4 Database Architecture

The persistence layer is underpinned by a **PostgreSQL** relational database, with the schema managed through **Prisma ORM**. The logical data model is organized into four interrelated

domains: identity and profiles, training plans, nutrition, and messaging.

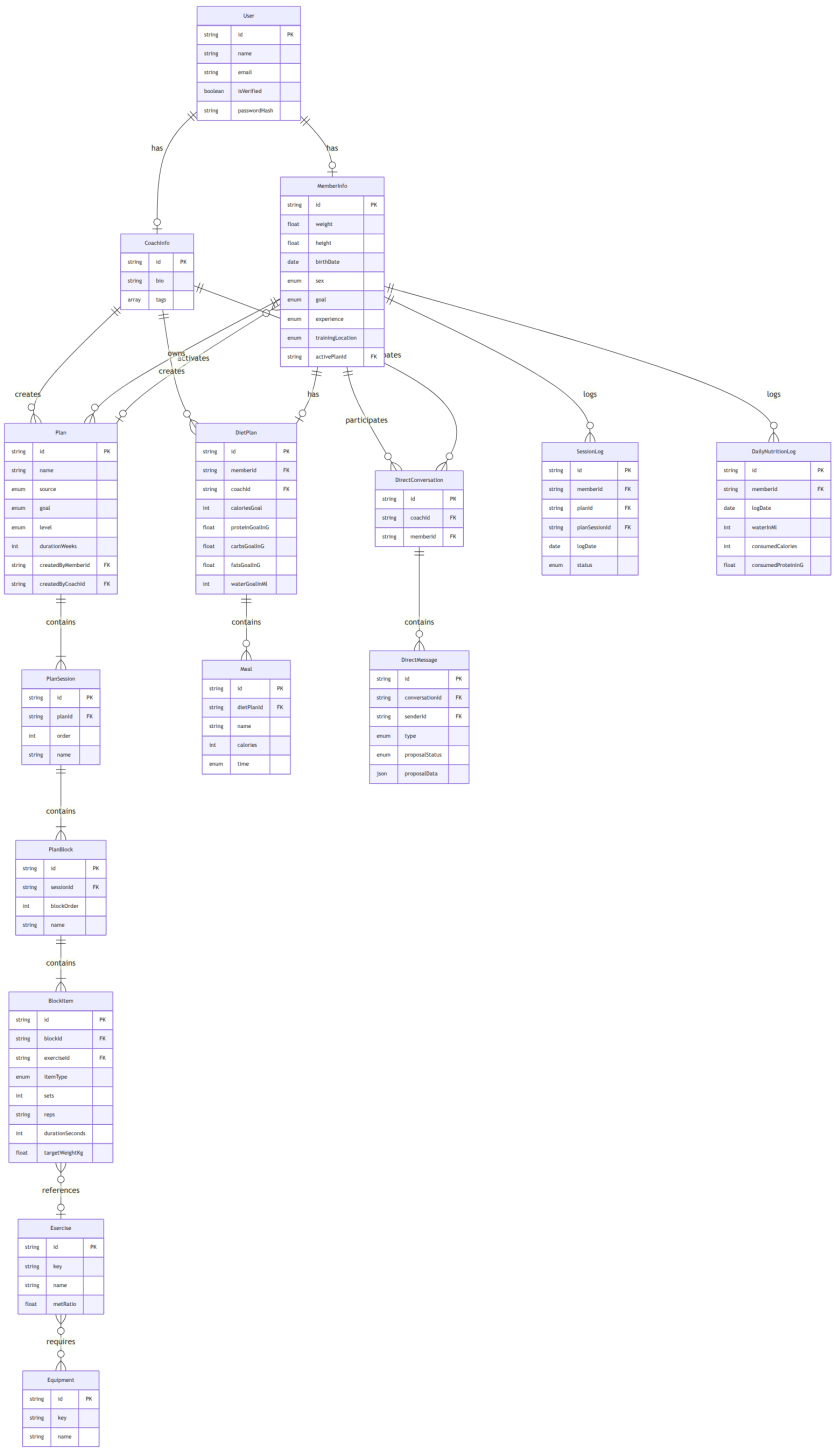


Figure 4.14: Entity-Relationship Diagram: Full Database Schema --- User, MemberInfo, CoachInfo, Plan, PlanSession, PlanBlock, BlockItem, Exercise, Equipment, DietPlan, Meal, DirectConversation, DirectMessage, SessionLog, DailyNutritionLog.

The schema enforces referential integrity through foreign key constraints and supports efficient retrieval of nested training hierarchies (plan → sessions → blocks → items). The

use of enumerations for attributes such as `goal`, `experience`, and `sex` ensures domain-level consistency.

4.5 AI Models and Services Used

The platform's intelligent behavior is realized through the integration of multiple artificial intelligence components that implement the agent model introduced in Chapter 1. Instead of generating generic responses, the system combines user profile data, stored workout and diet history, retrieved contextual information, and controlled service calls to support personalized coaching decisions. This also reflects the domain analysis of Chapter 2, where safe recommendations depend on goals, equipment, recovery, preferences, and limitations.

4.5.1 Large Language Model (Google Gemini)

The conversational coaching agent and the computer-vision-based meal logger rely on **Google Gemini** (model `gemini-2.5-flash`), accessed via the Vercel AI SDK.. The LLM is harnessed for three primary tasks:

- Providing real-time, context-aware fitness and nutrition guidance using a structured tool-calling approach.
- Analyzing food images and generating approximate nutritional breakdowns.
- Automatically generating descriptive titles for chat threads.

The AI chat agent orchestrates a set of purpose-built tools that grant it secure, read-limited access to the member's profile and plan repository.

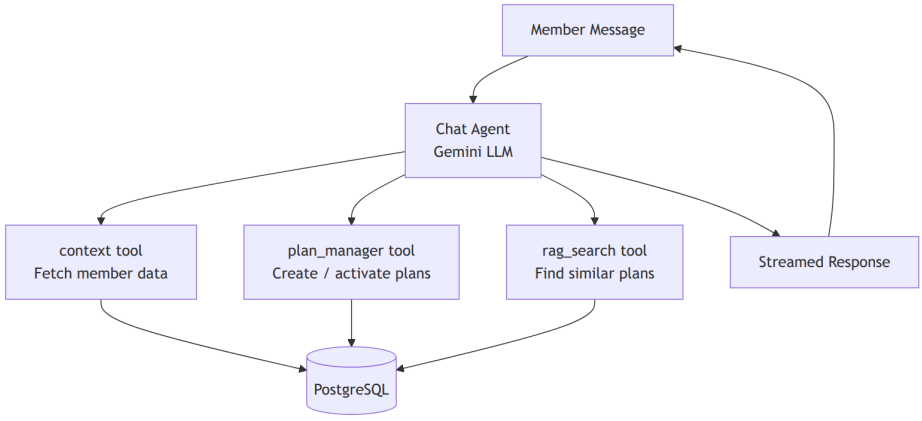


Figure 4.15: AI Chat Agent Tool Flow --- member message processed by Gemini LLM, which calls context, plan_manager, and rag_search tools, all backed by PostgreSQL.

4.5.2 Embedding Model (Ollama / nomic-embed-text)

To implement the retrieval-augmented generation system, the project employs the **nomic-embed-text** model served through an **Ollama** instance. The model transforms workout plan descriptions into dense 768-dimensional embedding vectors, which are persisted in the `WorkoutRagEmbedding` table using the `pgvector` extension. Cosine similarity queries enable the discovery of semantically similar plans, forming the knowledge backbone for AI-generated recommendations.



Figure 4.16: RAG Embedding & Search Flow --- member query embedded via nomic-embed-text, vector similarity search via pgvector, top-K results summarized by Gemini LLM.

4.5.3 Plan Manager Tool --- Approval Flow

All AI-initiated modifications to a member's plan state are governed by a cryptographic approval protocol to prevent unauthorized or accidental changes. When the LLM drafts a workout plan, the system produces a tamper-evident approval token. The plan is only persisted after the member explicitly submits that token back through the chat interface.

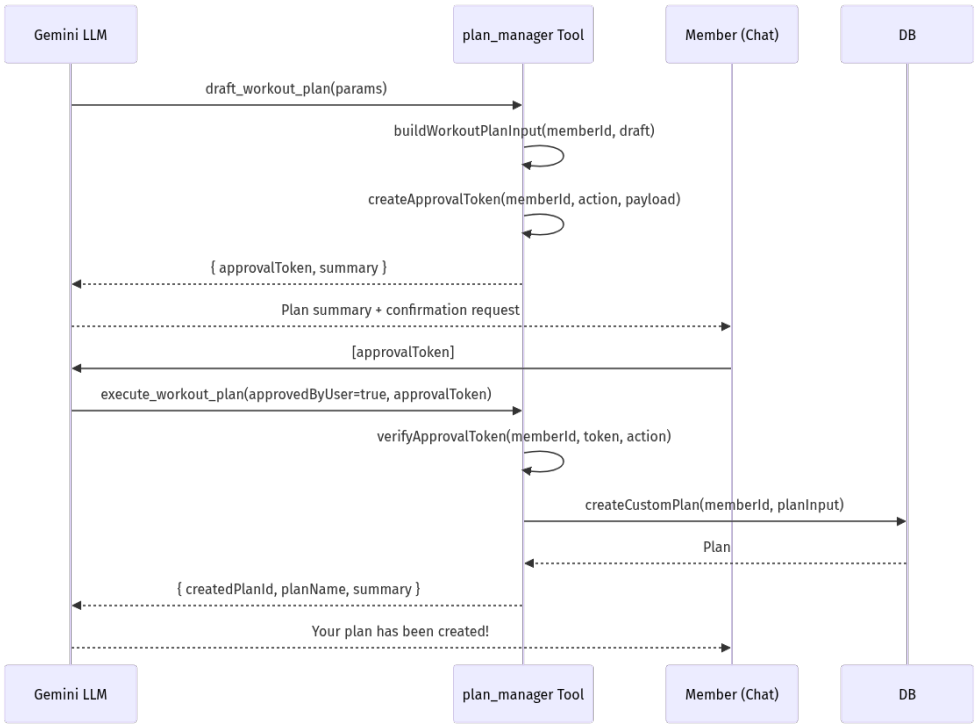


Figure 4.17: Plan Manager Approval Flow --- LLM drafts plan, creates approval token, member confirms via chat, token verified, plan persisted.

4.6 Application Architecture

The solution is built as a **Bun**-powered monorepo orchestrated by **TurboRepo**, with the principal API server implemented in **ElysiaJS**. This monorepo structure promotes code sharing, consistent tooling, and parallelized task execution.

