MOODFLIX

SOFTWARE DEVELOPEMENT LIFE CYCLE

Read More

OUR AGENDA

Planning 01

Requirements 02

Design 03

Developement 04

Testing 05

Deployment 06

Maintenance 07

PLANNING

Choices / Theories Learned

- Planning Phase – Key Points
- Define app scope & goals
- Check feasibility (tech, cost, operations)
- Gather requirements early
- Identify & plan for risks
- Choose methodology: Waterfall vs Agile
- Plan resources & timeline

Our Choice / Application

- Scope & Goal: Mobile app recommends movies based on mood
- Feasibility: Prototype-first, no heavy backend needed initially
- Requirements: Core features: mood selection, movie database, search, favorites
- Project Management: Agile/Scrum-lite with short iterations
- Resources: Figma for design, small mobile friendly design, free movie data sets, small team

Reason for Our Choice

- Clear scope avoids feature creep
- Prototype-first validated idea quickly & cheaply
- Agile allowed easy feedback-driven changes
- Lightweight tools kept costs and complexity low

REQUIREMENTS

Choices / Theory Learned

- Gather functional and non-functional requirements from stakeholders (users, developers, business team).
- Use interviews, surveys, and observation to understand user moods and preferences.
- Prioritize requirements using MoSCoW method (Must have, Should have, Could have, Won't have).

Reason / Justification

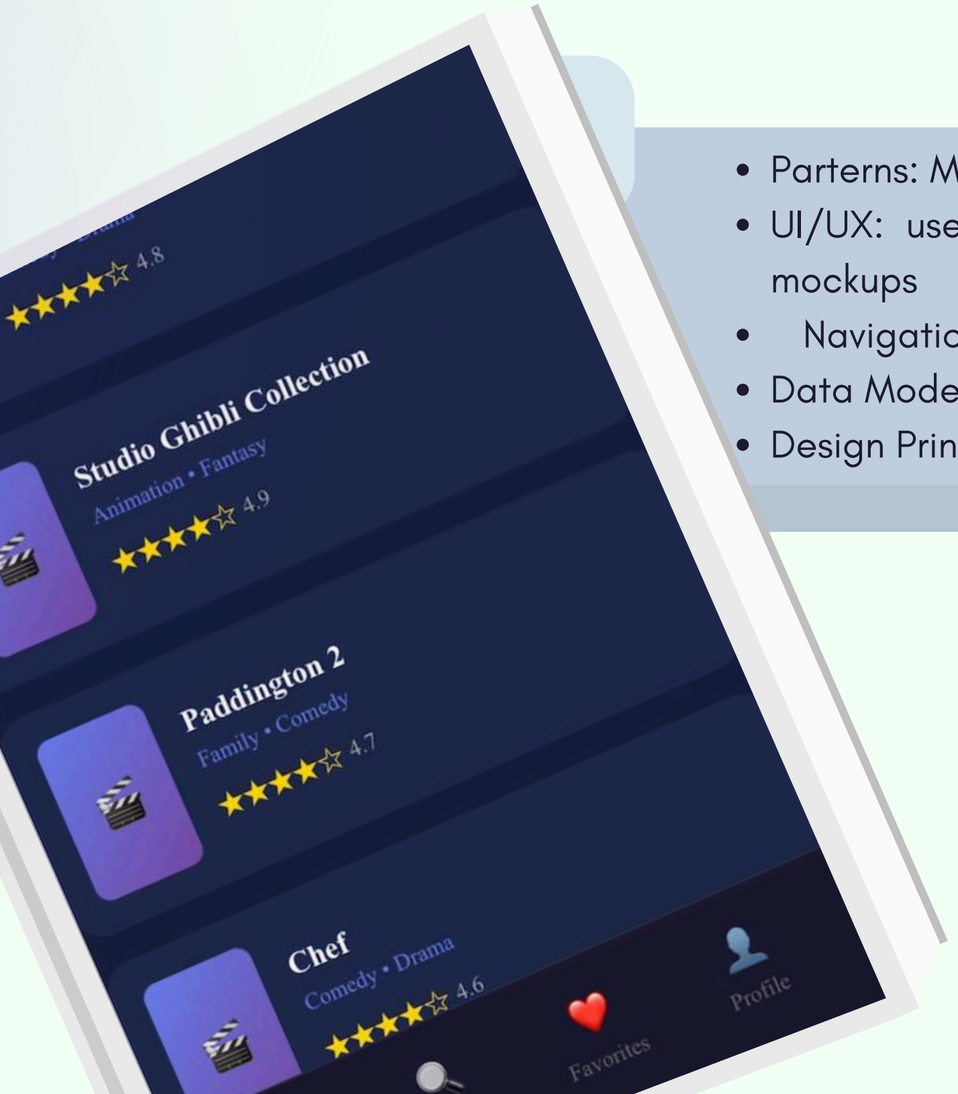
- Ensures MoodFlix addresses real user emotions effectively.
- Reduces development risk by clarifying requirements upfront.
- Supports delivering a satisfying, stress-free movie experience tailored to users' feelings.

