ASC
AI STUDY COMPANION

DESIGN DOCUMENT

Team Members:

Seth Morgan

Cameron Calhoun

Jonathan Buckel

Gage Keslar

Professor:

Dr. Weifeng Chen

Course:

Senior Project I: Software Engineering

University:

Pennsylvania Western University, California Campus

INSTRUCTORS NOTES:

Table of Contents

Abstract

This paper is layout the design requirements for the A.I Study Companion (ASC) project. ASC will be able to generate questions based on the users chosen subject of either math, history, geography, English, and computer science. The user will be able to generate formatted flashcards as well for each subject. The program will be able to track the users' progress and designate a skill level based on previous answers. Users will be able to track their process. All user's data will be stored locally, allowing the user to export or update data to and from ASC. This document will give the developer team an insight into how to build and manage this project as it describes the coupling and cohesion of individual modules, dataflow, design organization and the decision of which programming languages will be used.

Document Description

Purpose and Use

This document guidelines the developers to the required architecture to build this project to functionality. It has details of the different modules used, their functionality, cohesion / coupling to other modules, data flow, and other design specifications. The scope of the project is meant to be finished and presented before the end of the 2025 spring semester. This document will ensure a timeline and set expectations to meet deadlines.

Ties to Specification Document

The previous specifications document serves as an outline for what modules would be needed to complete the project. This document will flush out those modules in greater detail. The previous document was to give the client an understanding of how the program fits the criteria of their needs and how the developers planned to do so.

Intended Audience

The concept in this document is meant to aid the developers of the ASC project. As it will go into greater detail of the aspects of the project. Those who do not understand the technical side of software engineering stand to gain little benefit from reading. This will be meant to look back upon and serve as a guide when developing code, testing, and understanding the eventual completion of this project.

Project Block Diagram w/ Description

The ASC project will consist of three codebases. A front-end codebase that houses the web application the user will interact with, a back-end codebase that will handle all processes, including user requests and conversion of data from the LLM, and the LLM itself, a private codebase which will be prompted to generate questions and send them to users. The user will make a selection through the front-end to determine which state the program will enter. The back-end will look at the state the user has selected, and respond accordingly. If the selected process involves generating questions through use of the LLM, the back-end will then communicate with the LLM, and parse the data into a format that can be presented to the user. From there the user can respond to the changed state of the front-end, and the cycle repeats.

