

Fratzeskos Dafereras

Product Manager · Product Owner · Business Analyst

The complete portfolio, exactly as it reads online: three roles, three products, and the artifacts behind them. Every page of the site is reproduced here, with the diagrams included.

fratzeskosdafereras.com

fratzeskos.dafereras@gmail.com · +30 697 253 6609

Dublin, Ireland · open to remote in Europe

CONTENTS

What is in this document

Home	Fratzeskos Dafereras	3
Job Experience	Netcompany-Intrasoft	17
Job Experience	Bespot	25
Job Experience	National Defence College	33
Project	EzyFlat	38
Project	OptimaLifts	48
Project	Flashcards	63
Get in touch	Contact	72

● Open to Product Manager · Product Owner · Business Analyst roles

Turning business needs into working solutions.

I'm **Fratzeskos**, Product Owner & Business Analyst at **Netcompany-Intrasoft**, where I own the backlog across **5 Agile teams** on a five-year EUIPO modernisation programme, and co-founder of **EzyFlat**. With a background in Management Science & Technology from the Athens University of Economics & Business, I enjoy turning opportunities into practical solutions and figuring out how to make them happen. I work at the point where stakeholders, requirements and delivery meet, and that is the part of the job I enjoy most.

Product discovery

Requirements analysis

Backlog ownership

Scrum & Agile delivery

Roadmap planning

Feature prioritisation

Product Owner & Business Analyst

Netcompany-Intrasoft

The link between an EU institution's needs and five delivery teams

Analysing requirements and owning the backlog



Worked at Netcompany-Intrasoft Bespot Programmes ACEin, AUEB JA Greece

Athens University of Economics & Business

Career

Job Experience

Three roles in three very different environments: an EU institution, a product startup and a military IT office. Each has its own page with the detail of what I did.

Product Owner & Business Analyst

Nov 2025 - Present

Netcompany-Intrasoft

Client: European Union Intellectual Property Office (EUIPO)

I own the backlog across **5 Agile teams (50+ people)** on a five-year programme to modernise the EUIPO information system with automation and AI. I act as the single channel between the client and the delivery teams. The clearest result so far is an AI capability for trademark examiners that reached production in **3 months** and passed UAT in a single cycle.

Backlog ownership

Requirements analysis

Multi-team coordination

AI & automation

Daily stand-ups

Stakeholder alignment

Product demos

Jira & Confluence

5

Agile teams

50+

people

30+

releases

5

year programme

Associate Product Manager Intern

Mar 2024 - Jul 2024

Bespot

Retail location-intelligence startup

Owned a new product from market research through to shipped user stories, supported the existing product, and built an internal improvement-request system end to end. That included a custom prioritisation model and a four-dimensional bubble chart the product team used to identify quick, high-impact wins.

Market research

Product documentation

Prioritisation model

User stories

Agile ceremonies

Data visualisation

2

products supported in parallel

3

subsystems built solo

3

teams

10+

people

IT & Network Administrator

Jan 2025 - Oct 2025

National Defence College

Compulsory military service, IT office

Ten months running the IT function for an entire facility, covering networks, hardware, software and data. Every request went through Jira, where I triaged and assigned it. I maintained dashboards that made the state of the infrastructure visible at a glance, and worked with the team on scripts that automated the repetitive data work.

Team coordination

Jira intake & triage

Monitoring dashboards

Windows Server

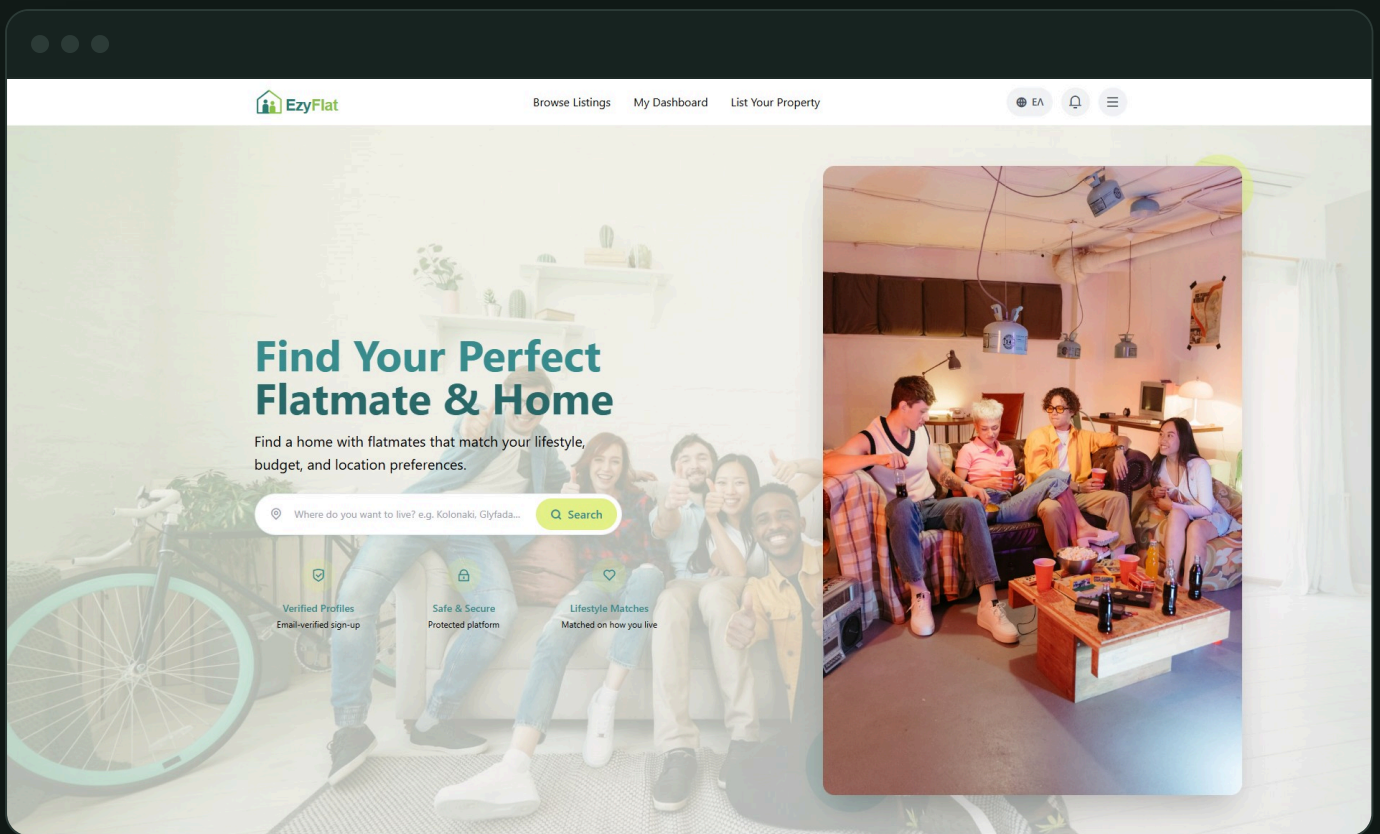
pfSense

Scripted automation

Projects

Selected project work

A live marketplace I co-founded and still run, a computer-vision startup I took through an incubator, and a full software development lifecycle project from my degree. Each has a detailed case study.



01 · Live in production

EzyFlat

Co-founder & Product Lead · Nov 2025 – Present

From the initial idea through to launch, I took ownership of the product: the data model, the feature set, and a prioritisation sequenced against liquidity, trust and launch risk rather than what was most interesting to build, backed by market research and a metric set defined before anything shipped.

Requirements analysis

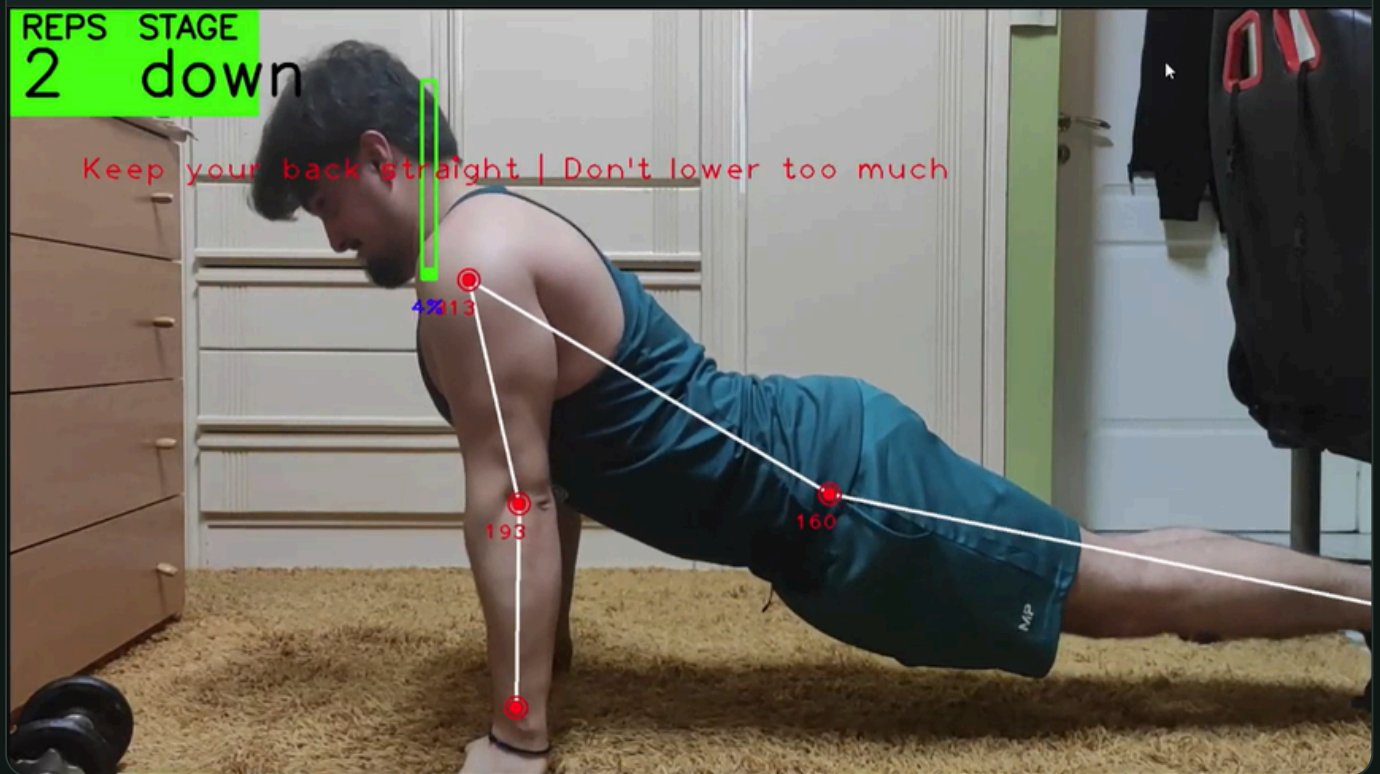
Feature definition

Product specification

Strategy & planning

Market research

Launch readiness



02 · Startup, idea to MVP

OptimaLifts

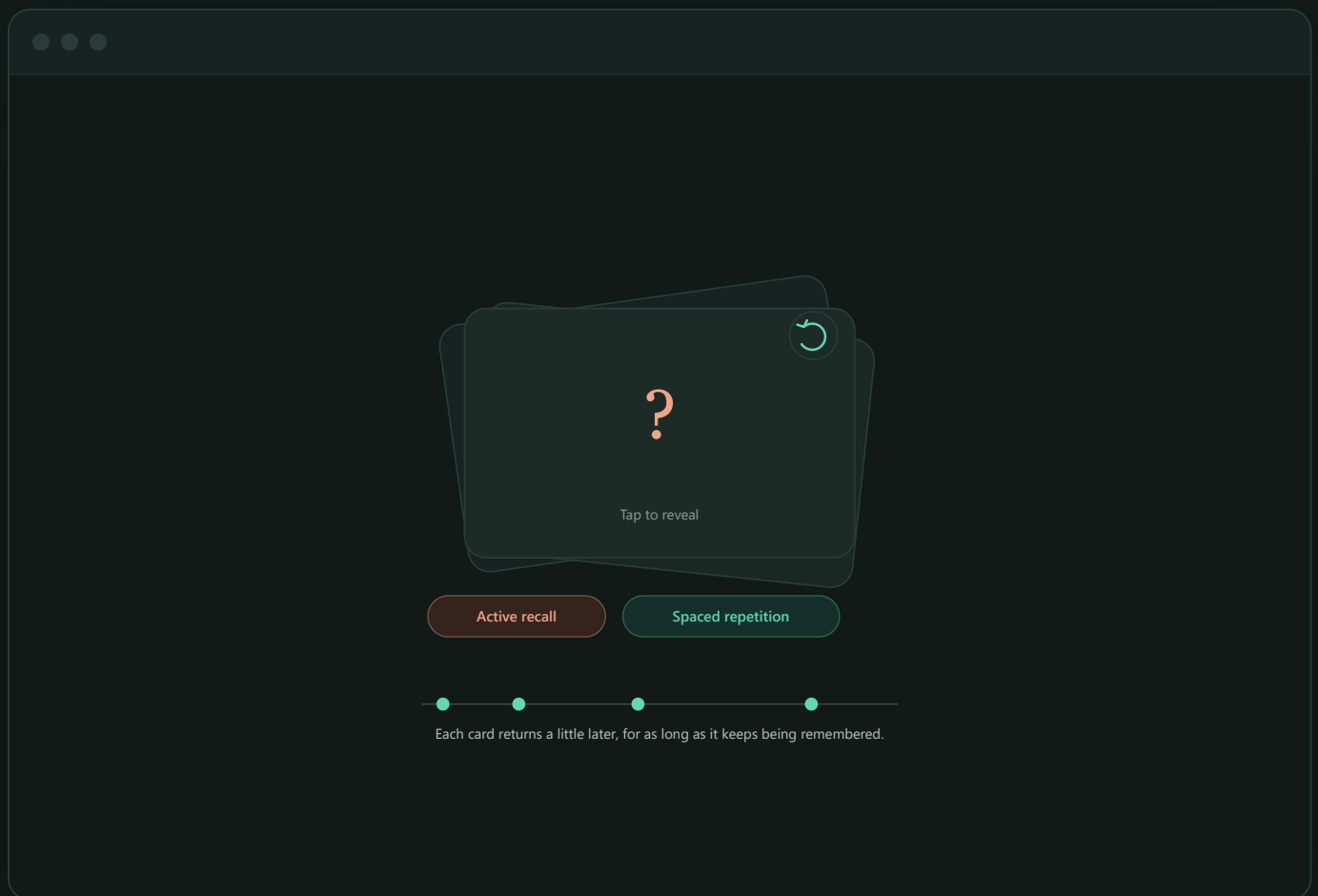
Co-founder & Product Designer · Sep 2023 – Jul 2024

A workout assistant that uses the phone camera and computer vision to count reps and correct form in real time. I owned the vision, the roadmap, the user stories and the business model. I also moved the product from wearable hardware to a software-only build once the unit economics stopped working, and ran the hiring process for our computer-vision engineer.