Chosen Approach / Application

- Applied user-centered analysis to focus on emotional-based recommendations.
- Created user personas to capture specific needs.
- Defined core features: mood detection, personalized movie suggestions, seamless UX.

DESIGN

Overview

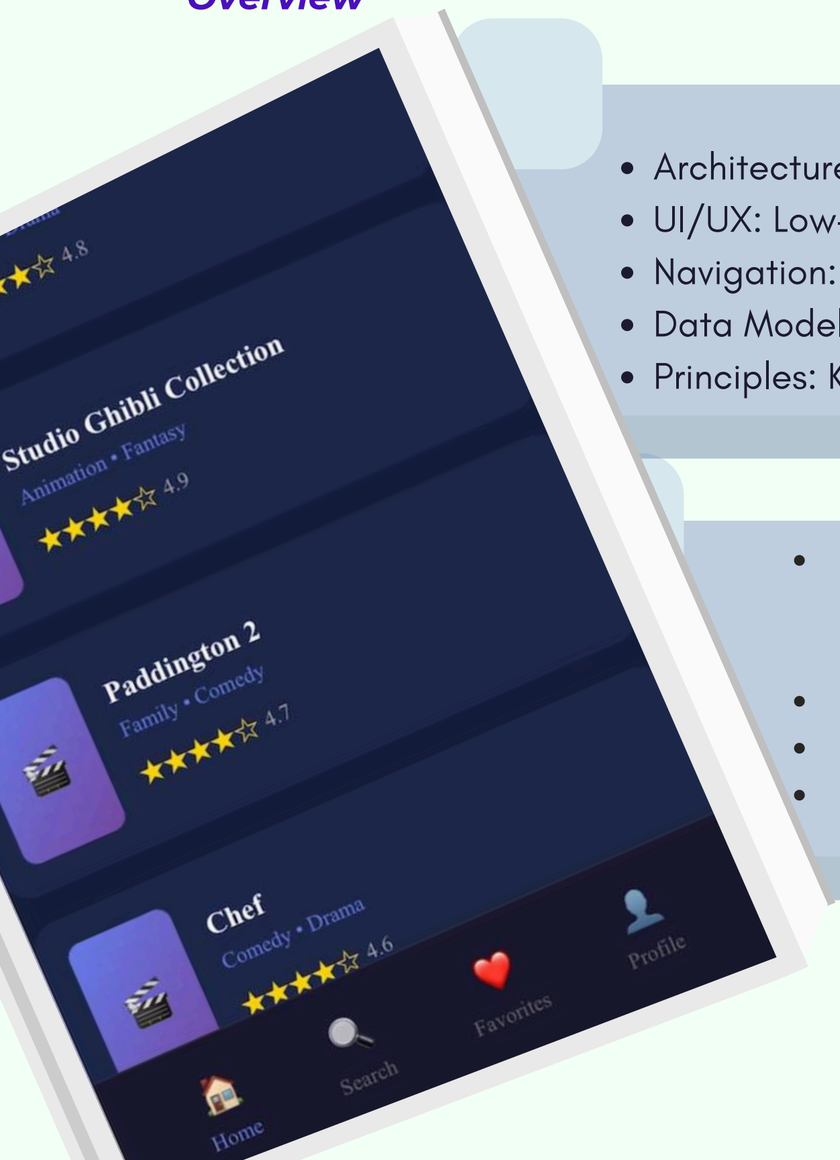


Choices and Theories:

