PART III – Reflections and Peer Evaluation:

III-a: Prepare a PDF file named "M3_Reflections.pdf", with the following information.

1. Fill out "Table 1 – Task Distribution" below:

ID	Task (a short description)	Team Member	Duration
1	Implement Google OAuth authentication flow (frontend & backend)	Jimmy & Kayli	3 hours
2	Design and implement friend request system with SSE real-time updates	Jimmy & Kayli	5 hours
3	Design and implement activity feed with real-time updates	Jimmy & Kayli	4 hours
4	Design and implement Firebase Cloud Messaging for push notifications	Jimmy & Kayli	4 hours
5	Implement initial simple movie search and TMDB API integration	Jimmy & Kayli	3 hours
6	Implement watchlist feature and synchronization with rankings	Jimmy & Kayli	3 hours
7	Improve movie ranking comparison algorithm	Jimmy & Kayli	6 hours
8	Improve recommendation algorithm with user preference analysis	Jimmy & Kayli	5 hours
9	Implement trending movies feature for new users	Jimmy & Kayli	2 hours
10	Implement movie detail endpoints and watch providers	Jimmy & Kayli	3 hours
11	Implement in-app trailer playback with WebView	Jimmy & Kayli	3 hours

12	Design and implement Jetpack Compose UI for all screens	Jimmy & Kayli	8 hours
13	Implement DataStore for local data persistence	Jimmy & Kayli	2 hours
14	Backend API server setup with Express.js and MongoDB	Jimmy & Kayli	4 hours
15	Write comprehensive API documentation (Section 4.1 - Component Interfaces)	Jimmy & Kayli	4 hours
16	Document frameworks and libraries (Section 4.4)	Jimmy & Kayli	2 hours
17	Create sequence diagrams structure (Section 4.6)	Jimmy & Kayli	2 hours
18	Document NFR implementation (Section 4.7)	Jimmy & Kayli	2 hours
19	Review and correct Requirements_and_Design.md documentation	Jimmy & Kayli	3 hours
20	Testing and bug fixes across all features	Jimmy & Kayli	6 hours
21	Debug and fix watchlist-ranking synchronization	Jimmy & Kayli	2 hours
22	Fix duplicate feed entries bug on rerank	Jimmy & Kayli	1 hours
23	Deployment and server configuration	Jimmy	2 hours
24	Created Project on Azure and added Members	Muskan & Vansh	2 hours
25	Setting up android studio and local backend (google API key setup for local backend)	Muskan & Vansh	1.5 hours
26	Implemented the movie ranking Algorithm	Muskan & Vansh	4 hours

27	Implemented the Recommendations Algorithm	Muskan & Vansh	2 hours
28	Made the sequence diagram for the compare movie use case and the recommendation feature use case	Muskan & Vansh	2 hours

2. Fill out "Table 2 – Time and AI Reliance Distribution" below:

	Time Spent on the Assignment (hours)	**AI Reliance for the assignment (0-100%) (Estimate)**
Jimmy Chen	77 Hours	90%
Kayli Cheung	79 Hours	90%
Vansh Khandelia	15.5 Hours	50%
Muskan Bhatia	15.5 Hours	50%
Group Overall	187 Hours	70%

3. Your reflections on the use of AI:

3.1 Pick 4-5 most major tasks from the table above and specify:

Task 1: Friend Management with Real-Time Updates

3.1.1 What was your task?

Deliver the complete friend experience: searching by email, sending/canceling/responding to requests, surfacing incoming/outgoing lists, and reflecting status changes instantly through SSE and push notifications. On backend that meant building Friendship/FriendRequest models, REST routes, SSE broadcasting, and FCM hooks. On Android it required new screens, view models, and Compose components that consumed those APIs and stayed in sync with live events.

3.1.2 Which AI Interfaces, Tools, and Models have you used?

- Claude Code
- GitHub Copilot
- ChatGPT5

3.1.3 What was your strategy for utilizing the tool?

I first asked Claude to enumerate existing user/auth infrastructure so the new routes slotted in cleanly. After generating base controllers and SSE service updates, I manually hardened them (rate limiting, bidirectional friendship creation). For Android, I used Copilot to spin up ViewModel state holders and LazyColumn layouts, then iteratively tested while Kayli exercised the flows. We used Claude as a diff tool whenever we refactored to ensure both client and server stayed aligned.

