TheALLDictionaries

Young-jae Moon

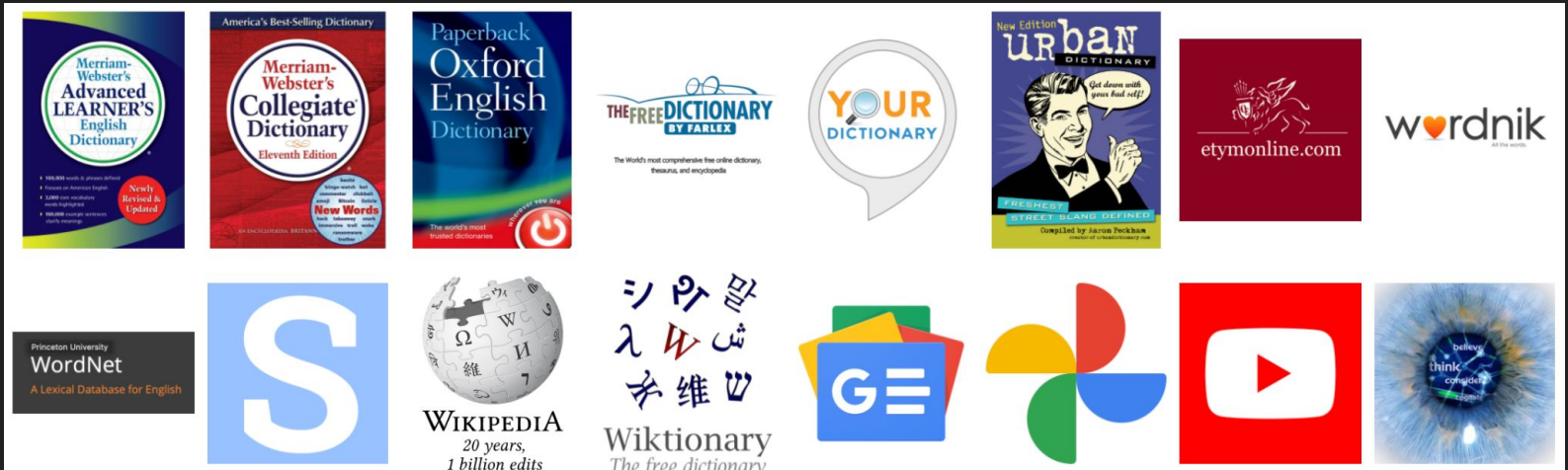
Quick Reminder

Problem Statement

1. People need to look up words, but just 1 source may not be sufficient to understand.
2. While you can look up words across multiple sites...
    a. time consuming and
    b. hence some users just give up trying to understand the word.
3. Once they look up the word they may want to study it further for memorization.

Solution

1. Our website will make it convenient for users to look up a word from 16 different sources.
2. The users can also participate in various typing games to improve their English vocabularies and grammar.
3. The user can chat in a group chatting page for fun.



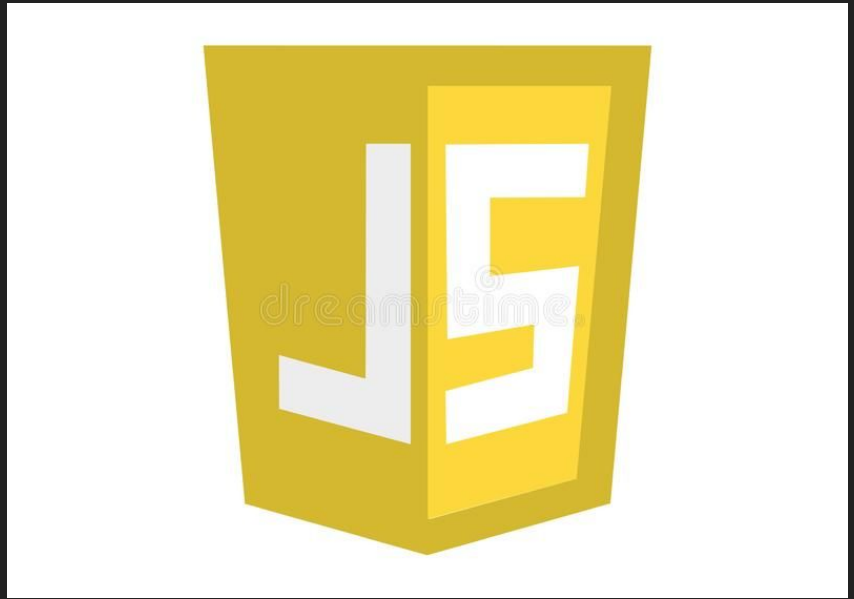
System Architecture Overview

Programming Languages used 1. Javascript

Version number: ECMAScript 2020

JavaScript improves the user experience by allowing the pages to be interactive.

