**IoT Development CA1: PillPal**

**By Pixel Health**

|  |
| --- |
| **TEAM MEMBERS** |

[SOFIA]: Database, Backend, Hardware

[IKER]: Hardware, Backend, Database

[FAVOUR]: UI/UX Design, Frontend, Testing

[IKRAM]: Frontend, Hardware, GitHub

|  |
| --- |
| **IDEA OVERVIEW** |

|  |  |
| --- | --- |
| **[IKRAM]** 15/09/25 | Create idea document in Notion  Add an initial placeholder for the project idea document to establish a  shared location for concept notes and future architecture details. This  commit creates the file only; content will be added in subsequent edits.  Rationale: capture the artifact early so the team can begin structured  iteration and maintain a clean history from the first change. |
| **[IKRAM]** 17/09/25 | Add detailed concept options and target users  Populate the idea document with four candidate concepts, including  target users and proposed input/output modalities. This clarifies scope  and accessibility goals ahead of prototyping.  Motivation: document the candidate directions and accessibility  considerations to inform selection and risk assessment.  <https://www.notion.so/Ideas-26fee64928c780429ea6eeb2b3ed068a> |

**Pillpal, The Smart Medication Reminder System**

The PillPal system is an IoT-based smart pillbox and mobile companion application designed to help users manage their medication schedules efficiently. The portable pillbox device (about the size of an AirPods case) provides reminders through vibration, LED light, and buzzer alerts, while the app sends notifications, records medication history, and allows easy schedule management.

The system aims to support users who need consistent medication adherence by combining physical and digital reminders. It can be paired to the app via WiFi and carried anywhere, offering accessibility, reliability, and convenience.

**Target Users:** Elderly, visually impaired, or people with memory challenges.

* Motion sensor, buzzer, LED, light sensor
* Large buttons, audio cues, feedback
* Helps anyone manage medication schedules

|  |
| --- |
| **UX DESIGN AND TESTING** |

|  |  |
| --- | --- |
| **[IKRAM]** 26/09/25 | Begin primary and secondary user personas  Create a rough template of personas to represent key target audiences for PillPal.  Motivation: define representative users to guide design decisions, ensure accessibility considerations, and validate scheduling and notification features during development. |
| **[FAVOUR]** 30/09/25 | Define user target segments and use motivations  Document the primary user segments and describe their routines, challenges, and how the product addresses each need. This expands upon earlier persona work by grouping users based on medication type and lifestyle patterns.  Motivation: clarify user categories to validate core system features, guide UI design, and ensure the reminder logic addresses timing-critical use cases across distinct lifestyle groups. |
| **[FAVOUR]** 18/10/25 | Document system environment and user context  Describe the environmental conditions, external interfaces, and human factors influencing PillPal. Define where and how users will interact with both the mobile app and physical pillbox, highlighting contextual challenges and accessibility needs.  Motivation: provide design context and usability constraints to inform interface layout, alert behavior, and hardware ergonomics across varied user environments. |
| **[SOFIA]** 20/10/25 | Define complete system concept and design overview  Add a detailed system description covering the concept model, functionality, and technical design of the PillPal smart medication reminder. Explain how hardware and software components interact to deliver synchronized medication alerts, data logging, and user feedback.  Motivation: define an integrated hardware–software solution that ensures reliable medication adherence through redundant reminders, user control, and accessible interaction design. |
| **[IKRAM]** 22/10/25 | Document end-to-end user tasks and flows  Add the Task section describing how users set up, operate, and maintain the PillPal system across common scenarios. Tasks cover onboarding, schedule management, alerts, history review, customization, maintenance, and offline behavior.  Motivation: provide a task-centered view to guide UX flows, validation, and test scenarios, ensuring reliable reminders and accessible user interactions across real-world contexts. |
| **[IKER]** 25/10/2025 | User personas and accessibility needs with HTAs included  Create actually detailed personas to anchor design decisions, covering demographics,  motivations, challenges, environment, and interface requirements. Focus on  time-sensitive medication use, privacy, and accessibility.  Motivation: ground UI/UX, alert logic, and accessibility in real user  needs, reducing design risk and guiding testing scenarios.  <https://www.notion.so/UTSE-271ee64928c780099cfac5acb5e220ca> |

**1. USER (UTSE)**

* **Workaholics:** People who are constantly on the go with demanding schedules that have to take time-sensitive medications (like ADHD stimulants). If their dose is missed they may suffer the consequence of symptom breakthrough or sleep issues. PillPal will help them pack their medication on the go or remind them to prevent missed doses.
* **Contraceptive Users:** People who take oral contraceptives that require a consistent daily timing within a 3 hour window(Planned Parenthood, n.d.) to maintain that is it effective. Missing birth control pills leads to hormone level drops, reducing effectiveness and increasing pregnancy risk, along with side effects like breakthrough bleeding or nausea (K Health, 2022).
* **Athletes/Gym-goers:** People who workout regularly/professional athletes take daily performance supplements (pre-workout, protein). These are timed around their workout for better results. Taking supplements too early can cause the effects wear off and too late have no benefit during workout.

**Demographic / Character**

* Aged 17-66, but age inclusive
* Active lifestyle, busy schedules, values independence in managing their own help
* Owns smartphone, technical skills are basic to moderate comfort with digital platforms
* Struggle with routine, just need medication backup

**Motivations**

* Stay ahead of medication schedules without stress and anxiety.
* Prevent relapse of symptoms.
* Manage their own health, not wanting to rely on others – independence.
* Take medication privately without attracting attention.

**2. TASK ANALYSIS (UTSE)**

If necessary, found at: <https://www.notion.so/UTSE-271ee64928c780099cfac5acb5e220ca>

**3. SYSTEM ANALYSIS (UTSE)**

* When the system is powered on, the pillbox connects to the user’s smartphone application via WiFi. The app displays a simple interface where users can sign in, create an account, and manage their medication schedules.
* Users can add new medications by entering the name, dosage times, and repetition frequency (daily, weekly, or custom). Each medication can be assigned to one of six compartment slots in the pillbox. Once saved, the schedule is transferred to the pillbox, which vibrates, beeps, and lights up at the set times to remind the user.
* The app also sends push notifications before each dose and logs whether the user took, snoozed, or missed the medication. If the phone is unavailable, the pillbox continues to alert the user independently.
* Users can view their medication history, check streaks for consistency, and clear or review past logs. The settings section allows customization of notification preferences such as sound, vibration, LED light, reminder timing, and accessibility options like a dyslexia-friendly font.
* The hardware uses a Raspberry Pi Zero connected to a PiSugar 2 battery for portability and power management. The system includes vibration motors, LED indicators, and a buzzer, all protected and managed through basic electronic components such as transistors, resistors, diodes, and capacitors on a perfboard.
* Each user action is confirmed by either the app or the device (for example, when saving a medication or acknowledging a reminder), ensuring consistent feedback and reliability in use.

