Easy A

Software Design Specification

Group 4 - 2-24-2023 - v0.03

Garrett Bunkers, Alder French, Connor Maclachlan, Juan Rios, Blake Skelly

This file is the Software Design Specification for University of Oregon CS 422 Project 1 by Group 4. The EasyA system is made to help students view UO class data in a way that may be beneficial to their GPA by providing useful options to view the data such as with a graph that shows what percentage of students passed a class under different instructors. This document provides details about the system's design, including its architecture and intended use cases.

Table of Contents

1. SDS Revision History

Date	Author	Description
1-15-2023	Group 4	Created the initial document from template, each team member contributed to filling out the sections outlined in the project criteria for the first deliverable.
2-02-2023	Connor M	Updated system overview. Removed irrelevant modules that did not make the final deliverable.
2-03-2023	Alder	Updated Diagrams for Software Architecture, as well as each module according to Software Architecture Modeling described in chapter 6 of “Sommerville (2015) Software Engineering” and the UML notation outlined in the “UML_Kieras.pdf” on the class website.
2-04-2023	Alder	Updated Software Architecture and Module “Design Rationale”s, as well as rewrote the “List of Components” and “Module Interaction” in the Software Architecture section.

2. System Overview

The system consists of 3 major parts: a graphical user interface (GUI), a local data storage, and a data scraper. A simple GUI will be used to accept input from the user and display the graphs of the requested data. The GUI will also allow a system admin user to alter the existing data. This data will be kept in a local storage file that was retrieved by the data scraper. The GUI will have its own built-in data controller, updating the data in the local storage file and moving data around as it needs. The data scraper is used to retrieve the desired external data from the UO class webpage. Overall, the system will allow for easy retrieval, viewing, and management of the desired class data.

3. Software Architecture

Software Architecture

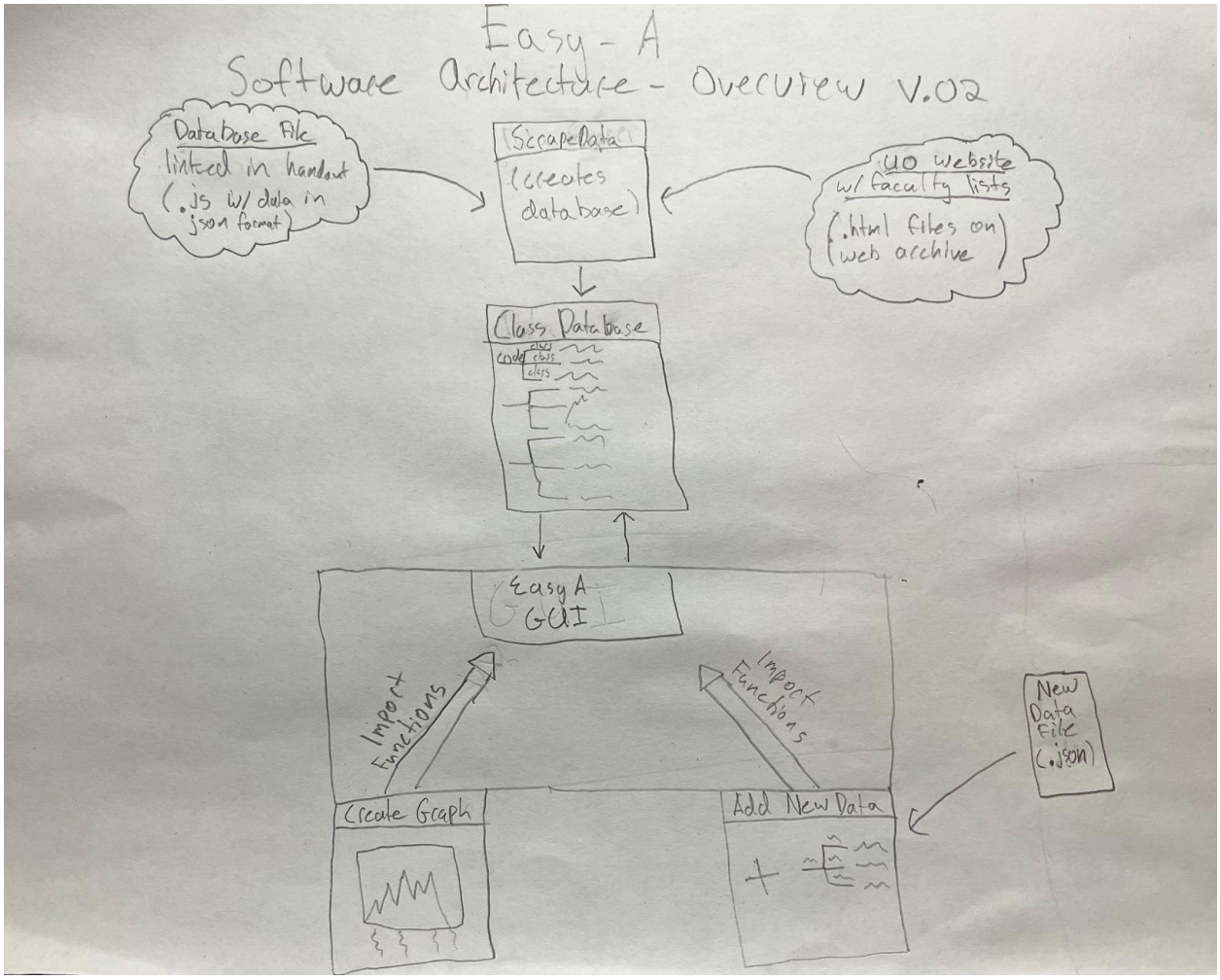
1. List of Components (*Figure 1.*)
    a. Database File from Handout - the UO class data embedded on the webpage ending in .js on the project handout
    b. Scrape Data (module)- Python script that scrapes data from the UO class data webpage and puts it in a .json file that contains all the class data in JSON format. Then it scrapes

