TRIBHUVAN UNIVERSITY

Institute of Science and Technology

A Project Report

On

"Anime Recommendation System using Word2Vec"

Submitted to:

Department of Computer Science and Information Technology

Amrit Campus

In partial fulfillment of the requirements for the Bachelor's Degree in Information Technology

Submitted By:

Madhu Aryal (206/077)

Prabin Raj Amatya (207/077)

Prashansa Rai (208/077)

Under the Supervision of

Chhetra Bahadur Chhetri

2081

Student’s Declaration

We hereby declare that project report entitled “**Anime Recommendation System using Word2Vec**” submitted in the partial fulfilment of the requirement for Bachelor’s Degree in Information Technology of **Tribhuvan University** is our original work and not submitted for the award of any other degree diploma, fellowship, or any other similar title or prize.

………………………… Madhu Aryal

………………………… Prabin Raj Amatya

………………………… Prashansa Rai

Supervisor’s Recommendation

I hereby recommend that this project, prepared under my supervision entitled “**Anime recommendation system using Word2Vec**", a platform that suggests similar anime to the users on the basis of their preferences including their favorite anime and staff, using collaborative filtering for user-based recommendations in partial fulfilment of the requirements for the degree of Bachelor in Information Technology is processed for the evaluation.

……………………………….

Mr. Chhetra Bahadur Chhetri

Project Supervisor

Amrit Campus

Thamel, Kathmandu

Certificate

This is to certify that the project entitled **Anime Recommendation System using Word2Vec** prepared and submitted by **Madhu Aryal, Prabin Raj Amatya** and **Prashansa Rai** has been examined by us and is accepted for the award of the degree of Bachelor in Information Technology awarded by **Tribhuvan University**.

External Examiner	……………..…..	……………..
Asst. Prof. Dhirendra Kumar Yadav Project Coordinator Department of CSIT	…..…..……...	……………..
Mr. Chhetra Bahadur Chhetri Supervisor Lecturer Department of CSIT	………………	……………

Acknowledgement

The successful completion of our final year project would not have been possible without the invaluable guidance and encouragement we received. We are deeply grateful to our project supervisor, **Mr. Chhetra Bahadur Chhetri**, for his unwavering support and insightful advice throughout the development process. Our sincere appreciation also goes to **Amrit Campus** for providing us with an excellent platform to work on and refine our project.

Under the mentorship of **Mr. Chhetra Bahadur Chhetri** and the **Amrit Campus** team, we have gained profound knowledge of technologies and algorithm essential for building this system. This journey has strengthened our understanding of complex systems and has prepared us to tackle real-world challenges in this field.

We would also like to extend our gratitude to the ASCOL team for their valuable feedback, approval, and continuous support during this learning experience. Additionally, we appreciate the contributions of online communities, friends, and family members, whose assistance played a crucial role in shaping and refining our application.

In conclusion, this project has been a collaborative effort, and we are truly thankful to everyone who has contributed to its success.

Madhu Aryal (T.U. Exam Roll No: 206/077)

Prabin Raj Amatya (T.U. Exam Roll No: 207/077)

Prashansa Rai (T.U. Exam Roll No: 208/077)

Abstract

The Anime Recommendation System using Word2Vec aims to help users easily find anime that matches their interests. With thousands of anime available, it can be difficult to decide what to watch next. This system provides personalized recommendations based on user preferences including their favorite anime and staff. Users can create an account, browse anime, add their favorite shows, and receive tailored suggestions. The system also allows users to view anime details along with staff details. Additionally, administrators can manage the anime database by adding new anime details to keep the recommendations relevant. By using a recommendation engine, this platform improves the user experience by reducing the time spent searching for new anime and making discovery more enjoyable. The goal of this project is to create a simple and effective way for anime fans to explore and find anime they will love.

Keywords: ***Anime recommendation, personalized suggestions, user preferences, anime database, Word2Vec*.**

Table of Contents

List of Abbreviations

API: Application Programming Interface

BERT: Bidirectional Encoder Representations from Transformers

CBOW: Continuous Bag of Words

CSS: Cascading Style Sheet

HTML: Hypertext Markup Language

IDE: Integrated Development Environment

JS: JavaScript

JSON: JavaScript Object Notation

MAL: My Anime List

ML: Machine Learning

MUI: Material User Interface

PostgreSQL: Postgre Structured Query Language

UI: User Interface

Word2Vec: Word to Vector

List of Figures

List of Tables

Chapter 1: Introduction

1.1. Introduction

As more than 200 pieces of animation works broadcast annually in Japan in recent years, animation market scale in a broad sense including contents such as related goods has reached approximately 1.8 trillion yen. It has become more difficult for users to find out anime corresponding to their own preference from among huge number of animation works as well as to recognize the whole picture of contents related to them. [1] Thus, the Anime Recommendation System is designed to assist users in discovering anime that match their preferences. Recommender systems look at what people in a group like and then they suggest items that you might also enjoy or find useful. [2] With an extensive library of anime available, choosing what to watch can be a challenging task. This system streamlines the selection process by offering personalized recommendations based on user interests and favorite picks. Users can browse different anime, mark favorites, and receive tailored suggestions. Additionally, the system includes a search feature that allows filtering by preferences. Administrators have the ability to update the database by adding or removing anime, ensuring that recommendations remain relevant. The primary objective of this system is to enhance user experience by making anime exploration easier and more customized.

1.2. Problem Statement

Recommender systems are used by websites like Amazon and Netflix to help customers find products or shows they might like. These systems make shopping and watching more fun, keeping people interested longer. However, when it comes to anime, websites websites like MyAnimeList and Anime-Planet recommend anime mostly based on what other users have watched, but since these systems rely on watch history, they face issues like the Cold Start Problem, where new users with no watch history get poor recommendations and Popularity where lesser-known anime don't get recommended as often**.** Instead of just using watch history, the system could suggest anime based on user searches for a specific anime and what users have added to their favorites, including their favorite anime and staff members like directors. This would make recommendations more personalized and help users discover anime that match their specific interests, even if they are not widely known.

The purpose of the project is to create a recommendeation system that recommend anime to the user according to their preferences.

1.3. Objectives

The main objective of our project is:

- To personalize anime recommendations based on user preferences.

1.4. Scope and Limitation

Scope:

- To develop a user-friendly interface to the users.
- To provide personalized anime recommendations to help users easily discover and watch anime.

Limitation:

- The system is only available on the website, not as a mobile app.
- Users need an internet connection and a supported device to use the system.

1.5. Development Methodology

For this project, we are adopting the Waterfall Model, which structures the development process into a series of clearly defined, sequential phases. Each phase has its own objectives and is completed before moving on to the next. This method was selected because the requirements are already well-defined, and it’s more efficient for our team to fully complete one phase before proceeding to next. The five phases in this model are:

- Requirements Collection: Here, potential requirements are gathered, analyzed, and documented into functional specifications.
- Analysis: The system specifications are converted into a business model that guides development and supports a feasibility study.
- Design: The business model is translated into both logical and physical system designs.
- Implementation: In this phase, the source code is written based on the previously defined models, logic, and requirements.

- Testing: The system undergoes thorough testing to identify and fix any errors, ensuring that all aspects of the system work as expected.

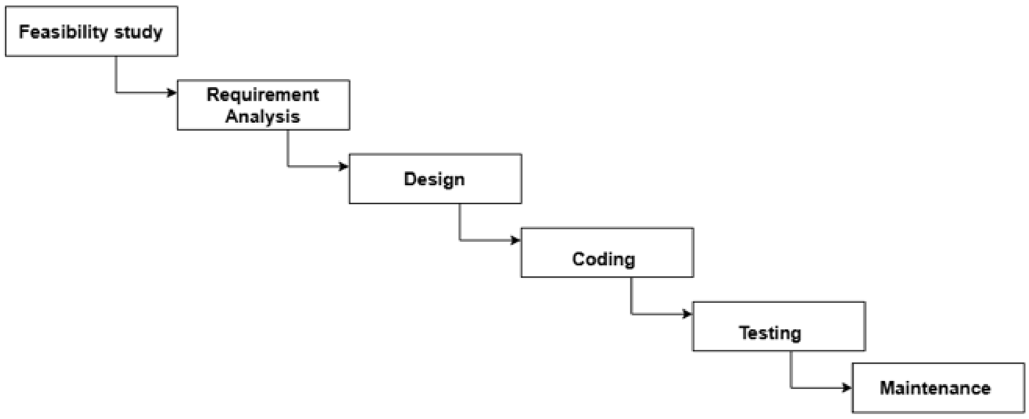


Figure 1.1 Waterfall Methodology

1.6. Report Organization

In Chapter 1: We introduce the motivation behind building the system, outlining the problem being addressed, the objectives of the project, its features, and its scope and limitations.

In Chapter 2: This chapter presents a review of existing literature, including relevant journals, articles, and previous research. We also discuss the background related to the topic.

In Chapter 3: We explore the system analysis phase, which covers system, requirements, and feasibility analysis.

In Chapter 4: Here, we delve into the system design and algorithms, presenting key diagrams such as the class diagram, object diagram, and activity diagram.

In Chapter 5: This chapter details the implementation and testing procedures, as well as the tools used throughout the development process.

In Chapter 6: Finally, we summarize the conclusions of the project and discuss potential future improvements.

Chapter 2: Background Study and Literature Review

2.1. Background Study

Anime is a global entertainment phenomenon with millions of fans worldwide. Anime talks about deep themes, similar to literature and art films hence making it more than just entertainment [3]. The rise of streaming platforms like Crunchyroll and Netflix has expanded people's access to anime content. However, due to the variety of anime, viewers often have difficulty finding anime that fits their tastes. This has led to the development of anime recommendations that aim to showcase better content. These systems analyze user data (like viewing history and ratings) to recommend relevant brands. Recommendation algorithms often use clustering, content filtering, or a combination of these techniques to provide personalized recommendations. Integration matches users with similar tastes, while filtering-based content offers recommendations based on specific characteristics. The hybrid approach offers two ways to improve the accuracy of recommendations. Recent advances in machine learning, especially deep learning, have improved the performance of these systems. These technologies allow for better analysis of user behavior and preferences. As anime becomes more popular around the world, the need for a consistent approval process will become increasingly important. As the recommendation algorithm continues to improve, viewers will find more entertainment and personalization.

2.2. Literature Review

Before the arrival of streaming services, anime enthusiasts had limited options for discovering new shows. They mostly relied on TV broadcasts, DVDs, or recommendations from friends and magazines. This process often required considerable effort and time. With the rise of digital platforms, accessing anime has become more convenient, but the sheer number of available titles can make selection overwhelming. To address this challenge, recommendation systems have been developed to assist users in finding anime that aligns with their interests. Various platforms use different methods to refine their recommendation processes.

MyAnimeList (MAL)

MyAnimeList allows users to track their watched anime, rate series, and receive personalized recommendations. The platform generates suggestions based on user ratings, reviews, and watch history while also showcasing trending and seasonal anime.

AniList

AniList focuses on customization, analyzing user preferences, interactions, and genre interests to generate tailored recommendations. It also provides discussion forums and curated lists for better user engagement.

Kitsu

Kitsu categorizes anime based on themes and genres, enabling users to explore content that aligns with their viewing patterns. The system observes user activity and suggests relevant titles accordingly.

Netflix

Netflix employs machine learning algorithms to study watch history, user behavior, and preferences to offer personalized anime recommendations.

Crunchyroll

Crunchyroll enhances recommendations using content-based filtering and user ratings. By analyzing user interactions and show attributes, it delivers suggestions that match individual preferences.

These platforms utilize various recommendation techniques, such as collaborative filtering, content-based filtering, and hybrid models, to enhance accuracy. Despite advancements, challenges such as improving precision and catering to new users remain. Ongoing research aims to refine these systems for a better user experience and more relevant recommendations.

Chapter 3: System Analysis

3.1. System Analysis

In this project, we gathered and analyzed information to identify problems and suggest improvements. We studied the system in detail to understand its structure, challenges, and key factors. Through system analysis, we identified important variables, examined different components, and worked towards finding the best possible solution. We also conducted feasibility studies to assess the practicality of our approach. These studies helped us outline system activities, evaluate different strategies, and determine the most effective path forward. Additionally, we considered potential challenges that might arise during development and planned solutions to address them. Our goal was to ensure a smooth and efficient system with minimal risks and maximum effectiveness.

3.1.1. Requirement Analysis

After studying the current system and understanding its limitations, we will now focus on identifying the specific requirements for the new website. It's important that the website meets the needs and interests of the people who will be using it. We'll gather information on what features are most important to users and how we can make the website user-friendly and functional. The analysis phase plays a key role because it helps us plan the project carefully before we start developing the website. By analyzing these requirements, we can make sure the website solves the problems of the existing system, offers useful features, and provides a great experience for users. This careful planning will guide the development process and help avoid any issues later on.

Software Specification:

- Operating System: Windows 10/11(or above)
- Front End: HTML, CSS, JavaScript
- Back End: Python, PostgreSQL
- IDE: Visual Studio Code
- Framework: Django, React

Hardware Requirements:

- Minimum: 8GB RAM
- Processor: Intel Pentium 4(or above)
- Storage: 100GB Hard Disk/ SSD(or above)

Now, requirements can be classified into functional and non-functional.

