Development Plan
Flick Picker

Team 7, 7eam
Talha Asif - asift
Jarrod Colwell - colwellj
Madhi Nagarajan - nagarajm
Andrew Carvalino - carvalia
Ali Tabar - sahraeia

Table 1: Revision History

Date	Developer(s)	Change
Sept 21/22	Talha	Updating Workflow Plan
Sept 24/22	Talha	Updating Sections 1-3
Sept 25/22	Madhi	Updating Sections 6 and 7
Sept 26/22	Ali	Updating Sections 5 and 8
Sept 26/22	Madhi	Small revisions

Contents

This documentation details the entire development plan, from team meetings, to the workflow, all the way to the technology that Flick Picker will use. 7eam fully understand their responsibilities, project goals, and upcoming challenges.

1 Team Meeting Plan

There will be an ad hoc meeting weekly to ensure the team is on the same page, where a time is picked that works for all five team members. The reason for the meeting will also be clearly stated before meeting, and we will start as soon as everyone is in the call.

2 Team Communication Plan

7eam will utilize Discord as their main form of communication, a server has been created with all forms of communication in it currently. It is expected if something urgently needs a specific team member, they will be responsive within 24 hours.

3 Team Member Roles

Everyone shares responsibilities and is required to be flexible when needed, but there are areas of development each individual specializes in to keep work split evenly.

Table 2: Team Developer Roles

Team Member	**Role**
Talha	DevOps, Full-Stack Developer
Jarrod	Back-End Developer, Back-End Tester
Madhi	Full-Stack Developer, Full-Stack Tester
Andrew	Front-End Developer, Front-End Tester
Ali	Front-End Developer, Front-End Tester

4 Workflow Plan

Git Workflow:

- *develop* branch will be the single source of truth, where the team reviews changes before they get merged. Thus the *develop* branch will have restricted permissions on it, preventing direct merges without admin overwrite, and only one developer will be the admin, Talha

- Any changes have to be on their branch, and a PR has to be cut with a full green checklist to get it merged into *develop*
- The checklist will grow as development on the application continues, currently, the checklist is a single item, where the PR must have two approvals from the team. The future checklist items are as planned:
  - Entire test run has to be successful to ensure *develop* is in a healthy state
  - Test coverage delta must not be reduced unless redundant tests are taken out
  - Custom Linting rules (ESLint + Airbnb styling) must pass
  - Snyk/Dependabot checks must pass, ensuring the packages used do not have vulnerabilities
- Each PR must have at minimum a description filled out, and a relevant feature ticket must have a Jira issue attached along with it
- If a feature has an attached ticket, the PR title must start with [CAP-##] and then a title
- Every time a PR is made, automation will ping #pr-bot on Discord so the team is aware of changes being made

Issue Tracking - Jira:

- All development changes need a descriptive ticket cut before it is ready for code review. Automation will link the PR to the ticket and vice versa as well
  - Descriptive means the ticket must have a title and acceptance criteria (AC) to complete the ticket
- The status of the ticket must be updated on the board. Most importantly, ticket status should be "In Progress" so multiple developers do not start working on the same feature
- Points will be arbitrary for the ticket, estimated by the developer who is working on it, this will be an indication of how complex the work is for the reviewers
- Utilize Jira's ticket types to have issue classifications:
  - Story: Ticket describing a new feature
  - Bug: Fixing existing code
  - Task: Small changes that do not fall in either of the above categories

4.1 Example Feature Workflow

An example developer workflow for a feature will be as follows:

1. Jira ticket is created with a description and AC, assigned to a developer, then the ticket is moved to “In Progress” while it is in development
2. Branch is made for development and then marked “Ready for Review” when the owner thinks the AC are met
3. Reviewers ensure AC are met and healthy coding practices are followed
4. PR is merged, and the ticket is moved to “Done”

4.2 Example Documentation Workflow

An example documentation workflow will be as follows:

1. Asynchronous discussion on Discord is done to choose which sections of a documentation to update
2. Branch is made for updates and then marked “Ready for Review” when the owner thinks the documentation has been sufficiently written
3. PR is merged

5 Proof of Concept Demonstration Plan

The main risk for the success of the project would be the application not being able to find something desirable to watch for the user group. The sole purpose of the application is to find shows or movies that all or most members of the group will like, taking all of their specific preferences into account, such as the genres they like. However, the project will be useless if it is unable to do this. The algorithm used to find new media to watch based on the preferences of multiple people should be effective in finding something that as many people as possible will enjoy watching, not just a few people of the group, or worst-case scenario, none of the group. During our proof of concept demo, we can prove we can overcome this risk by using mock test data to represent different user groups, using group sizes from $2 - 100+$. If a resultant movie or show shares the majority of the mock group’s preferences, we will know it is an effective algorithm.

