|  |  |
| --- | --- |
| A picture of a winding road and trees  Special Academe Services  Project documentation | Abstract  This document shows a detailed documentation and project proposal in response to the mini-project from web technology and design  By Brian Msane  Together with: Neliswa Maziya and Thandolwethu Nhlabatsi; 202203673, 202203763, 202203673 |

ial Academe Services Documentation

# Table of Content

1. Project Overview
   1. Introduction
   2. Problem Identification or Statement
   3. Project objective
   4. Contribution (significance of proposed solution)
   5. Stakeholders
2. Requirement Gathering and Feasibility Study
   1. Requirement gathering techniques
   2. Sample interview questions
   3. Sample questionnaire
   4. Functional and non-functional requirements
   5. Feasibility Study
      1. Technical feasibility
      2. Economic feasibility
      3. Schedule feasibility and scope
3. Solution Modeling (DFD)
   1. Use case diagrams
   2. Context Diagram
   3. Level 1 data flow diagram
   4. Level 2 data flow diagram
   5. Data dictionary
4. Database and Normalization
   1. Database used
   2. Schemas
   3. Progressive normalization (0NF, 1NF, 2NF, 3NF, BCNF)
   4. Query optimization
5. Testing and Verification
   1. Unit tests and CI/CD
   2. User acceptance testing (Automation)
   3. Validation
      1. Form validation
      2. Field validation
      3. Page validation
6. Development Tools
   1. Front-end tools
   2. Backend tools
   3. SCRUM framework
   4. Application programming interfaces (APIs)
7. Wireframe
   1. Tools
   2. System wireframe
8. Conclusion

# Project Overview

* 1. