Computer Science Department

Software Engineering & Business Analysis

Bachelor's Thesis

MovieCrush

Movie Tracking Platform with Personalized Recommendations, Detailed Ratings, and Social Features

Anastasiia Syvak

Supervisor
KSE, Denys Zavhorodnii, dzavhorodnii@kse.org.ua

Submission date
16 June 2026

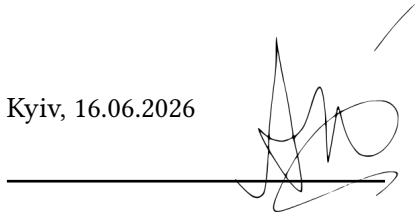
Academic Integrity Statement

I, undersigned, hereby declare that this capstone project is the result of my own work.

- All ideas, data, figures and text from other authors have been clearly cited and listed in the bibliography.
- No part of this project has been submitted previously for academic credit in this or any other institution.
- All code, diagrams, and third-party materials are either my original work or are used with permission and properly referenced.
- I have not engaged in plagiarism or any form of academic dishonesty.
- Any assistance received (e.g. from peers, tutors, or online forums) is acknowledged in the acknowledgements section.

I understand that failure to comply with these declarations constitutes academic misconduct and may lead to disciplinary action.

Place, date Kyiv, 16.06.2026

Signature 

Contents

Acknowledgements

First of all, I want to thank my parents. For giving me the opportunity to study at such a great university, for their support every single time, for always believing in me when I didn't believe in myself - they always knew I would make it. They have always been my biggest role models in life, and everything I have now is because of them. I also want to thank my mom for being someone I can tell absolutely everything to - every secret, every worry, every thought. And I want to thank my dad for teaching me tennis when I was a child - that's what led me to become the head of the KSE Table Tennis Club and find amazing friends at university, who made these student years the best ones.

I want to thank my grandparents - they were always ready to help me with anything, and were even willing to learn how to code just to make things easier for me. Thank you for the fact that when I had classes until 9 PM, I could get to Akademmistechko in 20 minutes instead of an hour and a half, and thank you for always cooking so much and so well every time I came over, as if it was a wedding and not just my last class of the day. I want to thank my brother for always showing up - early or late - even though he wasn't a student here at that moment, just to help or support me.

I want to thank Veronika Anepska for absolutely everything - for always listening to me, for every party, every conversation, every idea, for believing in each other and supporting each other. Together we built discipline - not just opening TikTok every day to keep our streak alive, but also learning Italian every day, doing yoga every day, setting goals and reaching them together. I'm really happy that this diploma was made with you. I'm sure we have an amazing future ahead, because we're doing everything to make it happen.

I want to thank Bohdana Seheda for sitting next to me in an extra English class in first year and Maryna Chernetska for the fact that we were brought together in first year by the same problem - neither of us understood anything. Those small things helped me find such great friends.

Thank you to Ivan Suprun - for always being there to support me and after talking to you, everything always feels lighter. Thank you to Marko Hlibovytskyi for always telling me I would be fine when I felt like giving up, and for helping me install Windows when I needed it.

I really want to thank the EBD27 and BE27, and my Verkhovyna company. Thank you to Dariia Kozlova for the Integration parties, for Halloween vampires, and for inviting me in my fourth year to get through the hard winter moments and the power outages at Hotel Verkhovyna together. Thank you to Dmytro Shevchuk, Matvii Onyshchenko, Artem Litskevych, Dmytro Vasylchuk, Oleksandr Siryi - you are the best. Every holiday, every party, every memory with you is one of the best in my life. Thank you for every tennis game, every good lunch, every conversation, You are all absolute legends. Thank you to Oleh Skakun for always cheering me up and for driving me home, and for the two most interesting episodes in Master Chef.

Thank you to my friend from Italy, Gabriel, whom I met just before the full-scale invasion started - for helping me feel more confident, for every Eurovision night, for respecting

my family and my country, and for the invitation to Italy. Thank you also to my friend from Spain, Markel - I'm sure we will meet someday too.

Thank you to Polina Mamchur, Sofiia Tyshchenko, Sofiia Krotova, Yaroslava Mala, Kateryna Shashkina and Matvii Talalaievskyi for every moment we went through together, for finding something positive even during the worst deadline weeks, and for always helping. And thank you to my table tennis team - for every training, every tournament, every victory.

I also want to thank Kateryna Pyvovarova for our conversation on the 5th floor of KSE - it was you who came up with the Soulmate feature idea, and I'm really glad that conversation happened.

I want to thank the teachers who left a real mark. Thank you to Taras Pavlov - he taught mathematics in the first year and managed to make difficult topics understandable, and more than that - he made me love them even more. Thank you to Yevhen Kubiuk for the practical sessions on algorithms and data structures: I did not miss a single class, because you rarely meet someone who loves what they do so genuinely and wants every student to truly understand. Thank you to Yaroslav Propoi for the practical product analytics course - it was after this course that I understood I want to build my life around analytics. Thank you to Vadym Yaremenko and Ihor Pysmennyi for their huge contribution to my knowledge. Thank you to Volodymyr Skochko - he returned my assignments with new comments every single time, and at some point I genuinely did not know when it would end. But at work I understood what he was preparing me for: clients are often unsatisfied, there are many rounds of revisions, and you just have to keep going. I am ready for that now. And a separate thank you for agreeing to help me install a virtual Windows on my Linux - even though it did not work out, what matters is that he did not say no and tried to help.

Thank you to academic director Artem Korotenko - over four years our leadership changed several times, and Artem inherited a situation that was not easy to walk into. He handles it well, and more importantly - you can always come to him with a question and get real help or at least honest advice. That is not something to take for granted.

And a very big thank you to my supervisor Denys Zavhorodnii. For every meeting, for the time spent reviewing my work, for the guidance, and for walking me through this entire process from start to finish. It was a real pleasure working with you.

Thank you to absolutely everyone who came into my life during these years and made it the best it could be - in such difficult times.

Abstract

Modern film and TV viewers face a fragmented viewing experience: tracking what they have watched, what they want to watch, and what to watch next is split across phone notes, social media saves, and disconnected rating services. A survey of 262 respondents conducted as part of this work confirmed the problem: 49.8% rely on phone notes to track films and 83.6% discover content through social media rather than dedicated platforms. Existing solutions such as Letterboxd, IMDb, and TV Time each cover only part of the user journey and suffer from weak personalization, limited social interaction, or incomplete coverage of both films and TV series within a single catalog.

This capstone project documents the complete development of **MovieCrush** - a cross-platform mobile application that unifies film and TV tracking, personalized recommendations, social discovery, and personal analytics in one product. The system was implemented as a working product with: a hybrid recommendation engine combining ALS collaborative filtering with TMDB Discover candidate generation and Google Gemini re-ranking, annual "Wrapped"-style analytics with personal viewing statistics, a Soulmate feature that matches users by taste similarity, an onboarding flow that solves the cold start problem through actor selection and rating elicitation, and a social layer with follows, detailed ratings with comments, and shared lists. The mobile client uses React Native with TypeScript, the backend Node.js and Express, with a PostgreSQL database and a dedicated Python microservice for the collaborative filtering model.

To validate the concept, three complementary research methods were applied before development began: a quantitative survey of 262 respondents, five qualitative in-depth interviews, and a structured competitor gap analysis covering Letterboxd, IMDb, TV Time, and Moviebase. The recommendation model was initially trained on synthetically generated rating data to simulate early-stage user behavior and validate recommendation quality before large-scale user adoption. The model was evaluated using ROC AUC, achieving a score of 0.877 - substantially above the random baseline of 0.5. User research established Product-Segment Fit for the target audience of 18-24 year old viewers in Ukraine, with 75.5% of respondents willing to pay for a premium tier. The resulting application shows that a single mobile product can replace the disconnected tools viewers currently use and deliver a personalized, socially-aware discovery experience grounded in user research.

1 Introduction

1.1 Context and Motivation

I have been obsessed with films and series for as long as I can remember. My friends know that if they want to talk about cinema, I am always available. Films help me switch off, step into another world, experience someone else's problems, and discover something new. If I had grown up in California instead of Ukraine, I would probably have studied acting, not computer science.

This personal connection is also what made me notice the problems with existing tools. For two years I used both Letterboxd and TV Time, and over that time I found enough frustrations to fill a product backlog. I started with phone notes - a list of films I wanted to watch, and eventually moved to Letterboxd because I wanted to actually track what I had seen and follow friends to see what they were watching. But even there, things were missing. No series support. No personalized recommendations. Statistics locked behind a paywall. No Ukrainian language. And the social layer felt thin.

The decision to build MovieCrush came from a simple idea: instead of renting someone else's apartment and living with their design choices, build your own - with exactly the layout you always wanted. I am the target audience of this product, which made every design and product decision easier to reason about. I was not building for an imaginary user. I was building for myself and people like me.

The problem is bigger than my own experience. Most people still track films in phone notes and discover what to watch through social media instead of a dedicated app - this was confirmed by a survey carried out for this project and is covered in detail in the Research chapter. Existing solutions each cover only part of the user journey - Letterboxd handles films but ignores series, TV Time covers series but has weak recommendations and no social transparency, and IMDb is a reference database rather than a personal tracking tool. The core question "what should I watch tonight?" remains poorly answered by all of them.

1.2 Objectives

The goal of this capstone project is to design, build, and evaluate a cross-platform mobile app that brings film and TV tracking, recommendations, social discovery, and analytics into one product.

The specific objectives are:

- Build a cross-platform mobile app with React Native and Expo. The same codebase runs on both Android and iOS. For this project, the app was distributed to other people as an Android APK file, while the iOS version was run on a personal device through Expo Go
- Implement a hybrid recommendation engine combining ALS collaborative filtering with TMDB Discover candidate generation and Google Gemini re-ranking
- Design and evaluate a Soulmate matching algorithm that finds users with the most similar taste using a weighted combination of five similarity metrics

- Build a Wrapped feature that generates personalized yearly viewing statistics
- Validate the recommendation model using standard information retrieval metrics and evaluate the product through real user testing

1.3 Research Approach

Before writing a single line of code, three complementary research methods were applied: a quantitative survey of 262 respondents, five qualitative in-depth interviews, and a structured competitor gap analysis covering Letterboxd, IMDb, TV Time, and Moviebase. So product decisions rested on real user needs, not guesses.

Development followed an iterative sprint model over three months. The recommendation system went through five optimization iterations - each starting with a problem, forming a hypothesis, running an experiment, and measuring the result. This approach is documented in full in the Validation section.

1.4 Structure of This Thesis

The thesis is organized as follows:

- **Research** - user research findings, competitor gap analysis, business model, and domain-specific technical challenges
- **Design** - system architecture, technology stack decisions, database schema, capacity planning, and load estimation
- **Implementation** - development sprints, UX design, key algorithms (Soulmate similarity and hybrid recommendation pipeline), and deployment setup
- **Validation** - unit testing, ML model evaluation across five iterations, manual testing with real user feedback, requirements validation, and limitations

2 Research

2.1 Research Methodology

To validate the problem and understand user needs before building MovieCrush, three complementary research methods were used:

- **Quantitative survey** - a structured questionnaire distributed to 262 respondents to measure user behavior, pain points, and feature demand at scale.
- **Qualitative user interviews** - five in-depth interviews conducted to uncover motivations, frustrations, and unmet needs that surveys cannot capture.
- **Competitor analysis** - a systematic review of existing film tracking platforms to identify gaps in the market and opportunities for differentiation.

  All three were applied before development began, so product decisions rested on user and market evidence.

2.2 Business Process Research

2.2.1 Stakeholder Analysis

The table below outlines the key stakeholders, their interests, influence, and the engagement strategy.

Stakeholder	Interest	Influence	Risk / Opportunity	Strategy
End users (film enthusiasts)	Discover content, track watches, find like-minded people	HIGH. They define product success	Risk: low loyalty, low premium conversion	Deep user research, fast iteration based on feedback
Streaming services (Netflix, MEGOGO)	Grow and retain subscribers	MEDIUM. May or may not provide affiliate opportunities	Opportunity: affiliate revenue. Risk: refusal to cooperate	Position MovieCrush as a traffic source for their platforms
Data providers (TMDb API)	Monetize their database, grow API usage	HIGH. Without their data the app is impossible	Risk: API terms change, price increases, quota limits	Use free tier at launch, plan migration to paid tier at scale
AI providers (Google Gemini)	Monetize AI models	HIGH. Recommendation quality defines product uniqueness	Risk: high API costs, unpredictable policy changes	Optimize requests, budget for AI costs from the start

Competitors (Letterboxd, TV Time)	Protect market share	MEDIUM. May copy features or run aggressive marketing	Risk: user churn	Focus on unique features (Soulmate, Wrapped)

Table 1: MovieCrush stakeholder analysis

2.3 Product Discovery and Business Analysis

Before writing any code, a business analysis was done to define what the product should be, who it is for, and how users would move through it. The full discovery board is available at: Miro Board - MovieCrush Business Analysis.

The board shows the original product vision from before development started. Some features changed or were removed during the process based on what was realistic to build - this is mentioned throughout the thesis where it matters.

2.3.1 Vision and Mission

Vision: To create a home for movie lovers - a place to share impressions, discover new films through friends, and relive every emotion. MovieCrush helps you find not just your next favorite film, but also the people who will love it with you.

Mission: To solve the "what should I watch?" problem for good. Using personalization and community to help users discover, track, and share movies they will truly love - making every viewing decision easy and every opinion heard.

2.3.2 Value Proposition

A Value Proposition Canvas was developed during discovery to map the main user problems, the jobs users are trying to do, and what MovieCrush offers in response (see Appendix AD).

2.3.3 Customer Journey

A customer journey map covers five stages - Awareness, Consideration, Purchase, Engagement, and Renewal - tracing how users discover MovieCrush, what makes them sign up, and what keeps them using it (see Appendix AD).

2.4 Quantitative Research: User Survey

The survey consisted of 5 blocks of questions:

- **Demographics:** age, content language, profession
- **Digital habits:** streaming subscriptions, monthly spend
- **Viewing habits & pain points:** how many films per week, how they currently track films, what they like/dislike about existing apps
- **Feature validation:** interest in MovieCrush features (AI recommendations, Wrapped, Soulmate, detailed ratings, challenges)
- **Monetization:** attitude to ads, willingness to pay for premium (3 USD/month)

The full survey is available here. Raw responses from all 262 respondents are available in Google Sheets.

2.4.1 Survey Distribution & Incentive

To maximize participation, the survey was distributed via Slack to the KSE student community. To motivate respondents, 4 cinema tickets were raffled among participants - 2 winners, 2 tickets each for any film and any date of their choice.

The announcement generated strong engagement - 43 reactions, well above the 2-6 typical for KSE Slack posts - and both winners later confirmed receipt of their cinema tickets. The announcement, community reactions, and winner confirmations are shown in Appendix AA.

The charts for each block are collected in Appendix AB.

2.4.2 Demographics

The survey collected 262 responses. The majority of respondents - **90.5%** - were aged 18-24, which matches both the distribution channel (the KSE community) and the primary target audience for MovieCrush. **82.7%** have active streaming subscriptions, a sign of purchasing power and a paying habit.

In terms of content language, **42.4%** consume content in a mix of Ukrainian and English, **36.3%** primarily in Ukrainian, and **21.4%** in English only - confirming that the app must support both languages. Some respondents mentioned russian-language content. But my position is clear: russian will not be supported. MovieCrush targets Ukraine, Europe, and the US - not the CIS market.

2.4.3 Viewing Habits & Pain Points

43.1% of respondents watch 2-5 films or series per week - the core active audience that needs tools for tracking and recommendations.

When asked how they currently track films, **49.8%** said they use phone notes - a clear signal that no proper tool exists. Only 19.8% use dedicated services like Letterboxd or IMDb, suggesting low awareness or poor fit with existing solutions. Additionally, 7.6% use TikTok or Instagram to save films - a new behavioral pattern no existing app addresses. 24.9% do not track films at all, representing untapped latent demand.

2.4.4 Competitor Pain Points

When asked what bothers them about existing services like IMDb and Letterboxd, respondents highlighted recurring problems:

- **Poor recommendation algorithms** - platforms suggest popular content, not personalized picks
- **No TV series or anime** - Letterboxd ignores a huge segment of content
- **No statistics** - users want a Spotify Wrapped equivalent for films
- **Outdated interface** - IMDb is losing younger users due to poor design
- **No Ukrainian language** - a barrier for the mass Ukrainian audience
- **Fragmented lists** - content saved across multiple platforms with no unified view

At the same time, users value what currently works: trust in IMDb ratings, seeing what friends watch, viewing diaries, and watchlists. These became the foundation for MovieCrush's core feature set.

2.4.5 Where Users Look for Recommendations

83.6% of respondents look for recommendations on social media (Instagram, TikTok, Twitter), and **76%** rely on friends. Only **7.3%** use specialized apps like Letterboxd or TV Time - existing platforms are failing at discovery.

58% actively discuss films with friends and family, and **37%** do so occasionally - strong demand for social features.

2.4.6 Feature Validation

Respondents were asked which features they would be willing to pay for. The top two were the **hybrid recommendation system** (42%) and **personal statistics/Wrapped** (35.9%). Social features like Soulmate (14.5%) showed lower willingness to pay but high viral potential - making them a strong fit for the free tier.

During the survey, the planned feature was described as an "AI assistant". In the final implementation it became a hybrid system combining Alternating Least Squares (ALS), TMDB Discover, and Gemini re-ranking - a more technically sound solution to the same need.

The personal statistics feature (Wrapped) received an average rating of **4.24 out of 5** - the highest of all features tested.

2.4.7 Social Features

When asked which social feature mattered most, **48.9%** chose seeing what their friends are watching and rating. **38.5%** wanted shared lists, and **35.1%** wanted to compare tastes with friends.

2.4.8 Monetization Insights

67.6% of respondents would accept ads in a free version, validating the freemium model. **23.3%** said ads would bother them even in a free version - this segment is the guaranteed base for premium conversion.

On pricing, **75.5%** of respondents would consider paying 3 USD or more per month for a premium package including AI recommendations, extended statistics, Soulmate search, and no ads. Only **9.9%** said they would not pay at all.