3.1.1.1. Functional Requirements:

The functional requirements can be:

- Manage anime and staff data
- Manage user data
- Anime recommendations
- Login and Register
- Search and filter

Use Case Diagram:

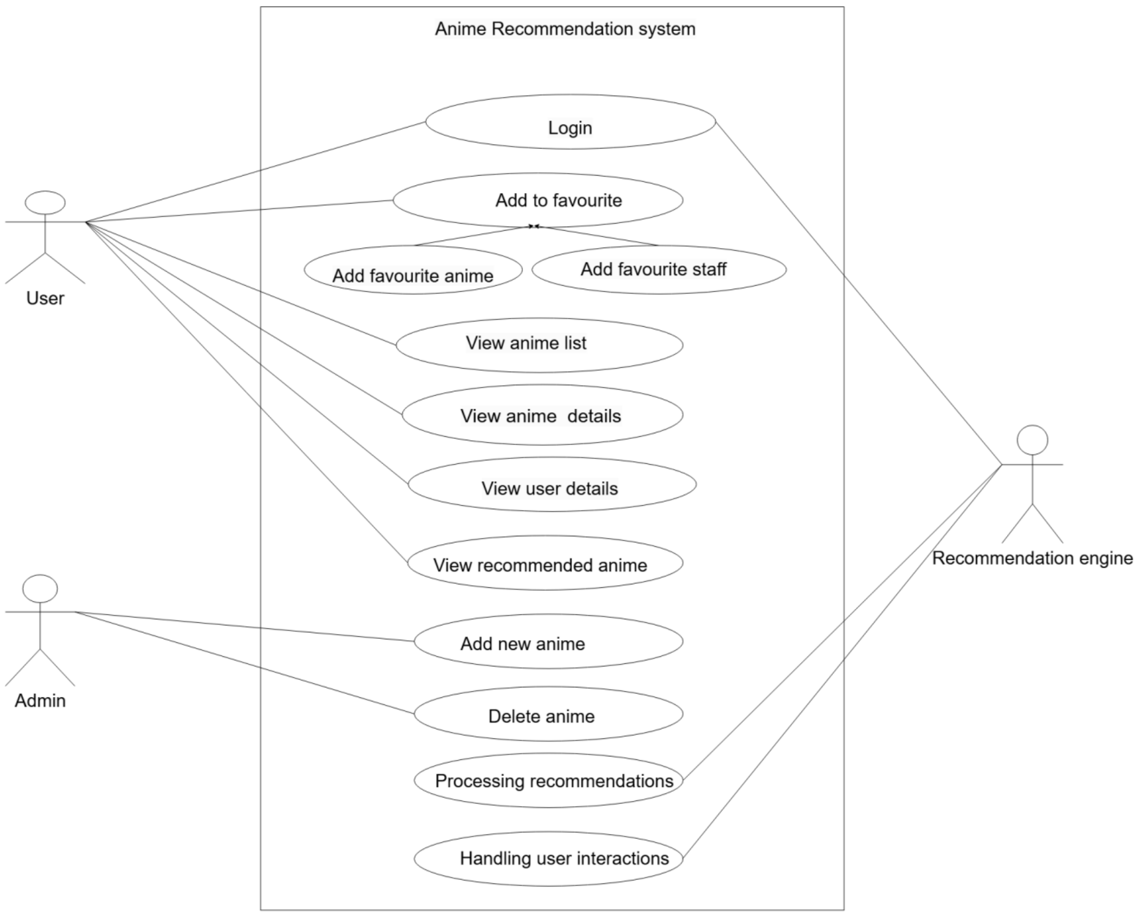


Figure 3.1: Use Case Diagram

In this diagram we have three actors named as ‘admin’, ‘user’ and ‘recommendation engine’. There are different use cases namely ‘register’, ‘login’, ‘add to favourite’, ‘view anime list’, ‘view anime detail’, ‘add new anime’, ‘delete anime’, ‘processing recommendations’, ‘handling user interactions’, ‘view recommended anime’, ‘manage users’, ‘log out’. All admin, user and recommendation engine have certain relationships with the use cases. User can register, login, view anime list, view anime detail, make recommendation and logout. While admin can login, register, add new anime, update anime, delete anime, manage users, and logout. And recommendation engine can login, process recommendations, handle user interactions. Here, we mainly discuss the relationship between actors and the use cases.

3.1.1.2. Non-Functional Requirements

Some of the non-functional requirements for the project are:

- Speed: One of the non-functional requirements is speed. System must run quickly and request/response from the server must be fast.
- Usability: System must be easy for user to use. So, we have to care about usability perspective.
- Standards: Coding standards must be strictly followed while system development.
- Other standards should also be taken in consideration.
- Scalability: We also have to think about scalability perspectives i.e., how we can increase the sale and scale properly, while developing the system.

3.1.2. Feasibility Study

When evaluating the viability of the project, we must consider its feasibility from several angles.

3.1.2.1. Technical Feasibility:

The project can be developed using personal laptops, and the software technologies chosen for the development are free and open-source. Therefore, the technical aspects of the project are feasible.

3.1.2.2 Operational Feasibility:

The system is designed to be intuitive and user-friendly, requiring no special training to operate. Users can easily search for the anime they wish to watch due to the simple interface. Both admins and users can access the platform without any issues, making the project operationally feasible.

3.1.2.3 Economic Feasibility:

The required software and hardware are affordable within the project's budget.
Additionally, since internet access is widely available today, users will be able to visit the site with ease, making it economically viable.

3.1.2.3 Schedule Feasibility:

The project can be completed within the given time frame. The tasks are well-planned and there is enough time allocated for each phase of development. The timeline is realistic and achievable, ensuring that the project will be delivered on time.

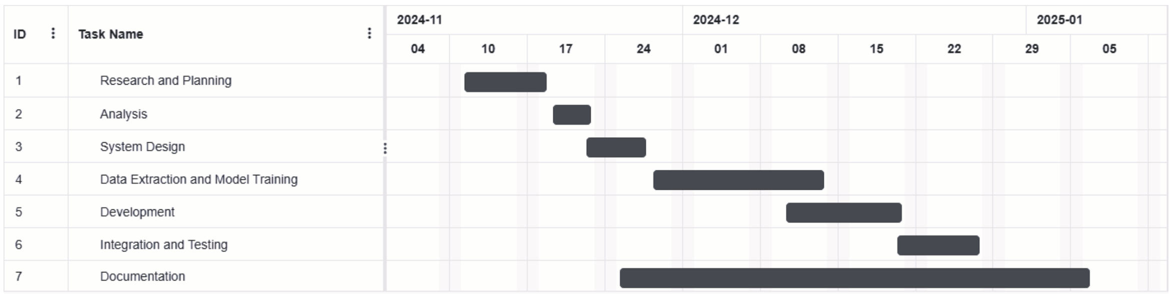


Figure 3.2: Gantt Chart

3.1.3 Analysis

Since we are using object-oriented approach for developing the system, we have to analyze Object modelling using Class and Object Diagrams, Dynamic modelling using State and Sequence Diagrams and Process modelling using Activity Diagrams. This three modelling are mainly analyzed for developing a system so that we can get better software for the system. These modelling are discussed below as:

3.1.3.1. Object Modelling:

The modelling technique of object modelling is used in the creation and modelling of software. It describes the system's static structure. Here we are creating Class and Object diagram.

I. Class Diagram:

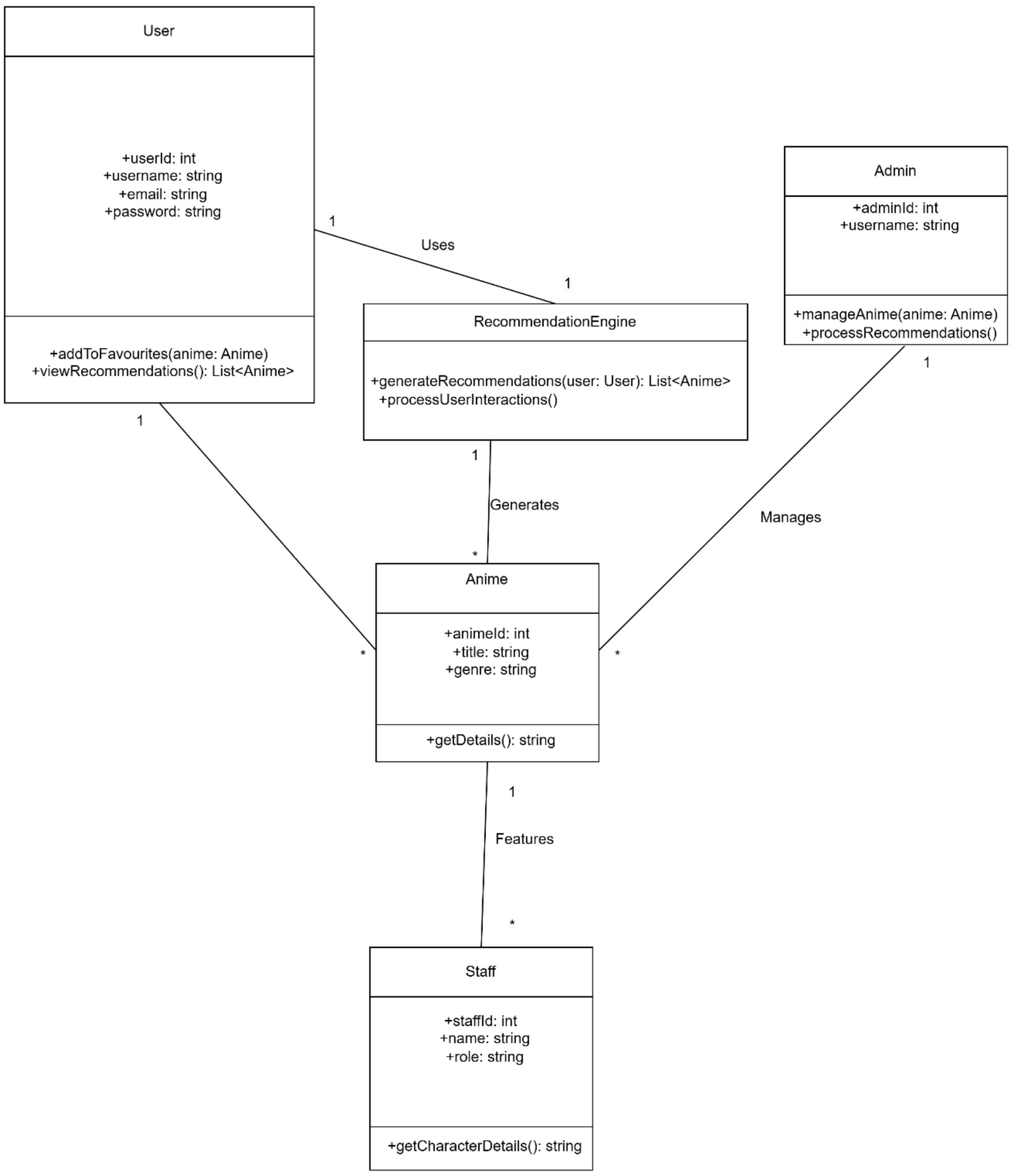


Figure 3.3: Class Diagram

II. Object Diagram:

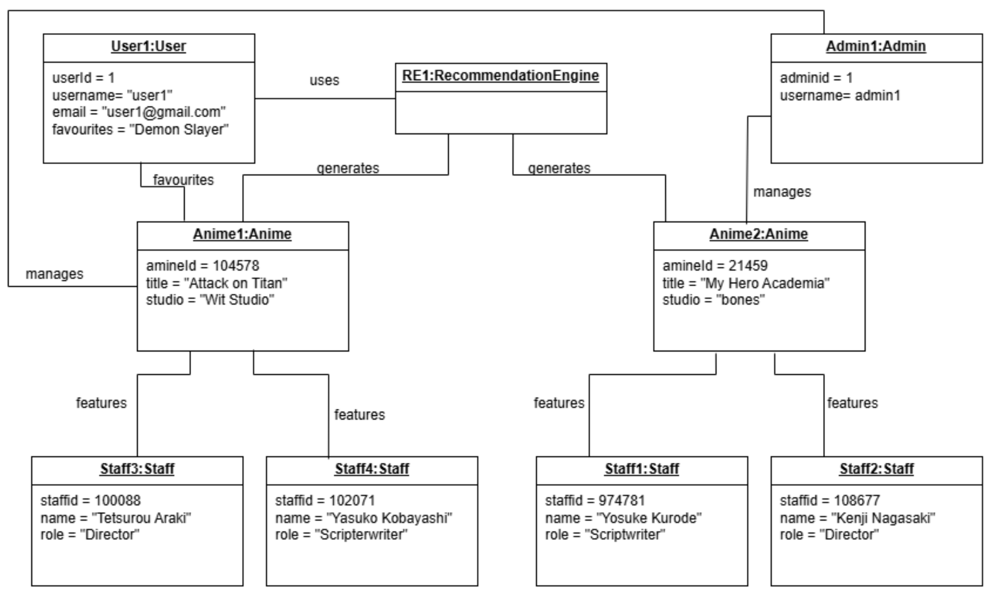


Figure 3.4: Object Diagram

3.1.3.2 Dynamic Modelling:

Dynamic modeling is about showing how things happen over time in a system and the order in which they occur. It helps us understand how the system changes and works step by step. For dynamic modeling, we have created two types of diagrams: state diagrams and sequence diagrams.