the old UO website on the wayback machine to get the names of the “Regular Faculty” in the natural science departments. Note that it uses a python dict where each class code is a key to internally manage the class data

c. Class Database - a .json file that contains all the class data in “a.” in JSON format
d. EasyA GUI (module) - a Python script that contains the code for the interface of our EasyA system. The interface has 2 main options that open 2 different sub interfaces. 1 for students to look at graphs of the class data in “c.”, and 1 for a system administrator to update the class data in the database file (“c.”) with new data. Note that it uses a python dict where each class code is a key to internally manage the class data, like in “b.”
e. New Data File - a .json file that contains example new class data, with regular faculty marked, for the purpose of demonstrating how one can add new class data to the database

2. Module Interaction
    a. First our system uses its Data Scraper (*Figure 1.*) to get the original UO class data from the .js webpage given in the Project handout, and put it in a .json file that contains the class data in JSON format. This .txt file will act like a database for our system, and it will be stored in our github repo. Once the Data Scraper is finished running, the EasyA GUI will read this .json file and use it to create an instance of a python dictionary where every class code is a key at runtime. That way it can easily access the class data in the dictionary to display graphs and class data as the user desires. Note that the GUI itself has both the function to update the data in the database given a new file as well as the function to view graphs of the data in the database, the latter of which is imported from another file containing the functions to create the graphs.

3. Architecture Diagram - *Figure 1.*



4. Design Rationale
    a. We would like to keep our system's design as simple as possible, that way we can put more time into designing a user-friendly interface. That's why we opted for using a simple .json file in JSON format as a database for the class data instead of an actual Mongo or MySQL database. This simplicity-first strategy carries over into the other parts of our system architecture, and is why we chose to make only 2 modules. One to prepare the class database, and another to view and update it. It is also why we chose to prevent the class data scraper and GUI from direct communication, they only work together via the class database.