- Parterns: MVC, MVVM, layered
- UI/UX: user-centered Design, wireframes, high-fidelity mockups
- Navigation Partterns: Bottom Bar vs Hamburger menu
- Data Model: Relational VS NoSQL
- Design Principle: SOLID, KISS, DRY, accessibility

DESIGN

Overview

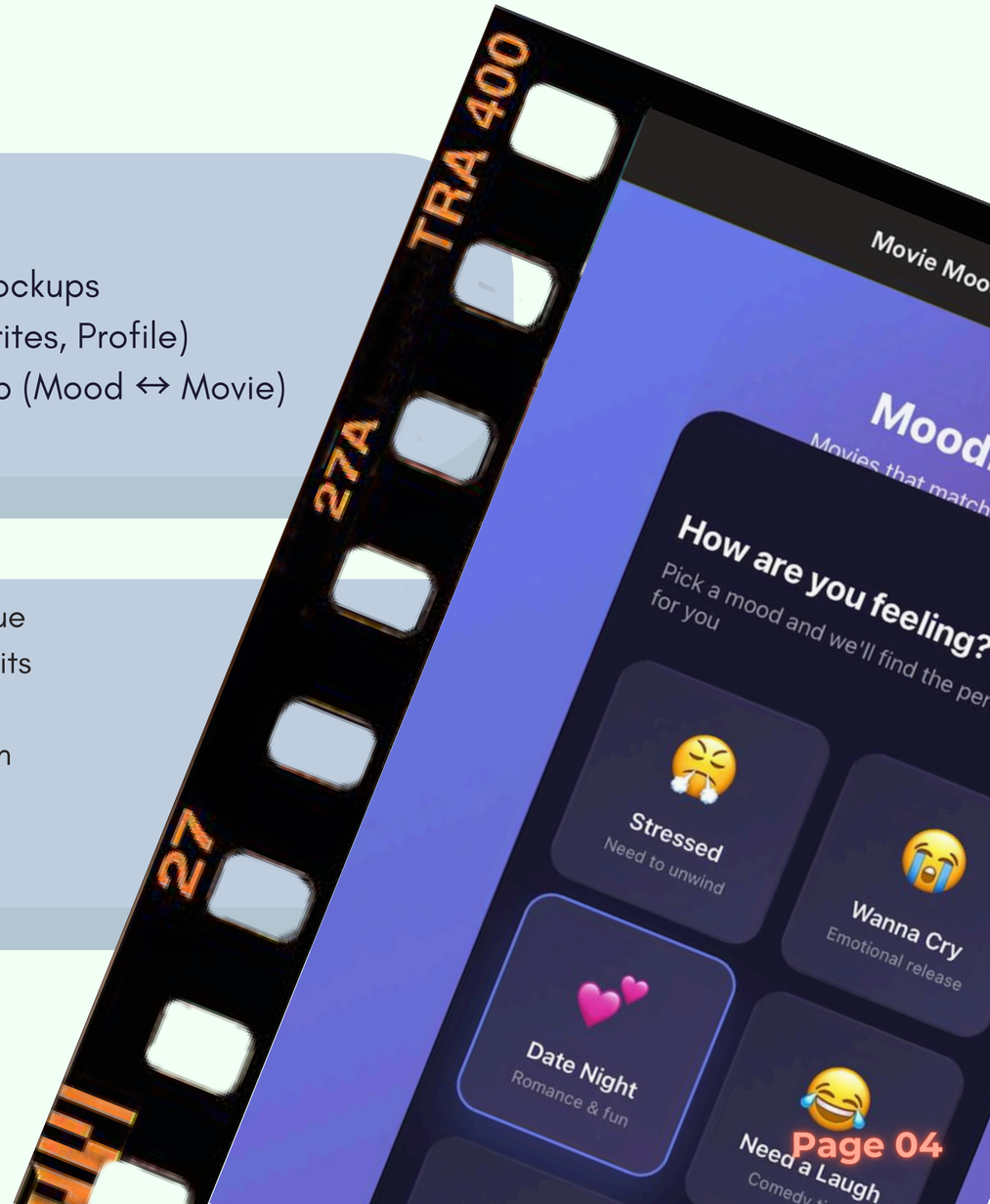


Our Application

- Architecture: MVC pattern
- UI/UX: Low-fidelity wireframes → high-fidelity Figma mockups
- Navigation: Bottom navigation bar (Home, Search, Favorites, Profile)
- Data Model: ER diagram with many-to-many relationship (Mood ↔ Movie)
- Principles: KISS for simplicity, SOLID for maintainability