I. State Diagram:

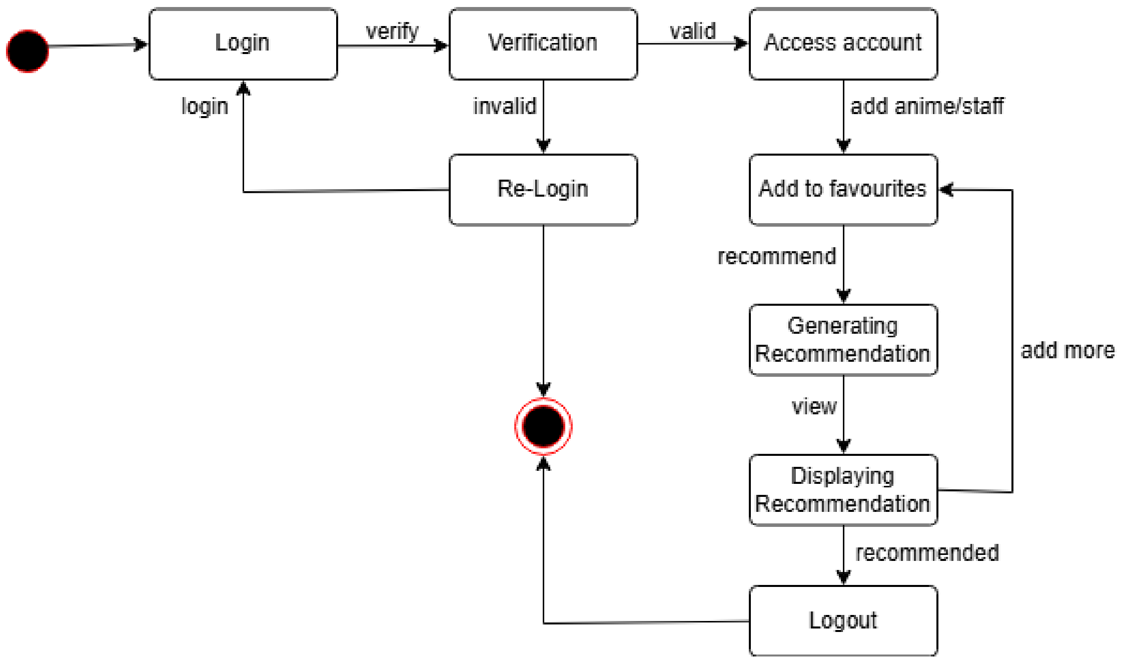


Figure 3.5: State Diagram

II. Sequence Diagram:

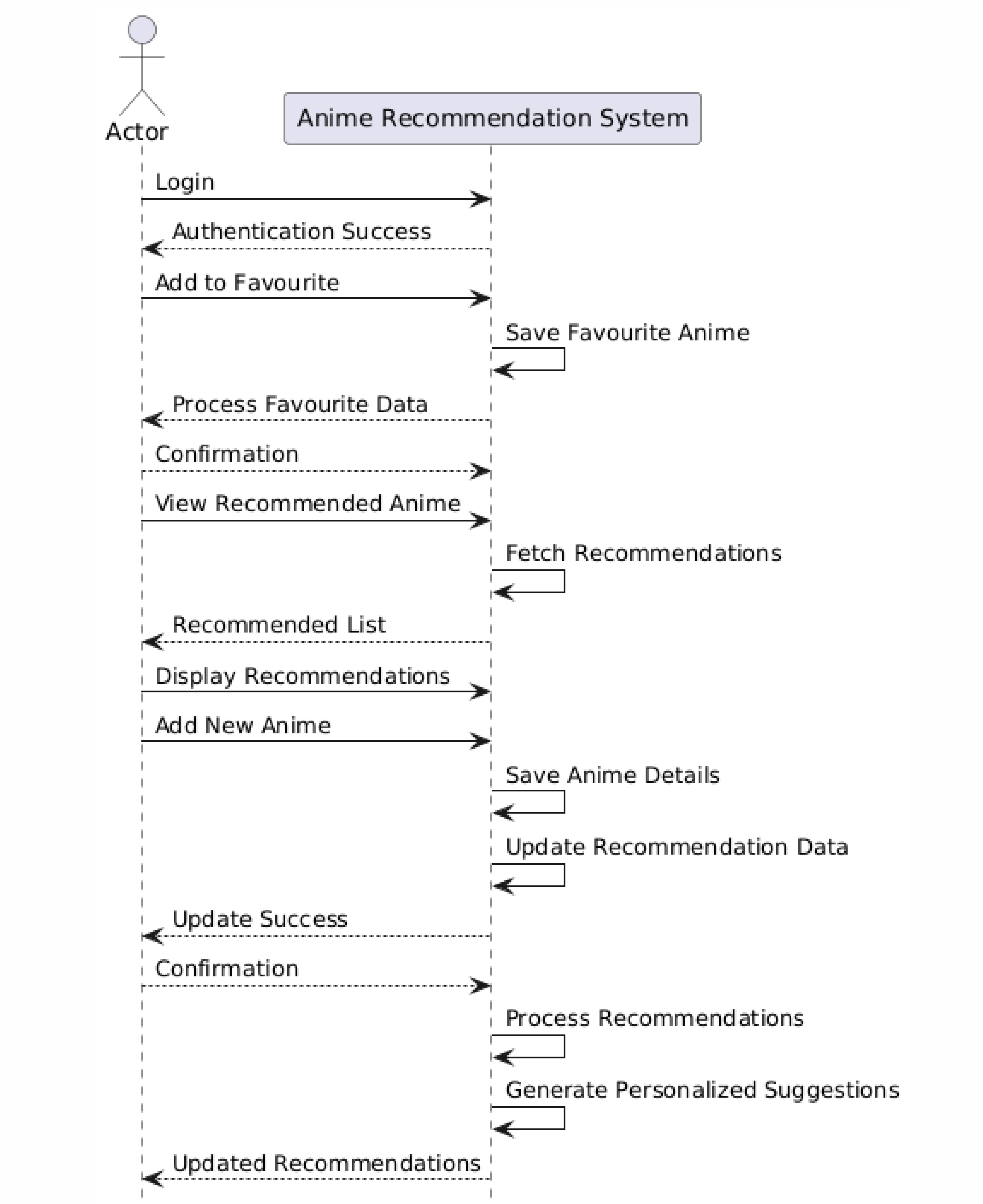


Figure 3.6: Sequence Diagram

3.1.3.3. Process Modelling:

Here, we talk about a task in a project and how it moves through an activity diagram. The activity diagram shows the overall flow of tasks in the project.

Activity Diagram:

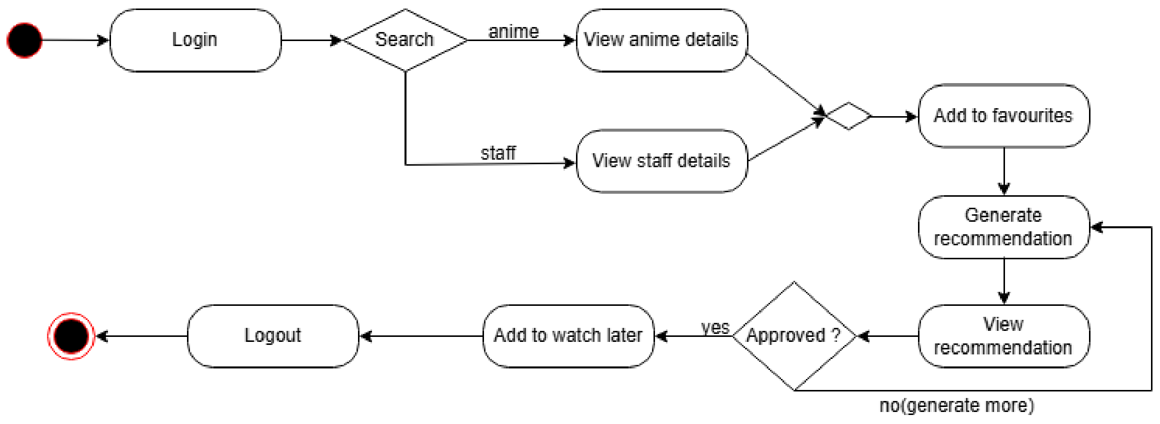


Figure 3.7: Activity Diagram

Chapter 4: System Design

4.1. System Design

In this step, the overall structure and functionality of a system has been defined. An object-oriented approach in both design and analysis has been used for this project.

4.1.1. Architecture Design

The architecture design for the project is 3-tier architecture (client server architecture).

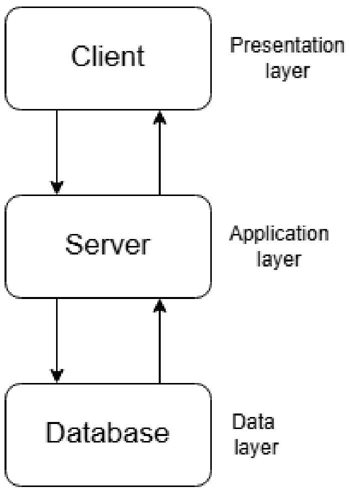


Figure 4.1: 3-Tier Architecture

4.1.1.1. Presentation Layer:

The client includes the UI and display of the system. It sends requests to the server and displays responses to the user. It consists of:

- Homepage
- User login/registration form
- User profile
- Admin login form
- Admin dashboard
- Search engine
- Recommendation engine

4.1.1.2. Application Layer:

The server is the middle layer between the client and database. It processes requests from the client, applies business logic and retrieves or updates data from the database.

4.1.1.3. Data Layer:

Database is responsible for storing and managing structured data. We have used Postgres as database. Our database is currently hosted in Aiven.

Database Tables:

The database tables specified in the project are:

AnimeData: Stores anime data

StaffData: Stores staff data

User: Stores user credentials

Favourite: Stores the user's favorite anime/staff

Flowchart:

The flowchart of a project which represents the logic of the system in pictorial form is designed.

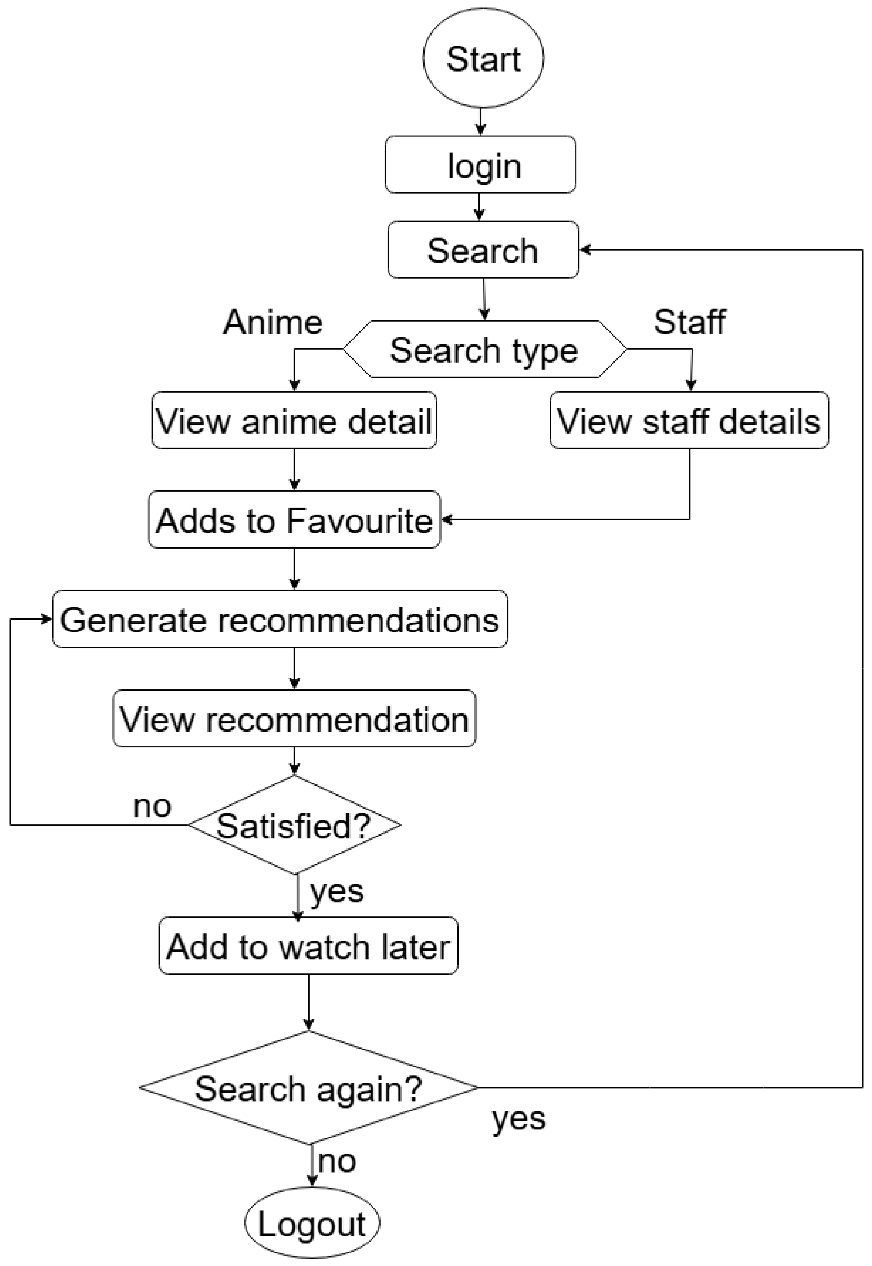


Figure 4.2: Flowchart

4.1.2 Component Design:

The working of the individual component in the system is defined. Following is the Component Diagram to show the main modules and how they interact in the system:

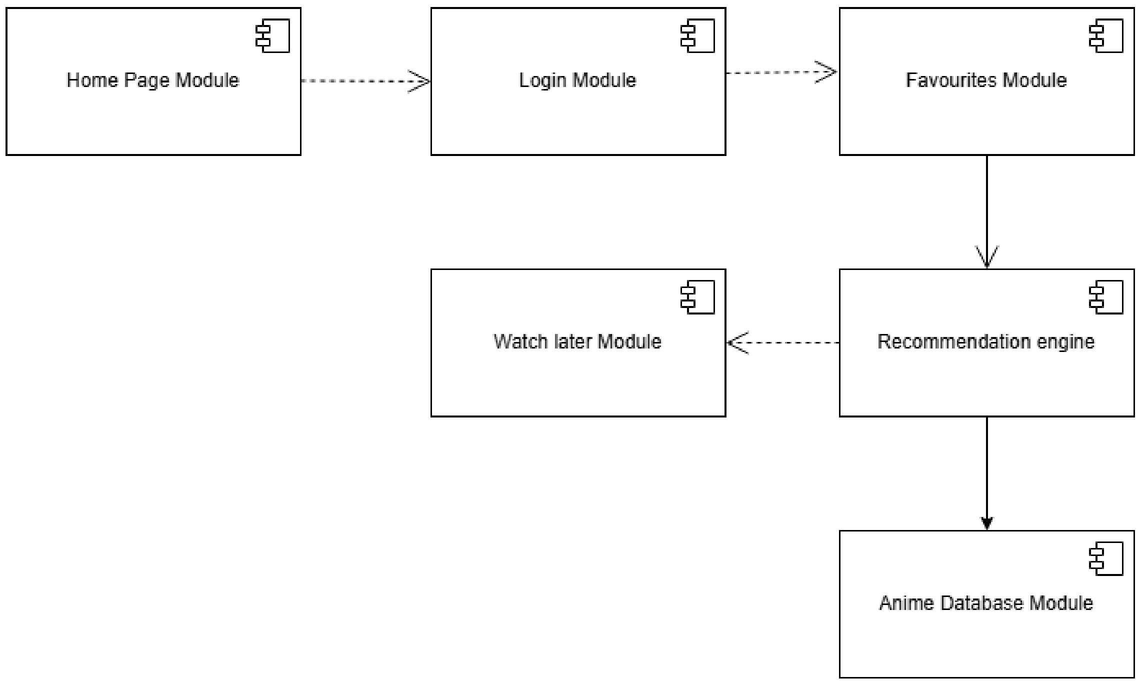


Figure 4.3: Component Diagram

4.1.3 Deployment Design:

The system's infrastructure, environment and deployment strategies are defined. Following is the Deployment Diagram to display the system's physical architecture.