1 yr ACEin incubator track

3 mentors helping the team

Finalist JA Greece Startup



03 · University project

Flashcards

Requirements, data & backend, team of 4 · AUEB, 2024

A study app where the first decision was a requirements one: which evidence-based learning methods, active recall and spaced repetition, to build around, before any code was written. Working in a team of four, I covered use cases, the data model and a structured set of test cases.

4 team members

2 learning-science mechanics built in

4 core entities modelled

5 SDLC artifacts delivered

[More work](#)

Other projects

Selected university projects

Business analysis

Morgan Stanley, digitising HR workflows

A full analysis and design cycle on the appraisal, training and motivation processes of a Morgan Stanley desk in London. The work covered problem framing based on real employee feedback, actor and stakeholder analysis, a requirements-to-use-case traceability matrix, screen design, a costed feasibility study, and a rollout plan that started with a pilot group to manage the change resistance we expected.

Requirements

UML use cases

Feasibility study

Change management

Process analysis

KFC, business process management

End-to-end analysis of a KFC restaurant as a work system, covering customer segments (in-store, online, delivery aggregators), the CHAMPS operating standard, the order-management technology stack, the full role hierarchy from restaurant general manager down to team members, and the processes that connect them. Modelled in BPMN and ARIS.

BPMN

ARIS

As-is modelling

Work system analysis

Data analysis

Supermarket point-of-sale segmentation

Clustering on real transaction data to build customer segments and market basket associations using RapidMiner and Excel, then translating the clusters into merchandising and promotion recommendations that a category manager could act on.

RapidMiner

Clustering

Basket analysis

Excel

Entrepreneurship

City Vault, Start for Future

Designed and pitched a business idea for an international entrepreneurship programme, backed by a strategic business plan and a validated business model.

Business planning

Pitching

Business modelling

Skills & tools

What I work with

The methods and tools I have used on real projects, grouped by the kind of work they support.

Product & delivery

Jira Confluence Scrum Kanban

Backlog refinement Roadmapping

User stories Acceptance criteria

Release planning Product demos

PRD writing Prioritisation frameworks

Analysis & modelling

SRS analysis BPMN ARIS

UML use cases As-is / to-be

Requirements engineering

Stakeholder analysis Feasibility studies

Competitive audits

AI / automation products

Data & insight

SQL MySQL SQL Server SQLite

Power BI RapidMiner SPSS

Google Analytics Excel MATLAB

Design & build

Figma Next.js / React Python Java

JavaScript Visual Studio Claude Code

Salesforce SoftOne ERP Git

Education and certifications

2019 - 2024

BSc Management Science & Technology

Athens University of Economics & Business, majoring in Information Systems & e-Business. A degree that sits between business and engineering, which is the ground product roles are built on.

2017 - 2019

BSc Electrical & Computer Engineering

National Technical University of Athens. I chose to discontinue it and move to a degree closer to products, systems and business.

Certifications & programmes

Artificial Intelligence and Machine Learning, UCERT

2026 · Code UGRSPUICT325726

Compass Cycle, year-long incubator programme, ACEin (AUEB)

2023 - 2024

Start for Future, City Vault, international entrepreneurship programme

2022

Website Development, Grow with Google

2020

Development of a Digital Marketing Strategy, Grow with Google

2020

Entrepreneurship, Personal Branding & Funding Sources, Mindspace & Infinity Greece

2020

First Aid at Work, The London First Aid Training, London

2017

Competencies in PLC Automation Systems, Aemifesa, Barcelona

2016

Languages: Greek (native) · English (C2, fluent)



How I ended up in product

I started in electrical engineering, then switched to Management Science & Technology at AUEB once I realised the problems I actually cared about were not circuits, but **systems and the people using them**. Every project since has come back to the same question: someone has a need, someone else has a team, and my job is to make the space between them small enough that the team can just build.

OptimaLifts, which I co-founded during my studies, is where I learned first-hand that a well-built business model can still be the wrong one: we reached a national final, went through **ACEin's year-long incubator track**, hired our computer-vision engineer, and still had to walk away from the original product. **Bespot** is where I saw the other side, joining as a product intern to learn how product management works inside an established team with a real backlog and real ceremonies.

The discipline I care about most is holding a direction long enough for it to prove itself, while staying honest enough to change it the moment the evidence says otherwise.

Today I'm a **Product Owner & Business Analyst at Netcompany-Intrasoft** and co-founder of **EzyFlat**, which gives me both sides of the job at once: the enterprise discipline of stakeholders, ceremonies and release cadence, and the founder reality of deciding what to build with nobody telling you. I'm looking for a product role where I can bring both of those strengths together.

How I work

- 1 **Understand the problem first**, with research into the market before deciding anything, as on EzyFlat.
- 2 **Define what to build** as use cases, user stories and acceptance criteria, as on Flashcards and OptimaLifts.
- 3 **Decide what comes first** with a clear prioritisation model, as at Bespot.

4 **Deliver with the team** through daily stand-ups, early demos and an accurate backlog, as at [Netcompany-Intrasoft](#).



Dublin, Ireland

Open to remote in Europe



EN C2 · EL native

Working comfortably in both



PM · PO · BA

Roles I'm targeting

← Job Experience

Netcompany-Intrasoft · Current role

Connecting an EU institution and five delivery teams

I own the backlog across five Agile teams on a five-year programme to modernise the EUIPO information system with automation and AI. Requests from the client and questions from 50+ delivery people come through me. The clearest proof that this works is an AI capability that went from basic requirements to production in three months.

Company

Netcompany-Intrasoft [↗](#)

Client

EUIPO, EU Intellectual Property Office

My role

Product Owner & Business Analyst, the single channel between the client and the delivery teams.

Scale

5 Agile teams, 50+ people, 30+ releases

Timeline

Nov 2025 – Present

Outcome

Signs Comparer reached production in three months and passed UAT in a single cycle, and the client asked for the same working model on later projects.



WHY A SINGLE FUNNEL

Nothing reaches the team unclarified, and nothing reaches the client unverified. One person holding both sides means the shared understanding cannot quietly drift apart over a five-year programme.

The context

The client is the **European Union Intellectual Property Office (EUIPO)**, the EU agency responsible for registering and managing trademarks and designs that are valid across every EU member state. Its information system is what examiners, legal staff and applicants across the whole of the EU rely on to handle those cases, which is the scale the modernisation programme has to work at, on a **five-year plan** rather than a quarterly roadmap.

What we are adding to that system is automation and AI, which changes how the institution's own people work day to day.

The core challenge

Two groups that both know a great deal about completely different things, and cannot afford to misunderstand each other over five years. My role sits between them.

Signs Comparer: a selected delivery, from basic requirements to production in three months

Signs Comparer was the first project on the programme where I took full ownership of the delivery end to end. The results were strong enough that the client asked us to carry the same working model into the projects that followed.

Signs Comparer is an AI-driven capability that assists EUIPO examiners with **trademark opposition and cancellation cases**. This is judgement-heavy work, so the tool has to fit the way examiners actually reason or it will not get used.

My role on it was the same one I carry across the programme: analysis and clarification, so the team could build without guessing. There were many open points, and the client did not always know exactly what they wanted going in, so I drew it out by sharing **development updates and UI proposals** with the business users continuously, well before anything was final, and using their feedback to close out clarifications. Misunderstandings surfaced while a screen was still a proposal, not a built feature.

1

dedicated team, 10 people

3 mo

basic requirements to production

1

UAT cycle to sign-off

WHY IT SHIPPED IN THREE MONTHS

The usual way

Business users first see the product at UAT, by which point every misunderstanding is already built.

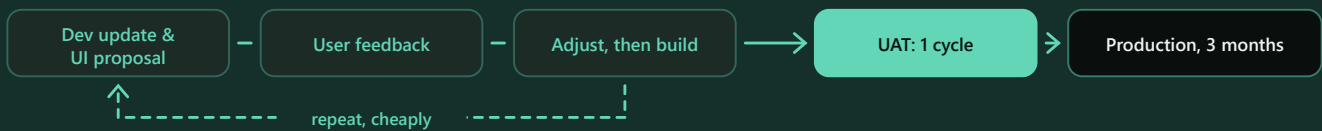


Every correction cycle costs a round trip through the whole team, and the date moves.

Feedback arrives at the end, when it is most expensive to act on.

What we did on Signs Comparer

Dev updates and UI proposals went to the business users continuously, so a misunderstanding was caught as a proposal, not as shipped code.



Feedback arrives while it is still free to act on. UAT stops being a discovery phase and becomes a confirmation.

The client asked that upcoming projects follow this exact same working model.

The same amount of feedback, collected at the point where acting on it is still cheap.

What that produced:

- A workflow that stayed **exceptionally lean** and closely aligned to real user needs.
- **UAT completed in a single cycle** with only minor issues, because the business users had already seen and shaped every screen by that point.
- **Production in three months**, a record for this client given the size of the programme.

The result that mattered most

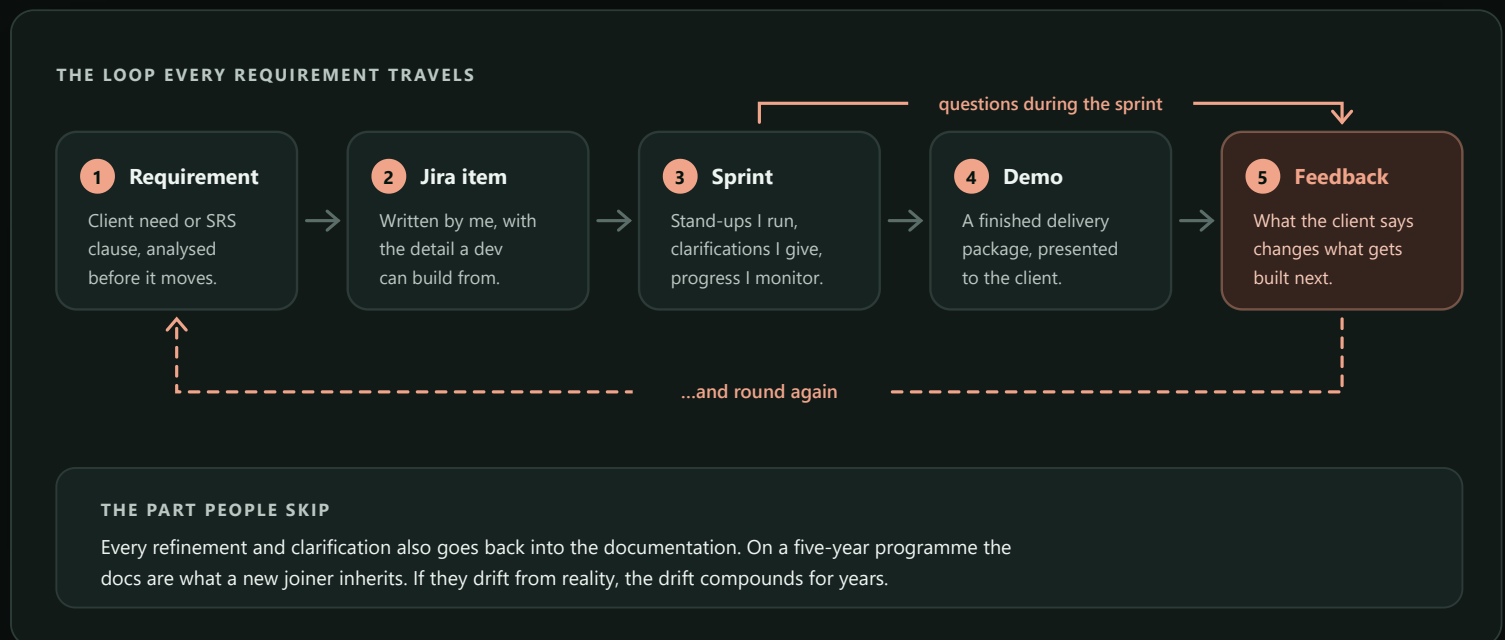
Delivering quickly is one thing. More significant was that the client saw how efficiently it went and **specifically asked that upcoming projects follow the same direction and working model**. The approach outlasted the project itself.

What I own across the programme

The backlog, across five teams

Every work item in Jira that comes out of a product requirement is mine to open, write and track, across **five Agile teams of 10+ people each**. This is where a client's high-level specification, general and light on detail by design, gets turned through analysis and clarification into something a developer can build without guessing. I know what is in the backlog, what is coming next, and where each delivery stands at any given moment, because when someone asks, I need to have the answer.

At this scale, keeping the backlog accurate is what keeps five teams and the client working from the same picture of what matters next and what is already done.



The loop every requirement travels, including the documentation update that most teams skip.

Analysing requirements and carrying them to the team

I study and analyse all the SRS and product documentation closely enough to hold a clear picture of the system. New requests from the client come to me first. I analyse them, work out what they imply and carry them to the team, and when a developer's question is about something we specified, the documentation usually settles it straight away. Working closely with the Project Manager keeps the commercial and delivery views of the programme aligned.

Turning requirements into Jira items

Once a requirement is clear, I write it up as Jira work items with the detail a developer can build from, so the team can pick them up in a sprint without having to guess what was meant.

Daily stand-ups, questions and client feedback

I run the daily stand-ups with all five teams and the weekly report meetings with the client. I manage the sprints and the Jira boards, and keep the dashboards current so that progress monitoring takes a glance rather than a meeting. Release planning and delivery monitoring I handle in partnership with the Project Manager. For me, facilitating means making sure a difficult point gets raised early enough to do something about it.

Questions from the team come back the same way: I answer from what I already know, or I find the person who can. I then report back to the client, who gives feedback while the work is still in progress, so we can make improvements while there is still room to adjust.

