

BABEȘ-BOLYAI UNIVERSITY CLUJ-NAPOCA
FACULTY OF MATHEMATICS AND COMPUTER
SCIENCE
SPECIALIZATION Computer Science English

DIPLOMA THESIS

Wellnest: Enhancing User Engagement and Personalization in Mobile Wellness through AI and Sensor Technology

Supervisor
Associate Professor, PhD. Bocicor Maria Iuliana

Author
Bucur Mihnea-Andrei

2025

ABSTRACT

This thesis introduces **Wellnest**, an innovative mobile application designed to support users in achieving balanced health and fitness by providing personalized, context-aware recommendations. Unlike typical fitness trackers, Wellnest combines real-time sensor data, user input, and motivational elements to create a dynamic wellness experience that adapts to individual progress and preferences.

The project is grounded in an understanding of how step tracking technologies, such as accelerometers and gyroscopes, can reliably monitor physical activity. It also draws on behavioral science, applying gamification principles informed by models like Self-Determination Theory and the Fogg Behavior Model to encourage sustained user engagement.

Built with a cross-platform approach using React Native and FastAPI, Wellnest connects seamlessly with Firebase for secure data management and real-time synchronization. The app leverages external services, including OCR for scanning nutrition labels and APIs that provide tailored workout routines. Users benefit from AI-powered meal and exercise plans personalized to their goals, ensuring recommendations remain relevant and motivating.

A notable feature of Wellnest is its custom workout builder, which allows users to add and organize exercises along with beginner-friendly descriptions to promote proper form and understanding. The app also includes a built-in calorie calculator that estimates energy expenditure based on activity type, intensity, and personal metrics, helping users make informed decisions aligned with their fitness objectives.

By integrating sophisticated software architecture, intelligent personalization, and proven motivational strategies, Wellnest offers a comprehensive platform that supports users on their journey to healthier lifestyles with adaptable, engaging guidance.

The work herein reflects an original, independent effort by the author, completed without unauthorized collaboration or assistance.

Contents

1	Introduction	1
1.1	Background and Motivation	1
1.2	Problem Statement	2
1.3	Objectives	2
1.4	Declaration of Generative AI and AI-Assisted Technologies in the Writing Process	2
1.5	Thesis Structure	2
2	Technological and Behavioral Foundations	4
2.1	Step Tracking Technology	4
2.1.1	Pedometers and Sensor-Based Counting	4
2.2	Mobile Sensors in Health Apps	5
2.2.1	Sensor Types	5
2.2.2	Sensor Comparison	6
2.3	Gamification in Health Applications	6
2.3.1	Gamification Principles	6
2.3.2	Behavioral Theories Supporting Gamification	6
2.4	Effectiveness of Gamification	7
2.4.1	Empirical Evidence	7
2.4.2	Engagement Impact of Game Elements	8
2.5	Limitations and Considerations	8
2.5.1	Design Challenges	8
2.5.2	Recommendations	8
3	Token-Based Authentication and JSON Web Tokens (JWT)	9
3.1	Introduction	9
3.2	Concept of Token-Based Authentication	9
3.2.1	Advantages	9
3.3	JSON Web Tokens (JWT)	10
3.3.1	JWT Structure	10
3.3.2	Claims	11

3.4	Password Security: Hashing and Salting	11
3.4.1	Hashing	11
3.4.2	Salting	11
3.4.3	Key Stretching	11
3.4.4	Workflow Integration	11
3.5	Authentication Flow with JWT and Password Hashing	12
3.6	Security Considerations	12
3.7	Additional Use Cases of JWT	13
3.8	Conclusion	13
4	Mobile Development	15
4.1	Introduction	15
4.2	How Mobile Applications Work	15
4.3	Native Architecture: Android and iOS	16
4.3.1	Android Architecture	16
4.3.2	iOS Architecture	16
4.4	Cross-Platform Development with React Native and Expo	17
4.5	Development Lifecycle	17
4.6	UI/UX Design Principles	18
4.6.1	Platform-Specific Design Guidelines	18
4.6.2	UI/UX Best Practices	18
4.7	Performance Optimization	19
4.8	Security Considerations	19
4.9	Conclusion	19
5	Development of the Wellnest Application	21
5.1	Introduction	21
5.2	Use Case Diagram	22
5.3	Requirements and Specifications	24
5.3.1	Functional Requirements	24
5.3.2	Non-Functional Requirements	32
5.4	Backend Design and Technologies	33
5.4.1	System Architecture Overview	33
5.4.2	Technology Stack and Integrations	34
5.4.3	AI-Powered Personalized Recommendations	37
5.5	Security and Privacy	40
5.5.1	User Data Protection	40
5.5.2	Data Storage and Encryption	40
5.5.3	Secure Communication Channels	41
5.5.4	Third-Party Services and API Keys	41

5.5.5	Error Handling and Fail-Safe Design	41
5.5.6	Privacy by Design	41
6	Technical and Architectural solutions	42
6.1	Project Structure Philosophy	42
6.2	Component Architecture and Reusability	43
6.3	Framework and Tooling Choices	45
6.4	Custom Hooks for Data Logic	45
6.5	Gamification and Feedback System	46
6.6	Design Strategy and UX Focus	46
6.7	Summary	47
7	OCR Ingredient Analysis Functionality	48
7.1	Overview	48
7.2	Frontend Design	48
7.2.1	Image Selection	48
7.2.2	Text Extraction	49
7.2.3	Ingredient Parsing and Classification	50
7.2.4	Results Display	50
7.3	Backend Integration	52
7.4	Security Measures	53
7.5	Testing and Reliability	53
8	Conclusion	55
	Bibliography	56

Chapter 1

Introduction

1.1 Background and Motivation

In recent years, the global focus on health and wellness has increased dramatically, driven by rising awareness of lifestyle-related diseases and the importance of preventive care. However, many individuals face challenges in maintaining healthy habits due to busy schedules, lack of personalized guidance, and limited motivation.[Eur25]

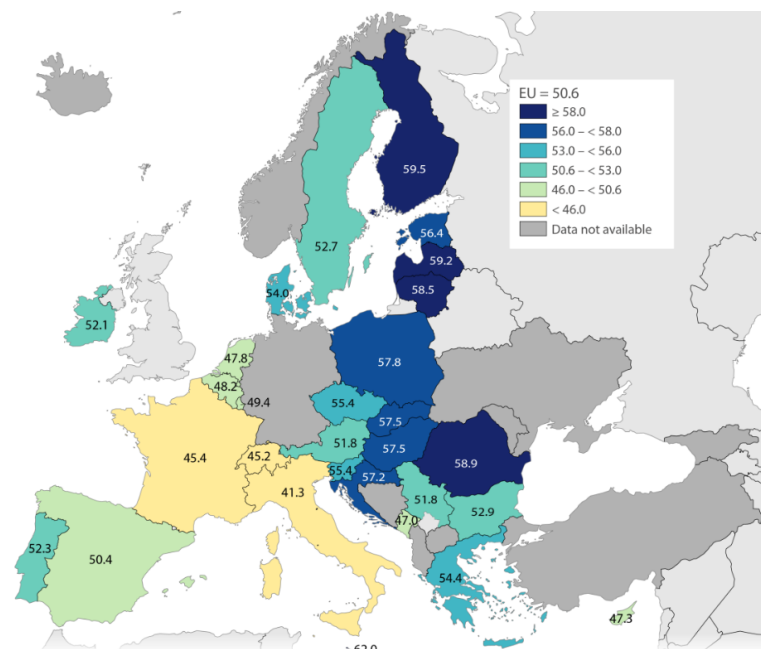


Figure 1.1: Share of overweight people aged 16 or over(2022)

Mobile health applications have emerged as a promising solution to these issues by offering users easy access to health monitoring, personalized advice, and motivational tools anytime and anywhere.

1.2 Problem Statement

Despite the abundance of health and fitness apps available on the market, many suffer from fragmentation, focusing on only one aspect such as activity tracking or meal logging. This often leads to user frustration and discontinuation. Additionally, existing applications sometimes lack reliable security measures or fail to provide meaningful, personalized insights based on real data. There is a clear need for an integrated platform that combines multiple wellness features, ensures data security, and uses intelligent algorithms to adapt to individual user needs.

1.3 Objectives

This thesis presents *Wellnest*, a cross-platform mobile application designed to fill this gap by offering a holistic and secure wellness experience. The key objectives include:

- Developing a modular app that integrates real-time step tracking, personalized meal and workout recommendations, and OCR-based ingredient analysis.
- Implementing robust security mechanisms such as bcrypt password hashing and JWT-based authentication.
- Leveraging AI and third-party APIs to provide adaptive, context-aware wellness guidance.
- Enhancing user motivation and engagement through gamification features informed by behavioral psychology models.

1.4 Declaration of Generative AI and AI-Assisted Technologies in the Writing Process

In the development of this thesis, the author utilized ChatGPT by OpenAI to assist with refining language, enhancing clarity, and improving overall structure. The author carefully reviewed and edited all AI-generated suggestions and retains full responsibility for the content's originality, accuracy, and academic integrity.

1.5 Thesis Structure

This thesis is structured as follows:

- **Chapter 2: Technological and Behavioral Foundations**
Provides the theoretical and technological background underpinning the Wellnest application, including behavioral models and relevant technologies.
- **Chapter 3: Mobile Development**
Discusses the principles and tools used in developing cross-platform mobile applications, focusing on React Native and Expo.
- **Chapter 4: Token-Based Authentication and JSON Web Tokens (JWT)**
Covers the concepts of secure authentication, session management, and the use of JWT in mobile applications.
- **Chapter 5: Development of the Wellnest Application**
Describes the overall development process, key features, and integration of Wellnest's various modules.
- **Chapter 6: Technical and Architectural solutions**
Explores the system architecture, including frontend-backend communication, APIs, and data flow.
- **Chapter 7: Authentication Functionality**
Details the implementation of user registration, login, and session handling in Wellnest.
- **Chapter 8: OCR Ingredient Analysis Functionality**
Explains the ingredient scanning and analysis feature, including OCR integration and local database matching.
- **Chapter 9: Conclusion**
Summarizes the work done, results achieved, and outlines future research directions.

Chapter 2

Technological and Behavioral Foundations

2.1 Step Tracking Technology

Step tracking is a key functionality in modern health applications. It allows users to monitor their daily physical activity using mobile and wearable devices. The tracking process relies on embedded sensors that detect motion patterns and translate them into data such as step count, distance, and calories burned [BTLC17].

2.1.1 Pedometers and Sensor-Based Counting

Pedometers, also known as step counters, detect movement through various mechanisms and convert that into a step count. These are embedded in devices such as smartphones, fitness trackers, and smartwatches.

Types of Step Counters

- **Mechanical Pedometers** – Detect movement through pendulum action.
- **Electronic Pedometers** – Use accelerometers for improved accuracy.
- **Sensor-Driven Apps** – Found in smartphones using built-in accelerometers and gyroscopes.

Accuracy Challenges

In a study conducted on 20 healthy individuals [PW24], step accuracy was evaluated by comparing different placements of pedometers:

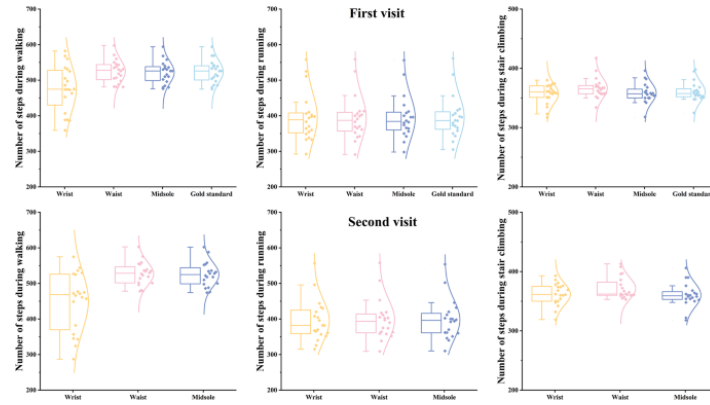


Figure 2.1: Accuracy of Pedometer Based on Body Placement [PW24]

- **Wrist-worn devices** tend to produce the most inaccurate results during walking.
- **Waist and midsole** positions offer more reliable data .
- Activities such as stair climbing and running increased error variance.

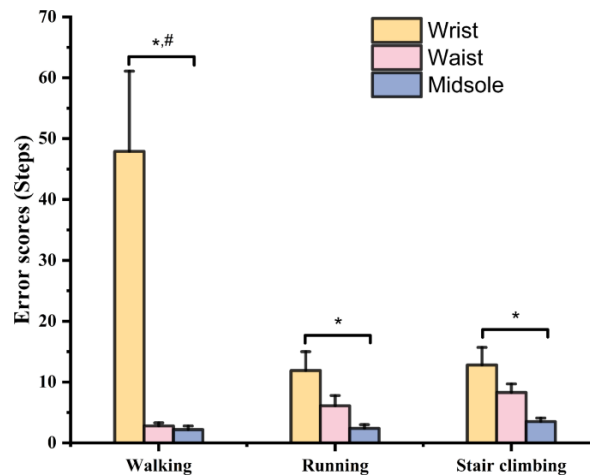


Figure 2.2: Error Rate in Step Counting by Device Location [PW24]

2.2 Mobile Sensors in Health Apps

Mobile and wearable devices integrate multiple sensors to facilitate step detection and physical activity monitoring.

2.2.1 Sensor Types

- **Accelerometers** – Measure linear acceleration; essential for detecting movement patterns.

- **Gyroscopes** – Detect angular movement and orientation.
- **GPS Sensors** – Track geographic movement and estimate distance, useful for running or walking outdoors.

2.2.2 Sensor Comparison

Sensor	Function	Accuracy
Accelerometer	Step detection	High
Gyroscope	Movement orientation	Medium
GPS	Location tracking	High (outdoors)

Table 2.1: Comparison of Common Sensors in Fitness Tracking

2.3 Gamification in Health Applications

Gamification refers to the application of game-design principles in non-game environments. In the context of health apps, it increases motivation and long-term adherence by making physical activity more engaging. Common elements used to implement gamification strategies are summarized in Table 2.2.

2.3.1 Gamification Principles

According to Deterding [DDKN11], gamification in fitness applications includes:

Table 2.2: Common Gamification Elements in Health Applications

Element	Description
Points	Instant feedback for achieving daily goals
Badges	Recognition for reaching specific milestones
Leaderboards	Encourages competition through social comparison
Challenges	Time-bound tasks that build consistency

2.3.2 Behavioral Theories Supporting Gamification

Self-Determination Theory (SDT)

SDT suggests that gamified apps can support intrinsic motivation by fulfilling:

- **Autonomy** – Customizable user goals.
- **Competence** – Feedback through progress bars or levels.

- **Relatedness** – Social elements like friend competitions.