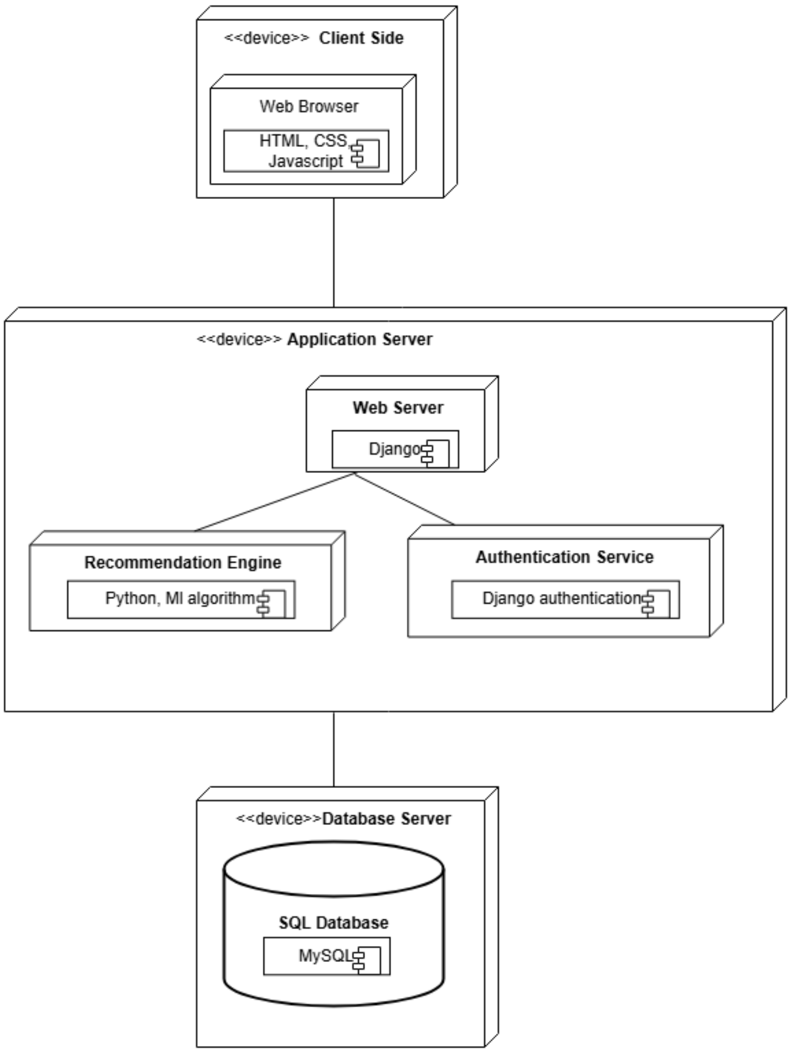


Figure 4.4: Deployment Diagram

4.2. Algorithm Details

4.2.1. Overview of the Model

Our anime recommendation model is trained to provide personalized anime suggestions based on user preferences as well as anime suggestions based on general trends. Our system utilizes a Word2Vec-inspired embedding model, where anime titles are represented as vectors in a high-dimensional space. Websites like IMDB allow users to keep track of their favorite movie in favorites lists.. Our model is trained to capture relationships between anime based on a user's favorite lists which we gathered from

IMDB equivalent of anime such as MAL and Anilist ensuring that anime that occur frequently in favorite lists appear together in vector space.

To enhance recommendation quality, we trained multiple versions of the model, varying in epoch count, embedding dimensions, and input structure (target-only vectors vs. target + context vectors). These variations were analyzed and compared to optimize accuracy and efficiency.

We use word2vec over more powerful models like BERT because training BERT requires huge amount of data and is very computationally expensive. Comparatively word2vec models are much more lightweight and scalable. [4]

The recommendation process involves converting a user's liked anime into vector representations and retrieving the closest matches using cosine similarity

The type of vector used varies depending on the type of request:

- **Independent Anime Search**: This feature is seen when searching for a particular anime. Recommendations are generated using the searched anime as input, so that similar anime are generated based on general trends.
- **Personalized Recommendations**: This feature is used when recommending based on a user's favorite list. This approach considers the anime in user’s favorites list, which leads to more personalized and relevant recommendations.

4.2.1.1. Understanding Word2Vec

Word2Vec is a neural network-based model used to generate dense vector representations of words based on their surrounding words in a given corpus. It captures semantic relationships between words by positioning similar words closer together in a high-dimensional vector space. Word2Vec models goes beyond capturing just the similarity between words but can also capture an capture semantic relationships between words through simple algebraic operations. For example vector ("King") - vector("Man") + vector("Woman") will result in vector("Queen"). [5]

The two primary training approaches when training a Word2Vec model are:

- **Continuous Bag of Words (CBOW):** We train the model by making it predict a target word based on its surrounding words.

- **Skip-gram:** We train the model by making it predicts surrounding words given a target word.

The idea behind Word2Vec is that words appearing in similar contexts will have similar vector representations, which allow operations like cosine similarity to measure relationships between words.

For our model we use the **Skip-gram** model, which learns to associate words (or in our case, anime) with their surrounding context by training on large dataset of user favorites list. This will be explained more clearly in upcoming topics

4.2.1.2. Skip-gram Model Training Process

In Skip-gram model the model aims to predict the surrounding words given a target word in a sentence. In other words, for a target word w_t, the model learns to predict the context words w_c that occur around it in a fixed-size window.

For example, in the sentence: **“**Canine bites human.”

If we consider “bites” as the target word, the surrounding words, such as “Canine”, “human” are the context words. The Skip-gram model learns to predict the context words from the target word.

In the Skip-gram model:

- Positive samples are context words that appear near a given target word in a sentence. These are words that occur in the same context window.
- Negative samples are randomly chosen words from the vocabulary that do not appear in the context of the target word.

Let’s break down how the training works using an example:

Given the sentence: “The quick brown fox jumps over the lazy dog.”

We could take the word “fox” as the target word and define a context window of size 2, meaning we look at the two words before and two words after the target word.

- **Positive samples**: In this case, the context words for “fox” would be “quick”**,** “brown”**,** “jumps”**,** “over”

The model will learn to predict these context words from the target word “fox.”

- **Negative samples**: These are words that do not appear in the context window. For instance, we may randomly select words like “dog”, “the” or “lazy” as negative samples.

Mathematical Representation

We represent the target word (w_t), and context word (w_c) in fixed size vectors which are initialized at random.

- **Target embedding vector** w_t : Represents the target word.
- **Context embedding vector** w_c : Represents the context word.

The Skip-gram model aims to compute the probability of observing a context word w_c given a target word w_t as:

$P(w_c \mid w_t)=P(w_c) * P(w_t)$

Where:

- $P(w_c) * P(w_t)$ is the dot product of the target and context embedding vectors.
- Usually there is a denominator $P(w_t)$ which normalizes the probability. However we only need the raw value of the probability therefore normalization is not necessary.

Loss Function and Optimization

The Skip-gram model uses categorical cross-entropy loss to compare the predicted probability of the context word w_c with the actual probability context word. The loss function is calculated as:

$L = -\sum_{i}^{k} \mathbf{y}\mathrm{i} * \log(\mathbf{P}(\mathbf{wc}\mathrm{i} \mid \mathbf{wt}))$

Where:

- P(wc|wt) is the predicted probability for the context word w_c given the target word w_t
- L is the loss for the current training sample.
- k is the total number of samples
- y is the actual probability or the label of the context word. 1 for positive samples and 0 for negative samples

To minimize the loss and improve the embeddings, we use gradient descent. The derivatives of the loss function with respect to the target embeddings are used to update the embeddings:

$$\frac{dL}{dwt} = \frac{d\left(-\sum_{i}^{k} \mathbf{y}\mathrm{i}{*}\log(\mathbf{P}(\mathbf{wc}\mathrm{i} \mid \mathbf{wt}))\right)}{dwt}$$

4.1.2.3 Adapting skip gram model for anime recommendation

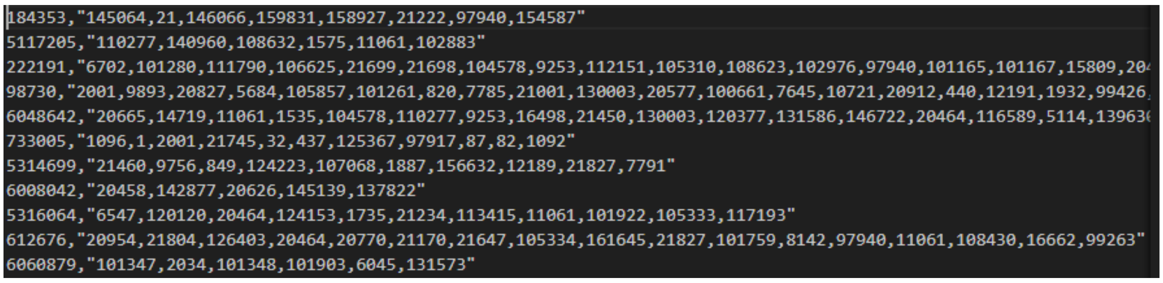
For our model, We can represent each anime as a word and each favorite list a a sentence[citation]. Each anime has two separate vector representations:

- **Target embeddings:** Represent the anime itself.
- **Context embeddings:** Represent the anime when it appears in a surrounding context.

Overview of Dataset

Our dataset looks as shown in table 4.1. The column in the left represents user Id and column in the right represents Ids of the anime that is in User's favorite list. The Id is unique to each anime. For example in figure 4.5 the id 145064 corresponds to the anime Jujutsu Kaisen Second Season. This means the Jujustsu Kaisen Second Season is in user 184353's favorite list. Here Jujustsu Kaisen Second Seasons Id 184353 acts as a word and the favorite list "145064,21,146066,159831,158927,21222,97940,154587" acts as sentence where each word in the sentence is separated by a comma

Table 4.1: Training Dataset



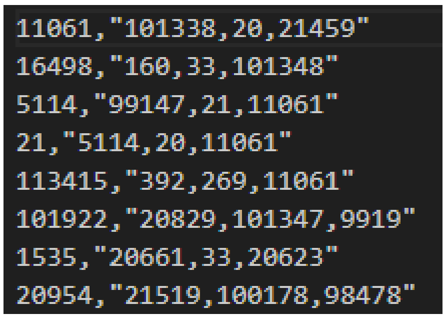
```
184353,"145064,21,146066,159831,158927,21222,97940,154587"
5117205,"110277,140960,108632,1575,11061,102883"
222191,"6702,101280,111790,106625,21699,21698,104578,9253,112151,105310,108623,102976,97940,101165,101167,15809,20
98730,"2001,9893,20827,5684,105857,101261,820,7785,21001,130003,20577,100661,7645,10721,20912,440,12191,1932,99426,
6048642,"20665,14719,11061,1535,104578,110277,9253,16498,21450,130003,120377,131586,146722,20464,116589,5114,13963
733005,"1096,1,2001,21745,32,437,125367,97917,87,82,1092"
5314699,"21460,9756,849,124223,107068,1887,156632,12189,21827,7791"
6008042,"20458,142877,20626,145139,137822"
5316064,"6547,120120,20464,124153,1735,21234,113415,11061,101922,105333,117193"
612676,"20954,21804,126403,20464,20770,21170,21647,105334,161645,21827,101759,8142,97940,11061,108430,16662,99263"
6060879,"101347,2034,101348,101903,6045,131573"
```

Testing Data:

Semantic Test

To evaluate the model's performance on semantic tasks, we collected additional data from Anilist.A popular anime database and community platform. Anilist provides recommendations of similar anime as well. We perform semantic tests by seeing how many anime that Anilist provides as similar is flagged as similar by our model. Dataset for semantic tests looks as in table 5.10. The column on left represents our selected anime and the column on right represents anime similar to our selected anime.