Demos and client feedback

When a delivery package is complete I present the demo to the client. What comes back is the list of things we understood slightly differently from how they intended, and we make the corresponding changes.

Documentation

When a refinement or clarification changes what is true, I update the documentation. On a five-year programme, the documentation is what the next person inherits.

Where the product work sits

Passing a message along is straightforward. The value sits in the analysis in the middle: deciding what a request really means, what it costs, what it conflicts with, and what has to be true before it can be built. That judgement is the core of the role, and the meetings are where it shows.

Day-to-day responsibilities

Client and stakeholders

- **Client and stakeholder management and alignment**, keeping expectations and reality in the same place.
- **Client needs analysis and product requirements alignment**, keeping what the client asked for and what ends up specified in the same place.
- **Product demos and stakeholder feedback**, and the modifications that follow.
- **Daily stand-up facilitation and client reporting**.

Requirements

- **Requirements analysis and clarification**, including the questions nobody thought to ask.
- **Translating requirements into Jira items** that a developer can build from.
- **Updating documentation** based on every refinement and clarification.

Delivery

- **Backlog management, sprint management and progress monitoring**.
- **Release planning, delivery management and monitoring**.
- **Jira boards and dashboards management**.

Collaboration

- **Cross-functional team collaboration** with developers, QA, architects and the Project Manager.

← Job Experience

Bespot · Product internship

Owning a new product, supporting an existing one, and building an internal system from scratch

I owned a new product from market research through to shipped user stories, supported the existing product, and designed and built an internal improvement-request system end to end, including the prioritisation model behind it.

Company

Bespot [↗](#)

Context

Retail location intelligence

Problem

Improvement ideas across the company were not being captured effectively, and a new in-app feedback product needed an owner.

My role

Associate Product Manager Intern, primary owner of the new product.

Team

3 teams, 10+ people, working with the Lead Product Manager and the dev teams

Timeline

Mar 2024 – Jul 2024

Outcome

The product team used the internal system and its bubble chart to decide what to build next.

What this internship covered

I worked directly with the Lead Product Manager on strategy and decisions for both the existing product and a new one, was made primary owner of that new product, and shipped an internal system that the product team then used to decide what to build next.

2

products supported in parallel

3

subsystems built solo

4

variables in a custom evaluation model

3

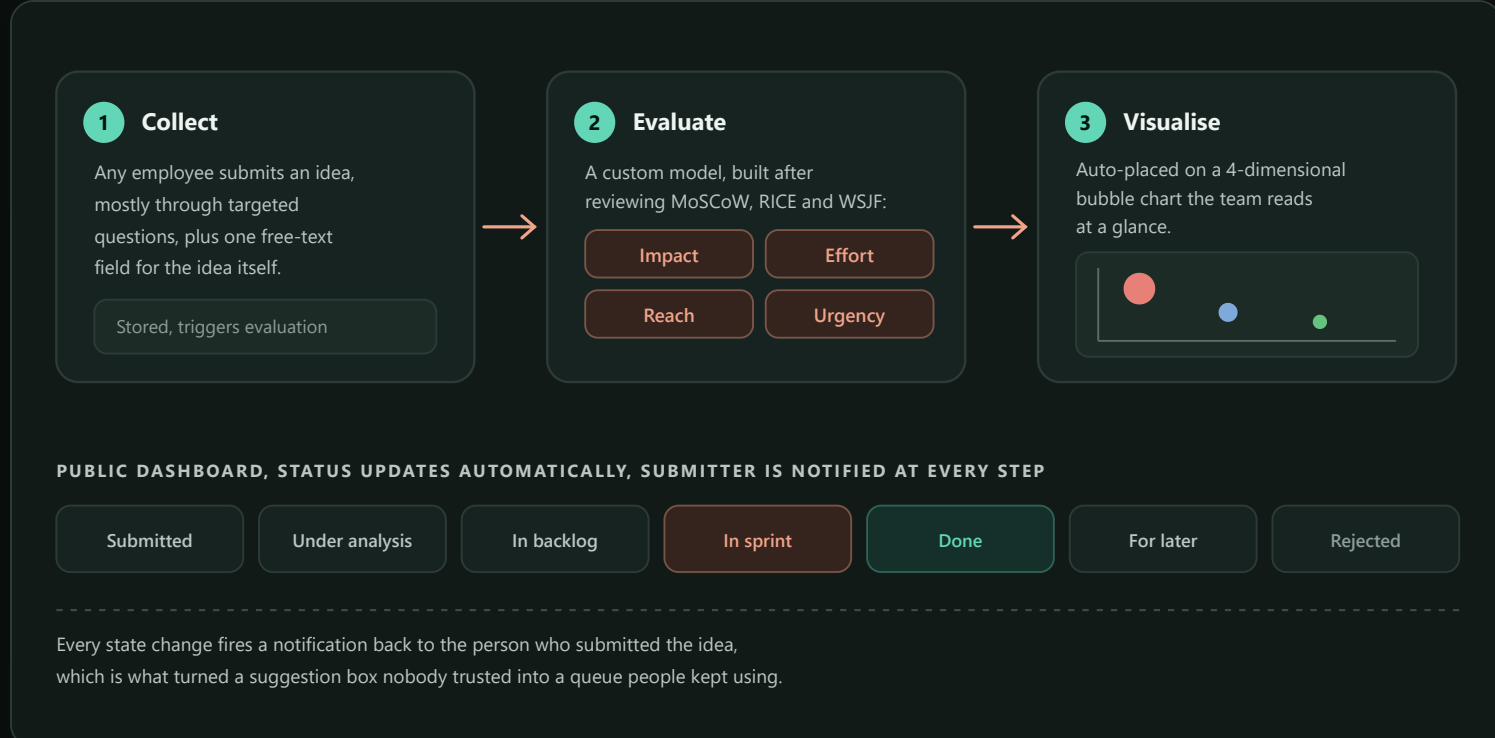
teams I worked with

10+

people on those teams

The internal system I built end to end

People across the company had ideas about how to improve the product and the way we worked, but the way of capturing them was not effective. The Lead Product Manager wanted a new system that would not just do that work more effectively, but increase the number of requests coming in, which meant the system had to actually attract the employees using it far more than the old one did. Building it was assigned to me and I delivered the whole thing: **three subsystems** covering collect, evaluate and visualise, with extensive feedback from the end users throughout and a study of how similar solutions handle the problem.



Collect, evaluate and visualise, plus the public board where every request's state is visible.

Collect

A form built mostly around **targeted questions**, so that most of every submission arrived structured enough to be evaluated, plus one open field where the user described the idea or problem in their own words. Submitting stored the request and automatically triggered the next stage.

Evaluate, using a custom prioritisation model

There are established models for this, including MoSCoW, RICE and WSJF. I studied them and none fit the specific needs of this project cleanly, so I built a **custom model combining variables from several of them**, had it approved by the Lead Product Manager, and applied it to every request. The four variables were:

Variable	What it asks
Impact	If we did this, how much better would it actually get?
Effort	What does it cost us to build?
Reach	How many people does it affect?
Urgency	Does delaying this cost us something?

Choosing a custom model over RICE

I did not build a custom model out of a preference for building things from scratch. The Lead Product Manager had already told me what he wanted to know quickly about each request, and what should weigh most when judging one, so I had a clear brief to follow rather than a blank page. That brief is what led to the blend of variables above: built for what mattered to us, so the visualisation that followed could foreground exactly that.

Visualise, with four dimensions on one chart

Once evaluated, a request was automatically placed on a **four-dimensional bubble chart**. Each variable maps to a visual property, so the prioritisation can be read at a glance rather than sitting in a spreadsheet nobody opens.

Dimension	How it's drawn
Reach	Bubble diameter, so a bigger bubble means more people affected
Effort	X axis, so the less effort required the further left it sits
Impact	Y axis, so the more impact it has the higher it sits
Urgency	Colour: green for low, blue for medium, red for high

The result is a chart the product team could read in seconds. A bubble that is **top-left, large and red** has high impact and wide reach, is urgent, and costs little to build, making it a quick and significant win. That single reading rule is why the chart was adopted.

A public dashboard, so nothing was lost

Every request also appeared on a **public kanban board** whose status updated automatically as the request moved through *submitted*, *under analysis*, *in backlog*, *in sprint* and *done*, plus *for later* and *rejected*. Each time a request changed state, the person who submitted it was notified and could follow the progress themselves.



How user feedback shaped the solution

The public kanban was not in the original brief, it came out of the feedback I collected from users during the project. Several people told me one of the biggest reasons they had stopped submitting new requests to the **old system** was that they never knew what happened to the ones they had already sent, and felt like they got little back even when their idea did lead to a real change in the product. That was the answer to the business need behind the whole system: collect more requests, and get employees actually using it. The kanban and its notifications solved exactly that: users could see where their request stood at any point, and watching it move gave them a sense that it was being taken seriously and was actually shaping the product, which is what brought people back to using the system.

A new product I owned

The product was an **in-app feedback collection system**. After a user completed specific actions in the client's application, it captured what they thought while the experience was still fresh. I was assigned as primary owner.

The work, in the order it happened:

- **Extensive market research.** I looked at how other companies already solve this and built real domain knowledge before forming an opinion, because an opinion formed too early is hard to move away from later.
- **A catalogue of competitor features and capabilities**, followed by the harder step of deciding which of them actually fit *our* case rather than copying the longest list.
- **A minimum feature set** assembled from that analysis, covering the smallest version that would genuinely work.
- **Value-adding features on top**, chosen from user research rather than from the competitor list. This is what decides whether a product is a copy or a deliberate choice.

That analysis became the basis for the **product documentation**, which I co-authored with one other person, creating the BRD and the requirements behind it. From those documents I produced the **user stories in Jira** and assigned them to the relevant teams, which is the point where the product stopped being a document and started being built.

From there I followed development closely to support the team, made the necessary modifications during backlog refinement and sprint planning, and kept the documentation updated as the product changed. The team ran full Agile Scrum and I took part in every ceremony.

FROM THE BRD TO ITEMS A TEAM CAN BUILD

The in-app feedback product. Each branch names how its stories were sliced, because the reason is the part worth showing.

BUSINESS NEED, FROM THE BRD

Capture what a user thinks about an experience while it is still fresh.

EPIC 1 Ask at the right moment

Feature: trigger on a completed action

SLICED BY WORKFLOW STEP, THEN BY BUSINESS RULE

Prompt after a defined action completes

Never ask twice for the same action

Hold a cool-down between prompts

EPIC 2 Capture the response

Feature: the response itself

SLICED SIMPLE FIRST, RICHER VERSION LATER

Submit a rating

Add an optional comment

Dismiss without answering

EPIC 3 Get it to the team

Feature: make responses readable

SLICED BY OPERATION

See responses for one feature

Filter by action and by date

----- ABOVE: MINIMUM FEATURE SET, FROM THE COMPETITOR ANALYSIS

BELOW: VALUE ADDING, CHOSEN FROM USER RESEARCH, NOT FROM THE COMPETITOR LIST

Let a user turn prompts off

Follow-up question based on the rating

Notify the owning team

Ask only after repeated use of a feature

Attach a screenshot to a comment

Group responses by theme

HOW TO READ IT Business need → epic → feature → story. Every story is a slice of behaviour a user can see, never a technical layer.

Hierarchy note: the Scrum Guide defines only "Product Backlog item". Epic, feature and story are a working convention from XP and common tooling, and the levels above are the ones used on this product. Redrawn from the structure, not exported from Jira.

Above the dashed line is the minimum feature set assembled from the competitor analysis. Below it are the value-adding features, which came from user research rather than from the competitor list.

The existing product

Bespot's established product precisely tracks customer location inside retail stores. My role here was supporting rather than owning, with a specific responsibility: to know the product and the market well enough to *propose* modifications and optimisations, not only to process the ones handed to me.

A few additional responsibilities came with that:

- Backlog refinement for the product, and writing new user stories.
- Sprint monitoring.

- Working closely with the team to resolve questions and **bridge business and technology**, which is largely about making sure both sides mean the same thing by the same word.
- Taking part in demos, then structuring the conclusions from client feedback and discussing them with the Lead Product Manager to identify where we could improve and what should change.

What I took from it

- **User feedback can hand you the business need, not just a symptom to patch.** The public kanban did not come from a spec, it came from users explaining why they had stopped using the system at all.
- **Make the decision visible.** The same four numbers in a table would have been ignored. On a chart with one clear reading rule, they were used.
- **Do the market research before forming an opinion.** This is the habit that later shaped the competitor matrix at [OptimaLifts](#) and the competitive audit at [EzyFlat](#).
- **A prioritisation model has to reflect what the decision-maker already values.** Getting it explicitly signed off before applying it was worth more than any further refinement to the formula.

← Job Experience

National Defence College · Military service

Running the IT function of a facility for ten months

My compulsory military service was spent in the IT office of the National Defence College, responsible for the technology of the entire facility. I turned an ad-hoc support function into something with an intake queue, a triage model, a monitoring dashboard and, with the team, a set of scripts.

Company

National Defence College [↗](#)

Context

Compulsory military service

Scope

Whole-facility infrastructure & accounts

Timeline

Jan 2025 – Oct 2025

Problem

IT support for the whole facility ran ad hoc, with no shared view of the demand.

My role

IT & Network Administrator, managing the team in the IT office.

Team

The IT office team of the National Defence College

Outcome

An intake queue in Jira, a triage model, monitoring dashboards and, with the team, automation of the repetitive data work.

