**CS673 Software Engineering** 

**Team 4 - Project Name**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Alex Kolbin | Configuration Leader | *AK* | 05/15/2022 |
| Eric Ashton | QA Leader | *EA* | 05/15/2022 |
| Ellie Nerney | Team Leader | *EN* | 05/16/2022 |
| Rui Mao | Design/Implementation Leader | *RM* | 05/16/2022 |
| Sujeet Kumar | Risk Management Leader | *SK* | 05/16/2022 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1** | **Alex Kolbin** | **05/15/2022** | **Configuration Plan** |
| **1** | **Eric Ashton** | **05/15/2022** | **QA Plan** |
| **1** | **Ellie Nerney** | **05/16/2022** | **Management Plan (a)** |
| **1** | **Sujeet Kumar** | **05/16/2022** | **Risk Management (b)** |
| **1** | **Rui Mao** | **5/16/2022** | **Requirements** |

[Overview](#_g6igqliy7rm)

[Related Work](#_bf21eadgjj29)

[Proposed High level Requirements](#_rgyo4hi9stmq)

[Management Plan](#_ts358bsdtbcv)

[Objectives and Priorities](#_nxeeppkjxgn4)

[Risk Management (need to be updated constantly)](#_tk7yixobah8p)

[Timeline (need to be updated at the end of each iteration)](#_iksrndohvx29)

[Configuration Management Plan](#_j5uvivmxqcsp)

[Tools](#_dzly5b9kz982)

[Deployment Plan if applicable](#_sd8zu6r3jisd)

[Quality Assurance Plan](#_vra5ptwu59qx)

[Metrics](#_vwjduhc9wuah)

[Code Review Process](#_hx3eaiwb8v3m)

[Testing](#_l9xnpmd6hh0y)

[Defect Management](#_5amsh8h9f0c7)

[References](#_pd9euov6m4du)

[Glossary](#_ty3i2nqffhtc)

# Overview

There are many successful commercial products using the recommendation system in our daily life. The recommendation system makes it easier to find the products that the customer needs. Since the success of recommendation systems improves our life quality, we are motivated to develop this project.

The potential users are frequent movie audiences who rates the movie.

The basic functionality of the system is to find the top N movies that the user would like to give the high rating. The possible technologies are Artifical Neural NetworkTensorflow, Angular, Relational Database, pyodbc.

# Related Work

Netflix. Our recommendation system is more light-weighted compared to the Netflix recommendation system. Unlike Netflix, we will rank the recommendations by the likelihood that the user would like this movie instead of the likelihood that the user would watch this movie.

# Proposed High level Requirements

* 1. Functional Requirements  
     Essential Features (the core features that you definitely need to finish):

1. As a Design and Implementation leader, I want to develop a recommendation model which predicts the movie so that the user would find the movie he/she likes. 1 week.
2. As a Design and Implementation leader, I want to create a database to store the movie data including genre, director, major actors and the user ratings so that we can train our model. 1 week
3. As a Design and Implementation leader, I want to develop a backend that interacts with the database and the frontend so that we can retrieve user input and provide feedback. 1 week
4. As a Design and Implementation leader, I want to design the frontend that accepts user input and displays a list of recommendations so that the user can have good experience. 1 week
   * 1. Desirable Features (the nice features that you really want to have too):

As a Design and Implementation leader, I want to develop the system to recommend the movie that the user would most likely to watch based on the user preferences and the popularity of the movie..

* + 1. Optional Features (additional cool features that you want to have if there is time):

As a Design and Implementation leader, I want to save the user preferences so that the system can find similar users to make more accurate recommendations.

* 1. Nonfunctional Requirements
     1. Security requirements

Ensure data integrity. The movie data and the user profile cannot be altered

# Management Plan

## Objectives and Priorities

**Complete Essential Features**

Our first priority will be to implement all of the essential functional requirements outlined above. These can be broken into 4 overarching categories:

* 1. Recommendation model- predicts a movie a user might enjoy
  2. Front-end- displays movie recommendations produced by model
  3. Database- hosted online and holds movie data, user data, metadata
  4. Back-end- connects the database to the front end

Each of these categories can and will be broken up into many tickets and stories over the iterations.

