MOG

MIN MAXING FITNESS

Your personalized fitness app



Johnny Do, Fabian

MOGLOG

MIN MAXING FITNESS

ROADMAP

1

VISION

What Moglog Solves

2

CONCEPT

How the app works from onboarding

3

IMPLEMENTATION

Architecture, data flow, stack and main technical decisions.

4

EVAL

Challenges, current result

5

DEMO

Live APP walkthrough

SUMMARY

VISION & PROJECT GOAL

> "THE APP SHOULD ADAPT TO THE USER – NOT THE OTHER WAY AROUND."

WHY NOW?

Generic fitness apps often ignore actual user context: level, time, goal and equipment.

PROJECT GOAL

Build a functional, understandable and extensible Android prototype.

CORE PROMISE

- Personalized onboarding
- Generated training plan
- Workout logging
- Exercise library
- Fitness-focused AI coach

CONCEPT

FROM PROBLEM TO SOLUTION

BEFORE MOGLOG

- Fixed or generic routines
- Beginners feel overwhelmed
- Advanced users miss depth
- No strong connection between profile and plan

WITH MOGLOG

- Onboarding captures goal, level, time, location and equipment.
- A training profile becomes the basis for plan generation.
- Training, exercise search and coaching all build on the same user context.

PRODUCT MODULES

MOGLOG AT A GLANCE

Dashboard

Shows the current objective, overall state and next actions.

Train

Displays the generated plan and logs completed workouts.

Intel

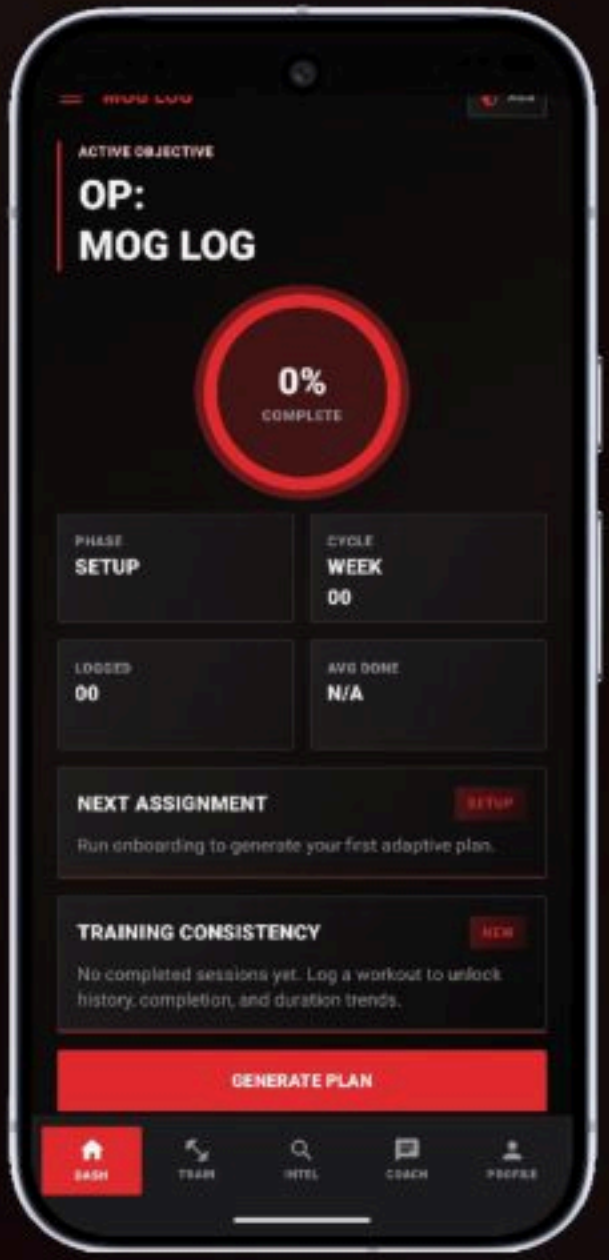
Lets users search the exercise library and filter by relevance.

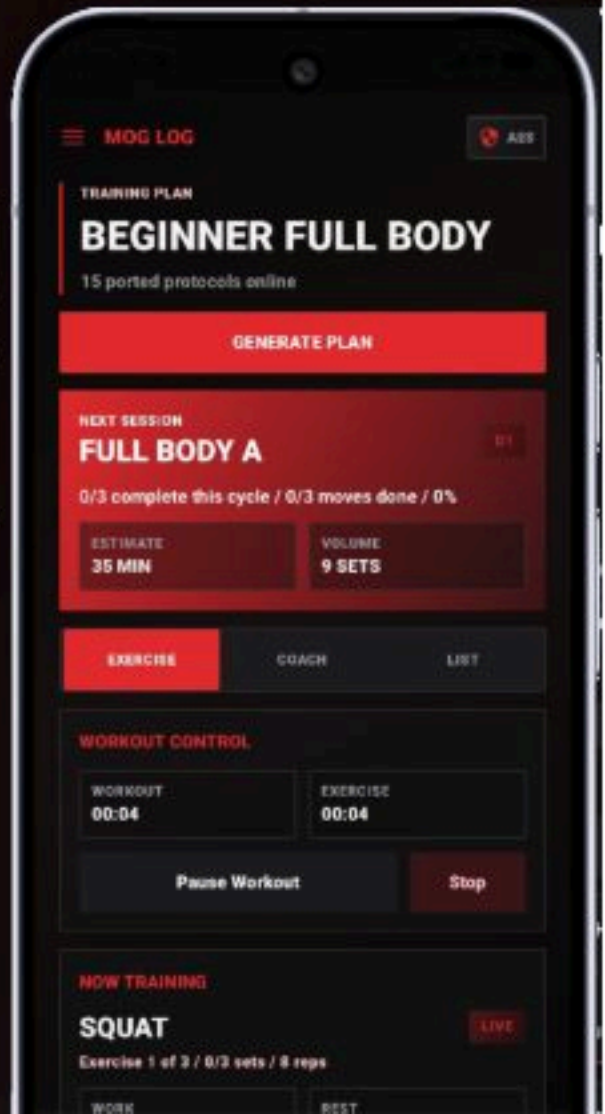
Coach

Answers short fitness-related questions inside one app context.