**4. ENVIRONMENT (UTSE)**

**Context**

* The system is primarily used in non-domestic settings such as the workplace, public transport or sport facilities where medication needs to be taken on time.
* The system can also be used in domestic settings such as at home or in a dorm room, placed on a bedside table, bathroom counter or attached with a magnet to the fridge.
* Users interact with the mobile application when scheduling medications, checking their daily plan or confirming doses taken. They interact with the physical box when taking medication at scheduled times or restocking compartments.
* The system is used as a personal organisational tool to manage medication schedules, take medication on time and privately and carry it conveniently wherever.
* Users are likely to interact with the system while multitasking, such as commuting, during a busy workday or before/after gym sessions.

**External Interface**

* The application runs on smartphones, requiring minimal navigation with clear icons, labels and text on a simplified interface.
* The user typically operates the box from a sitting position at a desk or kitchen table, or while standing in a bathroom or locker room, with the box worn on a lanyard, clipped to a backpack or placed on a surface.
* Users interact with the mobile application when scheduling medications, checking their daily plan or confirming doses taken. They interact with the physical box when taking medication at scheduled times or restocking compartments.
* Users may need to temporarily step away from other activities to access the box and confirm medication taken (e.g., leaving a meeting to take medication privately).
* The interface requires approximately 2 taps to complete most tasks, allowing for quick, distraction-free navigation.
* The app and physical box work together to provide alerts (audible buzzer, vibration, on-screen alarm) that users can customize based on their preferences.
* Since users are often handling medication in various environments (noisy gyms, quiet offices, public spaces), the system provides customizable alert types to ensure discretion.

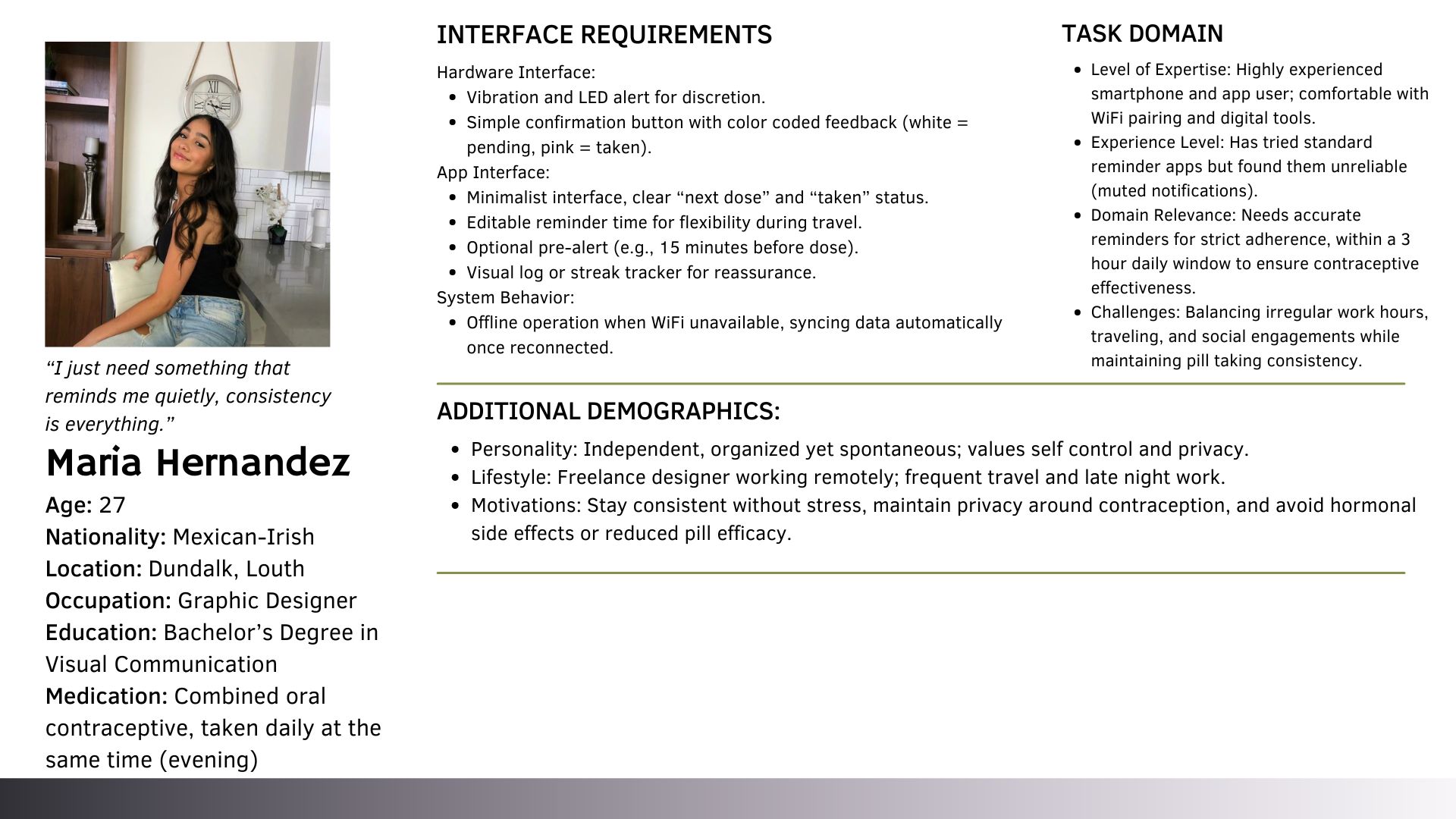
**Human Factors**

* Users may be busy in other activities (working, exercising, commuting) while interacting with the system, requiring short bursts of attention rather than sustained focus.
* Potential for external distractions such as colleagues, public transport noise, gym activity or family members/pets must be considered when designing alert types and interaction flows.
* Privacy is important for users who may not wish to disclose their medication needs in public or professional settings. The system's discreet design (compact size, subtle alerts) addresses this concern.
* The system's low visual complexity and icon-driven interface help users quickly recover from interruptions or resume tasks.
* As the system is used across diverse age groups and abilities, users with different technical levels and visual or reading impairments must find it accessible. Features like soft colours, high contrast text, large buttons and optional dyslexia-friendly fonts support better understanding.
* The physical box must be easy to clean, as users handle it regularly when taking medication.