4.6.1 Layered Architecture

The backend adheres to a strict separation of concerns, enforced through the **Sandly** dependency injection container. Each layer has a clearly defined responsibility:

- **Routes** map HTTP requests to business logic operations.
- **Logic** encapsulates domain rules and orchestrates repository calls.
- **Repositories** provide a clean abstraction over data access, hiding the specifics of Prisma or external services.
- **Infrastructure** comprises concrete implementations of database drivers, AI clients, object storage, and messaging.
- **AI Agents** use the Logic layer and custom tool repositories to interact with the LLM in a controlled manner.

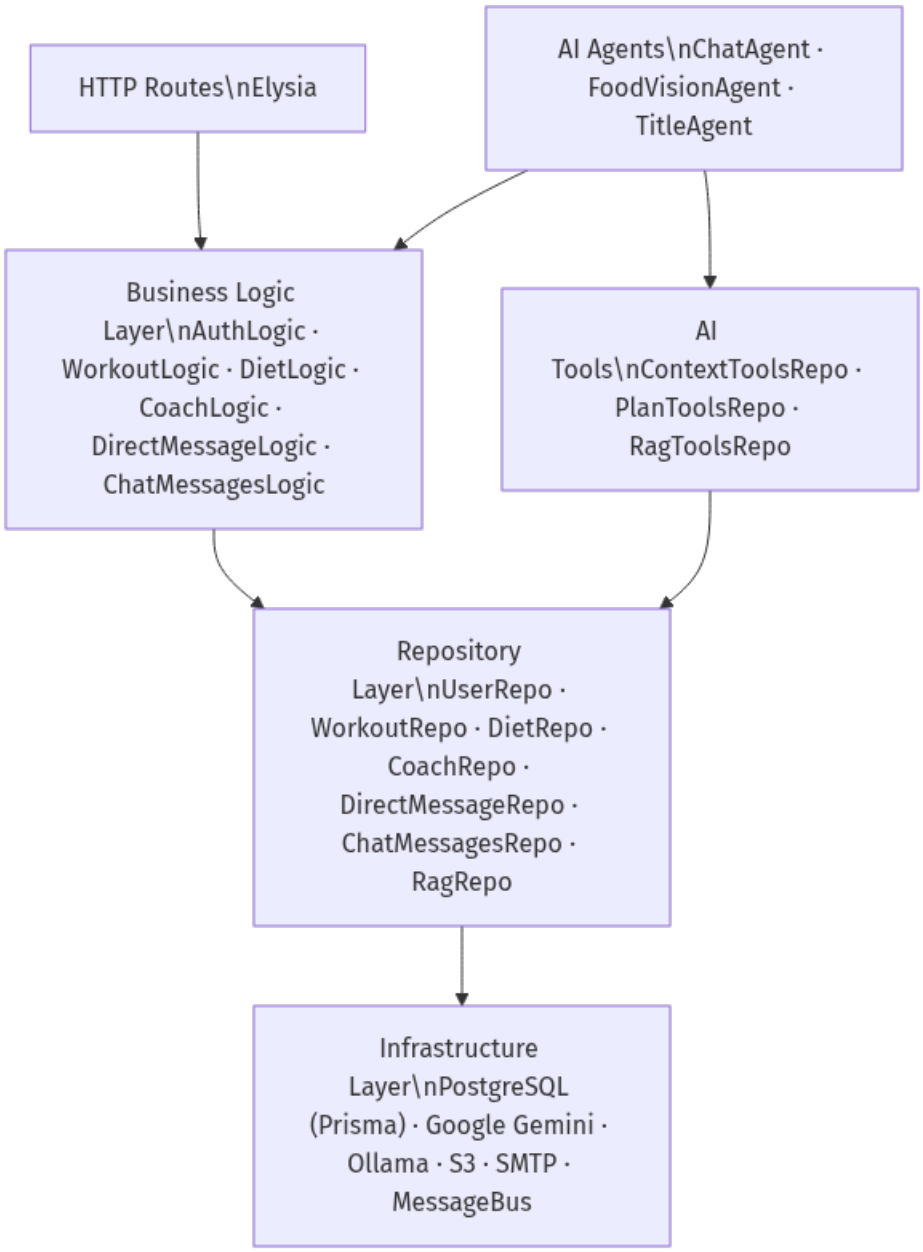


Figure 4.18: Layered Backend Architecture --- Routes → Logic → Repositories → Infrastructure, with AI Agents and Tools interacting with Logic and Repos.

4.6.2 Component Diagram

The high-level deployment view aggregates the constituent components and their interconnections.

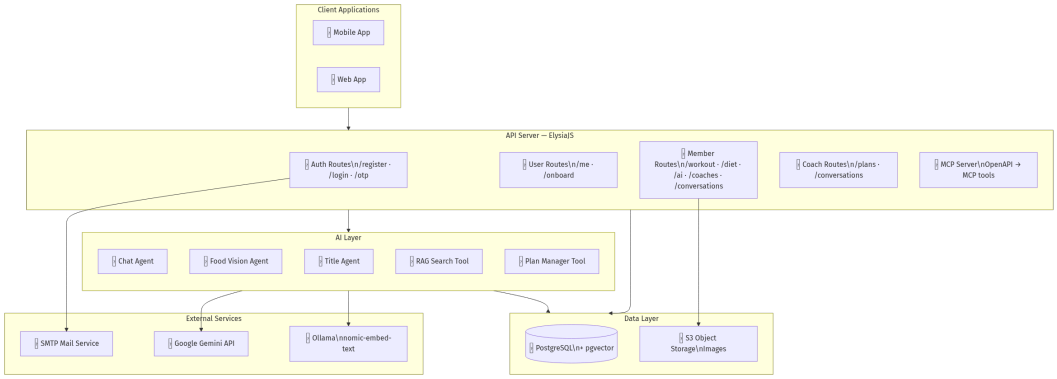


Figure 4.19: Full Component Diagram --- Mobile/Web clients connect to ElysiaJS API server, which orchestrates AI Layer (Chat, FoodVision, Title agents, RAG Search, Plan Manager), Data Layer (PostgreSQL + pgvector, S3), and External Services (Gemini API, Ollama, SMTP).

The architecture is designed for extensibility: new AI providers can be swapped by implementing the appropriate repository interfaces, and additional microservices can be introduced without disrupting the core API.

4.7 Conclusion

Integrating requirements analysis, UML diagrams, database modeling, AI services, and layered architecture provides a robust design framework for the proposed fitness coaching application. This design connects intelligent-agent mechanisms with practical fitness tracking needs and addresses the fragmentation observed in similar applications. Incorporating these design artifacts into development prepares the transition from specification to implementation, enabling the next chapter to present the mobile client, backend API, database layer, AI services, and deployment environment as a direct realization of the proposed system blueprint.

Chapter 5

Implementation of the Proposed Application

5.1 Introduction

Integrating a React Native mobile client with a Bun/Elysia backend, Prisma-managed database, and AI services transforms the proposed design into a working fitness tracking application. The mobile interface supports daily interaction and progress monitoring, whereas the backend manages authentication, business logic, data persistence, and AI-powered guidance. This chapter examines the development environment, technologies, architecture, user interfaces, API integration, and deployment workflow, illustrating how the design of Chapter 4 implements the AI-agent foundation, fitness-management requirements, and integration needs established in the previous chapters.

5.2 Development Environment

5.2.1 Hardware

The development was carried out on both the HP Pavilion-17 laptop and the Desktop PC, which provided sufficient computational resources for running the development environment, emulators, local backend services, database tools, and concurrent applications. Testing was performed on the physical mobile device to ensure compatibility with Android usage, while the production backend was validated on two Fly.io machines configured with 512MB RAM and one shared CPU each.

5.2.2 Used Technologies and Tools

The tools employed in this project span three categories: those dedicated to the backend server, those specific to the mobile frontend, and those shared across both subsystems.

Table 5.1: Hardware used during development

Device	Specifications
HP Pavilion-17 (Laptop)	RAM: 16GB DDR4 / CPU: Intel Core i5--10th Gen / GPU: NVIDIA GeForce GTX 1660Ti / Storage: 512GB SSD / OS: Windows 11
Samsung Galaxy S20 Ultra (Phone)	RAM: 12GB / Internal Memory: 256GB / Screen: 6.9" AMOLED / OS: Android 12
Fly.io Machines (2x Production VMs)	RAM: 512MB per machine / CPU: 1 shared CPU per machine / Region: Paris (cdg) / Purpose: backend deployment and redundancy
Desktop PC	RAM: 32GB DDR4 / CPU: AMD Ryzen 5 8500G / GPU: AMD Radeon 740M / Storage: 1TB SSD

Backend Tools

1. **Bun and Elysia --- Backend Runtime and Web Framework**. Bun provides a high-performance JavaScript runtime with native TypeScript support, while Elysia compiles route handlers AOT for optimal throughput and automatically propagates types to the OpenAPI specification.

2. **Prisma ORM with PostgreSQL**. Prisma provides type-safe database access to PostgreSQL via a declarative schema and auto-generated client, with the `@abcx3/prisma-bun-adapter` bridging Prisma to Bun's runtime and a `vector(768)` column enabling pgvector-powered semantic search.

3. **Sandly --- Dependency Injection Container**. Sandly composes layered dependencies (Infra, Repos, Logic, AI) into request-scoped containers, enforcing separation of concerns and enabling isolated unit testing through layer mocking.

4. **better-result --- Railway-Oriented Error Handling**. better-result models every operation as a typed `Result<T, E>` discriminated union with `TaggedError` subclasses, making all failure modes explicit in the type system and eliminating unchecked exceptions.

5. **Vercel AI SDK with Google Gemini and Ollama**. The Vercel AI SDK unifies Google Gemini 3.1 Flash Lite for conversational AI (coaching, plan generation, food analysis) and Ollama with `nomic-embed-text` for local vector embeddings powering RAG-based workout recommendations.

6. **jose --- JWT Authentication**. jose provides HS256 JWT signing for short-lived access tokens and SHA-256 hashing for refresh tokens, securing the dual-token authentication scheme.

7. **Sharp and Bun S3Client --- Media Processing and Storage**. Sharp resizes and compresses user-uploaded images, while Bun's built-in S3Client stores them in S3-compatible object storage, decoupling media from the application server.

8. **@toon-format/toon --- Structured LLM Output**. TOON serializes AI responses in a token-efficient structured format, reducing latency and API costs for LLM-powered coaching, generation, and analysis features.