Profile

Stores onboarding inputs and the active training profile.





PRODUCT MODULES

MOGLOG AT A GLANCE

Dashboard

Shows the current objective, overall state and next actions.

Train

Displays the generated plan and logs completed workouts.

Intel

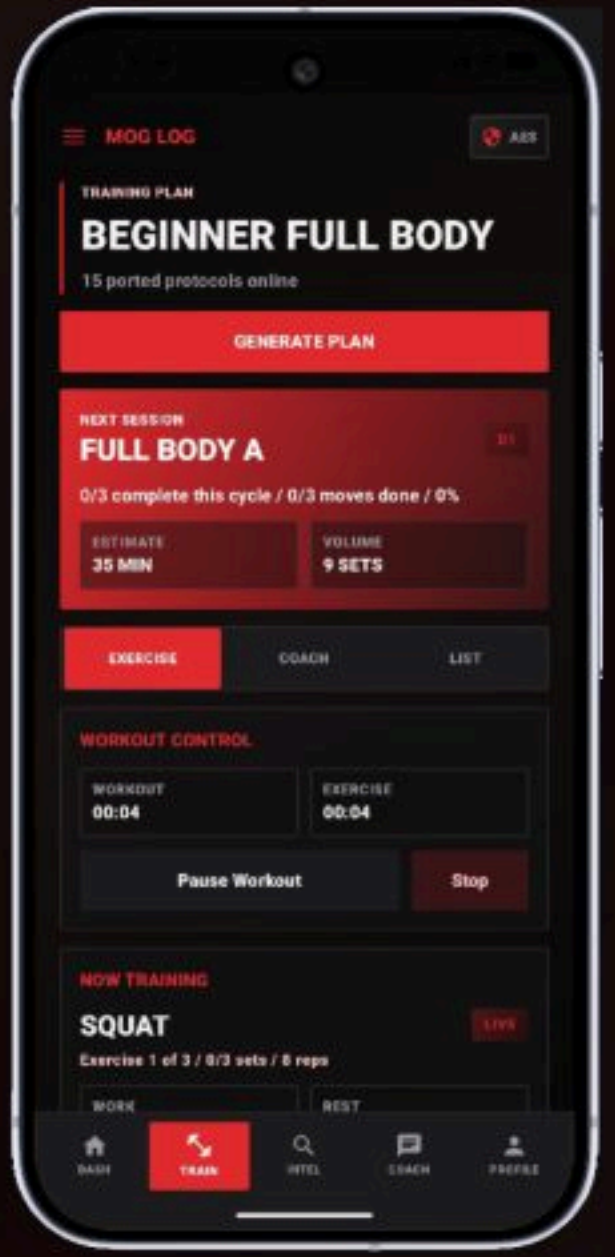
Lets users search the exercise library and filter by relevance.

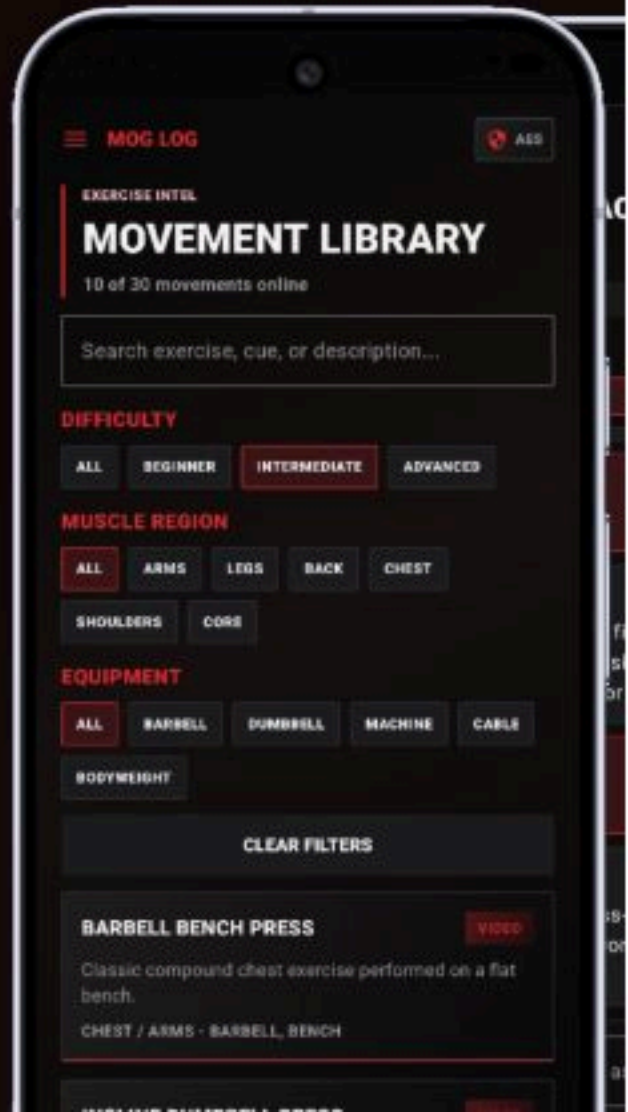
Coach

Answers short fitness-related questions inside one app context.

Profile

Stores onboarding inputs and the active training profile.





PRODUCT MODULES

MOGLOG AT A GLANCE

Dashboard

Shows the current objective, overall state and next actions.

Train

Displays the generated plan and logs completed workouts.