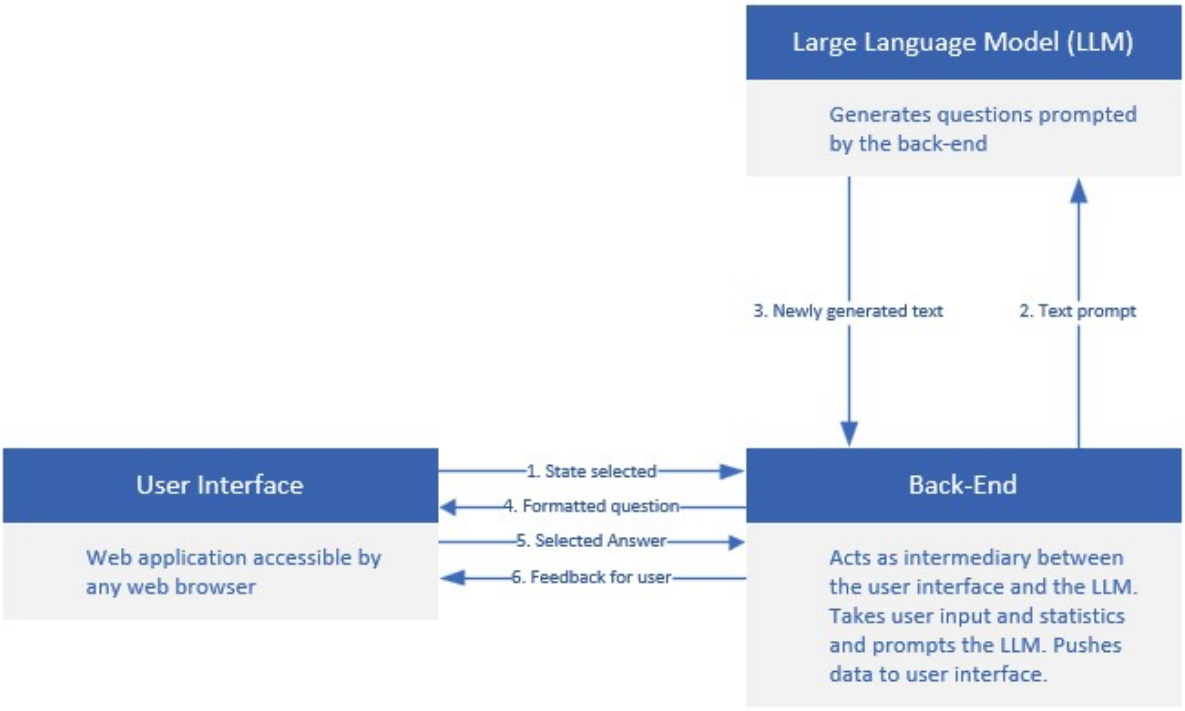


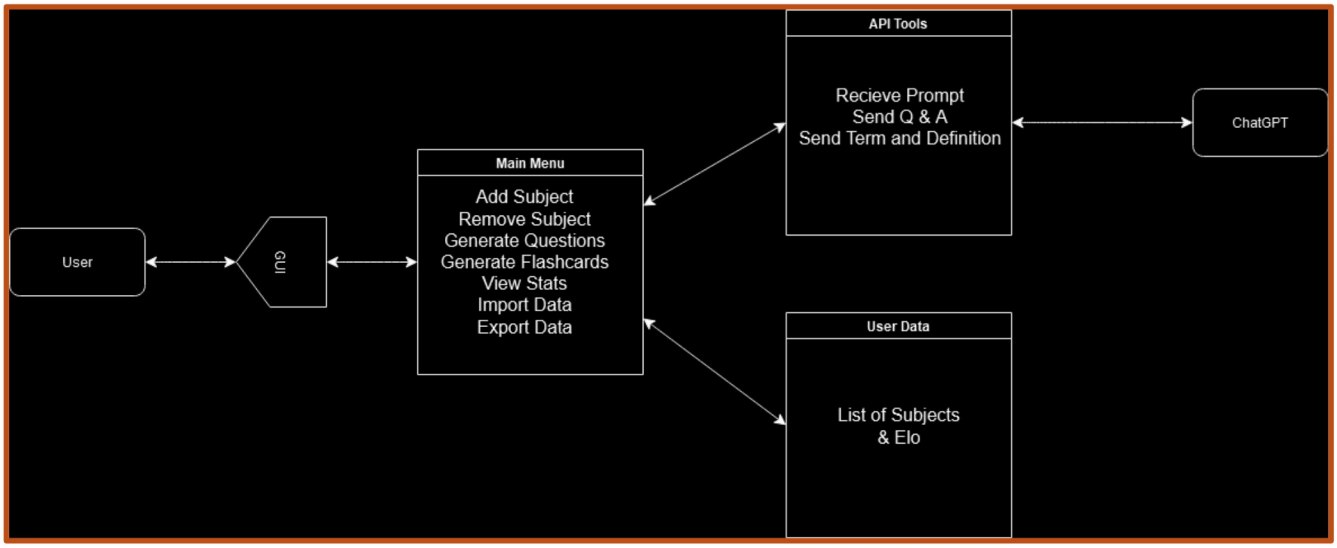
Figure 1: Block Diagram

Design Details

System Modules and Responsibilities

Architectural Diagram

Figure 2: Architectural Diagram



The architecture diagram (Fig. 2) shows the relationship between the user interacting with the graphical user interface (GUI). Based on their selection in the main menu will determine if there is to be any changes in the user data or to use the interactions of the API and ChatGPT. The API will be able to generate questions and answers or terms and definitions by use of the large language model (LLM). That data is parsed to fit their criteria as flashcards and questions and gave back to the user in their chosen form. User data will hold the users chosen subject and elo. The user may import and export their data through the main menu.