4. Software Modules

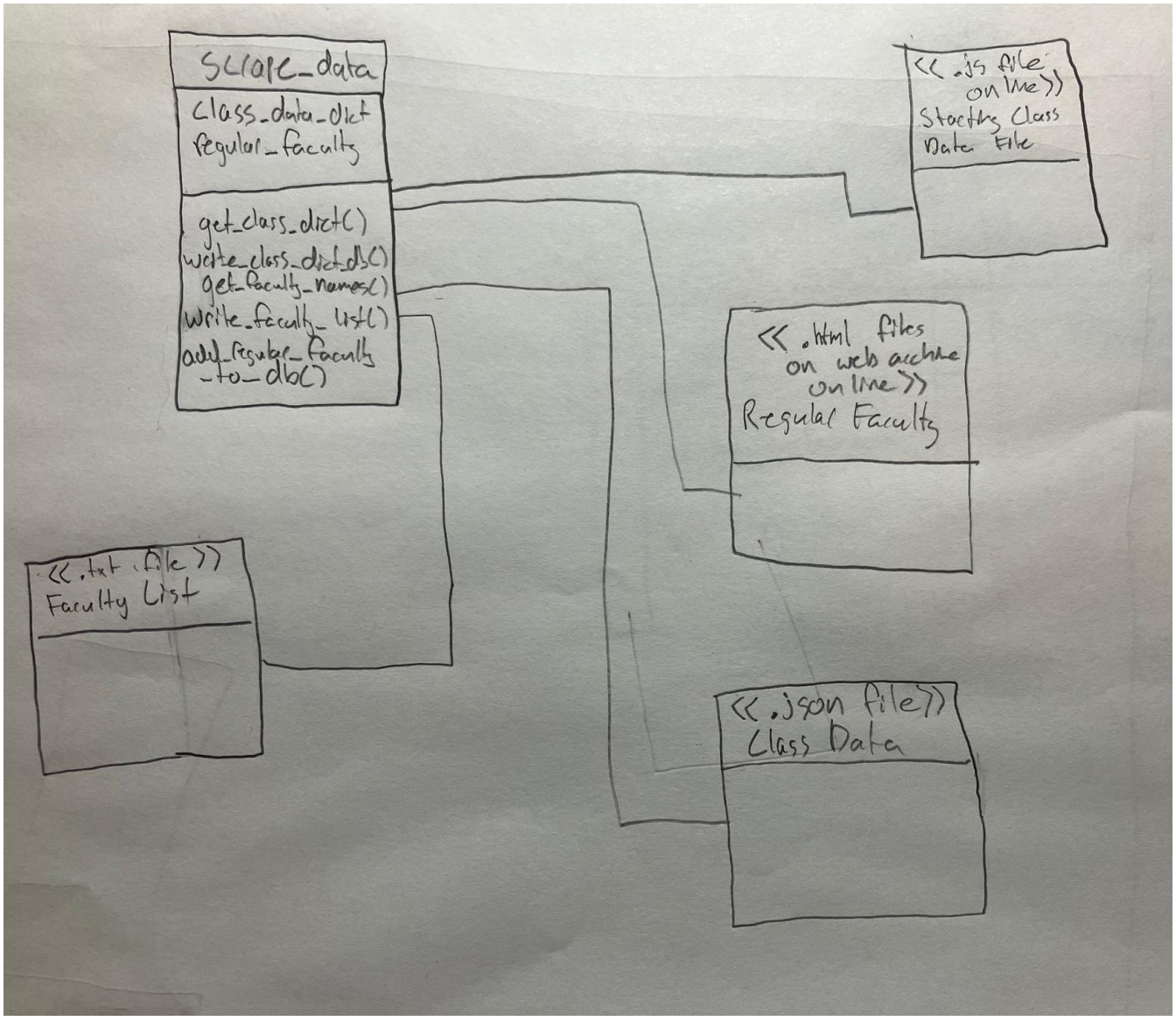
4.1. ScrapeData

a. Role & Primary Function

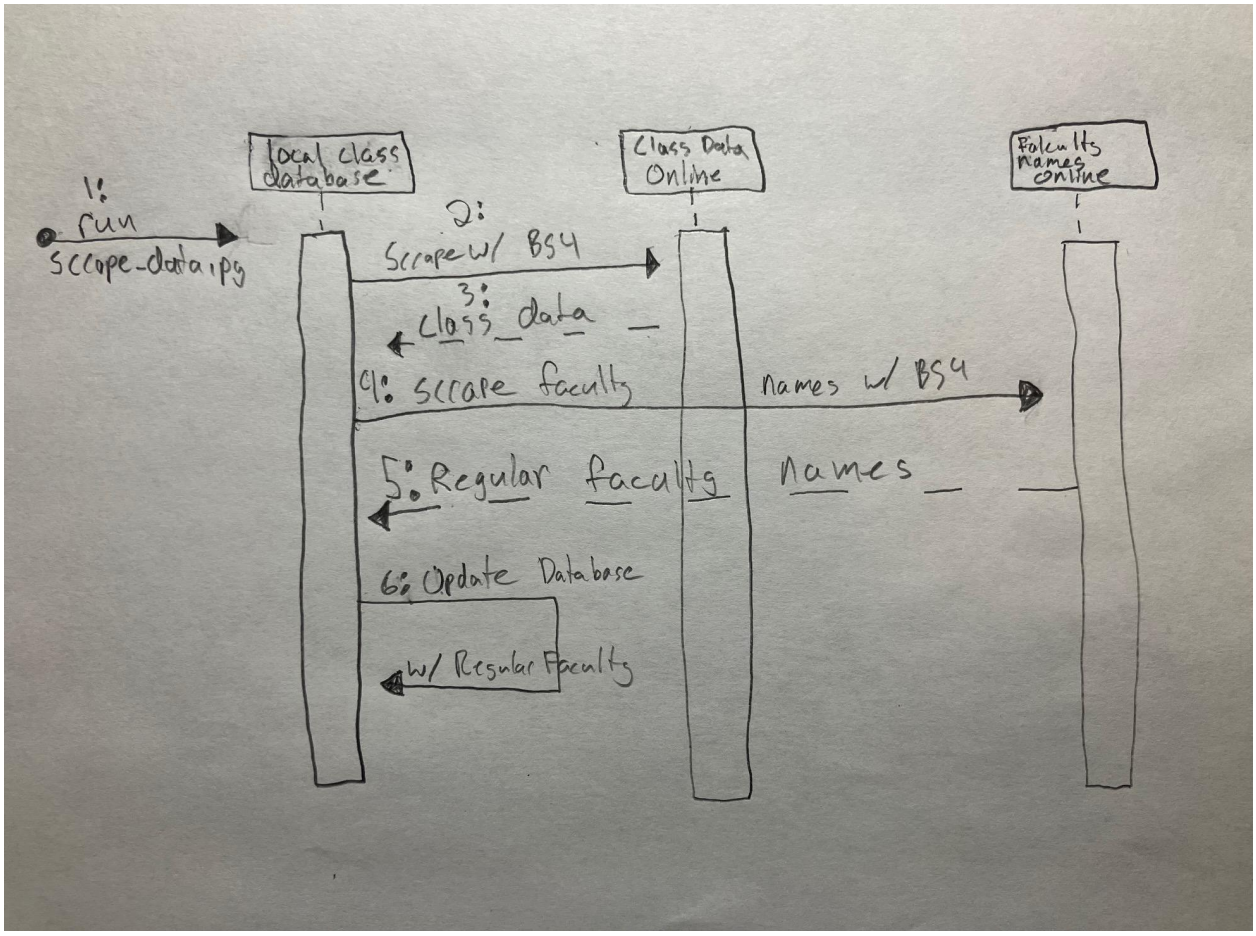
i. Retrieves class data from the webpage and creates a dictionary that is written into a .json file. Then it scrapes the UO natural science department's regular faculty names from the old UO website on the wayback machine, and uses them to modify the database by adding an indicator to each class element that indicates whether or not its instructor is a regular faculty member. Once this is finished the EasyA GUI is ready to run and use the class database to generate graphs.