Intel

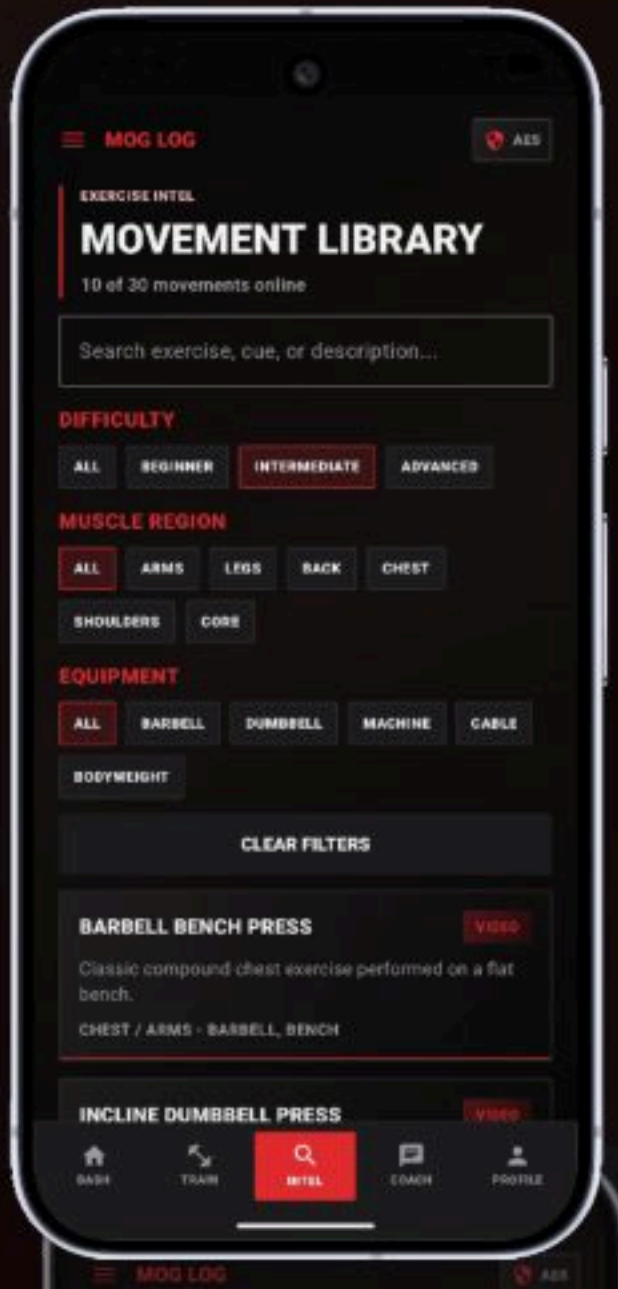
Lets users search the exercise library and filter by relevance.

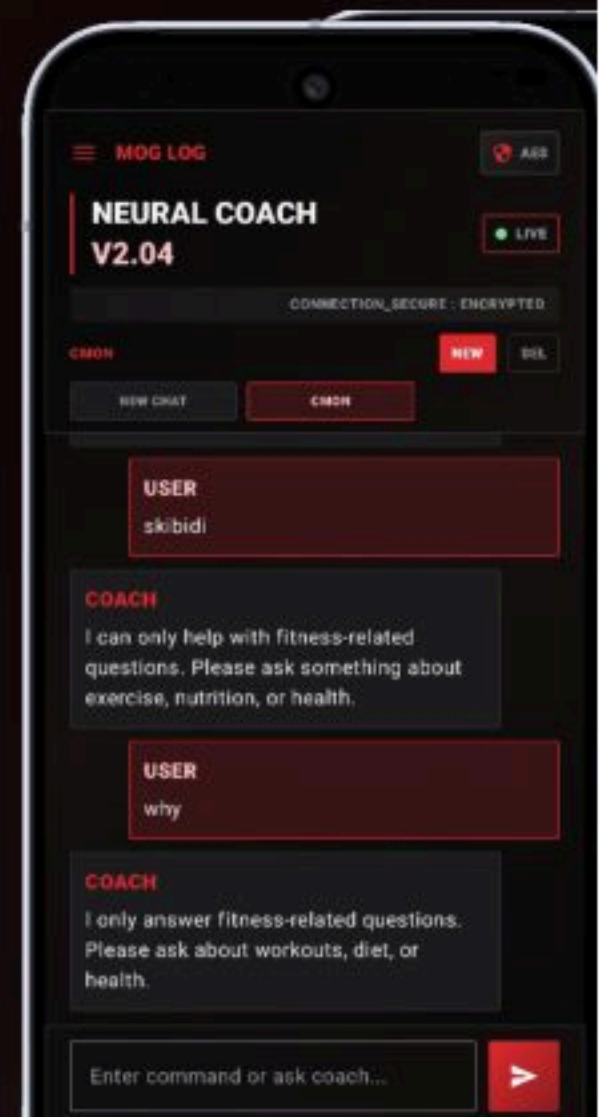
Coach

Answers short fitness-related questions inside one app context.

Profile

Stores onboarding inputs and the active training profile.





PRODUCT MODULES

MOGLOG AT A GLANCE

Dashboard

Shows the current objective, overall state and next actions.

Train

Displays the generated plan and logs completed workouts.