Module Cohesion

Cohesion between elements in the project is to be considered high as all elements are connected through the main menu. The user may enter any state by utilization of the main menu. Each state will feed data to and from each element which will return to the user based on their choices. This is shown in Figure 2, as the connection between all elements return to the main menu after diverging. For example, the use of the API tools being able to send data to ChatGPT, ChatGPT sending data back to API tools and then back to the main process once again. Each element fulfils a well-defined purpose and is closely related to each other (GeeksForGeeks, 2024).

Module Coupling

The interdependence between elements relies on ChatGPT to be able to produce questions and flashcards. While the ability to track subjects, subject's elo and the creation of prompts to feed into ChatGPT shows high levels of cohesion, the only way of accessing the LLM is using API tools. The creation of responses falls solely on ChatGPT's capabilities as well. To limit the potential of faults between the LLM and the rest of the project, there must be precise prompts and stable connection to the LLM at all times.

Design Analysis

Data Flow/Transaction Analysis

The dataflow of ASC is a repetitive process that can be represented as a non-terminating finite state machine. The user goes to ASC's website and starts in the initial state. Based on decisions made by the user, ASC will transition through its given states and sometimes interact with the LLM to generate questions. After completing the required processing in whichever state the program has transitioned to, ASC will then show the user the results of this processing (E.G., the displayed skill data, a generated question, a success message for adding or removing subjects, etc.), and transition back to its initial state to wait for the next input.

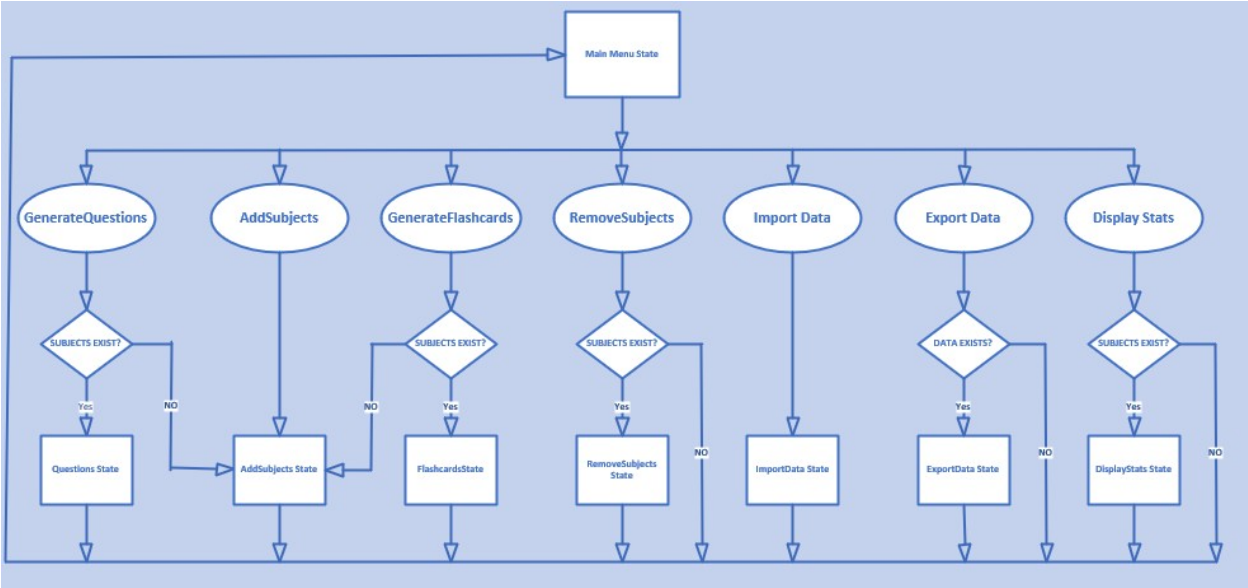


Figure 3: Data Flow Diagram

Design Organization

Detailed Tabular Description of Classes

Class name: Subjects

Class description: The Subject class is a parent class that houses all generic methods and attributes applicable to all classes. This class will not be instantiated on its own but serves as a blueprint for every subject that inherits it.