[MM23]

Fogg Behavior Model

Behavior occurs when **motivation**, **ability**, and a **trigger** converge [Fog09]. Gamified fitness apps support this by:

- Increasing motivation through rewards.
- Lowering ability threshold with user-friendly interfaces.
- Providing clear triggers such as notifications and reminders.

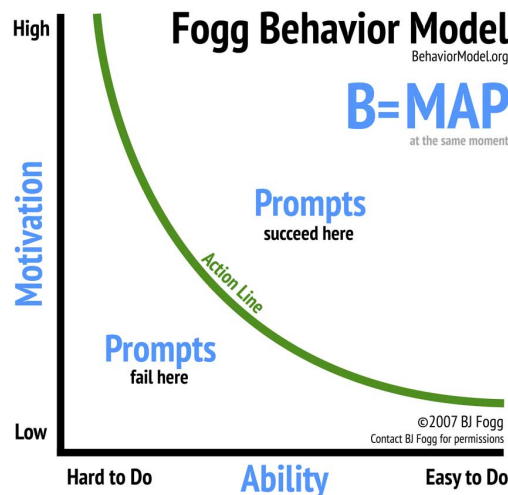


Figure 2.3: Fogg Behavior Model Applied to Fitness Engagement [Fog09]

2.4 Effectiveness of Gamification

2.4.1 Empirical Evidence

A 12-week study compared step counts between users of gamified and non-gamified fitness apps.[MFH⁺24]

- Gamified app users increased daily steps by an average of **2,000**.
- The difference was statistically significant across all demographic groups.

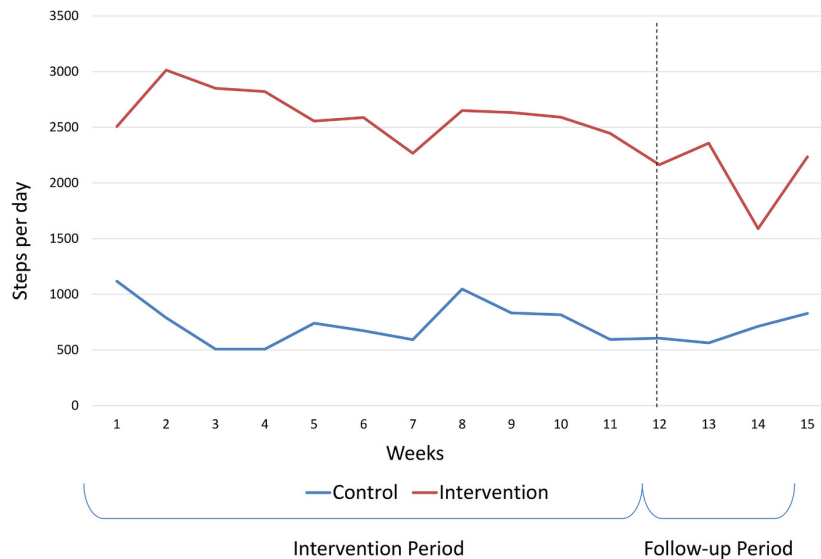


Figure 2.4: Daily Step Increase in Gamified vs Non-Gamified Apps [MFH⁺24]

2.4.2 Engagement Impact of Game Elements

Game Element	Purpose	Engagement Level
Points	Real-time rewards	High
Badges	Visual milestones	Medium
Leaderboards	Competition	High
Challenges	Goal-setting	Very High

Table 2.3: Gamification Features and Their Impact on User Engagement

2.5 Limitations and Considerations

2.5.1 Design Challenges

- **Overjustification Effect:** Users may become dependent on extrinsic rewards.
- **User Fatigue:** Motivation can decline over time.
- **One-size-fits-all Models:** Gamification must adapt to diverse user profiles.

2.5.2 Recommendations

- **Personalization:** Adaptive challenges and reward types.
- **Social Integration:** Encourage team participation and sharing.
- **Balanced Incentives:** Combine intrinsic goals (health) with extrinsic ones (badges).

Chapter 3

Token-Based Authentication and JSON Web Tokens (JWT)

3.1 Introduction

In contemporary web and mobile application development, authentication is pivotal for securing access to resources and services. Token-based authentication has gained prominence, especially in stateless architectures like RESTful APIs, due to its scalability, security, and cross-platform compatibility.

3.2 Concept of Token-Based Authentication

Token-based authentication entails issuing a token to a user upon successful login. This token, typically a JSON Web Token (JWT), is included in subsequent requests to access protected resources. Unlike traditional session-based authentication, tokens are self-contained, allowing servers to remain stateless and simplifying scalability [KHJK16].

3.2.1 Advantages

- **Statelessness:** Eliminates the need for server-side session storage, enhancing scalability.
- **Cross-Platform Compatibility:** Tokens can be used across various clients, including web and mobile applications.
- **Enhanced Security:** Reduces exposure by not sending credentials with every request.

- **Flexibility:** Tokens can carry custom claims to convey user roles, permissions, and other metadata.

3.3 JSON Web Tokens (JWT)

JWT is an open standard (RFC 7519) that defines a compact and self-contained method for securely transmitting information between parties as a JSON object [RPMK24].

3.3.1 JWT Structure

A JWT consists of three parts:

1. **Header:** Specifies the token type and the signing algorithm.
2. **Payload:** Contains claims, which are statements about an entity and additional data.
3. **Signature:** Ensures the token's integrity and authenticity.

These parts are encoded in Base64URL and concatenated with periods, forming the structure: `header.payload.signature`.

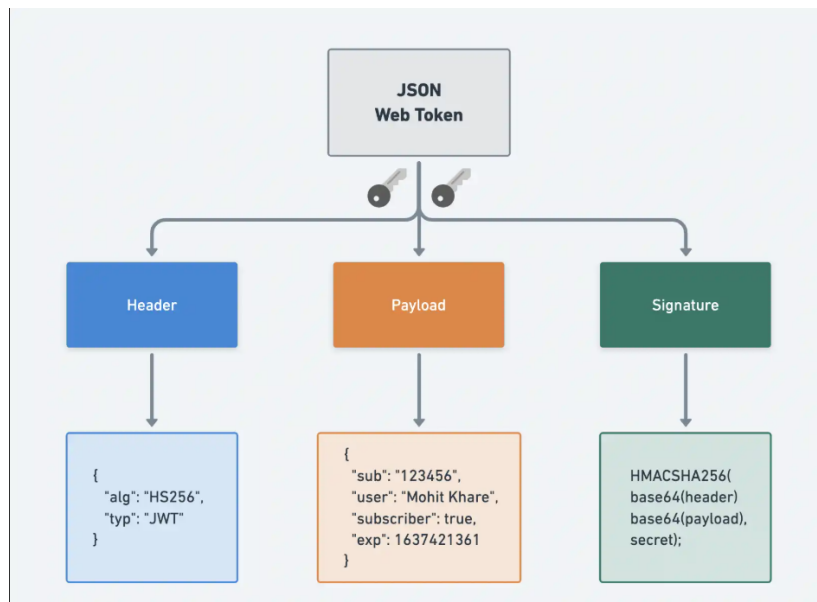


Figure 3.1: Structure of a JSON Web Token (JWT) [Kha20]

3.3.2 Claims

Claims are categorized as:

- **Registered Claims:** Predefined claims such as `iss` (issuer), `exp` (expiration time), and `sub` (subject).
- **Public Claims:** Custom claims with public names to avoid collisions.
- **Private Claims:** Custom claims agreed upon between parties for internal use.

3.4 Password Security: Hashing and Salting

Storing passwords in plaintext is a significant security risk. Instead, passwords should be transformed using cryptographic hashing before storage.

3.4.1 Hashing

Hashing is a one-way function that converts a password into a fixed-length string, making it computationally infeasible to reverse-engineer the original password.

3.4.2 Salting

Salting involves adding a unique random value to the password before hashing. This technique prevents attackers from using precomputed hash tables (rainbow tables) to crack passwords, ensuring that identical passwords yield distinct hashes.

3.4.3 Key Stretching

Modern password hashing algorithms like `bcrypt` implement key stretching, performing multiple rounds of hashing to increase computational cost for attackers attempting brute-force attacks [BEN21].

3.4.4 Workflow Integration

During user registration, the password is concatenated with a salt and then hashed before being saved in the database. Upon login, the submitted password undergoes the same process and is compared to the stored hash for verification.

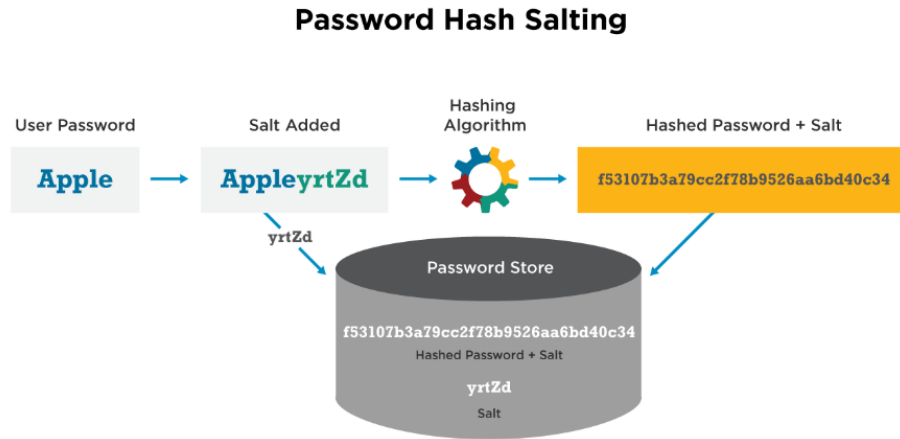


Figure 3.2: Process of Password Hashing and Salting
[Wor21]

3.5 Authentication Flow with JWT and Password Hashing

A typical authentication process leveraging JWT and secure password storage consists of the following steps:

1. The user submits login credentials (email and password).
2. The server retrieves the stored hashed password for the given user.
3. The server hashes the submitted password (including salt) and compares it with the stored hash.
4. If the passwords match, the server generates a JWT containing relevant claims about the user.
5. The token is signed using a secret key and sent back to the client.
6. The client includes the JWT in the `Authorization` header for subsequent API requests.
7. The server validates the token's signature and expiration before granting access.

3.6 Security Considerations

Despite the advantages, several security best practices must be observed:

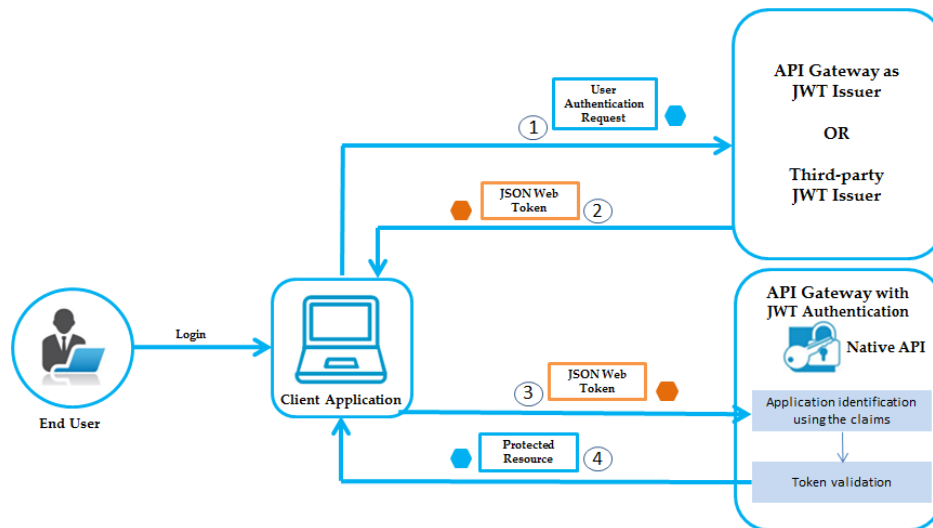


Figure 3.3: Typical Authentication Flow Using JWT [Sof23]

- **Token Expiration:** JWTs should have a reasonable expiration time to limit the window of misuse.
- **Secure Storage:** Tokens must be stored securely on the client side to prevent theft, such as via cross-site scripting (XSS).
- **Revocation Mechanisms:** Stateless tokens complicate revocation; additional infrastructure may be needed to invalidate tokens.
- **Signature Verification:** Servers must always verify the token's signature to ensure authenticity and integrity.

3.7 Additional Use Cases of JWT

Beyond authentication, JWTs are widely used for:

- **Authorization:** Embedding roles and permissions for access control.
- **Data Exchange:** Securely transmitting information between services.
- **Single Sign-On (SSO):** Allowing users to authenticate once and access multiple systems.

3.8 Conclusion

Token-based authentication using JSON Web Tokens has become a fundamental technique for secure and scalable user authentication in modern applications. By

combining robust password hashing methods with JWTs, systems can ensure the protection of user credentials while maintaining stateless and flexible authentication workflows. Proper implementation and adherence to security best practices are essential to maximize the benefits of this approach.

An overview of key concepts related to token-based authentication and password security is presented in Table 3.1, highlighting fundamental mechanisms such as JWT structure, password hashing, salting, and key stretching.

Concept	Description
Token-Based Authentication	Uses a signed token to authenticate users without server-side sessions
JSON Web Token (JWT)	Compact, self-contained token format with header, payload, and signature
Password Hashing	One-way transformation of passwords to prevent plaintext storage
Salting	Adding a unique value to passwords before hashing to prevent rainbow table attacks
Key Stretching	Increasing computational effort in hashing to deter brute-force attacks

Table 3.1: Key Concepts in Token-Based Authentication and Password Security

Chapter 4

Mobile Development

4.1 Introduction

Mobile development encompasses the design and implementation of software applications for portable devices such as smartphones and tablets. These applications can be developed using platform-specific languages and tools (native development), or through frameworks that support deployment across multiple platforms from a single codebase (cross-platform development). The two most prominent mobile operating systems are **Android** (developed by Google) and **iOS** (by Apple), both offering distinct ecosystems, tools, and development paradigms.

4.2 How Mobile Applications Work

Mobile applications operate by interfacing with a device's operating system through well-defined APIs and frameworks. Most applications follow a layered model, consisting of:

- **Frontend (Presentation Layer):** The user interface, typically built using technologies like XML (Android), SwiftUI (iOS), or JavaScript-based tools in cross-platform development.
- **Backend (Business Logic Layer):** Handles core application logic, navigation, data processing, and communication with databases or external services.

Mobile apps are executed in a sandboxed environment, ensuring secure access to hardware and user data. Communication with external resources typically happens via HTTPS APIs, while device functions such as the camera or sensors are accessed through system-level permissions and services [CGd21].