2.4.9 Open-ended Responses

The final open-ended question asked what would motivate users to switch to a new film app. The most frequent themes across 262 responses were:

Theme	Mentions
AI / Personalized recommendations	25+
Clean and beautiful interface	20+
Statistics / Wrapped	15+
Social features (friends, lists)	15+
Large library / anime / series	12+
Support for Ukrainian product	10+

Unique features (Soulmate, challenges)	8+
Price / free access	7+
Niche content	5+

Table 2: Most frequent themes in open-ended responses

2.4.10 Ideal Customer Profile

Based on all survey findings, the target segment is:

- **Age:** 18-24 years old (90.5% of respondents)
- **Status:** Students (32.3%) and young IT professionals (17.7%)
- **Language:** Mix of Ukrainian and English (42.4%)
- **Viewing habits:** 2-5 films/series per week (43.1%)
- **Spending:** up to 25 USD/month on subscriptions
- **Pain point:** Using phone notes to track films (49.8%)
- **Wants:** hybrid AI recommendations (42%) and personal statistics (35.9%)

> **Strong Product-Segment Fit** for: "Ukrainian tech-oriented youth aged 18-24, actively consuming content, unsatisfied with existing solutions, and willing to pay 3 USD/month for a quality personalized experience."

2.5 Qualitative Research: User Interviews

Five in-depth interviews were conducted with target users before development began. Recordings are available at: Telegram channel.

2.5.1 Methodology

In-depth interviews were chosen over surveys because open-ended questions force respondents to formulate their own opinions rather than selecting from predefined options. Individual format allows the interviewer to follow unexpected threads and uncover insights that are not on the surface. Group interviews were avoided to prevent social influence on answers.

Respondents were selected to represent different life situations: two students, one working professional, and two family respondents. The only selection criterion was an interest in watching films or series. Unlike the quantitative survey - where 90.5% of respondents were aged 18-24 - the interview participants represented a broader age range. This was intentional: to capture different behavioral patterns and avoid confirming only the assumptions of one demographic group.

Each interview was conducted at a convenient time and location for the respondent. All participants gave verbal consent to recording and further use of the data.

2.5.2 Interview Findings

Time spent searching varies dramatically. Some respondents find something in 5 minutes, others spend up to 3 hours. Regardless of the actual time, all respondents considered the search process cognitively exhausting:

> "If within 2-3 minutes I can't find anything - the desire just passes and I don't watch anything at all."

Phone notes are universally used but universally disliked. Every respondent who keeps lists uses phone notes or saved TikToks. All of them described this as inconvenient:

> "It's not convenient because there are too many actions - you always need to adjust it, add things, delete things."

TikTok and Instagram edits are the primary discovery tool. Respondents do not rely on streaming algorithms to discover new content. Instead, they use short video edits to decide whether a film matches their mood before committing to watch it.

Friends' recommendations are trusted more than ratings. IMDb ratings were specifically criticized for being misleading:

> "I don't like IMDb ratings - they often spoil my impression of a film. I would rate it 10, but IMDb shows 3."

Filters are the most requested feature. When asked what they want in an app, almost every respondent mentioned filters:

> "The more filters, the better - the faster you can find what you want."

A successful recommendation creates a strong emotional reaction. When asked how they feel after finding the perfect film, respondents used words like "euphoria" and "delight" - the emotional payoff of a good recommendation is real.

2.5.3 User Persona

Based on the interviews, the following persona was developed:

Name	Anna
Age	25

Status	Junior specialist, active schedule
Platforms	Netflix, HD Rezka, YouTube
Discovery	TikTok edits, Instagram, friends
Tracking	Phone notes - finds it inconvenient
Pain point	Spends too much time searching, recommendations don't match expectations
Motivation	Emotional relief after a busy week
Key request	Better filters, see what friends watch

Table 3: User persona derived from qualitative interviews

2.5.4 Implications for MovieCrush

The interviews directly shaped the following product decisions:

- **Mood-based filters** - respondents choose films by mood, not genre alone
- **Social feed** - seeing what friends watch is more trusted than any algorithm
- **Detailed ratings with comments** - numeric ratings are not trusted without context
- **Simple onboarding** - "the more actions required, the less likely people are to use it"
- **Watchlist and tracking** - replacing phone notes with a proper tool was a universal need

2.6 Competitor Analysis

2.6.1 Overview of Existing Solutions

The competitive landscape includes four main players, each with clear limitations.

2.6.1.1 Letterboxd

Full analysis with screenshots

Founded in 2011 in New Zealand, Letterboxd is the market leader in social film tracking with 17 million registered users as of 2024. Its strengths are a strong community, detailed film pages, and a well-known brand among film enthusiasts. The main weaknesses are that it covers films only, no TV series, has no personalized recommendations, and locks statistics behind a paid subscription. Its annual revenue is estimated at 5.5-19 million USD.

2.6.1.2 TV Time

Full analysis with screenshots

Founded in 2013 in France, TV Time has around 30 million registered users and focuses on TV series tracking. It is free for users but monetizes through selling anonymized viewing data to studios and streaming services - a model that creates a conflict of interest between the product and its users. Key weaknesses include a poor interface, no friend ratings visible, weak recommendations, and problems with Ukrainian language support.

2.6.1.3 IMDb

Full analysis with screenshots

Founded in 1990 and owned by Amazon, IMDb is the world's largest film and series database with over 670 million monthly site visits. It is a reference tool rather than a social platform - there are no friend connections or communities. Registration requires an Amazon account, the interface is overloaded, and the platform is deeply tied to the Amazon ecosystem which limits its independence as a product.

2.6.1.4 Moviebase

Full analysis with screenshots

A German indie project launched in 2017, Moviebase uses TMDB as its data source and supports 39 languages. It has good tracking features and an anti-spoiler system in comments, but has no social features at all, no gamification, and is only available on Android. 53% of its downloads come from Brazil, which limits its appeal in other markets. Annual revenue is estimated at 50,000-60,000 USD.

2.6.1.5 Market Comparison

Metric	Letterboxd	TV Time	IMDb	Moviebase
Registered users	17M	30M	100M+	500K
Content	Films only	Series + films	Films + series	Films + series
Social features	Basic	Groups only	None	None
Personalized recs	No	Poor	Basic	Basic
Statistics / Wrapped	Pro only	No	No	Pro only
Soulmate / matching	No	No	No	No
Ukrainian language	No	Partial	Yes	Yes
Revenue model	Subscriptions + ads	Data sales	Ads + Amazon	Subscriptions + ads
Est. annual revenue	5.5-19M USD	4.2M USD	224M USD	50-60K USD

Table 4: Key metrics comparison across competing platforms

Platform visibility data from AppMagic (Jan 2021 - Jan 2026) reflects the same gap: Letterboxd leads with a Featuring Score of 3.3M, while Moviebase scores just 65 [1] (see Appendix ACA).

2.6.2 Competitor Gap Analysis

The table below maps each competitor's key weaknesses to how MovieCrush addresses them.

Competitor	Key Weaknesses	How MovieCrush Addresses Them

Letterboxd	1. Films only - no TV series support 2. No personalized recommendations 3. Limited social interaction (likes and comments only)	1. Unified catalog: films, series, and individual episodes 2. Hybrid recommendation system: ALS collaborative filtering with Gemini re-ranking 3. Deep social layer: Soulmate, detailed ratings with comments, Best Actor voting
TV Time	1. Poor UX and outdated interface 2. Weak language support 3. Cast-only - no directors or screenwriters 4. Poor recommendations 5. No direct user monetization (revenue from data only) 6. Friends' ratings not visible	1. Modern, intuitive UI/UX built as a core priority 2. Full Ukrainian language support 3. Complete credits: actors, directors, screenwriters 4. Personalized recommendations via ALS + Gemini 5. Direct user monetization through premium subscription 6. Social feed showing friends' ratings and activity
IMDb	1. No social features (no friends, no communities) 2. Overloaded, cluttered interface 3. Tied to Amazon ecosystem 4. No personal analytics beyond basic counters	1. Social layer as a core product feature: follows, Soulmate 2. Clean, user-experience-focused design 3. Independent service with email-based registration 4. Deep personal analytics via Wrapped
Moviebase	1. No social features 2. No gamification 3. Analytics locked behind paywall 4. Narrow geographic reach (53% Brazilian audience)	1. Soulmate and social ratings as core free features 2. Gamification planned for future development 3. Wrapped analytics available to all users 4. Designed for Ukrainian, European, and US markets

Table 5: Competitor gap analysis and MovieCrush solutions

Core thesis: MovieCrush combines the strengths of existing platforms while eliminating their key weaknesses, and introduces unique social features.

2.7 Research Limitations

These results reflect a non-representative sample - 90.5% of respondents were aged 18-24. We found **Product-Segment Fit**, not full Product Market Fit.

The following limitations should be considered when interpreting these results:

1. **Non-representative sample** - 90.5% of respondents were aged 18-24, which correlates with the author's age and social circle.
2. **Narrow demographic** - results reflect the views of students in technical and economic fields only.
3. **Conclusion:** We cannot claim full Product Market Fit for the entire market. Instead, we found **Product-Segment Fit** for the segment of "youth aged 18-24, students, technical specializations." Validation on adjacent segments (25-34, non-IT professions) remains a goal for future work.

2.8 Economic Analysis

2.8.1 Revenue Model

MovieCrush uses a freemium model with two tiers. The free tier includes full tracking, social features, Wrapped analytics, and basic recommendations - more than most competitors offer for free. Revenue at this level comes from ads and affiliate links to streaming platforms.

The paid tier - MovieCrush Pro - unlocks the Soulmate feature, AI-powered recommendations, and Instagram story templates. These features do not exist in competing apps, which justifies a price above the market baseline.

2.8.2 Pricing Strategy

Competitor pricing ranges from 13 USD/year (Moviebase) to 49 USD/year (Letterboxd Patron). Letterboxd Pro at 19 USD/year is the closest benchmark - its main selling point is statistics, which MovieCrush gives away for free.

MovieCrush Pro is priced at **23.88 USD/year (1.99 USD/month)** for the annual plan and **2.99 USD/month** for monthly. The 1 USD/month difference between the two creates a clear reason to choose the annual plan.

Plan	Price	Per month	Includes
Free	0 USD	-	Full tracking, social features, Wrapped, ads
Pro - annual	23.88 USD/year	1.99 USD	No ads, Soulmate, AI recommendations, story templates
Pro - monthly	2.99 USD/month	2.99 USD	Same as annual

Table 6: MovieCrush pricing tiers

2.8.3 Cost Breakdown

Current monthly infrastructure costs at the student project stage:

Service	Plan	Monthly cost
Render - backend	Free tier (current)	0 USD
Render - CF service	Free tier (current)	0 USD
Render Postgres	Basic	7 USD
Google Gemini API	Free tier (current)	0 USD
Resend	Free tier	0 USD
TMDB API	Free tier	0 USD
EAS Build	Free tier	0 USD
Total (current)		**7 USD/month**
Total (at scale)		**23-26 USD/month**

Table 7: Monthly infrastructure costs

At the current stage, almost all services run on free tiers, bringing the actual monthly cost to just 7 USD - only Render Postgres requires a paid plan. At scale, moving to paid tiers for the backend, CF service, and Gemini API would bring the total to around 23-26 USD/month.

2.8.4 Investment Requirements

No external investment was needed to build and launch MovieCrush. All infrastructure runs on free or low-cost tiers. The only real cost was time. If the product were to grow beyond the current scale, the first investment would go toward moving to paid Render plans and a larger Postgres instance to handle more users.

2.8.5 Financial Projections

Using the realistic scenario of 100,000 MAU (220,000 registered users) and a 5% conversion to Pro:

Source	Annual estimate
Pro subscriptions (11,000 users multiply 23.88 USD)	262,680 USD
Ads (free tier users)	20,000-40,000 USD
Affiliate links	5,000-15,000 USD
Total	**287,000-317,000 USD**

Table 8: Revenue projection at 100,000 MAU

These numbers are projections based on market benchmarks, not real revenue. The product is at an early stage and has not yet been monetized.

2.8.6 Future Monetization Plans

The next step in monetization is affiliate partnerships with streaming platforms such as JustWatch, where MovieCrush earns a small commission when a user clicks through to

watch a film on a streaming service. This model is already used by Letterboxd and fits naturally into the product since users already look for "where to watch" information.

Selling anonymized user data was considered but rejected - it creates a conflict between the product and its users, which goes against the core idea of MovieCrush as a user-first platform.

2.9 Domain-Specific Analysis

2.9.1 Value Chain

The value chain below shows MovieCrush's dependencies and strategic position.

Participant	Role	Impact on MovieCrush
Studios / Rights holders	Own rights to films	Posters and metadata used via TMDb API - direct usage without official API carries DMCA risk
Data aggregators (TMDb)	Collect and maintain metadata: cast, descriptions, posters	Critical dependency. TMDb is the primary data source - open, well-maintained, and best suited for localization
Streaming services (Netflix, MEGOGO)	Final content providers	Their libraries define what users can watch. Their own recommendation engines are direct competitors
Competitors (Letterboxd, TV Time)	Social film tracking platforms	Set UX standards, monetization models, and user expectations

Table 9: Value chain analysis for the film tracking domain

2.9.2 Technical Challenges

Building a film tracking platform involves several domain-specific technical challenges that shaped architectural decisions in MovieCrush:

- **AI recommendation complexity.** Cinematic taste is multimodal - a person may love a director, actor, or atmosphere rather than a genre - which a simple genre filter cannot capture. MovieCrush addresses this with a hybrid ALS + Discover + Gemini pipeline (see Section 4.3).
- **Cold start.** New users have no watch history for collaborative filtering. An onboarding flow collects favorite actors and ratings to seed initial recommendations (see Section 4.3).
- **TMDb caching for analytics.** A local PostgreSQL cache of TMDb metadata avoids excessive API calls during Wrapped and recommendation building; regular browsing uses a short-lived in-memory LRU cache.

2.9.3 Legal & Economic Constraints

- **Copyright on visuals:** Using posters and metadata for informational purposes is generally accepted when sourced through official APIs like TMDb. Direct commercial use without proper licensing carries legal risk. MovieCrush uses only TMDb-provided assets within their terms of service.

3 Design

3.1 Architecture Overview and Requirements Alignment

The MovieCrush architecture targets the requirements from the domain analysis: personalized recommendations, user data storage, and social features between users.

The system is built as a modular monolith for the main backend, plus a separate machine learning microservice for recommendations. This approach keeps the main business logic clear and easy for one developer to maintain, while moving heavy ML calculations into an independent service that can be updated and scaled separately.

3.1.1 Functional Requirements

3.1.1.1 Personalized Movie Recommendations

Recommendations use a hybrid pipeline: ALS collaborative filtering generates candidates, TMDB Discover adds variety, and Gemini re-ranks the combined pool into a final categorized list, with a separate cold-start path for new users during onboarding. The full algorithm is detailed in Section 4.3.

3.1.1.2 Social Features and "Soulmate" Matching

The social system works on a follower model between users. In addition, there is a "movie soulmate" search feature - the user with the most similar taste. Similarity is calculated as a weighted combination of five metrics (rating, watched, actor, mood, and disliked overlap), detailed with their weights in Section 4.3. The resulting similarity_score is stored in PostgreSQL for fast sorting and matching. Search is only performed among users who have explicitly agreed to take part in this feature.

3.1.1.3 Comprehensive Rating System

The rating architecture supports both a simple overall score (scale 1-10) and a detailed aspect rating: direction, acting, script, music, and visual effects (each on a scale 1-5). This allows leaving either a quick rating or a detailed review.

3.1.1.4 Search and Content Discovery

Information about movies and series is obtained through the catalog module, which calls the TMDB API. The module uses lazy loading: detailed movie data is fetched only when the user opens its page. At the same time, metadata is stored in a PostgreSQL table (tmdb_media_cache), so the next request goes to the local database instead of TMDB - faster and cheaper. This cache is also used when generating annual Wrapped statistics. The cache follows a Read-Through strategy - metadata is fetched from TMDB on first access and stored locally for future use. There is currently no TTL on cached entries, as fields like title, genres, cast, and release year rarely change after a title's release.

3.1.1.5 User Engagement Features

The MovieCrush Wrapped module collects user actions over a year (views, ratings, moods, favorite actors) and builds personalized statistics. Calculations rely on cached movie metadata, so dozens of requests to TMDB are not needed at generation time.

3.1.2 Non-Functional Requirements

- **Performance**. Critical read operations are optimized with two-level caching: movie metadata is stored in a PostgreSQL table (tmdb_media_cache), and frequently used data is kept in the application's in-memory LRU cache. This reduces load on both the external TMDB API and the database. Gemini re-ranking results are cached for 24 hours.
- **Data Consistency**. For critical operations (ratings, lists, social connections), ACID transactions in PostgreSQL are used. Integrity is also ensured by foreign keys and constraints at the database schema level.
- **Scalability**. The modular monolith with clear boundaries between modules (auth, profile, movies, soulmate, recommendations, and so on) allows individual parts to be extracted into independent services in the future. The ML component is already a separate service, deployed and scaled independently.
- **Security**. The system uses JWT authentication with short-lived access tokens and long-lived refresh tokens. Passwords are hashed with the bcrypt algorithm. The API is protected by security HTTP headers (helmet), rate limiting, and CORS policies. The mobile app communicates with the backend over HTTPS - Render automatically provides a TLS certificate for all public services.
- **Reliability**. Both services run as Docker containers, and the database is co-located with the backend, removing an extra network hop. The recommendation model is currently retrained manually; scheduled retraining is planned (see Limitations).

3.2 System Architecture and Major Decisions

3.2.1 Modular Monolith Architecture Decision

Decision: Implement a modular monolith architecture with clear boundaries between domains instead of a microservices architecture.

Reasoning: Given the team size (one developer), project timeline, and current system scale, a modular monolith provides several benefits:

- **Development efficiency.** One developer does not waste time setting up inter-service communication, distributed queues, and separate deployments for each service. This allows faster feature implementation and easier system debugging.
- **Data consistency.** Critical operations, such as marking a movie as watched, update several tables at once: user_movie_actions, user_lists, and counters in users. All of this runs inside a single ACID PostgreSQL transaction - no complex distributed transaction logic between services.
- **Deployment simplicity.** One Docker container - one deployment. The CI/CD pipeline stays simple and predictable.

- **Future scalability.** The system modules have clear boundaries (auth, profile, movies, lists, follows, soulmate, recommendations, wrapped, etc.) so if needed, a single module can be extracted into an independent service without rewriting the whole application. The resource-heavy ML component is already separated this way - as a separate CF service on Python.

Trade-offs considered: A microservices architecture provides independent scaling for each component and technological flexibility. However, its complexity and operational costs greatly outweigh the benefits for the current project with one developer.

3.3 System Context and External Interactions

The system context diagram shows where MovieCrush sits among external services and users. It integrates with external services for content, AI, and email.