|  |
| --- |
| **Primary, Secondary and Tertiary Personas (PRIMARY PERSONA)** |

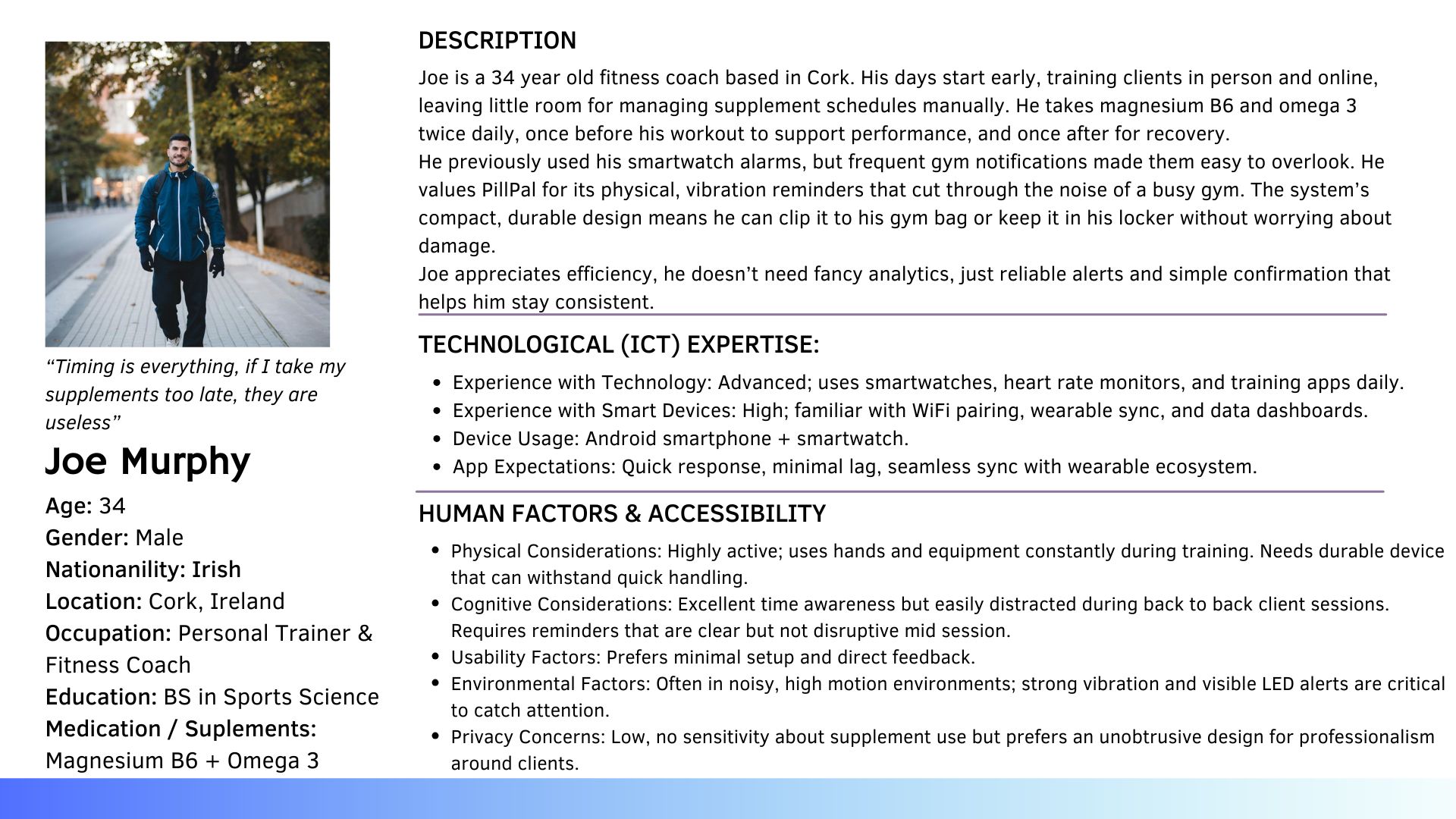
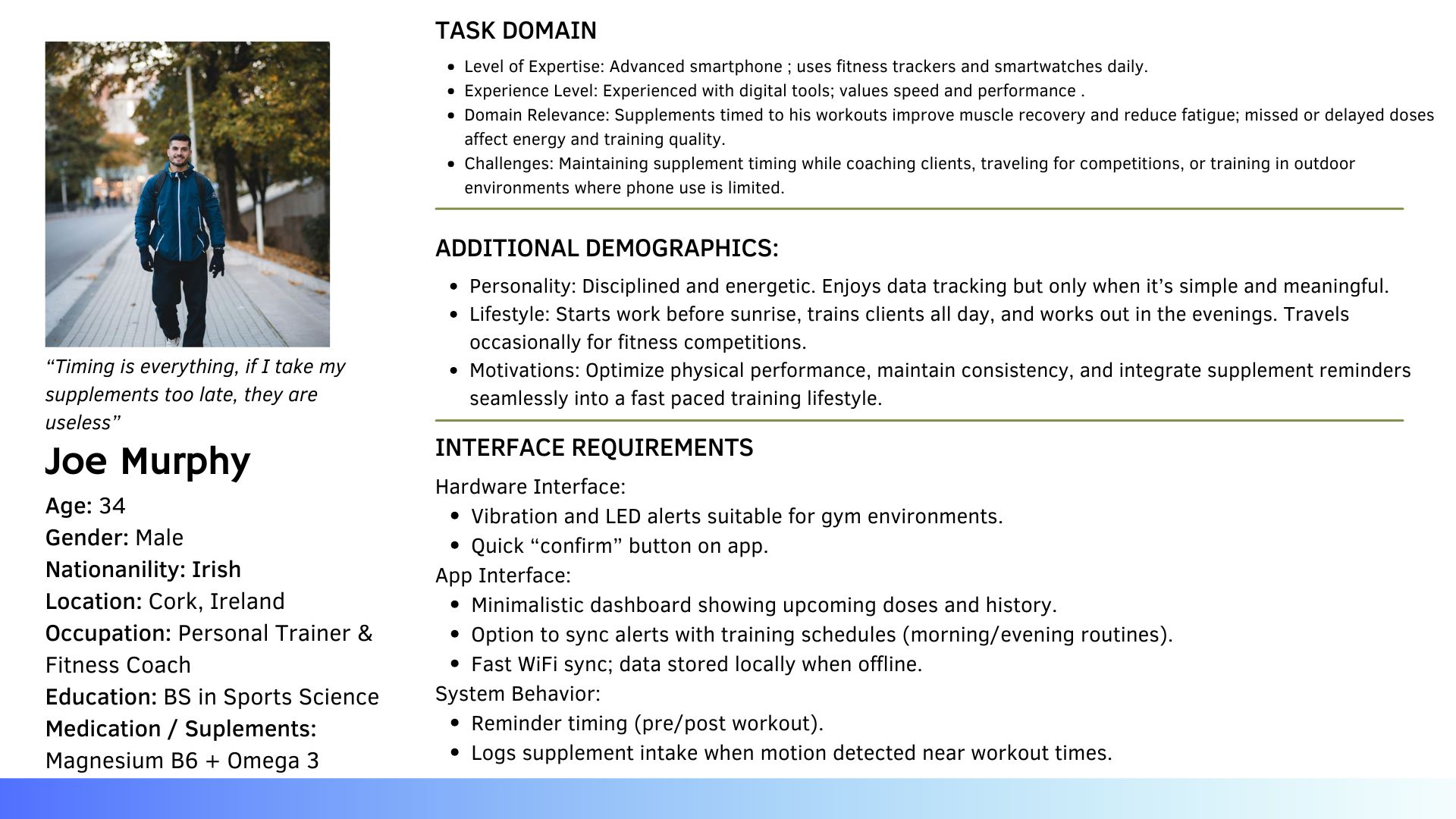
****