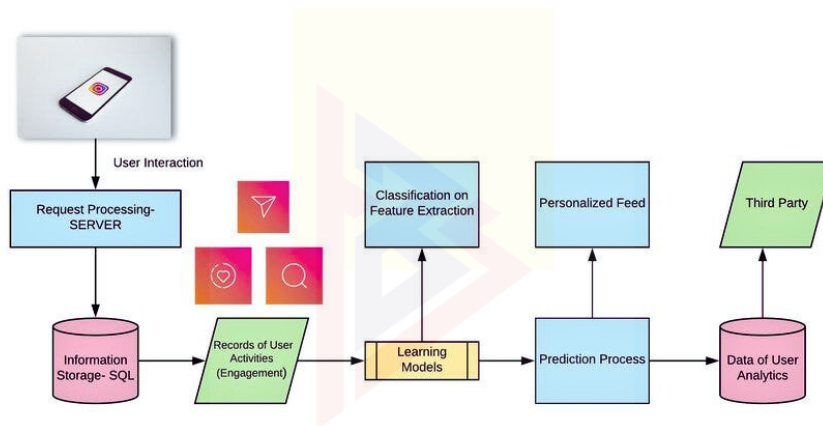


Figure 4.1: General Mobile Application Architecture
[Tec24]

4.3 Native Architecture: Android and iOS

4.3.1 Android Architecture

Android is built on the Linux kernel and follows a modular, layered architecture:

- **Linux Kernel:** Manages low-level hardware access, power, memory, and security.
- **Native Libraries and Android Runtime (ART):** Provides essential libraries (e.g., SQLite, OpenGL) and the runtime environment for executing applications.
- **Application Framework:** Offers higher-level components like Activities, Services, Content Providers, and Broadcast Receivers.
- **Apps:** Typically developed in Kotlin or Java and compiled into APKs (Android Package).

4.3.2 iOS Architecture

iOS is a closed-source platform optimized for Apple's hardware, structured in a tiered stack:

- **Core OS:** Provides low-level system support and security, including encryption and sandboxing.
- **Core Services:** Handles networking, file access, iCloud, and local data persistence.
- **Media Layer:** Supports audio, graphics, video rendering, and animations.

- **Cocoa Touch:** Offers UI frameworks such as UIKit or SwiftUI for building interfaces and handling user input.

4.4 Cross-Platform Development with React Native and Expo

To address the challenge of maintaining separate codebases for Android and iOS, developers often turn to cross-platform frameworks. One of the most widely adopted is **React Native**, developed by Meta (Facebook). It allows developers to write applications in JavaScript (or TypeScript) and render them as native components on both Android and iOS [NC12].

Built on top of React Native, **Expo** is an open-source framework and platform that simplifies the development process:

- Provides pre-built modules for common mobile functionalities like notifications, camera, and GPS.
- Supports hot reloading for rapid development and testing.
- Eliminates the need to configure native build environments during early stages.

While Expo accelerates development and is ideal for MVPs or smaller applications, developers may “eject” from Expo to gain full access to native code when customization or performance optimization is required.

4.5 Development Lifecycle

The mobile app development lifecycle follows an iterative, often agile-based, process. It aligns with software engineering lifecycle models:

Stage	Key Activities
Planning	Requirements gathering, target audience analysis, platform selection
Design	Wireframes, information architecture, UI/UX mockups
Development	Frontend/backend logic, API integration, state management
Testing	Functional tests, UI/UX feedback, performance benchmarking
Deployment	Release on Google Play Store / Apple App Store, version control
Maintenance	Analytics, bug fixing, OS compatibility updates

Table 4.1: Mobile App Development Lifecycle

4.6 UI/UX Design Principles

User Interface (UI) and User Experience (UX) design play a crucial role in mobile app success. Drawing from human-computer interaction theory, good UI/UX focuses on ease of use, efficiency, and visual clarity.

4.6.1 Platform-Specific Design Guidelines

- **Material Design (Android):** A design language developed by Google emphasizing visual hierarchy, motion, and adaptive layouts.
- **Human Interface Guidelines (iOS):** Apple’s standards promote clarity, deference, and depth using gestures and animation.

4.6.2 UI/UX Best Practices

- Minimize cognitive load by reducing unnecessary input fields or actions.
- Implement consistent navigation and gestures across screens.
- Ensure legibility and accessibility (color contrast, text size, screen reader support).
- Use feedback indicators (e.g., loading spinners, haptic responses) to inform users.

4.7 Performance Optimization

Performance is a fundamental concern in mobile app development due to hardware and battery limitations. Efficient apps enhance retention and user satisfaction.

- **Efficient Rendering:** Avoid unnecessary UI re-renders; use flat lists and virtualization.
- **Lazy Loading:** Load components and resources only when needed.
- **Caching and Storage:** Use local storage (e.g., AsyncStorage, SQLite) for offline access.
- **Profiling Tools:** Android Profiler and Xcode Instruments assist in identifying performance bottlenecks.

4.8 Security Considerations

Securing user data is both an ethical obligation and often a legal requirement (e.g., GDPR, HIPAA). Best practices in mobile security include:

Security Measure	Purpose
TLS/SSL Encryption	Secures data in transit to servers
Biometric Authentication	Offers secure, device-level user login
Obfuscation and Minification	Protects source code from reverse engineering
Keychain/Keystore	Securely stores sensitive credentials
App Sandboxing	Isolates application data and prevents cross-app attacks

Table 4.2: Common Mobile App Security Measures

4.9 Conclusion

Modern mobile development brings together platform-specific architecture (Android, iOS) and cross-platform flexibility (React Native, Expo) to meet the increasing demands of users and businesses. Understanding the distinctions between native and hybrid solutions helps developers choose the right tools for the job. As development frameworks evolve and integrate more advanced features, building high-performance, secure, and engaging mobile applications has become more accessible

than ever. Tools like Expo democratize app creation, enabling even small teams to deliver polished products across platforms using a unified technology stack.

Chapter 5

Development of the Wellnest Application

5.1 Introduction

Wellnest is a cross-platform mobile application designed to promote holistic health and fitness through seamless tracking, intelligent feedback, and gamified user engagement. By integrating real-time sensor data, AI-powered recommendations, and personalized goal tracking, Wellnest supports users in cultivating sustainable wellness habits.

The application is built using a modular **client-server architecture** that separates presentation, business logic, and data management layers. The **frontend**, developed using *React Native with TypeScript and Expo*, provides a responsive and intuitive user interface for both Android and iOS devices. The **backend** leverages *Node.js* and *Firebase* services to manage user authentication, store health data, handle business logic, and deliver push notifications.

To facilitate seamless and persistent user interactions within the app, **AsyncStorage** is utilized for locally storing the user's email address on the device. This stored email serves as a key identifier for making authenticated HTTP requests to the backend APIs, enabling the app to fetch and update user-specific data such as profile information, fitness metrics, and personalized settings without requiring repeated login inputs. This approach enhances the user experience by maintaining session continuity and reducing network overhead.

At the core of Wellnest is a contextual health monitoring engine, which continuously collects and analyzes user-specific data such as step counts, caloric intake, workout activity, and personal fitness goals. This data is interpreted through a combination of rule-based logic and AI-driven mechanisms (Groq AI for generating meal and workout suggestions), enabling the app to provide personalized, adaptive

guidance.

Firebase Realtime Database and **Cloud Firestore** serve as the primary data repositories, enabling real-time synchronization of user metrics such as step history, nutritional logs, and workout progress. External APIs including *Microsoft Computer Vision OCR*[Mic25] (for scanning nutritional labels) and the *Ninja Exercise API*[Nin25] (for exercise data) are integrated to enhance the app's functionality.

Security and privacy are fundamental components of the system. Wellnest uses **bcrypt** for secure password hashing with per-user salts, and all communication between the application and backend services is secured via **HTTPS**. Sensitive user data is encrypted both in transit and at rest, and access to private data is strictly controlled using **Firebase Authentication**.

In summary, Wellnest integrates real-time activity monitoring, intelligent suggestion engines, and behavioral motivation strategies to create a dynamic and supportive fitness ecosystem. The sections that follow provide a detailed analysis of each architectural module and the technical decisions that underpin the application's functionality and user experience.

5.2 Use Case Diagram

The use case diagram below illustrates the main interactions between the user and the Wellnest application. It serves as a high-level overview of the system's primary functionalities, capturing the relationships between the user (actor) and the features offered within the app.

The diagram emphasizes both core and auxiliary functions, such as logging meals and workouts, interacting with AI-generated plans, viewing personal data, and navigating through the app's custom drawer.

5.3 Requirements and Specifications

5.3.1 Functional Requirements

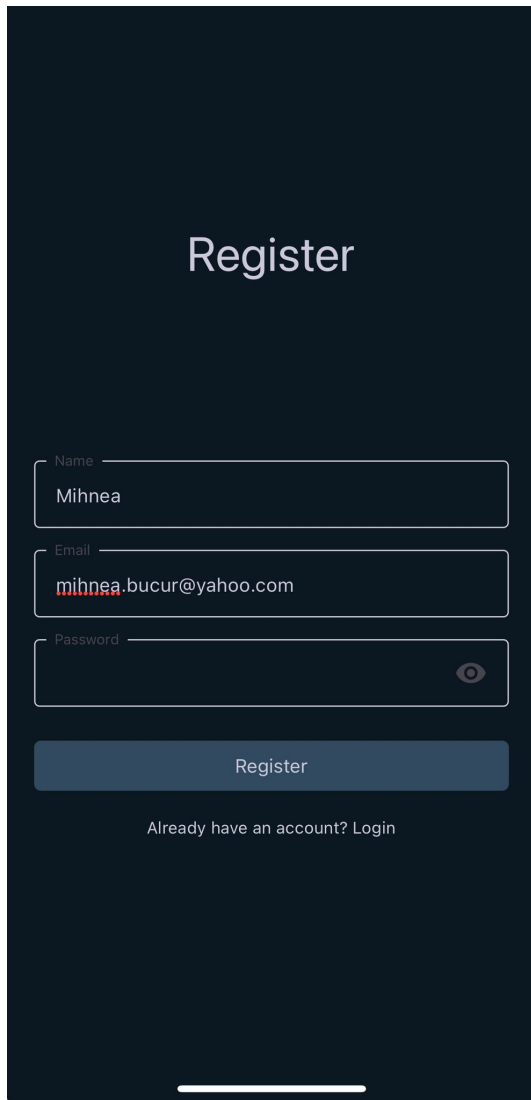
The core features of the Wellnest application are designed to provide users with a seamless experience in tracking their health and fitness data. The following outlines the detailed functional requirements grouped by feature.

User Authentication

Authentication is a vital feature of Wellnest, enabling secure user registration and login through an email/password system. The frontend offers simple and intuitive forms with real-time validation to enhance user experience.

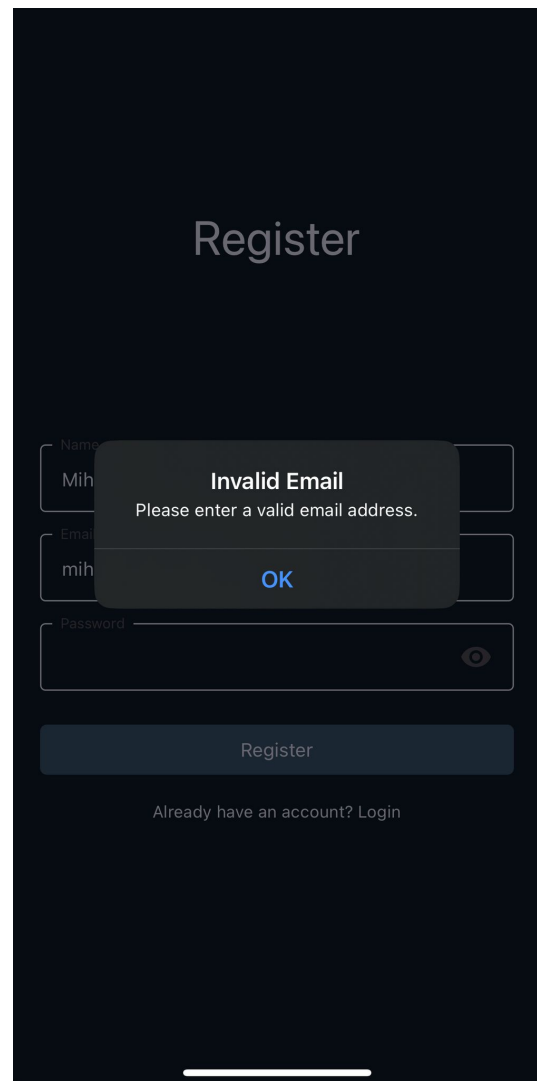
Registration

Users provide their name, email, and password, with validations ensuring email format correctness and password strength (minimum eight characters with letters and numbers). Immediate feedback alerts users to any input errors.



The registration screen features a dark blue background with the title 'Register' at the top. Below the title are three input fields: 'Name' with the value 'Mihnea', 'Email' with the value 'mihnea.bucur@yahoo.com', and 'Password' which is currently empty. A red error message 'Invalid email' is visible below the email field. At the bottom of the form is a 'Register' button, and below that is a link that says 'Already have an account? Login'.

Figure 5.1: Registration screen with input fields



This screen shows the same registration form as Figure 5.1, but with an error dialog box overlaid. The dialog box has a dark background and contains the text 'Invalid Email' in bold, followed by 'Please enter a valid email address.' and an 'OK' button. The background form is dimmed, showing the 'Name' field with 'Mihnea', the 'Email' field with 'mihnea.bucur@yahoo.com', and the 'Password' field. The 'Register' button and the 'Already have an account? Login' link are also visible at the bottom.

Figure 5.2: Registration error when email is missing '@'

Login

Users enter their registered email and password. The system verifies credentials and provides clear error messages if authentication fails.