Class data members: subjectName, subjectElo, subjectBreakpoints subjectPrompts

Class member functions: setSubjectName(), getSubjectName(), setSubjectElo(), getSubjectElo(), setSubjectBreakpoints(), getSubjectBreakpoints(), setSubjectPrompts(), getSubjectPrompts()

Class name: English

Class description: The English class is a derived class of Subjects. This allows English to house all methods and attributes defined in Subjects, as well as override them to fit the needs of this specific subject. English will be an instantiable class that can be stored in the users record of tracked subjects.

Class data members: subjectName, subjectElo, subjectBreakpoints subjectPrompts

Class member functions: setSubjectName(), getSubjectName(), setSubjectElo(), getSubjectElo(), setSubjectBreakpoints(), getSubjectBreakpoints(), setSubjectPrompts(), getSubjectPrompts()

Class name: History

Class description: The History class is a derived class of Subjects. This allows History to house all methods and attributes defined in Subjects, as well as override them to fit the needs of this specific subject. History will be an instantiable class that can be stored in the users record of tracked subjects.

Class data members: subjectName, subjectElo, subjectBreakpoints subjectPrompts

Class member functions: setSubjectName(), getSubjectName(), setSubjectElo(), getSubjectElo(), setSubjectBreakpoints(), getSubjectBreakpoints(), setSubjectPrompts(), getSubjectPrompts()

Class name: Geography

Class description: The Geography class is a derived class of Subjects. This allows Geography to house all methods and attributes defined in Subjects, as well as override them to fit the needs of this specific subject. Geography will be an instantiable class that can be stored in the users record of tracked subjects.

Class data members: subjectName, subjectElo, subjectBreakpoints subjectPrompts

Class member functions: setSubjectName(), getSubjectName(), setSubjectElo(), getSubjectElo(), setSubjectBreakpoints(), getSubjectBreakpoints(), setSubjectPrompts(), getSubjectPrompts()

Class name: ComputerScience

Class description: The ComputerScience class is a derived class of Subjects. This allows ComputerScience to house all methods and attributes defined in Subjects, as well as override them to fit the needs of this specific subject. ComputerScience will be an instantiable class that can be stored in the users record of tracked subjects.

Class data members: subjectName, subjectElo, subjectBreakpoints subjectPrompts

Class member functions: setSubjectName(), getSubjectName(), setSubjectElo(), getSubjectElo(), setSubjectBreakpoints(), getSubjectBreakpoints(), setSubjectPrompts(), getSubjectPrompts()

Class name: Math

Class description: The Math class is a derived class of Subjects. This allows Math to house all methods and attributes defined in Subjects, as well as override them to fit the needs of this specific subject. Math will be an instantiable class that can be stored in the users record of tracked subjects.

Class data members: subjectName, subjectElo, subjectBreakpoints subjectPrompts

Class member functions: setSubjectName(), getSubjectName(), setSubjectElo(), getSubjectElo(), setSubjectBreakpoints(), getSubjectBreakpoints(), setSubjectPrompts(), getSubjectPrompts()

Class name: APITools

Class description: APITools is a utility class defined to house all utility functions for interacting with an LLM through various APIs. This will allow for clean reusable code, improve the readability of the program, and significantly help the writability of ASC.

Class data members: N/A

Class member functions: generateQuestion(), generateFlashcards(), solveMathEquation()

Class name: User

Class description: The User class will house a given users tracked subjects, as well as functionality pertaining to user information.

Class data members: trackedSubjects

Class member functions: addSubject(), removeSubject(), importData(), exportData()

Functional Description

Subjects Class

setSubjectName()

Input:

setSubjectName() takes a string representing the name of the subject.

Output:

setSubjectName() will set the value of subjectName with the input string.