|  |
| --- |
| **Primary, Secondary and Tertiary Personas (SECONDARY PERSONA)** |

**A close-up of a person's profile

AI-generated content may be incorrect.**

|  |
| --- |
| **Primary, Secondary and Tertiary Personas (TERTIARY PERSONA)** |

****

|  |
| --- |
| **HTA #1 (TASK ANALYSIS)** |

**A diagram of a company

AI-generated content may be incorrect.**

|  |
| --- |
| **HTA #2 (TASK ANALYSIS)** |

**A chart with colorful squares

AI-generated content may be incorrect.**

|  |
| --- |
| **HTA #3 (TASK ANALYSIS)** |

**A diagram of a company

AI-generated content may be incorrect.**

|  |
| --- |
| **USER INTERFACE** |

|  |  |
| --- | --- |
| **[FAVOUR]** 03/10/25  **[FAVOUR]** 08/10/25 | Design full PillPal UI screens in Figma  Create a complete set of high-fidelity interface mock-ups for the PillPal mobile application using Figma. The designs cover onboarding, account management, medication scheduling, history, notifications, and settings flows.  Motivation: establish a cohesive, accessible UI prototype that supports PillPal’s functional requirements and provides a polished user experience consistent with the project’s accessibility goals.  Test UI interactions and navigation in Figma  Conduct interactive testing of the PillPal mobile interface using the Figma prototype to validate transitions, button responses, and user flow logic. The goal was to ensure an intuitive experience with minimal friction across all key tasks.  Motivation: verify that the designed interactions align with user expectations, support quick task execution, and reinforce accessibility principles before implementation. |
| **[SOFIA]** 09/10/25 | Design watch UI and pillbox model in Figma  Completed the design and visual modeling of the PillPal smartwatch  interfaces and physical pillbox casing using Figma. This update expands  the hardware and UI/UX modeling phase by visualizing both the wearable  interface and the mechanical layout of the pill storage system.  Outcome:  This commit visualizes the core hardware and UI experience of PillPal,  demonstrating how users will interact with the wearable screen and the  portable pillbox. It aligns visual design and functionality for both  hardware and digital components of the system.  <https://www.figma.com/design/QTRpeSIoosEHFiKMrgtxLi/Untitled?node-id=0-1&p=f&t=uyh70csyVxqY9v6G-0> |
| **[SOFIA]** 24/10/25 | Write UI & user interaction documentation for PillPal  Completed detailed written documentation describing the PillPal mobile  application’s user interface, connection process, accessible data, and  functional interactions between the app and the IoT pillbox device.  This section directly addresses the project requirement to explain how  users connect, what data is available to them, and what functionality  they have to view and interrogate the data.  Outcome: Provided a comprehensive explanation of the PillPal interface and its  interaction with the device, ensuring the report clearly demonstrates  how users connect, view, and interact with their medication data. |

**1. User Interface Overview**

The PillPal mobile application features a clean, accessible, and intuitive interface designed to assist users in managing time-sensitive medications. The interface uses large buttons to support clarity and usability for users of all ages, including those with visual or cognitive impairments.

All screens were designed in Figma (PLEASE VISIT THE LINK TO VIEW IN DETAIL):

<https://www.figma.com/design/KdbZUIp1p2ycNemRNkvxU7/Pill-Pal?node-id=0-1&p=f&t=c4YWAVTdaG3kDCIb-0>

* 1. **Onboarding and Authentication**

**Screens:**  
Welcome, Sign In, Create Account, Email Verification

**Function:**

* Users register or log in through a secure interface.
* Establish WiFi pairing between the app and the PillPal device.
* One-time onboarding setup with clear prompts.

**Data Available:**  
User credentials (email, password, name).

Screens screenshot of a login screen

AI-generated content may be incorrect.**Mock-up Example:**

**1.2 Dashboard / Home Screen**

**Description:**  
The home screen greets the user by name and displays current date and quick access to their medications, reminders, and refill alerts.

**Functionality:**

* View daily medication schedule.
* Monitor adherence streaks and refill reminders.
* Tap medication entries to view or edit details.

**Notifications:**

* Push alerts for upcoming doses, missed medications, or refills.
* Alerts via sound, vibration, or LED indicators.

**History:**

* Log of taken, missed, or snoozed doses with timestamps.
* Visual streak tracker for adherence consistency.

A screenshot of a cell phone

AI-generated content may be incorrect.**Mock-up Example:**

**1.3 Medication Management Screens**

**Screens:**  
Add Medication, Select Times, Choose Compartment, Add Notes

**Functionality:**

* Add new medications by entering name, dosage times, and frequency.
* Assign medications to one of six pillbox compartments.
* Define repeat cycles (daily, weekly, custom).
* Include optional notes for usage instructions

**Data Available:**  
Medication name, schedule, dosage, compartment, and notes.

**Mock-up Examples for 1.3 Medication Management Screens:**Screens screenshot of a phone

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**1.4 Settings and Accessibility**

**Functionality:**

* Adjust sound, vibration, LED, and reminder timing.
* Accessibility settings for high contrast mode and dyslexia-friendly fonts.
* View account details, privacy policy, and device connection status.

**Mock-up Example:**

Screens screenshot of a phone

AI-generated content may be incorrect.

**2. User Connection and Data Access**

* The mobile app connects to the PillPal device via WiFi.
* Upon pairing, the app synchronizes schedules and transmits updates to the device.
* If the phone disconnects, the device continues operating independently using stored schedules.
* When reconnected, all adherence logs are synced back to the app.