b. Interface to other Modules
   i. Does not directly interface with any other module. Instead it extracts the grade data from the webpage given in the project handout and saves it to a .json file in our project's directory. This text file is accessed by the other modules and acts as our projects database for grade data.

c. Static Model - *Figure 2. (Note: scrape_data.py contains no Python classes as it is not intended for use by other files, but run beginning to end, this class model is an abstraction of its behavior)*



d. Dynamic Model - *Figure 3.*

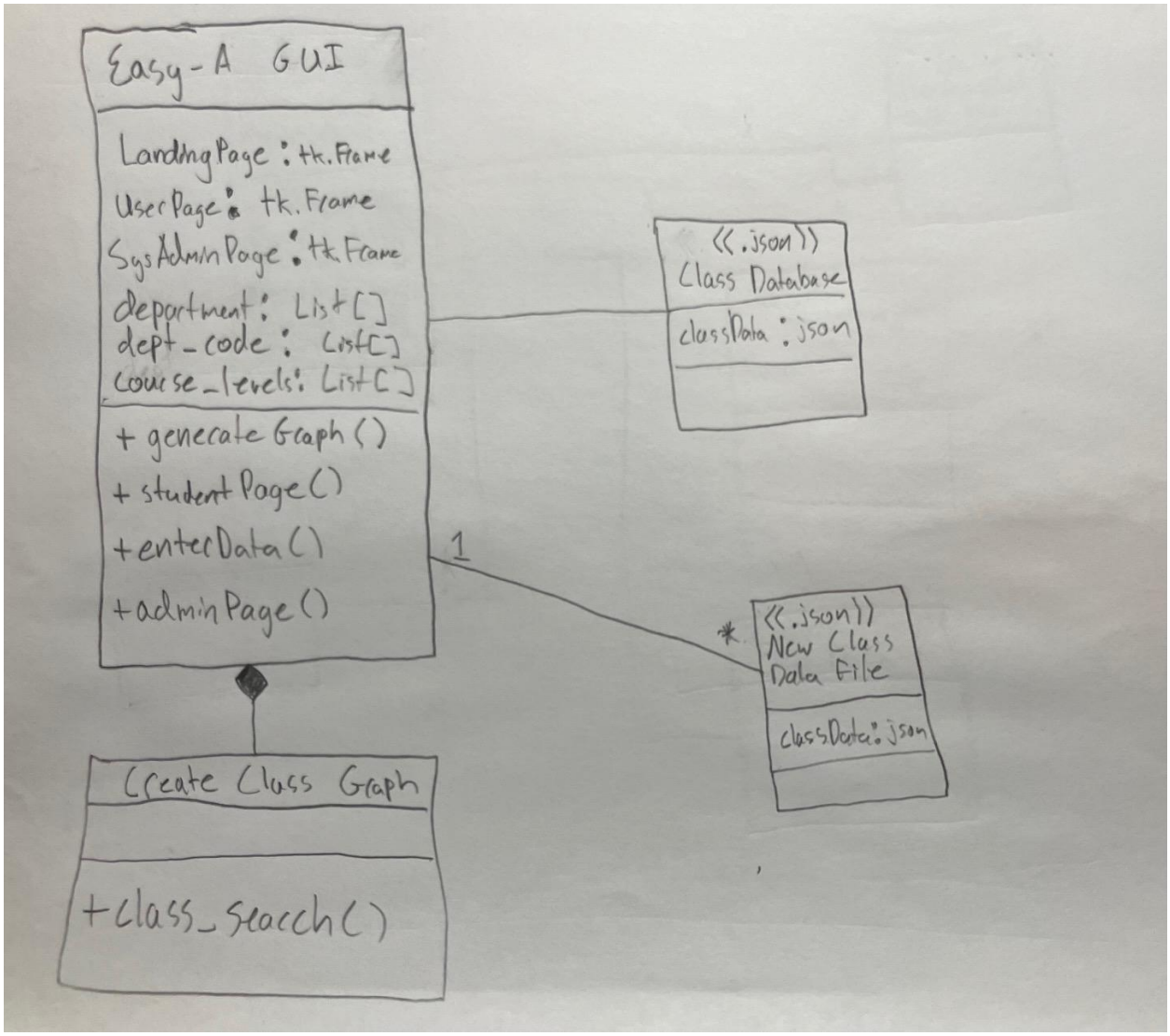


e. Design Rationale
   i. ScrapeData was designed to simply scrape the class data and regular faculty names from online and use them to create a database with all the necessary class information in .json format. This .json file will then act as a database for our project. Note that ScrapeData does not directly interface with any other modules, it only interacts with them via the .json class database. This is intentional, as ScrapeData must run "like a script" i.e. it runs sequentially, and does not interact with directly with the other modules