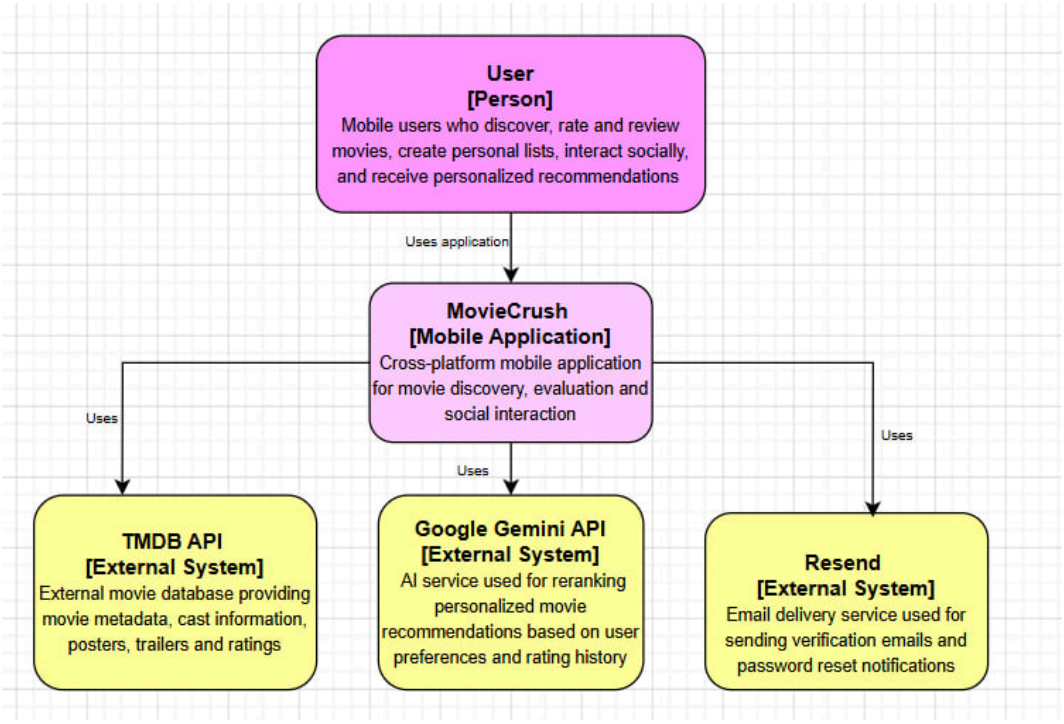


Figure 1: System Context Diagram - MovieCrush Platform Ecosystem. Available here

3.3.1 Main External Integrations

3.3.1.1 TMDB API

Provides the full catalog of movies and series: metadata, actor information, posters, backdrops, trailers, and external ratings. To stay within free tier limits, two caching levels are implemented: an in-memory LRU cache with a 5-minute TTL for hot requests, and permanent metadata storage in a PostgreSQL table (tmdb_media_cache) for data that does not change often (title, genres, cast, release year).

3.3.1.2 Google Gemini 2.5 Flash

Used for re-ranking recommendations. It receives a candidate list from the ALS model and TMDB Discover, analyzes the user's taste based on their ratings, and picks the 15 best options, assigning a category to each. Results are cached in the database for 24 hours to avoid calling the model on every request.

3.3.1.3 Resend

A transactional email service used to send email verification letters during registration and password recovery emails.

3.3.2 Mobile Users

The main users of the platform, who interact with MovieCrush through a mobile app for Android and iOS built on React Native/Expo. The app is built as an APK for Android using EAS Build, the iOS version is cross-platform compatible but requires an Apple Developer account for public distribution.

Users can: search and discover new movies and series, leave detailed ratings, create their own lists, follow other users and find their movie soulmate, and receive personalized recommendations.

3.4 Container Architecture and Service Decomposition

The architecture is described at two levels of the C4 model. The container diagram gives the high-level technical view - the mobile app, the backend as a modular monolith, the separate machine learning (CF) service, the database, and external integrations - and the component diagram then zooms into the backend container to show its internal modules.

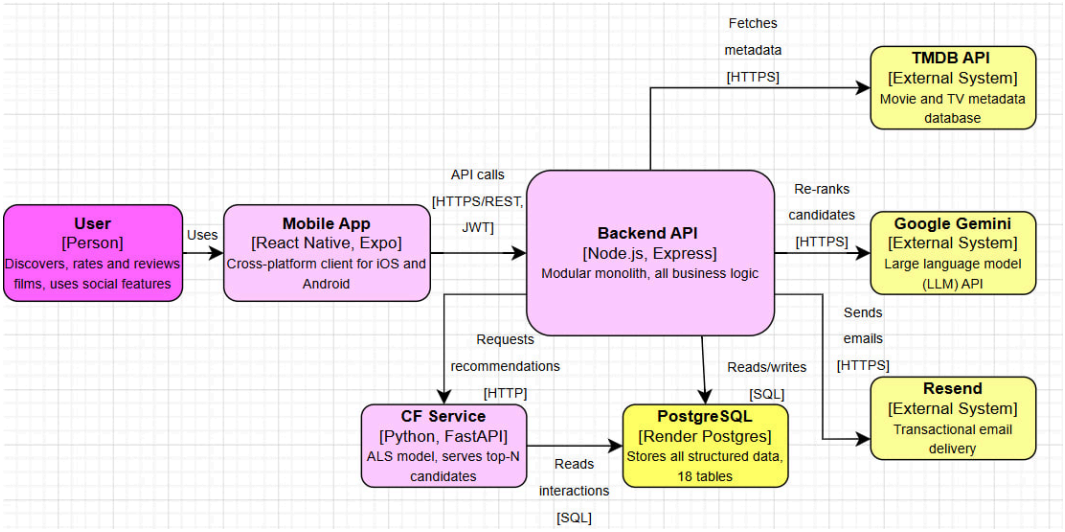


Figure 2: Container Diagram - MovieCrush Architecture and Data Flows. Available here

The container diagram shows how the parts communicate: the mobile app calls the backend over HTTPS/REST with JWT authentication, the backend persists all data in PostgreSQL, requests recommendation candidates from the CF service over HTTP, and integrates with TMDB, Google Gemini, and Resend. The CF service reads user interactions

from the same database. Hosting on Render is shown as a deployment annotation on the containers rather than as a separate node, since it is infrastructure rather than a domain integration.

Zooming into the backend, the component diagram shows its internal structure as a modular monolith - eleven feature modules inside a single deployable container, where only the modules that need external data reach out to the CF service, Gemini, TMDB, and Resend.

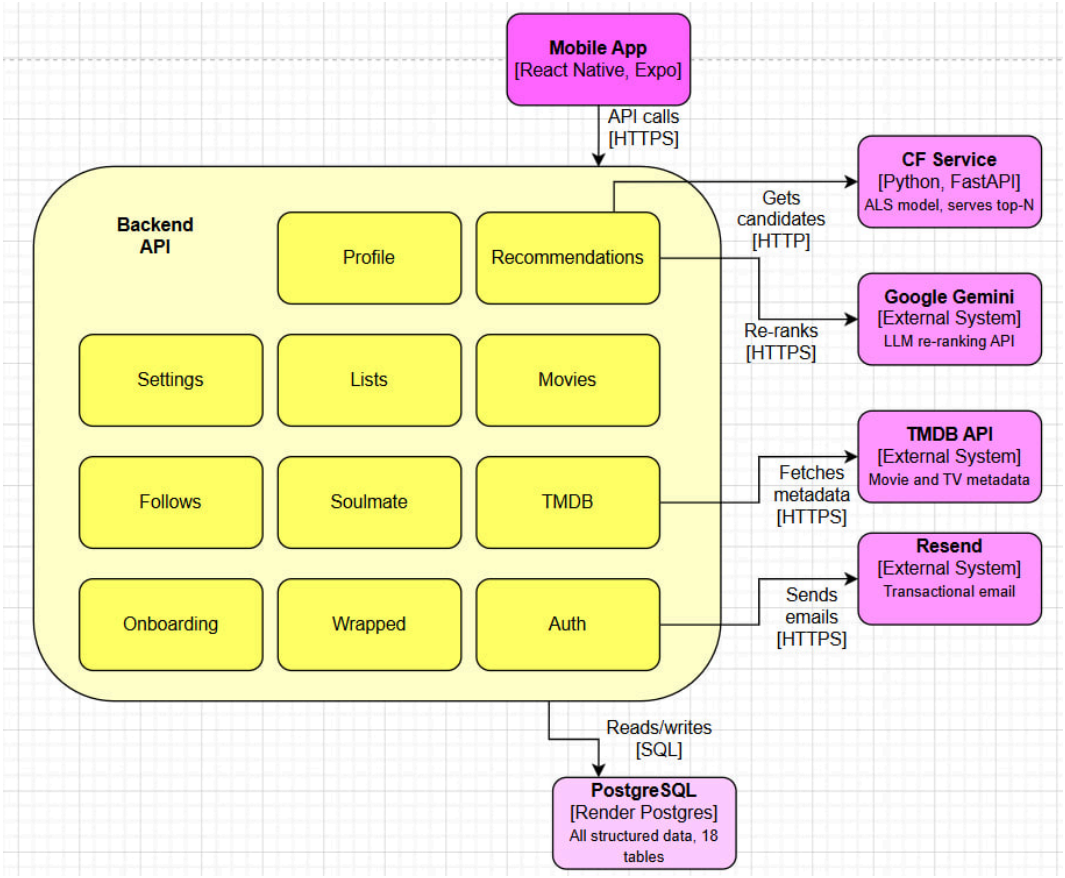


Figure 3: Component Diagram - Backend API internal modules. Available here

3.4.1 Backend Modules

The backend is organized into eleven feature modules, each owning its own routes, controller, and service. Two additional folders - **`shared`** (common user queries and types) and **`tmdb_cache`** (an internal caching service) - support these modules but are not stand-alone modules, so they are not counted among the eleven. Modules that need movie data (Movies, Recommendations, Follows) do not call the TMDB API directly; they access it through the TMDB module, which is the single integration point with the external TMDB API.

Module	Responsibility	Key Functions
Auth	Authentication	Registration, login, email verification, password reset

Module	Responsibility	Key Functions
Profile	User profile	View and edit name, avatar, Instagram/ Telegram links
Settings	Configuration	Username, password, soulmate consent, account deletion
Movies	Movies and series	Ratings (1-10 + 5-criteria detailed score), mood tagging, comments, best actor vote, episode tracking
Lists	Lists	Watchlist, favorites, watched, custom lists with privacy toggle
Follows	Social connections	Follow/unfollow, view friends' ratings on a specific title, user search
Soulmate	Soulmate search	Similarity score across 5 metrics, result storage, on-demand recompute
Recommendations	Recommendations	Hybrid pipeline: ALS via CF service + Gemini reranking, cold start for new users
TMDB	Catalog	Movie/series/person details, search, trailers, cast via TMDB API with LRU cache
Onboarding	Onboarding	Favourite actors and movies selection during signup for cold start
Wrapped	Yearly statistics	top genres, actors, directors, moods, total watch time, cinema vibe

Table 10: Backend modules and their responsibilities

3.4.2 Data Storage Layers

Storage	Purpose
PostgreSQL	Main database - 18 tables, ACID transactions. Stores everything: accounts, ratings, lists, social connections, TMDB metadata cache, recommendation results
LRU in-memory cache	Caches TMDB API responses at the process level, TTL 5 minutes, maximum 500 entries
tmdb_media_cache (PostgreSQL)	Permanent storage of movie metadata (title, genres, cast, year). Used by Wrapped and recommendations
user_ai_recommendations (PostgreSQL)	Cache for Gemini re-ranking results, TTL 24 hours

Table 11: Data storage layers and their purposes

3.5 Technology Stack Selection

3.5.1 Frontend - Mobile Application

Technology	Decision	Justification
React Native	Cross-platform mobile framework	One codebase for both iOS and Android. As a solo developer, building two separate native apps was not realistic. Expo adds helpful build tools and allows updates without going through app stores.
Expo + EAS Build	Build and distribution	EAS Build creates the APK in the cloud, so no local Mac is needed. For Android testing, the APK is shared directly without a Play Store account.
TypeScript	Type system	Catches type errors at compile time. With many API response shapes across the app, having typed interfaces makes it much easier to find mismatches between the mobile client and the backend.
React Navigation	Navigation	The standard navigation library for React Native. Works well on both platforms and handles all the screen changes in the app.

Table 12: Frontend technology decisions

3.5.2 Backend

Technology	Decision	Justification
Node.js	Runtime	JavaScript on the server means the same language as the mobile client, which reduces switching between languages. The non-blocking I/O model also works well for an API that makes many requests at the same time to TMDB, Gemini, and the CF service.
TypeScript	Type system	Strict mode is turned on across the whole codebase. Typed request and response interfaces make code changes safer and reduce the chance of runtime errors in production.

Technology	Decision	Justification
Express.js	HTTP framework	Lightweight and simple, which fits the modular monolith structure well. Each module sets up its own router, so the codebase stays organised without the extra weight of a bigger framework.
pg	PostgreSQL client	Raw SQL gives full control over queries, which matters for complex joins in the soulmate algorithm and the Wrapped calculations.
bcryptjs	Password hashing	Standard for storing passwords securely. Passwords are never stored as plain text.
jsonwebtoken	Authentication	JWT with separate short-lived access and long-lived refresh tokens. Stateless tokens work well for a mobile client that cannot keep server-side sessions.
helmet	Security headers	Sets HTTP security headers with a single middleware call.
express-rate-limit	Rate limiting	Limits each IP to 200 requests per minute on all `/api/*` routes.
lru-cache	In-memory caching	Saves TMDB API responses in process memory with a 5-minute lifetime and a maximum of 500 entries. No separate caching service like Redis is needed.
resend	Email delivery	Handles verification and password reset emails. Simpler to set up than self-hosted email and has better delivery success.
Jest + ts-jest	Unit testing	Used to test the soulmate similarity algorithm, ALS service logic, Wrapped calculations, auth validators, episode helpers, and TMDB helpers.

Table 13: Backend technology decisions

3.5.3 CF Service - Collaborative Filtering

Technology	Decision	Justification
Python	Language	The ML ecosystem in Python is much more mature than in Node.js. Running the model in a separate Python service keeps the backend clean and free of heavy libraries.

Technology	Decision	Justification
FastAPI	HTTP framework	Async-friendly and minimal. Faster to write than Flask for a service that only exposes a few endpoints.
implicit	ALS library	Provides a fast ALS implementation for feedback data. BM25 weighting is applied to the user-item matrix before training to account for different interaction strengths - watched, rated, favourited.
scipy	Sparse matrices	The user-item matrix is mostly empty by nature. scipy's CSR format stores only non-zero entries, which keeps training and prediction memory-efficient.
scikit-learn + matplotlib	Evaluation	Used offline to measure model quality and to tune settings before deploying the trained model.
psycopg2	PostgreSQL client	Reads user interaction data directly from the shared PostgreSQL database to build the training matrix.

Table 14: CF service technology decisions

3.5.4 Database

Technology	Decision	Justification
PostgreSQL	Primary database	ACID-compliant relational database. The data model - users, ratings, lists, social connections, soulmate scores, recommendation cache - is relational with many foreign key constraints and multi-table transactions. JSONB columns are used to store cast data in `tmdb_media_cache`.

Table 15: Database technology decisions

3.5.5 Infrastructure and Deployment

Technology	Decision	Justification
Docker	Containerisation	Multi-stage images (separate build and runtime stages) keep the final runtime small.
Render	Cloud hosting	Docker-based deploy with automatic HTTPS and no infrastructure management; chosen after AWS was dropped. Full migration rationale in Section 4.4.

Technology	Decision	Justification
Render Postgres	Managed PostgreSQL	Managed PostgreSQL co-located with the backend, which removes the public-internet hop and keeps the database off the public network; automatic daily backups. Migration story in Section 4.4.
GitHub Actions	CI/CD	Runs TypeScript checks, the Jest suite, a production build, and Docker image builds on every push. Pipeline detailed in Section 4.4.

Table 16: Infrastructure and deployment decisions

3.6 Cross-cutting Concerns

3.6.1 Security

3.6.1.1 Authentication and Authorisation

The system uses JWT authentication with two separate tokens. After a successful login, the user receives an access token that lives for 15 minutes and a refresh token that lives for 30 days. The access token is sent in the Authorization header as **`Bearer <token>`** with every API request. The authMiddleware checks the token signature and makes sure the token type is access, since refresh tokens are rejected on protected endpoints. All backend modules apply this middleware before handling any request.

3.6.1.2 Passwords

Passwords are hashed using bcryptjs before saving. Plain text passwords are never stored anywhere in the system.

3.6.1.3 Data Transfer Security

All communication between the mobile app and the backend happens over HTTPS. Render automatically provides and renews TLS certificates for all public services, so manual certificate management is not needed.

3.6.1.4 API Protection

The backend uses helmet to set security HTTP headers on every response, and express-rate-limit to limit each IP address to 200 requests per minute on all **`/api/*`** routes.

3.6.2 Monitoring and Observability

The current monitoring approach is deliberately simple and matches the scale of the project.

3.6.2.1 Logging

The backend uses structured logging via the pino library instead of plain console output. A shared logger instance writes JSON-formatted logs, with the log level controlled by an environment variable (info in production, debug in development, with pino-pretty for readable local output). HTTP requests are logged automatically through the pino-http

middleware, which records method, URL, and status code for every request, while application code logs key events - startup, database connection status, and errors - through the same logger. Render automatically captures this output and shows it in the service dashboard. The CF service follows the same principle using Python's standard logging module, configured centrally with timestamp, level, and logger name, and used for model training progress and recommendation requests.

3.6.2.2 Health Check

The backend has a **`/health`** endpoint that returns **`{ status: "ok" }`**. Render uses this to monitor service availability and automatically restarts the container if it stops responding.

3.6.2.3 Possible Improvements

For a production system with higher load, the next steps would be centralised log aggregation across both services, request tracing with correlation IDs propagated from the backend to the CF service, and automated alerts on error rates and response time. In the current version of the project, this was outside the scope.

3.6.3 Error Handling

All database operations are wrapped in try/catch blocks. For critical operations that update several tables at once - for example, marking a movie as watched, which updates user_movie_actions, user_lists, and counters in users - explicit transactions with BEGIN, COMMIT, and ROLLBACK are used. If any step inside a transaction fails with an error, all changes are undone and the database stays in a consistent state.