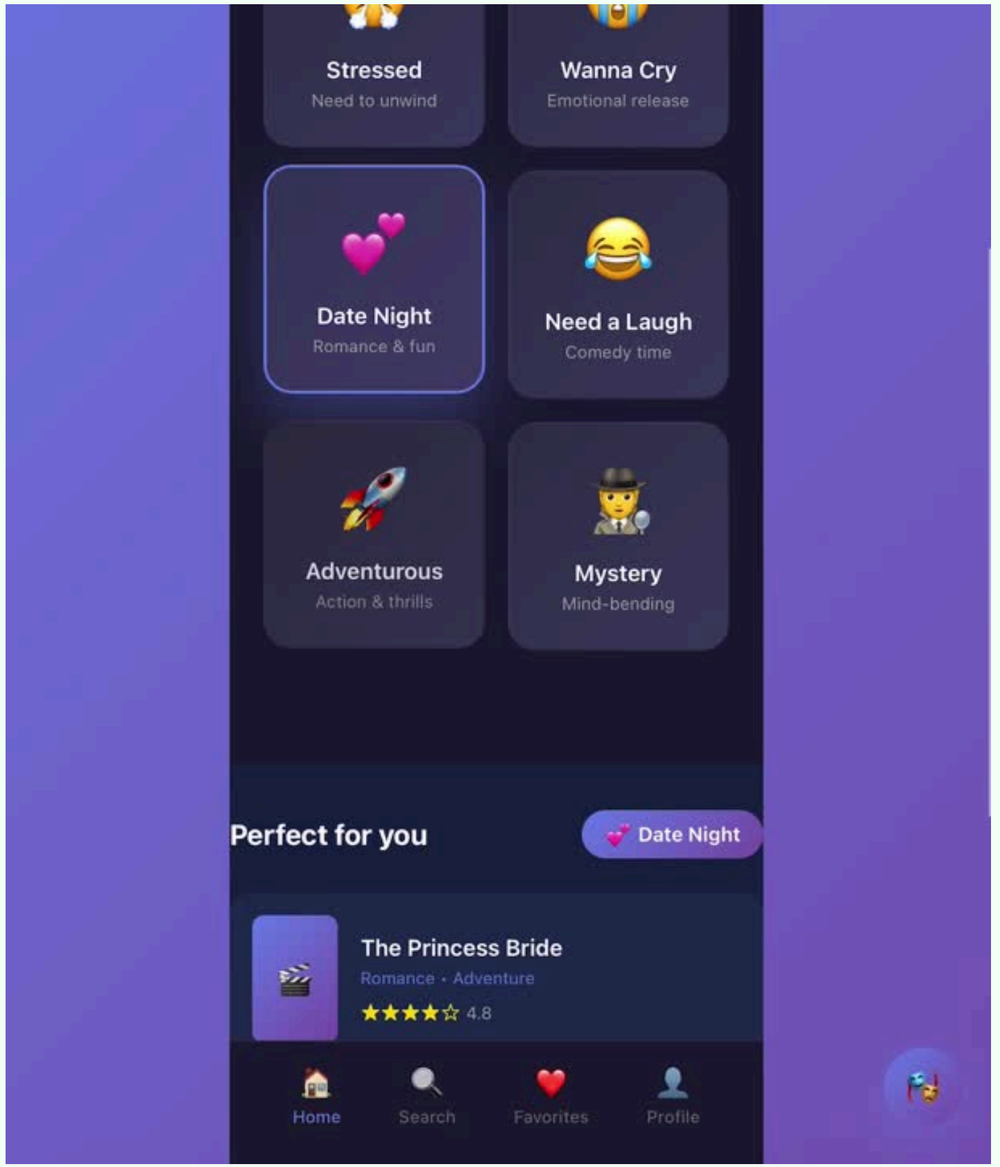
Why?