Return Parameters:

The only values to be returned would be exceptions thrown during error checking.

Types:

Strings

getSubjectName()

Input:

getSubjectName() takes no input values.

Output:

getSubjectName() outputs the string stored in subjectName.

Return Parameters:

getSubjectName() will return the string stored in subjectName, and an exception if the value in subjectName is empty.

Types:

Strings

setSubjectElo()

Input:

setSubjectElo() takes the integer value the subjects Elo should be set to.

Output:

setSubjectElo() takes the integer value the subjects Elo should be set to.

Return Parameters:

The only values to be returned would be exceptions thrown during error checking.

Types:

Integers

getSubjectElo()

Input:

setSubjectElo() takes no input values.

Output:

getSubjectElo() outputs the integer stored in subjectElo.

Return Parameters:

getSubjectElo() will return the value stored in subjectElo, as well as an exception if the current subject does not have a value for subjectElo

Types:

Integers

setSubjectBreakpoints()

Input:

Integer values to be inserted into the dictionary subjectBreakpoints.

Output:

subjectBreakpoints updated with new breakpoint values

Return Parameters:

setSubjectBreakpoints() will have no return parameters other than exceptions thrown during error checking.

Types:

Integers, Dictionaries

getSubjectBreakpoints()

Input:

getSubjectBreakpoints() takes no input values.

Output:

getSubjectBreakpoints() outputs the breakpoint a user is on for the specified subject.

Return Parameters:

The integer value referenced by the input key.

Types:

Integers, Dictionaries

setSubjectPrompts()

Input:

setSubjectPrompts() takes a string, and an integer, representing the subject prompt.

Output:

subjectPrompts updated with new prompt.

Return Parameters:

No return values other than exceptions thrown during error checking.

Types:

Strings, Dictionaries

getSubjectPrompts()

Input:

getSubjectPrompts() takes no input values.

Output:

getSubjectPrompts() outputs a prompt that fits for the subject's current elo.

Return Parameters:

String value contained within the subjectPrompts dictionary.

Types:

Strings, Dictionaries

English Class

setSubjectName()

Implementation same as in parent class Subjects.

getSubjectName()

Implementation same as in parent class Subjects.

setSubjectElo()

Implementation same as in parent class Subjects.

getSubjectElo()

Implementation same as in parent class Subjects.

setSubjectBreakpoints()

Implementation same as in parent class Subjects.

getSubjectBreakpoints()

Implementation same as in parent class Subjects.

setSubjectPrompts()

Implementation same as in parent class Subjects.

getSubjectPrompts()

Implementation same as in parent class Subjects.

History Class

setSubjectName()

Implementation same as in parent class Subjects.

getSubjectName()

Implementation same as in parent class Subjects.

setSubjectElo()

Implementation same as in parent class Subjects.

getSubjectElo()

Implementation same as in parent class Subjects.

setSubjectBreakpoints()

Implementation same as in parent class Subjects.

getSubjectBreakpoints()

Implementation same as in parent class Subjects.

setSubjectPrompts()

Implementation same as in parent class Subjects.

getSubjectPrompts()

Implementation same as in parent class Subjects.

Geography Class

setSubjectName()

Implementation same as in parent class Subjects.

getSubjectName()

Implementation same as in parent class Subjects.

setSubjectElo()

Implementation same as in parent class Subjects.

getSubjectElo()

Implementation same as in parent class Subjects.

setSubjectBreakpoints()

Implementation same as in parent class Subjects.

getSubjectBreakpoints()

Implementation same as in parent class Subjects.

setSubjectPrompts()

Implementation same as in parent class Subjects.

getSubjectPrompts()

Implementation same as in parent class Subjects.

ComputerScience Class

setSubjectName()

Implementation same as in parent class Subjects.

getSubjectName()

Implementation same as in parent class Subjects.

setSubjectElo()

Implementation same as in parent class Subjects.

getSubjectElo()

Implementation same as in parent class Subjects.

setSubjectBreakpoints()

Implementation same as in parent class Subjects.

getSubjectBreakpoints()

Implementation same as in parent class Subjects.

setSubjectPrompts()

Implementation same as in parent class Subjects.

getSubjectPrompts()

Implementation same as in parent class Subjects.