For calls to third-party services (TMDB, Gemini), the current handling is a plain try/catch with a graceful fallback. A production-grade improvement would be a circuit breaker with retry: transient failures (e.g. a 500) are retried after a short delay, and if a service keeps failing, the breaker opens and requests are short-circuited for a cooldown period instead of repeatedly hitting a service that is already down [2]. This is noted as a future improvement.

3.6.4 Data Consistency

The database schema ensures data integrity at the PostgreSQL level. All tables with user data have foreign keys with ON DELETE CASCADE - for example, ratings, lists, comments, follows, and soulmate matches are automatically deleted if the user itself is deleted. This makes sure no records are left in the database without a link to an existing account.

3.7 Database Design

The database schema was designed before implementation, with each table in terms of field types, constraints, and storage requirements. For each table, the maximum size per row was calculated in bytes, and then projected across three growth scenarios to estimate total storage needs.

3.7.1 Schema Overview

The final implementation contains 18 tables. The schema went through several revisions during development - some tables from the original plan were removed (for example, a separate **`movies`** table was replaced by the **`tmdb_media_cache`** approach), and new ones were added based on emerging requirements.

3.7.2 Storage Estimation Methodology

For each table, the row size was calculated by summing the maximum byte size of each field. Three projection scenarios were used:

Scenario	MAU	Registered users (× 2.2)
Conservative	30 000	66 000
Realistic	100 000	220 000
Optimistic	250 000	550 000

Table 17: Growth scenarios used for storage estimation

The 2.2 coefficient converts MAU to total registered users - it accounts for the typical ratio between accumulated and active audiences in social apps, based on the assumption that around 45-55% of registered users remain active after 12 months.

For the content catalog, the Pareto principle (80/20 rule) was applied: since roughly 20% of titles receive 80% of user interactions, only 294,000 out of 1.26 million TMDB movies were considered realistic candidates for active caching. This avoided overestimating storage for the media cache table.

User behaviour assumptions were also applied per table - for example, comments were estimated using a 90/9/1 split: 90% of users write zero comments, 9% write 1-5, and 1% write 10-50+, giving an average of 0.47 comments per user.

3.7.3 Key Design Decisions

Dual ID system in users table. Each user has an internal **`BIGINT id`** used for all foreign key relationships and joins, and a separate **`CHAR(36) uuid`** exposed in public-facing APIs. This prevents internal IDs from being enumerable in API responses.

Counter columns in users table. Fields like **`movies_watched`**, **`followers_count`**, and **`friends_count`** are stored as denormalised counters updated by application logic rather than computed on every query. This trades a small write overhead for faster profile reads.

JSONB for cast data. The **`top_cast`** column in **`tmdb_media_cache`** stores the top 10 actors as a JSONB array rather than a separate join table. Since cast data is always read as a unit and never queried individually, this avoids an unnecessary join on every movie detail request.

ON DELETE CASCADE on all user-owned data. Every table that stores user content - ratings, lists, comments, follows, soulmate matches - has a foreign key to **`users(id)`** with **`ON DELETE CASCADE`**. Deleting a user account automatically removes all associated data without requiring application-level cleanup logic.

The full storage calculations and per-field justifications were documented during the discovery phase and are available at: Data Modelling Spreadsheet.

3.8 User Growth Projections

To estimate system load, realistic user growth targets were defined first. Benchmarking against competitors was considered but rejected because Letterboxd has 17 million monthly users after 13 years on the market, which is not a useful baseline for a new product.

Instead, growth projections were built around four key drivers specific to MovieCrush:

- **Wrapped virality** - the strongest driver, similar to how Spotify Wrapped generates millions of social media posts every December
- **Soulmate feature** - social motivation to invite friends and find matches
- **AI recommendations** - a useful feature that drives retention
- **Paid marketing** - eventual budget for user acquisition

Three scenarios were defined:

Scenario	Year 1 MAU	Year 2 MAU	Year 3 MAU
Conservative	30,000	100,000	250,000
Realistic	100,000	500,000	1,000,000
Optimistic	250,000	1,200,000	2,500,000

Table 18: MAU growth projections across three scenarios

The realistic scenario is grounded in four concrete arguments:

1. Ukraine has 30 million people. Capturing 0.3% of the population gives 100,000 users - a realistic target for a quality niche app.
2. If 20% of 100,000 users share their Wrapped report on Instagram (20,000 posts) and each brings 2 new users, that adds 40,000 users through virality alone.
3. Expanding to the US and Europe is a planned next step after the Ukrainian market is established. These markets are larger and would make 500 000+ users achievable in year 2-3 with the right positioning.
4. Comparable social apps at launch - Goodreads, early Duolingo - showed similar growth patterns in their first two years.

DAU is estimated at 20% of MAU, which is a standard ratio for social and entertainment apps. These numbers feed directly into the load estimation in the next section.

3.9 Capacity Planning and Load Estimation

Before making architectural decisions, a load estimation was done to understand how many requests the system would need to handle at different growth stages. The full calculations are available at: Load Estimation Spreadsheet.

One user makes roughly 81 requests per day across all actions: opening the app, searching for films, viewing movie pages, rating, commenting, and browsing recommendations. Using 20% of MAU as DAU, requests per second (RPS) were calculated for three scenarios:

Scenario	MAU	DAU	RPS
Conservative - Year 1	30,000	6,000	6
Conservative - Year 2	100,000	20,000	20
Realistic - Year 1	100,000	20,000	20
Realistic - Year 2	500,000	100,000	90
Optimistic - Year 1	250,000	50,000	50
Optimistic - Year 2	1,200,000	240,000	220

Table 19: RPS estimates across growth scenarios

Request types follow a read-heavy pattern: 70% GET (browsing movies, profiles, recommendations), 20% POST (ratings, comments, lists), 8% PUT (profile and settings updates), and 2% DELETE.

The key findings from this analysis directly shaped architectural decisions:

- **6 RPS (conservative Year 1)** - a single server handles this comfortably. This confirmed that starting with a modular monolith on a single Render instance was the right call, with no need for load balancing at launch.
- **90 RPS (realistic Year 2)** - at this level, caching becomes critical. This justified the two-level caching strategy: in-memory LRU for TMDB responses and PostgreSQL tables for recommendation results.
- **220+ RPS (optimistic)** - would require horizontal scaling and potentially extracting high-traffic modules into separate services. The modular monolith structure was chosen specifically to make this transition possible without rewriting the whole system.

4 Implementation

4.1 Development Sprints

MovieCrush was built over three months in an iterative way, split into three sprints. Before the first sprint, a full discovery phase was completed and presented as a product pitch.

4.1.1 Initial Presentation - Discovery Phase

Before any coding started, a detailed product presentation was prepared. At this stage the following was completed:

- Vision and Mission of the product
- Goals and reasoning for why the idea is promising
- Full market analysis and competitor research
- Description of unique features and how they differ from existing solutions
- Target audience profile and stakeholder analysis
- Business analysis: monetization model, pricing strategy, user growth projections
- Risk assessment and mitigation plan
- First logo ideas
- Design mockups in Figma for five key screens

Deliverables set for Sprint 1:
- Software Architecture Description
- Technology Stack Selection & Justification
- Cost Breakdown & Initial Infrastructure Setup
- Business Process Research Finalization
- MVP implementation

4.1.2 Sprint 1 - Month 1: Research and Architecture

Goals: finish the research phase, design the architecture, and prepare the technical base before coding began.

What was done:

The main outcome of this sprint was a large quantitative study: a survey of 262 respondents distributed through the KSE Slack community. It measured demand for product features, willingness to pay, and current pain points of the target audience. After analyzing the data, charts, and conclusions for each feature - AI recommendations, Wrapped, Soulmate, detailed ratings - Product-Segment Fit was confirmed for the 18-24 age group.

In parallel, data modeling was completed: 26 tables were designed with data types, sizes, and constraints defined, entity relationships and join tables were built (movie_genres, movie_countries). Load was estimated across three growth scenarios, and the 90/9/1 rule was applied to the comments table. Total database size at full load was estimated at 6.4 GB. RPS was also calculated for conservative, realistic, and optimistic scenarios.

AI recommendations were also scoped this sprint. Three services were reviewed - OpenAI, DeepSeek, and Hugging Face Inference API - along with two recommendation strategies: Embedding API + similarity and LLM generation. DeepSeek was selected as the external AI service, and the idea of an in-app AI chat (similar to ChatGPT but limited to film topics) was dropped in favor of a single AI assistant button that generates personalized recommendations.

A preliminary tech stack was defined (Redis for caching, DeepSeek API, AWS for deployment) along with initial architecture diagrams. Both the stack and the diagrams changed significantly during implementation - this is covered in the relevant sections.

Retrospective: No coding was done in this sprint intentionally - the priority was solid research and architecture planning. The number of database tables changed between planning and implementation: from 26 designed to 18 built, as some tables turned out to be unnecessary or were replaced by a different approach (for example, a separate movies table was replaced by the tmdb_media_cache strategy).

Deliverables set for Sprint 2:
- Stack setup
- Registration and login
- Movie search
- Adding movies to lists and creating custom lists
- Rating movies and writing comments
- Movie, series, and actor detail pages
- Following other users

4.1.3 Sprint 2 - Month 2: MVP

Goals: build the core of the product - authentication, content catalog, rating system, and main screens.

What was done:

Authentication. A full auth flow was built: registration with field validation, email confirmation, login, and password recovery. Validation runs on the client before any request, with per-field errors shown on a touched state. On the backend, bcrypt verifies passwords, and the comparison always runs - against a dummy hash even when the user does not exist - so account existence is not leaked through response timing. After login, the access and refresh tokens are stored and navigation resets to Home, so the user cannot return into the auth flow.

Core screens. The main screens - Home, Search, Movie, Series, Person, Profile, Settings, and the cold-start Recommendation screen - were implemented; their layout and visuals are covered in the UX section and the screen gallery (Section AC). A few implementation details are worth noting: search uses a 500 ms debounce with a 2-character minimum, and the Home and Movie screens fetch all their data in parallel via `Promise.all`, so returning to a screen triggers no refetch. When a movie page opens, its TMDB data - details, cast, gallery, videos, and similar titles - is fetched together with the user's own rating and list state in a single batched request.

Data protection. Every state-changing request passes through **`authMiddleware`**, which returns 401 without a valid access token and rejects refresh tokens on protected endpoints. Ownership is enforced at the query level - deleting a custom list, for example, checks both the list ID and that it belongs to the current user.

Content storage. The movie catalog is not duplicated in the database - only **`tmdb_id`** references are stored, and full metadata is fetched from TMDB on demand and cached.

Retrospective: Following other users was not finished in this sprint and was moved to Sprint 3 as a priority. It also became clear that DeepSeek would not work for AI recommendations - the committee pointed out that the model could ignore a large prompt or return irrelevant results. Finding an alternative became a task for the next sprint.

Deliverables set for Sprint 3:
- Wrapped analytics
- Improved AI recommendations
- Soulmate matching algorithm
- Following users and social features
- Challenges, Instagram share templates, language switch, PDF export, premiere notifications (planned, not implemented due to reprioritization)

4.1.4 Sprint 3 - Month 3: Key Features and AI

Goals: build the features that make MovieCrush different - Soulmate, Wrapped, the social layer, and AI recommendations. Rethink the recommendation engine.

What was done:

Social Features / Follows. A full follower model was built: following and unfollowing other users, viewing their public and default lists, and mutual follow status (friends). When two users are friends, their Instagram or Telegram contacts can optionally be visible to each other. Users can be found through the global search. On movie, series, and episode pages, ratings from followed users are now shown.

Soulmate. An opt-in feature (toggled in Settings) that finds the user with the most similar film taste. The weighted five-metric scoring, eligibility filtering, and the once-per-day recompute are described in full in Section 4.3; this sprint delivered the feature end to end, from candidate filtering to the result UI.

MovieCrush Wrapped. Personalized yearly statistics: total watch time, number of movies, series, and episodes, top genre, top director, top actors, most common mood after watching, typical watch time (cinema ritual), and time spent with the favorite actor. Calculations use cached metadata from tmdb_media_cache - no TMDB requests are needed at generation time. Regeneration is available once per day for testing purposes.

AI Recommendations - New approach. During this sprint, a research study was done: 9 models were tested with 5 different prompts each - Claude Opus 4.7, Claude Haiku 4.7, Claude Sonnet 4.6, GPT 5.4, GPT 5.4 mini, GPT 5.5, Gemini Flash, Gemini Pro, and Gemini Thinking. The prompt design was informed by two papers: [3] recommended passing up to 75 movies with ratings and is_liked/is_disliked flags, and using a composition rule for result categories (strong_match, diversity, hidden_gem). [4] showed that prompt

structure matters more than model choice, and that cheaper models like Gemini Flash give comparable quality at 90% lower cost.

The full model comparison, prompt designs, and per-model outputs are documented at: AI Recommendations Research - Notion.

However, the committee raised concerns about a purely LLM-based approach: the model couldn't return art-house picks, get lost in a long prompt, or miss context when given many movies. This led to the decision to move to a hybrid pipeline with collaborative filtering.

CF Service - New recommendation engine. A separate Python/FastAPI microservice was built around an ALS collaborative-filtering model, with Gemini repurposed from the main recommendation source into a re-ranker over a pre-filtered candidate pool. An onboarding flow was added to collect initial signals (favorite actors and movies) so new users get recommendations until ALS has enough data. The full three-stage pipeline and the cold-start path are detailed in Section 4.3; this sprint delivered the working end-to-end implementation.

Retrospective: Several planned deliverables were intentionally not built - after reviewing survey results, the top priorities were clear: Wrapped (rated 4.24/5), AI recommendations, and social features. Not implemented: Challenges, Instagram share templates, language switch, PDF export, and premiere push notifications. These are planned for future development.

4.2 UX Design

4.2.1 Design Philosophy

MovieCrush uses a dark theme as the only design option. This was a deliberate choice - film posters are colorful by nature, and a dark background makes them stand out without competing visual noise. All screens share the same color palette, typography, and spacing system for a consistent feel.

4.2.2 Visual Identity

Color palette. The palette uses four main colors: pure black (**`#000000`**) as the background, gold (**`#FFD700`**) as the primary action color, soft pink (**`#FFAFCC`**) as the accent used for the logo and headings, and white (**`#FFFFFF`**) for body text. Card surfaces use **`#111111`** and **`#1A1A1A`** to create subtle depth without breaking the dark theme. Error states use red (**`#FF4D4D`**) and success states use green (**`#00CC66`**).

Typography. The entire app uses Poppins - a geometric sans-serif font with four weights: Regular, Medium, SemiBold, and Bold. Poppins was chosen for its clean look on small screens and its good readability at both large display sizes and small caption sizes. All font constants are centralized in a single file so that changing a weight applies consistently across the whole app.

Logo. The MovieCrush logo combines the letters M and C using the brand's gold-to-pink gradient. The C is styled to suggest a heart shape, reinforcing the "crush" concept. The icon was designed to hold up at small sizes - as seen on the home screen and in the app drawer alongside Netflix and Spotify.

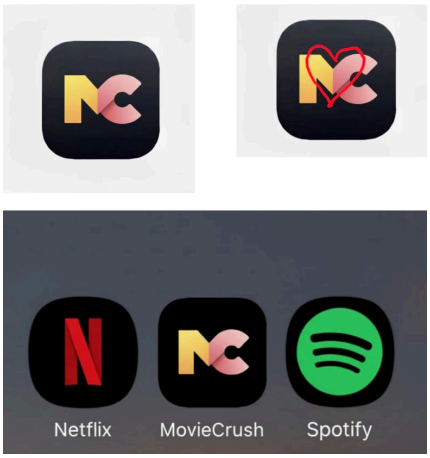


Figure 4: MovieCrush app icon

4.2.3 Screen Overview

The three screens below carry the core experience. The remaining screens - Onboarding, Home, Search, Series, Profile, and Actor - are collected in the screen gallery (Appendix AC).

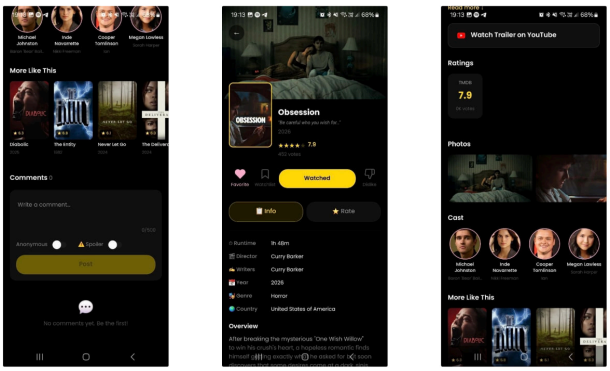


Figure 5: Movie Screen - hero backdrop, action bar, Info and Rate tabs

The Movie Screen is the heart of the app: a full-width backdrop with the poster overlaid, four quick actions (Favorite, Watchlist, Watched, Dislike), and two tabs - Info (details, trailer, gallery, cast, similar titles, comments) and Rate.

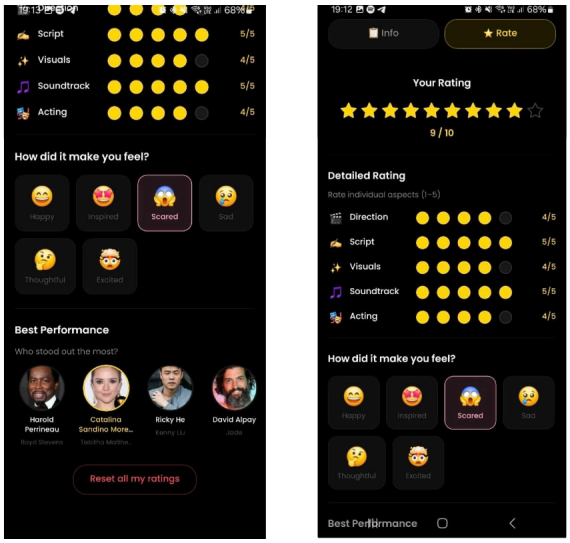


Figure 6: Rate tab - star score, 5-criteria breakdown, mood picker, best performance

The Rate tab carries the full rating system: a 10-star overall score, a breakdown across five criteria (Direction, Script, Visuals, Soundtrack, Acting - each 1-5), a six-option mood picker, and a Best Performance vote.