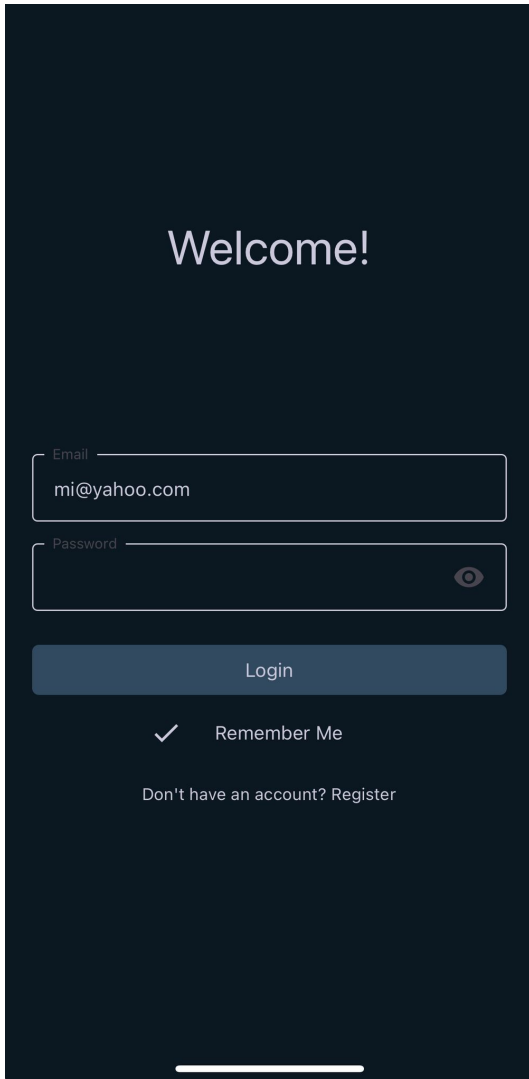


Figure 5.3: Login screen with Email and Password fields

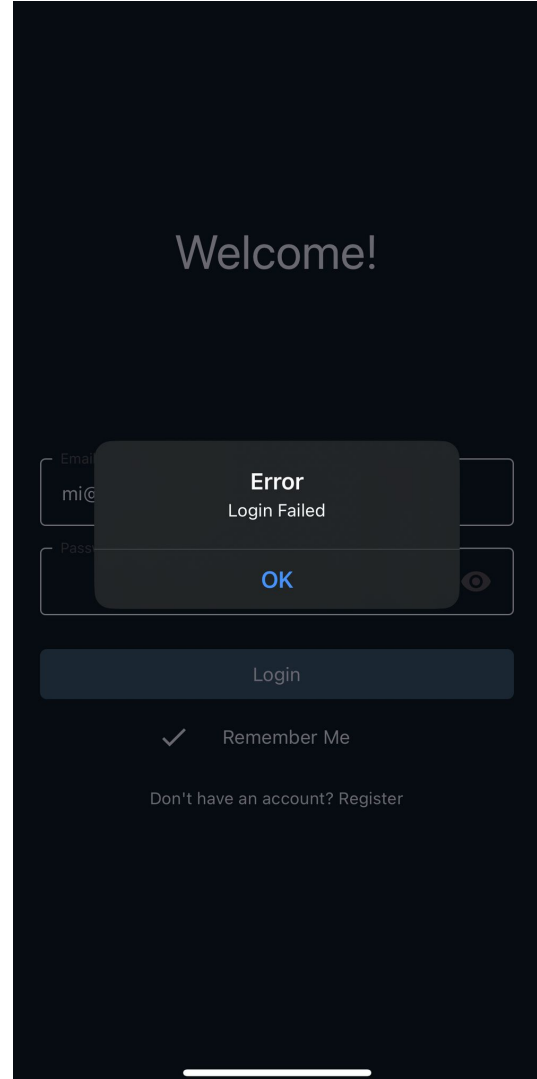


Figure 5.4: Login error feedback for invalid credentials

Backend Security

Passwords are hashed with `bcrypt` before storage to ensure security. The second parameter of `bcrypt`'s hash function (in this case, '10', as shown in Listing 5.1) specifies the salt rounds, determining the computational complexity and thus the strength of the hashing process. On login, passwords are verified and, upon success, a JWT token is issued to maintain a secure session for 24 hours. All communication uses HTTPS, with robust error handling on both frontend and backend to safeguard user data.

Listing 5.1: Node.js Registration Logic

```
const { email, password, name } = req.body;  
const hashedPassword = await bcrypt.hash(password, 10); // 10 = salt rounds
```



```
const newUser = await createUser(email, hashedPassword, name);
res.status(201).json({ message: "User registered successfully!" });
```

The overall authentication flow — from user input to secure access control — is illustrated in Figure 5.5. It highlights key steps such as password hashing, database storage, and JWT issuance.

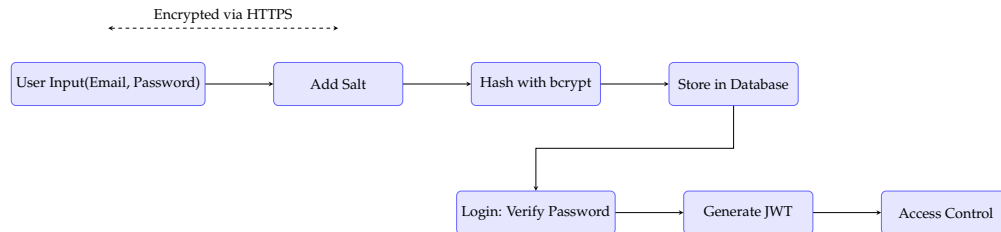
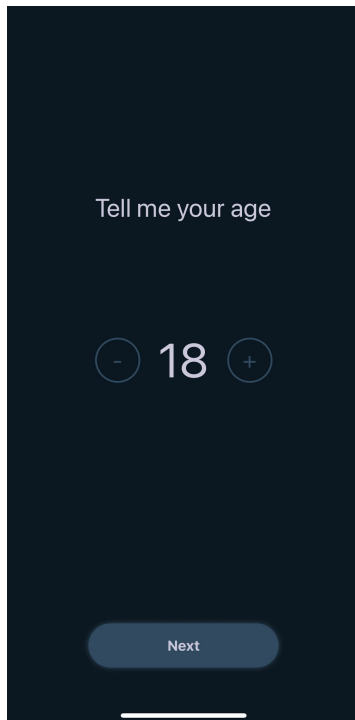


Figure 5.5: Authentication Flow: From User Input to Firebase Access Control with Salting and Hashing

Onboarding

Basal Metabolic Rate (BMR) represents the amount of energy, measured in calories, that a person’s body requires to maintain basic physiological functions at rest, such as breathing and circulation. It serves as the foundation for calculating daily calorie needs, which are then adjusted according to the user’s activity level and fitness goals.[VKS23]

The onboarding process guides first-time users through a series of simple and interactive steps where they provide essential personal data such as age, weight, height, and activity level. Each of these steps is presented through dedicated screens that make data entry intuitive and visually clear, as illustrated in Figures 5.6 to 5.9.



Tell me your age

- 18 +

Next

Figure 5.6: Set Age Screen

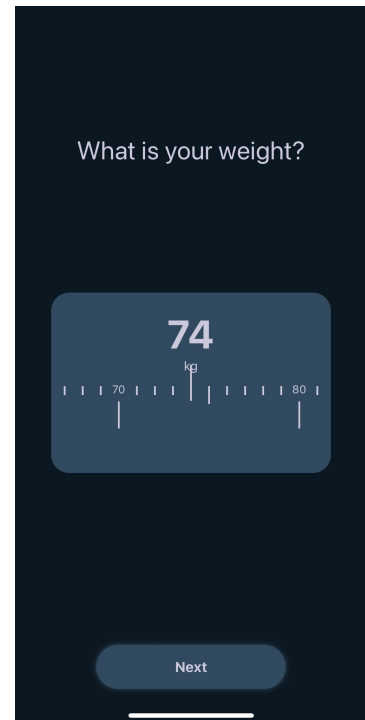


What is your height?

176 cm

Next

Figure 5.7: Set Height Screen

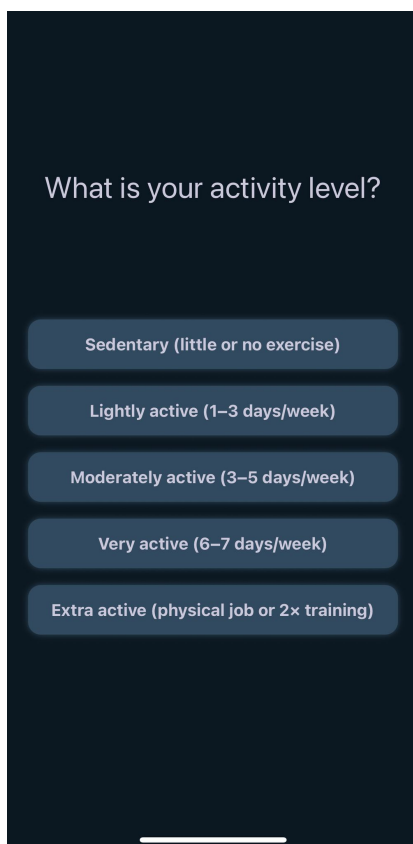


What is your weight?

74 kg

Next

Figure 5.8: Set Weight Screen



What is your activity level?

Sedentary (little or no exercise)

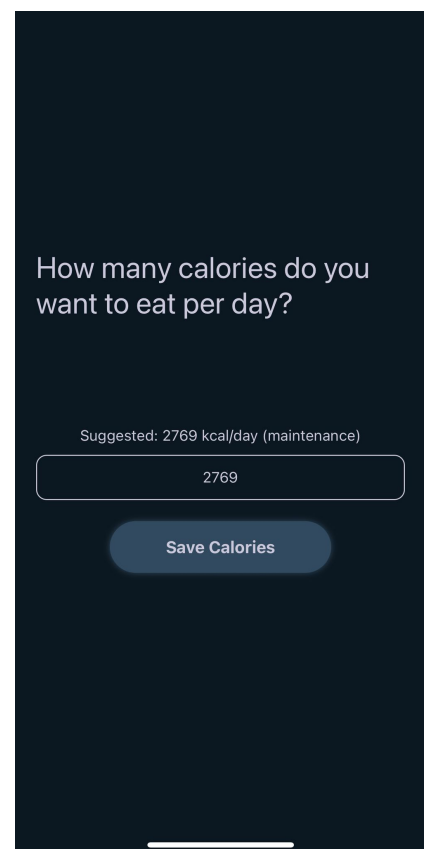
Lightly active (1–3 days/week)

Moderately active (3–5 days/week)

Very active (6–7 days/week)

Extra active (physical job or 2× training)

Figure 5.9: Set Activity Level Screen



How many calories do you want to eat per day?

Suggested: 2769 kcal/day (maintenance)

2769

Save Calories

Figure 5.10: Set Calories Per Day Screen

The process typically begins with selecting the user's age (Figure 5.6), followed by height and weight input (Figures 5.7 and 5.8) using scrollable pickers. Then, users select their activity level (Figure 5.9) from predefined categories ranging from sedentary to highly active. Finally, the app calculates the daily calorie needs and presents the result on a summary screen (Figure 5.10).

This collected data is used to calculate the BMR using the Mifflin-St Jeor formula:

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

The app then adjusts the BMR by an *activity factor* based on the user's reported lifestyle, which scales the daily calorie needs to reflect energy expenditure beyond resting functions. The activity factors applied in the app are as follows:

- Sedentary (little or no exercise): 1.2
- Lightly active (light exercise/sports 1-3 days/week): 1.375
- Moderately active (moderate exercise/sports 3-5 days/week): 1.55
- Very active (hard exercise/sports 6-7 days/week): 1.725
- Extra active (very hard exercise/sports & physical job): 1.9

Thus, the final daily calorie maintenance goal is calculated as:

$$CalorieGoal = BMR \times ActivityFactor$$

This personalized calorie target helps users align their diet and fitness routines with their goals, whether they aim for weight loss, maintenance, or muscle gain.

To persist this data, the app sends the collected user metrics including age, weight, height, activity level, and calculated calorie goal to the backend via an HTTP PATCH request. In a React Native environment, the data update will look like this:

```
const updateUserData = async (email, data) => {
  try {
    await axios.patch(`${API_BASE_URL}/users/${email}`, data);
  } catch (error) {
    console.error("Failed to update user data", error);
  }
};

updateUserData(userEmail, userData);
```

By structuring onboarding in this stepwise manner and using informative screens, the app ensures accurate data collection and fosters user engagement, ultimately enabling personalized health and nutrition guidance tailored to each individual.

Step Tracking via Expo Sensors

We utilize Expo's `Pedometer` sensor to track user steps in real time.

- First, the app checks for pedometer availability using `Pedometer.isAvailableAsync()`.
- If available, it retrieves the number of steps taken within a given day using `Pedometer.getStepCountAsync()`.
- This is done by specifying the start and end of the day.
- A dashboard is presented to users, displaying key daily metrics such as total steps, calories burned, calories consumed, and remaining calories based on dietary input and activity levels.

To enable live updates, we subscribe to step count changes via `Pedometer.watchStepCount()`, accumulating steps as the user moves.

Distance Estimation We estimate the distance traveled by multiplying the total step count by an average step length of 0.762 meters[Wik24]:

$$Distance(meters) = Steps \times 0.762$$

Calories Burned Estimation Calories burned are roughly estimated using the formula:

$$Calories = Steps \times 0.04$$

This assumes approximately 0.04 calories burned per step for an average adult[Sur20].

Data Views and Storage Users can view their activity aggregated daily, weekly, and monthly. Step history and derived metrics are stored in the cloud to enable long-term tracking and synchronization across devices.

Progress Photo Upload Feature

The Wellnest app allows users to periodically upload progress photos to visually monitor changes over time. This feature encourages users to maintain motivation by providing a clear, visual record of their fitness journey.

Users can select images directly from their device’s gallery, with the app first requesting permission to access the media library. Upon selection, images are uploaded securely to a backend server where they are stored with associated timestamps. The photos are then displayed in chronological order within the app, allowing users to easily review their progress.

This feature promotes user engagement by offering a simple and intuitive way to document visual progress, reinforcing consistency through visible tracking, and providing motivational feedback by comparing past and current states. The key components and benefits of this feature are summarized in Table 5.1.

Aspect	Description
Image Selection	Users choose photos from their device gallery to document progress.
Permission Handling	The app requests media library permissions to ensure privacy and security.
Upload Process	Images are uploaded to a remote backend where they are stored with timestamps.
Display	Uploaded images are shown in a gallery view sorted by upload date for easy review.
User Motivation	Visual progress encourages continued engagement and achievement tracking.

Table 5.1: Key Components and Benefits of the Progress Photo Upload Feature

Custom Calorie Counter

- Display the user’s daily calorie budget based on their fitness goal.
- Subtract calories from the total as meals are logged.
- Automatically update remaining calories after each meal entry.

Meal Logging

- Enable manual meal and calorie logging.
- Utilize AI to suggest meals or snacks tailored to remaining daily calories.
- Integrate the Food Ingredients API to fetch nutritional data for logged meals, improving accuracy and ease of use.

Workout and Training Tracker

- Allow logging of workouts including exercises, reps, and weights.

- Enable users to create customized workout plans using Ninja API exercise data.
- Provide a mini logbook to track workout progress and weights lifted over time.
- Support filtering exercises by muscle group and difficulty level, and include a search bar to find specific exercises efficiently.

Local Notifications Engine

To encourage users to maintain daily activity habits, the app includes a local notification system built with `expo-notifications`. These notifications are motivational in tone and appear at randomized intervals.

They are delivered locally on the device, requiring no server infrastructure. This ensures low latency and privacy, while still promoting habit reinforcement.

The notification messages are stored in a JSON file containing various motivational phrases. The system randomly selects messages from this collection to keep the notifications fresh and engaging.