Table 4.2: Semantic Test Dataset

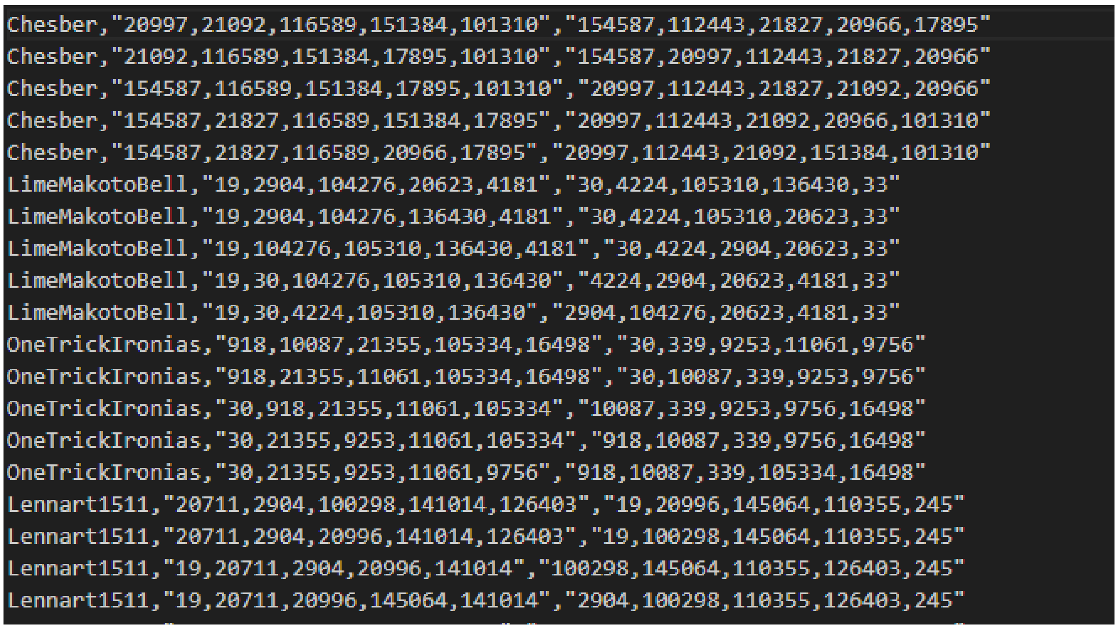


```
11061,"101338,20,21459"
16498,"160,33,101348"
5114,"99147,21,11061"
21,"5114,20,11061"
113415,"392,269,11061"
101922,"20829,101347,9919"
1535,"20661,33,20623"
20954,"21519,100178,98478"
```

Syntactic Test

For evaluating syntactic tasks, we collected additional data from MAL another popular anime database and community flatform. We collect user’s favorite lists just like for our training data. We collected test data from a different website so we get a very different dataset than our training data which is optimal for testing a model. We then create multiple combination of a user’s list so we can test the model’s behavior on different version of the same list. Dataset for syntactic tests looks as in table below. The column on the left represent our user and the column on the right represents user favorites.

Table 4.3: Syntactic test dataset



```
Chesber,"20997,21092,116589,151384,101310","154587,112443,21827,20966,17895"
Chesber,"21092,116589,151384,17895,101310","154587,20997,112443,21827,20966"
Chesber,"154587,116589,151384,17895,101310","20997,112443,21827,21092,20966"
Chesber,"154587,21827,116589,151384,17895","20997,112443,21092,20966,101310"
Chesber,"154587,21827,116589,20966,17895","20997,112443,21092,151384,101310"
LimeMakotoBell,"19,2904,104276,20623,4181","30,4224,105310,136430,33"
LimeMakotoBell,"19,2904,104276,136430,4181","30,4224,105310,20623,33"
LimeMakotoBell,"19,104276,105310,136430,4181","30,4224,2904,20623,33"
LimeMakotoBell,"19,30,104276,105310,136430","4224,2904,20623,4181,33"
LimeMakotoBell,"19,30,4224,105310,136430","2904,104276,20623,4181,33"
OneTrickIronias,"918,10087,21355,105334,16498","30,339,9253,11061,9756"
OneTrickIronias,"918,21355,11061,105334,16498","30,10087,339,9253,9756"
OneTrickIronias,"30,918,21355,11061,105334","10087,339,9253,9756,16498"
OneTrickIronias,"30,21355,9253,11061,105334","918,10087,339,9756,16498"
OneTrickIronias,"30,21355,9253,11061,9756","918,10087,339,105334,16498"
Lennart1511,"20711,2904,100298,141014,126403","19,20996,145064,110355,245"
Lennart1511,"20711,2904,20996,141014,126403","19,100298,145064,110355,245"
Lennart1511,"19,20711,2904,20996,141014","100298,145064,110355,126403,245"
Lennart1511,"19,20711,20996,145064,141014","2904,100298,110355,126403,245"
```

Data collection

The dataset for the anime recommendation system was collected from two primary sources: AniList and MyAnimeList(MAL)**,** which are both widely used anime tracking platforms. From these websites we collected the many user's favorite anime lists, which is used for training and testing our recommendation model.

Primary Dataset: AniList Favorites

For model training, we extracted user's favorite lists from AniList. Anilist provides data through its GraphQL API**.** A total of 236,000 user's favorite lists were pulled, allowing the model to learn meaningful relationships between anime titles based on real user preferences. The use of GraphQL enabled efficient querying, reducing redundant data retrieval and optimizing API calls.

Optimizing Data Collection with Multithreading

Since MyAnimeList does not provide an official API for bulk data extraction, we implemented multithreading to maximize the number of requests sent per second. This parallel processing approach reduced data collection time by making multiple simultaneous requests.

We collect data and then preprocess it before using it for training and testing the recommendation model. This dataset serves as the foundation for generating accurate and personalized anime recommendations, improving model robustness across different user preferences.

Generating Training Data

- Each user's favorite anime list is treated as a sequence (similar to a sentence in text processing).
- For each anime in the list, the model selects surrounding anime within a context window size **as** positive samples (anime that appear together). For our model we use window size of 5.
- For improving training efficiency we also generated negative samples, which are anime that are randomly selected that did not appear in the same list. These serve as counterexamples, which helps the model differentiate meaningful relationships from random occurrences. We generated 5 negative sample for each target word for our model.
-

Sampling Table for Anime Recommendation

In natural language processing, frequent words like "the" and "is" are down sampled so that frequently occurring words do not dominate training and less frequently words can be sampled more often. Similarly, in our anime dataset, highly popular anime titles may dominate training, while lesser known titles may not get sufficient representation. To counter this, we use a sampling distribution that ensures a more balanced sampling.

We adopt a power-based transformation on the frequency of each anime to generate a probability distribution that allows better representation across the dataset. The probability of selecting an anime for training is computed as:

$P(A_i) = f(A_i)^{0.5} / \sum f(A_j)^{0.5}$

Where:

- $P(A_i)$ is the probability of sampling anime .
- $f(A_i)$ is the frequency of anime in the dataset.
- $\sum f(A_j)^{0.5}$ is the sum of frequency to the power 0.5 of all anime in dataset
- The exponent reduces the dominance of highly frequent anime while still maintaining their presence.

- The denominator normalizes the probabilities so they sum to 1.

After Making these Adjustments word2vec skip gram model, works as a anime recommendation system.

4.2.1.4 Model Versions

Word2Vec is a highly effective model however wrong combination of hyperparameters can produce poor quality vectors [6]. Therefore we train multiple models with different hyperparamters. Following are the various versions of the model

- **50 Dimensions - Target Vector**: This model utilizes a 50-dimensional embedding focused solely on target vectors.
- **50 Dimensions - Target + Context Vector**: This version employs a 50-dimensional embedding with both target and context vectors.
- **100 Dimensions - Target Vector (Trained for 5 Epochs)**: This model features a 100-dimensional target vector and is trained for 5 epochs, providing a balance between complexity and training duration.
- **100 Dimensions - Target + Context Vector (Trained for 5 Epochs)**: Similar to the previous model but incorporating both target and context vectors for a more comprehensive analysis.
- **150 Dimensions - Target Vector**: This version employs a 150-dimensional embedding with only target vectors, aiming to capture more intricate relationships.
- **150 Dimensions - Target + Context Vector**: This model uses a 150-dimensional embedding that includes both target and context vectors, enhancing the syntactic analysis.
- **100 Dimensions - Target Vector (Trained for 3 Epochs):** A version with a 100-dimensional target vector, trained for 3 epochs, focusing on learning efficiency.
- **100 Dimensions - Target + Context Vector (Trained for 3 Epochs)**: This final model combines a 100-dimensional embedding with both target and context vectors, trained for 3 epochs.

By exploring these eight model versions, we aim to identify the optimal configuration that balances performance, generalization, and computational efficiency in our anime recommendation system

Chapter 5: Implementation and Testing

5.1. Implementation

We used the iterative waterfall development model, to develop an anime recommendation system. This model allowed us to refine each phase of the project, ensuring thorough documentation and review at each stage. Below are the details of the implementation of the system, outlining our approach to data collection, model development, web application creation, and testing.

5.1.1 Application Implementation

5.1.1.1. Technologies Used

Frontend Technologies:

i. **Languages:**

- **JavaScript:** JavaScript outside of react was used for having a fine grained control over the complex animations used in our website.
- **HTML (Hyper Text Markup Language):** Html was used to give a structure to our web page. To define the various components of our web application
- **CSS (Cascading Style Sheets):** CSS was used to change the appearance of the various html elements as per our needs and for animations along with javascript.

ii. **Framework:**

- **React:**

  React was used as a framework because it allowed us to create reusable UI components, manage the state of applications efficiently, communicate with the backend easily with libraries like react query.

iii. **UI Component Library:**

- **Material-UI (MUI):** MUI was used because it provides pre-built components which are responsive and easy to customize.

Backend Technologies:

i. **Framework:**

- **Django:**
  We use Django as a backend framework because it provides a robust structure for building web applications, it includes features like an ORM (Object-Relational Mapping), authentication, and admin interfaces.

ii. **Database:**

- **PostgreSQL:**
  PostgreSql is used because it provides powerful querying capabilities and ensures data integrity, making it suitable for applications that require reliable data storage and retrieval.

Code Editor:

- **Visual Studio Code:**
  A lightweight but powerful source code editor developed by Microsoft. It supports a wide range of programming languages and provides features such as debugging, syntax highlighting, and extensions for enhanced functionality. Visual Studio Code is favored for its user-friendly interface and versatility, making it suitable for both frontend and backend development.

5.1.1.2. Frontend Development

The frontend module of the anime recommendation system is designed to deliver an engaging user interface, facilitating seamless navigation throughout the application. Users can easily search for anime, view personalized recommendations, and manage their favorite titles with minimal effort.

The application is built using React components, and complex animations are implemented using JavaScript and CSS to enhance user experience. The key React components of our application include:

- **Cube:**
  This dynamic feature serves as the centerpiece of our recommendation system. The Cube is designed to resemble a Rubik's Cube, twisting and turning in random

patterns whenever the recommendation button is clicked. Once the animation concludes, each side of the Cube displays a unique anime recommendation. This visually captivating animation not only entertains users but also keeps them engaged and eager to discover new titles.

- **Navbar:**
  The Navbar component provides easy access to different sections of the website. Users can navigate easily between the homepage, anime page, staff page, and user page, enhancing the overall user experience.
- **AnimeList, StaffList, UserList:**
  These components display a lists of all available anime, staff, and users on the platform, respectively. Each list is designed to be easily navigable, allowing users to quickly find specific entries and access detailed information.
- **AnimeDetails, StaffDetails, UserDetails:**
  These components present all the available information for a selected anime, staff member, or user. Each detail view is designed to show data of a particular anime, member or user including descriptions, ratings, and other information, ensuring that users have access to everything they need.
- **Login and Signup:**
  These components enable user authentication, allowing users to log in and sign up. The Login and Signup forms are designed to be user-friendly.

Overall, the frontend module prioritizes user engagement and ease of use, making it an essential part of the anime recommendation system.

5.1.1.3. Backend Development

The backend module of the anime recommendation system is built on the Django framework, the backend is structured to support RESTful APIs, enabling easy communication with the React frontend. This architecture allows the application to deliver dynamic content and personalized recommendations based on user preferences.

The backend processes requests sent to specific endpoints, each designed to handle distinct operations related to the anime recommendation system. Below are the key API endpoints and their functionalities:

- **https://127.0.0.1/api/anime:**
  This endpoint handles anime-related operations, such as retrieving a list of available anime, adding new titles, updating existing anime information, and deleting titles. It communicates directly with the database to ensure that users receive accurate and up-to-date information, sending responses back to the frontend in a structured format.
- **https://127.0.0.1/api/staff:**
  This endpoint performs operations related to the staff involved in the anime industry, including retrieving staff details, managing staff records, and updating information. By efficiently managing staff-related data, this endpoint enhances the user experience by providing relevant information about creators, voice actors, and other contributors to the anime.
- **https://127.0.0.1/api/user:**
  This endpoint manages user-related operations, including user registration, authentication, and profile management. It allows users to update their favorite's list and manage their personal details. By securely handling user data, this endpoint ensures a smooth and personalized experience for each user.
- **https://127.0.0.1/api/recommendation:**
  This endpoint returns personalized anime recommendations based on the user's favorites. This endpoint passes user's favorite list to our model after which the model returns recommendations based on user's preferences which it returns to the frontend.
- **https://127.0.0.1/api/search:**
  This endpoint facilitates searching for specific anime, staff, or users. When request is sent to this endpoint it searches the anime from the database then uses our model to generate similar anime based on general trends which it returns to the frontend.

Overall, the backend module is designed to ensure that the anime recommendation system operates seamlessly, providing users with responsive, personalized experiences while maintaining high standards of security and data integrity. Through the implementation of well-defined API endpoints, the backend effectively bridges the gap between the frontend and the database, supporting the overall functionality of the application.

5.1.1.4. Database Development

The database for the anime recommendation system is essential for efficiently storing and managing data related to users, anime titles, staff members, and personalized recommendations. We chose PostgreSQL as our relational database management system (RDBMS) due to its robust features and support for complex queries, ensuring both data integrity and optimized performance.

To interact with the database, we utilized Django's powerful libraries. Specifically, we employed django.db.models to define models that correspond to the tables in the database. Each model represents a specific data entity, enabling seamless mapping between the application and the database structure. Additionally, we leveraged the django.db.models.QuerySet to perform various database operations, including creating, retrieving, updating, and deleting records. This abstraction allows for efficient and straightforward query handling while maintaining the flexibility and scalability of the application.

5.1.1.5. Security Measures

Below are the security measures taken to make our application secure

- ·**Use of latest version of Django**: We use the latest version of Django because outdated or vulnerable versions of Django can expose the application to known security vulnerabilities.
- ·**Use of TLS**: We enable https therefore Transport Layer Security (TLS) is employed to encrypt data transmitted between the client and server. This ensures data integrity and confidentiality because TLS involves encrypting the data being transferred .This protects sensitive information, such as user credentials and personal data, from interception by malicious entities.
- **User Input Validation**: User Input cannot be trusted. Attacks like SQL injection and Cross-Site Scripting can be done through inputs in forms. Therefore, we validate and sanitize user input to prevent injection attacks, such as SQL injection

and Cross-Site Scripting (XSS). We do it by utilizing Django's built-in form validation and model methods, we ensure that only safe and expected data is processed by the application.