Intel

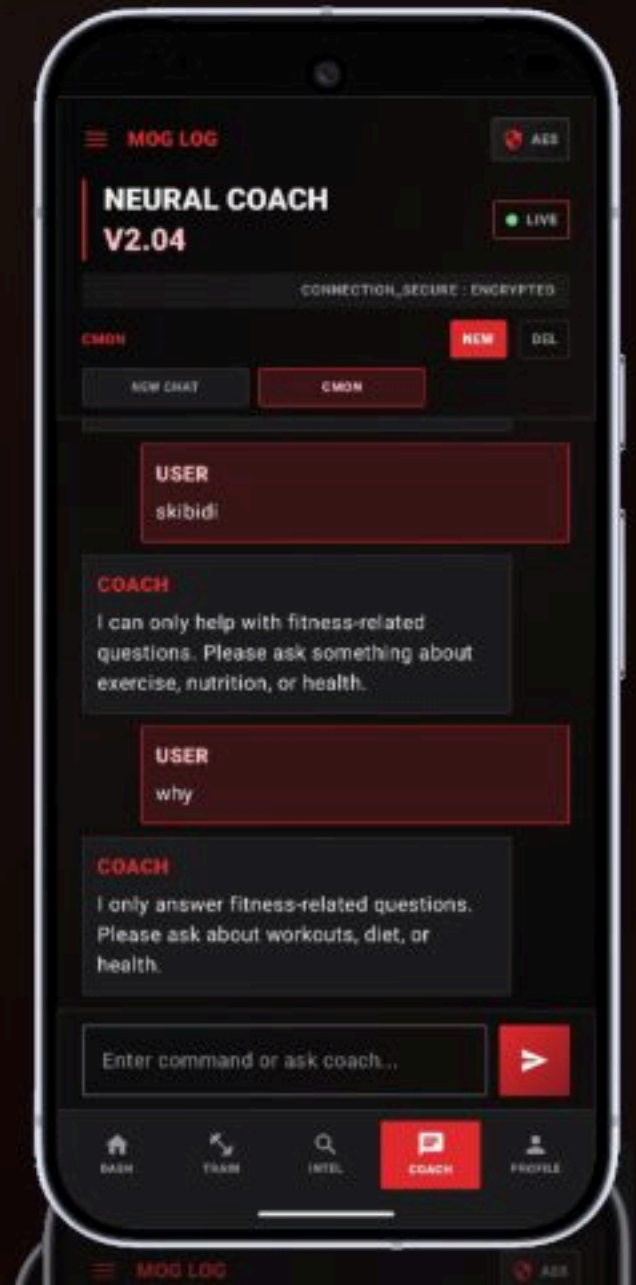
Lets users search the exercise library and filter by relevance.

Coach

Answers short fitness-related questions inside one app context.

Profile

Stores onboarding inputs and the active training profile.





PRODUCT MODULES

MOGLOG AT A GLANCE

Dashboard

Shows the current objective, overall state and next actions.

Train

Displays the generated plan and logs completed workouts.

Intel

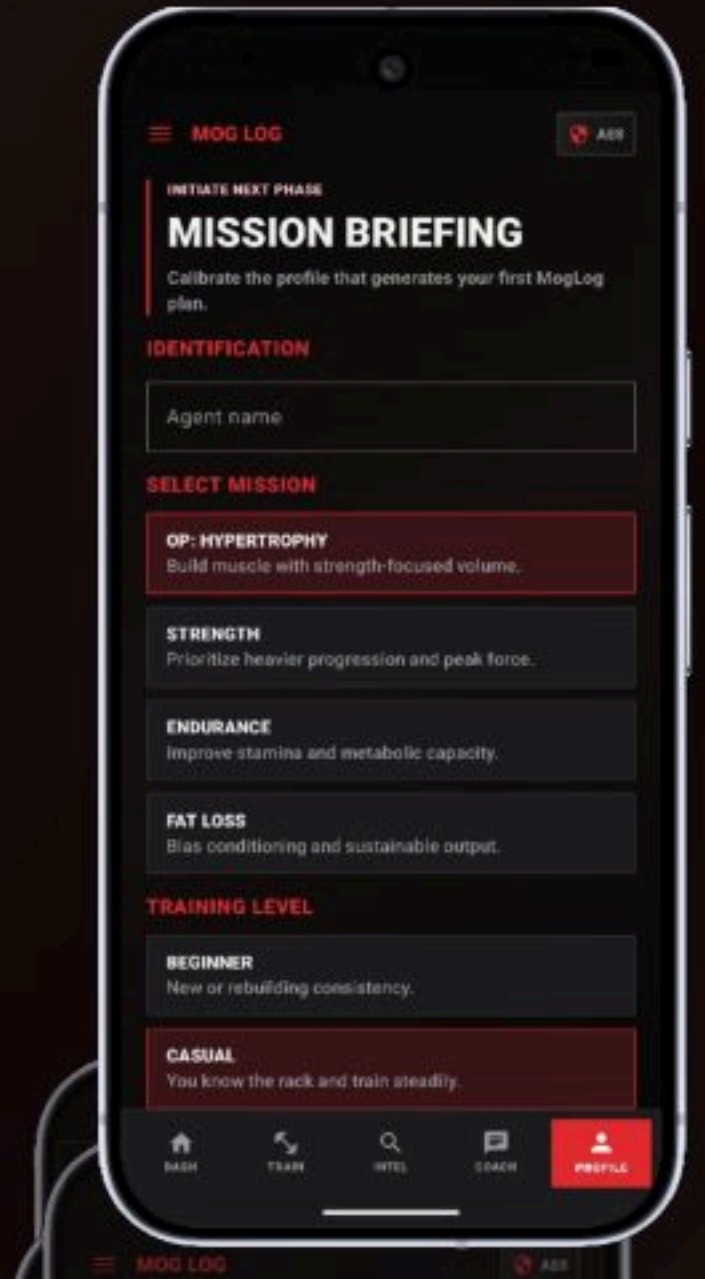
Lets users search the exercise library and filter by relevance.

Coach

Answers short fitness-related questions inside one app context.

Profile

Stores onboarding inputs and the active training profile.



MOGLOG

MIN MAXING FITNESS

END TO END APP JOUNRY

1

ONBOARDING

User enters goal, level, frequency, duration and equipment.

2

TRAINING PROFILE

The app stores these preferences as structured profile data.

3

PLAN GENERATION

A rule-based generator derives an initial workout plan.

4

APP USAGE

The user trains, searches exercises and asks the coach.