9. **evlog --- Structured Logging**. evlog produces typed, structured log entries per request via its Elysia plugin, enabling effective production debugging and log aggregation.

10. **Docker --- Containerization and Deployment**. The API is packaged as a multi-stage Docker image --- pruned via Turborepo, compiled into a standalone binary via `bun build --compile`, and copied into a minimal Alpine runtime --- enabling reproducible deployments across environments.

Frontend (Mobile) Tools

11. **React Native and Expo**. React Native enables cross-platform mobile development from a single TypeScript codebase, while Expo SDK 54 with expo-router v6 provides file-based routing, hot reloading, and a managed native runtime with companion libraries for status bars, splash screens, fonts, linking, icons, and device constants.

12. **React Navigation Stack**. react-native-screens, safe-area-context, and gesture-handler provide native screen containers, safe area handling, and gesture-driven navigation, with keyboard-controller managing input visibility.

13. **TanStack Query (React Query)**. TanStack Query manages server state with automatic caching, background refetching, and synchronization for all API calls across workouts, diet, progress, and coaching.

14. **Zustand**. Zustand provides minimal, performant client-side stores for authentication tokens, UI preferences, and local data caches with selective subscriptions.

15. **NativeWind**. NativeWind brings Tailwind CSS utility classes to React Native, enabling responsive design, dark mode, and a consistent design system with minimal custom styling.

Shared Tools

16. **TypeScript**. TypeScript adds static type checking across the full stack, reducing runtime errors and improving IDE support in both the mobile client and backend server.

17. **Zod --- Runtime Schema Validation**. Zod validates all data structures project-wide --- HTTP payloads, environment variables, AI tool inputs, and user forms --- with auto-conversion to JSON Schema for OpenAPI generation.

18. **OpenAPI Specification and MCP Protocol**. The `@elysiajs/openapi` plugin introspects every registered route and produces an auto-generated OpenAPI 3.0.3 spec, served as an interactive Scalar UI at `/docs` and surfaced as MCP tools at `/mcp` for AI-assisted development workflows.

5.3 The Application Architecture

Our fitness application is built on established software engineering principles and design patterns that ensure maintainability, scalability, and clean code organization. Following the layered design introduced in Chapter 4, the system is partitioned into two principal subsystems: the mobile client, which handles user interaction, daily tracking, and local state, and the backend server, which encapsulates business logic, authentication, data persistence, and AI capabilities. They communicate through a well-defined REST API contract described by an auto-generated OpenAPI specification. This organization is especially important because, as established in Chapter 1, reliable AI agents require well-managed user and fitness context rather than isolated prompts.

5.3.1 Backend Server Architecture

The backend follows a strict layered architecture with unidirectional dependencies: each layer depends only on the layer directly beneath it. Dependencies are wired at runtime through a dependency injection container, ensuring that each layer can be tested in isolation.

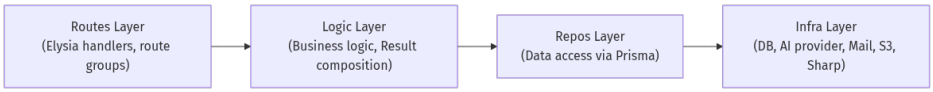


Figure 5.1: Backend layered architecture: Routes → Logic → Repos → Infra.

Routes Layer

The **Routes Layer** defines the HTTP surface of the API.. Each route file declares one or more Elysia handlers, specifying the HTTP method, path, request validation schema (derived from Zod DTOs), and the response schema. Route handlers are thin: they extract parameters from the request, delegate to the corresponding logic function, and map the returned `Result` to an appropriate HTTP response --- a 200 for success, a 4xx for client errors, or a 5xx for internal failures. Routes are organized into a `v1` namespace with domain sub-groups (`/auth`, `/user`, `/member/workout`, `/member/diet`, `/member/ai`, `/coach/conversations`, etc.).

Logic Layer

The **Logic Layer** contains all business rules and orchestrates cross-cutting operations. Every logic function returns a `Result<T, E>` --- a discriminated union that is either `Ok<T>` or `Err<E>`. Domain errors are modelled as `TaggedError` subclasses, creating an explicit error taxonomy. For example, when a member attempts to activate a workout plan that is already active, the logic returns an `Err<PlanAlreadyActiveError>` rather than throwing an exception. This railway-oriented style forces calling code --- typically route handlers --- to handle every error variant explicitly.

Logic functions depend on repository interfaces (injected via Sandly) rather than concrete implementations, enabling them to be unit-tested with in-memory or mock repositories.

Repos Layer

The **Repos Layer** abstracts database access behind interface boundaries. Each repository encapsulates the Prisma queries for its domain --- `WorkoutRepo`, `DietRepo`, `UserRepo`, `ConversationRepo`, etc. Repositories accept and return domain types, not raw Prisma models, insulating the upper layers from schema changes.

Infra Layer

The **Infra Layer** provides concrete implementations for external concerns: the PostgreSQL connection via Prisma, the AI provider (Google Gemini for chat, Ollama for embeddings), an SMTP mailer for OTP delivery, an S3-compatible object store for user-uploaded images, and the Sharp library for image processing and compression.

Dependency Injection Composition

The four layers are composed at startup using Sandly:

```
AppLayer = AILayer.merge(LogicLayer)
 .provide(ReposLayer)
 .provide(InfraLayer)
```

Each HTTP request receives a `ScopedContainer` derived from this composition, guaranteeing that database connections, AI provider sessions, and other transient resources are allocated per-request and cleaned up after the response is sent.

5.3.2 Mobile Client Architecture

Model-View-ViewModel (MVVM)

While React Native naturally follows a component-based architecture, we have adopted principles from the Model-View-ViewModel pattern. The separation of concerns is maintained through:

- **Models:** TypeScript interfaces and API response types define our data structures.
- **Views:** React Native components handle presentation logic.
- **ViewModels:** Custom hooks (using TanStack Query) manage data fetching and state derivation.

This separation makes testing easier and keeps business logic decoupled from UI rendering.

Layered Organization

The mobile client is organized into distinct layers, each with a well-defined responsibility:

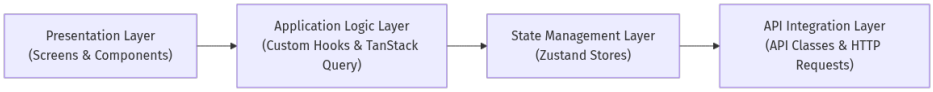


Figure 5.2: Mobile client layered architecture: Presentation → Application Logic → State Management → API Integration.

The **Presentation Layer** comprises full-screen views and reusable UI components. Screens are organized around feature domains --- authentication, workout planning, diet tracking, progress analytics, AI coaching, and messaging. Components are further subdivided into primitives (buttons, inputs, cards) and domain-specific composites (workout cards, macro circles, streak indicators, charts).

The **Application Logic Layer** consists of custom React hooks that orchestrate data flows. TanStack Query hooks manage server state: they issue requests through the API layer, cache responses, invalidate stale data, and refetch in the background. Derived state --- such as computed macros or progress percentages --- is produced within these hooks rather than in components.

The **State Management Layer** employs Zustand stores for client-side state that does not originate from the server: authentication tokens, UI preferences, onboarding completion status, and temporary form data. Zustand's minimal API avoids the boilerplate typical of larger state management solutions while remaining performant through selective subscriptions.

The **API Integration Layer** encapsulates all HTTP communication in domain-specific classes --- `AuthAPI`, `WorkoutAPI`, `DietAPI`, `CoachingAPI`, `MessagingAPI`, and `AIAPI`. Each class exposes stateless static methods that accept typed payloads and return typed responses, providing a clean boundary between the network transport and the rest of the application.

Component Structure

The application is organized into logical sections:

- **Screens:** Full-screen views corresponding to each application route.
- **Components:** Reusable UI components organized by type (primitives, domain-specific cards, charts).
- **Hooks:** Custom React hooks for data fetching and state management.
- **Stores:** Zustand stores for global application state (auth, user, workout, progress, UI).
- **Utils:** API classes, formatters, and utility functions.
- **Constants:** Theme definitions and constant values.

5.3.3 API Contract and Integration Layer

The mobile client and the backend server are coupled through a strictly typed API contract that is generated automatically from the server's own route definitions.

OpenAPI Specification

Elysia's `@elysiajs/openapi` plugin introspects every registered route, extracts the Zod-derived JSON Schema for request bodies and responses, and produces a complete **OpenAPI 3.0.3** document. This document is served as raw JSON at `/openapi.json` and rendered as an interactive **Scalar** UI at `/docs`. Because the specification is a direct compilation of the source code, it never drifts out of sync with the implementation --- a property that hand-written documentation cannot guarantee.

Model Context Protocol (MCP) Endpoint

The server additionally exposes a **Model Context Protocol** endpoint at `/mcp` using `elysia-mcp` and `mcp-from-openapi`. Every REST endpoint is automatically surfaced as an MCP tool, enabling AI-enabled development environments --- including Visual Studio Code and the OpenCode agent --- to invoke backend operations

as native tool calls. The development environment's MCP configuration points to `https://finals-project.fly.dev/mcp`, establishing a direct bridge between the AI assistant and the production API.

Mobile Consumption

On the mobile side, the `API Integration Layer` issues standard HTTP requests through the `expo/fetch` API.. Each API class maps directly to a backend route group:

Table 5.2: Mobile API classes and their corresponding backend route prefixes

Mobile API Class	Backend Route Prefix	Purpose
`AuthAPI`	`/v1/auth`	Registration, login, token refresh, OTP verification
`WorkoutAPI`	`/v1/member/workout`	Exercise catalogue, plan CRUD, session logging, progress
`DietAPI`	`/v1/member/diet`	Plan management, meal logging, water tracking, food analysis
`CoachingAPI`	`/v1/member/coaches`	Coach directory, coach details, conversation creation
`MessagingAPI`	`/v1/member/conversations`	Conversation listing, message sending, real-time SSE streaming
`AIAPI`	`/v1/member/ai`	AI conversation management, chat streaming

Standard request-response interactions use the `expo/fetch` API with JSON serialization, while the AI coach feature employs **Server-Sent Events (SSE)** via the `react-native-sse` library, enabling the mobile client to receive streaming token responses in real time.

5.3.4 Development Workflow

The development workflow follows a unidirectional pipeline from business logic definition to mobile consumption, ensuring end-to-end type safety at each stage.