**Data accessible to the user includes:**

* Medication list and schedules.
* Notification history and adherence logs.
* Personalized settings and account information.
* The user cannot access or modify backend data or device firmware, ensuring security and reliability.

**3. Functionality and Data Interaction**

**Users can:**

* View daily and weekly medication schedules.
* Review and export medication history.
* Receive reminders and confirm, snooze, or mark doses as missed.
* Adjust alert preferences and accessibility settings.

Each user action updates the device and backend database, ensuring synchronized data integrity between the mobile app and physical pillbox.

**4. Testing**

**4.1 Prototype Testing**

* Conducted using **Figma interactive prototypes**.
* Tested navigation flow, visual hierarchy, and accessibility color contrast.
* Iterative feedback led to simplified button labels and improved readability.

**4.2 Functional Testing**

* Verified account creation, WiFi connection, schedule synchronization, and alert triggers.
* Ensured local and remote data consistency after reconnection.
  1. **Usability Testing**

Participants completed key tasks:

* + Adding a medication.
  + Acknowledging a reminder.
  + Reviewing adherence history.
  + Collected qualitative feedback to optimize clarity and reduce interaction steps.

**4.4 Accessibility Testing**

* Tested readability in various lighting conditions.
* Confirmed usability with high-contrast and dyslexia-friendly fonts.
* Ensured large tap targets for quick and accurate user input.

|  |
| --- |
| **SYSTEM ARCHITECTURE MODEL** |

|  |  |
| --- | --- |
| **[SOFIA]** 20/09/2025 | Design system architecture and communication flow on Miro  Create the initial system architecture diagram showing the interaction between the client application, cloud backend, and Raspberry Pi Zero device. Define the communication protocols and their respective roles to clarify system responsibilities.  Motivation: establish a clear, modular architecture that supports both reliable IoT messaging and responsive user interactions while allowing for offline operation on the device. |

**A diagram of a diagram of a system

AI-generated content may be incorrect.System Architecture (Miro)**

REST API is used between the mobile app and the backend (e.g. logging in, account creation, managing medication schedules).

(Request-Response interactions)

MQTT is used between the backend and the pillbox device (Pi 0) for real-time or event communication.

Backend publishes commands e.g. reminder updates/vibration triggers, pillbox responds by publishing status updates (e.g. lid opened).

Syncing is taken care of by Mosquitto broker and MQTT. We use prebuilt technology to avoid writing too much code from scratch/reinventing the wheel. <https://miro.com/app/board/uXjVLURLQ68=/>

|  |
| --- |
| **DATABASE AND ERD** |

|  |  |
| --- | --- |
| **[SOFIA]** 15/10/25 | Draft ERD v1 in Miro  Create the initial entity–relationship diagram to capture the core data domains for PillPal. Define baseline entities, their attributes, and high-level relationships to support scheduling, devices, and adherence.  Included entities:  - users and devices  - medications linked to users and compartments  - compartments (with slot\_number constraint)  - med\_schedule\_rules for repeat timing  - dose\_instances and dose\_events for adherence tracking  - notification\_settings and device\_state  Relationships show how medications connect to specific devices and compartments, with dose history captured through events.  Motivation: establish a baseline data model to visualize relationships and prepare for normalization and indexing in future drafts. |
| **[SOFIA]** 20/10/25 | Normalize ERD to 3NF (v2)  Update the ERD to achieve third normal form (3NF), reducing redundancy and improving flexibility in user, device, and medication management.  Key changes from v1:  - Split user information: created new user\_profiles table separate from  users to isolate personal data (full\_name, username, timezone)  - Added compartment\_assignments to record which medication occupies  which slot over time (supports historical tracking)  - Introduced med\_times table to store multiple time entries per schedule  rule (1..N relationship from med\_schedule\_rules)  - Removed direct compartment\_id and device\_id from medications to reduce  dependency and improve normalization  - Refined device\_state and notification\_settings to reference parent  entities via FKs only  - Adjusted dose\_instances and dose\_events to reflect clear one-to-many  history with created\_source for origin tracking  - Ensured all FKs and unique constraints align with referential integrity  Motivation: improve data integrity, enable historical relationships, and ensure each entity represents a single concept while preserving query efficiency.  <https://www.notion.so/Ideas-26fee64928c780429ea6eeb2b3ed068a> |
| **[SOFIA]** 21/10/25 | Implement MySQL schema, analytical views and queries  Translate the finalized 3NF ERD into a normalized MySQL schema. Define  tables, foreign keys, unique constraints, and views for reporting on  upcoming doses and adherence rates.  Schema highlights:  - users + user\_profiles linked via FK (ON DELETE CASCADE)  - device\_pairings to manage user-device history  - compartments and compartment\_assignments for slot tracking  - medications linked to users with date validity checks  - med\_schedule\_rules and med\_times for structured scheduling logic  - dose\_instances and dose\_events for adherence and event history  - notification\_settings and device\_state for personalization and telemetry  - Comprehensive indexing on common query columns (user\_id, med\_id,  scheduled\_at, status)  Motivation: produce a deployable relational foundation that matches the  3NF design, supports scalability, and integrates cleanly with backend  and analytics components. |

**ERD (v1)**

**A screenshot of a diagram

AI-generated content may be incorrect.**

The initial ERD for PillPal was designed around the user as the central point of the system, since every interaction, device, and medication links back to them. The structure shows how each user connects to their assigned smart pillbox, schedules medications, and tracks adherence history through the companion application. Each user has a corresponding set of personal details and preferences stored within their profile and notification settings. The user is directly linked to one or more devices, which represent the physical pillboxes. Every device can contain up to six compartments, represented by the *compartments* table with a unique slot number constraint. These *compartments* were originally connected directly to the *medications* table, allowing each *medication* to be assigned to a physical slot. The *medications* table also included attributes such as medication name, notes, and start and end dates, while *med\_schedule\_rules* handled repetition types and timing. Events like when a pill was taken, snoozed, or missed were logged in *dose\_instances* and *dose\_events*, allowing the system to store and review medication history.

At this stage, while the ERD captured all required data, there were redundancies and dependencies between *devices*, *compartments*, and *medications*, which needed further normalization.

**A diagram of a computer

AI-generated content may be incorrect.ERD 3NF (v2)**

The final ERD was refined and normalized to Third Normal Form (3NF) to reduce redundancy, improve scalability, and ensure data integrity across all relationships.

User data was split into two separate entities: *users* and *user\_profiles*, to separate authentication details from personal information. This allows for better privacy management and data organization.