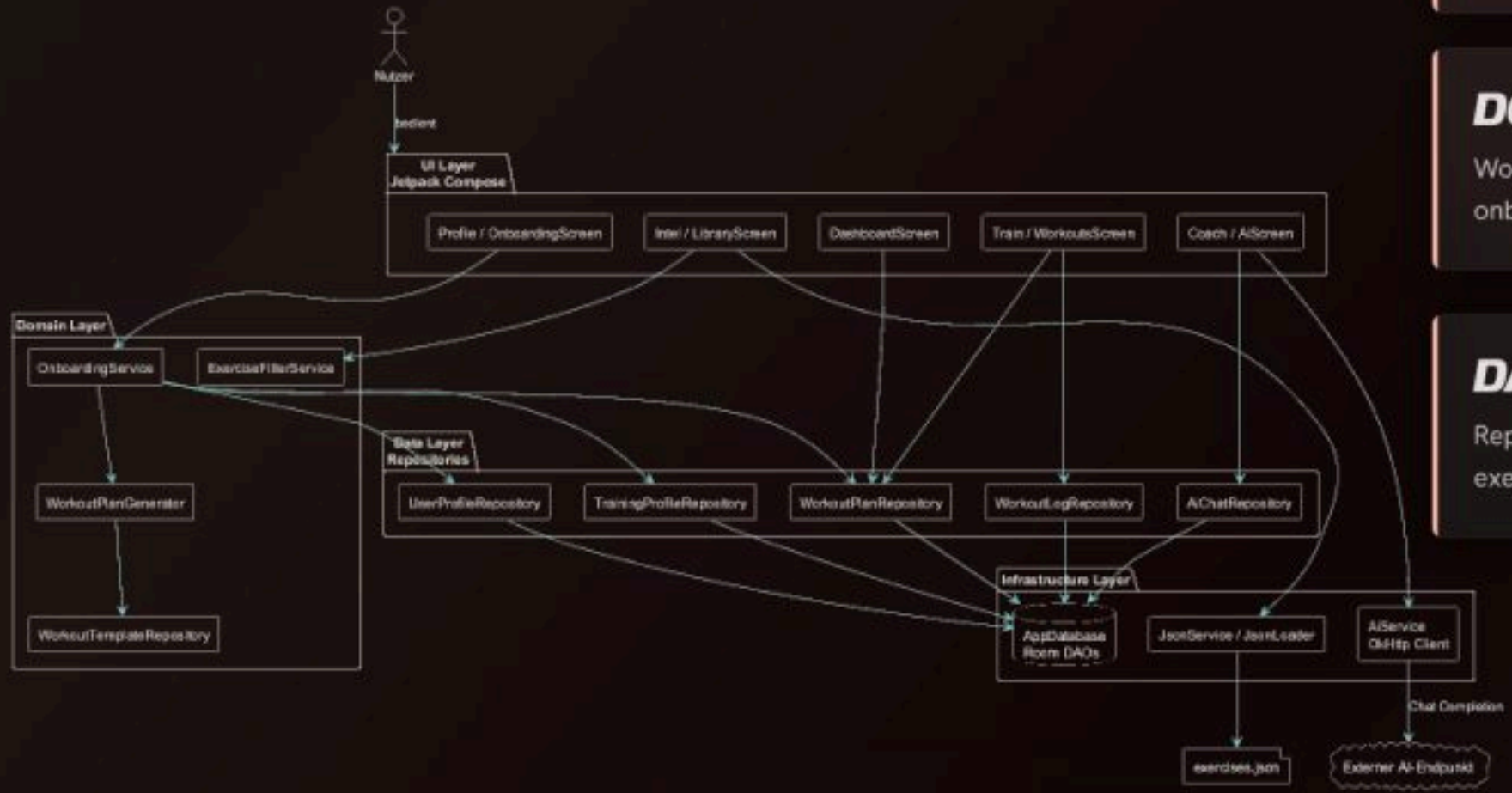
5

LOGGED STATE

Plans, logs and chat sessions persist locally for future use.

SYSTEM OVERVIEW

SOFTWARE ARCHITECTURE



UI LAYER

Jetpack Compose screens for Dashboard, Train, Intel, Coach and Profile.

DOMAIN / SERVICE LOGIC

Workout plan generation, exercise filtering, onboarding logic and AI orchestration.

DATA / INFRASTRUCTURE

Repositories connect to Room, JSON-based exercise data and the external AI endpoint.

IMPLEMENTATION RATIONALE

WHY THE ARCHITECTURE LOOKS LIKE THIS

LOCAL PERSISTENCE

Profiles, generated plans, workout logs and chat sessions are stored locally in Room to keep the prototype self-contained.