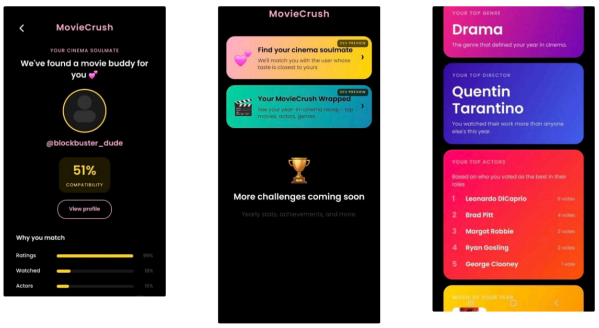


Figure 7: Challenges Screen - Soulmate result and Wrapped yearly stats

The Challenges Screen hosts the two signature features. Soulmate shows the matched user with a compatibility percentage and a per-metric breakdown; Wrapped presents Spotify-style full-screen slides for top genre, director, actors, and more.

4.2.4 Design Decisions

Figma mockups. Initial screen layouts were designed in Figma before coding began. The mockups covered the five core screens. During implementation some layouts changed - new ideas came up and certain components were improved beyond the original design. The Figma file is available at: MovieCrush Figma.

Logo concept. The logo ideas are documented at: Logo Ideas - Notion.

Dark theme only. A light theme was considered but dropped - the app is used mostly in low-light settings, and poster art reads better on a dark background.

Emoji as visual language. Section headers and filter chips use emoji alongside text labels. This was a deliberate choice to keep the interface feeling friendly and expressive rather than corporate - consistent with the target audience of 18-24 year olds.

4.3 Key Algorithms

4.3.1 Soulmate Similarity Algorithm

The Soulmate feature finds the user whose film taste is closest to yours. The algorithm computes a similarity score between two users based on five independent metrics, combines them into a single weighted score, and saves the best match if it passes a minimum threshold.

4.3.1.1 Five Similarity Metrics

Each metric measures a different dimension of taste overlap and uses a different mathematical technique suited to the nature of the data.

Metric 1 - Rating Cosine Similarity (weight: 0.45). This is the strongest signal. For every movie both users have rated, the algorithm builds two parallel numeric vectors and computes cosine similarity between them: a user who consistently rates 9/10 and a user who rates 7/10 for the same films still show high similarity if their relative preferences match. A minimum overlap of 3 jointly rated movies is required - below that, the score defaults to 0.

Metric 2 - Watched Overlap / Jaccard (weight: 0.22). Measures what share of total watched titles both users have in common, so the score naturally accounts for users with very different library sizes - a user with 10 watched movies and a user with 200 are compared fairly.

Metric 3 - Actor Overlap / Jaccard (weight: 0.16). Each time a user rates a movie, they can vote for the best actor in that film. This metric computes Jaccard Index (Jaccard similarity) similarity between the two users' sets of voted actors - capturing a deeper layer of taste alignment that goes beyond genre preferences.

Metric 4 - Mood Cosine Similarity (weight: 0.11). After watching a film, users select a mood: Happy, Inspired, Scared, Sad, Thoughtful, or Excited. Each user's mood history is treated as a frequency vector - how many times each mood was selected - and Cosine Similarity (cosine similarity) is computed across the shared mood space. Two users who consistently feel the same emotions after watching the same types of films are considered more compatible.

Metric 5 - Disliked Overlap / Jaccard (weight: 0.06). Shared dislikes are a weak but real signal - if two users consistently dislike the same films, their tastes are likely to align in the opposite direction as well. Jaccard similarity is computed on each user's set of disliked titles.

4.3.1.2 Weighted Score

The five metrics are combined into a single final score using fixed weights that sum to exactly 1.0. Rating carries the highest weight because numeric ratings are the most explicit and precise signal a user provides. Watched overlap is second - a shared viewing history is

a strong implicit signal even without explicit ratings. Actor preferences and mood patterns follow as secondary signals. Shared dislikes have the lowest weight: agreement on what to avoid is meaningful but less decisive than positive preference alignment.

Metric	Method	Weight
Rating similarity	Cosine similarity	0.45
Watched overlap	Jaccard similarity	0.22
Actor overlap	Jaccard similarity	0.16
Mood similarity	Cosine similarity (bag of counts)	0.11
Disliked overlap	Jaccard similarity	0.06

Table 20: Soulmate similarity metrics and their weights

4.3.1.3 Full Search Flow

The algorithm first filters eligible candidates - active users who have given consent to the feature and have watched at least 5 movies. For each candidate, all five metrics are computed in parallel to minimize latency, and the weighted score is calculated. The algorithm keeps track of the highest-scoring candidate across all comparisons.

A match is saved only if the final score exceeds the minimum threshold of 0.30 - below this value the algorithm returns no match rather than suggesting a low-quality result. The result is stored in the `user_soulmate_matches` table with the full metric breakdown, so the UI can show the user exactly why they were matched - for example, Ratings 99%, Watched 18%, Actors 16%. Recompute is limited to once per 24 hours.

4.3.2 Hybrid Recommendation Pipeline

The recommendation system is a three-stage pipeline: collaborative filtering generates candidates, TMDB Discover adds diversity, and Gemini re-ranks the combined pool into a final list.

4.3.2.1 Stage 1 - ALS Collaborative Filtering (Python CF Service)

The CF service is a separate Python microservice on FastAPI. It trains an ALS model from the `implicit` library [5] on a user-item interaction matrix built directly from the PostgreSQL database.

Interaction matrix construction. Each user-movie interaction is converted to a preference score that reflects the strength of the signal. Adding a movie to favorites scores highest (5.0), followed by high ratings (4.0 for 8+, 3.0 for 6-7, 2.0 for 4-5), then watchlist additions (1.5), and finally movies that were only marked as watched without a rating (1.0). The resulting matrix is stored in scipy Compressed Sparse Row (CSR) format - since most user-item pairs have no interaction at all, a sparse representation keeps memory usage low.

BM25 weighting. Before training, Best Matching 25 (BM25) weighting [6] is applied to the matrix so that users with very large watch histories do not dominate the model's learned representations, while less active users still contribute meaningfully to the training signal.

Model training. The ALS model learns latent factor representations for both users and items by alternately solving for user factors and item factors while keeping the other fixed. [7] The model parameters (factors=24, iterations=30, regularization=1.0, alpha=20.0) were selected through a grid search optimizing Hit Rate@10 and Mean Reciprocal Rank (MRR). The trained model and index mappings are saved to disk so that inference does not require retraining on every request.

Inference. At request time, the model computes a score for every item the user has not yet interacted with and returns the top N candidates as TMDB IDs. Already watched and disliked titles are filtered out before returning results.

4.3.2.2 Stage 2 - TMDB Discover Candidates

ALS only recommends items that were present in the training matrix. To add variety and cover cases where a user's taste is unusual or their history is small, a second candidate source is built from the TMDB Discover API [8]. The system builds a user profile from watch history - top 3 genre IDs, top voted actor, and allowed content buckets (movie, tv, anime, dorama, animation) - and fires parallel Discover requests for each content type the user has shown interest in. Results are deduplicated and merged into a single candidate pool, each item tagged with its source (ALS or Discover).

4.3.2.3 Stage 3 - Gemini Re-ranking

The tagged candidate pool is sent to Google Gemini 2.5 Flash [9] along with a 10-film sample of the user's watch history, picked to span the full rating range - one film per rating level from 10 down to 1, so the model sees both what the user loves and what they dislike (with favorite and disliked flags included).

4.3.2.4 Cold Start Strategy

New users do not yet have enough watch history for ALS to produce meaningful results. For these users, a dedicated cold start service runs instead. During onboarding, users select favorite actors and rate a curated set of movies. The cold start service analyzes these signals: genre weights are computed per movie with higher-rated movies contributing more weight (rating 8+ contributes weight 3, rating 6-7 contributes weight 2), content buckets are inferred from the types of content the user responded positively to, and three parallel TMDB Discover queries run - by actor, by genre, and by popularity. Results from all three sources are merged using an interleaving strategy to keep the content type distribution balanced across the final batch of 25 recommendations. Once a user accumulates enough watch history, the system automatically switches to the full ALS + Gemini pipeline on the next recommendation request.

4.4 Deployment and CI/CD

4.4.1 Deployment Process

4.4.1.1 Infrastructure Decisions

Choosing where to host MovieCrush went through two iterations before landing on the final setup.

AWS (dropped). The original plan was to deploy on AWS. However, the AWS account had previously been blocked due to an outstanding balance. The decision was made not to restore it and look for an alternative - the risk of getting blocked again in the middle of active development was too high.

Render + Neon (intermediate). The next setup was Render for the backend and Neon as a separate managed PostgreSQL provider. This worked but had two real problems. First, the database was outside Render's network, which meant every query from the backend had to travel over the public internet to reach it. This added latency to every single request. Second, having the database reachable from the public internet is a security concern in itself - the database was not inside a private network with the backend, so it was exposed in a way that a production system should avoid.

Render only (final). Moving both the backend and the database to Render solved both problems. Render Postgres is managed PostgreSQL hosted in the same infrastructure as the backend services. Queries between the backend and the database stay inside Render's private network - no public internet hop, lower latency, and the database is not exposed outside the private network. Render also provides automatic daily backups and handles PostgreSQL maintenance automatically.

4.4.1.2 Docker Multi-stage Build

Both the backend and the CF service are packaged as Docker images using multi-stage builds. The multi-stage approach keeps the final image small: a build stage compiles TypeScript or installs all Python dependencies, and a separate runtime stage copies only what is needed to run.

For the backend, the build stage installs all dependencies including dev tools and compiles TypeScript to `dist/`. The runtime stage starts fresh, installs only production dependencies, and copies just the compiled output. This means the final image does not carry TypeScript, ts-node, or any other build-time tooling.

Both services are deployed on Render as separate web services. They can be updated and redeployed independently - for example, retraining the ALS model and redeploying the CF service does not require touching the backend.

4.4.1.3 Mobile Distribution

The mobile app is built using EAS Build (Expo Application Services). Building in the cloud removes the need for a local Mac to produce the iOS-compatible bundle. For Android testing and distribution, EAS produces an APK that can be shared directly - no Play Store account is required for this stage.

4.4.2 CI/CD Pipeline

The CI pipeline runs on GitHub Actions on every push to `main` and on feature, fix, and chore branches, as well as on pull requests to `main`. It has four jobs that run in a defined order.

Job 1 - Backend: TypeScript + tests. Installs Node.js 20, runs `npm ci`, then performs three checks in sequence: TypeScript type check, the Jest unit test suite, and a production build. All three must pass before the Docker build job is allowed to start.

Job 2 - Mobile: TypeScript check. Installs dependencies for the React Native project and runs a TypeScript type check. This catches type errors in the mobile codebase without needing to run a full Expo build on every push.

Job 3 - CF Service: install + import check. Sets up Python 3.11, installs requirements, and runs a quick import check to confirm that the core modules load without errors. This catches broken dependencies or import-time failures before a broken CF service image gets built.

Job 4 - Docker: build images. Runs only after the backend and CF service checks pass. Builds both Docker images to confirm that the full containerized build works end to end.

Deployment to Render is triggered automatically on every commit. Render detects the new commit and redeploys the service without any manual steps - so a passing CI run and a successful deploy happen together as part of the same push.

5 Validation

5.1 Requirements Validation

The tables below list all key requirements, how each one was tested, and the result.

5.1.1 Functional Requirements

Requirement	How tested	Result
Personalized recommendations based on user behavior	Offline evaluation - ROC AUC 0.8771	Pass
Cold start for new users	Manual testing with a new account	Pass
Onboarding flow for cold start signal collection	Manual testing with a new account	Pass
Ratings - overall (1-10) and detailed (5 criteria)	Manual testing + DB check	Pass
Search for movies, series, actors, and users	Manual live search testing	Pass
Lists - Watched, Watchlist, Favorites, custom	Manual testing + DB check	Pass
Social features - follows, friends' ratings	Manual testing between two accounts	Pass
Soulmate - finding a user with similar taste	Unit tests (soulmate_scoring, soulmate_similarity) + manual testing	Pass
Wrapped - personal yearly statistics	Unit tests (wrapped) + manual testing	Pass
Comments with anonymous and spoiler options	Manual testing of all comment modes	Pass
Episode tracking for series	Unit tests (episode_helpers) + manual testing	Pass
Mood tagging after watching	Manual testing + DB check	Pass
Actor page with filmography	Manual testing	Pass
Movie page with full content	Manual testing	Pass

Requirement	How tested	Result
Registration and login with validation	Unit tests (auth_validators) + manual testing	Pass
Email confirmation and password recovery	Manual flow testing	Pass
JWT authentication with refresh token rotation	Unit tests + middleware type check	Pass
Challenges and premiere notifications	-	Planned
Public iOS distribution	-	Requires Apple Developer account

Table 21: Functional requirements - testing method and result

5.1.2 Non-Functional Requirements

Requirement	How tested	Result
Rate limiting - 200 requests per minute per IP	Response headers check	Pass
HTTPS for all public endpoints	Verified via Render dashboard	Pass
Two-level cache - LRU in-memory + PostgreSQL	Code review + manual response time check	Pass
CI/CD pipeline with automated checks	GitHub Actions run on every push	Pass
Password hashing with bcrypt	Code review - no plain text passwords stored	Pass
Health check endpoint	GET /health returns status ok	Pass

Table 22: Non-functional requirements - testing method and result

5.2 Unit Testing

The backend uses Jest together with ts-jest for TypeScript support. All tests live in the `__tests__` folder and run automatically in the CI pipeline on every push. The focus is on critical business logic - the parts of the system where a bug directly affects data correctness or security.

5.2.1 What Is Covered

5.2.1.1 soulmate_scoring_test.ts and soulmate_similarity_test.ts

Together these cover the math behind the Soulmate algorithm: cosine similarity, Jaccard similarity, cosineSimilarityFromCounts, weightedSum, buildRatingVectors, and isRecomputeThrottled.

Edge cases get special attention: empty vectors, zero norms (to avoid division by zero), vectors of different lengths, and cooldown logic across time intervals. For example:

- two identical vectors produce similarity = 1.0

- two orthogonal vectors produce similarity = 0
- weightedSum with all-zero metrics returns 0
- the cooldown returns true if less than 24 hours have passed, false otherwise or when lastComputedAt is null

5.2.1.2 als_service_test.ts

Covers helper functions in the recommendation service: mediaTypeFromTmdb and releaseYearFromTmdb (parsing TMDB response fields), sliceToLimit, filterValidItems, and cacheRowToAlsItem. filterValidItems is checked to confirm it skips items without a title and preserves the order coming from ALS.

5.2.1.3 cold_start_service_test.ts

The largest suite, covering the onboarding-based cold start logic: getContentBucket, computeGenreWeights, getTopGenreIds, getAllowedBucketsFiltered, passesLanguageGenreFilter, and buildBatch. It also runs full onboarding taste scenarios end to end to confirm the batch composition stays balanced.

5.2.1.4 wrapped_test.ts

Covers yearly statistics helpers: determineCinemaVibe (the most common watch time of day), calculateTimeStats (total watch time), and calculateFanPercentile.

5.2.1.5 auth_validators_test.ts

34 tests covering input validation for registration and login: email format (with and without subdomains, dots, special characters), username rules (length, allowed characters), and password requirements (length, presence of digits and letters). The validators run before any data reaches the backend.

5.2.1.6 tmdb_helpers_test.ts

Covers parseTmdbId, which solves a real bug found during development: PostgreSQL stores BIGINT but the pg driver returns it as a string to avoid losing precision on 64-bit values. The helper accepts both numeric and string input and returns the correct number or null for invalid input. Tested cases include regular numbers, numeric strings, large 64-bit values, zero, negatives, NaN, Infinity, null, undefined, and empty string.

5.2.1.7 episode_helpers_test.ts

Covers buildAllEpisodesList, which flattens a series' seasons into a single (season, episode) list. Tested cases include skipping season 0 (Specials), empty input, seasons with zero episodes, and counting episodes across many seasons.

5.2.2 Results

All tests pass. The CI pipeline runs them automatically via `npm test` on every push to main and feature branches - if any test fails, the build does not pass and deployment does not happen.

One gap: there are no integration tests for end-to-end flows like the full recommendation pipeline, since those need live database and CF service connections.

5.3 Manual Testing

5.3.1 Testing Setup

The app was tested manually throughout the development process. Thirteen external testers used the app through Expo Go on my personal Android device. An Android APK was also built via EAS Build for direct distribution without the Play Store, though the main testing session was conducted through Expo Go. iOS is compatible but requires an Apple Developer account for public distribution, so it was demonstrated on a personal device only.

5.3.2 User Feedback

All thirteen testers praised the interface and said the app felt comfortable and polished to use. Feedback was given in Ukrainian and is translated here. Three representative responses are quoted below; the rest is synthesized at the end of this section.

Tester 1:

> I really like that this app has a dark theme, a light one would definitely not work here. As soon as you open the app you start thinking about what you want to watch, you feel comfortable, and you immediately imagine the atmosphere of a cinema or a cozy evening at home under a warm blanket with a series and a cup of tea.

They also noted that the social layer was a highlight - especially seeing friends' ratings directly on a movie or episode page, and that Wrapped analytics felt exciting and natural for a generation that loves tracking everything.

Two improvement requests came up: the "More Like This" section felt like it could show better recommendations, and they would like to see streaming availability directly on the movie page (for example, whether a film is on Netflix or Kyivstar TV). The "More Like This" issue was already addressed by switching from the TMDB `/similar` endpoint to `/recommendations` with a fallback. Streaming availability is planned for future development.

Tester 2:

> The interface is very intuitive - you really understand where to find everything, as if the app was built from a template, the developer clearly understood what goes where. The analytics part amazed me. On Spotify I wait to see my Wrapped like it's a holiday, I love sharing it in my Instagram stories so everyone knows how cool I am. I would really love to share film analytics in stories too. The recommendations are great - I saw films in my recommendations that I had already watched and loved, I just hadn't added them to my watched list in the app yet. And I watched one film from the list and gave it 9 out of 10. So I am very happy.

Notably, the tester noticed the recommendations matched their taste without any explanation of how the system works - a sign the ALS + Gemini pipeline behaved as intended.

Tester 3:

> I really like the colors - they don't hurt your eyes and they match each other. I was pleasantly surprised by the onboarding because I use Letterboxd and I have never seen anything like it there. I am really glad that I can immediately enter what I like and get recommendations based on that. I love that I can go to an actor's page and see where they appeared - sometimes I get really into an actor and now instead of googling I just go to their page from a film I watched and browse their filmography. And the fact that there is a split between Acting and Crew tabs means I can immediately open just the Acting tab. I also like that the rating is shown right on the poster so I don't have to open the film to see it. And I really appreciated the filter in recommendations especially the tab "russia is a terrorist state" - that is a nice reminder. In the future I would love a permanent filter where I can set that I never want to see content from a specific country in my recommendations at all.