Math Class

setSubjectName()

Implementation same as in parent class Subjects.

getSubjectName()

Implementation same as in parent class Subjects.

setSubjectElo()

Implementation same as in parent class Subjects.

getSubjectElo()

Implementation same as in parent class Subjects.

setSubjectBreakpoints()

Implementation same as in parent class Subjects.

getSubjectBreakpoints()

Implementation same as in parent class Subjects.

setSubjectPrompts()

Implementation same as in parent class Subjects.

getSubjectPrompts()

Implementation same as in parent class Subjects.

APITools Class

generateQuestion()

Input:

generateQuestion() takes a Subject as an input parameter.

Output:

generateQuestion() outputs a question to a user, allows them to answer it, and updates the subjects Elo rating.

Return Parameters:

This method will have no return values, other than exceptions thrown from error checking.

Types:

Subjects

generateFlashcards()

Input:

generateFlashcards() takes a Subject as an input parameter.

Output:

generateFlashcards() outputs a series of flashcards to a user and provides them with a printable document

Return Parameters:

This method will have no return values, other than exceptions thrown from error checking.

Types:

Subjects

solveMathEquation()

Input:

solveMathEquation() takes a string as an input parameter.

Output:

solveMathEquation() outputs an answer to a math question, for use in formatting a math question generated.

Return Parameters:

solveMathEquation() returns a string containing the solution to the math problem provided, along with exceptions thrown during error checking.

Types:

Strings

User Class

addSubject()

Input:

addSubject() takes a Subject as an input parameter.

Output:

addSubject() outputs an updated trackedSubjects list that holds the newly added subject.

Return Parameters:

This method returns nothing other than exceptions thrown from error checking.

Types:

Subjects, Lists

removeSubject()

Input:

removeSubject() takes a Subject as an input parameter.

Output:

removeSubject() outputs an updated trackedSubjects list with the specified Subject removed

Return Parameters:

This method returns nothing other than exceptions thrown from error checking.

Types:

Subjects, Lists

importData()

Input:

importData() takes no input parameters.

Output:

importData() outputs a prompt which allows user to import saved data from a JSON file.

Return Parameters:

This method returns nothing other than exceptions thrown from error checking.

Types:

Object literals

exportData()

Input:

exportData() takes no input parameters.

Output:

exportData() outputs a JSON file which contains the information stored in trackedSubjects.

Return Parameters:

This method returns nothing other than exceptions thrown from error checking.

Types:

Object literals

Narrative/ PDL

Main Menu:

1. Add Subjects
2. Generate Questions
3. Generate Flashcards
4. Remove Subjects
5. Import Data
6. Export Data
7. Display Stats

Subjects:

1. Add Subject
    a. Select from English, Math, Geography, Computer Science or History.
    b. Is selected subject not on list?
        i. No? - Add Subject to user data
        ii. Yes? - Do not add subject
2. Remove Subject
    a. Select subject listed above.
    b. Is subject on the list?
        i. No? - Do not process.
        ii. Yes? - Remove subject and elo

Generate Questions / Flashcards:

1. Select Subject
    a. Does subject exist?
        i. No? - Go back to main menu
        ii. Yes? - Generate Questions / Flashcards

    Create LLM prompt
    a. Gather Subject
    b. Subject's Current Elo
    c. Insert both into prompt

    Prompt ChatGPT
    a. Send prompt to ChatGPT
    b. Receive response from ChatGPT
    c. Parse response to designated feature
2. Generate Question
    a. Display message and multiple choice questions to user
    b. User selects answer
        i. Correct Answer? - Adjust Elo positively
        ii. Incorrect Answer? - Adjust Elo negatively.
3. Generate Flashcards
    a. Create PDF document for user download

Import / Export Data

1. Import

a. Import user data

2. Export

a. Does user data exist?

i. No? - Ignore, go to main menu

ii. Yes? - Export User JSON File for download.