EXERCISE KNOWLEDGE

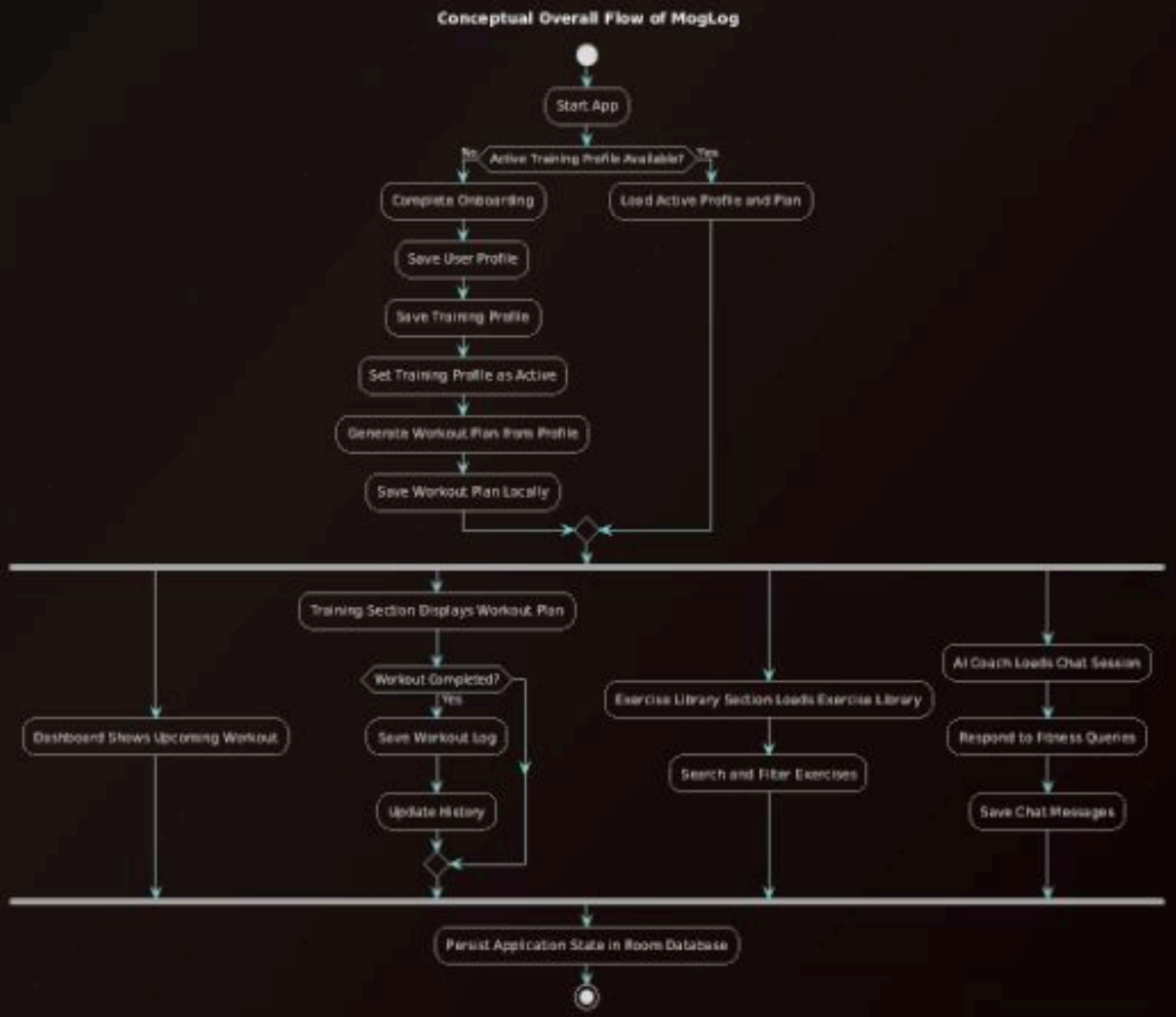
The exercise library comes from local JSON assets and can be filtered without backend calls.

ANDROID APP

- Compose UI
- Domain services
- Repositories
- **Room + JSON state**

EXTERNAL SERVICE ONLY WHERE IT ADDS VALUE

The AI Coach uses an external endpoint, but only as a focused feature. The main app logic stays on-device.



1 STARTUP CHECK

At launch, the app checks whether an active training profile already exists.

2 PROFILE-DRIVEN PATH

If no profile exists, onboarding runs first. Otherwise, the app directly loads the active profile and plan.

3 SHARED STATE FOR ALL MODULES

Train, Intel and Coach all use the same persisted state, which keeps the experience connected.

SYSTEM OVERVIEW

ONBOARDING LOGIC

INPUT

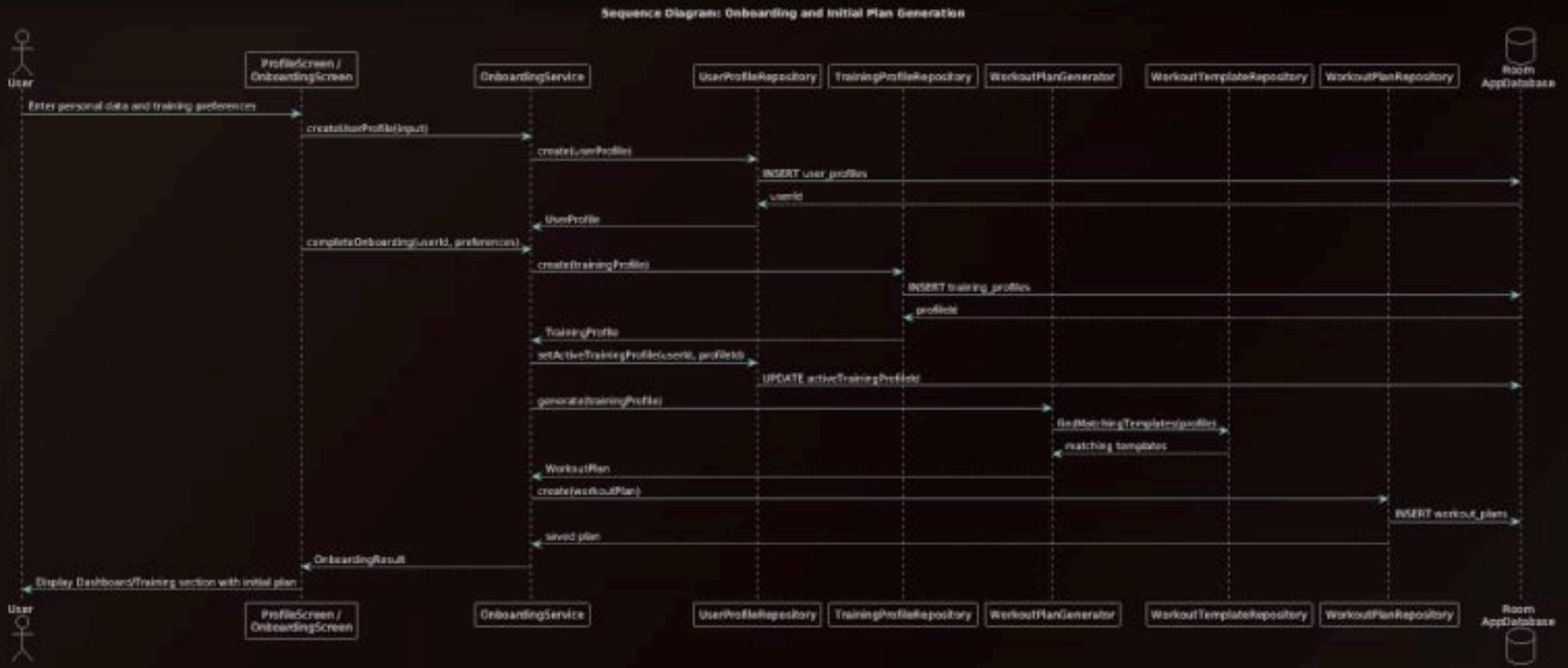
The user provides profile and training preferences.