How this role relates to product work

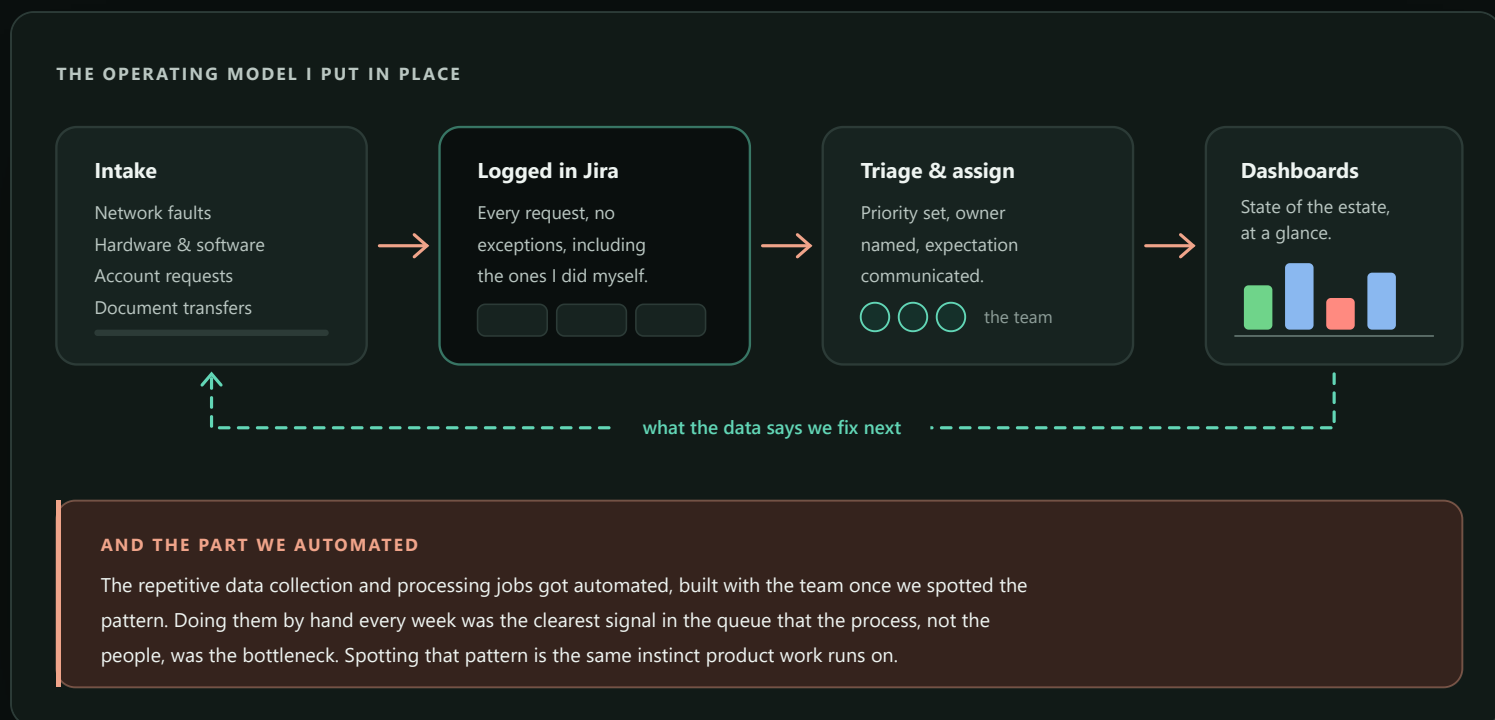
This is not a product role and I would not present it as one. It is here because of what it required me to build: a way to see all the demand at once, decide what mattered first, distribute it across people, and account afterwards for where the time went.

Scope of the role

I was responsible for supporting **all the technology infrastructure of the building**, covering the networks, the hardware, and the software and data issues that came with them. In practice that meant network maintenance and development, hardware and software troubleshooting, Windows Server administration and network security with pfSense.

I also **managed the team** staffing that department. That is the part that changed how I work. When several people could do a given job, deciding who does it, and being able to account later for what everyone did, becomes essential.

Everything went through the queue



I logged every piece of work that needed doing in **Jira** and triaged it before assigning it: how many people or systems it affected, whether someone was already blocked by it, and how much effort it would take to clear, rather than working the queue in the order things arrived. Then I distributed it to the right person, *including myself*. That last detail is what made it work: if the person running a queue leaves their own work out of it, the queue stops reflecting the real workload within about a week.

With everything in one place, I kept **dashboards** updated with the corresponding data so the state of the infrastructure and the networks could be monitored immediately rather than discovered later. The goal was to spot a pattern in the queue long before it turned into an outage, rather than to report upwards.

Accounts, machines and documents under protocol

I managed the accounts of all personnel at the facility, which meant being responsible for the security of those accounts and of the computers behind them. Document transfer was part of the role as well, carried out under **specific security protocols** rather than at my own discretion.

Working somewhere the rules are non-negotiable is useful preparation for working somewhere the rules are legal rather than military. GDPR obligations on [EzyFlat](#) and an EU institution's compliance requirements at [Netcompany](#) both felt familiar because of this.

The most useful constraint

Military service gives you a fixed team, no budget and no ability to choose your tools. Every improvement had to come from reorganising what already existed. That is far better training for prioritisation than an environment where a problem can be solved by spending on it.

What I automated

Some of the work, mostly **collecting and processing data**, was repetitive, time-consuming and identical every time. Once we spotted that as an optimisation opportunity, I worked with the team to build the automation that fixed it, rather than writing it alone from scratch.

Nobody asked for it. It came out of the queue. When the same task appears every week in the same shape, the queue is telling you that the process is the bottleneck rather than the people. Noticing that pattern and acting on it is the same instinct product work relies on, with the difference that here the users I was unblocking were the team and myself.

What I carried forward

- **Repetition is a requirement in disguise.** The tasks we ended up automating had been announcing themselves in the queue for months.
- **Instrument the thing before you are asked to report on it.** The dashboards existed so we could act early, which is a different design goal from dashboards that exist to be shown.

- **Improvement under a hard constraint is a skill.** With no budget and no choice of tools, the only lever left is how the work is organised, and that proved to be the most effective one.

← All projects

EzyFlat · Live in production

A flatsharing marketplace built around the household, not the listing

Finding a room in Greece still happens mostly in Facebook groups. EzyFlat is the product I co-founded to address that, and I co-own it with the rest of the founding team, from the problem statement to the release decision.

My role

Co-founder & Product Lead

Team

Small founding team

Timeline

Nov 2025 – Present

Outcome

Live in production at ezyflat.com.

Find Your Perfect Flatmate & Home

Find a home with flatmates that match your lifestyle, budget, and location preferences.

📍 Where do you want to live? e.g. Kolonaki, Glyfada...

🔍 Search



Verified Profiles
Email-verified sign-up



Safe & Secure
Protected platform



Lifestyle Matches
Matched on how you live



Demo data, for privacy reasons.

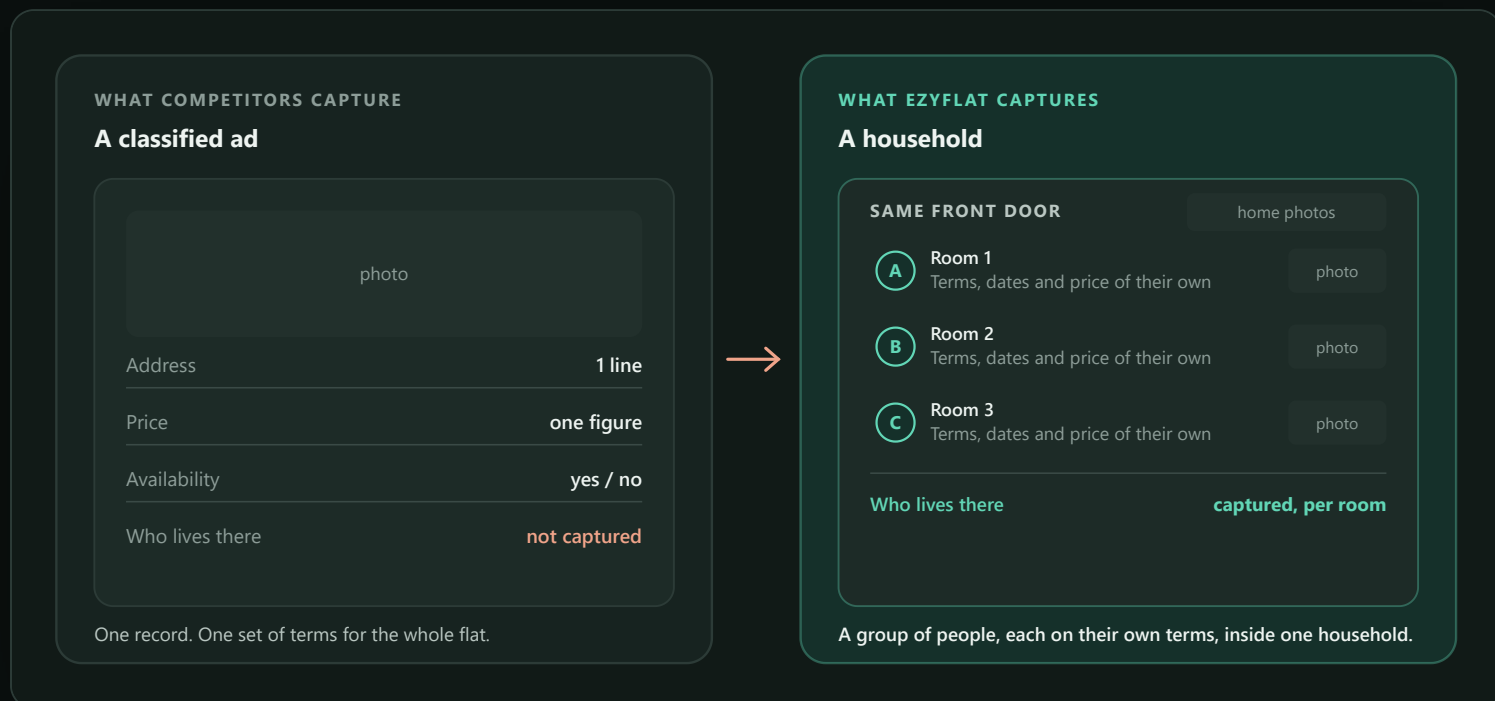
The problem

Shared renting in Greece has real demand but no product built around the household. Students and young professionals find rooms through Facebook groups, word of mouth, and generic real-estate portals designed to sell whole apartments. All of those channels share the same three weaknesses: you cannot filter properly, you cannot verify anything, and **you cannot find out who you would actually be living with** until you are standing in their hallway.

That third point is the one the existing products have not modelled. A room listing is a property record, but a flatshare is a *household*: a group of people, on a set of terms, for a set of dates, with different economics per room inside the same front door. I decided early that EzyFlat would model the household, and that this, rather than a nicer search page, would be the part that is difficult to copy.

The core product decision

Competitors model the classified ad. We model who lives there, on what terms and for how long, because that is the decision a renter is actually making.



A classified ad has one owner and one set of terms. A household has several, and EzyFlat models every one of them.

What I owned

On a founding team the boundary between product and everything else is thin. In practice I own:

- **Problem and scope:** defining the problem, the need behind it, and how solving it connects to the business, plus what is in the MVP and, just as importantly, what is explicitly out of it.
- **Requirements:** product requirement and feature definition, the data model, entity relationships, status state machines, and the business rules that have to hold on every screen.

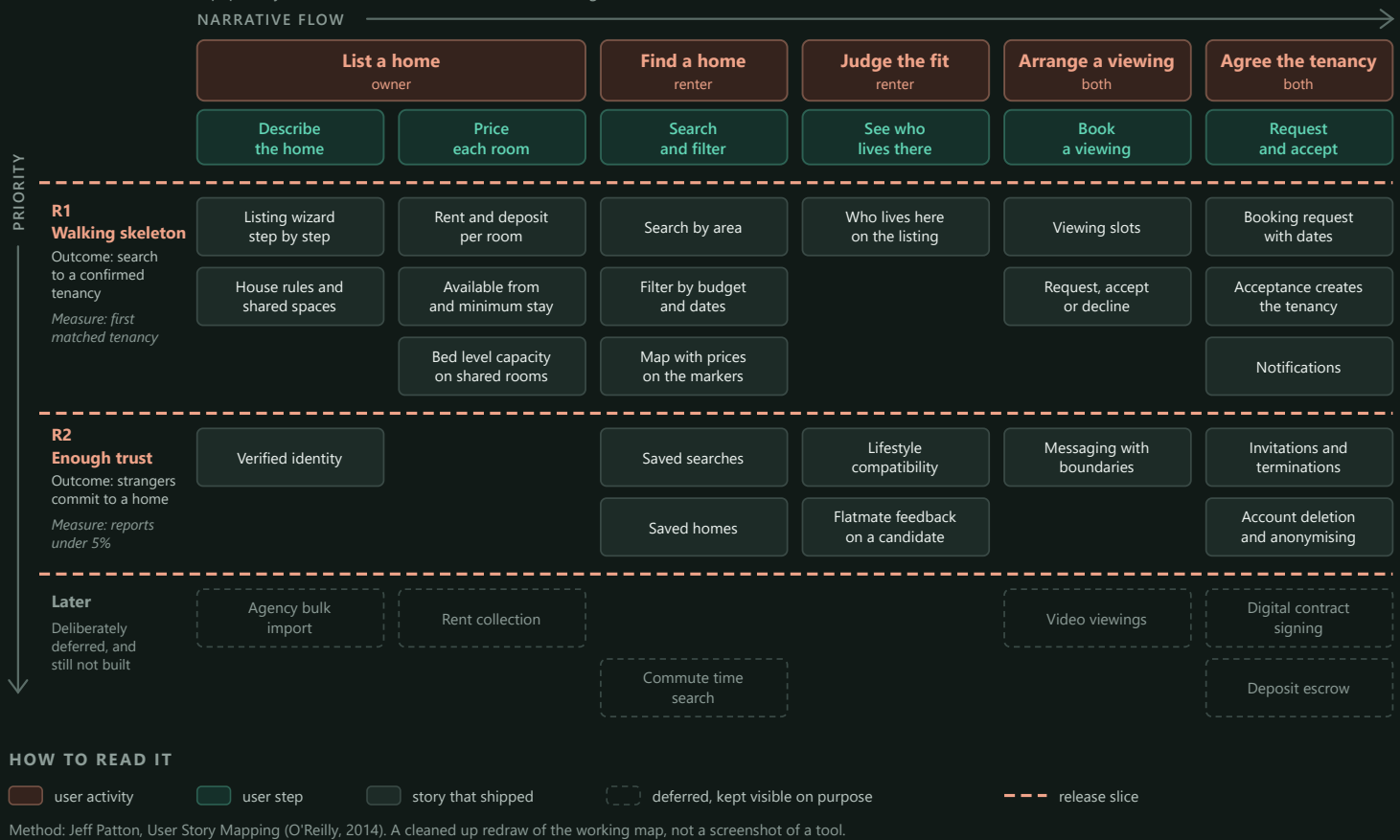
- **Product documentation:** the specs the team built from, kept current as the product changed.
- **Prioritisation:** sequencing features against liquidity, trust and launch risk rather than against what is most interesting to build.
- **UI/UX design:** the four primary user flows, states, wireframes and the interface design for every screen.
- **Market research:** the competitor landscape, target user needs, and where EzyFlat could be structurally different rather than simply better looking.
- **Development:** building the platform end to end alongside a small team, from data model to a fully functional product.