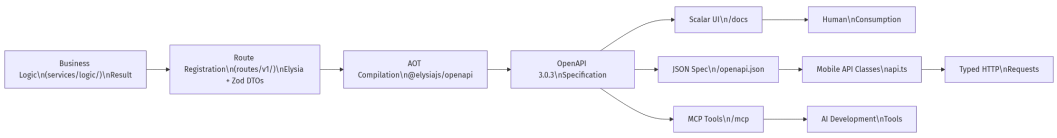


Figure 5.3: Development pipeline from business logic to mobile consumption --- Elysia route definitions generate OpenAPI spec, Scalar UI, MCP tools, and typed mobile API classes.

The numbered stages below elaborate on each step in the pipeline:

1. **Business Logic Definition.** Domain operations are implemented as pure functions in the `services/logic/` layer. Each function accepts typed inputs, performs its operation using injected repository dependencies, and returns a `Result<T, E>` that exhaustively encodes both success and error outcomes.

2. **Route Registration.** A corresponding Elysia route handler is defined in `routes/v1/`. The handler declares its Zod DTO schema for request validation, invokes the logic function, and maps the returned `Result` to an HTTP response. Elysia's AOT compilation transforms the handler definition into an optimized request processor at build time.

3. **Automatic Documentation Generation.** During server startup, `@elysiajs/openapi` traverses all registered routes, collects the Zod-derived JSON Schema from each handler, and assembles a complete OpenAPI 3.0.3 document. No separate documentation step is required --- the specification is always a faithful reflection of the running code.

4. **API Surface Distribution.** The generated OpenAPI schema is served openly: as a human-readable Scalar UI at `/docs`, as a machine-readable JSON file at `/openapi.json`, and as MCP tools at `/mcp`. The MCP endpoint is consumed by AI development tools, enabling agentic workflows where an AI assistant can directly query or mutate backend state during development.

5. **Mobile Integration.** Mobile developers reference the OpenAPI specification to ensure that API class method signatures match the backend contract. The typed interfaces in `api.ts` mirror the server's Zod DTOs, providing compile-time validation that the client sends correctly shaped payloads and correctly interprets responses.

This pipeline imposes a single source of truth --- the Elysia route definition --- from which the OpenAPI spec, MCP tools, and mobile type definitions are derived, eliminating the coordination overhead typical of separate frontend and backend development.

5.3.5 Production Deployment and Observability

The backend server is deployed on **Fly.io**, a platform-as-a-service provider that runs containerized applications across global edge regions.

Deployment Pipeline

The server is packaged as a multi-stage Docker image. The first stage uses Turborepo to prune the monorepo to only the API workspace. The second stage installs production dependencies and compiles the TypeScript source into a standalone binary via `bun build --compile`.

The final stage copies only this binary and the Prisma migration artefacts into a minimal Alpine image. The resulting container is deployed to the `cdg` (Paris) region with 512 MB of RAM and a single shared CPU.

Health Monitoring

Fly.io's HTTP load balancer periodically sends GET requests to the `/health` endpoint. The health handler returns a 200 status code only when the database connection, AI provider, and S3 storage are all reachable, ensuring that unhealthy instances are automatically rotated out of the load balancer pool.

Structured Logging with evlog

All server-side log output is produced through the **evlog** structured logging library. Each log entry carries structured metadata --- service name, environment, request identifier, and severity level --- rather than unstructured text. This structured format is essential for effective log aggregation and querying in production.

Production Observability with Grafana

Fly.io provides a built-in **Grafana** dashboard for each deployed application. All structured log output from the server instance is forwarded to this dashboard, where it can be filtered, searched, and visualized alongside CPU, memory, and network metrics for the running machine. The Grafana dashboard gives operators a unified view of application health, error rates, request latencies, and resource utilization, enabling rapid diagnosis of production issues without requiring access to individual server instances.

5.4 Application Interfaces

5.4.1 API Documentation Interface

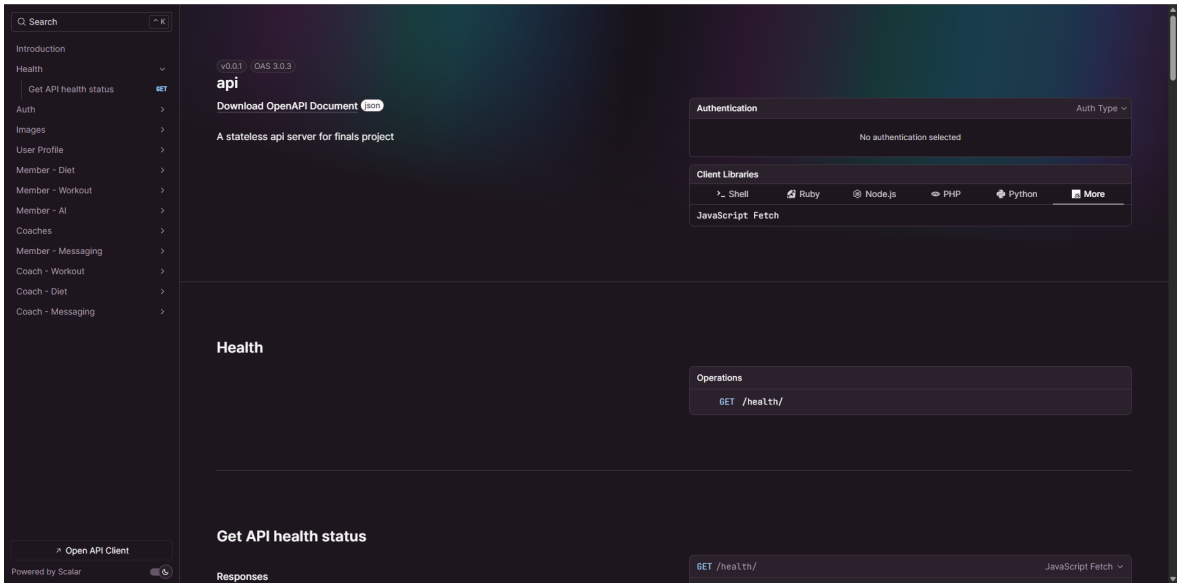


Figure 5.4: OpenAPI Documentation --- interactive Scalar UI interface at the `/docs` endpoint.

The backend server exposes an interactive API documentation interface at the `/docs` endpoint, rendered by the **Scalar** UI library. This interface is generated automatically from the OpenAPI 3.0.3 specification produced by `@elysiajs/openapi`. Developers and testers can inspect every endpoint's request parameters, request body schema (derived directly from Zod DTOs), response codes, and authentication requirements. The interface also supports live request execution against the production server, enabling rapid experimentation without external tools such as Postman or cURL.

5.4.2 Fly.io Deployment Dashboard

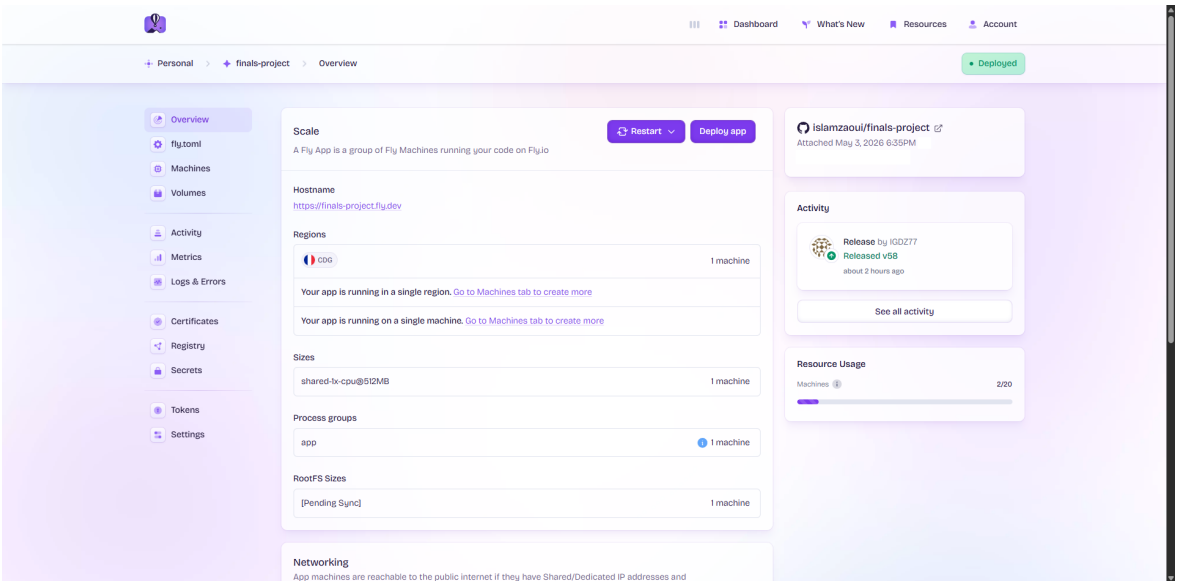


Figure 5.5: Fly.io App Dashboard --- overview of deployed application operational status, VM instances, logs, and environment configuration.

The **Fly.io** dashboard provides a high-level overview of the deployed application's operational status. From this interface, operators can monitor the running VM instances, view live and historical logs via `fly logs`, inspect environment variables and secrets, scale the number of allocated VMs, and trigger manual deployments. The dashboard lists the configured health check endpoints, custom domains, and SSL certificate status, consolidating all infrastructure management tasks into a single web interface.

5.4.3 Production Monitoring Dashboard

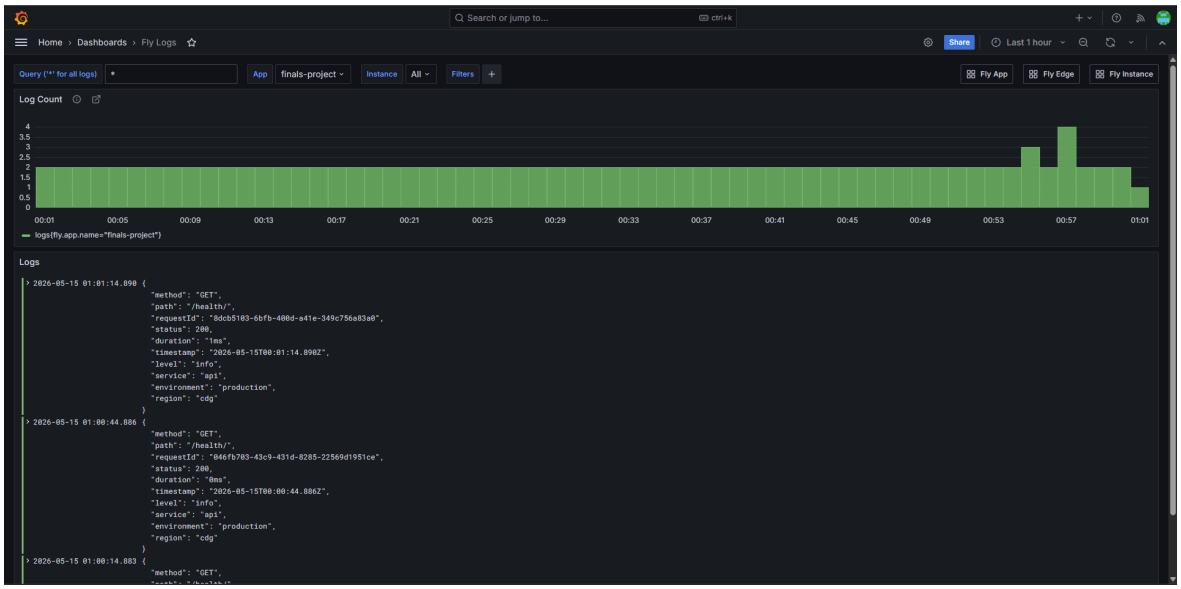


Figure 5.6: Grafana Dashboard --- real-time visualizations of CPU, memory, network traffic, and request latency for the deployed application.