5.3.3 Feedback Across All Testers

Grouping the feedback from all thirteen testers shows clear patterns. The most praised features were the analytics (Wrapped), the Soulmate matching, friends' ratings, the actor page with full filmography, and the ability to rate a whole series without rating each episode. One tester with an analytics background remarked that features like Soulmate and Wrapped would noticeably improve retention.

The most requested improvements clustered around a few themes: showing where a movie can be streamed (with a direct link), automatic watched-tracking when a film is viewed on an external service, push notifications for new episodes and cinema releases, separating films and series within lists, and importing an existing watchlist on signup

instead of adding movies by hand. Smaller suggestions included animated transitions in Wrapped, a Spotify link for soundtracks, a private-account option, and larger rating stars in onboarding for accessibility. Notably, most of the top requests - streaming availability, push notifications, and share templates - already match the planned roadmap (see Limitations and Future Improvements), confirming the direction rather than redirecting it.

5.3.4 Bug Found and Fixed During Testing

During the session with Tester 3, a performance issue was discovered. Opening a profile list with 30 films triggered 30 parallel requests to TMDB through the backend proxy. Switching between lists quickly exceeded the rate limit and returned 429 Too Many Requests errors.

The fix was a new **`POST /tmdb/media/batch`** endpoint - a single request that reads all metadata from **`tmdb_media_cache`** in one SQL query using **`WHERE tmdb_id = ANY(...)`** and only fetches missing entries from TMDB. Instead of 30 requests, the list now loads with 1. The cache is shared across users, so popular films are fetched from TMDB only once for the entire service.

5.4 ML Model Evaluation

5.4.1 Problem Statement

The goal was to build a collaborative filtering recommendation system that, based on user behavior (watched films, ratings, favorites), produces personalized recommendations - films the user is likely to want to watch next.

The initial plan was to use LightFM - a more flexible library with support for content-based and collaborative hybrid filtering. However, LightFM has no official Windows support, which made development impossible in the available environment. The **`implicit`** library was chosen as a replacement because it implements the same ALS algorithm from the original Hu, Koren, Volinsky (2008) paper [7], has full Windows support, and supports BM25 weighting and confidence-aware training.

5.4.2 Data Flow

The full pipeline from raw data to recommendations:

- PostgreSQL (user_movie_actions, user_detailed_ratings)
- build_matrix() - sparse CSR matrix (users on items)
- BM25 weighting (reduces influence of popular items and superfan users)
- ALS training - user and item embeddings
- .recommend(user_id) - top-N items
- HTTP response - Node.js backend - mobile app

5.4.3 Synthetic Training Data

Since there were not enough real users at the time of evaluation, the training dataset was generated synthetically.

The initial generation script (**`generate_ratings_for_all_users.py`**) split 208 users into 8 fixed genre profiles and selected films randomly from large pools. The problem was

that two users with the same profile would watch completely different random films - the overlap between them was tiny, and the model could not learn the pattern "users who watched X also watch Y" because such pairs did not exist in the data.

A new script **`generate_realistic_data.py`** was built with a fundamentally different approach: films were grouped into 17 semantic clusters (marvel_mcu, john_wick_franchise, nolan_films, tarantino, ghibli, pixar, lotr_hobbit and others) where films inside a cluster are genuinely related - same franchise, director, or style. Each user has 2-4 primary clusters with high ratings, 1-3 affiliated clusters through a cross-cluster affinity graph, 1-2 disliked clusters with low ratings, and 2-4 random films as noise. Real user accounts used for manual testing were protected and excluded from synthetic data generation.

5.4.4 Iterative Optimization Process

Development went through 5 iterations, each producing a measurable improvement in metrics. The approach was: baseline, problem, hypothesis, experiment, measurement.

The per-iteration and final result charts are collected in Appendix AE.

5.4.4.1 Iteration 1 — Baseline

Configuration: 208 synthetic users, 315 films, 10K interactions. ALS with factors=32, iterations=20, regularization=0.1. No BM25 weighting. Ratings used directly as confidence weights.

Four problems were found: incorrect Leave-One-Out (LOO) methodology where a separate model was trained per user (208 models), missing BM25 weighting, wrong confidence interpretation where ratings were used directly as weights instead of being scaled through the confidence parameter alpha as the original ALS paper requires, and dislikes added as weak positive signals with weight 0.1 which the model interpreted as mild interest rather than rejection.

5.4.4.2 Iteration 2 - Algorithmic Fixes

Changes: single model trained on all users (correct LOO), BM25 weighting added, alpha parameter introduced, dislikes fully removed from the training matrix and used only as a post-filter at inference time, structured preference scoring introduced.

5.4.4.3 Iteration 3 - Realistic Data Generation

After switching to the cluster-based generation script:

Key finding: data quality mattered more than the algorithmic fixes. Hit Rate@10 jumped from 17% to 46% - a larger gain than all previous improvements together.

5.4.4.4 Iteration 4 - Full Evaluation (Classification + ROC AUC)

A full set of metrics was added: Precision@K, Recall@K, F1@K, NDCG@K, and ROC AUC.

ROC AUC methodology for collaborative filtering. There is no standard classification scenario in CF, so a negative sampling approach was used: for each test user, 1 known relevant item is taken and 50 random items the user has not watched are sampled as negatives. The model scores all 51 items and the ROC curve is built on (y_true, y_scores).

5.4.4.5 Iteration 5 - Hyperparameter Tuning (Grid Search)

A systematic grid search over 60 combinations was run to find the optimal configuration instead of manual tuning.

Search space: factors {12, 16, 20, 24, 32}, alpha {20, 40, 60, 80}, regularization {0.1, 0.5, 1.0}.

Each combination was trained with fixed random_state=42 on the same train/test split and evaluated on the same test pairs.

Key observations: all top-5 configurations by HR@10 and MRR use alpha=20 - lower than the initial value of 40. For a sparse matrix, lower confidence weighting works better because it does not overfit on popular items. factors=24 showed the best balance. regularization=1.0 was optimal - on a small matrix, stronger smoothing is needed.

Selected configuration: **factors=24, alpha=20, regularization=1.0**.

5.4.5 Final Results

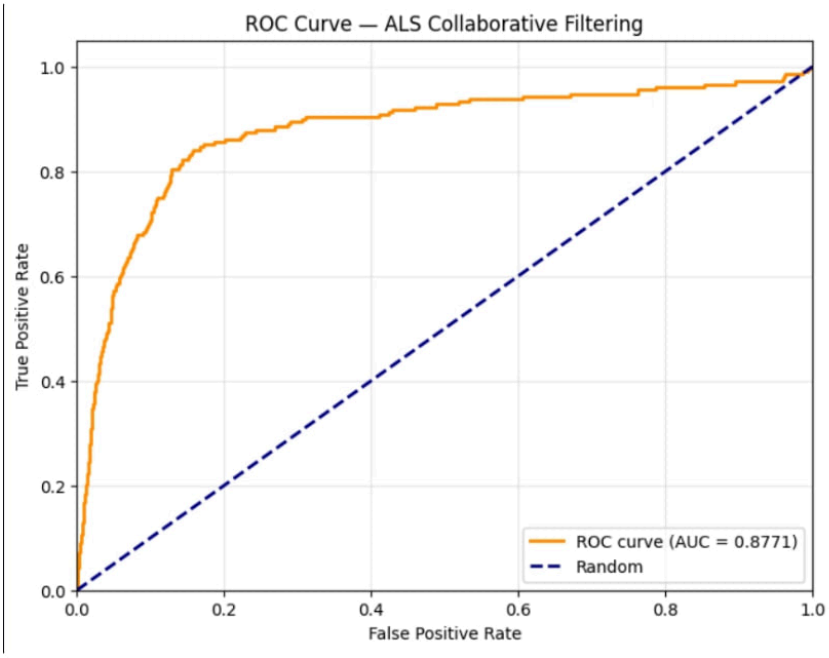


Figure 8: ROC Curve - ALS Collaborative Filtering. AUC = 0.8771, well above the random baseline of 0.5

ROC AUC = 0.8771 means the model correctly ranks a relevant film above a random irrelevant one in 87.71% of cases.

K	Hit Rate@K	Precision@K	NDCG@K
1	10.58%	28.85%	0.1058
5	32.69%	58.37%	0.2174
10	51.92%	74.13%	0.2802
20	63.94%	82.09%	0.3115

Table 23: Final ALS model metrics after all iterations. MRR: 0.2639. Average rank when found: 12.11

5.5 Limitations and Future Improvements

5.5.1 Current Limitations

Synthetic data for ML evaluation. The ALS model was evaluated on a generated dataset that models realistic viewing patterns, but not on real users. Production metrics may differ - either better (more data means better embeddings) or worse (real taste patterns are more complex than synthetic ones).

Small interaction matrix. The current training matrix is 208 by265 - very small for a production system. ALS works better on larger matrices where there are more cross-user signals. As the user base grows, the metrics should improve.

Manual model retraining. The model is currently retrained manually. In production, automated cron jobs are needed to keep the model up to date with new user data.

No TTL on tmdb_media_cache. Movie metadata is cached without an expiry time, so if TMDB updates a poster or other field, the cache will not pick up the change.

iOS distribution. Public distribution on iOS requires an Apple Developer account. At the moment the iOS version is only shown on a personal device through Expo.

Email verification in the test environment. The free Resend plan without a custom domain can only send emails to the account owner's address. For testing purposes, email verification was made optional - the mechanism is kept in the codebase and ready for production use.

Limited external testing. Thirteen testers had access to the app through Expo Go on my device. Testing across a wider range of devices remains a task for the next stage.

5.5.2 Planned Improvements

- Automated ALS model retraining on a schedule (cron job)
- Challenges and gamification - achievements, badges, and viewing goals
- Push notifications for premiere dates of tracked series
- Language switch in the app (Ukrainian/English)
- Instagram Stories share templates for Wrapped and Soulmate results
- Public iOS release after getting an Apple Developer account
- Custom domain for Resend with full email verification for all users
- Periodic tmdb_media_cache refresh with a weekly TTL
- Affiliate partnerships with streaming platforms (JustWatch model)
- Monetization through MovieCrush Pro subscription

6 Conclusion

6.1 Project Summary

MovieCrush is a cross-platform mobile application that unifies film and TV tracking, personalized recommendations, social discovery, and personal analytics in a single product. The system was built over three months as a solo project, resulting in a React Native mobile client, a Node.js and Express backend with 11 domain modules, a PostgreSQL database with 18 tables, and a separate Python microservice running an ALS collaborative filtering model.

The recommendation engine is a three-stage hybrid pipeline: ALS collaborative filtering generates candidates, TMDB Discover adds variety, and Google Gemini 2.5 Flash re-ranks the combined pool and assigns each result a category - strong match, diversity, or hidden gem. For new users, a dedicated cold start service uses onboarding signals to serve initial recommendations until enough watch history accumulates for ALS. The Soulmate feature finds the user with the most similar taste using a weighted combination of five metrics: rating cosine similarity, watched overlap, actor overlap, mood similarity, and disliked overlap. The Wrapped feature generates personalized yearly statistics across top genres, directors, actors, moods, and total watch time.

The ALS model was tuned across five optimization iterations on a synthetically generated dataset, reaching a final ROC AUC of 0.877 - substantially above the random baseline of 0.5. Thirteen external users tested the app and provided feedback that directly influenced product decisions and surfaced a performance bug that was found and fixed during testing.

6.2 Comparison with Initial Objectives

The core objectives set at the start of the project were met. The full application was built and deployed end to end. The backend, the Python CF service, and the PostgreSQL database run on Render, while the React Native client is distributed to other people as an Android APK. The hybrid recommendation pipeline (ALS, TMDB Discover, Gemini), the Soulmate algorithm, and Wrapped analytics were all implemented and tested. Wrapped was especially well received - several testers compared it directly to Spotify Wrapped.

Several planned features were not implemented within the timeline, mainly Challenges and gamification and Instagram Stories share templates. They were consciously deprioritized after survey results showed that Wrapped (rated 4.24 out of 5), AI recommendations, Soulmate, and social features mattered most to the target audience. The app is fully cross-platform, but public iOS distribution was left out because it requires a paid Apple Developer account - the iOS version runs on a personal device through Expo Go.

6.3 Encountered Difficulties

The hardest part of this project was the recommendation system - not technically, but in terms of decision-making. The overall approach to recommendations changed four times before settling on the final pipeline.

The first plan was to use an AI chat assistant similar to ChatGPT but limited to film topics. This was dropped early in favor of a single AI assistant button that generates personalized recommendations using DeepSeek. DeepSeek was then dropped after the committee pointed out that a purely LLM-based approach could ignore large prompts, lose context when given many films, or return irrelevant results. Nine models across the Claude, GPT, and Gemini families were tested with five prompts each, before the same concern was confirmed: a pure LLM approach was too unpredictable for personalization at scale.

The final solution was a hybrid pipeline where collaborative filtering does the heavy lifting and Gemini acts only as a re-ranker on a pre-filtered candidate pool. This architecture is more reliable, more explainable, and produces better results - but it took four iterations of rethinking to get there. Looking back, this iterative process of forming a hypothesis, testing it, and being willing to throw it away and start again was the most valuable engineering lesson of the whole project.

Other notable challenges included the AWS account being blocked due to a billing issue; the Neon + Render split database setup adding latency and exposing the database to the public internet, solved by moving everything to Render Postgres; and a director_similarity metric in the Soulmate algorithm that was computing against an empty table, silently wasting part of the weight budget until it was caught in a code review and removed, after which the remaining five weights were rebalanced to sum to 1.0.

6.4 Future Perspectives

The most immediate next steps are automated ALS model retraining on a cron schedule, a custom domain for Resend to enable email verification for all users, and a public release on the App Store and Google Play (which also requires moving the TMDB API to a paid plan to handle the request volume).

On the product side, the most requested features are Instagram Stories templates for Wrapped and Soulmate, a permanent country filter in recommendations, streaming availability on film pages, a PDF/CSV watch-list importer, and Challenges with achievements. A Ukrainian language switch, the Pro subscription, and custom profile photos are already stubbed in Settings as upcoming.

Longer term, as the real user base grows and the interaction matrix becomes larger and denser, the ALS model is expected to improve - the current evaluation used a synthetic dataset of 208 users, and real behavioral patterns at scale will provide a much richer training signal. At that point, online evaluation with A/B testing would replace the current offline metrics.

Glossary

***ALS* – Alternating Least Squares**: A collaborative filtering algorithm that learns hidden user and item factors by solving for one set while keeping the other fixed, then switching - the alternating part. It is a common choice for implicit feedback data such as views, clicks, or purchases. 9

***BM25* – Best Matching 25**: A weighting method from information retrieval. When applied to a user-item matrix, it reduces the influence of very frequent interactions so that the most active users or most popular items do not dominate the model. 40

***CF* – Collaborative Filtering**: A recommendation technique based on the idea that users who agreed in the past will likely agree in the future. If two users rated the same movies in a similar way, a movie one liked can be suggested to the other.

***cold start* – Cold Start Problem**: In recommender systems, the situation when there is not enough data about a new user or item to make good recommendations.

***cosine similarity* – Cosine Similarity**: A measure of similarity between two vectors based on the angle between them rather than their magnitude. Two users who rate films in the same relative pattern score high even if one rates consistently higher. 39

***CSR* – Compressed Sparse Row**: A memory-efficient format for storing sparse matrices that keeps only the non-zero entries - suited to a user-item matrix where most pairs have no interaction. 40

***Hit Rate@K* – Hit Rate at K**: The share of test cases where the relevant item appears in the model's top-K recommendations. Hit Rate@10 = 0.52 means the correct film was in the top 10 about half the time.

***Jaccard similarity* – Jaccard Index**: A similarity measure for two sets: the size of their intersection divided by the size of their union. Ranges from 0 (no overlap) to 1 (identical sets). 39

***LOO* – Leave-One-Out**: An evaluation method where one known relevant item is hidden per user, and the model is scored on how well it ranks that held-out item among others. 50

***MRR* – Mean Reciprocal Rank**: A metric that checks how high up the first correct recommendation appears in a ranked list. The higher the correct item, the better the score. 41

***NDCG* – Normalized Discounted Cumulative Gain**: A ranking metric that rewards placing relevant items near the top of the list and gives less credit to relevant items lower down.

***Precision@K* – Precision at K**: Of the top-K items the model recommends, the share that are actually relevant. Higher means fewer irrelevant suggestions near the top.

***ROC AUC* – Receiver Operating Characteristic - Area Under Curve**: A metric that measures how well a model ranks a relevant item above an irrelevant one. A value of 0.5 means random guessing, and 1.0 means a perfect ranking.

Bibliography

[1] AppMagic, "Film Tracking Apps - Market and Visibility Dashboard." [Online]. Available: https://appmagic.rocks/dashboards/673fc5d7c17b3

[2] GeeksforGeeks, "What is Circuit Breaker Pattern in Microservices?." [Online]. Available: https://www.geeksforgeeks.org/system-design/what-is-circuit-breaker-pattern-in-microservices/

[3] R. Sun, X. Li, A. Akella, and J. A. Konstan, "Large Language Models as Conversational Movie Recommenders: A User Study," *arXiv preprint arXiv:2404.19093*, 2024, doi: 10.48550/arXiv.2404.19093.

[4] G. Kusano, K. Akimoto, and K. Takeoka, "Revisiting Prompt Engineering: A Comprehensive Evaluation for LLM-based Personalized Recommendation," in *Proceedings of the 19th ACM Conference on Recommender Systems (RecSys '25)*, ACM, 2025, pp. 832–841. doi: 10.48550/arXiv.2507.13525.

[5] B. Frederickson, "implicit: Fast Python Collaborative Filtering for Implicit Feedback Datasets." [Online]. Available: https://github.com/benfred/implicit

[6] S. Robertson and H. Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009, doi: 10.1561/1500000019.

[7] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets," in *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM)*, IEEE, 2008, pp. 263–272. doi: 10.1109/ICDM.2008.22.