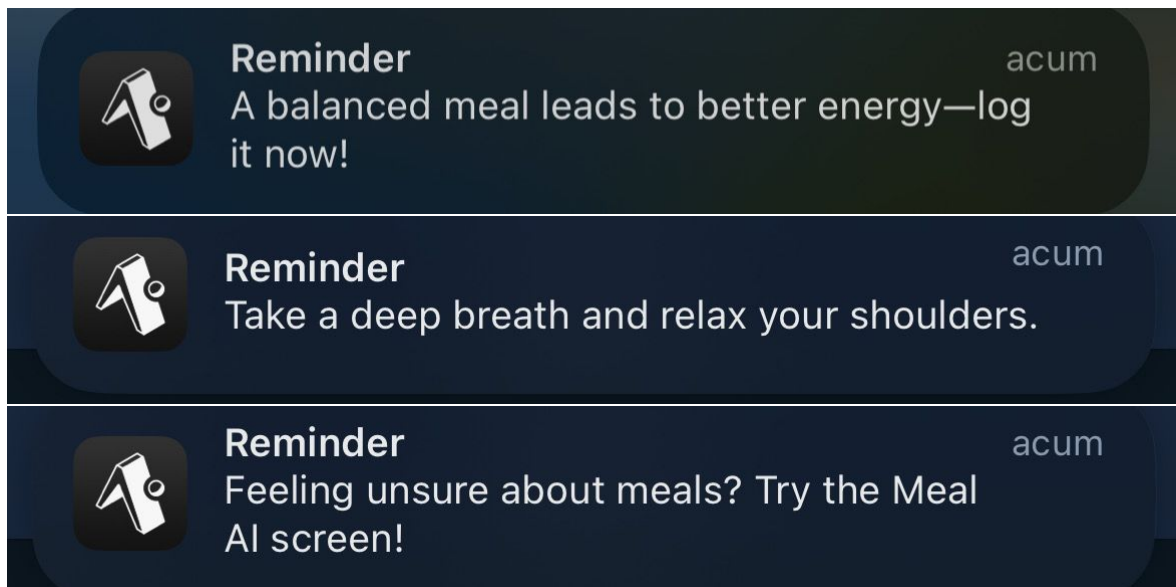


Figure 5.11: Example of notifications

5.3.2 Non-Functional Requirements

Performance

- The application must perform optimally, with fast loading times and responsive UI interactions.

- Background processes, such as step tracking, should not significantly impact battery life.
- Data synchronization between the app and Firebase should be real-time and responsive, with no delays in the user interface.

Platform Compatibility

- The app must be built using React Native to be compatible with both Android and iOS platforms.
- The app's design should ensure that the user interface adapts correctly to different screen sizes and resolutions on both platforms.

Usability

- The user interface should be intuitive and easy to navigate.
- The app should include onboarding steps for new users to get them set up with their health data and fitness goals.
- The app must include helpful tooltips and support documentation for advanced features.

Reliability and Availability

- The app should be highly available.
- The app should handle potential failures gracefully, with clear error messages and recovery options.

5.4 Backend Design and Technologies

5.4.1 System Architecture Overview

The backend of Wellnest is developed using Node.js with Express.js, following a modular and layered architecture that promotes maintainability, scalability, and separation of concerns.

The structure consists of several key components:

- **Models:** Define the core data structures and entities used throughout the application.

- **Controllers:** Implement the business logic and coordinate data manipulation, interacting with the database asynchronously to perform create, read, update, and delete (CRUD) operations.
- **Routes:** Define the API endpoints and map incoming HTTP requests to corresponding controller functions. Middleware such as authentication is applied at this level to secure protected resources.
- **Services:** Provide reusable and modular functionality that supports controllers by encapsulating complex logic or external API interactions.
- **Server Initialization:** The main server file initializes the Express application, configures middleware, and mounts the route handlers to respond to client requests.

The backend leverages Firebase Firestore for data persistence, offering a scalable and real-time database solution that ensures secure management and synchronization of user data.

This architecture enables clear code organization, facilitates asynchronous operations essential for a responsive user experience, and allows for straightforward extension as the system evolves.

The backend architecture is illustrated in Figure 5.12, demonstrating how each component interacts within the system.

5.4.2 Technology Stack and Integrations

The backend of Wellnest is built using **Node.js** with the **Express.js** framework, following a modular and layered architecture. Express.js facilitates the creation of RESTful APIs that handle asynchronous operations efficiently, enabling responsive communication between the mobile frontend and backend services.

The application uses standard HTTP methods to manage user interactions. **GET** requests retrieve data such as user activity logs, meal records, and progress summaries. **POST** requests allow users to submit new information, like logging meals, recording workouts, or creating daily entries. **PATCH** requests are used to update existing resources, such as modifying meal details or user preferences.

Each API response includes appropriate HTTP status codes: **200 OK** for successful retrieval or updates, **201 Created** when new resources are added, **400 Bad Request** for invalid input data, **401 Unauthorized** if authentication fails, **404 Not Found** when requested resources do not exist, and **500 Internal Server Error** to indicate server-side processing issues.

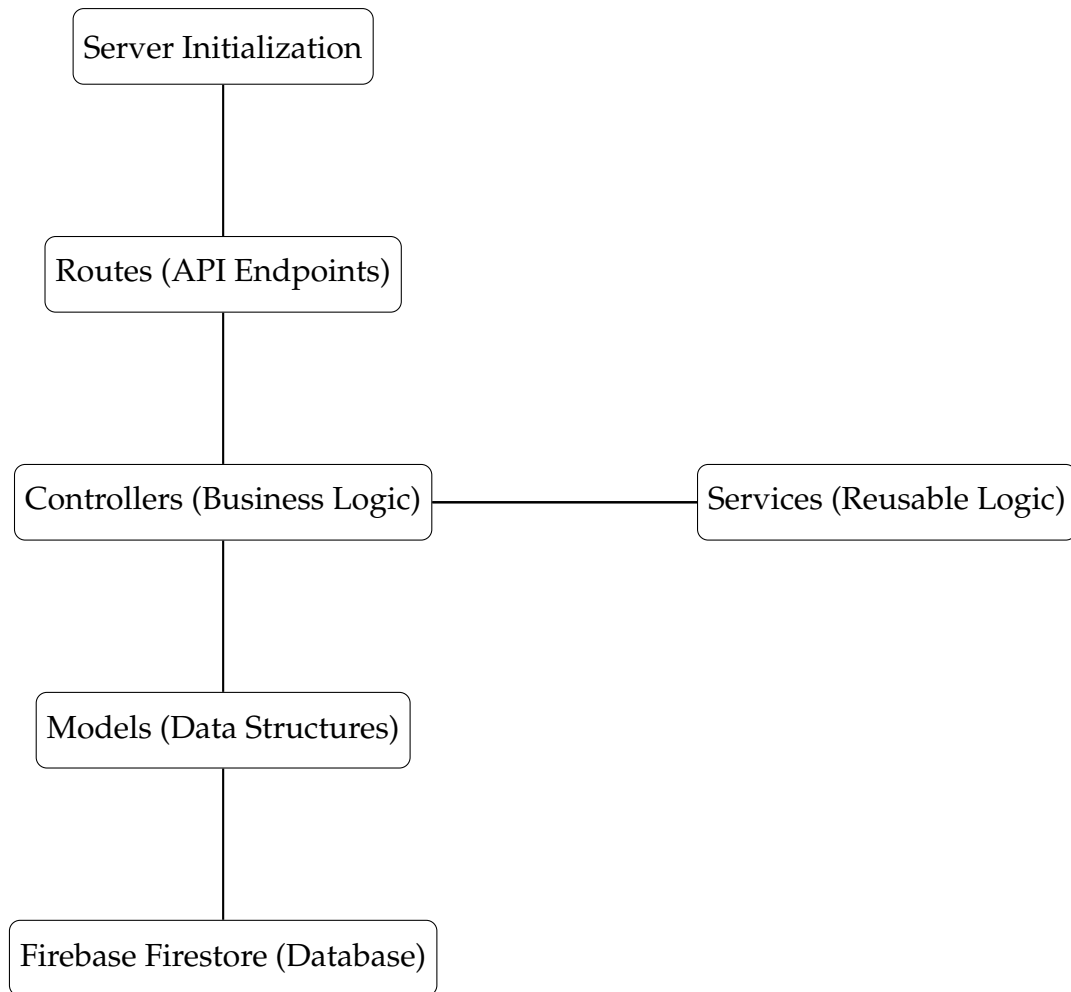


Figure 5.12: Overview of the backend system architecture

This well-organized backend structure using Express.js supports maintainability, scalability, and secure data handling, all essential for delivering a smooth and personalized user experience in Wellnest.

Firebase Firestore – Data Storage and User Authentication: Firestore serves as the centralized data platform for Wellnest, managing both user authentication (via JWT tokens) and real-time data storage. Two core collections drive the system:

- **Users:** Stores profiles, activity data, goals, streaks, and point totals.
- **Activity Logs:** Records detailed information about meals, workouts, and step counts.

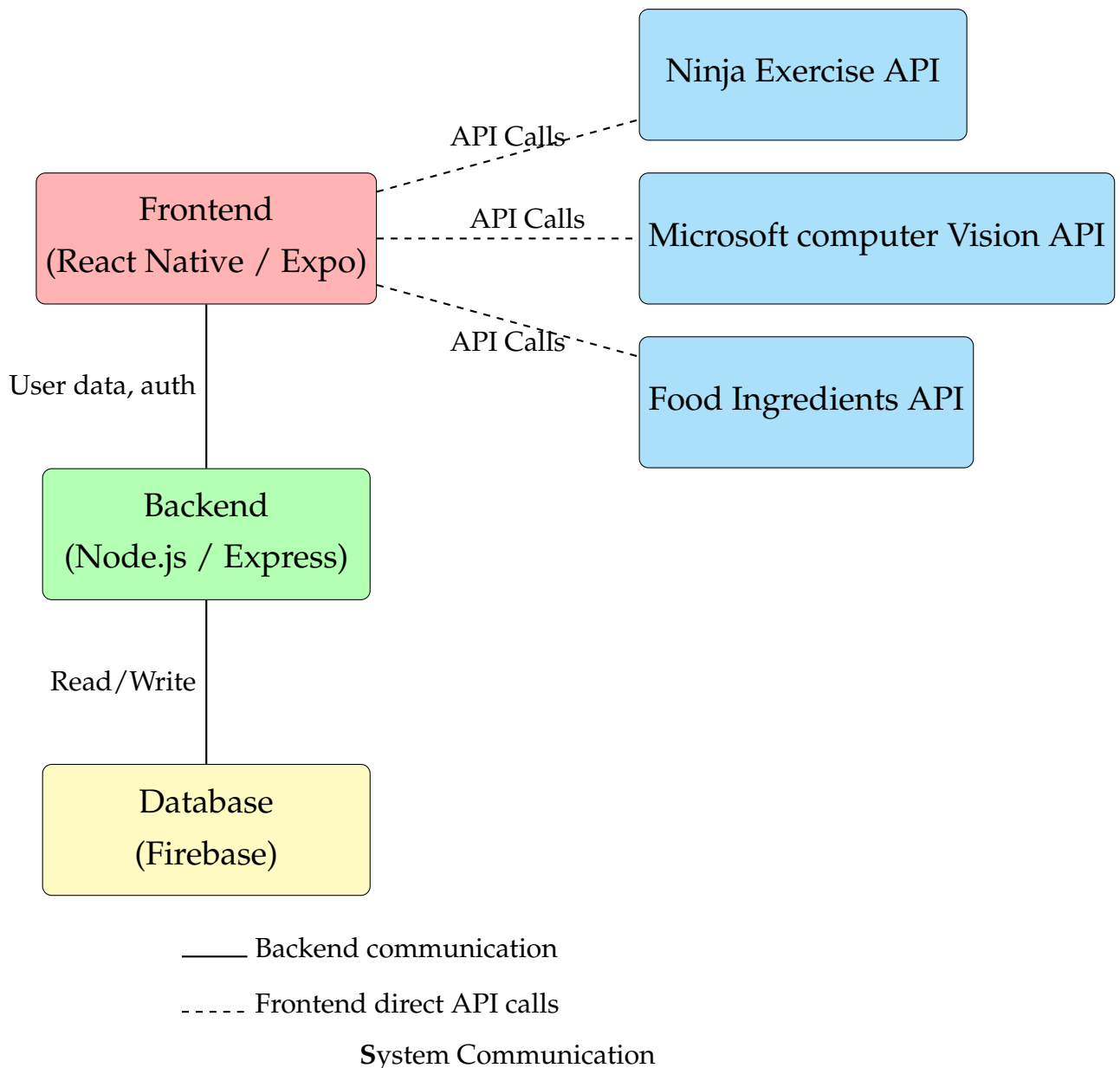
This setup allows seamless data syncing and minimizes latency for user-facing updates.

Microsoft Computer Vision API – Ingredient Recognition and Analysis: Wellnest uses the Microsoft Computer Vision API to extract text from photos of food labels or meals. The system then analyzes the identified ingredients, providing users with

insights on why certain ingredients are beneficial or harmful, helping to make informed dietary choices.

Ninja Exercise API – Workout Content Integration: Wellnest uses the Ninja Exercise API to fetch a wide range of exercises categorized by type, muscle group, and difficulty. This allows the system to dynamically assemble workout plans based on user preferences and fitness levels.

Gamification Engine – Scoring and Motivation: The backend includes a built-in scoring and achievement module that awards users for positive actions like reaching step targets or completing daily challenges. These points feed into a leveling system, which is tracked in Firestore and surfaced to the user via the mobile dashboard. Streaks, badges, and reward unlocks are also handled by this logic.



This architecture ensures efficient, secure communication by separating frontend, backend, and external APIs. It reduces server load, improves responsiveness, and allows each component to scale and evolve independently—providing a flexible, robust foundation for Wellnest’s future growth.