3.1.4 Advantages of using AI tools:

- Boilerplate acceleration: Claude/Copilot handled repetitive schema definitions, DTOs, and Retrofit signatures.
- Edge-case reminders: AI highlighted scenarios like self-invite prevention, duplicate requests, and bilateral friendship cleanup.
- Live event wiring: Claude suggested SSE client management patterns (Map of user IDs to response sets) that we adopted.
- Notification integration: ChatGPT provided clear FCM payload structures which we adapted for friend requests/acceptances.

3.1.5 Disadvantages of using AI tools:

- State drift: Copilot occasionally generated stale field names, which caused runtime crashes until we manually synced models.
- Security sanity checks: AI underplayed rate limiting and authorization checks; we added manual guards for abuse cases.
- SSE disconnect handling: Claude's initial res.on('close') code leaked listeners; manual profiling revealed and fixed it.
- UX nuances: AI didn't account for empty-state design or inline error surfacing—we layered that in via manual iterations.

Task 2: Implement Firebase Cloud Messaging for Push Notifications

3.1.1 What was your task?

Set up Firebase Cloud Messaging (FCM) for sending real-time push notifications to users when friends rank movies or send friend requests. This involved configuring Firebase project credentials, implementing backend service using firebase-admin SDK to send notifications, and handling device token registration on Android using firebase-messaging library.

3.1.2 Which AI Interfaces, Tools, and Models have you used?

- Claude Code (claude-sonnet-4-5-20250929)
- ChatGPT5
- GitHub Copilot

3.1.3 What was your strategy for utilizing the tool?

First, used ChatGPT to understand Firebase setup requirements and get step-by-step configuration instructions. Then used Claude Code to implement the backend notification service by reading Firebase Admin SDK documentation and generating the service code. For Android integration, relied heavily on GitHub Copilot to autocomplete the Kotlin code for handling FCM tokens and notification callbacks. Claude Code helped integrate the notification sending into existing endpoints (friend request, movie ranking completion).

3.1.4 Advantages of using AI tools:

- Configuration guidance: ChatGPT provided clear steps for Firebase console setup, preventing common misconfiguration issues
- Boilerplate reduction: Copilot generated 70%+ of the repetitive Kotlin code for notification handling
- Integration assistance: Claude Code identified all the points in the codebase where notifications should be triggered
- Error handling: AI suggested proper try-catch blocks and fallback behavior when notifications fail

3.1.5 Disadvantages of using AI tools:

- Outdated documentation: ChatGPT sometimes referenced older Firebase SDK versions; had to manually verify against current docs
- Testing limitations: AI couldn't test actual notification delivery; had to manually test on physical device
- Security concerns: Initial Claude-generated code had the Firebase private key hardcoded; had to manually move it to environment variables
- Google services JSON: AI wanted me to commit google-services.json, but that's a security risk (later had to remove from git history)

Task 3: Implement Movie Ranking Comparison Algorithm

3.1.1 What was your task?

Implement an interactive binary comparison algorithm for ranking movies. Users compare a new movie against existing ranked movies in pairwise comparisons, and the system determines the final rank position. The algorithm needed to minimize comparisons (logarithmic time) while ensuring accurate placement and handling edge cases like empty lists or duplicate movies.

3.1.2 Which AI Interfaces, Tools, and Models have you used?

- ChatGPT5

3.1.3 What was your strategy for utilizing the tool?

First, I used ChatGPT to brainstorm different ranking algorithm approaches (binary search, quicksort-like comparisons, Elo rating). After deciding on binary search, I used ChatGPT to implement the backend logic in movieComparisonController.ts. Then it helped design the session-based state management (in-memory map) to track comparison progress across multiple HTTP requests.