[8] The Movie Database, "TMDB API Documentation." [Online]. Available: https://developer.themoviedb.org/docs

[9] Google, "Gemini API Documentation." [Online]. Available: https://ai.google.dev/gemini-api/docs

A | Appendix

AA Survey Materials

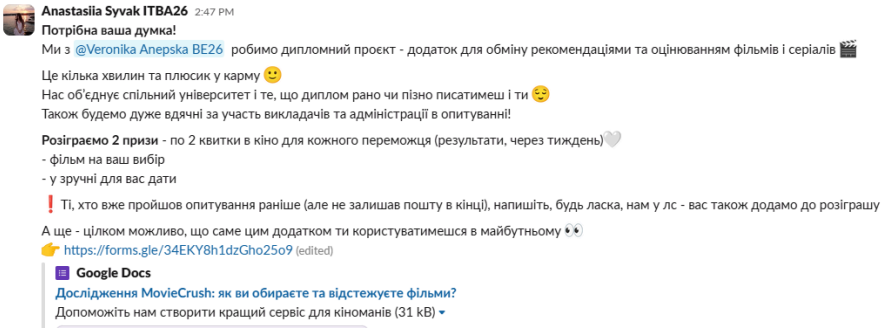


Figure 9: Survey announcement in KSE Slack

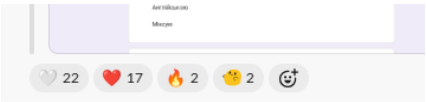


Figure 10: Community reactions - 43 emoji responses to the announcement

AAA Raffle Process

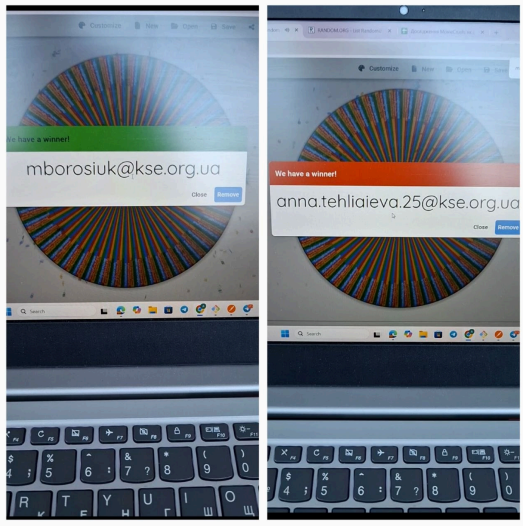


Figure 11: Random wheel spinner used to select raffle winners

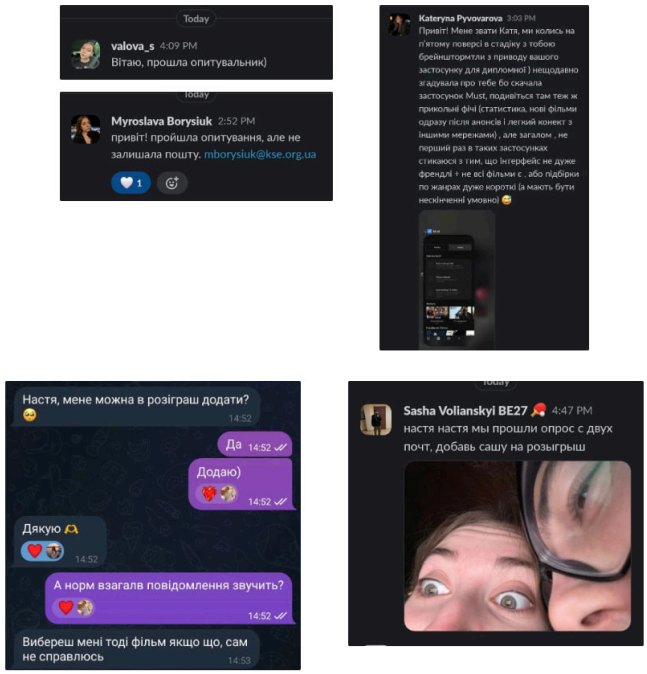


Figure 12: Messages from participants requesting to join the raffle

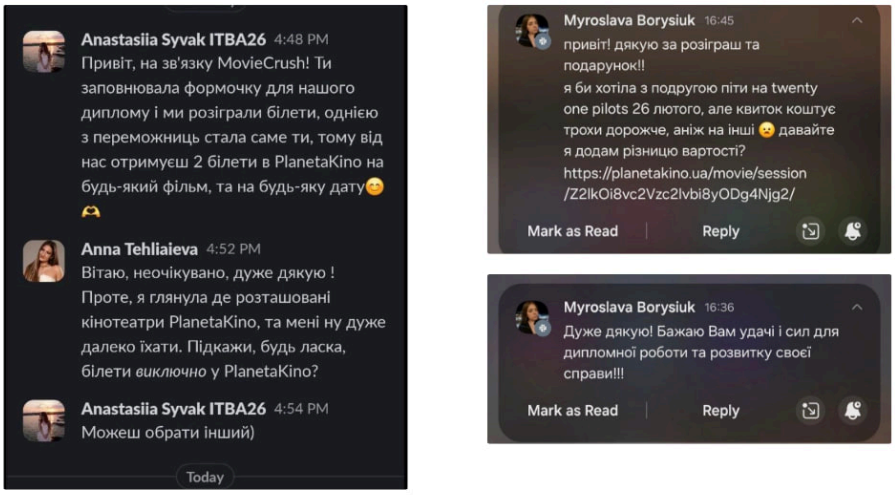


Figure 13: Winner confirmation messages

AB Additional Survey Charts

ABA Demographics

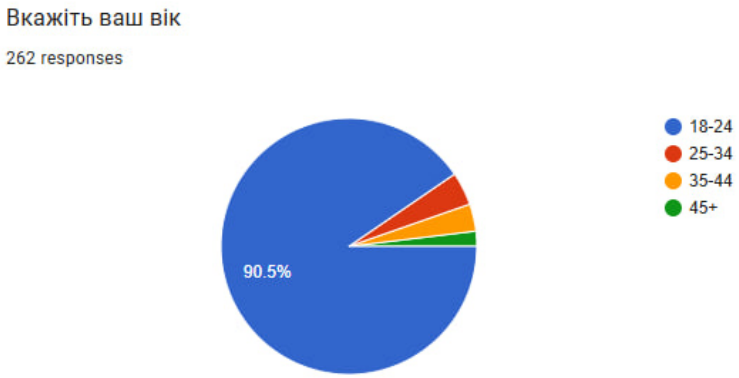


Figure 14: Age distribution of survey respondents

Figure 15: Language in which respondents consume film content

ABB Viewing Habits

Figure 16: Average number of films or series watched per week

Як саме ви зараз запам'ятовуєте/відстежуєте фільми, які хочете подивитися або вже подивилися?		
Не відстежую	59	24.9%
Нотатки в телефоні	118	49.8%
Використання наших конкурентів	47	19.8%
Збережені в тік ток/інстаграм	18	7.6%
Списки в стрімінгах (Netflix)	39	16.5%

Figure 17: How respondents currently track films

ABC Recommendation & Social

Figure 18: Where respondents look for recommendations

Figure 19: Most important social features

ABD Feature Validation & Monetization

Figure 20: Features respondents would pay for

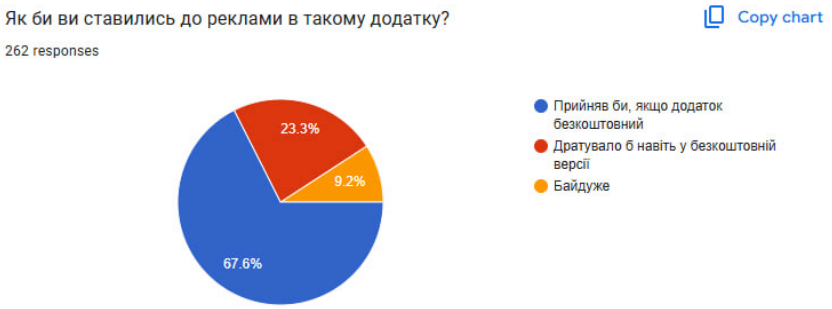


Figure 21: Attitude toward ads

Figure 22: Maximum monthly price for premium

ABE Wrapped Statistics Feature Rating

The Wrapped feature received the highest average rating of all tested features - **4.24 out of 5** - making it the most anticipated feature among respondents. 53.8% gave it the maximum score of 5.

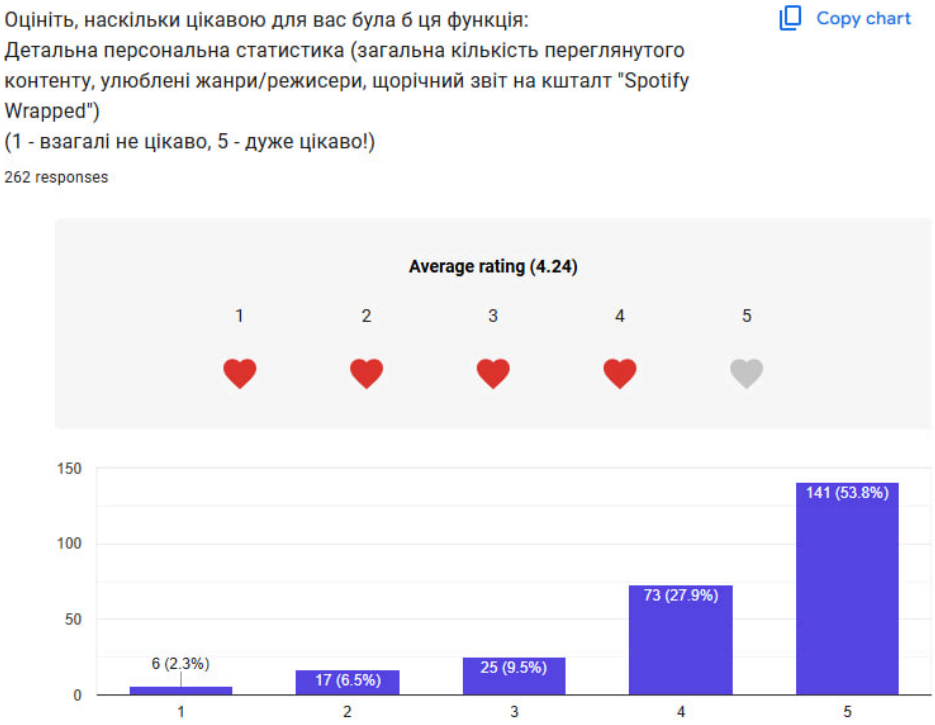


Figure 23: Rating distribution for the Wrapped statistics feature — average 4.24/5

ABF Monthly Spending on Subscriptions

Figure 24: Monthly spending on content subscriptions among respondents

ABG Active Streaming Subscriptions

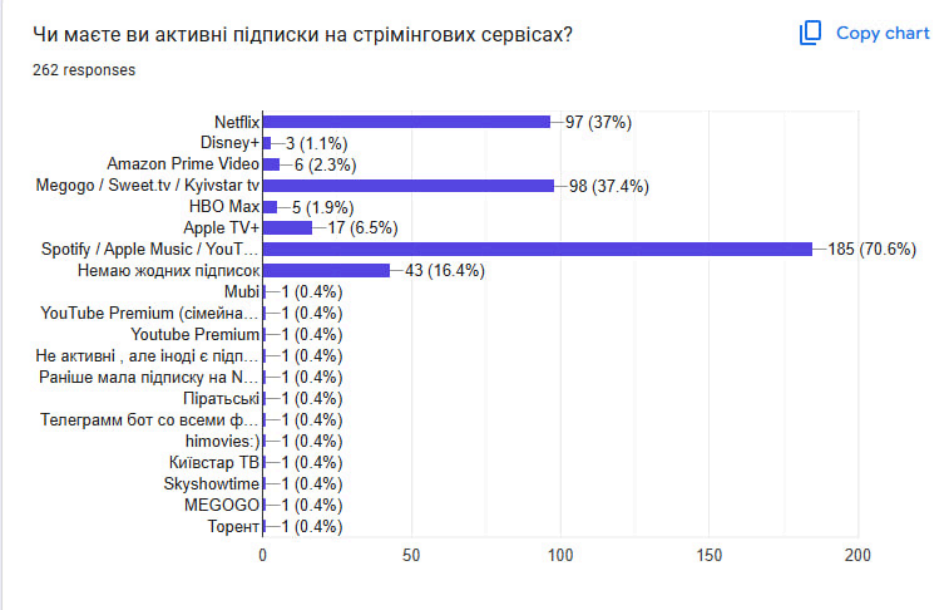


Figure 25: Active streaming subscriptions among respondents

Підписки на стрімінгових платформах	Кількість людей	Відсоток людей
Немає жодних підписок	41	17.3%
Є підписки	196	82.7%

Figure 26: Streaming subscription summary - 82.7% have at least one active subscription

ABH Respondent Profession & Field of Activity

Note: this profession breakdown is based on 237 responses (an earlier snapshot); the final survey count was 262.

Segmenting professions was challenging because 89.9% of respondents were aged 18-24, meaning most were students. Some listed only “student”, others listed their specialization, and those already working listed only their job title. To handle this, three separate breakdowns were created:

1. **General table** - all segments combined
2. **Profession table** - “Student” as a category plus job titles of those already working
3. **Field of activity table** - students filtered by their area of study, regardless of employment status

Of the respondents: 69 are not working, 51 did not answer, and approximately 117 indicated a profession (though some may have listed their future profession rather than a current job).

Ваша професія/сфера діяльності	
Пусті значення	47
Специфічна відповідь	4
Студент	46
IT	34
Аналітика	17
Економіка	18
Інші професії	14
Менеджмент	7
Аудит	8
Студент, Психологія	3
Право	4
Психологія	7
Викладач	8
Студент, IT	11
Студент, Економіка	6
Студент, Право	3

Ваша професія	
Пусті значення	47
Специфічна відповідь	4
Студент	69
IT	34
Аналітика	17
Економіка	18
Інші професії	14
Менеджмент	7
Аудит	8
Право	4
Психологія	7
Викладач	8

Сфера діяльності	
Пусті значення	47
Специфічна відповідь	4
Студент	46
IT	45
Аналітика	17
Економіка	24
Інші професії	14
Менеджмент	7
Аудит	8
Право	7
Психологія	10
Викладач	8