Scoping the MVP with a story map

A flat list of features tells you what someone wants built. It does not tell you what a person is trying to do, or which items have to exist together to be worth anything. So before writing a backlog I mapped the product as a story: the things people do, left to right, in the order you would explain them to someone who had never seen it.

User story map

Narrative flow across the top, priority down the side, and horizontal lines cutting the releases.



The backbone reads left to right. Priority runs down. The first line is the release decision.

The top row is the backbone, and it reads as a sentence. Someone **lists a home**, then someone else **finds a home**, then **judges the fit**, then they **arrange a viewing**, then they **agree the tenancy**. Two sides of a marketplace, one story. Under each step sit the individual pieces of work, stacked with the most essential nearest the top, and horizontal lines cut the map into releases.

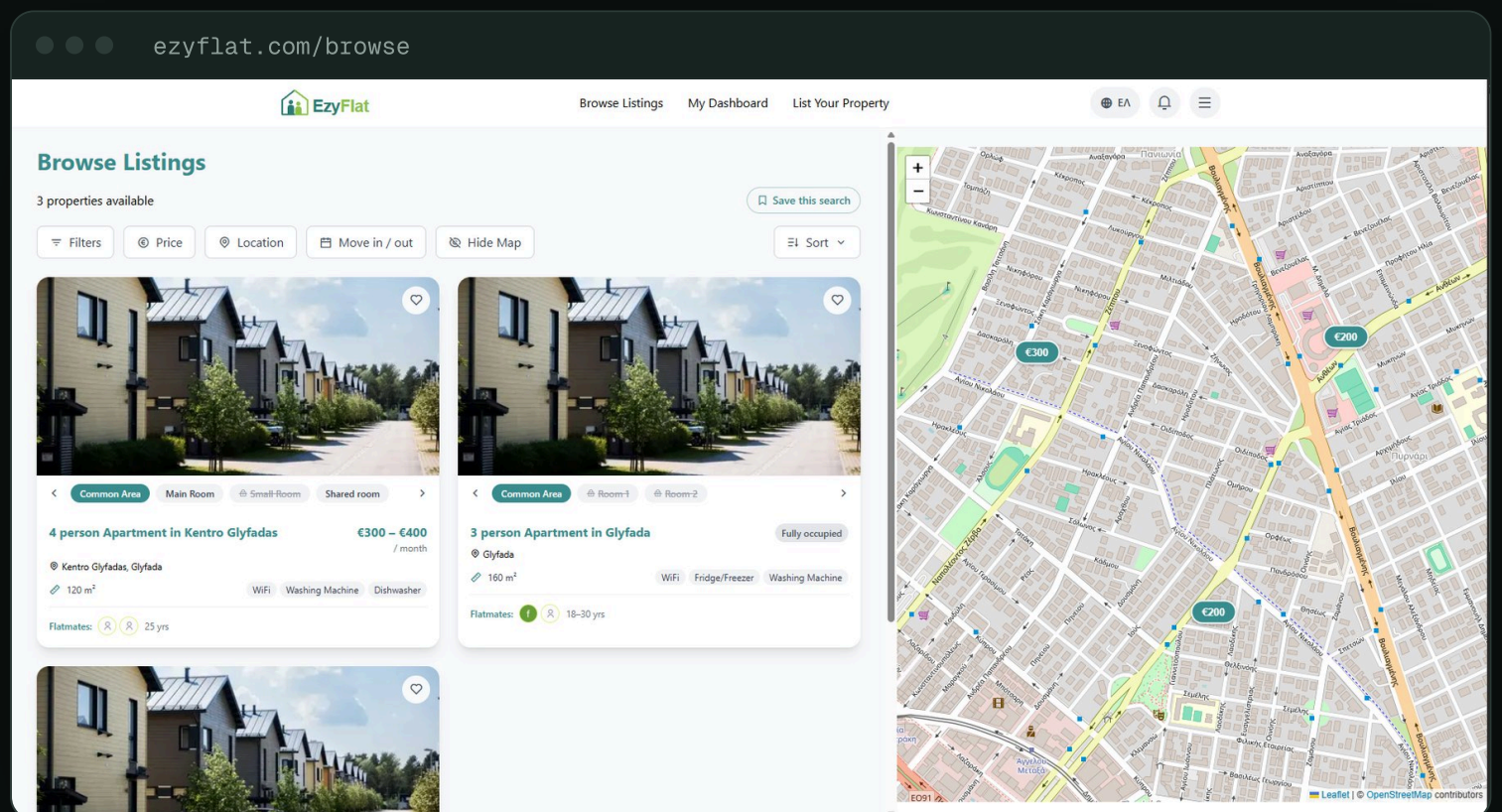
The first release had to cover every column

The temptation with a marketplace is to build one side beautifully and the other side later. That gives you a broken product rather than a smaller one: excellent search with no way to actually agree a tenancy is only a catalogue. So the first slice takes at least one story from *every* column, even where that meant the thinnest possible version of it. The test it had to pass was a single sentence: can one renter go from searching to a confirmed tenancy in a real home, without anyone stepping in by hand.

The second slice is about trust, because a flatshare asks strangers to commit to living with each other, and nothing in the first slice earns that. Verification, messaging with boundaries, compatibility and the ability to leave cleanly all sit there. Its measure is the report rate in the next section, which is the number that says whether the trust work is holding.

The bottom band is the part I find most useful to show. Rent collection, deposit handling, contract signing and agency bulk import are all reasonable things for this product to do, and none of them are built. Each one adds regulatory and financial surface area, and none of them helps answer the only question the first release existed to answer, which is whether two strangers will actually agree a tenancy through the platform. Keeping them visible on the map, rather than deleting them, is what makes it a decision rather than an omission.

One clarification, since these artifacts get confused. This is a story map, not a journey map and not a process model. It has no emotion row and no satisfaction curve, because it is not describing how it feels to use the product. It has no gateways or actor swimlanes, because it is not describing how work flows through a system. It has release lines, because its job is to decide what gets built and in what order.



Browse: filters, per-room pricing on every card, and a map that carries the prices with it. Demo data, for privacy reasons.

How I measure whether it works

I defined the metric set before the features shipped, so that "is this working?" has an answer rather than an opinion. None of them is a vanity number. Each one sits at a step in the funnel where a two-sided marketplace typically loses users.

Metric	MVP target	What it tells me
Signup → profile completion	70%+	Whether onboarding asks for too much before it gives anything back
Listing completion rate	50%+	Whether the listing wizard is too long for the supply side
Viewing request response rate	40%+	Whether the marketplace is alive or just populated
Time to first contact	< 48 hours	The single number that predicts whether a user comes back
Report rate	< 5%	The quality floor, and the point at which trust work takes priority over growth

How the pricing model creates liquidity

The cycle that keeps competitors small

Landlord charged heavily to list

Few landlords bother listing

Renter finds little to search

Renter goes back to Facebook groups

The platform stays thin, because the side holding the supply had the least reason to show up.

The cycle EzyFlat is built for

Free and attractive to landlords

Real supply, real liquidity

Renter gets real value, pays to stand out

Revenue funds growth on both sides

more reason to attract landlords

The platform grows because both sides get real value from it.

The basic experience of using EzyFlat stays free. What renters pay for is standing out, not access.

Same market, opposite cycle: who gets charged decides whether the platform stays thin or gets real liquidity.

Competitor research started from one observation. Nearly every flatsharing platform in Greece stays small, with very few live listings, while the same search runs at real volume inside Facebook groups.

That gap was worth understanding. The listings and requests already circulating in those groups, on top of the research I had run, proved the demand exists and can be marketed to. It simply was not reaching the existing platforms.

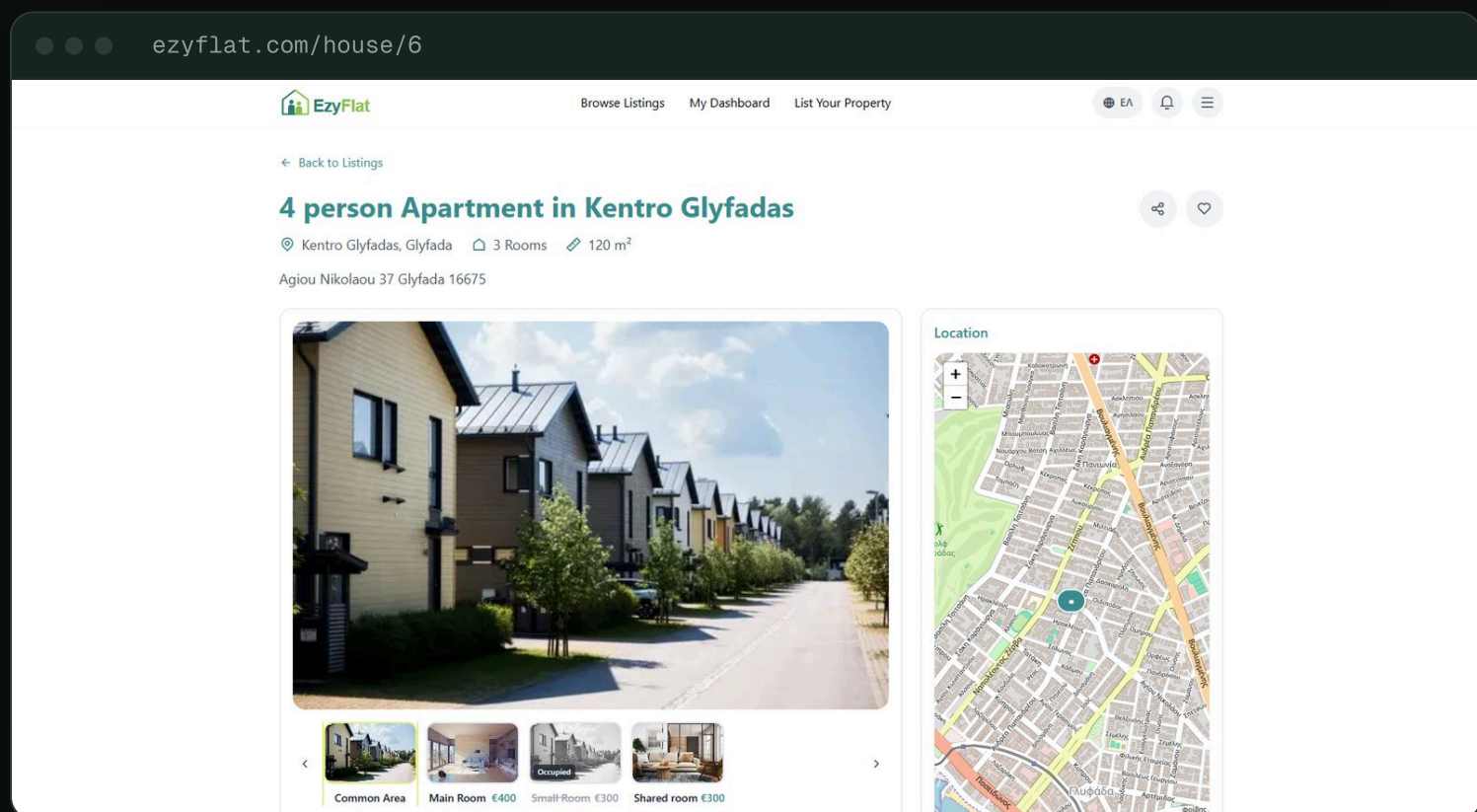
Their pricing models explained a large part of it. Most charge the landlord side heavily, and **the landlord is the side with the leverage**, the one who actually brings the supply. A renter gets no value from a platform with nothing listed on it, so those platforms stay thin and users go back to the channels that have more rooms to search.

The base platform stays free for a renter. The paid packages are about standing out in a market where demand runs far ahead of supply: they exist to help someone win a competitive search, not to gate access to it.

- **Renter subscriptions:** optional tiers that give a genuine edge over everyone else chasing the same room. This is where most of the revenue is designed to come from.
- **Landlord packages:** paid separately, for the extra capabilities that come with more complex portfolio management.
- **Partner services:** additional home-related services, outsourced to partners.

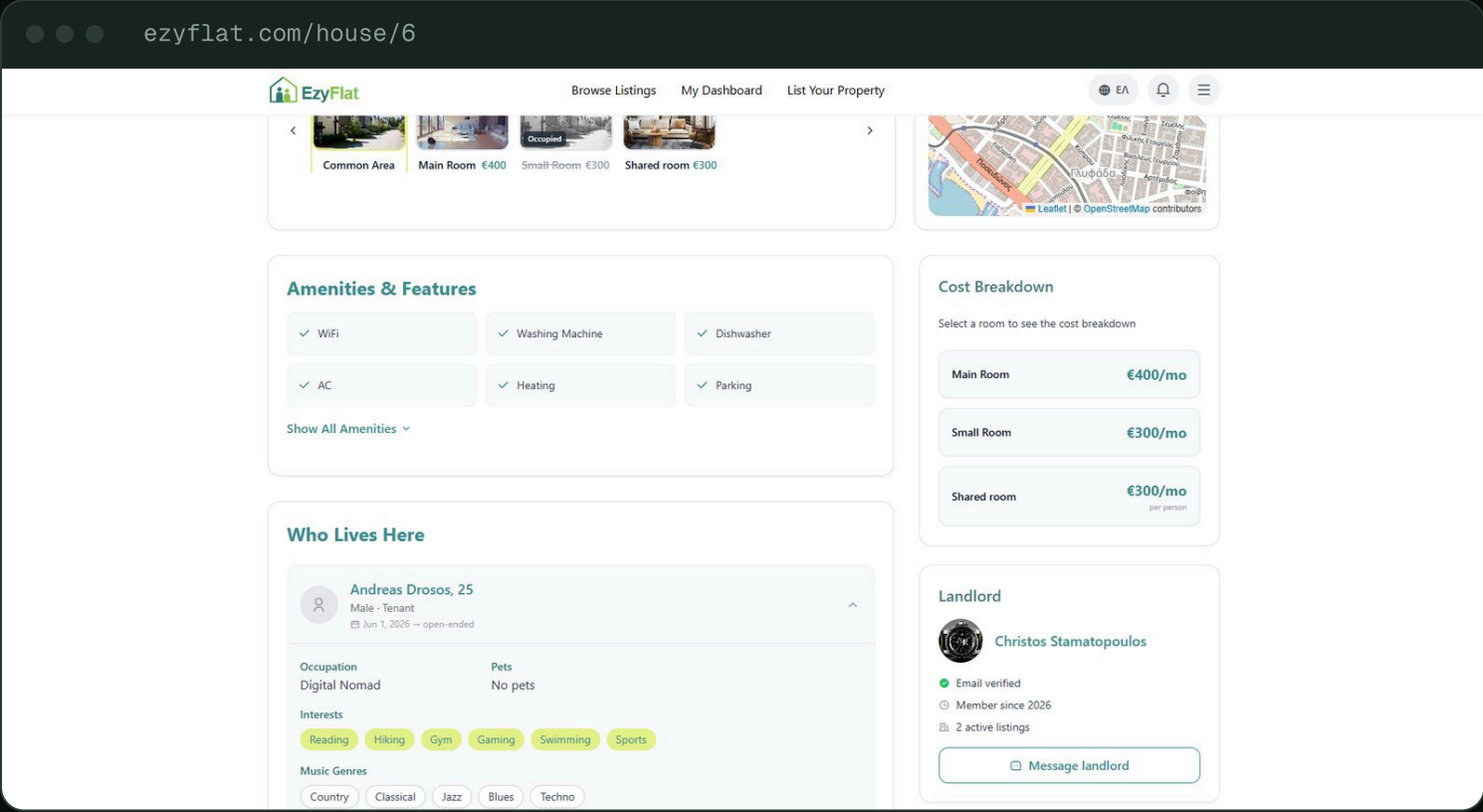
None of this runs from day one. It switches on once the user base reaches specific targets, the same sequence-against-liquidity logic that shaped what shipped and in what order.