6 Technology

6.1 Languages/Frameworks

Flick Picker will be a full-stack web application. The frontend layer of the application will be built with React.js. React.js is a web application framework that is based on the Node.js environment. The programming language behind React.js is TypeScript, a flavour of JavaScript. TypeScript ensures code correctness due to its strongly-typed nature, and catches any JavaScript runtime errors prior to code compilation.

Another key portion of the Flick Picker application is the backend. The backend will be a RESTful API that acts as intermediary between the frontend and the database to manage user data. It will also contain the core logic of our movie recommendation system and our movie matching algorithm. The choice of framework for the backend will be Express.js, a Node.js framework for RESTful APIs. Our language choice is TypeScript, as we felt that it would be the most effective to keep languages consistent at a full-stack level.

6.2 Testing

Within both the frontend and backend, unit tests will be present to look for sufficient code quality and correctness. The developer will write appropriate unit tests to reach desired code coverage, and tests will be ran prior to each code deployment. Unit testing will be done with use of Jest, a JavaScript/TypeScript testing framework. Jest also contains functionality for producing code coverage results.

Integration tests will also implemented to test the application as a whole, ensuring that it functions together as expected. Cucumber.js will be the choice of framework for integration tests. Gherkin is the language used for Cucumber tests. Gherkin follows the "Given, When, Then" structure for a set of "steps" AKA a test case. Cucumber.js also utilizes JavaScript/TypeScript when defining step functions (the logic behind steps).

6.3 Linting

Our team will follow the Airbnb JavaScript Style Guide for our linting standards. Since our team utilizes TypeScript, our project will use a flavour of ESLint that includes the Airbnb standards (Airbnb ESLint Package).

6.4 Storage & Deployment

For our database storage and app deployment preferences, our team will utilize Google's Firebase ecosystem. We noticed that Firebase has a variety of features that are easy to setup and are scalable. Our database will be hosted through the Cloud Firestore feature. Cloud Firestore allows us to store and query our data within a NoSQL database.

Our application deployment will be done through Firebase Hosting, as this product is already optimized for applications built with the Node.js runtime environment. Firebase Hosting also features one-click app deployments.

6.5 Continuous Integration (CI)

To reiterate from Section 4, our project will utilize a variety of tools for CI. Our workflow plan will be executed for every Pull Request (PR). Below are a list of tools used by our workflow plan.

- GitHub Workflow: For seamless integration with our GitHub repositories, we will implement our workflow plans through GitHub Workflows.
- Node.js & npm: Since both our frontend and backend is built on Node.js, our workflows will use it for building, testing, and linting checks. For example, GitHub Workflow will execute `npm build` to check whether the project builds successfully. Note that GitHub Workflow supports Node.js.
- Snyk/Dependabot: This tool is used to check for any dependency vulnerabilities. Any vulnerabilities will be alerted, so that we could handle this accordingly.

6.6 Tools/Libraries Needed

Authentication: As introduced in Section 6.4, the Firebase ecosystem will be a key part in our application development. Firebase Authentication will be utilized for secure user sign-up and login. Firebase Authentication also features login authentication through Google and Facebook accounts. Integrating Google & Facebook authentication will be a key stretch goal of ours, as it provides users a more convenient method for logging into our application.

Performance Metrics: Firebase Performance Monitoring will allow our team to monitor our application's performance metrics. The dashboard features key metrics like response times, success rates, payload sizes etc. Throughout the development process, our team will continue to monitor this dashboard to troubleshoot or optimize our application's performance.

External APIs Required: Our application will need to fetch movie and TV show data from one or more external APIs. Below is a list of APIs that we may integrate with our project. Note that these may be subject to change, as there are many factors to consider including potential cost, availability, dataset size, response data etc.

- Open Movie Database (OMDb API) - This is a free Movie & TV Show database API. Note that there is a max of 1000 requests per day for the free version. Depending on our needs, we may need to consider paying for the premium tier.
- MyAnimeList API - This is a database API for Anime Movies & TV shows.

7 Coding Standard

As mentioned within Section 6.3, our team will be following the Airbnb JavaScript Style Guide coupled with ESLint for our style standards. For our code approval process, it will contain strict guidelines to ensure code quality. Developers must add at least 2 reviewers to their Pull Request (PR). Once these reviewers have thoroughly read, understood, and agree with the code changes proposed, then they can proceed with approving the PR. The PR must also pass our Continuous Integration checks (described in Section 4), which will ensure that the project builds successfully, passes all unit tests etc.

8 Project Scheduling

The team will meet once a week, every Monday at 1:00 PM starting from September 26, 2022 to discuss current milestones, responsibilities, and progress. The team will also have a brief virtual meeting two hours before any deliverable's due date to review the deliverable's contents together. If a deliverable is finished early by the team before its due date, the next deliverable can/will be worked on.