5.4.3 AI-Powered Personalized Recommendations

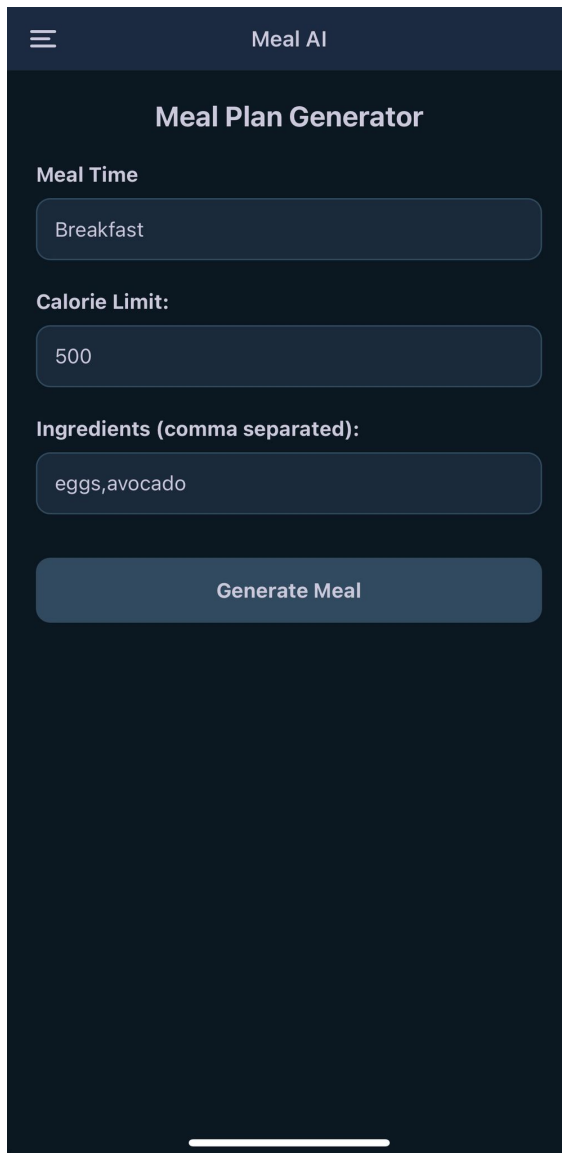
Wellnest enhances user experience by integrating advanced AI capabilities to generate personalized meal recipes and workout plans. These features are powered by a state-of-the-art language model accessible via the Groq SDK, which allows the backend to interact with AI in a controlled and precise manner.

Tailored Meal Plan Generation

The meal recommendation system uses AI to create healthy, easy-to-prepare meal recipes that respect each user’s unique dietary goals and ingredient availability. The AI is guided by a meticulously designed prompt that specifies the meal time (e.g., breakfast, lunch), lists the ingredients on hand, and imposes a strict calorie limit.

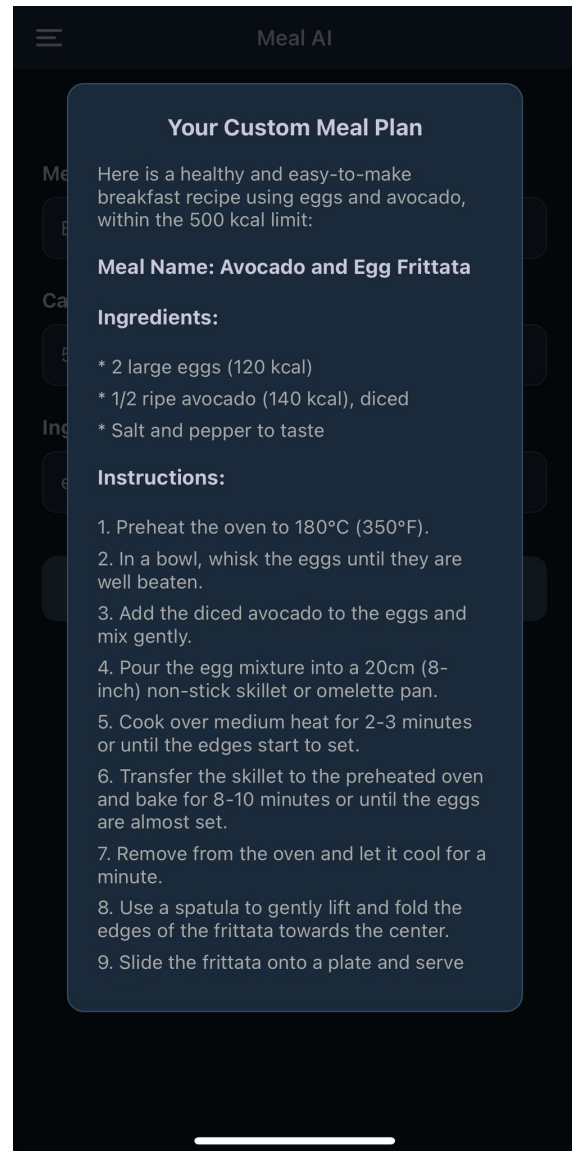
This prompt design ensures that the AI focuses exclusively on generating a single recipe formatted in a user-friendly way. The response includes:

- A clear meal name,
- An ingredient list with precise quantities,
- Step-by-step cooking instructions,
- The total calorie count for the meal.



The screenshot shows the 'Meal AI' app interface. At the top, there's a hamburger menu icon and the text 'Meal AI'. Below this is the 'Meal Plan Generator' section. It contains three input fields: 'Meal Time' with 'Breakfast' entered, 'Calorie Limit:' with '500' entered, and 'Ingredients (comma separated):' with 'eggs,avocado' entered. At the bottom of this section is a large blue button labeled 'Generate Meal'.

Figure 5.13: User input for AI meal recommendation



The screenshot shows the 'Meal AI' app interface displaying the generated meal plan. At the top, there's a hamburger menu icon and the text 'Meal AI'. Below this is a dark blue box titled 'Your Custom Meal Plan'. Inside the box, it says: 'Here is a healthy and easy-to-make breakfast recipe using eggs and avocado, within the 500 kcal limit:'. Below this is the 'Meal Name: Avocado and Egg Frittata'. Then, under 'Ingredients:', it lists: '* 2 large eggs (120 kcal)', '* 1/2 ripe avocado (140 kcal), diced', and '* Salt and pepper to taste'. Finally, under 'Instructions:', it lists nine steps: 1. Preheat the oven to 180°C (350°F). 2. In a bowl, whisk the eggs until they are well beaten. 3. Add the diced avocado to the eggs and mix gently. 4. Pour the egg mixture into a 20cm (8-inch) non-stick skillet or omelette pan. 5. Cook over medium heat for 2-3 minutes or until the edges start to set. 6. Transfer the skillet to the preheated oven and bake for 8-10 minutes or until the eggs are almost set. 7. Remove from the oven and let it cool for a minute. 8. Use a spatula to gently lift and fold the edges of the frittata towards the center. 9. Slide the frittata onto a plate and serve.

Figure 5.14: Generated meal suggestion output

By structuring the prompt with detailed instructions and a specific format, Wellnest guarantees consistency and relevance in the generated recipes, making them practical and easy for users to follow.

Customized Workout Plan Creation

Similarly, the workout recommendation feature leverages AI to produce balanced, effective exercise plans tailored to the user's preferences. Users specify the number of workout days, targeted muscle groups, and optional notes or constraints.

The AI prompt instructs the model to:

- Develop a strict workout plan covering only the requested muscle groups,

- Include an appropriate number of exercises per day to fully engage the muscles,
- Structure each day with warm-ups, multiple exercises (with sets, reps, and rest times), and cool-downs,
- Avoid exercises unrelated to the selected muscle groups,
- Present the plan in a clear, standardized format.

This carefully crafted prompt ensures the AI-generated workouts are precise, focused, and practical, empowering users to follow routines aligned perfectly with their fitness goals.

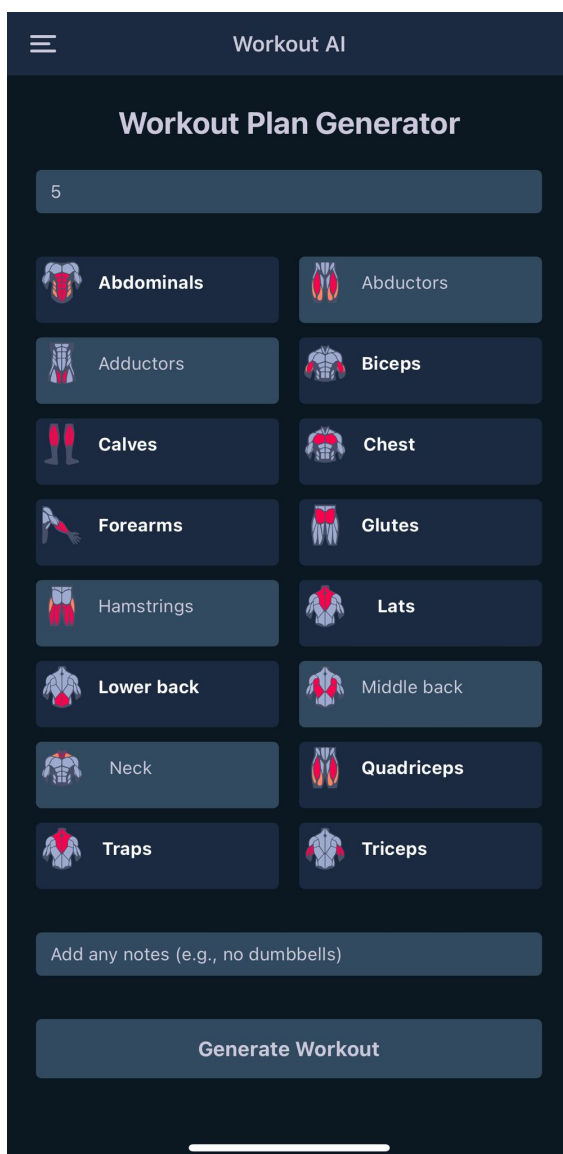


Figure 5.15: User input for AI workout recommendation

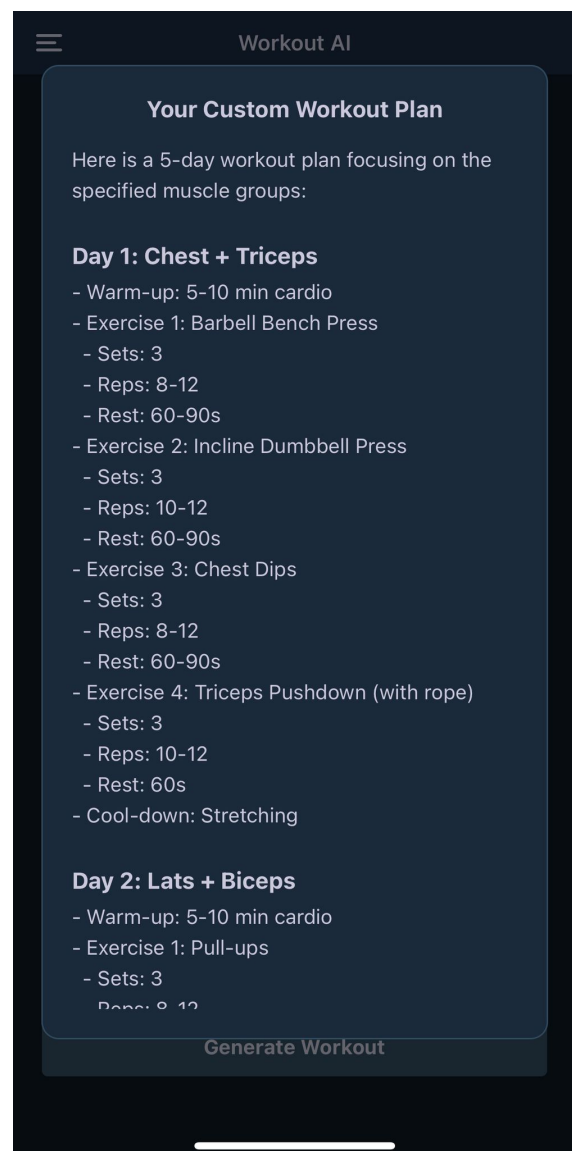


Figure 5.16: Generated workout suggestion output

Benefits of AI Integration

The strategic use of AI through well-constructed prompts provides several advantages:

- **Personalization:** Each recommendation is uniquely tailored to individual user data and preferences.
- **Consistency:** Structured prompt formatting results in uniform outputs that are easy to parse and present within the app.
- **Efficiency:** Users receive instant, actionable plans without needing to search or create them manually.
- **Scalability:** The backend can easily extend AI-powered features to new domains by designing appropriate prompts.

Incorporating this AI-driven approach transforms Wellnest into an intelligent wellness assistant, offering users meaningful guidance that adapts dynamically to their lifestyle and goals.

5.5 Security and Privacy

5.5.1 User Data Protection

To ensure the safety of sensitive user data such as email addresses, fitness logs, and personal metrics (e.g., weight, goals, achievements), Wellnest employs multiple layers of protection. These include client-side validation, secure database rules in Firebase, and access controls that restrict data visibility to authenticated users only.

5.5.2 Data Storage and Encryption

All sensitive user data is securely stored in Firebase's real-time database. While Firebase handles encryption of data at rest and in transit by default, Wellnest also applies field-level encryption for select data categories before writing them to the database.

In addition, Firebase Cloud Firestore's built-in security rules are leveraged to prevent unauthorized reads or writes based on user authentication status and ownership of data.

5.5.3 Secure Communication Channels

All communication between the Wellnest application and backend services occurs over HTTPS, ensuring end-to-end encryption. This protects against man-in-the-middle attacks and ensures the confidentiality and integrity of data in transit.

5.5.4 Third-Party Services and API Keys

Wellnest integrates third-party services such as Firebase Analytics and Azure OCR for specific functionality. All third-party interactions are governed by the principle of least privilege — only the required data is sent, and no sensitive user data is ever exposed to external systems beyond what is strictly necessary.

API keys and service credentials are stored in environment variables and are excluded from version control via `‘.gitignore’` to avoid accidental exposure.

5.5.5 Error Handling and Fail-Safe Design

Robust error handling mechanisms are in place throughout the app to prevent sensitive information leakage. Error responses never include internal exception traces or user data. When a failure occurs, user-friendly feedback is shown, and fallback mechanisms ensure the app remains stable.