- User-first experience: Clean dark purple UI (unique creativity and mystery) with bright emoji buttons fits the cinema vibe
- Scalability: MVC enables easy feature expansion
- Familiar navigation lowers learning curve
- Data integrity ensures reliable mood-to-movie matching



DESIGN

Overview



DEVELOPEMENT

Choices / Theory Learned

- During SDLC development, key choices include programming languages, frameworks, database systems, and integration methods.
- Theory emphasizes using the most suitable technology to meet functional and non-functional requirements.
- Agile development principles allow iterative coding, continuous testing, and user feedback integration.

Application / Choice for MoodFlix

- We chose React for front-end, Node.js for back-end, and MongoDB for the database to support real-time mood-based recommendations.
- Implemented modular code structure to handle dynamic movie suggestions based on user emotions.
- Continuous testing ensured the recommendation system aligns with moods: 💔, 😂, 😴, 😡.

Reason / Why

- These choices enhance flexibility, scalability, and user experience.
- Agile approach allows quick adjustments to improve emotional accuracy of movie recommendations.
- Goal: Watching a movie feels effortless, fun, and emotionally “just right”.

TESTING

Choices / Theory Learned

- We learned about various testing methods: unit testing, integration testing, system testing, acceptance testing, and usability testing.
- Each method ensures software quality at different levels: functionality, performance, and user experience.

Application / Chosen Approach

- For MoodFlix, we applied system testing and user acceptance testing (UAT).
- System testing ensures that the mood-based recommendation algorithm correctly matches movies to user emotions.
- UAT involves real users interacting with the app to confirm it delivers a seamless, intuitive, and satisfying movie selection experience.

Reason / Why

- Testing at both system and user levels guarantees that MoodFlix works reliably under various scenarios.
- Ensures users receive accurate recommendations that align with their emotional state, enhancing satisfaction and engagement.
- Reduces the risk of errors before the final launch, ensuring a high-quality product.

DEPLOYMENT

Choices / Theories Learned

- Centralized Portal Approach - A single official app store, like Apple's App Store or Google Play, where most apps are published.
- Decentralized Portal Approach - Developers can publish apps on multiple independent portals, offering flexibility but less control.
- Open-source platforms (e.g., Android, Symbian) allow more developer freedom. Closed platforms

(e.g., iOS) require strict approval and control by the platform provider.

Application / Chosen Approach

We choose the Centralized Portal Approach (e.g., deploying through the Google Play Store or Apple App Store). To do this:

- Register as a developer on the chosen platform and pay the annual fee.
- Carefully review and comply with all app store policies.
- Optimize the app by cleaning up code and removing logs or unnecessary files.
- Prepare high-quality visual assets, including the app icon and store screenshots.
- Package the application in the required format (.apk for Android or .ipa for iOS).
- Upload the application to the centralized portal and complete the required metadata such as app description, category, and pricing.
- Launch the app and monitor performance metrics using the app store dashboard.

Reasons

- Wider reach: access to millions of global users
- Trust & Security: Verified apps gain user confidence.
- Simplified Process: Handles billing, updates, and distribution.
- Built-in Promotion: Ratings and featured sections boost visibility.
- Data Insights: Provides analytics for future improvements.

MAINTENANCE

Choices / Theories (Models of Maintenance)

- Emergency-Oriented Maintenance → Fix urgent bugs/crashes.
- Feature-Oriented Maintenance → Frequent new features + stability.
- Constant Maintenance → Regular, short-cycle updates, focus on process discipline.
- (Also covers Corrective, Adaptive, Perfective, Preventive maintenance types).

Chosen Model / Application

- Constant Maintenance
- Apply short iterations (10–20 days).
- Regular bug fixes, improvements, and security patches.
- Add new features occasionally when user demand arises.
- Use Agile methods for flexibility.

Why This Choice?

- Balances stability and adaptability.
- Enhances user satisfaction with continuous improvements.
- Maintains market competitiveness with timely updates.
- Supports Agile principles for fast response to change.

THANK YOU

THANK YOU

Aung Ko Ko (240702402730)

Aung Kyaw Paing (240702401939)

Si Thu Htut (240702401885)

Pognleap Peung (220702403584)

Pyae Phyo Kyaw (240702401794)

Nguy Reach (230702403125)

Thin Lay Pyay (230702403116)

End