Figure 27: Three-way breakdown of respondent profession and field of activity

AC Screen Gallery

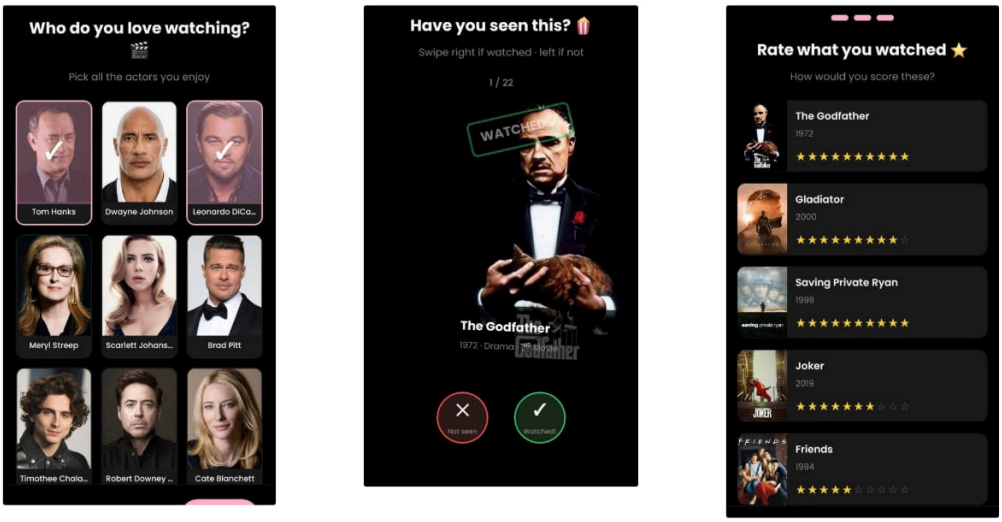


Figure 28: Onboarding - actor and movie selection for cold start

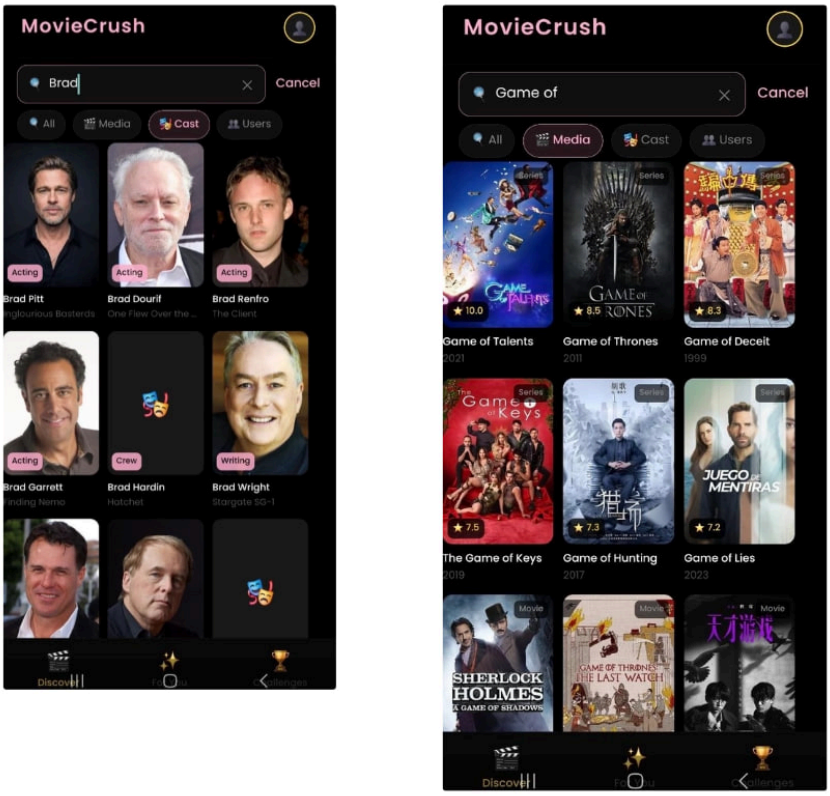


Figure 29: Search - live results across media, cast, and users

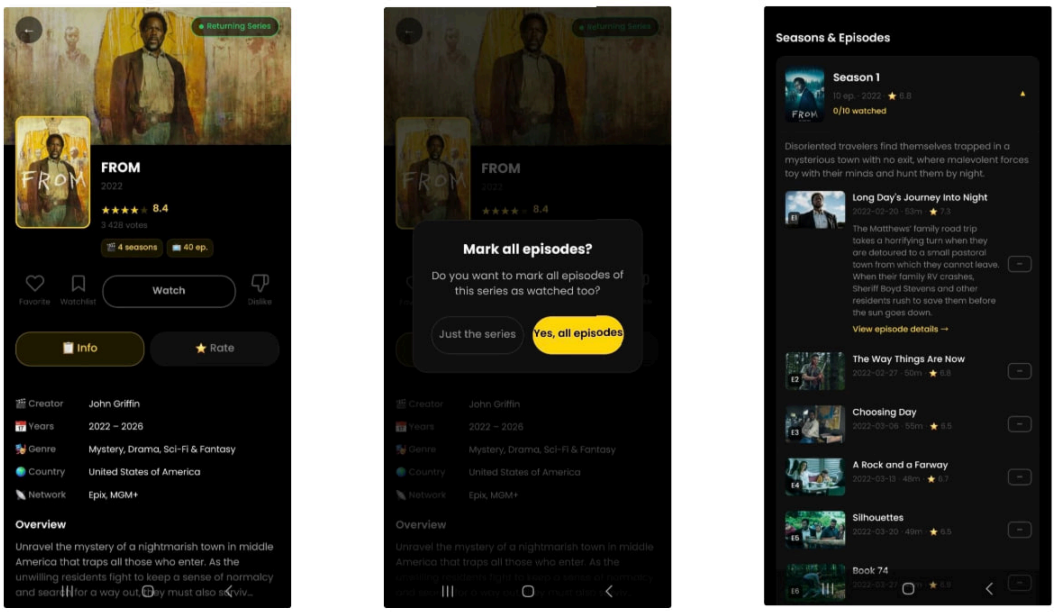


Figure 30: Series - seasons and per-episode tracking

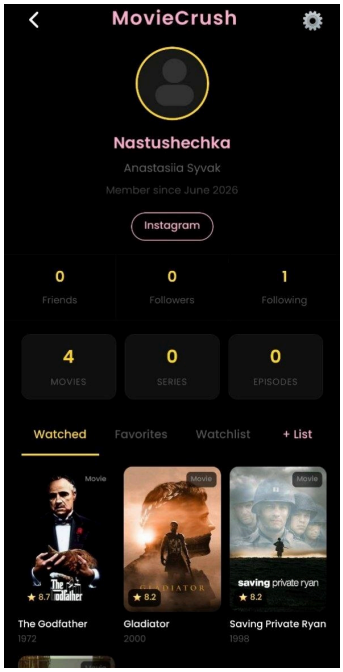


Figure 31: Profile - stats and lists

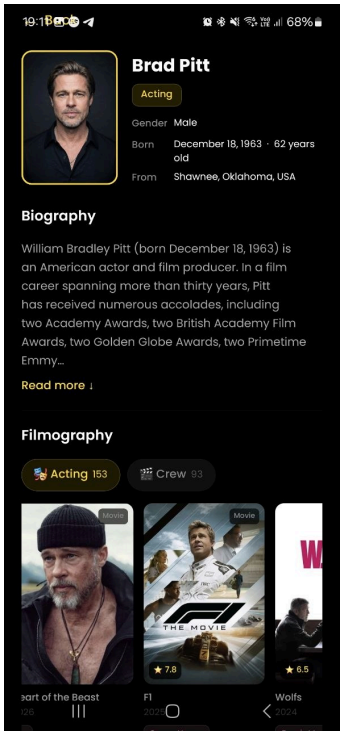


Figure 32: Actor - bio and filmography

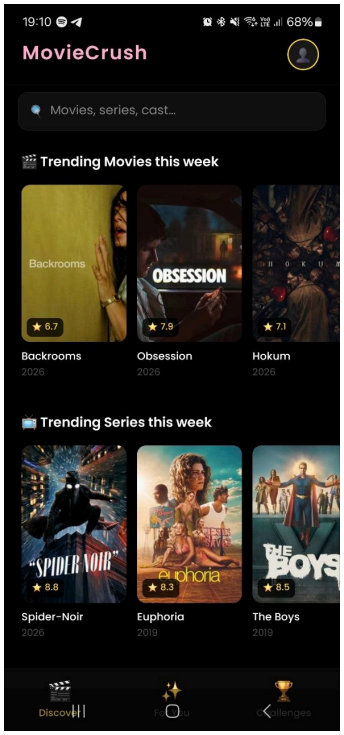


Figure 33: Home - trending and search

ACA Competitor Visibility (AppMagic)

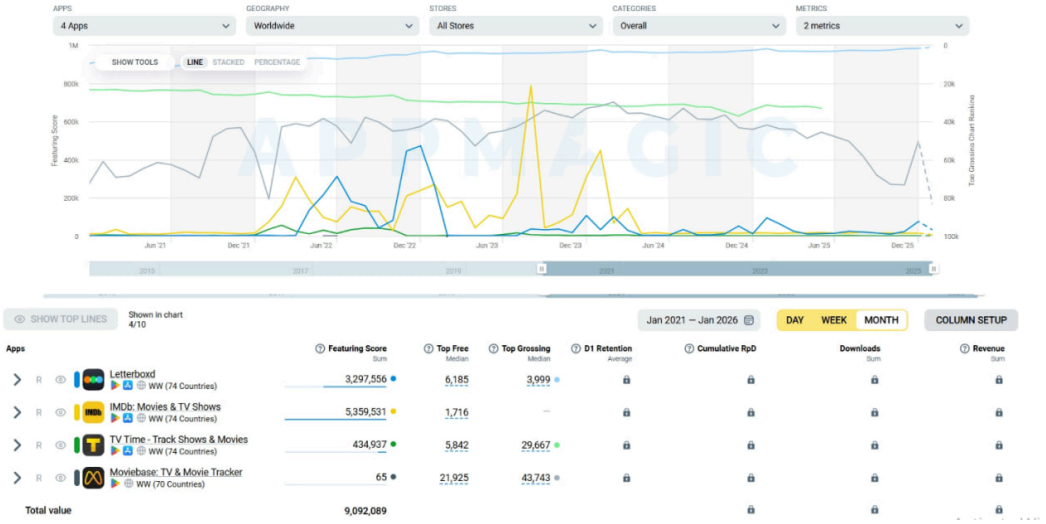


Figure 34: Featuring Score and chart rankings across four competitors, Jan 2021 - Jan 2026 (AppMagic)

AD Discovery Artifacts

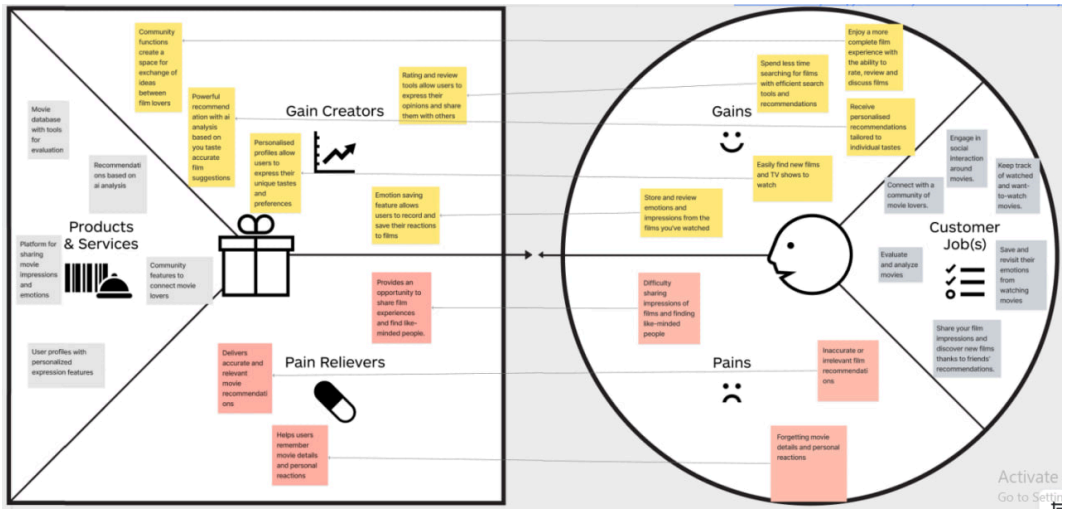


Figure 35: Value Proposition Canvas developed during the discovery phase

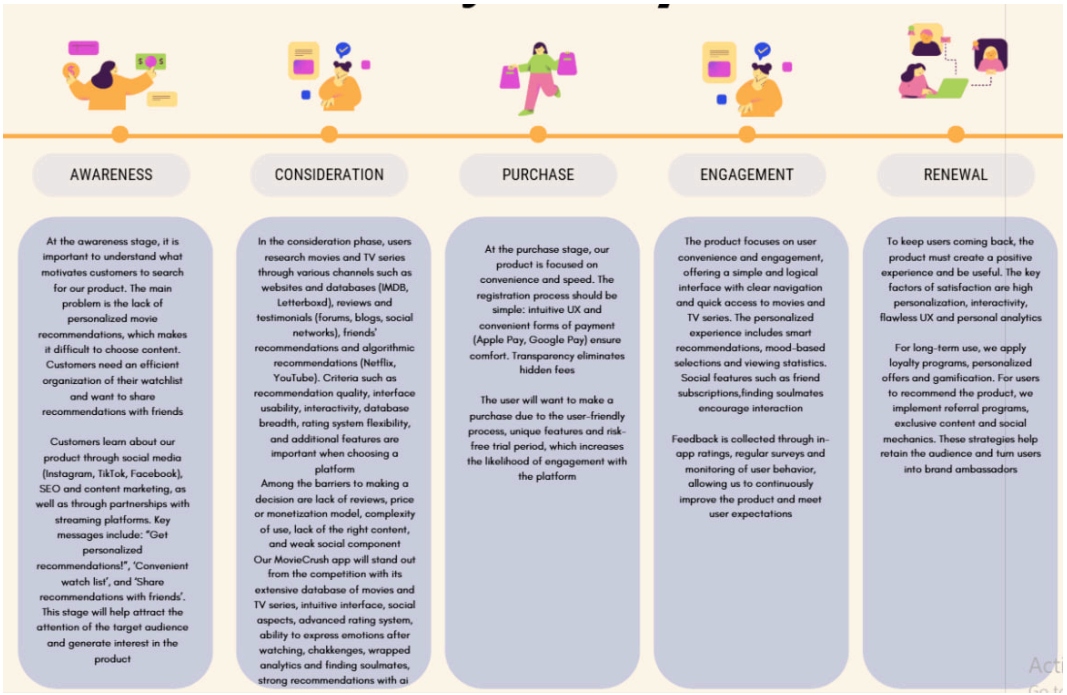


Figure 36: Customer Journey Map - from discovery to long-term retention

AE ML Evaluation Results

AEA Iterative Optimization

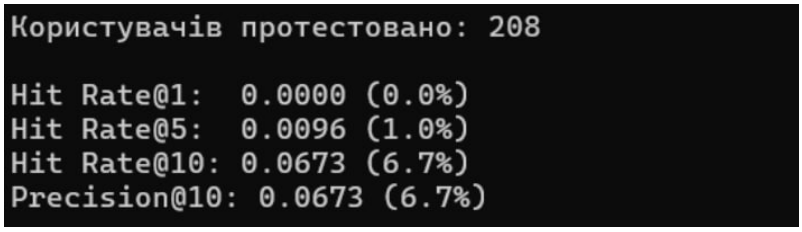


Figure 37: Iteration 1 - baseline results

Figure 38: Iteration 2 - after algorithmic fixes (Hit Rate@10 tripled)

Figure 39: Iteration 3 - after realistic data generation

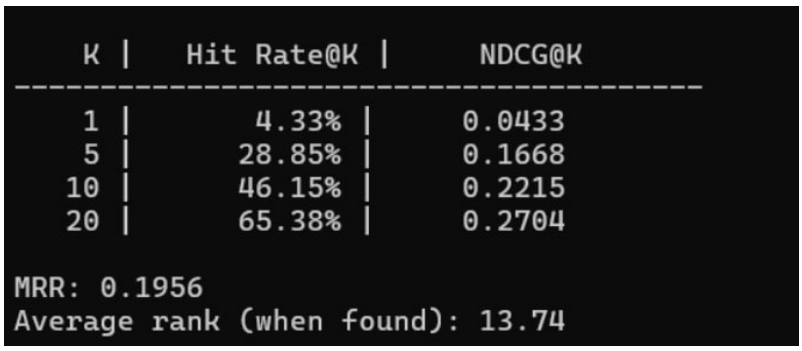


Figure 40: Iteration 4 - Hit Rate and NDCG

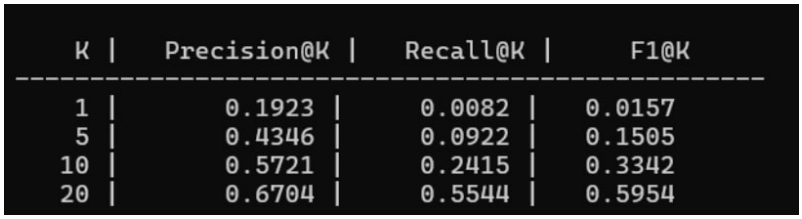


Figure 41: Iteration 4 - Precision, Recall, F1, ROC AUC = 0.8946

AEB Grid Search

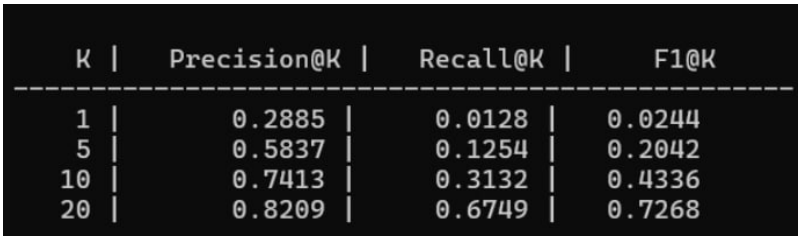
```
Building matrix...
Fetching interactions...
Matrix shape: (208, 265)
Sparsity: 89.78%
Total interactions: 5633
Shape: (208, 265), NNZ: 5633

Users to test: 208
Total combinations to test: 60

 factors  alpha   reg |   HR@5  HR@10  HR@20     MRR  AvgRank
------------------------------------------------------------------------
      12     20   0.1 | 23.56% 40.38% 64.42%  0.2087     14.4  [1/60]
      12     20   0.5 | 23.08% 43.27% 66.83%  0.1882     14.3  [2/60]
      12     20   1.0 | 25.00% 40.87% 64.42%  0.1899     15.1  [3/60]
      12     40   0.1 | 17.79% 37.98% 62.02%  0.1554     15.8  [4/60]
      12     40   0.5 | 18.27% 31.73% 54.81%  0.1613     18.2  [5/60]
      12     40   1.0 | 17.79% 39.42% 57.69%  0.1345     18.4  [6/60]
      12     60   0.1 | 21.63% 35.10% 59.62%  0.1873     17.2  [7/60]
      12     60   0.5 | 21.15% 31.73% 50.00%  0.1734     18.7  [8/60]
      12     60   1.0 | 20.67% 32.69% 50.48%  0.1728     17.9  [9/60]
      12     80   0.1 | 21.15% 35.58% 56.73%  0.1794     16.8  [10/60]
      12     80   0.5 | 15.87% 29.33% 47.60%  0.1578     18.9  [11/60]
      12     80   1.0 | 22.60% 30.29% 47.12%  0.1921     18.1  [12/60]
      16     20   0.1 | 30.29% 47.12% 68.75%  0.1986     13.5  [13/60]
      16     20   0.5 | 28.85% 42.79% 70.67%  0.1859     14.1  [14/60]
      16     20   1.0 | 27.40% 50.96% 70.19%  0.2122     13.1  [15/60]
      16     40   0.1 | 23.08% 37.50% 60.58%  0.1578     17.0  [16/60]
      16     40   0.5 | 28.85% 46.15% 65.38%  0.1956     13.7  [17/60]
      16     40   1.0 | 26.44% 43.27% 59.62%  0.1985     15.1  [18/60]
      16     60   0.1 | 22.60% 38.94% 56.25%  0.1861     17.4  [19/60]
      16     60   0.5 | 22.60% 37.50% 57.21%  0.1811     16.7  [20/60]
      16     60   1.0 | 26.92% 38.94% 61.06%  0.2039     15.0  [21/60]
      16     80   0.1 | 18.75% 36.54% 55.29%  0.1663     18.0  [22/60]
      16     80   0.5 | 21.63% 33.17% 49.52%  0.1987     17.4  [23/60]
      16     80   1.0 | 25.48% 37.98% 54.81%  0.2012     16.6  [24/60]
      20     20   0.1 | 29.33% 51.92% 71.63%  0.2044     11.3  [25/60]
      20     20   0.5 | 30.29% 50.96% 72.60%  0.2104     12.1  [26/60]
```

Figure 42: Grid search - 60 hyperparameter combinations

AEC Final Results

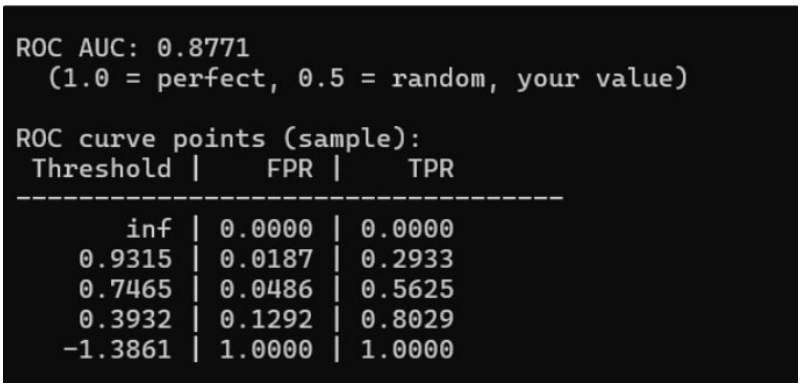
```
   K |   Hit Rate@K |     NDCG@K
-------------------------------------
   1 |      10.58% |     0.1058
   5 |      32.69% |     0.2174
  10 |      51.92% |     0.2802
  20 |      63.94% |     0.3115

MRR: 0.2639
Average rank (when found): 12.11
```

Figure 43: Final - Hit Rate and NDCG



K	Precision@K	Recall@K	F1@K
1	0.2885	0.0128	0.0244
5	0.5837	0.1254	0.2042
10	0.7413	0.3132	0.4336
20	0.8209	0.6749	0.7268

Figure 44: Final - Precision, Recall, F1



ROC AUC: 0.8771
(1.0 = perfect, 0.5 = random, your value)

ROC curve points (sample):

Threshold	FPR	TPR
inf	0.0000	0.0000
0.9315	0.0187	0.2933
0.7465	0.0486	0.5625
0.3932	0.1292	0.8029
-1.3861	1.0000	1.0000

Figure 45: Final ROC AUC = 0.8771