The *compartment\_assignments* table was introduced to record historical relationships between compartments and medications, allowing the system to track which medication occupied each slot at any given time. The *med\_times* table was added to store multiple daily times per medication rule, creating a one-to-many relationship from *med\_schedule\_rules*.

Additionally, *device\_pairings* were improved to keep a historical log of devices linked to each user, supporting features like multiple pillboxes per account. The *device\_state* and *notification\_settings* tables are also changed to hold current device telemetry (battery level, last sync) and user preferences (sound, vibration, LED alerts, and accessibility settings). Redundant foreign keys were removed from *medications*, ensuring cleaner separation between physical and logical entities.

The final schema ensures that every component is clearly defined, historically traceable, and efficiently queryable for analytics and synchronization with the IoT device.

* Least-privilege DB user for the Pi (INSERT/SELECT only).
* Input validation and query parameterization (SQLAlchemy) to prevent injection.
* (If time permits, could implement) Backups: daily snapshot of the cloud DB; optional rolling 7-30 day retention.
* Data minimization: store only what you need (no PHI beyond reminders/events).

**Please refer to the file in the .zip, titled ‘pillpal\_db.sql’ for the database schema and test queries.**

Intended for testing in XAMPP.

|  |
| --- |
| **DATA, DATA STORAGE, AND DATA PROCESSING** |

|  |  |
| --- | --- |
| **[IKER]** 23/10/25 | Document device power, connectivity, and data flow  Completed detailed technical documentation outlining how the PillPal hardware operates, connects to the internet, and processes data locally and in the cloud. This document defines the full data lifecycle for the final Pi Zero prototype.  Work completed:  - Described hardware power system.  - Defined network and connectivity model.  - Explained local data collection and caching.  - Described cloud synchronization process.  - Added security section.  Outcome: This document formalizes how the PillPal device is powered, connected,  and synchronized securely. It serves as the system’s technical blueprint  for network architecture, data handling, and protection mechanisms. |
| **[SOFIA]** 24/10/25 | Update connectivity doc to align with system architecture  Revised and refactored the data documentation to reflect the finalized system architecture based on MQTT message handling and REST API integration. This update replaces the previous SQLite-based offline caching model with a more efficient Mosquitto MQTT broker design that matches the project’s architectural diagram and communication workflow.  Outcome: Document now fully aligns with the system architecture design, using  MQTT for reliable data persistence and REST APIs for control logic.  Provides a consistent explanation across documentation and diagrams. |

The pillbox runs on a Pi Zero powered from a PiSugar battery/charger. It connects to the internet through my phone’s Wi-Fi hotspot. A Flask service on the Pi controls the buzzer and vibration motor via GPIO and small NPN transistor drivers, and logs events and battery to a local file. The phone app talks to the Pi over the hotspot using a simple REST API for status and to mark doses. When the Pi has internet, it securely syncs to our cloud database (for the demo we’ll use XAMPP; later this could be AWS). We secure everything with WPA2, HTTPS, token-based auth, least-privilege DB users.

**What data the device will gather**

From the Pi (device-side):

* Reminder events: when a scheduled alert fires (timestamp).
* Sensor events: when the motor/buzzer/LED is activated and for how long.
* User actions: “Taken” (open/closed Lid), “Snooze”, “Dismiss” (from the phone app command).
* Battery telemetry: percentage/voltage (via PiSugar API), charging state, last-low-battery time. (Link to documentation for this: <https://www.pisugar.com/blogs/pisugar-blog/display-raspberry-pi-battery-indicator>)
* Device health: uptime, last sync time.

**How the data gathered by your device will be stored**

On the Pi 0, a local Mosquitto MQTT broker will be used to temporarily store and forward data. The pillbox publishes sensor readings and events (e.g. lid open/close, battery level) to this local broker instead of sending them directly to the cloud. Mosquitto is configured with message persistence and an MQTT bridge to the cloud broker. This means that when the Internet connection is unavailable, all outgoing messages are safely stored on the Pi’s local Mosquitto queue. Once connectivity is restored, the broker automatically synchronizes the stored messages with the cloud backend. This approach removes the need for a separate database on the device while ensuring no data is lost during connectivity outages.

TLDR: On the Pi (offline cache): A local Mosquitto MQTT broker provides message persistence and storage/forwarding capability. Events and telemetry are queued locally when offline and automatically forwarded to the cloud once the wifi connection is back, eliminating the need for a dedicated database on the device.

For testing/development purposes we would start by using XAMPP (MySQL) on localhost and later for production we would switch to AWS.

**How the data gathered by your device will be processed**

Local logic on Pi (Flask app):

The cron scheduler wakes at the times set by the user in the app, drives the GPIO to vibrate/beep/flash, and logs to a text file.

The Flask API exposes endpoints the Android app calls over Wi-Fi (same hotspot LAN) for:

`GET /status` (battery %, next reminder)

`GET /schedule` (device cache of user’s plan)

`POST /action` (Taken/Snooze/Dismiss)

A sync worker periodically:

* Uploads new local events to the cloud (`POST /api/events`).
* Downloads the authoritative schedule from the cloud (`GET /api/schedule`) and updates the local cache.

No Bluetooth path is needed. All data flows between the Pi, the phone client and the cloud over Wi-Fi using REST/JSON/MQTT.

**Transport & auth:**

* Use HTTPS/TLS for any internet traffic (Let’s encrypt on the cloud API).
* JWT or signed tokens: short expiry, refresh securely.
* Avoid hardcoding secrets (passwords), load from env vars or a `.env` file with correct permissions.

|  |
| --- |
| **CRON JOBS** |

|  |  |
| --- | --- |
| **[FAVOUR]** 23/10/25 | Research cron and define PillPal job schedules  Studied Linux cron syntax and scheduling patterns to automate time-sensitive tasks on the Raspberry Pi. Verified examples on-device, took notes, and drafted the initial crontab for medication checks, sync, backups, and log maintenance.  References:  - Kubernetes CronJob docs (syntax refresher)  - crontab.guru examples (cheatsheet)  Notes:  - Tested a sample cron on the Pi to confirm execution; captured findings  in a local notepad file  - Jobs will call Flask/CLI endpoints or scripts that in turn publish  MQTT/REST updates as needed  - Future hardening: use systemd timers for drift control if required,  and redirect stdout/stderr to rotated logs  Motivation: establish reliable, time-based automation for reminders, sync, and housekeeping on the Raspberry Pi to ensure consistent operation with minimal manual intervention. |