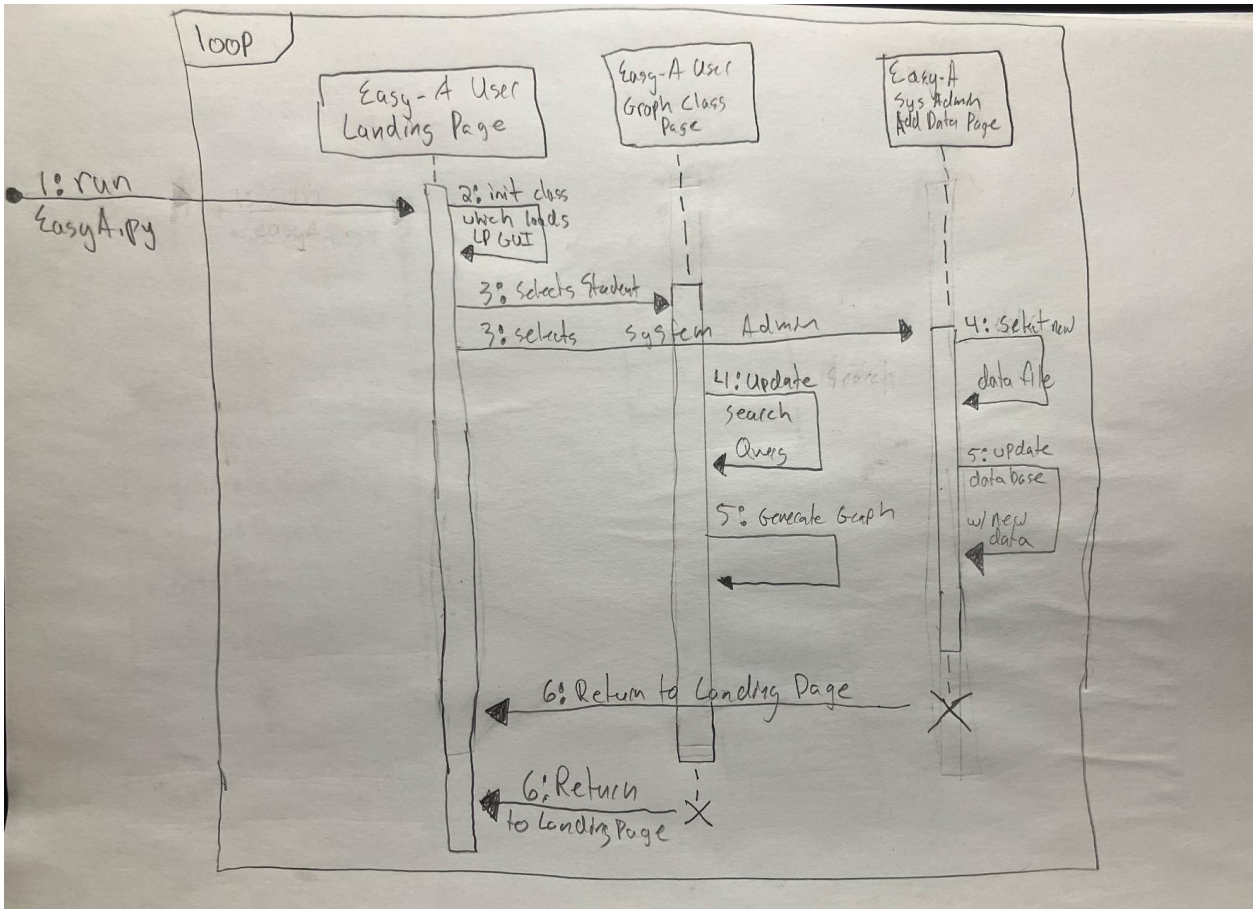
4.2. EasyAInterface

a. Role & Primary Function
   i. Has 2 primary functions: 1. Retrieves specific data from the class data file and displays bar graphs. 2. Provides an interface for a system administrator to update the class database with new data given a .json file with new class data.
b. Interface to other Modules
   i. EasyAInterface interacts with the class data json file by reading it and storing data in a dictionary for making graphs. It also updates the class database .json file given a .json file with new class data.