ECMAScript 2020 is the latest stable version of JavaScript.



Programming Language used 2. Python

Version number: 3.8.11

v3.8.11 is the latest stable version for Python 3.8

We have chosen Python 3.8 instead of Python 3.9 for stability when deployed via Heroku.

Type of database - SQL

We decided to use SQL over NoSQL database.

Relational database allows normalization to reduce data redundancy and improve data integrity.

Since we are not using Node.js and Express.js for the back-end framework and use React.js or Vue.js for the front-end framework, we decided not to use MongoDB which stores data in binary JSON (BSON) format.

In other words, we will not employ

1. MERN (MongoDB, Express.js, React.js, Node.js) stack
2. MEVN (MongoDB, Express.js, Vue.js, Node.js) stack

DBMS used - 1. MySQL

Version Number: MySQL Community Edition 8.0.23

Reasons:

1. free to use
2. most popular RDBMS.
3. v8.0.23 is the latest stable version of MySQL.
4. Amazon RDS supports MySQL Community Edition version 8.0.

CSS Frameworks used - Bootstrap 5

Version Number: v5.0.0-beta 3

Reasons:

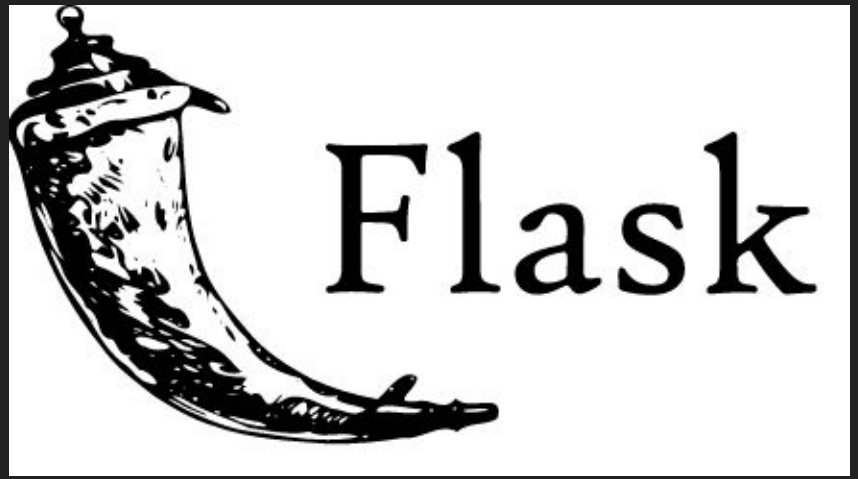
1. Bootstrap saves time to stylise the website.
2. jQuery is removed for Bootstrap 5 which, in turn, increased performance compared to when using Bootstrap 4.
3. Bootstrap 5 supports responsive font sizes.
4. Bootstrap Navbar has been optimised for Bootstrap 5.
5. We can integrate Bootstrap and Flask using the Flask-Bootstrap package.

Back-end frameworks used - Flask

Version number: 1.1.2

Reasons:

1. Flask is a microframework that enhances the performance of a web app.
2. Flask is better than Django in terms of performance and initial set up time.
3. v1.1.2 is the latest stable version of Flask.