Fly.io provides a built-in **Grafana** dashboard for each deployed application. All structured log output from the server instance is forwarded to this dashboard, where it can be filtered by severity, service name, or time range. The dashboard also provides real-time visualizations of CPU utilization, memory consumption, network traffic, and request latency for the running machine. This unified observability interface allows operators to correlate application errors with infrastructure metrics, diagnose performance regressions, and monitor the health of the deployment without direct shell access to the server instance.

5.4.4 Authentication and Onboarding

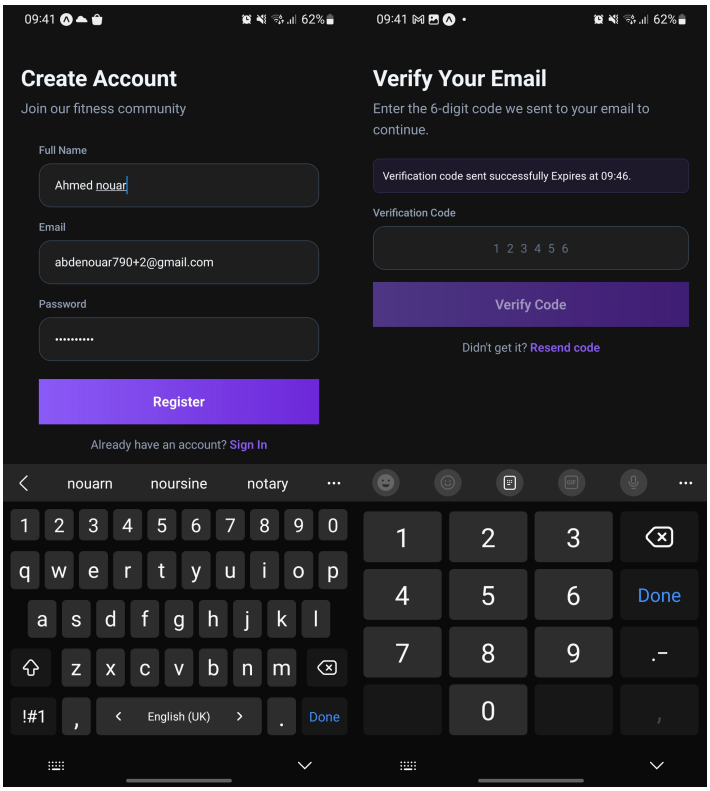


Figure 5.7: Login Screen --- email and password authentication with registration option.

The authentication screen allows users to log in with their email and password. Users without an account can register directly through the application. The onboarding flow captures essential user information including height, weight, fitness goal, experience level, and available training location, creating a personalized experience from the start.

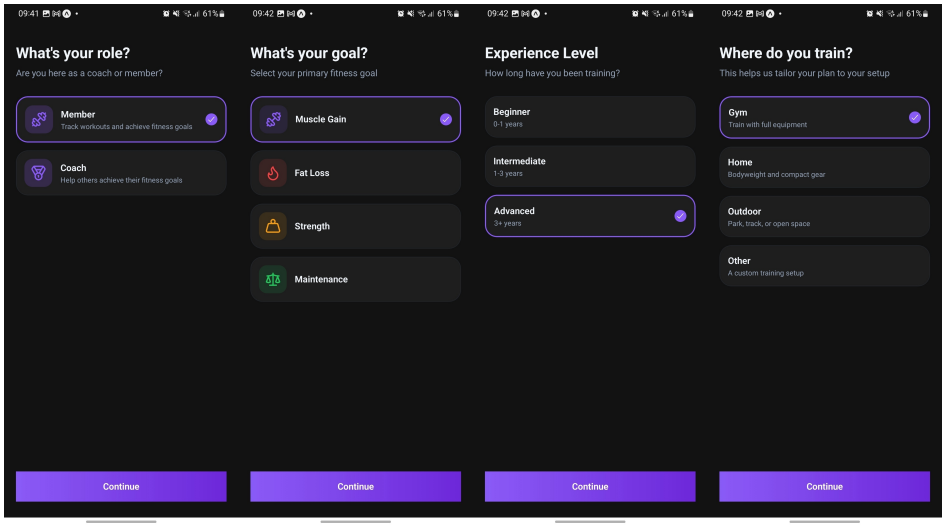


Figure 5.8: Onboarding Screens --- step-by-step profile creation and initial setup for personalized recommendations.

The onboarding process is step-by-step, guiding new users through profile creation and initial setup. This ensures that personalized recommendations and workout suggestions are tailored to their specific circumstances and goals.

5.4.5 Dashboard and Home Screen

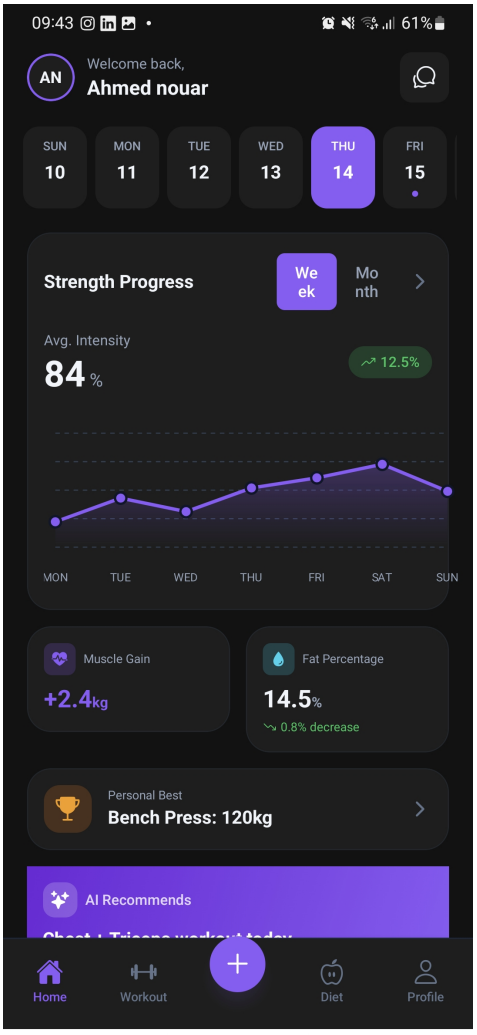


Figure 5.9: Dashboard/Home Screen --- key metrics, daily streaks, and quick-access features.

The dashboard serves as the main hub of the application, displaying key metrics and quick-access features. Users can see their current stats (calories consumed, water intake, workouts completed), view daily streaks, and access primary features like starting a workout or logging a meal. Quick action buttons provide convenient shortcuts to frequently used functions.

5.4.6 Workout Planning and Tracking

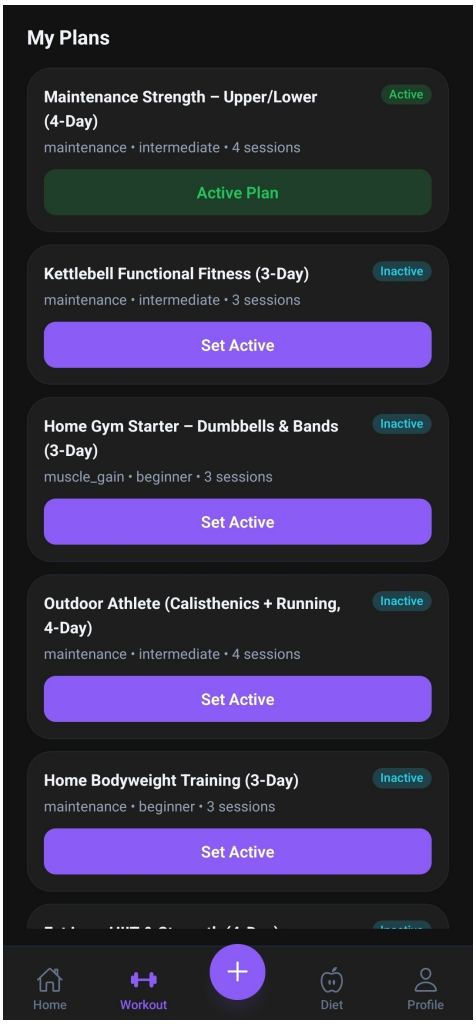


Figure 5.10: Workout Plans List --- browsing pre-designed and custom workout plans with duration, intensity, and goal indicators.

The workout planner allows users to browse pre-designed workout plans or create custom plans tailored to their fitness level and goals. Each plan specifies the duration, intensity level, and target goal (muscle gain, fat loss, strength, maintenance), implementing the core training principles discussed in Chapter 2, especially exercise selection, training volume, progressive overload, consistency, and recovery awareness.

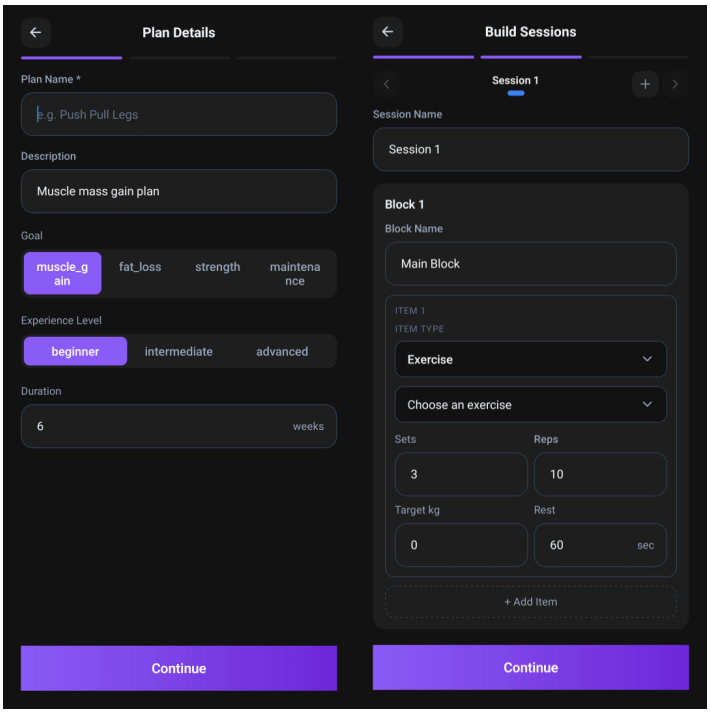


Figure 5.11: Create Workout Plan --- defining sessions, blocks, and exercises with sets, reps, rest periods, and target weights.