What shipped



A listing opens on the room picker, with each room priced individually and carrying its own availability state. Demo data, for privacy reasons.

Following the map: search and filtering, listing creation as a multi-step wizard, room- and bed-level availability, viewing slots and requests, booking requests that create a real tenancy on acceptance, verification, listing-scoped and direct messaging, saved homes and saved search areas, notifications, household management with invitations and terminations, a lifestyle compatibility engine, and account lifecycle including GDPR deletion.



Further down the same page, the “Who lives here” section shows the household model that classified-ad competitors do not have. Demo data, for privacy reasons.

[See the live product →](#)

← All projects

OptimaLifts · Startup, idea to MVP

Building a digital coach, and changing the version that could not scale

OptimaLifts started as wearable sensors, reached a national startup final, went through a year-long incubator track, and came out the other side as a software product. This case study covers the decision to change direction.

Problem

People who lift weights get their execution and their intensity estimate wrong, and both are invisible to them.

My role

Co-founder & Product Designer, owning the vision, the roadmap, the user stories and the business model.

Team

4 co-founders, a hired computer vision engineer and 3 mentors

Timeline

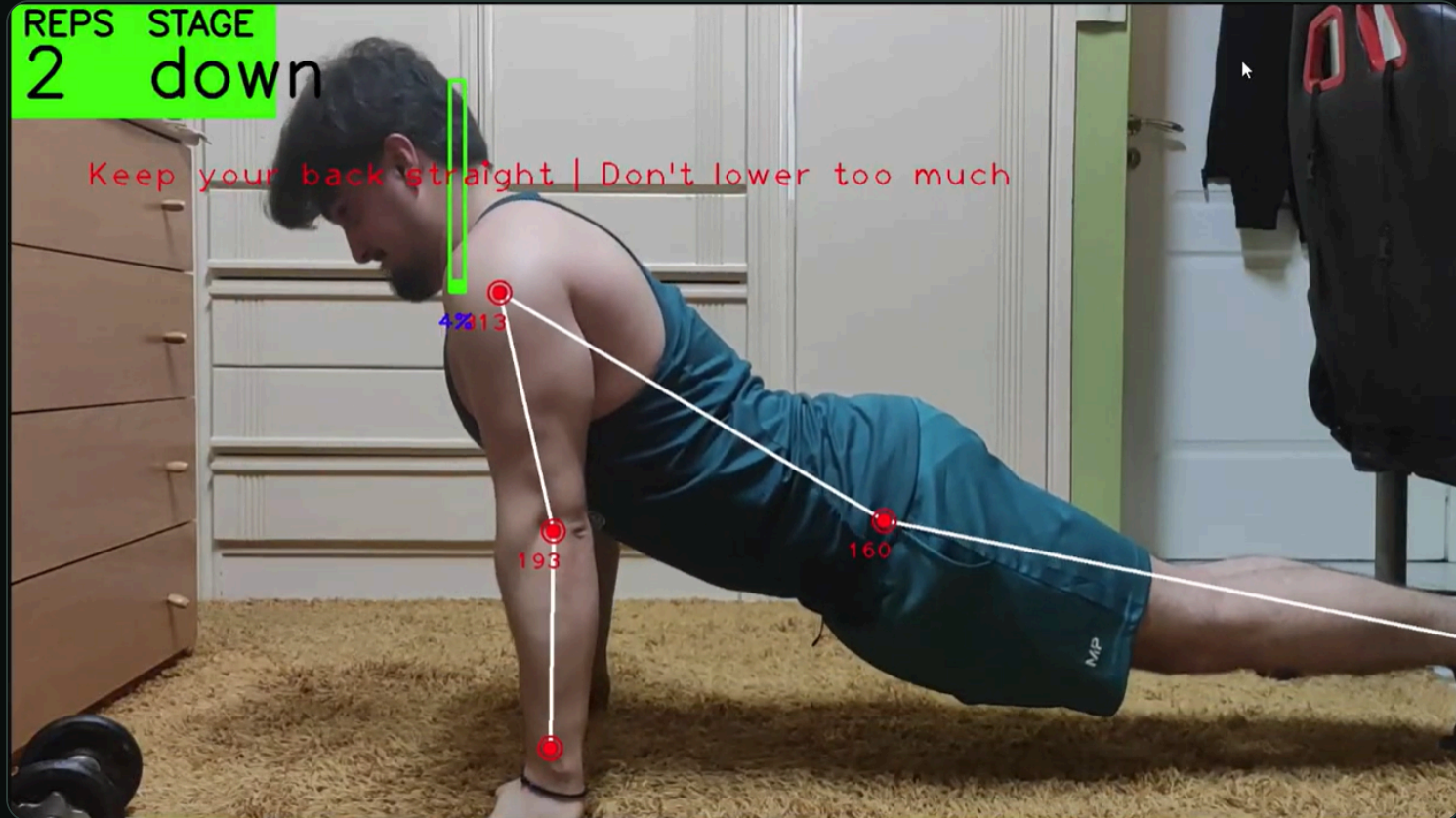
Sep 2023 – Jul 2024

Outcome

JA Greece startup finalist and graduate of the year-long ACEin Compass Cycle incubator.

REPS STAGE
2 down

Keep your back straight | Don't lower too much



The problem

People who lift weights consistently get two things wrong. They get the **execution** wrong, meaning technique, posture and tempo, and they get the **intensity estimate** wrong, training either too far from failure to grow or too close to it to recover. Both are invisible to the person doing them, which is why a coach is valuable.

The evidence behind the pitch, sourced rather than asserted:

- 53% have been injured at least once through incorrect execution of an exercise.
- 32% of those injuries left permanent pain.
- 47% misjudge the intensity they are actually placing on the muscle.

Every number in that list carries a source rather than an assumption, a habit I kept for the competitive audit work I do now. That framing mattered more than it might appear. It moved the product away from being a fitness tracker, a saturated category competing on step counts, and towards being **a substitute for a coach**, which is something people already pay significant money for.

1 yr

ACEin Compass Cycle
incubator track

4

co-founders

3

mentors guiding the
team

4

revenue streams
modelled to five years

The competitor matrix that shaped the roadmap

I benchmarked OptimaLifts against four established players in the market on five capabilities, each chosen because it maps to a user need rather than to a spec sheet:

FIVE CAPABILITIES, BENCHMARKED AGAINST FOUR ESTABLISHED PRODUCTS

Competitors are not named here. The comparison is about capability, not about naming other people's products.

	OptimaLifts	Competitor 1	Competitor 2	Competitor 3	Competitor 4
1 Set and repetition tracking	Full	Full	Full	None	Full
2 Movement and technique tracking	Full	Full	None	Partial	None
3 Speed and intensity tracking	Full	None	Full	Partial	None
4 R.I.R. calculation reps in reserve, how many are left before failure	Planned	None	None	None	None
5 Affordable and discreet nothing extra to buy, charge or wear to the gym	Partial	None	None	None	None

EMPTY IN EVERY PRODUCT BENCHMARKED

Rows 1 to 3 are contested and every product covers one or two of them. Only Competitor 3 combines technique with intensity, and it does neither fully. Rows 4 and 5 are empty across the whole set, so that is where the product was aimed.

WHERE OPTIMALIFTS WAS BEHIND

The first demo covered far fewer exercises than the established products ship. Row 5 is only partly won: the phone removes the purchase and the wearable, but the user still has to place the camera. And it measures nothing directly, it infers.

HOW TO READ IT

Full, does this properly

Partial, present but limited

None, absent

Planned, on the roadmap

Method: four products benchmarked hands on, alongside the wider market scan, during 2023 and 2024. A snapshot of the category at that time, not maintained since. Names withheld deliberately; each column is one real product that was tested.

The four products are not named, deliberately. What matters is the shape: rows 1 to 3 are contested, rows 4 and 5 are empty across every competitor. The row where OptimaLifts was behind is on the diagram too, because a matrix where one product wins everything is not worth reading.

- **Set and repetition tracking:** counting sets and reps without the user having to. The most contested row: three of the four competitors did it fully, and so did OptimaLifts.
- **Movement and technique tracking:** seeing whether each rep is performed correctly. Only one competitor did it fully and one partially, while OptimaLifts covered it fully.
- **Speed and intensity tracking:** measuring how fast and how hard each rep is. One competitor did it fully and one partially; that same partial product was the only one combining technique with intensity, and it did neither fully.
- **R.I.R. calculation,** meaning reps in reserve, or how many repetitions remain before muscular failure. No competitor offered it. For OptimaLifts it was planned on the roadmap rather than built.

- **Affordable and discreet implementation:** nothing extra to buy, charge or wear to the gym. No competitor managed it. OptimaLifts only partly did: the phone removes the purchase and the wearable, but the user still has to place the camera.

The matrix showed the field splitting clearly. The established players each covered one or two of the first three capabilities, very few combined technique with intensity, **not one of them offered R.I.R. calculation at all**, and the products that came closest were expensive and conspicuous. **Rows 4 and 5 were where the opportunity sat**, so I made them the two things the product was not allowed to compromise on.

“Affordable and discreet” as a product requirement

Every competitor asked the user to wear extra equipment to the gym. That is a behaviour change with a financial cost, a charging routine and a social cost, all imposed before the user has received any value. Scoring it as a row in the competitor matrix is what later made the move from hardware to camera a straightforward decision rather than a difficult one.

Changing direction: from sensors to the camera

The first version was a wearable, with sensors recording movement, rep speed and volume and feeding a digital coach. It took us to a **national final at the JA Greece startup competition** with a working Arduino and IoT prototype, a full business plan and a complete business model. It also came with a unit economics problem that could not be argued away.

	Wearable version	Camera + computer vision
Cost to the user	~\$130 hardware, plus subscription	Zero, since the phone is already in the gym bag
Cost to us	~\$60 per unit, plus manufacturing, supply chain, returns	Software, plus model training effort

	Wearable version	Camera + computer vision
Adoption friction	Buy, wait for delivery, wear it, charge it	Install, prop the phone up, start
Technique feedback	Inferred from limited sensor placement	Full body pose, which was the signal we needed
Cost of being wrong	Inventory	A release

So we changed direction. The concept stayed the same, counting the reps, judging the form and estimating how close to failure the lifter is, while the sensing moved to the smartphone camera and a pose-detection model. That meant hiring for a skill nobody on the team had.

Hiring for a skill I did not have

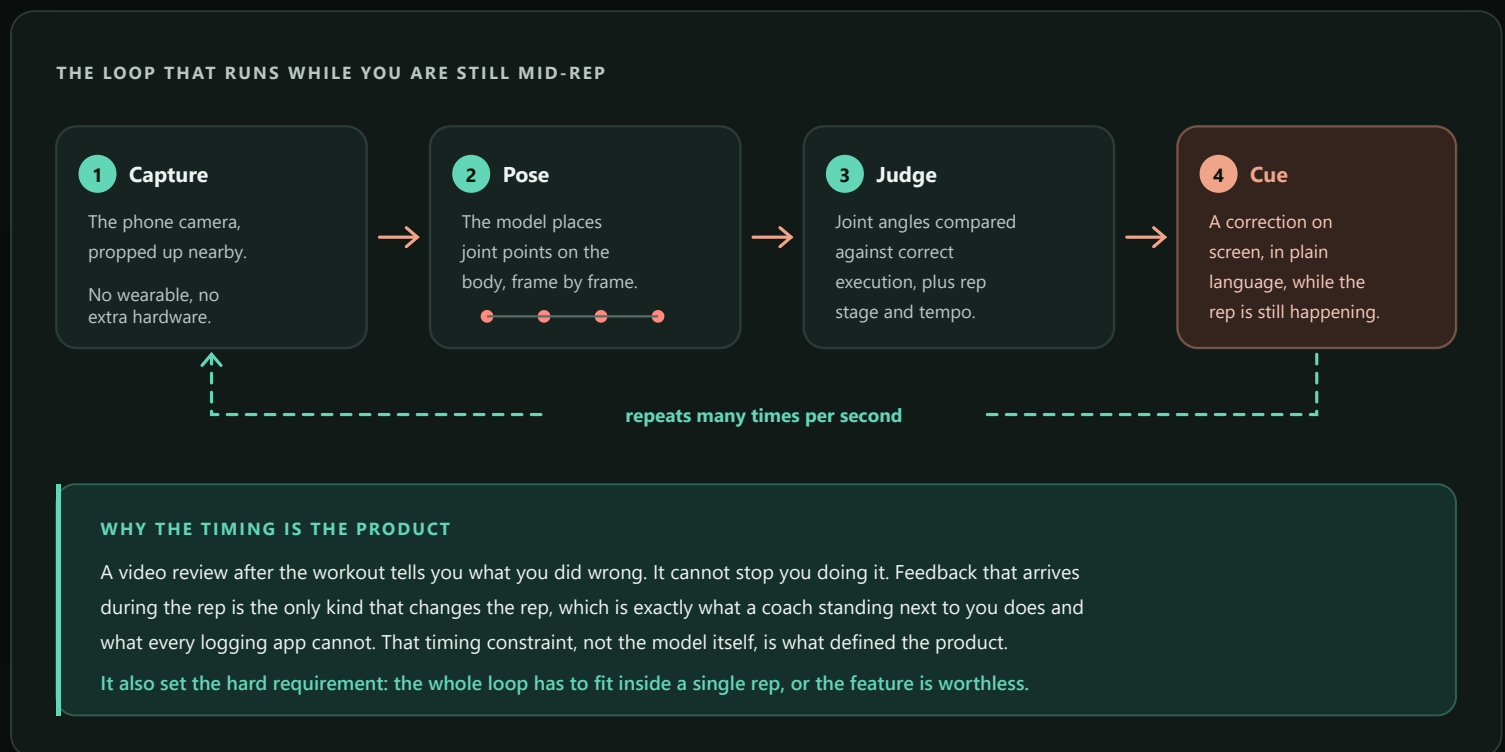
I ran the process for our **Computer Vision Engineer** end to end. I wrote the job description, sourced and tracked applicants, ran first interviews from a written structure, kept written notes on each candidate so the decision was comparable rather than remembered, and handled the replies, including the rejections.