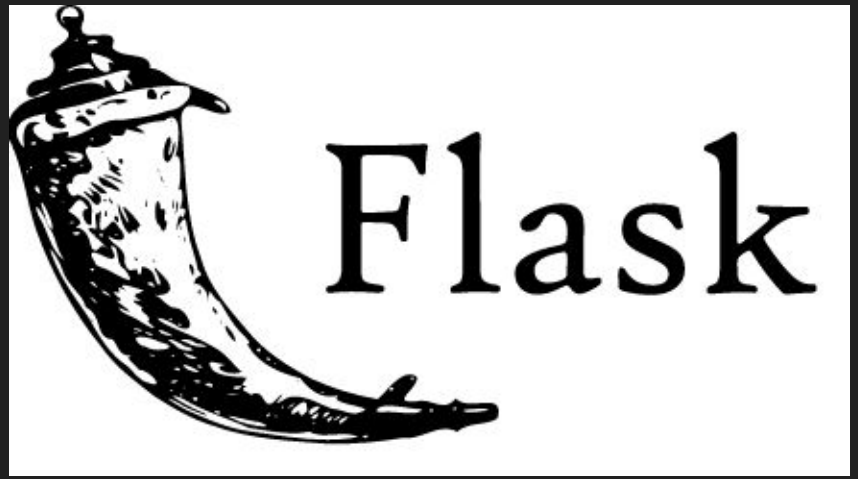
Libraries used - 1. Werkzeug

Version Number: 1.0.1

WerkZeug will be used to encrypt the passwords of all accounts created.

WerkZeug is developed by Flask.

v1.0.1 is the latest stable version of WerkZeug.



APIs used

1. Merriam-Webster's Learner's Dictionary with Audio (v3)
2. Merriam-Webster's Collegiate® Dictionary with Audio (v3)
3. Oxford Dictionaries API (v2.5)
4. Custom Search API (N/A)

3rd party packages used

1. beautifulsoup4 (4.9.3)
    a. Reason: to scrape results from web pages.
2. Flask-Bootstrap (3.3.7.1)
    a. To make html files more readable.
    b. To make the process of creating a new html page faster.
    c. To do server side rendering (SSR) effectively.
3. GoogleNews (1.5.7)
    a. Reason: to show results from Google News

3rd party packages used

4. gunicorn (20.1.0)
    a. Reason: to deploy the app using Heroku
5. PyDictionary (2.0.1)
    a. Reason: to show search results from WordNet and Synonym.com
6. PyMySQL (1.0.2)
    a. Reason: to connect with Amazon RDS
7. requests (2.25.1)
    a. Reason: to receive JSON files from various APIs.
8. udpy (2.0.0)
    a. Reason: to show search results from Urban Dictionary

3rd party packages used

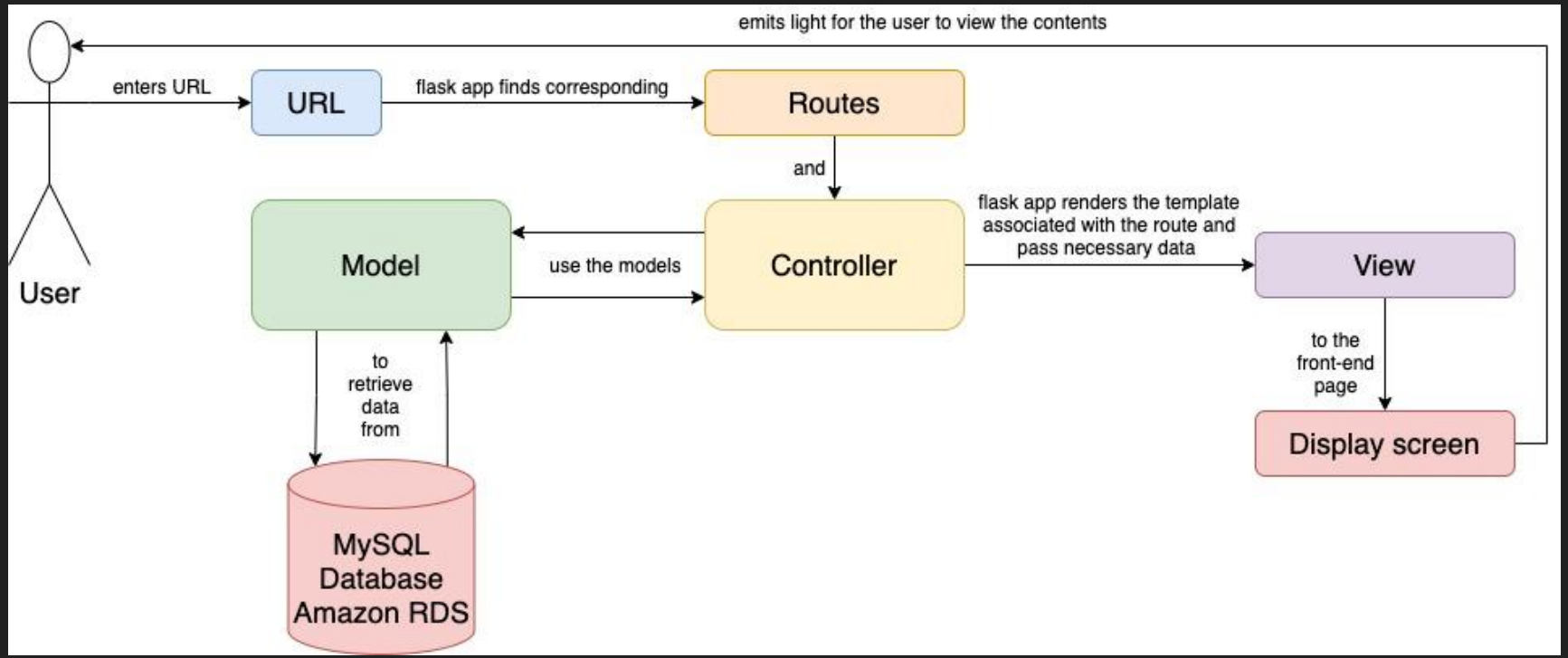
9. Wikipedia-API (0.5.4)
    a. Reason: to show search results from Wikipedia
10. wiktionaryparser (0.0.97)
    a. Reason: to show search results from Wiktionary
11. youtube-search-python (1.4.3)
    a. Reason: to show video results from YouTube