When creating a custom workout plan, users can define multiple training sessions per week. Each session is composed of blocks (like warm-up, main workout, cooldown), and each block contains individual exercises with specified sets, reps, rest periods, and target weights.

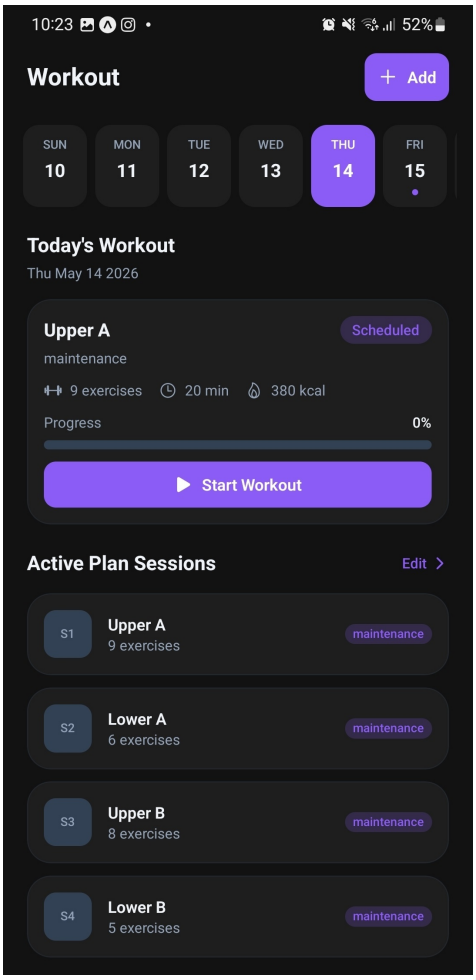


Figure 5.12: Active Workout Session --- live tracking interface with current exercise, target metrics, and set logging controls.

During a workout, users see a live tracking interface that displays the current exercise, target metrics, and provides controls to log completed sets. The application tracks performance over time, allowing users to monitor progress and adjust future sessions based on actual performance data.

5.4.7 Nutrition and Diet Management

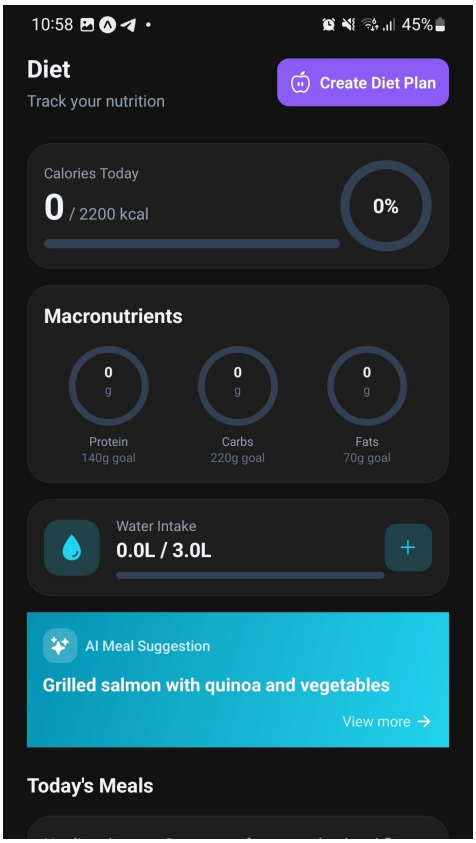


Figure 5.13: Diet Plan Overview --- daily nutritional goals for calories, protein, carbs, fats, and water intake.

The diet planning screen enables users to set daily nutritional goals (calories, protein, carbs, fats, water intake). Users can browse recommended meals or create personalized meal plans that align with their targets. This integrates nutrition with the user's goals and coaching context, reflecting the nutrition-management principles identified in Chapter 2 and avoiding the separation between nutrition and training observed in several applications reviewed in Chapter 3.

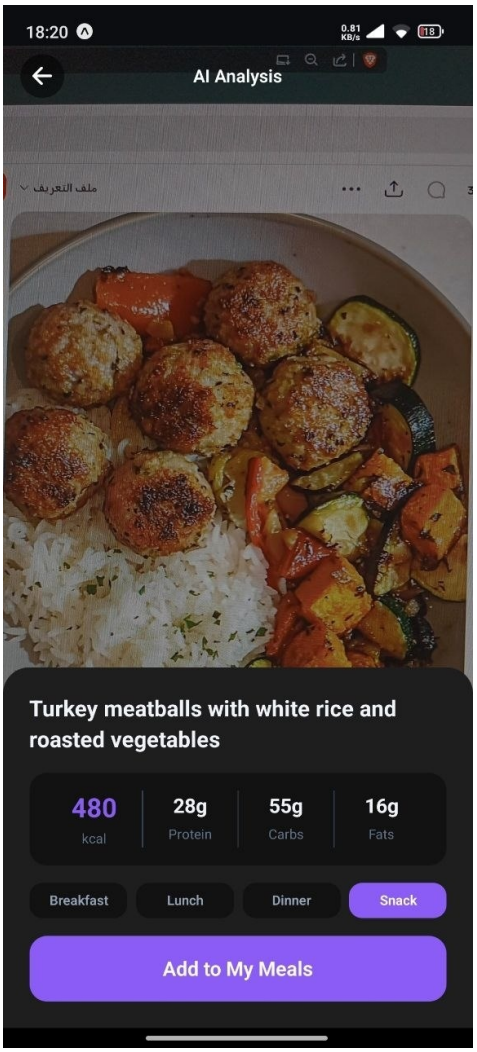


Figure 5.14: Meal Logging --- manual entry and AI-powered food image analysis for automated nutritional estimation.

Users can log meals manually or use the innovative food image analysis feature, which leverages AI to automatically detect and analyze food photographs, estimating nutritional content and updating daily totals. This feature significantly enhances user engagement by simplifying the logging process.

5.4.8 Progress Tracking

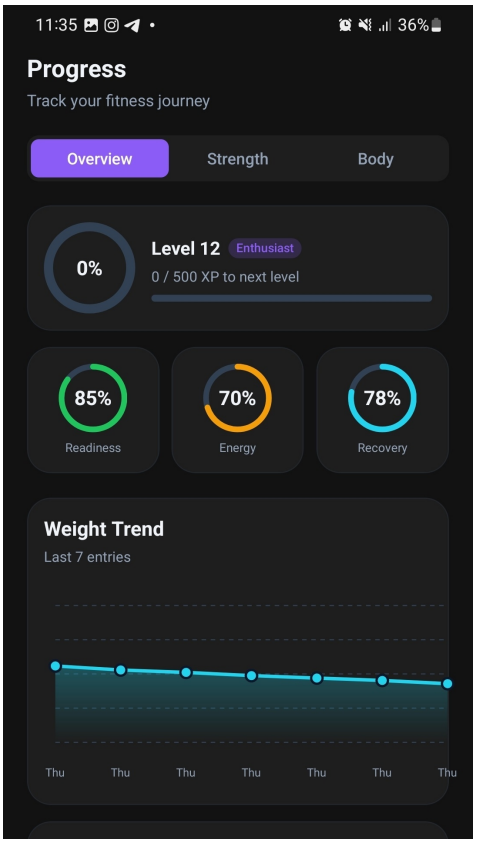


Figure 5.15: Progress Charts --- line charts for weight trends, bar charts for workout frequency, and personal record tracking.

The progress screen provides visual analytics including line charts for weight trends over time, bar charts comparing workout frequency, and personal record tracking. These visualizations help users understand their fitness journey and celebrate milestones.

5.4.9 AI Coach and Messaging

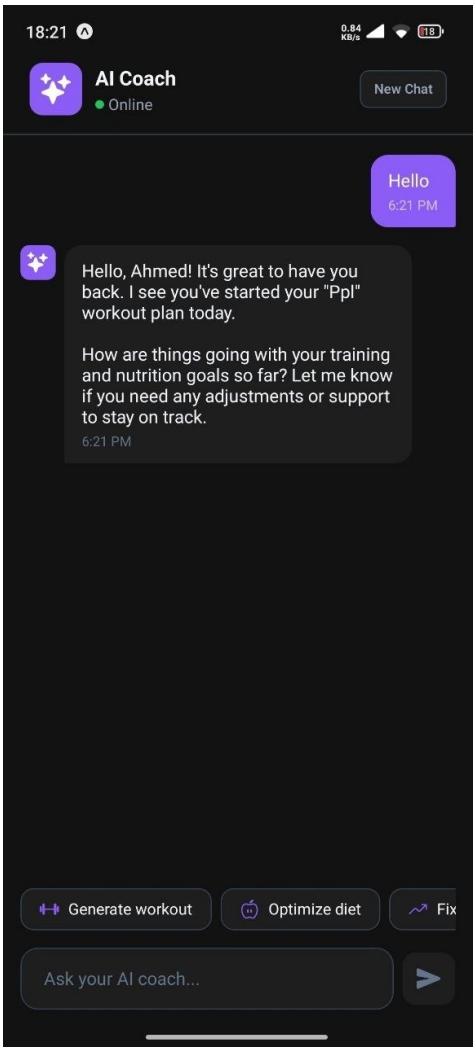


Figure 5.16: AI Coach Chat --- natural language conversations with personalized guidance on workouts, nutrition, and form.

The AI coach feature provides personalized guidance through natural language conversations and represents the main implementation of the agent foundation presented in Chapter 1. Users can ask questions about their workouts, nutrition, form, and receive real-time recommendations. The AI maintains conversation history and combines stored user context, retrieved information, and backend service integration so that discussions remain coherent and recommendations can be adapted to the member's goals, equipment, history, and constraints.