- **Prevention of Open Redirects**: To mitigate the risk of open redirects, we validate and restrict redirect URLs to a predefined list of trusted domains. This prevents malicious users from redirecting legitimate users to harmful sites.
- **Utilization of Security Middleware**: We enable the Django's security middleware, such as django.middleware.security security Middleware to set various HTTP headers that protect against vulnerabilities, including XSS and clickjacking. This helps enhance the overall security posture of the application with minimal effort.
- **Secure use of cookies**: We configure cookies are with security attributes such as HttpOnly, Secure, and SameSite to prevent unauthorized access and mitigate risks associated with XSS and Cross-Site Request Forgery (CSRF). This ensures that user sessions are protected against common attacks.
- **Prevent Brute-Force Attacks Against Authorization**: Simple Measures such as account lockouts are employed to protect user accounts from brute-force attacks. These practices make it significantly more challenging for attackers to gain unauthorized access.
- **Implement CORS**: Cross-Origin Resource Sharing (CORS) is used to specify which domains are allowed to access the API. This prevents unauthorized domains from making requests, thereby enhancing the security of the application and protecting it from potential cross-origin attacks.

5.1.2 Model Implementation

Technologies Used

- **TensorFlow**: Used for model training, implementing neural networks, and handling computations efficiently.
- **Pandas**: Used for data manipulation, preprocessing, and handling large datasets.
- **Gensim**: Used for Word2Vec-style embeddings, generating anime vectors for recommendations.

- **Joblib**: Used for saving and loading trained models efficiently to avoid retraining every time.
- **Numpy**: Used for performing calculations such as dot products, mean and selecting random anime for negative sampling

Modules Description:

Implementation of Word2Vec our Anime Recommendation System

Step 1: Data is collected from Anilist.com using their api which uses GraphQL. We used the requests library to send requests.

```
def fetch_users_from_id(total_users, csv_filename):
    page = 34518
    per_page = 50
    unique_rows = set()
    unique_anime_ids = pd.DataFrame(columns=["anime_id"])

    with open(csv_filename, mode="w", newline="", encoding="utf-8") as file:
        writer = csv.writer(file)
        writer.writerow(["User ID", "Anime ID"])
        while len(unique_rows) < total_users:
            variables = {"page": page, "perPage": per_page  }
            response = requests.post(url, json={"query": query, "variables": variables})
            try:
                data = response.json()
            except ValueError:
                print("Error: Failed to parse JSON")
                break
            if data and "data" in data and "Page" in data["data"] and "users" in data["data"]["Page"]:
                users = data["data"]["Page"]["users"]
                for user in users:
                    user_id = user["id"]
                    if "favourites" in user and "anime" in user["favourites"]:
                        for anime in user["favourites"]["anime"]["nodes"]:
                          anime_id = anime["id"]
                          unique_row = (user_id, anime_id)
                          if unique_row not in unique_rows:
                            unique_rows.add(unique_row)
                            writer.writerow([user_id, anime_id])
                page += 1
```

We first open a CSV file and write the header row. This is the file where we store the data we collect from the API. We use the request library to send graphQL API requests to retrieve our desired data. We set the necessary parameters such as page and per page. Page determines the page number from which we retrieve the user and perPage determines how many users we retrieve per request. We loop and send request until the required number of unique (user_id, anime_id) pairs are collected. We also check if the response is as expected i.e does it have the keys “data”, “page”, “users”. Then we extract the users favorites from the response and save it in our csv files. We then increment the page number by 1 which moves to the next page for more data.

Step 2: We create a tokenizer using the tensorflow text tokenizer.

```
def main():
    #Getting data from csv files
    fav_anime_filtered = pd.read_csv(
        "Fav_Anime_Filtered_Final.csv", names=["User_Id", "Anime_Ids"]
    )
    corpus = fav_anime_filtered["Anime_Ids"]

    # Tokenizing the data
    tokenizer = tf.keras.preprocessing.text.Tokenizer(
        filters="", lower=False, split=",", oov_token="OOV"
    )
    tokenizer.fit_on_texts(corpus)
    joblib.dump(tokenizer, open("tokenizer.sav", "wb"))
```

We load the csv file where data was stored in the previous step into memory using the pandas library. Then the data is tokenized using the tensorflow library. We split the anime I’ds by “,” . This creates a vocabulary of unique anime in our dataset. We then save the tokenizer as a .sav file using the joblib library.

Step 3: For preprocessing we need to create positive and negative samples. To ensure frequently occurring anime do not dominate sampling we create a custom sampling table

```
def make_sampling_table(word_frequencies):
   sorted_word_count = {k: v for k, v in sorted(word_frequencies, key=lambda item:
   item[1], reverse=True)}  #sorting in order of frequency
    total = []
    for item, count in sorted_word_count.items():
        total.append(pow(count, 0.5))
    total_sum = numpy.sum(total)
    probs = numpy.divide(total, total_sum)
    return list(probs)
```

For the sampling table we sort the unique anime in our tokenizer according to frequency of their occurences. We set the probability of a anime being sampled by taking the total count of an anime and dividing by the total number of words in the vocabulary. We raise the total count by a power of 0.5 to flatten the probability curve. So that frequent words get sampled less often

Step 4: Now we preprocess the data which involves generating positive and negative from the data we collected. We use window size of sample

```
def generate_training_data(sequences, window_size, num_ns):
    targets, contexts, labels = [], [], []
    for sequence in tqdm.tqdm(sequences):
    #Generating positive samples
        positive_skip_grams, _ = tf.keras.preprocessing.sequence.skipgrams(
            sequence, vocabulary_size=vocab_size,  window_size=window_size,
            negative_samples=0,  seed=1)
   #Generating 5 negative sample for every positive sample
        for target_word, context_word in positive_skip_grams:
            negative_sampling_words = []
            negative_sampling_words = numpy.random.choice(
                vocab_words, size=num_ns, =False, p=sampling_table,)
            negative_sampling_words = [
                x for x in negative_sampling_words if x != target_word and x != context_word]
            while len(negative_sampling_words) != num_ns:
                additional_samples = numpy.random.choice(
                    vocab_words, size=num_ns - len(negative_sampling_words),
                    replace=False, p=sampling_table)
                additional_samples = [
                    x for x in additional_samples if x != target_word and x != context_word]
                negative_sampling_words.extend(additional_samples)
            context = tf.concat(
                [tf.constant([context_word], dtype="int64"),
                    tf.constant(negative_sampling_words, dtype="int64") ], axis=0,)
            targets.append(target_word)
```

Now we generate the target values, context values and labels. For each target value we generate 6 context values, one positive sample and the rest being negative. We generate the positive samples using the tensorflow library. For the negative sample we use the numpy.random.choice function. We use this function because we can use our own sampling table here. We generate 5 random anime form our tokenizers vocabulary and if any matches with our positive samples we generate more negative samples until there are 5. Labels are value 1 for positive samples and 0 for negative samples.

Step 5: For word2vec we define a custom model by subclassing tf.keras.Model. We define the embedding layers which represent the target and context vectors and we define what the model does during each forward pass in the call function.

```
class Anime2Vec(tf.keras.Model):
    def __init__(self,  vocab_size, embedding_dims, target_embeddings=None,
       context_embeddings=None ):
       #Target vectors
        self.target_embeddings = (target_embeddings if target_embeddings
            else tf.keras.layers.Embedding(
                vocab_size, embedding_dims, name="A2V_embedding"))
      #context vectors
        self.context_embeddings = (context_embeddings
            if context_embeddings else tf.keras.layers.Embedding(
                vocab_size, embedding_dims, name="A2V_context"))


        self.vocab_size = vocab_size
        self.embedding_dims = embedding_dims
        super(Anime2Vec, self).__init__()
    #Calculating dot products of target vectors and context vectors
    def call(self, pair):
        target, context = pair
        if len(target.shape) == 2:
            target = tf.squeeze(target)
        word_embedding = self.target_embeddings(target)
        context_embedding = self.context_embeddings(context)
        dots = tf.einsum("be, bce -> bc", word_embedding, context_embedding)
        return dots
```

Now we customize the tensorflow feed forward neural network model for our word2vec model. For this we create two embedding layers. One for target vectors and one for context vectors. Then in the call function we define what model does in each forward pass. So in the call function the model gets the vector of target anime from target embedding layer and context vector of context anime from the context embedding layer. Dot product of the vectors is taken and returned.

Step 6: We set Model Hyperparameters such as target embedding size, context embedding size, learning rate, loss function, metrics

```
tokenizer = joblib.load(open("tokenizer.sav", "rb"))
vocab_size = len(tokenizer.index_word) + 1
optimizer = keras.optimizers.Adam(learning_rate=0.0001)
 embedding_dims = 100
 anime2vec = Anime2Vec(vocab_size, embedding_dims,)
anime2vec.compile(optimizer=optimizer,
                loss=tf.keras.losses.CategoricalCrossentropy(from_logits=True),
                metrics=["accuracy"])
```

We define the vocabulary size of the model, which is the the length of the tokenizer vocabulary + 1. We then set the optimizer, which we get from the keras library. We use the keras optimizer with the learning rate of 0.0001. We set the embedding dims to 100 in the code above but we have set it to 50, 100 and 150 in different tests. We then Initialize our model. During compilation we set the loss and metrics as well. We use the categorical cross entropy to calculate loss and we set the metrics to be accuracy.

Step 7: Now the dataset is shuffled for more variation, batched for smoother gradient descent and cached and then the model is trained for a certain number of epochs.

```
BATCH_SIZE = 128
BUFFER_SIZE = 2048
dataset = tf.data.Dataset.from_tensor_slices(((targets, contexts), labels))
dataset = dataset.shuffle(BUFFER_SIZE).batch(
            BATCH_SIZE, drop_remainder=True)
dataset = dataset.cache().prefetch(buffer_size=tf.data.AUTOTUNE)
anime2vec.fit(dataset, epochs=epochs)
```

We use the mini batch gradient descent method because it leads to faster training and smoother convergence. We use a batch size of 128 due to time limitations. Buffer size is the number of data points loaded into the memory before feeding it to the model. We set it to 2048. We then create tensor slices for from our targets, contexts and labels. We shuffle the dataset for better generalization then cache and prefetch the data. We then fit the data on our model.

Step 8: After model is finished training we retrieve the target and context embeddings. We save the dimensions in one file and the metadata i.e. corresponding anime Id in a different file.

```
target_weights = anime2vec.get_layer("A2V_embedding").get_weights()[0]
context_weights = anime2vec.get_layer("A2V_context").get_weights()[0]
out_v = io.open("anime_semantic_dims100dims.tsv", "w", encoding="utf-8")
out_m = io.open("anime_semantic_metadata100dims.tsv", "w", encoding="utf-8")
out_sv = io.open("anime_syntactic_dims100dims.tsv", "w", encoding="utf-8")
out_sm = io.open("anime_syntactic_metadata100dims.tsv", "w", encoding="utf-8")
#Semantic Vectors
for index, anime in vocab.items():
    vector = target_weights[index]
    out_v.write("\t".join([str(weight) for weight in vector]) + "\n")
    out_m.write(anime + "\n")
#Syntactic Vectors
for index, anime in vocab.items():
    vector = target_weights[index] + context_weights[index]
    out_sv.write("\t".join([str(weight) for weight in vector]) + "\n")
    out_sm.write(anime + "\n")
#
```

After model is finished training we retrieve the entire target and context weights from the two embedding layers.We then loop over each anime in the vocabulary and retrieve its corresponding id and vector. Which are saved in tsv files. We simply use the python io library for this task.

Step 9: Finally we convert the word embeddings to KeyedVectors format. KeyedVectors format is optimized for similarity searches and using only the underlying embeddings means we do not have to use the entire model which saves memory

```
#Loading the vectors
embedding_dims = csv.reader(
  open("anime_semantic_dims100dims.tsv", "r", encoding="UTF-8"), delimiter="\t")
#Loading the metadata
tokens = csv.reader(
   open("anime_semantic_metadata100dims.tsv", "r", encoding="UTF-8"),
delimiter="\t")


AnimeKeyedVectors = gensim.models.KeyedVectors(100)
token_vector = {}
#Converting to KeyedVector format using gensim
for token, embedding_dim in zip(tokens, embedding_dims):
    if token[0] == "OOV":
        AnimeKeyedVectors.add_vector(token[0], list(map(float, [0] * 100)))
    else:
        AnimeKeyedVectors.add_vector(token[0], list(map(float, embedding_dim)))
AnimeKeyedVectors.save("100dims3EpochSyntacticAnimeKeyedVectors.kv")
```

Finally we retireve the data from the tsv files. We zip the anime metadata and the corresponding vector and loop over them.This results in adding each anime id and its corresponding vector to the genism keyed vector object. We then save the keyed vector object as a kv file.

5.2 Testing

5.2.1 Application Testing

5.2.1.1 Test Cases for Unit Testing

Here, we tested individual unit or components of a system. The purpose is to validate as each unit of the module performs as expected to be performed.