3.1.4 Advantages of using AI tools:

- **Algorithm optimization:** Claude suggested using binary search with range tracking instead of linear comparisons, reducing from O(n) to O(log n) comparisons
- **Edge case coverage:** AI identified edge cases I hadn't considered (empty list, duplicate movie, session expiration)
- **Code quality:** Copilot's autocomplete sped up writing TypeScript type definitions and validation logic
- **Testing suggestions:** Claude suggested test scenarios to verify the algorithm worked correctly

3.1.5 Disadvantages of using AI tools:

- **Over-engineering tendency:** Initial csuggestion was overly complex with persistent database storage for sessions; had to simplify to in-memory Map
- **Testing gaps:** AI-generated test scenarios didn't catch a race condition bug that only appeared with concurrent requests from the same user

Task 4: Build Feed Engagement UI (Likes, Comments, Sheets)

3.1.1 What was your task?

Deliver the end-to-end experience for engaging with feed items on Android: render like/comment counts, add optimistic like toggles, surface the full-screen comment sheet, and coordinate with backend SSE/FCM notifications. This required updating multiple Compose screens (feed list, movie detail sheet, comment bottom sheet) and ensuring the UX stayed responsive across loading states.

3.1.2 Which AI Interfaces, Tools, and Models have you used?

- Claude Code (claude-sonnet-4-5-20250929)
- GitHub Copilot
- ChatGPT5

3.1.3 What was your strategy for utilizing the tool?

I start by asking Claude to diff the latest backend changes so I knew which fields (like likeCount, commentCount, isLikedByUser) were available. Then, while wiring those fields into FeedActivityCard and CommentBottomSheet, I leaned on Copilot to scaffold snippets (e.g., AnimatedVisibility, LazyColumn placeholders). After assembling the flow, I used Claude's "read" tool to review the composed files to catch state bugs (like stale selections) and generated follow-up tweaks via targeted prompts.

3.1.4 Advantages of using AI tools:

- Rapid scaffolding: Copilot produced most of the repetitive Compose state code, letting me focus on UX polish.
- State auditing: Claude spotted spots where I forgot to reset selectedMovie or where coroutines needed rememberCoroutineScope().
- Design iteration: ChatGPT helped brainstorm concise copy and empty-state messaging that matched our tone.
- Regression checks: Claude compared pre/post diffs to ensure theming (colors, typography) stayed consistent.

3.1.5 Disadvantages of using AI tools:

- Compose churn: Copilot occasionally suggested deprecated APIs (rememberCoroutineScope misuse), so I had to double-check with docs.
- Over-eager animations: AI kept inserting fancy transitions that hurt performance; I scaled them back manually.
- Comment sheet race conditions: Claude didn't catch that opening the sheet while loading comments needed guard clauses; manual testing revealed it.
- Accessibility gaps: AI rarely suggested content descriptions or talkback hints—I had to add those deliberately.

Task 5: Recommendations Algorithm

3.1.1 What was your task? (a longer description)

My task was to implement the Recommendations Algorithm for our application. The feature analyzes ranked movies, and uses TMDB API to suggest a personalized list of recommended movies. This required designing and integrating a logic layer that dynamically updates suggestions based on changing user preferences.

3.1.2 Which AI Interfaces, Tools, and Models have you used for this task?

ChatGPT (GPT 5.0)

3.1.3 What was your strategy for utilizing the tool for this task?

I started by clearly explaining the goal of the feature to the model, including edge cases and how the algorithm needed to behave when user data was limited or ambiguous. I explained ChatGPT, the algorithm I wanted (Picking the top 20% Movies, and generating a list of recommended movies and removing already ranked movies from that list). Once ChatGPT generated a solution, I reviewed the code to make sure the logic aligned with our app's data structure and functionality. I then refined and tested the code in context to confirm it produced accurate recommendations.

3.1.4 What are the advantages of using AI tools (and particular models, if relevant) for this task?

- Idea Validation
- Faster coding support
- Helpful in catching potential gaps in logic or edge cases

3.1.5 What are the disadvantages of using AI tools (and particular models, if relevant) for this task?

- Had to sometimes rewrite code for performance and decrease redundancy.