TRANSFORMATION

The app stores a user profile and turns training choices into a structured training profile.

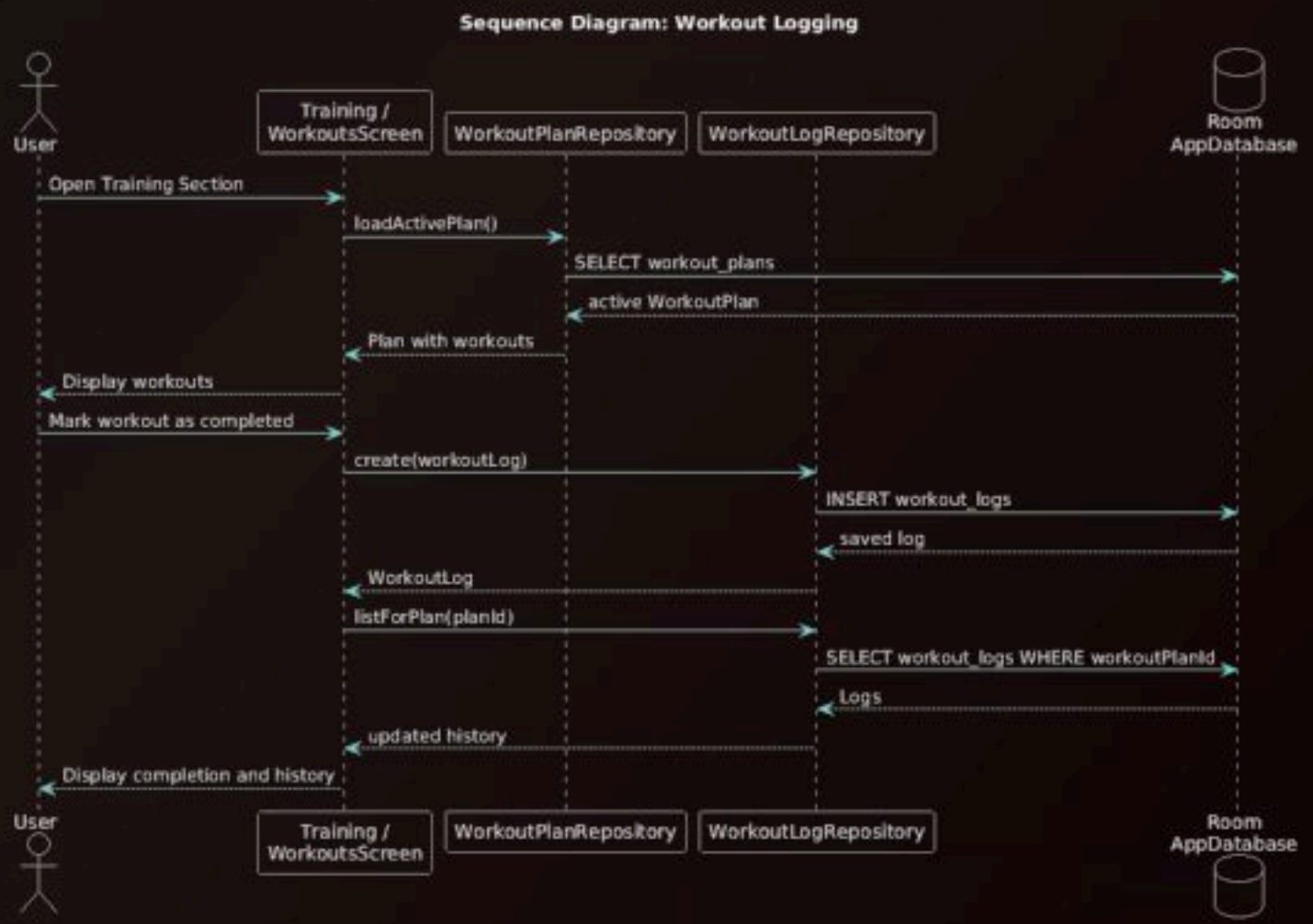
OUTCOME

That active training profile is passed to the generator, which creates the initial plan and stores it in Room.



SYSTEM OVERVIEW

WORKOUT LOGGING LOGIC



LOAD CURRENT PLAN

The Train area first retrieves the active plan from the repository.