Table 5.1: Test Case for Search Function

Symbol No.	**Test Cases**	**Input Test Data**	**Expected Results**	**Test Result**	**Pass/Fail**
1	Search for a valid anime name	Search Query : Takarajima	Search results displayed matching the anime name	Search results displayed matching the anime name	Pass
2	Search with an empty query	Search Query: " "	All Anime displayed	All Anime displayed.	Pass
3	Search with mixed-case letters	Search Query : taKariJima	Search results displayed matching the query (case insensitive).	Search results displayed matching the query (case insensitive).	Pass

Table 5.2: Test Case for Recommendation Functionality

Symbol No.	Test Cases	Input Test Data	Expected Results	Test Result	Pass/Fail
1	Request recommendations for a user with favorites	User Favorites: Takarajima, Turn A Gundam, Hunter X Hunter, Bakemonogatari	Recommendations displayed based on user's favorites list.	Recommendations displayed based on user's favorites list.	Pass
2	Request recommendations for a new user	User Favourites: null	Recommendations displayed based on typically liked anime.	Recommendations displayed based on typically liked items.	Pass

Table 5.3: Test Case for User Log In

Symbol No.	Test Cases	Input Test Data	Expected Results	Test Result	Pass/Fail
1	Enter valid email and password	Email: prabin@gmail.com Password: Amatya@666	Login successful, redirect to home page.	Login successful, redirect to home page.	Pass

2	Enter invalid email and valid password	Email: prabin@.com Password: Amatya@666	Login unsuccessful, display error message.	Login unsuccessful, display error message.	Pass
3	Enter valid email and empty password	Email: prabin@.com Password: null	Login unsuccessful, display error message.	Login unsuccessful, display error message.	Pass
4	Enter both email and password as empty	Email: null Password: null	Login unsuccessful, display error message.	Login unsuccessful, display error message.	Pass

Table 5.4:Test Case for User Registration Functionality

Symbol No.	Test Cases	Input Test Data	Expected Results	Test Result	Pass/Fail
1	Enter valid email, password, and confirm password	Email: prabin@gmail.com Password: Amatya@666 Confirm Password: Amatya@666	Registration successful, redirect to login page.	Registration successful, redirect to login page.	Pass

2	Enter invalid email format	Email: prabin@.com Password: Amatya@666 Confirm Password: Amatya@666	Registration unsuccessful, display error message.	Registration unsuccessful, display error message.	Pass
3	Enter mismatched password and confirm password	Email: prabin@gmail.com Password: Amatya@666 Confirm Password: Amatya123@666	Registration unsuccessful, display error message.	Registration unsuccessful, display error message.	Pass
4	Enter an already registered email	Email: madhu@gmail.com Password: Aryal@666 Confirm Password: Aryal@666	Registration unsuccessful, display error message.	Registration unsuccessful, display error message.	Pass
5	Enter empty fields for email and password	Email: null Password: null Confirm Password: null	Registration unsuccessful, display error message.	Registration unsuccessful, display error message.	Pass

6	Enter valid email and password but leave confirm password empty	Email: prabin@gmail.com Password: Amatya@666 Confirm Password: null	Registration unsuccessful, display error message.	Registration unsuccessful, display error message.	Pass

5.2.1.2 Test Cases for System Testing

Here, overall system is tested to evaluate either the system be compliances with its specified requirements or not. One of the system testing is the usability testing which is performed in the system.

Table 5.5: Test Case for Recommendation Changing with User's Favorites

Symbol No.	Test Cases	Expected Results	Test Result	Pass/Fail
1	Go to search bar and search for an anime	Anime appears in the results	Anime appears in the results	Pass
2	Click the add to favorites button and add the anime	The anime is visible in the favorites list and the add to favorites button disappears and remove from favorites button appears for the anime	The anime is visible in the favorites list and the add to favorites button disappears and remove from favorites for the anime	Pass
3	Remove the anime	Anime is removed	Anime is removed	Pass

	from favorites	from favorites list and remove from favorites button disappears add to favorites button appears again	from favorites list and remove from favorites button disappears add to favorites button appears again	
4	Update favorites and request recommendations	Recommendations reflect the updated favorites list.	Recommendations reflect the updated favorites list.	Pass

5.2.2. Model Testing

Testing in this project evaluates the effectiveness of the anime recommendation model by measuring how well it understands and represents anime relationships. We divide testing into two categories: semantic accuracy and syntactic accuracy, each assessed through a series of targeted tests. To determine the best-performing model, we run multiple versions with different configurations and compare their results.

Test Results

Semantic Tests:

Table 5.6: Test Case for Similarity Evaluation

Model Version	**Precision**	**Recall**	**F1 Score**	**Overall Accuracy**	**Positive Accuracy**	**Negative Accuracy**
50 Dimensions - Target Vector	0.54	0.26	0.35	0.61	26.14%	85.35%
50 Dimensions - Target + Context Vector	0.47	0.63	0.54	0.56	64.4%	52.37%

100 Dimensions - Target Vector (Trained for 3 Epochs)	0.76	0.12	0.21	0.63	12%	97%
100 Dimensions - Target + Context Vector (Trained for 3 Epochs)	0.4	1	0.57	0.4	100%	0%
100 Dimensions - Target Vector (Trained for 5 Epochs)	0.71	0.6	0.65	0.73	60.02%	83.44%
100 Dimensions - Target + Context Vector (Trained for 5 Epochs)	0.42	0.72	0.53	0.49	72.93%	33.15%
150 Dimensions - Target Vector	0.58	0.20	0.30	0.63	20.37%	90.35%
150 Dimensions - Target + Context Vector	0.43	0.68	0.51	0.53	68.62%	39.96%

Table 5.7: Outlier Detection Test

Model Version	**Correct Prediction Rate**
50 Dimensions - Target Vector	0.37
50 Dimensions – Target + Context Vector	0.38
100 Dimensions - Target Vector (Trained for 3 Epochs)	0.36
100 Dimensions - Target + Context Vector (Trained for 3 Epochs)	0.48
100 Dimensions - Target Vector (Trained for 5 Epochs)	0.66
100 Dimensions - Target + Context Vector (Trained for 5 Epochs)	0.31
150 Dimensions	0.38

- Target Vector	
150 Dimensions - Target + Context Vector	0.277

Table 5.8:Negative Prediction Test

Model Version	Correct Prediction Rate
50 Dimensions - Target Vector	0.95
50 Dimensions – Target + Context Vector	0.94
100 Dimensions - Target Vector (Trained for 5 Epochs)	0.94
100 Dimensions - Target Vector (Trained for 5 Epochs)	0.91
100 Dimensions - Target Vector (Trained for 5 Epochs)	0.99

100 Dimensions - Target Vector (Trained for 5 Epochs)	0.86
150 Dimensions - Target Vector	0.95
150 Dimensions - Target + Context Vector	0.87

Syntactic Tests

Table 5.9: One hit Percentage Test

Model Version	**Positive One hit Percentage**	**Negative One hit Percentage**
50 Dimensions - Target Vector	36.87%	0.19%
50 Dimensions – Target + Context Vector	84.59%	2.08%
100 Dimensions - Target Vector (Trained for 3 Epochs)	65.65%	1.047%
100 Dimensions –	88.48%	0.7%

Target + Context Vector (Trained for 3 Epochs)		
100 Dimensions - Target Vector (Trained for 5 Epochs)	60.29%	0.07%
100 Dimensions - Target + Context Vector (Trained for 5 Epochs)	80.67%	12.5%
150 Dimensions - Target Vector	49.12%	0.41%
150 Dimensions - Target + Context Vector	75.92%	10.1%

5.2.3 Result Analysis

5.2.3.1 Categories of tests

Testing is divided into two key categories: semantic accuracy and syntactic accuracy. These concepts originate from word vector models like Word2Vec and are adapted to evaluate the effectiveness of our anime recommendation system.

5.2.3.2 Semantic Accuracy

In Regular Text Processing:

Semantic accuracy measures how well a word vector model captures meaningful relationships between words. For example, in a well-trained model, the vector for “king” should be close to “queen,” and “Paris” should be close to “France.” This indicates that the model understands word meanings and their associations.

In the Recommendation System:

Semantic accuracy in our anime recommendation system evaluates whether the model correctly associates similar anime based on user preferences. If many users who like *Naruto* also like *Bleach*, a semantically accurate model should position them closely in the vector space. High semantic accuracy ensures that anime which occur together on favorites lists frequently are clustered together in vector space.

5.2.3.3 Syntactic Accuracy

In Regular Text Processing:

Syntactic accuracy measures how well a model understands a word’s relationship with surrounding words. For example, it should recognize that “run” and “running” are related forms of the same verb, or that “he is eating” follows proper grammatical structure. A high syntactic accuracy ensures that words are positioned correctly in relation to their context.

In the Recommendation System:

Syntactic accuracy in our anime recommendation system evaluates whether the model provides relevant recommendations based on a user’s favorites. This means that if a user has a specific set of favorite anime, the model should accurately recommend others that align with their interests. A syntactically accurate model ensures that recommendations are not just generally related but are also personalized and contextually relevant based on the user's favorites.

By evaluating both semantic and syntactic accuracy, we ensure that the model produces high-quality recommendations that reflect both strong direct associations and the ability to make useful, personalized suggestions.

5.2.3.4 Categories of tests

i. Semantic Tests

We perform 3 Semantic Tests which are given below

Types of Semantic Tests

a. Similarity Evaluation Test

The Similarity Evaluation Test is designed to assess the effectiveness of the anime recommendation model in identifying similar and dissimilar anime based on computed similarity scores. This test evaluates the model's semantic accuracy by comparing the model's predictions with actual user preferences, ultimately allowing us to measure the quality of recommendations.

Evaluation Metrics

To quantify the performance of the model during the Similarity Evaluation Test, we utilize several key metrics:

1. **Precision**: Precision measures the proportion of true positive predictions among all positive predictions made by the model. It reflects how many of the recommended similar anime were indeed similar to the target anime. A high precision indicates that when the model recommends an anime, it is likely to be relevant.

   Formula:

   $$\text{Precision}= \frac{\text{True Positive}}{\text{True Positive}+\text{False Positive}}$$

2. **Recall**: Recall measures the proportion of true positive predictions among all actual positive instances. It indicates how many of the truly similar anime were correctly identified by the model. A high recall suggests that the model is effectively capturing most of the relevant recommendations.

Formula:

Recall= $\frac{\text{True Positive}}{\text{True Positive+False Negative}}$

3. **F1 Score**: The F1 Score is the harmonic mean of precision and recall. It provides a balanced measure of the model's accuracy, especially when there is an uneven class distribution. A higher F1 Score indicates a better balance between precision and recall.

   Formula:

   F1 Score=2* $\frac{\text{Precision}*\text{Recall}}{\text{Precision + Recall}}$

4. **Accuracy:** Accuracy measures the proportion of correct predictions (both true positives and true negatives) among all predictions made by the model. It provides an overall assessment of the model's performance.

   Formula:

   Accuracy=$\frac{\text{True Positive+True Negative}}{\text{True Positive+True Negative+False Positive+False Negative}}$

5. **Positive Accuracy:** Positive accuracy measures the proportion of correct predictions for similar anime only. We calculate the sum of similar anime the model flagged as similar and then dividing by the total number of similar anime pairs in the test set.

   Formula:

   Positive Accuracy = $\frac{True\ Positive}{Total\ Number\ of\ Similar\ Anime\ Pairs\ In\ Test\ Set}$

6. **Negative Accuracy:** Negative accuracy measures the proportion of correct predictions for dissimilar anime only. We calculate the sum of dissimilar anime the model flagged as dissimilar and then dividing by the total number of dissimilar anime pairs in the test set.

   Formula:

   Positive Accuracy = $\frac{True\ Negative}{Total\ Number\ of\ Dissimilar\ Anime\ Pairs\ In\ Test\ Set}$

Calculating Metrics

During the Similarity Evaluation Test, we calculate these metrics based on the counts of true positives, false positives, true negatives, and false negatives as follows:

- **True Positives (TP):** The number of similar anime that were correctly identified by the model.
- **False Positives (FP):** The number of anime that were incorrectly identified as similar (but are not).
- **False Negatives (FN):** The number of similar anime that were not identified by the model.
- **True Negatives (TN):** The number of dissimilar anime that were correctly identified as not similar.

By analyzing these metrics, we can gain insights into the model's strengths and weaknesses in generating accurate recommendations. This comprehensive evaluation ensures that the model is not only identifying relevant anime but also minimizing incorrect suggestions, ultimately enhancing the user experience in the recommendation system

b. Outlier Detection Test

The Outlier Detection Test is a key evaluation method used in the semantic analysis of our anime recommendation model. This test assesses the model's ability to distinguish between similar and dissimilar anime by identifying outliers within a given set of recommendations. It serves as an important measure of the model's understanding of relationships between different anime titles.

Testing Procedure

1. **Selection of Anime:** For each target anime in the dataset, the model selects a set of similar anime based on the existing recommendations.
2. **Inclusion of a Dissimilar Anime:** A random dissimilar anime is chosen from the dataset, ensuring that it does not overlap with the selected similar anime. This dissimilar anime serves as the outlier in the test.

3. **Running the Model:** The model is then asked to determine which anime in the combined set (the similar anime and the outlier) does not belong. The expectation is that the model will identify the dissimilar anime as the outlier.
4. **Evaluation of Results:** The model's performance is measured by comparing its prediction with the actual outlier. If the model correctly identifies the dissimilar anime, it indicates that it has a robust understanding of the relationships between anime

Metrics for Evaluation

The success of the Doesn't Match Test is quantified by the following metric:

- **Correct Predictions:** The number of times the model correctly identifies the outlier from the set of similar anime.
- **Formula:** $\frac{\text{Total Number of times model identifies the correct outlier}}{\text{Total number of predictions}}$

This test is particularly valuable because it highlights the model's ability to see nuanced differences between anime titles. By successfully identifying outliers, the model demonstrates its capability to provide accurate recommendations that truly align with user preferences, thereby enhancing the overall effectiveness of the anime recommendation system.

b. Dissimilar Anime Identification Test

The Dissimilar Anime Identification Test is a crucial evaluation method in our semantic analysis framework for the anime recommendation model. This test focuses on assessing the model's ability to avoid recommending anime that do not align with a user's preferences, thereby enhancing the overall quality and relevance of the recommendations provided.

Testing Procedure

1. **Selection of Target Anime:** For each anime in the dataset, we retrieves a list of anime that are similar.

2. **Negative Sampling:** The model then identifies the top dissimilar anime with the target anime as a negative input. This allows the model to find anime that are least similar to the target.
3. **Evaluation of Recommendations:** The test checks whether any of the dissimilar anime overlap with the actual similar anime. If there is no overlap, it indicates that the model has successfully identified dissimilar options and refrained from recommending them.