Software Design Patterns

We will follow the model-view-controller (MVC) design pattern and client-server pattern to implement our project.

1. If the user enters an URL, the flask app will find the corresponding route to the given URL.
2. The controller corresponding to the route will use the model to retrieve all necessary data from Amazon Relational Database Service (Amazon RDS) through MySQL queries.
3. The flask app will render the template associated with the route and pass the data to the front-end page if necessary.
4. Then, the user will be able to view the web page from its web browser.

MWC Diagram


emits light for the user to view the contents
enters URL
URL
flask app finds corresponding
Routes
and
User
Model
use the models
Controller
flask app renders the template associated with the route and pass necessary data
View
to the front-end page
Display screen
to retrieve data from
MySQL Database Amazon RDS

Database design and normalisation (BCNF)

Table name: Account

Attributes:

1. userID U
2. isAdmin A

Functional Dependencies:

There is no functional dependency.

Normalization:

The relation is BCNF.

Table Name: AdminAccount

Attributes:

1. fullName F
2. nickName K
3. userID U
4. password W
5. email E
6. phoneNumber N
7. jobTitle J
8. profilePictureURL P

Functional Dependencies:

1. U → FKUWENJP
2. N → FK
3. E → FK

Normalization:

The relation is BCNF.

Table Name: UserAccount

Attributes:

1. fullName F
2. nickName K
3. userID U
4. password W
5. email E
6. phoneNumber N
7. isPremium I
8. subscriptionEndDate D
9. profilePictureURL P

Functional Dependencies:

1. U → FKUWENIDP
2. N → FK
3. E → FK
4. D → I

Normalization:

The relation is BCNF.

Table name: SendFriendRequest

Attributes:

1. senderID S
2. receiverID R

Normalization:

There is no need to normalize this table as ‘SendFriendRequest’ represents a relationship between two entities as shown below.

UserAccount (Entity) - SendFriendRequest (Relationship) - UserAccount (Entity)

Table name: IsFriendWith

Attributes:

1. userID U
2. friendID F

Normalization:

There is no need to normalize this table as ‘SendFriendRequest’ represents a relationship between two entities as shown below.

UserAccount (Entity) - SendFriendRequest (Relationship) - UserAccount (Entity)

Table name: HasSearchHistory

Attributes:

1. userID U
2. order O
3. word W

Functional Dependencies:

UO → UOW

Normalization:

The relation is in BCNF.

Table name: HasSourceOrder

Attributes:

1. userID U
2. order O
3. source S

Functional Dependencies:

UO → UOS

Normalization:

The relation is in BCNF.

Table name: HasWordRanking

Attributes:

1. userID U
2. highestScore H

Functional Dependencies:

U → UH

Normalization:

The relation is in BCNF.

Table name: HasSentenceRanking

Attributes:

1. userID U
2. highestScore H

Functional Dependencies:

$U \rightarrow UH$

Normalization:

The relation is in BCNF.

Table name: TextChallenge

Attributes:

1. title I
2. textType T
3. challengeText C

Functional Dependencies:

I → ITC

Normalization:

The relation is in BCNF.