It is the closest thing to product work that is not product work. You are defining a role from an ambiguous need, screening against criteria you had to create, and making an expensive and hard-to-reverse decision on incomplete information. It also changed how I write acceptance criteria. If I could not state what good looked like for a candidate, I had no business asking someone to build to it either.

The product: coaching that arrives during the rep, not after it

This is the part that makes OptimaLifts a coach rather than a log. The correction does not come at the end of the set, or in a video review that evening. It appears **while the repetition is still happening**, in the fraction of a second where the lifter can still change what their body is doing. So the lifter can fix the movement during the rep instead of hearing about the mistake afterwards.

Everything else in the product follows from that one constraint. The whole loop, from the camera frame to the words on screen, has to complete inside a single rep, which ruled out anything that needed to be uploaded, queued or reviewed later.



Capture, pose, judge, cue. The loop repeats many times a second, so the feedback lands while the rep is still in progress.

I mapped the use cases for the full product, covering profile and body data, auto-generated plans, workout navigation, guided execution with rest timers, live performance, a social feed, statistics and leaderboards, an "Optimal Academy" knowledge section with an AI chat, and settings. I then prioritised them into release groups, with the live coaching loop first, because a product that cannot do that convincingly does not need any of the rest.

The first demo detected all the main execution errors of the exercises it covered, with rep counting, stage detection and the live correction working end to end.

One of the user stories behind that loop, written the way I write them: the statement, the acceptance criteria as Given, When and Then, the rules any implementation has to respect, and an explicit list of what the story does *not* cover.

User story **Movement tempo feedback**

As a trainee working out alone, I want to know in real time when I am performing a repetition too quickly or too slowly, so that I can adjust my movement to the intended exercise tempo.

Acceptance criteria

AC1 Detect movement that is too fast

Given the trainee is performing a supported exercise in front of the camera,
When the trainee performs a repetition faster than the intended tempo,
Then the app displays an actionable cue while the repetition is still in progress.

AC2 Detect movement that is too slow

Given the trainee is performing a supported exercise,
When the trainee performs a repetition slower than the intended tempo,
Then the app displays an actionable cue while the repetition is still in progress.

AC3 Keep feedback actionable

Given a tempo deviation is detected,
When the corrective cue is displayed,
Then the cue tells the trainee how to adjust their pace rather than presenting a technical speed measurement.

AC4 Remove feedback when the tempo is corrected

Given a tempo corrective cue is being displayed,
When the trainee returns to the intended pace,
Then the cue clears automatically without requiring interaction with the device.

AC5 Avoid unreliable tempo feedback

Given the system cannot reliably assess the trainee's movement speed,
When there is insufficient visual information to determine the tempo,
Then the app does not display a tempo correction based on an uncertain assessment.

Business rules & product constraints

- The intended tempo is exercise-specific rather than a single universal speed.

- Tempo feedback is intended to influence the trainee's movement during the repetition, not merely provide a post-set measurement.
- Feedback should communicate an actionable adjustment rather than expose technical measurements.
- The system should prioritise reliable feedback over generating a correction when confidence is insufficient.

Out of scope

Range-of-motion assessment

Repetition counting

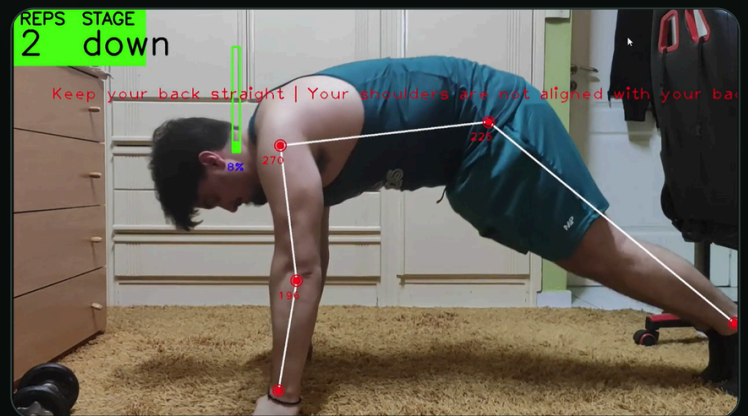
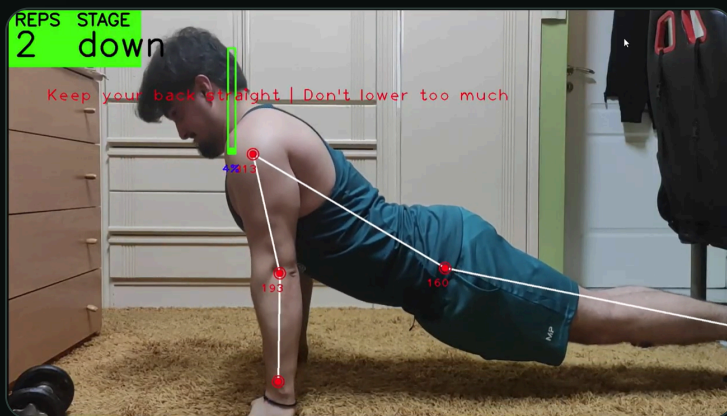
R.I.R. estimation

Post-set tempo statistics

Overall workout scoring

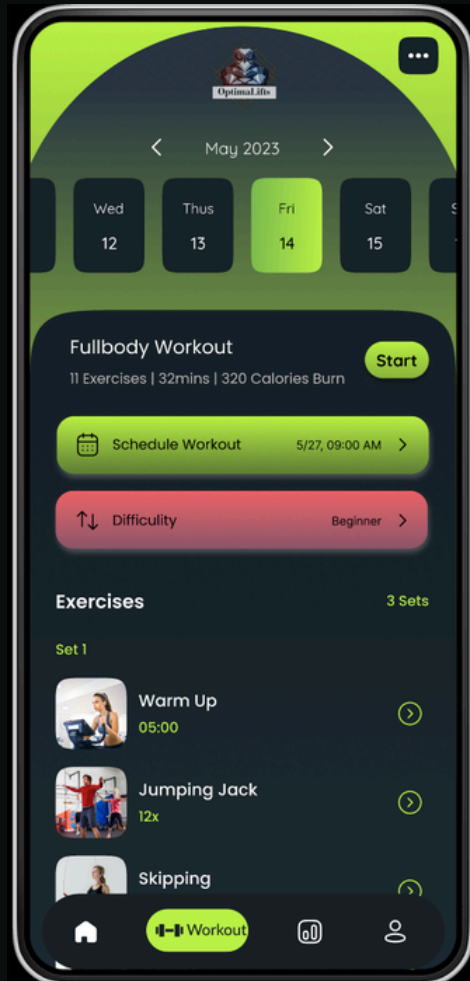
What it looked like

This is the working prototype, not a mockup: a skeleton overlay tracking the movement, the rep counter and stage, a range-of-motion bar that fills and empties as the trainee moves through the repetition, and the correction appearing in real time as the form breaks.



Two corrections caught mid-rep, with the joint angles the model is reading. This is the actual output of the computer-vision model, not a render.

Alongside the working loop I designed the interface for the fuller product: a home dashboard, a scheduled workout plan, the rest-timer analysis, a statistics view and workout history. The design question running through all of them was **what a lifter can act on at each moment**. Mid-set, the screen carries only what changes the current repetition, the rep count and stage, the range-of-motion bar and the corrections, all of it readable at a glance from across the room and none of it requiring a hand on the phone. Analytical detail such as R.I.R., time under tension and per-rep tempo would be noise at that moment, so it waits for the rest timer, when the trainee has eighty seconds and nothing to do with them. Matching the density of an interface to what the user can actually use right then, rather than to how much data the product happens to have, is a habit I have applied to every product since.



Design exploration beyond the shipped MVP: dashboard, workout plan, rest-timer analysis and history.

The business model for the camera version

Dropping the hardware invalidated the original model, which had leaned on equipment margin. I rebuilt it as a pure software model with **four layers, each one unlocked by the one before it**.

FOUR LAYERS, EACH UNLOCKED BY THE ONE BEFORE IT

1 Free

Rep counting and live form correction across the whole exercise library, plus workout logging.
The coach experience is free because that is the thing nobody believes until they see it.

acquisition, not revenue



2 Premium subscription, monthly or annual

R.I.R. estimation, tempo and time under tension, progressive overload planning and long-term history.
Both periods are offered, with annual pre-selected and discounted, because progressive overload is measured in months, so the annual term is the one over which the product actually proves itself.

the core of the revenue



3 Gyms and trainers, per seat

A trainer supervises more clients than they can physically stand next to. Per-seat licensing plus remote form review turns the same model into a tool that scales a trainer's capacity rather than replacing them.

higher value per account, slower sales cycle



4 Aggregate insight, only once there is scale

Anonymised execution data has value to equipment and fitness businesses, but it does not exist until layers 1 to 3 do.

Four layers, sequenced by what has to be true before each one can start earning.

Three of those decisions are worth explaining, because they are the ones a fitness product usually gets wrong.

The coaching itself is free, across every exercise. The instinct is to paywall the thing that makes you special, and in most categories that is right. Here it is backwards. Nobody believes a phone camera can correct their squat until they watch it happen to them, so a hard paywall would be charging for a claim rather than for a product. Locking most of the exercise library behind that paywall fails for the same reason, so the free tier covers all of it. Industry benchmarks bear this out: freemium in health and fitness converts at roughly 2 to 3%, while free trials that let the user experience the product first convert far higher, up to around 49% for card-gated trials. In a category built on a claim people do not believe, the demonstration *is* the acquisition channel. What premium sells is not access to the coach, it is everything the coach can tell you over time.

Premium defaults to annual, with monthly still available. Both periods are offered, because forcing a year's commitment on someone who is not ready is a good way to lose them. Annual is pre-selected and carries a real discount, and that is not a pricing trick. Progressive overload, the thing the paid tier actually measures, only becomes visible over months. A monthly plan asks the user to renew before the product has had time to prove anything, and invites churn precisely when the value is about to appear. Annual billing aligns the billing period with the period over which the product justifies itself, which is also why health and fitness is the one category where annual plans dominate, at around 61% of revenue.

B2B sits third because it changes the product's shape, not just its price. A trainer can only physically stand next to one client at a time, so per-seat licensing plus remote form review turns the same model into something that scales a trainer's capacity rather than competing with them. The aggregate insight layer sits last for the obvious reason: it does not exist as an asset until the layers above it have produced enough data to be worth anything.

The incubator

OptimaLifts was selected into **Compass Cycle**, the year-long incubator track of ACEin, the Athens Center for Entrepreneurship and Innovation at AUEB, and completed it in 2024. It covered a year of customer discovery, business modelling and growth planning with mentors, milestone planning against a time plan, and regularly having to defend the roadmap to people with no reason to be polite about it.

What I took from it

- **Put the uncomfortable criterion in the comparison table.** Scoring "affordable and discreet" as a row is what turned abandoning our own hardware into a calculation instead of an argument. A constraint that lives only in someone's head loses to whichever option has the better slide, every time.
- **Find the constraint that defines the product, then design everything around it.** Here it was timing: feedback had to land inside the rep or it was worthless. Once that was clear, it settled the architecture, the scope and the interface without further debate.
- **Write the honesty requirement into the acceptance criteria.** "Say you cannot see the movement rather than guess" is a product decision, not an engineering one. Leaving it unwritten means it gets traded away the first time someone wants a smoother demo.
- **A business model works as a sequence of layers.** Free coaching, annual premium, per-seat B2B, then data. Each layer exists because the one before it made it possible, and getting the order wrong is how products with good economics on paper still fail.

- **What I would do differently:** I validated the problem thoroughly and the *willingness to pay* hardly at all. I had injury statistics and a competitor matrix long before I had one person who had said they would pay for it. Now the pricing conversation happens in week one, not in the financial model.

← All projects

Flashcards · University project

A study app built around how memory works

A flashcards app built around two evidence-based learning methods, active recall and spaced repetition, taken through the full development cycle: requirements, data model, architecture and a working JSP application, delivered by a team of four.

Context

Information Systems Development & Architecture, AUEB

Timeline

Delivered Jan 2024

Problem

Flashcards are a commodity, so the product had to be built around how memory actually works.

My role

Requirements, data & backend, in a team of four.

Team

4 students

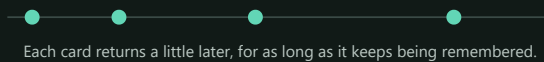
Outcome

A working JSP application with spaced repetition, delivered with use cases, a data model, the architecture and a set of test cases.



Active recall

Spaced repetition



Each card returns a little later, for as long as it keeps being remembered.

About this project

This is a university project delivered with three teammates. It earns a place here because it is the one piece of my work where the *entire* software development lifecycle is visible in a single deliverable, covering requirements, data model, architecture, object design and a set of test cases, and because it was built around a product idea with research behind it rather than being a standard CRUD exercise.