WRITE TRAINING EVENTS

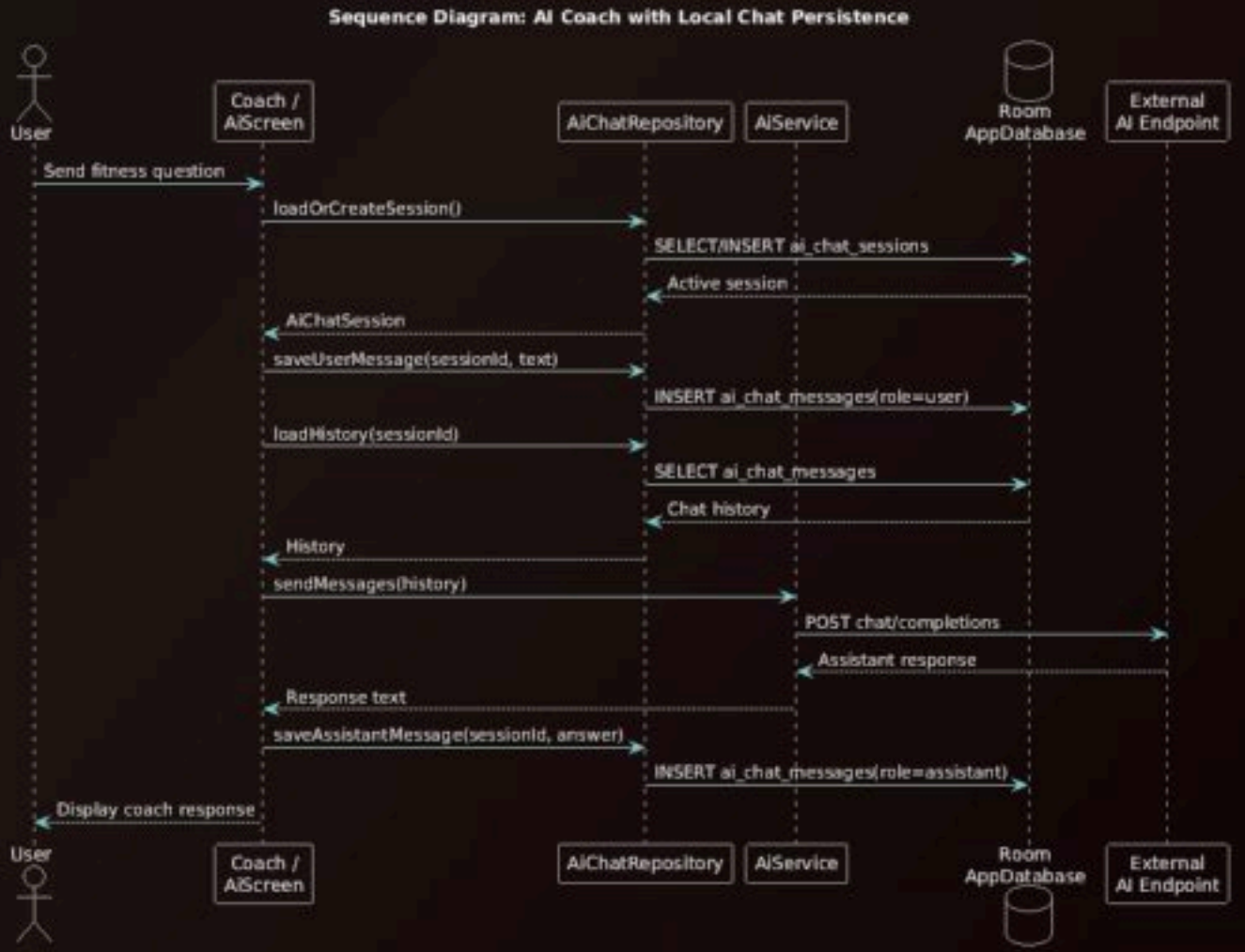
Completing a workout creates a WorkoutLog with time and progress-related values.

ENABLE FUTURE ADAPTATION

These logs already exist in the prototype and are the basis for later statistics and adaptive planning.

SYSTEM OVERVIEW

AI COACH LOGIC



SESSION-BASED CHAT

The Coach view loads or creates a chat session instead of acting like a one-off prompt box.

CONTROLLED EXTERNAL CALL

Messages are stored locally first, then sent to the AI service and external endpoint.

VALUE FOR THE USER

The answer is shown in-app and saved, giving the user persistent coaching support connected to the training context.

SYSTEM OVERVIEW

TECHNOLOGY STACK

FRONTEND

- Kotlin
- Jetpack Compose
- Material 3

PERSISTENCE

- Room / SQLite
- JSON exercise data

SERVICES

- Repositories
- Workout generator
- Exercise filter service

AI / NETWORK

- OkHttp
- External AI endpoint

WHY THIS STACK?

It supports a modern native Android prototype, clean separation of concerns, local persistence for core features and a lightweight path to integrating AI where it is actually useful.

KEY DESIGN DECISIONS

LOCAL-FIRST *PROTOTYPE*

No heavy backend was needed to demonstrate personalization.

This keeps the prototype robust and
easier to demo live.

RULE-BASED PLAN *GENERATION*

Further prototyping deterministic plan
generation is easier to validate than
opaque recommendation logic.

It clearly shows how user input changes the output.

AI AS SUPPORT – *NOT THE CORE*

The coach helps interpret and support
training.

The main product value still comes from structured profiles, plans and logs.

MAIN CHALLENGES

01

PERSONALIZATION LOGIC

Logic had to do more than collect answers; it required dynamic routing and real-time state evaluation to synthesize actionable protocols.

02

SYSTEM STRUCTURE

Several disparate modules had to work synchronously, demanding a rigid architectural framework to prevent data bottlenecks.

03

RIGHT PROTOTYPE SCOPE

Critical decisions were made regarding immediate implementation versus future architectural scalability to hit launch targets.

PHASE 05 // SYSTEM EVALUATION

LIVE APP DEMO

READY FOR ACTION

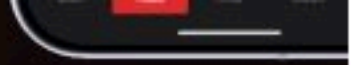
Initiate the live prototype sequence. All systems calibrated for peak performance demonstration.

START PROTOTYPE WALKTHROUGH →

● SYSTEMS ONLINE ● DATA SYNCED







TAKEAWAY

CONCLUSION & OUTLOOK

MOGLOG DEMONSTRATES HOW PERSONALIZED FITNESS PLANNING CAN BE STRUCTURED IN A NATIVE ANDROID PROTOTYPE.

The project goal was achieved: user data is actively transformed into training context, not just stored. The next logical steps are deeper progress tracking, adaptive planning and a more production-ready service architecture.

FINAL TAKEAWAY

- Clear vision
- Coherent architecture
- Meaningful prototype scope
- Demoable end-to-end flow