Table name: GenerateTextChallenge

Attributes:

1. creatorID C
2. title I
3. dateCreated D

Functional Dependencies:

CI → CID

Normalization:

The relation is in BCNF.

Table name: HasTextRanking

Attributes:

1. challengeTitle C
2. userID U
3. highestScore H

Functional Dependencies:

CU → CUH

Normalization:

The relation is in BCNF.

Table name: ChatRoom

Attributes:

1. roomName N
2. senderID S
3. receiverID R
4. chatMessage M
5. sendDate D

Functional Dependencies:

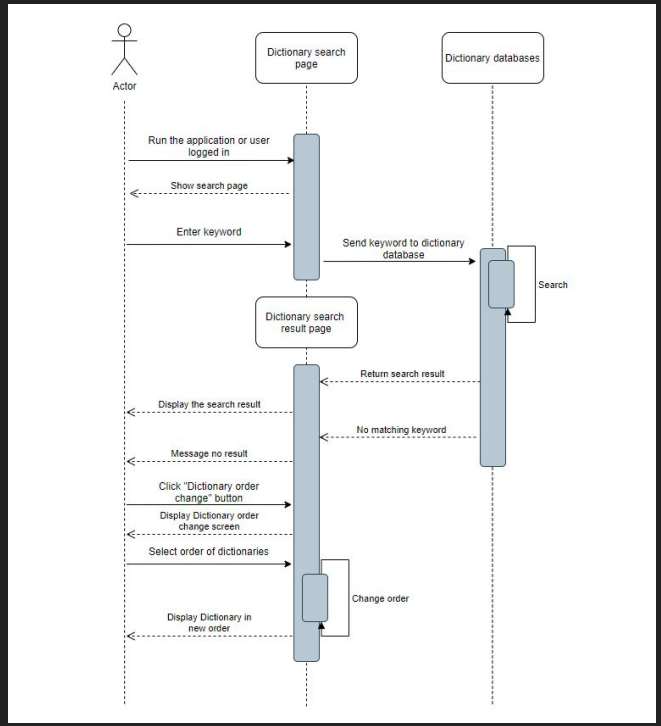
NSD → NSRMD

Normalization:

The relation is in BCNF.