Metrics for Evaluation:

The model's performance is measured based on the following criteria:

- **Correct Predictions:** The number of times the model successfully avoids recommending dissimilar anime that overlap with the similar recommendations.
- **Formula:** $\frac{\text{Total Number of times model flagged dissimilar anime as dissimilar}}{\text{Total Number of dissimilar anime}}$

The Dissimilar Anime Identification Test is essential for ensuring the integrity of the anime recommendation system. By effectively filtering out unrelated titles, the model enhances its ability to provide accurate and relevant recommendations to users. This not only improves user satisfaction but also strengthens the overall effectiveness of the recommendation system, making it a vital component of our semantic analysis efforts.

ii. Syntactic Tests

We perform one test for syntactic analysis which is given below

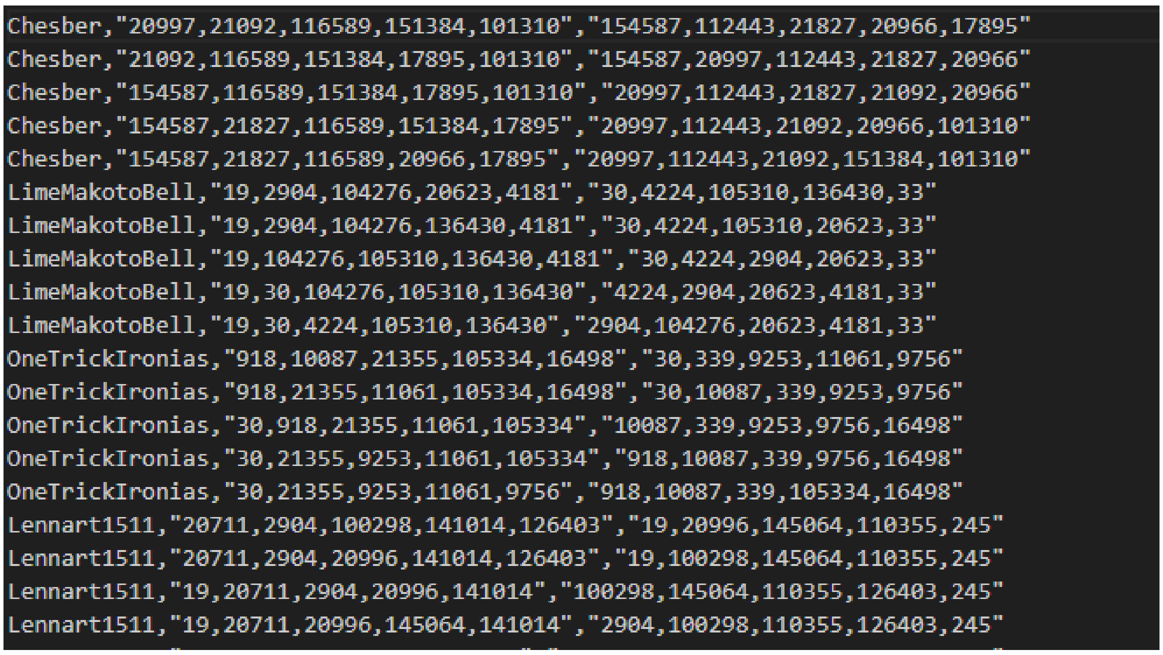
i. Semantic Tests

We perform 3 Semantic Tests which are given below

Overview of Dataset

Dataset for semantic tests looks as in table 5.11. The first column represents the user, the second column represents on left represents the one half of the user anime that is fed to the model and third column represents the second half of the user list that the model has to predict. Each user list can have multiple combinations of the two halves which is why each user appears multiple times in this dataset.

Table 5.10: Syntactic test dataset



```
Chesber,"20997,21092,116589,151384,101310","154587,112443,21827,20966,17895"
Chesber,"21092,116589,151384,17895,101310","154587,20997,112443,21827,20966"
Chesber,"154587,116589,151384,17895,101310","20997,112443,21827,21092,20966"
Chesber,"154587,21827,116589,151384,17895","20997,112443,21092,20966,101310"
Chesber,"154587,21827,116589,20966,17895","20997,112443,21092,151384,101310"
LimeMakotoBell,"19,2904,104276,20623,4181","30,4224,105310,136430,33"
LimeMakotoBell,"19,2904,104276,136430,4181","30,4224,105310,20623,33"
LimeMakotoBell,"19,104276,105310,136430,4181","30,4224,2904,20623,33"
LimeMakotoBell,"19,30,104276,105310,136430","4224,2904,20623,4181,33"
LimeMakotoBell,"19,30,4224,105310,136430","2904,104276,20623,4181,33"
OneTrickIronias,"918,10087,21355,105334,16498","30,339,9253,11061,9756"
OneTrickIronias,"918,21355,11061,105334,16498","30,10087,339,9253,9756"
OneTrickIronias,"30,918,21355,11061,105334","10087,339,9253,9756,16498"
OneTrickIronias,"30,21355,9253,11061,105334","918,10087,339,9756,16498"
OneTrickIronias,"30,21355,9253,11061,9756","918,10087,339,105334,16498"
Lennart1511,"20711,2904,100298,141014,126403","19,20996,145064,110355,245"
Lennart1511,"20711,2904,20996,141014,126403","19,100298,145064,110355,245"
Lennart1511,"19,20711,2904,20996,141014","100298,145064,110355,126403,245"
Lennart1511,"19,20711,20996,145064,141014","2904,100298,110355,126403,245"
```

One Hit Test

The One Hit Test is a pivotal evaluation method used in the syntactic analysis of our anime recommendation model. This test focuses on assessing the model's ability to generate recommendations based on users' individual preferences, ensuring that the recommendations align closely with the users' favorite anime.

Testing Procedure

1. **Data Collection:** For each user in the dataset, the model collects their favorite anime and splits their lists into two halves, one half is given to the model as input and the model has to predict the other half.
2. **Recommendation Generation:** Using the user's favorite anime as a positive input, the model generates a list of potential recommendations. This list includes the top anime that are most similar to the user's favorites.
3. **Negative Sample Generation:** In addition to generating positive recommendations, the model also identifies a list of dissimilar anime by treating the user's favorites as negative input and the model is made to predict anime that the user will dislike

4. **Evaluation of Results:** The model's performance is measured by comparing its prediction with the remaining half of the user's favorite list. If the model recommends at least one anime in the users list, it indicates that user can generate recommendations that users will like, and if the models negative recommendations do not match the users actual favorites, it indicates it won't generate recommendations users will dislike

Metrics of Evaluation: The effectiveness of the model is assessed by comparing the recommendations generated against the user's required anime. The following metrics are calculated:

- **Correct Recommendations Rate:** Percentage of users who got at least one correct recommendation.

  Formula:

  $$\frac{Total\ Number\ of\ times\ model\ recommended\ atleast\ one\ correct\ anime\ based\ on\ positive\ inputs}{Total\ Number\ of\ Users}$$

- **Negative Overlap:** Percentage of users who got at least one incorrect recommendation.

  Formula:

  $$\frac{Total\ Number\ of\ times\ model\ recommended\ one\ correct\ anime\ based\ on\ negative\ inputs}{Total\ Number\ of\ Users}$$

Semantic Analysis

From table 5.6 we can see some patterns in our results. We see that use of only target vectors tends to bring better results which is consistent with word2vec behavior in text processing where target vectors capture semantic relationships better. The models using target + context vectors however do tend to have better recall because they recommend more varied anime due to contextual influence. However, the low precision is a huge issue. We can conclude that models target + context vectors recommend more varied anime but also produce more irrelevant recommendations.

Another thing to note is the results based on increase in dimensions. While theoretically higher dimensional vectors should capture more nuanced relationship the 150 dimensions - Target Vector model only has a precision of 0.58 and recall of 0.20. This could be a result of overfitting and inability to generalize. This shows increasing dimensions doesn't necessarily improve performance

The 100 Dimensions - Target Vector (Trained for 5 Epochs) seems to be the best candidate for semantic tasks in our application. This model demonstrates a strong precision (0.75) but a lower recall (0.6). Therefore, the model is able to identify correct recommendations, but will miss a number of correct recommendations. However, with a f1 score of 0.73 this model provides the best compromise.

Syntactic Analysis

From table 5.9 we can see that the models using target + context vectors consistently outperformed those relying solely on target vectors. Context vectors provide additional information about the relationships between different anime, allowing the model to consider not just the immediate preferences of a user but also the broader relationships that exist within the dataset. This additional layer of information helps the model to better understand user intent and preferences, leading to more relevant and accurate recommendations. This is also consistent with word2vec behavior in text processing where target vectors alone are better for semantic analysis and target + context vectors are better for syntactic analysis

Another thing to note is that model trained for more epoch i.e 100 Dimensions – Target + Context Vector (Trained for 5 Epochs) performed more poorly than its less trained counterpart. This could be due to overfitting and poor generalization. The same trend can be seen when increasing dimensions. The 150 dimension models perform worse than both 50 and 100 dimension model likely due to overfitting and poor generalization again.

The 100 Dimensions - Target + Context Vector (Trained for 3 Epochs) seems to be the best model for syntactic tasks in our application. It achieved the highest positive one-hit percentage of 88.48%, indicating its effectiveness in recommending relevant anime based on user favorites. Its negative one hit ratio is also extremely low 0.7% indicating that it very rarely produces irrelevant recommendations.

Chapter 6: Conclusion and Future recommendations

The world is rapidly advancing in technology, making it essential to create smart and efficient systems for various purposes. The Anime Recommendation System simplifies anime selection by offering personalized recommendations based on user preferences and favorites. It enhances the user experience by providing search functionality and a user-friendly interface. Through this project, we have gained valuable skills such as:

- Understanding and implementing recommendation algorithms.
- Learning how to handle and manage a large anime database.
- Enhancing teamwork and problem-solving skills.
- Developing an interactive and engaging web-based system.
- Understanding user needs and improving user engagement.
- Realizing the importance of maintaining accuracy in recommendations.

Future Recommendations

As technology grows, the demand for personalized recommendation systems will increase. Similarly we can see wide growth in the anime industry so users prefer quick and accurate suggestions without spending hours searching for content. The Anime Recommendation System can be further improved with the following enhancements:

- Expanding the system to a mobile application for better accessibility.
- Improving the recommendation algorithm using AI and machine learning for more accurate results.
- Adding more features like anime trailers, reviews, and community discussions.
- Implementing cybersecurity measures to ensure data privacy and security.
- Integrating multiple payment gateways for premium or subscription-based features.
- With these improvements, the system can offer a more interactive, secure, and efficient anime discovery experience, making it a valuable platform for anime enthusiasts worldwide.

References

[1] O. Syoichiro, K. Hayaki, M. Masahumi, M. Soh and H. Jun'ichi, "AniReco: Japanese Anime Recommendation System," in *Entertainment Computing – ICEC 2017*, Springer, 2017.

[2] A. K. Joseph, "Introduction To Recommender Systems: Algorithms and Evaluation," *ACM Transactions on Information Systems,* vol. 22, no. 1, pp. 1-4.

[3] Suja and S. J. Napier, Anime from Akira to Princess Mononoke, New,York: Palgrave Macmillan, pp. 3-4.

[4] S. Arora, A. May, J. Zhang and C. Re, "Recommender Algorithm for Japanese Animes," Association for Computational Linguistics, 2020.

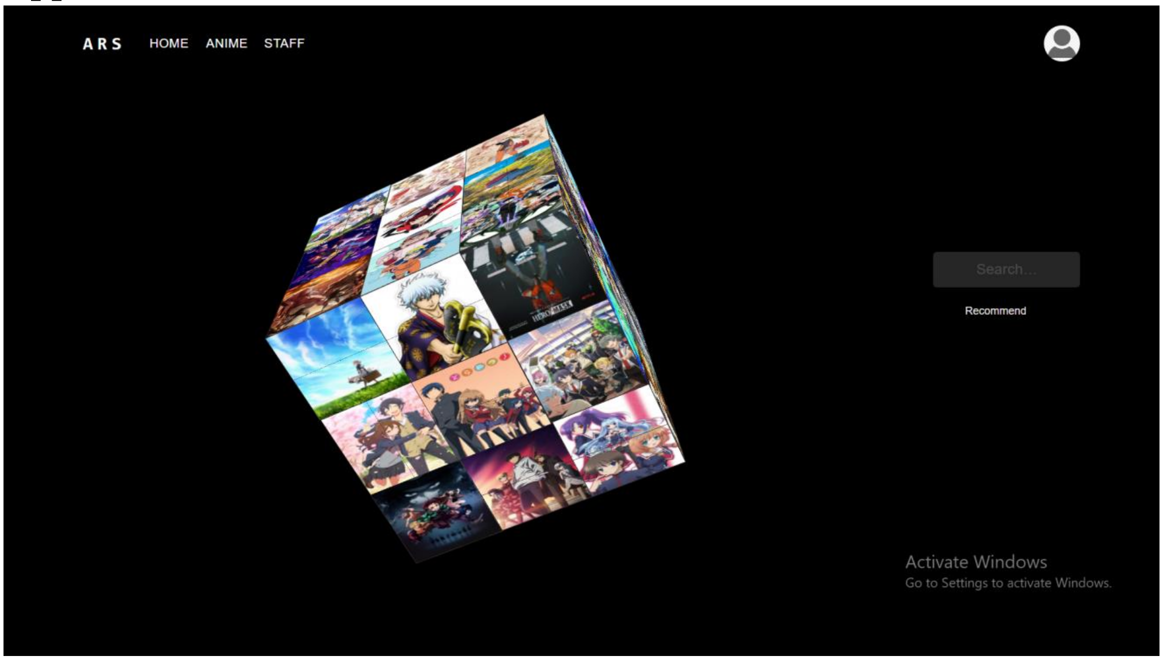
[5] T. Mikolov, K. Chen, J. Dean and G. Corrado, "Efficient Estimation of Word Representations in Vector Space," *Efficient Estimation of Word Representations in,* 2013.

[6] A. Samih, A. Ghadi and A. Fennan, "ExMrec2vec: Explainable Movie Recommender," *International Journal of Advanced Computer Science and Applications (IJACSA),* vol. 12, no. 8, 2021.

[7] T. . P. Adewumi, . L. Foteini and L. Marcus , "Word2Vec: Optimal hyper-parameters and their," *arXiv,* 2021.

Appendices

Appendix A: Screenshot


ARS
HOME
ANIME
STAFF
Search...
Recommend
Activate Windows
Go to Settings to activate Windows.