**Research using cron:**

<https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/>

**┌───────────── minute (0 - 59)**

**│ ┌───────────── hour (0 - 23)**

**│ │ ┌───────────── day of the month (1 - 31)**

**│ │ │ ┌───────────── month (1 - 12)**

**│ │ │ │ ┌───────────── day of the week (0 - 6) (Sunday to Saturday)**

**│ │ │ │ │ OR sun, mon, tue, wed, thu, fri, sat**

**│ │ │ │ │**

**│ │ │ │ │**

**\* \* \* \* \***

**0 3 \* \* 1** means this task is scheduled to run weekly on a Monday at 3 AM.

|  |  |  |
| --- | --- | --- |
| **Entry** | **Description** | **Equivalent to** |
| **@yearly (or @annually)** | **Run once a year at midnight of 1 January** | **0 0 1 1 \*** |
| **@monthly** | **Run once a month at midnight of the first day of the month** | **0 0 1 \* \*** |
| **@weekly** | **Run once a week at midnight on Sunday morning** | **0 0 \* \* 0** |
| **@daily (or @midnight)** | **Run once a day at midnight** | **0 0 \* \* \*** |
| **@hourly** | **Run once an hour at the beginning of the hour** | **0 \* \* \* \*** |

In PillPal, Cron will be used to automatically schedule time-sensitive tasks on the Raspberry Pi. The primary use case would be checking medication schedules every minute and triggering alerts (vibration, buzzer, app notification) when it's time for the user to take medication. Cron will also handle daily schedule syncing, data backup and cleaning up old logs to ensure system reliability.

**List of cron jobs for PillPal:**

1. Job: medication reminder checker

* Schedule: every minute
* Why: has to check every single minute if it's time to send medication alert
* (because users should be able to change minute in the app).
* CRON JOB: \* \* \* \* \*

2. History backup (every (6/10?) hours): backs up the users' data to the database.

* Job : data backup
* Schedule : every 8 hors
* Why : backup medication history to database
* CRON JOB: 0 \*/8 \* \* \*

3. Daily schedule checker (daily at midnight):

* Job: daily schedule checker
* Schedule: every day at midnight
* Why: make sure schedule is tracked properly in database
* CRON JOB: 0 0 \* \* \*

4. Cleanup for old logs (weekly): pi has limited storage so we have to delete the old system logs.

* Job : log cleanup
* Schedule : weekly
* Why : delete old logs to free storage
* CRON JOB : 0 3 \* \* 0

**Please refer to the file ‘simple cron job example.txt’ inside of the .zip folder for the cron jobs we have.**

|  |
| --- |
| **SECURITY** |

|  |  |
| --- | --- |
| **[FAVOUR]** 22/10/25 | Research GDPR and define security controls  Investigated requirements for protecting PillPal’s health-related data (medication names, schedules, history). Mapped controls to GDPR principles and OWASP guidance, and specified security for data in transit and at rest across app, Pi, and backend.  Motivation: provide a compliant, security-by-design foundation that  aligns with GDPR and OWASP, ensuring user trust and safe handling of  health-related data. |

**How the device and data will be secured**

Device (Pi):

* Wi-Fi: WPA2 hotspot; optionally lock to known SSIDs only.
* OS hardening: change default password, disable password SSH (use SSH keys).
* Flask: run behind gunicorn, bind to LAN only, rate-limit endpoints, validate JSON.

PillPal handles sensitive health data (medication names, schedules, keeping history). This is protected under privacy laws like GDPR (Europe) and HIPAA-adjacent concerns (US). Users need to trust their medical information is safe.

<https://gdpr.eu/what-is-gdpr/>

**Research into security measures:**

<https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html>

1. Authetication: User login with email and password (or we can leave it to google it to have them login so we arent responsible). password hashing (bcrypt).

2. Authorization: Only user can see their own data, no other user should be able to see another user's info. Implement sessions/tokens. Allow users to access their own data.

3. Encryption: data should be protected when it's stored in the database and when it's moving between the systems.

4. Data privacy: We should only collect what we actually need. We need user consent for storage. User should be allowed to delete their account along with all their data (like on Instagram, they do it like a 30 day time frame that the user can re-login to stop their account deletion and data deletion, but after 30 days it’s gone permanently). Of course, follow principles of GDPR.

**Security of data transfer:**

app-to-database server, raspberry pi-to-database, app-to-pi

Things like the users' schedule, their login info and their data/history should all be encrypted on the way to another system.

1. Database: Encrypt sensitive fields (medication names, notes, schedule, etc.).

2. Password hashing : Never storing plain text. bcrypt + salt before storing.

3. Storing locally and security: Since we are using kotlin, authentication tokens are stored in the device’s secure storage (android keystore, ios keychain). For pi, set file permissions to only the user using the running can "read/write" to them - chmod 600

4. Data storage and deleting: Don’t keep forever, only as long as needed, users can delete their account and all their data should be delete too (allow timeframe?).

Sources:

<https://auth0.com/blog/hashing-in-action-understanding-bcrypt/>

<https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>

<https://developer.android.com/privacy-and-security/keystore>

<https://www.circuitbasics.com/file-permissions-on-the-raspberry-pi/>

<https://www.maths.cam.ac.uk/computing/linux/unixinfo/perms#:~:text=700%20means%20you%20can%20do,Suitable%20for%20private%20text%20files>.

|  |
| --- |
| **HARDWARE PARTS** |

|  |  |
| --- | --- |
| **[IKER]** 06/10/25 | Draft ESP32 hardware list and purchase links  Create the initial Bill of Materials (BOM) centered on the ESP32  microcontroller and compile Amazon links for each item to estimate cost  and lead times.  BOM:  - ESP32-C6 touch screen (micro-controller)  - 3v coin size motor  - Transistor 2N222  - Reed Switch  - 220Ω resistance  - Diode  - 100 nf capacitor  Motivation: establish a concrete parts list and sourcing plan to unblock  prototype assembly and budgeting. |
| **[IKER]** 16/10/25 | Revise BOM for Pi Zero; place and receive order  Update the hardware design to use Raspberry Pi Zero as the core compute  module, finalize the new BOM, place orders, and confirm receipt of all  materials.  Sourcing and status:  - Replaced ESP32 items with Pi Zero + PiSugar bundle  - Placed orders (Amazon + official vendors); tracking IDs recorded  - All items delivered and inspected; quantities verified against BOM  Motivation: align hardware with revised software/OS requirements and  ensure all materials are on hand for assembly and bench testing. |

**Core**

* **Raspberry Pi Zero**
* **micro-SD**
* **Jumpers Dupont** female/ male – female to female

**Power**

* **PiSugar 2 (battery of 1200 mAh)** for Pi Zero (charges via USB-C cable)

**Sensors**

* **Coin vibration motors** 10×2.7 mm, 3–5 V
* **Buzzer 5 V**
* **LED simple** (3/5 mm) battery state indicator
* **Reed Sensor** (Open Lid/Closed Lid)

**Protection for electronics**

* **Transistors NPN 2N2222:**
* 1× for the motor, 1× for the buzzer.
* **Diodes 1N4148** 1x for motor/buzzer
* **Resistors:**
* **1 kΩ** (base for 2N2222),
* **220–330 Ω** (for LED).