c. Static Model - *Figure 4.*



d. Dynamic Model - *Figure 5.*



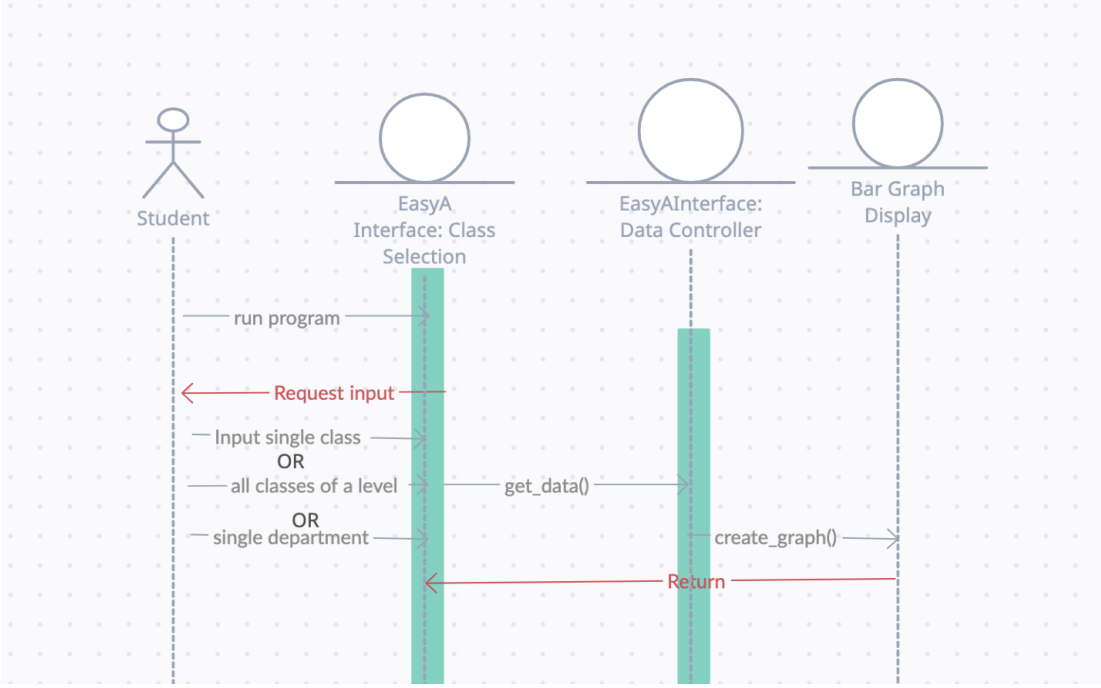
e. Design Rationale

i. The interface was designed to produce the graphs of the class data retrieved in a way that would be intuitive to its users. This data can be graphed in different ways like displaying top most As or least Ds and Fs, as specified in the SRS. It also contains the functions to update the class database given a new data .json file.

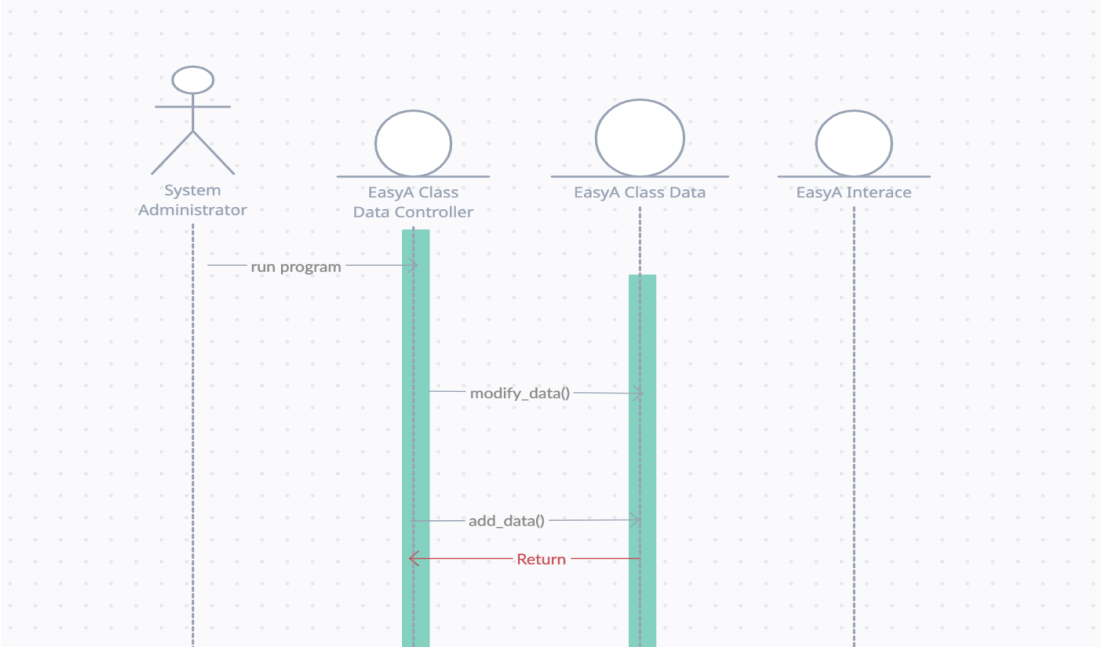
ii. Note - Change in Design Rationale for this Component: this component was originally going to only read the data in the class database and use it to create graphs, but we changed it to contain the function to update the class database given new data as well because this only took a few extra lines of code, and then we ended up creating another “create_graph.py” file whose functions we import into this model as the graph creation code was complex and easier to manage in a separate file.

5. Dynamic Models of Operational Scenarios (Use Cases)

Student Use Case:



System Administrator Use Case:



6. References

Sommerville (2015) *Software Engineering*, 10th edition, Pearson

Kieras *Basic UML Class Diagram Notation*, Handout, Anthony Hornof

7. Acknowledgements

This template builds slightly on a similar document produced by Stuart Faulk in 2017, and heavily on the publications cited within the document, such as IEEE Std 1016-2009.