View Stats:

1. Does user data exist?

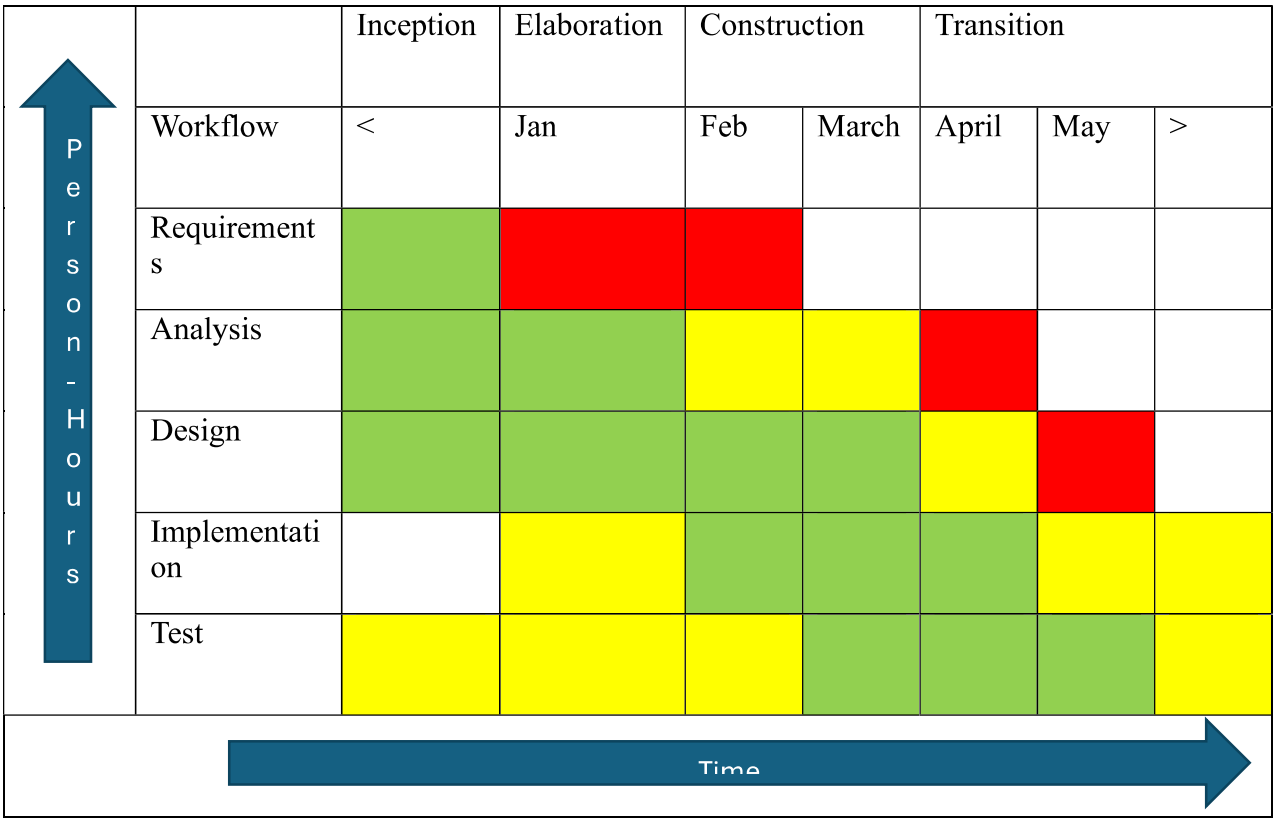
a. No? - Ignore, return to main menu.

b. Yes? - Display to GUI.

Decision

Our backend will be written with Python contained in a Docker Image, frontend with JavaScript with the React Framework. Since our project will be a webpage, JavaScript has been chosen to create the GUI with React framework objects. JavaScript and Docker allows our Python code to be contained within the webpage. The user will not have to download Python in order for the system to work since it will be contained in the image. The user will only have to have access to a web browser in order to begin using the product. The interactions between the GUI executes the associated Python code.