**Passive electric flow**

* **100 µF / 10 V**
* **100 nF:** for motor/buzzer

**Build**

* **Perfboards:** plaque for final setup (soldering)
* **Tape** 3M / VHB
* Superglue

|  |
| --- |
| **HARDWARE MODEL** |

|  |  |
| --- | --- |
| **[SOFIA]** 08/10/25 | Prototype ESP32 circuit and learn breadboard wiring (v  Collaborated with Iker to begin the initial hardware exploration  phase using the ESP32 microcontroller and breadboard tools. Together,  we watched YouTube tutorials to understand connectivity, GPIO mapping,  power rails, and motor control fundamentals before assembling the  first test circuit.  Activities completed:  - Reviewed online tutorials on ESP32 setup, breadboard use, and circuit  prototyping basics.  - Connected coin vibration motor to the breadboard via transistor and  resistor network based on initial hardware list  - Began first draft of circuit design on Fritzing, including ESP32, motor,  resistor, and power lines.  Motivation: build foundational understanding of breadboard wiring and  ESP32 control through teamwork and guided experimentation to prepare  for a more accurate second prototype. |
| **[IKER]** 11/10/25 | Finish Fritzing v1 diagram for ESP32 smartwatch  Created and completed the first conceptual wiring diagram on Fritzing  for the smartwatch-style medication reminder prototype. The diagram  outlines the initial hardware connections based on the early ESP32  model and serves as a reference for the physical prototype assembly.  Outstanding:  - The Waveshare ESP32-C6 touchscreen board still pending order  (main missing component)  - Next step will be replicating this design physically on a breadboard  once remaining parts are received  Motivation: establish a visual guide for component placement and wiring  to support assembly and debugging during early prototype testing. |
| **[IKER]** 13/10/25  **[IKER]** 19/10/25 | Discard initial smartwatch concept after feedback  Following the in-class pitch presentation, the lecturer provided  feedback regarding project component compatibility and scope. Based on  the requirements outlined for the assignment, the smartwatch medication  reminder concept (using the ESP32-C6 model) was deemed unsuitable for  continuation in its current form.  Next steps:  - Research and define a new design direction centered around a  standalone pillbox device powered by Raspberry Pi Zero  - Re-evaluate component list and system architecture  - Prepare revised concept sketches and hardware list for the next stage  Redesign project to Pi Zero pillbox and finalize hardware  After the lecturer’s feedback, the group collectively decided to pivot  from the smartwatch-based prototype to a standalone smart pillbox  solution that fully aligns with the project’s educational goals —  specifically, programming and integrating with a Raspberry Pi instead  of a microcontroller.  Work completed:  - Group discussion and decision to redesign the system around the  Raspberry Pi Zero as the main processing unit  - Selected all new hardware components compatible with the Pi and its  GPIO layout after research and verification against project criteria  - Confirmed feasibility of using the PiSugar 1200mAh battery for portability and power management  Outcome:  Successfully transitioned from the ESP32 smartwatch prototype to the  new Raspberry Pi Zero-based pillbox design. Final hardware list and  component selection complete, establishing the foundation for circuit  testing and physical prototyping. |
| **[IKER]** 20/10/25 | Begin Fritzing diagram for final Pi Zero pillbox  Started the digital circuit design for the final PillPal pillbox model  using Fritzing. This marks the first version of the completed wiring  layout based on the new Raspberry Pi Zero–based design finalized in the  previous stage.  Work completed:  - Selected all the components and started by connecting the led and coin size motor to the pi including the needed resistances, diode and transistor  -Tested connections by clicking on the breadboard holes to ensure all components are connected properly. Here we tested the reed sensor and further the blue capacitor was added to regulate voltage spikes.  -Simulated our PiSugar battery by adding a LiPo 1000 mAh battery to the diagram with a power booster shown on final result of the diagram  Outcome: Completed the first version of the Fritzing diagram for the final Pi  Zero pillbox, successfully connecting and testing all major components  for correctness and electrical stability. |
| **[IKER]** 21/10/25 | Finalize and refine Fritzing circuit for Pi Zero pillbox  Completed the final version of the Fritzing circuit diagram for the  Raspberry Pi Zero–based PillPal pillbox. All components were properly connected and organized to ensure correct operation and a clean design. The LED, motor, buzzer, and reed sensor were successfully implemented using transistors, resistors, diodes, and capacitors for protection and noise reduction.  A LiPo battery connected through a PowerBoost 500 Charger was used to simulate the PiSugar power module, providing stable 5V to the Raspberry Pi. The wiring was refined using the “bend wire” option in Fritzing to make the layout clearer and more professional. Overall, the setup represents a complete and well-structured circuit ready for testing.  Outcome: Produced a complete and visually refined circuit diagram accurately  representing the final PillPal prototype. The design is stable,  component-safe, and ready for real-world assembly and testing. |

**v1 using ESP32 (08/10/2025) Prototype Started:**

A diagram of a circuit board

AI-generated content may be incorrect.

**v1 using ESP32 (11/10/2025) Prototype Completed:**

A diagram of a circuit board

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.**v2 using Pi 0 (20/10/2025) Prototype:**

A close up of a battery

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A circuit board with wires and wires

AI-generated content may be incorrect.**v2 using Pi 0 (21/10/2025) Prototype Completed:**

**Please refer to the file in the .zip, titled ‘PillPal-Hardware Diagram.fzz’ for the Frtizing diagram.**

**How the hardware will be powered and connected to the internet**

* Power: PiSugar 1200 mAh boosts the Raspberry Pi Zero, charges the battery over USB-C and exposes battery telemetry (voltage/percentage). The Pi reads that battery status via the PiSugar daemon/CLI.
* Sensors: A vibration motor and a buzzer are each driven from 5v rails through 2N2222 NPN transistors, with 1 kΩ base resistors, 1N4148 diodes and a 100 µF bulk capacitor across 5v GND near the motor (to damp inrush). Status LED is current-limited by 220–330 Ω resistor.
* Network: The Pi connects to the internet using Wi-Fi to the user’s phone hotspot (WPA2). No Bluetooth is required. (If there’s no internet, the device works offline and syncs later.)