Retrieval-Augmented Workout Recommendations

The workout recommendation component uses a retrieval-augmented generation (RAG) workflow to ground AI responses in structured workout examples instead of relying only on the language model's general knowledge. The knowledge base is built from the Hugging

Face dataset **Comprehensive 7-Day Workout Plans**, which contains approximately 900 question--answer rows describing workout routines, weekly plans, and training recommendations [28]. Each row is embedded with `nomic-embed-text` through Ollama and stored in the PostgreSQL vector index. When a member asks for a workout recommendation, the backend retrieves semantically similar examples from this dataset, combines them with the user's profile, goals, equipment, and constraints, and passes the resulting context to the AI coach. This design makes generated plans more consistent with real workout structures while still allowing personalization for each user.

5.4.10 Member-Coach Messaging

The application also supports direct messaging between members and real human coaches. This feature complements the AI coach by giving users access to more reliable online coaching when they need professional judgment, personalized feedback, or clarification that should not depend only on automated recommendations. Through this channel, members can discuss workout plans, nutrition strategies, progress difficulties, injuries, schedule constraints, and proposed plan adjustments with their assigned coach.

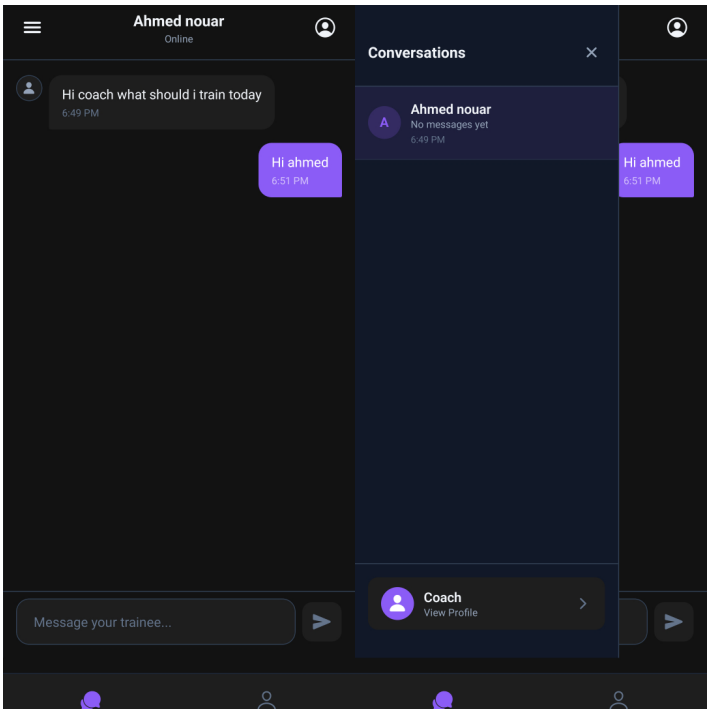


Figure 5.17: Messaging Interface --- real-time communication between members and human coaches with persisted conversation history.

Messages are persisted and organized by conversation, enabling both the member and the coach to refer back to previous advice, accepted proposals, and ongoing progress discussions. This human communication layer improves trust and accountability because the AI assistant can provide immediate support, while the coach can review complex situations and guide the

member with expert supervision.

5.4.11 Equipment and Preferences

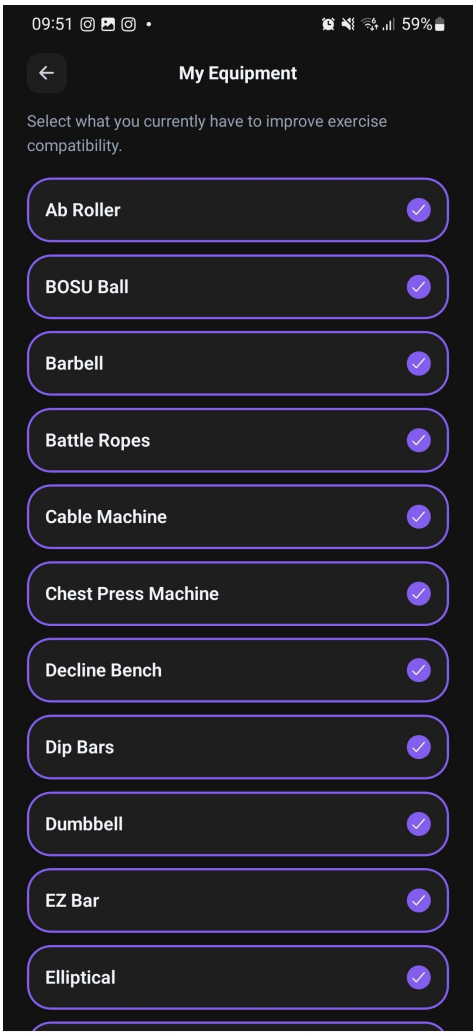


Figure 5.18: Equipment Selection --- specifying available equipment for filtering and recommending compatible exercises.

Users can specify which equipment they have access to (dumbbells, barbells, machines, etc.). This information is used to filter and recommend exercises that match their available resources, ensuring workout plans are practical and achievable.

5.4.12 User Profile

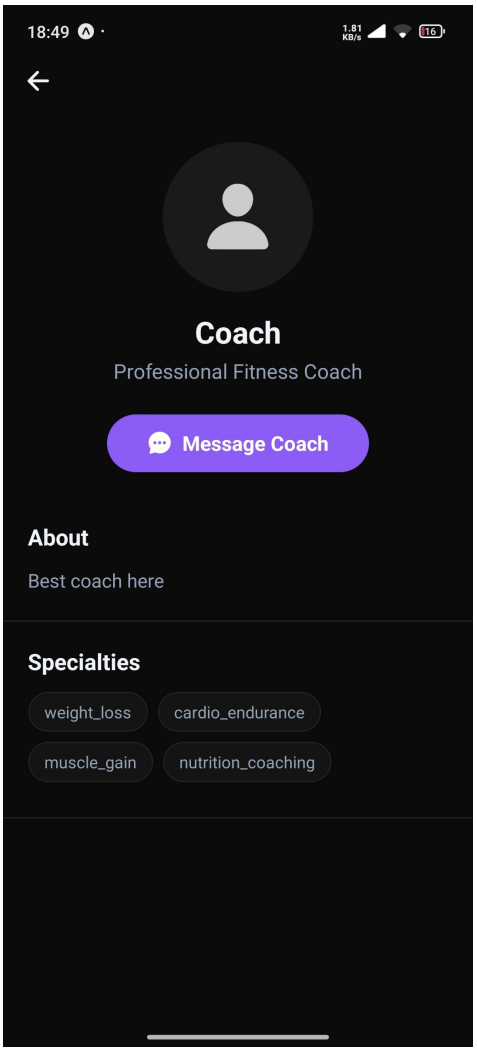


Figure 5.19: Profile Screen --- personal information, fitness goals, notification settings, and account management.

The profile screen allows users to view and update their personal information, fitness goals, and preferences. Users can also manage notification settings, view their subscription status, and access account settings.

5.5 API Integration and Backend Communication

Our application communicates with a REST API backend (`https://finals-project.fly.dev`) through a series of specialized API classes organized by domain:

Table 5.3: API classes and their responsibilities

API Class	Responsibility
`AuthAPI`	Handles user registration, login, token refresh, and onboarding
`WorkoutAPI`	Manages workout plans, exercises, session logging, and equipment
`DietAPI`	Handles diet plans, meal logging, water tracking, and AI-powered food analysis
`CoachingAPI`	Manages coach directory and coaching relationships
`MessagingAPI`	Handles conversations and real-time messaging
`AIAPI`	Powers the AI coach feature with conversation management

Authentication and Authorization

Access to protected endpoints is governed by a dual-token authentication scheme. Upon successful login or registration, the server issues two JSON Web Tokens: a short-lived **access token** (five-minute expiry, signed with HS256 via the `jose` library) carried in the `x-access-token` header, and a long-lived **refresh token** stored in the database with a SHA-256 hash. When the access token expires, the client can exchange the refresh token for a new access token through the `/v1/auth/refresh` endpoint without requiring the user to re-authenticate.

Sensitive operations --- such as coach-only plan management endpoints --- are additionally protected by role-based middleware that verifies the authenticated user's role (`member` or `coach`) before allowing the request to proceed. Account verification is handled through a time-limited one-time password (OTP) sent via email using the SMTP `nodemailer` library.

Real-Time Communication

The API supports both standard JSON responses and **Server-Sent Events (SSE)** for real-time streaming of AI chat messages. The `/v1/member/ai/conversation/:id/chat` endpoint returns a streaming response that pushes tokens progressively to the mobile client using the `react-native-sse` library, providing a fluid conversational experience without WebSocket overhead.

5.6 Conclusion

Integrating the mobile client, backend services, database layer, API contract, and AI modules provides a complete implementation of the proposed fitness tracking application. This implementation makes workout planning, exercise logging, nutrition management, progress tracking, coach communication, and AI guidance accessible through one coherent platform. Incorporating the research and design results into a deployed application demonstrates the practical value of the thesis, as the AI-agent concepts, fitness-management principles, and

comparative findings are realized in a functional full-stack system.

General Conclusion

Summary of Achievements

This thesis set out to design, develop, and evaluate a comprehensive mobile fitness tracking application enhanced with intelligent AI coaching capabilities. The project successfully delivered a full-stack platform that addresses the fragmented nature of existing fitness applications by integrating workout logging, nutrition tracking, AI-powered coaching, and coach-member communication within a single, coherent system. The work progressed from the study of AI agents and context management, to the analysis of fitness variables required for meaningful coaching, to the comparison of existing applications, and finally to the design and implementation of a concrete solution.

The key achievements of this work are as follows:

Theoretical Foundation: Chapter 1 provided a thorough review of AI agents, large language models, and context management techniques including Retrieval-Augmented Generation, system prompts, tool calling, and managed context providers, establishing the theoretical basis for the intelligent coaching features.

Domain Analysis: Chapter 2 explored the core principles of gym training, including exercise selection, progressive overload, periodization, nutrition management, and the role of AI in modern fitness coaching.

Comparative Study: Chapter 3 conducted a comprehensive comparative analysis of existing fitness applications across three regions --- Algeria, the Arab world, and globally --- identifying key limitations in data integration, personalization, and localized support that motivated the proposed solution.

System Design: Chapter 4 delivered a complete design specification using UML diagrams (use case, sequence, class, and activity diagrams), a normalized PostgreSQL schema managed through Prisma ORM, and a layered backend architecture built with Bun, ElysiaJS, and Sandly dependency injection.