**Maintain Good Code Quality**

Throughout the course of the project, we must prioritize code quality and testing to reach these goals:

1. Ensure the software has no known bugs
2. Full\* testing coverage

\* While truly full coverage is preferred, due to the time-constrained nature of this project it’s essential for every feature to be tested but that may be a mix of automated and manual testing.

**Incorporate all separate components into a single working product**

By the end of this project, all of the disparate functional components (model, front-end, back-end and database) should be connected and communicating to each other, so it can be shipped as a whole, functional, product.

## Risk Management (need to be updated constantly)

Technology competence is the major risk associated as most of the team members are not familiar with the framework and programming language selected by the team, there is a huge dependency on individual developers.

**Risk Management Sheet Link:**

<https://docs.google.com/spreadsheets/d/1kcuFWIIEjFtHzJyVQh8WcUB9JUueS7hxYN5F2F3mq8A/edit#gid=0>

## Timeline (need to be updated at the end of each iteration)

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours | Presentation Recording Link (5-10 minutes for iteration 1 and 2) |
| --- | --- | --- | --- | --- |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |

# Configuration Management Plan

## Tools

**Git Tools:** Because we are using GitHub for version control, our team will also utilize GitHub Projects for issue management, and GitHub Issues for issue tracking. These two tools work closely together, and will allow us to have everything in a centralized platform.

**IDE Tools:** The team can use whatever IDE they prefer for local development. If we reach a point where our development requires a specific functionality provided by a particular IDE, then we may recommend that one to be used. If needed, members can use git functionality provided by most IDEs to help manage contributions to the GitHub repository.

**CI/CD Tools:** We will use GitHub Actions for CI/CD. It’s a great tool and will allow us to maintain that tight integration with the GitHub platform. I plan on writing workflows that will enable us to incorporate DevOps best practices like code scanning, linting, packaging and deploying. GitHub actions will allow us to dive right into CI/CD without having to worry about additional connections and authentication steps.

**Container Tools:** I think this will become more apparent when we have some code to work with and a more clear direction. If possible, I’d like to be able to package up our application using Docker and Docker Compose. This would allow us to have a straightforward deployment process. Also, we would be able to deploy this container in any environment. We haven’t decided the platform we will use to host the application yet (AWS, locally, other cloud providers, etc.), so having a containerized application will allow us to run it from anywhere.

**SAST/DAST Tools:** In the GitHub Actions marketplace there are free-to-use SAST and DAST scanning composite actions. We can incorporate these into our workflows to scan our code statically and dynamically. The SAST scan would produce a report before our application is packaged. The DAST scan would run after our application is packaged and deployed. This would run attacks on our website to see if there are any exploitable vulnerabilities. If we do incorporate these tools, they will likely be implemented closer to the end of our project.

**Other Tools:** As we begin coding, we may choose to utilize other tools. This will serve as a placeholder.

* 1. **Code Commit Guideline and Git Branching Strategy**  
     Our team will use the GitHub Flow branching strategy. This is a simple yet effective strategy, especially for smaller teams. Our main branch will be the only branch where we deploy from. Ideally, no one should commit directly to the main branch. I have tried to set up protection rules on the main branch, but unfortunately these rules cannot be applied unless I set the repository visibility to public, or the organization has a business subscription. Team members will create a new branch every time they begin work on a new issue. This branch can be called something like “[feature/bug/hotfix]/[issue number or general name]”. This will allow us to determine the nature of the branch, what it is being used for, and what GitHub Issue number it corresponds to. This will also allow us to automate the movement of issues on the GitHub Project board from column to column. After a team member has finished the work on their branch, they will either update their current branch with changes from the main branch (if there are any), or create a pull request against the main branch. The created pull request will generate a template for the team member to fill out to give additional information. Another team member will review the template, code changes, and approve the PR. The initial team member will then merge the code into main, and delete the branch. I’m hoping to set up the GitHub Project in a way such that closing out a PR will move the ticket from “in-progress” to “done”. An additional note: I’d like to enforce the requirement of approvals before a PR can be merged, but our current settings don’t allow us to do this. Also, every member has admin access to the repo so they can bypass these settings anyways.