5.5.6 Privacy by Design

Wellnest adopts a privacy-by-design approach, minimizing the collection and retention of personal data. No user data is shared or sold, and all storage is done solely for the purpose of enhancing the in-app experience. Users retain full control over their data, with the ability to request deletion at any time.

caption

Chapter 6

Technical and Architectural solutions

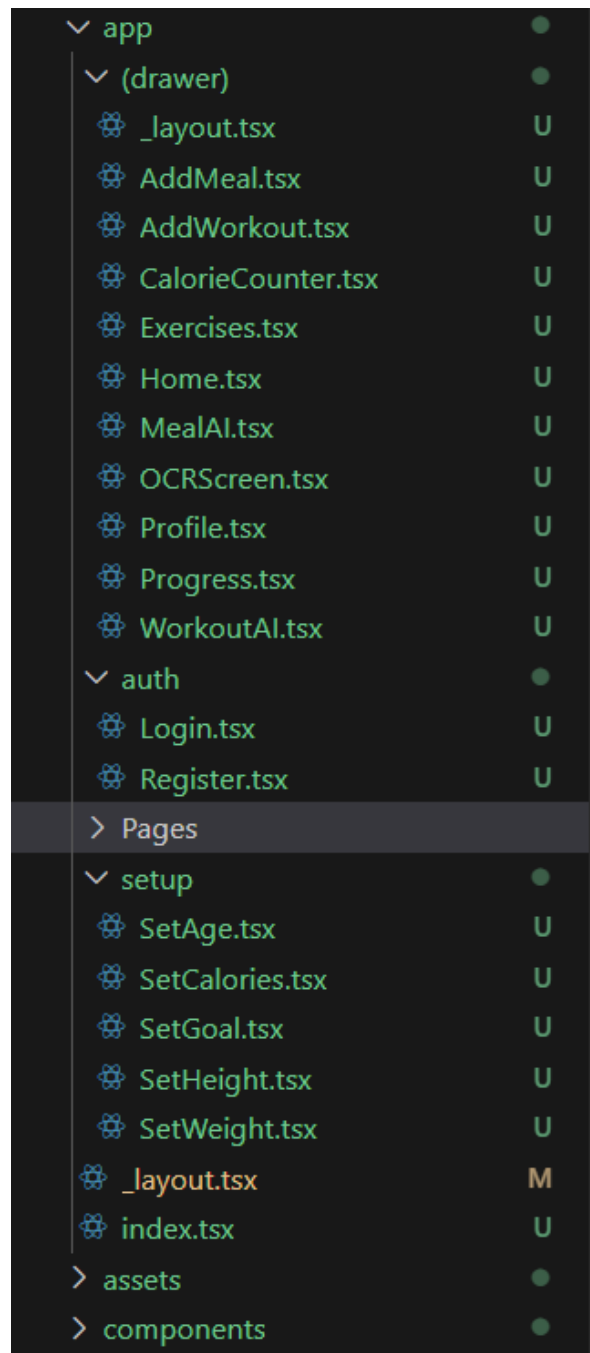
This chapter presents the architectural choices, design structure, and core technical decisions behind the Wellnest App. The focus is on clarity, efficiency, and user-centric functionality.

Every decision — from the use of Expo and React Native to reusable components and motivational notifications — was made to ensure the app remains maintainable, responsive, and engaging for users tracking their daily fitness activity. The design encourages healthy habits through thoughtful UX strategies and gamified features.

6.1 Project Structure Philosophy

The app uses a feature-first modular structure. Instead of organizing files strictly by type, it groups them by functionality and purpose. This design makes the codebase more scalable, easier to navigate, and better aligned with the user interface.

It encourages reuse of logic and visual elements within specific feature areas while keeping related files in proximity. Developers can focus on specific features with less context-switching, which accelerates debugging and iteration.



Project file structure

6.2 Component Architecture and Reusability

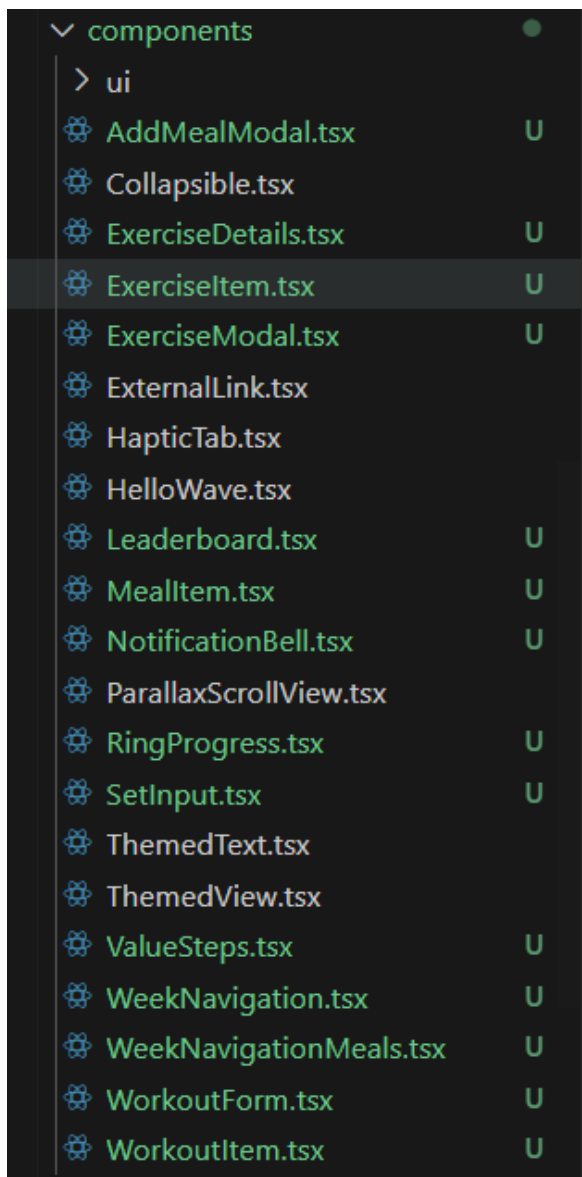
Reusable components form the backbone of the UI. These components are responsible for maintaining a consistent look and feel, simplifying updates, and isolating logic where necessary. They are built with clear inputs via props and avoid unnecessary state management.

This pattern makes components easier to test and scale. It also simplifies onboarding for other developers, as the structure is clear and the purpose of each component is specific. A summary of the core UI components and their purposes

is provided in Table 6.1.

Component	Purpose
RingProgress	Visualizes progress toward daily activity goals using a circular animated chart.
Value	Displays labeled metrics like step count, calories burned, or distance traveled. Used on multiple screens.
ExerciseModal	A modal interface for logging exercise sets and reps. Built for extensibility.
CustomDrawer	The custom drawer navigator with dark-mode support and user-friendly layout. Includes quick access to the profile screen.

Table 6.1: Summary of Key UI Components



Visuals of reusable UI components

6.3 Framework and Tooling Choices

The app is built using React Native with Expo. This decision enables fast development and cross-platform deployment from a single codebase. React Native allows the use of native performance features while still writing code in JavaScript or TypeScript.

Expo simplifies the process of integrating device capabilities (like motion sensors and notifications) and removes the need for native configuration for most features. It also speeds up testing, especially in solo or small-team development environments.

6.4 Custom Hooks for Data Logic

```
const startPedometer = async () => {
  const available = await Pedometer.isAvailableAsync();
  setIsAvailable(available);
  if (!available) {
    console.log('Pedometer not available');
    return;
  }

  // Get step count for the given date
  const startOfDay = new Date(date);
  startOfDay.setHours(0, 0, 0, 0);
  const endOfDay = new Date(date);
  endOfDay.setHours(23, 59, 59, 999);

  const { steps } = await Pedometer.getStepCountAsync(startOfDay, endOfDay);
  setSteps(steps || 0);
  setDistance((steps || 0) * 0.762); // Approximate: Avg step length
```

And use it when we take the data:

```
const { steps, calories, distance } = useHealthData(date);
```

The custom hook `useHealthData(date)` is responsible for retrieving, calculating, and formatting daily fitness data like step count, distance, and calories. This hook encapsulates all logic related to health metrics, allowing UI code to remain simple and declarative.

By abstracting data logic into a hook, multiple screens can use consistent data without duplicating logic. This pattern keeps screens clean and modular, making the app more maintainable and less prone to bugs.

6.5 Gamification and Feedback System

A lightweight point-based gamification system encourages users to stay consistent with their fitness goals. Each action earns points:

- +10 points for logging a new exercise
- +5 points per set logged in the exercise modal

Points accumulate and are displayed on the profile screen, where users can also view a basic leaderboard showing their ranking. This leaderboard adds a social or competitive element, making the experience more engaging and goal-driven.

The point system is deliberately kept simple to reduce cognitive load and promote quick daily use.

6.6 Design Strategy and UX Focus

User interface design is centered around speed, clarity, and ease of use. The visual language uses dark mode, large fonts, and interactive charts to present data in a motivational and accessible way.

The UI is optimized for:

- Quick logging of activities in under a minute
- Immediate feedback via visuals (progress rings, metrics)
- Usability during various times of day (dark theme preferred by night users)

Navigation is intuitive, relying on custom drawers and touch-friendly elements to create a frictionless experience.

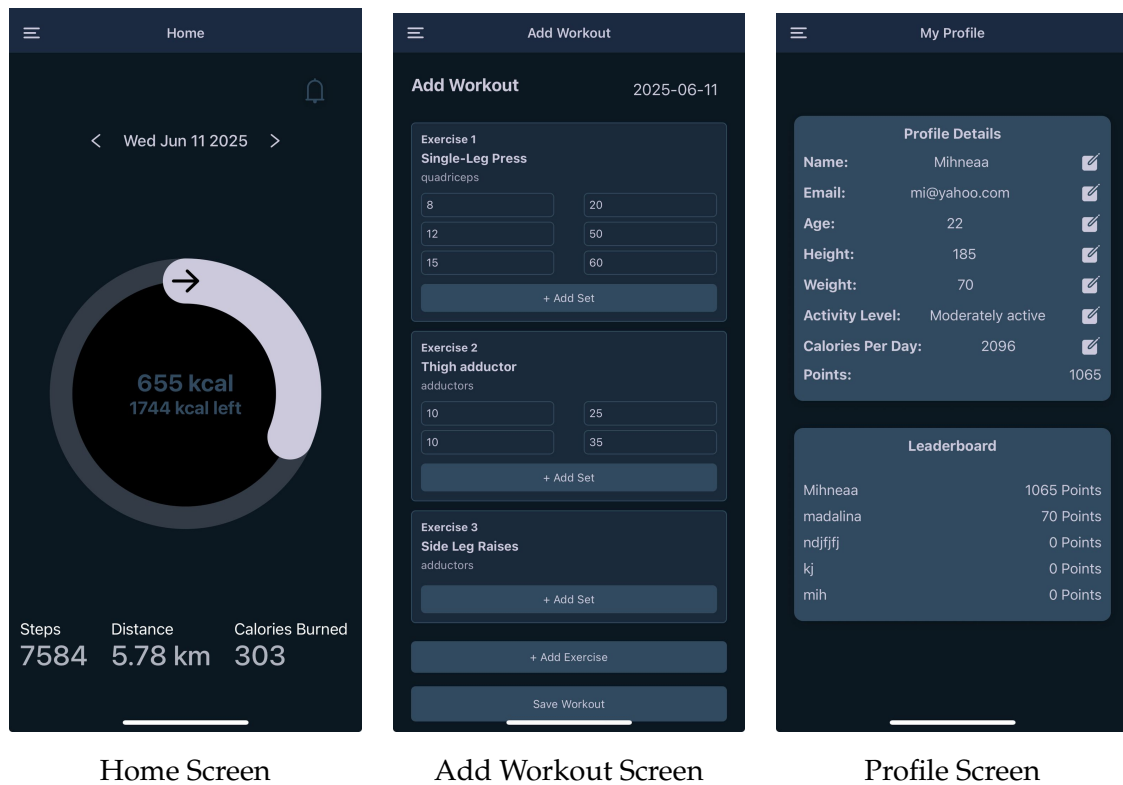


Figure 6.1: Main App Screens Displayed Side-by-Side

6.7 Summary

This architecture prioritizes modularity, reusability, and a user-centered design. The use of Expo and React Native offers rapid development, while custom components and hooks provide long-term maintainability.

Gamification, motivational notifications, and a streamlined UI create an experience that is both practical and engaging. Whether the app is used personally or developed further into a commercial product, its design foundations are scalable and flexible enough to support future growth.

Chapter 7

OCR Ingredient Analysis Functionality

7.1 Overview

The OCR Ingredient Analysis feature enables users to analyze food ingredient labels by extracting and interpreting text from images they upload. Built with React Native and Expo, this module leverages the **Azure Computer Vision OCR API** for text recognition and uses a local JSON database to classify ingredients based on their health impact. This functionality supports users aiming to make informed dietary choices by providing clear visual feedback on ingredients.

7.2 Frontend Design

The user interface guides users seamlessly through selecting an image, extracting text, and reviewing ingredient analysis results. The key components and flows include:

7.2.1 Image Selection

Users start by choosing an image of a food product's ingredient label from their device's photo library. The app uses the `expo-image-picker` library, which provides a native interface for accessing the device's media gallery. This library supports image cropping and editing, allowing users to focus on the ingredient list area, which improves the accuracy of subsequent OCR processing.

Upon selecting an image, the app updates its state with the image's URI, resets any previous OCR results, and prepares for a new extraction process. This intuitive flow ensures users can easily select and manage images without confusion or errors.

To improve the user experience, the app displays the chosen image prominently on the screen, confirming to users that their selection was successful and allowing them to visually verify the input before proceeding.

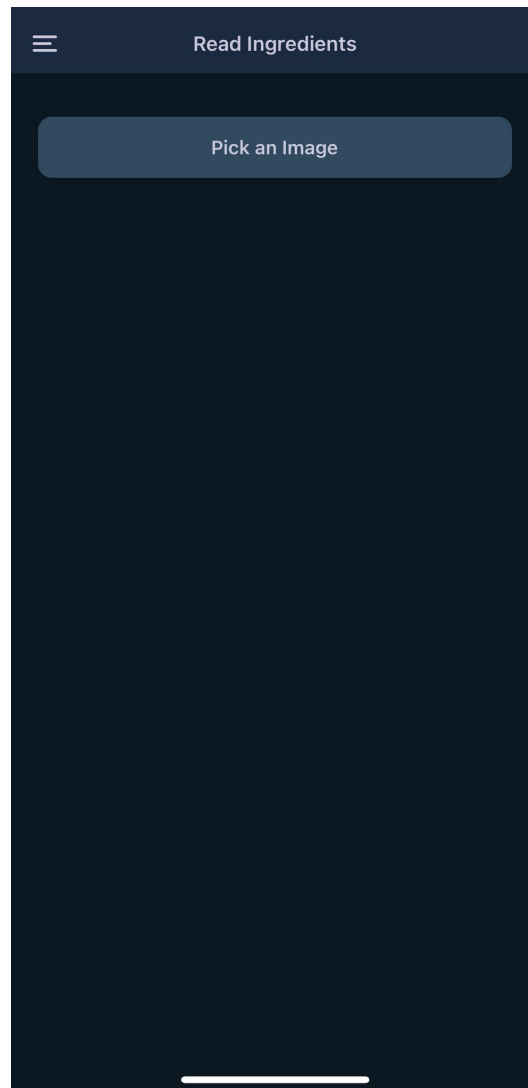


Figure 7.1: Initial OCR screen

7.2.2 Text Extraction

After selecting the image, users initiate text extraction by tapping the "Extract Text" button. The app converts the selected image into a binary blob and prepares a request compatible with the **Azure OCR API**. The Azure API accepts image data as a binary stream or via a URL; the app uses the binary method for enhanced privacy and control.

Once the image data is prepared, the app submits a POST request to Azure's Computer Vision endpoint, including the appropriate subscription key and region-specific endpoint in the request headers. During this asynchronous operation, a

loading indicator is displayed to inform users that the extraction is in progress, preventing duplicate submissions and improving perceived responsiveness.