Technical Implementation: Chapter 5 realized the design as a production-ready application using React Native and Expo for the mobile frontend, with a Bun/Elysia backend deployed on Fly.io. The platform integrates Google Gemini for conversational AI coaching and food image analysis, and Ollama's nomic-embed-text model for RAG-based workout

plan recommendations using pgvector similarity search.

Developer Experience: An automated OpenAPI 3.0.3 specification is generated directly from Elysia route definitions, providing interactive API documentation at `/docs` and Model Context Protocol (MCP) tools at `/mcp` for AI-assisted development workflows.

Contributions

The main contributions of this thesis can be summarized as:

A fully functional, cross-platform mobile fitness application that unifies workout tracking, nutrition management, and AI-powered coaching in a single platform.

Integration of a context-aware AI coaching agent that leverages RAG, tool calling, and structured output to generate personalized workout plans and provide real-time fitness advice.

A comparative analysis of fitness applications across Algerian, Arab, and global markets, highlighting gaps in data integration and localized personalization.

An OpenAPI-driven development pipeline that ensures end-to-end type safety from backend business logic to mobile client consumption through auto-generated API contracts.

A production deployment on Fly.io with structured logging via evlog and observability through Grafana dashboards.

Future Work

While the current implementation achieves its core objectives, several directions remain open for future enhancement:

Wearable Device Integration: Connecting with smartwatches and fitness bands to automatically capture heart rate, sleep patterns, and activity data for more accurate recovery and performance analysis.

Advanced Analytics and Prediction: Incorporating machine learning models to predict progress plateaus, recommend deload weeks, and forecast body composition changes based on historical data.

Injury Prevention Support: Adding features that analyze exercise technique, training load, recovery status, and user-reported discomfort to warn users about potential injury risks and recommend safer alternatives or rest periods.

Social and Community Features: Adding group challenges, leaderboards, and social sharing to increase user engagement and long-term adherence.

Expanded AI Capabilities: Integrating additional LLM providers, fine-tuning models on fitness-specific data, and developing more sophisticated multi-agent coaching workflows.

Offline Mode: Implementing full offline support with local data caching and synchronization to accommodate users with limited internet connectivity.

Multi-language Support: Extending the application to fully support French and Arabic interfaces alongside the current English implementation to better serve the Algerian and regional user base.

We believe the foundation laid by this work provides a solid platform for continued innovation in AI-driven fitness coaching and mobile health applications.

References

[1] A. Vaswani et al., ``Attention is all you need,'' *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[2] R. Bommasani et al., ``On the opportunities and risks of foundation models,'' *arXiv preprint arXiv:2108.07258*, 2021. [Online]. Available: https://arxiv.org/abs/2108.07258

[3] S. Yao et al., ``ReAct: Synergizing reasoning and acting in language models,'' *arXiv preprint arXiv:2210.03629*, 2022. [Online]. Available: https://arxiv.org/abs/2210.03629

[4] P. Lewis et al., ``Retrieval-augmented generation for knowledge-intensive NLP tasks,'' *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. [Online]. Available: https://arxiv.org/abs/2005.11401

[5] LangChain, *System prompts and guardrails in AI agents*, Accessed: 2026-03-28, 2023. [Online]. Available: https://python.langchain.com/docs/guides/safety

[6] LlamaIndex, *Managed context providers for AI agents*, Accessed: 2026-03-28, 2023. [Online]. Available: https://www.llamaindex.ai/guides/context

[7] Google Research, *The importance of context in AI agents*, Accessed: 2026-03-28, 2023. [Online]. Available: https://ai.googleblog.com/2023/07/context-in-ai-agents.html

[8] Anonymous, ``Goal achievement in fitness apps,'' in *ACM Conference*, 2019. [Online]. Available: https://dl.acm.org/doi/10.1145/3308558.3313432

[9] UCLA Health, *Are fitness trackers enough?* Accessed: 2026-03-28, 2022. [Online]. Available: https://www.uclahealth.org/news/article/are-fitness-trackers-enough

[10] Wikipedia, *Progressive overload*, Accessed: 2026-03-28. [Online]. Available: https://en.wikipedia.org/wiki/Progressive_overload

[11] Hasan Yildirim, *Exercises dataset*, GitHub repository. Accessed: 2026-05-31, 2024. [Online]. Available: https://github.com/hasaneyldrm/exercises-dataset

[12] Mayo Clinic, *Back exercises*, Accessed: 2026-03-28, 2026. [Online]. Available: https://www.mayoclinic.org/healthy-lifestyle/adult-health/in-depth/back-pain/art-20546859

[13] BarBend, *Different types of training periodization*, Accessed: 2026-03-28, 2022. [Online]. Available: https://barbend.com/different-types-of-training-periodization/

[14] Google Shared Link, *Periodization model image*, Shared image source. Accessed: 2026-05-31, 2026. [Online]. Available: https://share.google/EsiOxqoNGnzZQRziu

[15] AbhiFit, *TDEE calculator*, Accessed: 2026-03-28, 2026. [Online]. Available: https://abhifit.com/tdee-calculator/

[16] A. Jackson, *Getting started with a macro-balanced diet*, Beehive Meals blog post. Accessed: 2026-05-31, 2023. [Online]. Available: https://beehivemeals.com/blogs/customer-resources/getting-started-with-a-macro-balanced-diet

[17] Anonymous, *AI fitness coaching research*, 2025.

[18] Selina Wamucii, *Algerian food prices*, Accessed: 2026-03-28. [Online]. Available: https://www.selinawamucii.com/insights/prices/algeria/live-chicken/

[19] Power Fitness Club Dz, *Power fitness & sport nutrition*, Official website. Accessed: 2026-05-31, 2026. [Online]. Available: https://powerfitness-dz.club/

[20] BGR SMART SOLUTION, *BGR SMART GYM*, Google Play listing. Accessed: 2026-05-31, 2024. [Online]. Available: https://play.google.com/store/apps/details?id=io.bgrsmart.bgrsmartgym

[21] MayStrong, *Maystrong: Le sport, près de chez vous*, Official website. Accessed: 2026-05-31, 2026. [Online]. Available: https://maystrong-app.com/

[22] Kam Calorie, *Kam calorie app*, Apple App Store listing. Accessed: 2026-05-31, 2026. [Online]. Available: https://apps.apple.com/app/kam-calorie/id6748948785

[23] Madar Software, *Rashaqa: Steps, calorie counter*, Google Play listing. Accessed: 2026-05-31, 2026. [Online]. Available: https://play.google.com/store/apps/details?id=com.madarsoft.fitness

[24] Soraate, *Soraate calorie tracking app*, Official website. Accessed: 2026-05-31, 2026. [Online]. Available: https://soraate.com/

[25] Strava, *Strava mobile app*, Official website. Accessed: 2026-05-31, 2026. [Online]. Available: https://www.strava.com/mobile

[26] MyFitnessPal, *Myfitnesspal mobile app*, Official website. Accessed: 2026-05-31, 2026. [Online]. Available: https://www.myfitnesspal.com/

[27] Google Fitbit, *Fitbit app*, Official website. Accessed: 2026-05-31, 2026. [Online]. Available: https://www.fitbit.com/global/us/products/services/app

[28] CristiD7, *Comprehensive 7-day workout plans*, Hugging Face dataset. Accessed: 2026-05-28, 2024. [Online]. Available: https://huggingface.co/datasets/CristiD7/Comprehensive_7Day_Workout_Plans

Appendix A: Calculation Details

This appendix presents additional formulas and calculations used throughout the thesis for fitness assessment, training planning, and nutritional guidance.

.1 Total Daily Energy Expenditure (TDEE)

The Total Daily Energy Expenditure represents the total number of calories burned per day. It is calculated as follows:

$$\text{TDEE} = \text{BMR} \times \text{Activity Factor}$$

.1.1 Basal Metabolic Rate (BMR)

Using the Mifflin-St Jeor equation:

For men:

$$\text{BMR} = 10 \times \text{weight (kg)} + 6.25 \times \text{height (cm)} - 5 \times \text{age (years)} + 5$$

For women:

$$\text{BMR} = 10 \times \text{weight (kg)} + 6.25 \times \text{height (cm)} - 5 \times \text{age (years)} - 161$$

.1.2 Activity Factors

Table 4: Activity level multipliers for TDEE calculation

Activity Level	Description	Multiplier
Sedentary	Little or no exercise	1.2
Lightly active	Light exercise 1--3 days/week	1.375
Moderately active	Moderate exercise 3--5 days/week	1.55
Very active	Hard exercise 6--7 days/week	1.725
Extremely active	Very hard exercise, physical job	1.9

.2 Training Volume

Training volume for a given exercise is calculated as:

$$\text{Volume} = \sum_{i=1}^{n} (\text{Sets}_i \times \text{Reps}_i \times \text{Load}_i)$$

Where n is the number of sets performed.

.3 One-Rep Max Estimation (Epley Formula)

When a user performs a set with a submaximal load, the estimated one-rep maximum (1RM) is:

$$\text{1RM} = \text{Load} \times \left(1 + \frac{\text{Reps}}{30}\right)$$

.4 Body Mass Index (BMI)

$$\text{BMI} = \frac{\text{weight (kg)}}{\text{height (m)}^2}$$

.5 Macronutrient Distribution

Daily macronutrient targets based on total caloric intake:

Table 5: Macronutrient calorie values and recommended ranges

Macronutrient	**Calories per gram**	**Recommended range**
Protein	4 kcal/g	15--30% of total calories
Carbohydrates	4 kcal/g	45--65% of total calories
Fats	9 kcal/g	20--35% of total calories

Appendix B: List of Acronyms

Acronym	Full Form
AI	Artificial Intelligence
API	Application Programming Interface
BMI	Body Mass Index
BMR	Basal Metabolic Rate
CPU	Central Processing Unit
DNS	Domain Name System
DTO	Data Transfer Object
FR	Functional Requirement
GPU	Graphics Processing Unit
HIIT	High-Intensity Interval Training
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
IDE	Integrated Development Environment
JSON	JavaScript Object Notation
JWT	JSON Web Token
LLM	Large Language Model
MCP	Model Context Protocol
ML	Machine Learning
MVVM	Model-View-ViewModel
NFR	Non-Functional Requirement
NLP	Natural Language Processing
ORM	Object-Relational Mapping
OTP	One-Time Password
RAG	Retrieval-Augmented Generation
RAM	Random Access Memory
REST	Representational State Transfer
RPE	Rate of Perceived Exertion
SDK	Software Development Kit
SMTP	Simple Mail Transfer Protocol
SSD	Solid-State Drive
SSE	Server-Sent Events
TDEE	Total Daily Energy Expenditure
TPU	Tensor Processing Unit
UML	Unified Modeling Language