## Deployment Plan if applicable

We will embrace a DevOps flow such that any time a branch is merged into main, our GitHub Actions workflow will scan the code, build, package, and deploy it to our environment of choice. This will allow us to constantly make changes to our web application so that bug fixes, breaking changes, or new features can be implemented in a timely manner.

# Quality Assurance Plan

## Metrics

| **Metric Name** | **Description** |
| --- | --- |
| Number of Lines of Code | A measure of complexity |
| Number of Code Files | A measure of complexity |
| Defect Rate | Number of defects found per 1000 lines of code (KLOC) |
| Number of test cases | A measure of the extent of testing |
| Test Case Pass Rate | This is the number of test cases passed divided by the number of total test cases written. |

* 1. Coding Standard

For Python we will follow PEP 8 (van Rossum). For SQL we will follow the Microsoft T-SQL Syntax Conventions (“Transact-SQL Syntax Conventions (Transact-SQL) - SQL Server”).

## Code Review Process

Each developer submitting code should review and unit test their own code. The following checklist should be completed for each pull request:

*Description*

*Please include a summary of the changes and the related issue. Please also include relevant motivation and context. List any dependencies that are required for this change.*

*Type of change*

*Please delete options that are not relevant.*

*<checkbox> Bug fix (non-breaking change which fixes an issue)*

*<checkbox> New feature (non-breaking change which adds functionality)*

*<checkbox> Breaking change (fix or feature that would cause existing functionality to not work as expected)*

*<checkbox> This change requires a documentation update*

*How Has This Been Tested?*

*Please describe the tests that you ran to verify your changes. Provide instructions so we can reproduce. Please also list any relevant details for your test configuration*

*Checklist:*

*<checkbox> My code follows the style guidelines of this project*

*<checkbox> I have performed a self-review of my code*

*<checkbox> I have commented my code, particularly in hard-to-understand areas*

*<checkbox> I have made corresponding changes to the documentation*

*<checkbox> My changes generate no new warnings*

*<checkbox> I have added tests that prove my fix is effective or that my feature works*

*<checkbox> New and existing unit tests pass locally with my changes*

The QA manager will also review all code to create integration testing and confirm the steps in the checklist have been completed.

## Testing

We will use a mix of automated and manual testing. In general, automated testing is preferred. But given the short timeframe for this project, there may be some cases where manual testing is more time-efficient.

Unit testing will be primarily the responsibility of the developer. In general, we should follow test-driven design where unit tests are written before production code and production code is written to fulfill the unit test.

The QA leader is responsible for reviewing code to design and implement integration tests. The QA leader will also monitor checklist completion and request remediation where needed.

## Defect Management

We will track defects in GitHub issues. Any team member can report a bug as it is found. The bug report should specifically explain what they are seeing and how it is different from the requirements. The report is then assigned to the code author to address. The code author should either apply a fix or explain why they believe the original code is, in fact, aligned with the requirements. The bug reporter should confirm that the fix or explanation meets their expectations and close the issue.

The QA manager will review all bugs to

# References

“Transact-SQL Syntax Conventions (Transact-SQL) - SQL Server.” *Microsoft Docs*, 30 November 2021, https://docs.microsoft.com/en-us/sql/t-sql/language-elements/transact-sql-syntax-conventions-transact-sql?view=sql-server-ver15. Accessed 15 May 2022.

van Rossum, Guido. “PEP 8 – Style Guide for Python Code.” *Python Enhancement Proposals*, 11 May 2022, https://peps.python.org/pep-0008/. Accessed 15 May 2022.

# Glossary

KLOC: Thousands of lines of code

PR: Pull Request

SQL: Structured Query Language

QA: Quality Assurance