Choosing the learning method, not just building the app

Flashcards themselves are a commodity, since every study app has them. Before any of us wrote a line of code, we made a requirements decision that shaped everything downstream: which *learning method* the product would be built around, because that determines what the scheduling logic has to do. We chose two with real evidence behind them:

- **Active recall.** The user tries to produce the answer from memory before seeing it, rather than passively re-reading it. This is the method that genuinely strengthens memory, whereas re-reading mostly produces a false sense of familiarity.
- **Spaced repetition.** A card is shown again on a schedule calculated from how well and how quickly the user remembered it last time, rather than on a fixed daily cycle. Cards the user knows well appear less often, while difficult ones return sooner.

Both methods only work if the product tracks per-user, per-card history. That turned "build a flashcards app" into "design a data model that can drive a scheduling algorithm", a considerably harder problem.

4

team members

2

learning-science
mechanics built in

4

core entities modelled

5

SDLC artifacts delivered

Requirements, as use cases

Every screen traces back to a documented use case with an actor, a goal, the flow a developer could build straight from, and the alternative flows covering everything that can go wrong along the way. One of them:

Actor	Goal	Trigger
User	Create a new flashcard and add it to a selected deck.	The user chooses to create a new flashcard.

The user creates a flashcard by selecting the destination deck and providing the question, answer, and optionally a hint.

Preconditions

- The user is authenticated.
- The system can retrieve the user's available decks.

Postconditions

Success

- A new flashcard is created.
- The flashcard is associated with the selected deck.
- The flashcard is available for future study.

Failure

- No flashcard is created.
- Any information entered but not successfully submitted is not persisted.

Basic flow, main success scenario

- 1 The user chooses to create a flashcard.
- 2 The system presents the available decks.
- 3 The user selects the deck to which the flashcard will be added.
- 4 The system presents the flashcard creation form.
- 5 The user enters the question.
- 6 The user enters the answer.
- 7 The user optionally enters a hint.
- 8 The user confirms the creation.
- 9 The system validates the required information.
- 10 The system creates the flashcard and associates it with the selected deck.
- 11 The system confirms that the flashcard has been created successfully.

Alternative flows

A1 Decks cannot be loaded

Starts at step 2

- 1 The system cannot retrieve the user's available decks.
- 2 The system informs the user that the decks could not be loaded.
- 3 The user chooses to retry.
- 4 The system attempts to load the decks again.
- 5 If successful, the use case resumes at step 3.
- 6 If the decks still cannot be loaded, the use case ends without creating a flashcard.

A2 Required information is missing

Starts at step 9

- 1 The system determines that the question or answer is missing.
- 2 The system informs the user which required information must be provided.
- 3 The user provides the missing information.
- 4 The use case resumes at step 8.

A3 User cancels creation

Starts any time after step 4 and before step 10

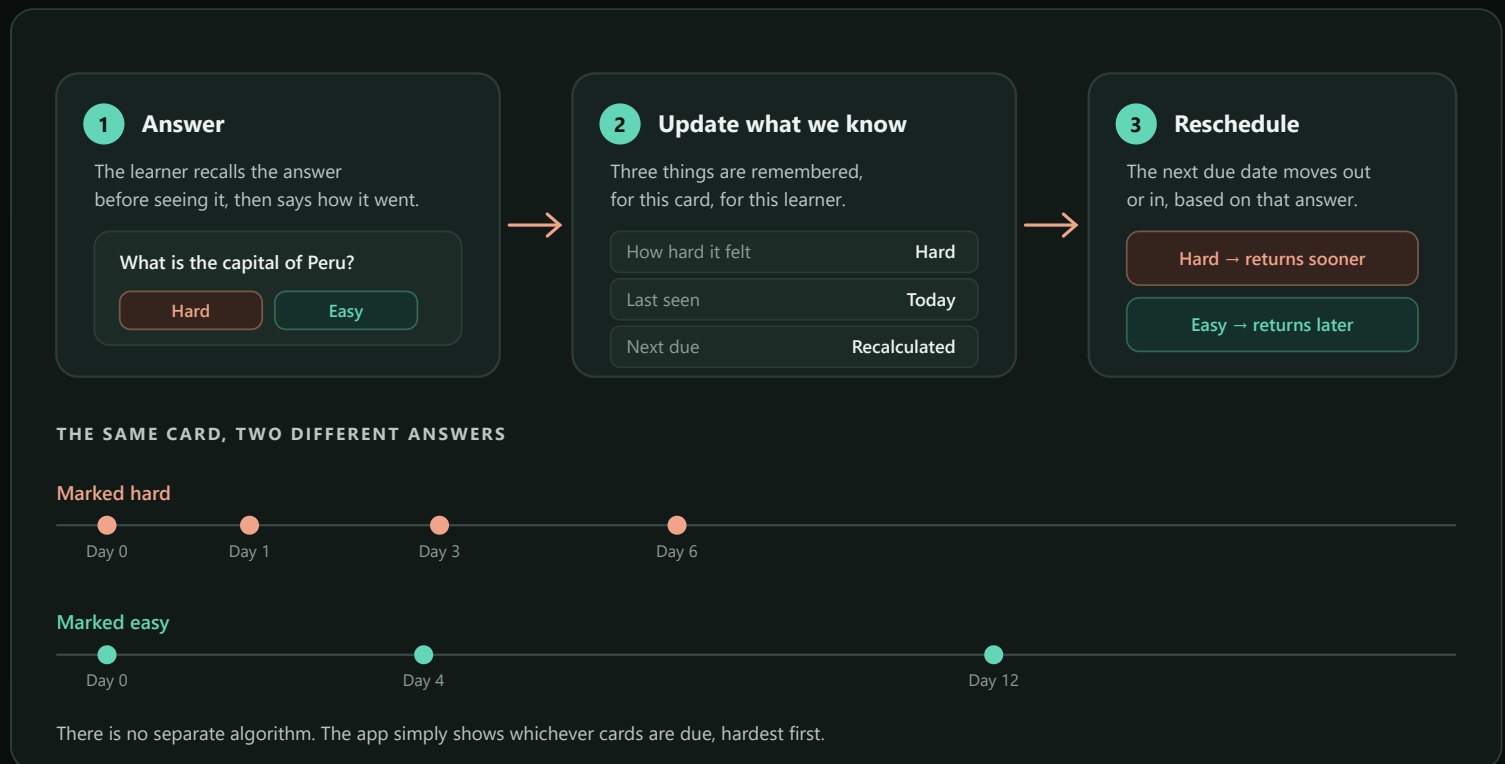
- 1 The user chooses to cancel the creation.
- 2 The system exits the creation flow without creating the flashcard.
- 3 The use case ends.

Special requirements

- The system must prevent creation of a flashcard without a question and answer.
- The hint is optional.
- The flashcard must belong to exactly one selected deck.

What the app remembers about every card

This is the part of the project I would point to first. Spaced repetition sounds like an algorithm, and it is not. It falls out of remembering three simple things for every card, for every learner: **how hard it felt last time, when they last saw it, and when it should come back**. Decide those three well and the question "when does this card reappear?" answers itself. Decide them badly and no amount of clever scheduling code will rescue it.



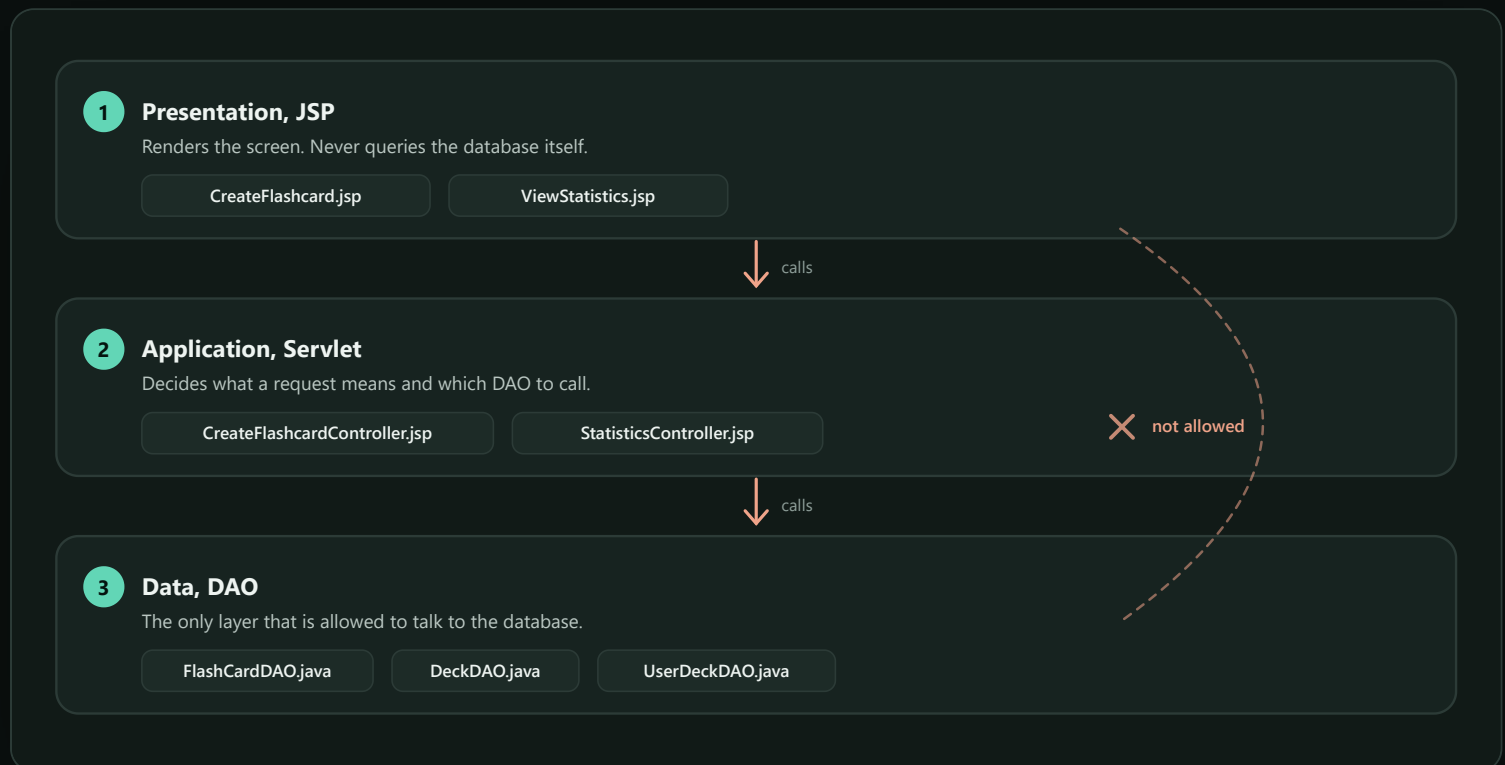
The three-step loop behind every review. The whole schedule rests on those three facts, kept per learner and per card, and nothing else.

A product decision as much as a technical one

It is a small, concrete example of a principle I use often: a feature that sounds like an algorithm is frequently just a question of what you decide to remember. Settling those three facts up front made the scheduling straightforward to build, and settling them badly would have made it close to impossible.

Architecture: three separated layers

The app is a classic JSP/Servlet stack, and we kept the three layers strictly separated: JSP pages for presentation, Servlet controllers for the application layer, and DAO classes talking to the database. Every page has a matching controller (`CreateFlashcard.jsp` → `CreateFlashcardController.jsp`), and every entity has a matching DAO (`FlashCardDAO.java`, `DeckDAO.java`, `UserDeckDAO.java`). Nothing in the presentation layer talks to the database directly.



Every page has a matching controller, and every entity has a matching DAO. The one rule that mattered most: nothing in the presentation layer talks to the database directly.

Testing and the rest of the cycle

The deliverable also included a structured set of test cases covering the core flows: sign-in and sign-up, deck and flashcard creation, input validation, deck editing and the spaced-repetition scheduling, alongside use cases for the statistics views. Each test names the screen, the input data and the expected response. Combined with the use case model, the data model and the two diagrams above, the project covers every stage of a classical development lifecycle on a single working product.

The ten test cases from the project deliverable, translated from Greek.

#	Screen	Input	Expected result
1	Sign in	Username: test name Password: test password	Signs in and redirects to the My decks dashboard.
2	Create deck	Deck title: new deck Category: new category	Signed in as test name: a confirmation message, and the new deck appears on the My decks dashboard and in the database.
3	Create flashcard	Front: "Is Unirep fun?" Back: "Yes!" Hint: "Ja"	Signed in as test name: a confirmation message, and the database is updated with the correct values.
4	Play deck	Open the play page without signing in	Redirects to the sign-in page.
5	Sign up	"george" in every field	Message: "Please include @ in the email address".
6	Sign in	Username: test name Password: wrong password	An error message and a return to the sign-in page.
7	Play deck	Signed in as test name on the "Business/IT acronyms" deck, every card played 6 minutes earlier, with SaaS and BI marked "Very Hard"	Only those two cards come back for spaced repetition. Afterwards the app shows "There are no spaced repetition flashcards awaiting review".
8	Play deck	Same deck, every card played 6 minutes earlier, with JDK marked "Hard" and ERP "Very Hard"	Only JDK comes back. Marked "Very Hard", the no-cards message appears. After quitting and playing again 10 minutes later, both cards come back for review.
9	Create flashcard	Front: test Back: empty Hint: test	Message: "Please fill in all the fields".

#	Screen	Input	Expected result
10	My decks dashboard	Choose Edit on “Business/IT acronyms”	Every flashcard shows with the correct fields, matching the database query <code>SELECT * FROM flashcard WHERE deckID = 14;</code>

What I carried forward

- **Decide what to remember before deciding what to build.** Spaced repetition depends on a small set of facts you choose to keep. I now look for this pattern regularly, asking whether the hard part of a feature is really a question of what the product needs to remember.
- **Choose a product method with evidence behind it.** Building around active recall and spaced repetition rather than a generic flashcard list is a small decision that changes what the entire schema has to support. I applied the same habit of checking the research before shaping a feature when planning the R.I.R. calculation at [OptimaLifts](#).

CONTACT

Let's talk

I am open to Product Manager, Product Owner and Business Analyst roles, and happy to walk through any of the work in this document in more detail.

Email	fratzeskos.dafereras@gmail.com
-------	--------------------------------

Phone	+30 697 253 6609
-------	------------------

LinkedIn	linkedin.com/in/fratzeskos-dafereras-0713011a1
----------	--

Portfolio	fratzeskosdafereras.com
-----------	-------------------------

Location	Dublin, Ireland · open to remote in Europe
----------	--

Availability	1 month notice · full-time, hybrid or remote
--------------	--