Implementation Timeline



Key: Green – High Priority, Yellow – Medium Priority, Red – Low Priority, Blank – No Priority

Design Testing

Testing will be an ongoing process during all steps of the development. This is to ensure that the team will finish completely and with as little faults as possible. This will be done with regular static and dynamic testing methods discussed in class. Using Git as a source control management system, we can conduct peer reviews on code before committing and finalizing code. Faults to be found when static testing will include unused variables, infinite loops, dead code, etc. Each individual module will also be dynamically tested to ensure that all class methods and attributes execute correctly with set testing data. Analyzing the output will determine faults to be addressed. For instance, when testing our APITools class for receiving output from the LLM, we want to make sure it will produce a response that we can parse and divide into

questions and answers. If the LLM was to give no response, we know that the input stream needs to be checked. If the LLM was to produce bad data, we know that the prompt output must be changed. After these tests conclude to no faults, the developers will be able to commit the changes to the Git repository and development on the next module may begin.

Resources

GeeksforGeeks. (2024, October 4). *Coupling and cohesion - software engineering*. https://www.geeksforgeeks.org/software-engineering-coupling-and-cohesion/

Schach, S. R. (2011). *Object-oriented and classical software engineering*. McGraw-Hill.

Appendix A: Team Details

Gage Keslar contributed to the following sections in this document:

- Design Details
    - Architecture Diagram
    - Module Cohesion
    - Module Coupling
- Design Organization
    - Narrative/PDL
- Decision
- Timeline

- Design Testing
- Writing Center

Cameron Calhoun contributed to the following sections in this document:

- Project Block Diagram
- Design Analysis
    - Data Flow Analysis
- Design Organization
    - Detailed Tabular description of classes
    - Functional Description

Seth Morgan contributed to the following sections in this document:

- Abstract
- Document Description
    - Purpose and Use
- Formatting & Editing
- Additional Research

Jonathan Buckel contributed to the following sections in this document:

- Document Description
    - Ties to Specification Document
    - Intended Audience
- Formatting & Editing
- Additional Research

Appendix B: Writing Center Report

Dr. Chen,

Hello, my name is Emily Esposito and I am a writing consultant for the Foundry Writing Center. At 2:00 Monday afternoon, I had an appointment with Gage Keslar to receive feedback for their Senior Project for the CMSC 4900 class.

They wanted to focus on overall grammar and mechanics. We found some later-order concerns such as consistent capitalization issues, and making sure non-proper nouns are not capitalized. Many of the things we discussed in our previous meetings were not a concern today, which shows their improvement. There was a question regarding the use of proper sentence structure for a particular section, so I advised they check with the sample assignment as well as with the professor to gain clarity. With that being said we did work through an example or two on how to correct it if that was an issue. The only other issue we found was a minor issue of overexplaining when trying to give a list. I suggested that their group describes the list and breaks down the other content in the following sentences for clarity.

If you have any other questions about our meeting, please do not hesitate to contact me at esp2609@pennwest.edu.

Sincerely,

Emily Esposito
Pennsylvania Western University California
Secondary Education Biology, *Junior*
Alpha Lambda Delta Honors Society, *Member*
Vulcan Dance Team, Vice President
University Choir, Secretary
Foundry Writing Center, *Lead Consultant*

Appendix C: Workflow Authentication

I, Gage Keslar, agree that I have performed all actions specified and to adhere to the design specifications laid out in this document.

Signature:  Date: 12/10/2024

I, Cameron Calhoun, agree that I have performed all actions specified and to adhere to the design specifications laid out in this document.

Signature: Cameron Calhoun Date: 12/10/2024

I, Seth Morgan, agree that I have performed all actions specified and to adhere to the design specifications laid out in this document.

Signature: Seth Morgan Date: 12/10/2024

I, Jonathan Buckel, agree that I have performed all actions specified and to adhere to the design specifications laid out in this document.

Signature: Jonathan Buckel Date: 12/10/2024