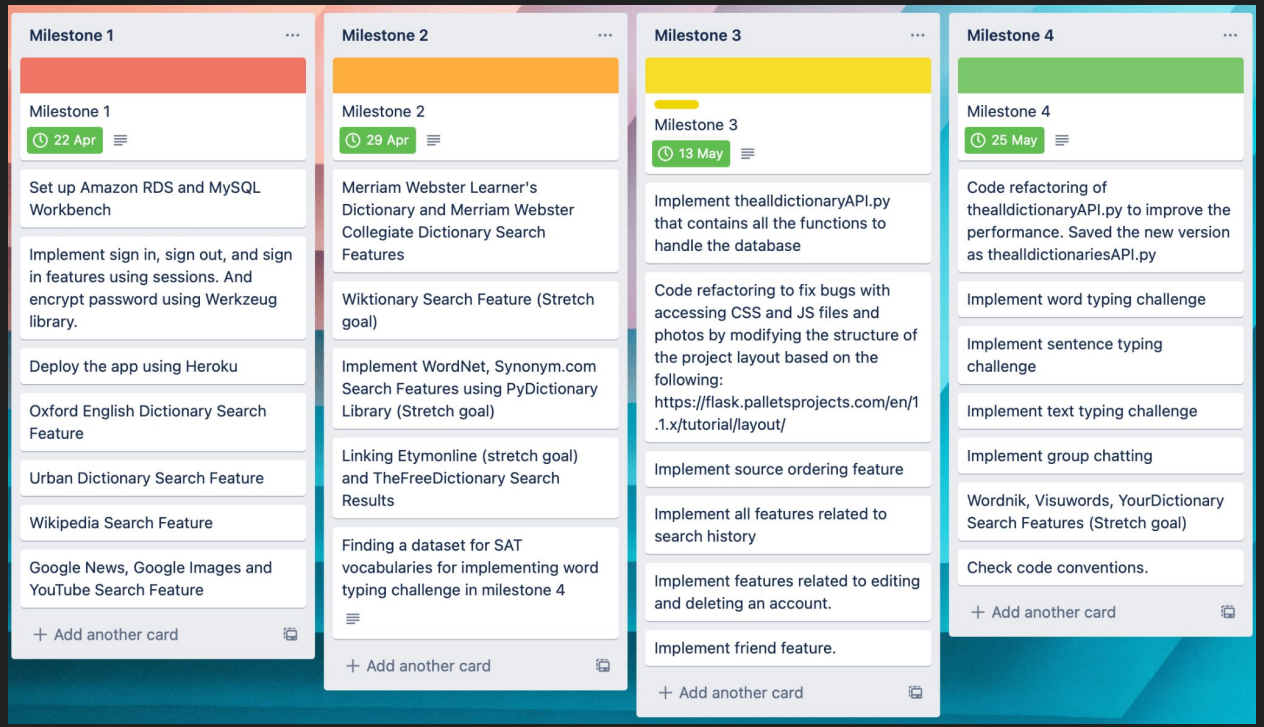
UML Sequence Diagram

UML Sequence Diagram - User looking up a word

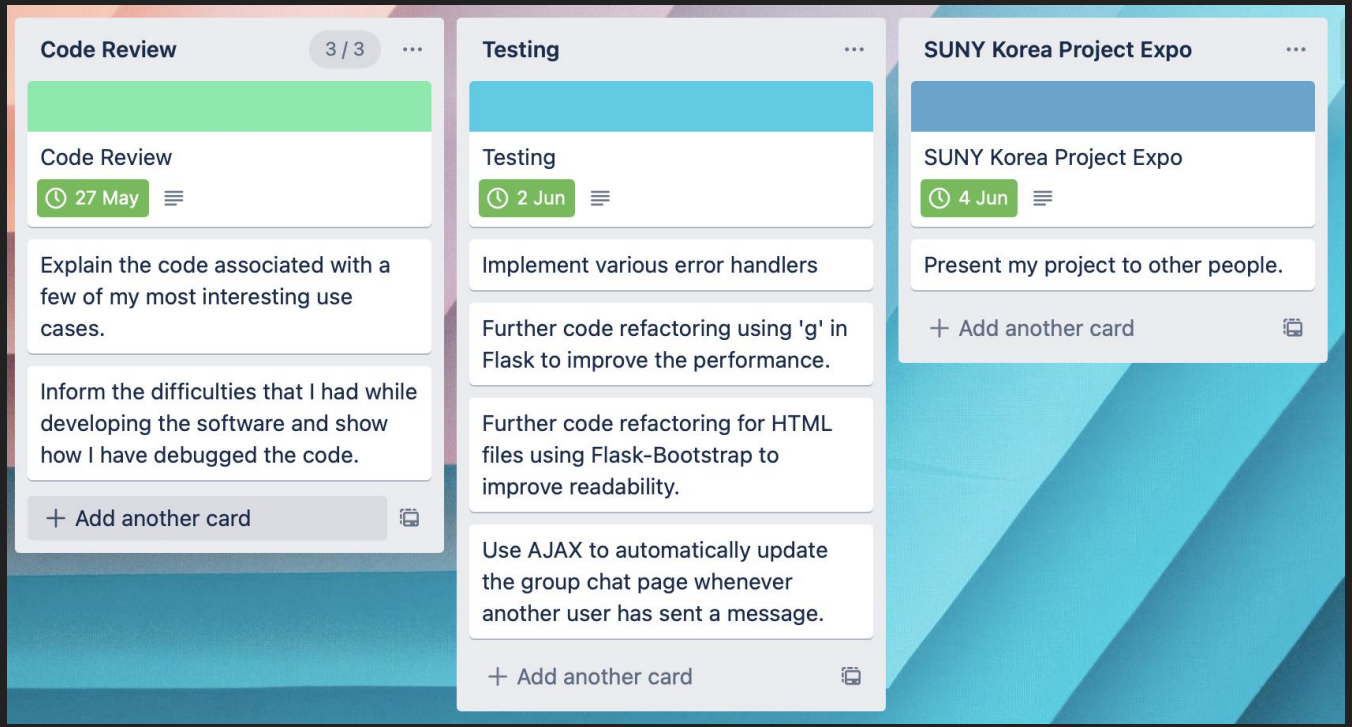


Schedule

Kanban board from milestone 1-4



Kanban board from code review to SUNYK Project Expo



Ask sth!