Upon receiving a response, the app aggregates all detected lines of text into a single string. It then attempts to isolate the ingredients list using keyword-based heuristics. If the OCR fails or returns no usable text, the app gracefully notifies the user with an appropriate message, maintaining a smooth user experience even in adverse conditions.

7.2.3 Ingredient Parsing and Classification

The app parses the extracted text to isolate the ingredients list, cleaning and splitting the text into individual ingredient names. These names are then matched against a locally stored ingredient database, classifying each ingredient as *good*, *bad*, *neutral*, or *unknown*.

An example ingredient entry from the JSON database is as follows:

```
{
  "name": "sugar",
  "label": "bad",
  "reason": "Excessive sugar intake is linked to obesity, type 2 diabetes,
and heart disease."
}
```

The parsing and analysis process can be briefly summarized with the following snippet:

```
const analyzeIngredients = (names) => {
  return names.map(name => {
    const match = ingredientDB.find(i => i.name.toLowerCase() === name.toLowerCase())
    return {
      name,
      label: match?.label || 'unknown',
      reason: match?.reason || 'No information available.',
    };
  });
};
```

7.2.4 Results Display

Ingredients are presented as color-coded pills: green for good, red for bad, gray for neutral, and dark gray for unknown. Users can tap on any ingredient pill to open a modal dialog explaining the rationale behind its classification.

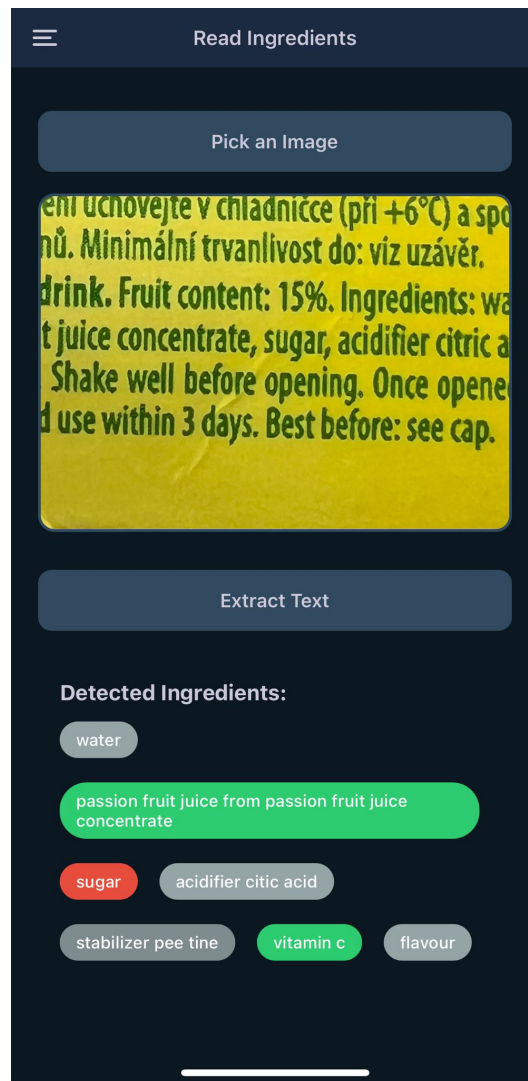


Figure 7.2: Result of the text extraction

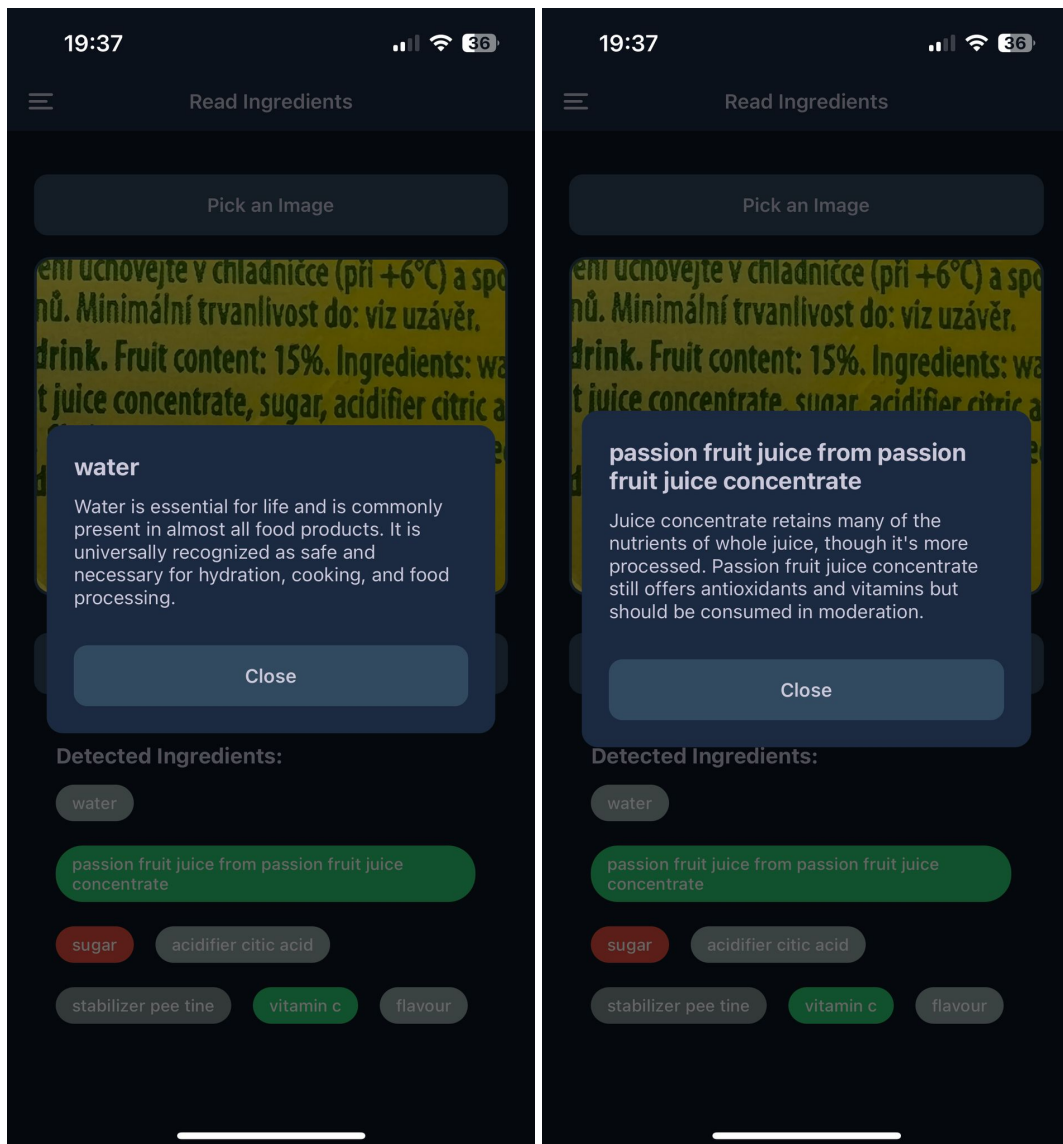


Figure 7.3: Reason of the ingredients

7.3 Backend Integration

The feature does not rely on a custom backend server. Instead, it uses:

- **Azure Computer Vision OCR API:** Accepts binary image data and returns structured OCR results, including regions, lines, and words. This API provides high accuracy and performance with support for printed and handwritten text.
- **Local JSON Database:** Stores ingredient health classifications, allowing offline analysis without extra network requests.

This design simplifies deployment, enhances privacy, and reduces latency while leveraging Azure's enterprise-grade text recognition service.

7.4 Security Measures

- **API Key Management:** The Azure subscription key is securely stored in environment variables, preventing exposure in source code.
- **Robust Error Handling:** Network requests and parsing routines are protected with error handling to avoid app crashes and provide user-friendly messages in case of failures.
- **Data Privacy:** Only the necessary image data is sent to the Azure OCR API; no personal user information is transmitted.

7.5 Testing and Reliability

The feature was tested extensively with several scenarios, including:

- Attempting OCR without selecting an image (process safely blocked).
- Handling blurry or low-quality images resulting in partial or no text extraction.
- Parsing partial ingredient lists when labels are partially visible.
- Labeling unknown ingredients gracefully when absent from the database.
- Managing API response errors with clear fallback messaging.

One of these scenarios and the app's handling of them are illustrated in Photo 7.4.

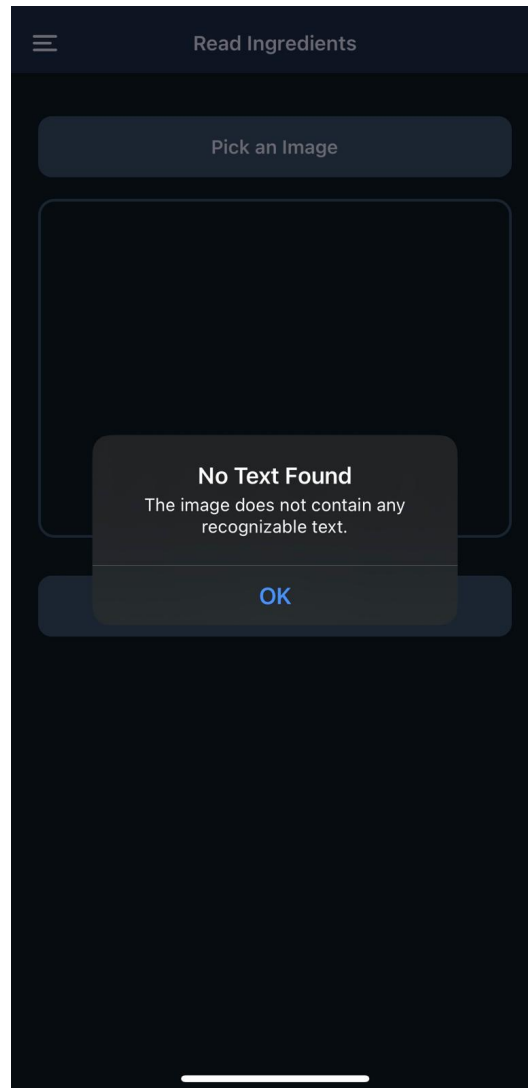


Figure 7.4: App screen during OCR testing showing error handling and ingredient recognition

Chapter 8

Conclusion

This thesis presented the design, implementation, and evaluation of Wellnest, a cross-platform mobile application dedicated to promoting holistic health and fitness. By leveraging a modular client-server architecture with a React Native frontend and FastAPI backend, Wellnest seamlessly integrates real-time step tracking, AI-powered personalized meal and workout recommendations, and gamification elements to enhance user motivation and engagement.

The application combines advanced prompt engineering techniques with sensor data and third-party APIs to deliver adaptive and context-aware wellness guidance. Key features such as nutritional label scanning, asynchronous backend processing, and secure user authentication via JWT contribute to a robust and scalable system.

Moreover, the inclusion of behavioral foundations and gamification strategies grounded in Self-Determination Theory and the Fogg Behavior Model supports sustained user adherence to healthy habits. Performance optimizations and security best practices ensure a smooth and trustworthy user experience across multiple platforms.

Overall, Wellnest exemplifies a modern approach to mobile health applications by merging cutting-edge AI, sensor technology, and motivational design to empower users in achieving long-term wellness goals. There are several opportunities to further enhance Wellnest in future iterations. Expanding the workout builder with video tutorials could better support beginners. Integrating social features such as group challenges and community support may increase user engagement and accountability. Additionally, adding functionality to scan barcodes and automatically detect product descriptions would streamline nutrition tracking and improve user convenience. Finally, extending compatibility with wearable devices and exploring offline functionality can broaden the app's accessibility and convenience.

Bibliography

- [BEN21] Toras Batubara, Syahril Efendi, and Erna Nababan. Analysis performance bcrypt algorithm to improve password security from brute force. *Journal of Physics: Conference Series*, 1811:012129, 03 2021.
- [BTLC17] David R. Jr Bassett, Lauren P. Toth, Scott R. LaMunion, and Scott E. Crouter. Step counting: A review of measurement considerations and health-related applications. *Sports Medicine*, 47(7):1303–1315, 2017.
- [CGd21] Thomas C G and Jayanthila devi. A study and overview of the mobile app development industry. *International Journal of Applied Engineering and Management Letters*, pages 115–130, 06 2021.
- [DDKN11] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: Defining gamification. volume 11, pages 9–15, 09 2011.
- [Eur25] Eurostat. Overweight and obesity - bmi statistics, 2025.
- [Fog09] B.J. Fogg. Fogg behavior model, 2009.
- [Kha20] Mohit Khare. Json web token (jwt) - an introduction. 2020.
- [KHJK16] Jan Kubovy, Christian Huber, Markus Jäger, and Josef Küng. A secure token-based communication for authentication and authorization servers. pages 237–250, 11 2016.
- [MFH⁺24] Alexandre Mazéas, Cyril Forestier, Guillaume Harel, Martine Duclos, and Aïna Chalabaev. The impact of a gamified intervention on daily steps in real-life conditions: Retrospective analysis of 4800 individuals. *Journal of Medical Internet Research*, 26:e47116, 2024. Published August 12, 2024.
- [Mic25] Microsoft. Azure ai vision with ocr and ai, 2025.

- [MM23] William K. McHenry and Erin E. Makarius. Understanding gamification experiences with the benefits dependency network lens. *Computers and Education Open*, 4:100123, 2023.
- [NC12] Anirudh Nagesh and Carlos Caicedo. Cross-platform mobile application development. 01 2012.
- [Nin25] API Ninjas. Exercises api, 2025.
- [PW24] Jiahao Pan and Shutao Wei. Accuracy and reliability of accelerometer-based pedometers in step counts during walking, running, and stair climbing in different locations of attachment. *Scientific Reports*, 14(1):27761, 2024.
- [RPMK24] Manish Rana, Ayush Pandey, Ankit Mishra, and Vishal Kandau. International journal on recent and innovation trends in computing and communication enhancing data security: A comprehensive study on the efficacy of json web token (jwt) and hmac sha-256 algorithm for web application security. *International Journal on Recent and Innovation Trends in Computing and Communication*, 11:4409–4416, 09 2024.
- [Sof23] Software AG. Jwt use case workflow — webmethods api gateway documentation. 2023.
- [Sur20] United States Geological Survey. Implications of flume slope on discharge estimates for 0.762-meter h flumes used at the edge of field. Technical report, USGS Publications Warehouse, 2020.
- [Tec24] Technobrain. Mobile app architecture: Everything you need to know. 2024.
- [VKS23] Natasha Verma, S. Senthil Kumar, and Anjali Suresh. An evaluation of basal metabolic rate among healthy individuals — a cross-sectional study. *Bulletin of Faculty of Physical Therapy*, 28(1):26, 2023.
- [Wik24] Wikipedia contributors. Effect of gait parameters on energetic cost, 2024.
- [Wor21] Wordfence. How passwords work and how they are cracked. 2021.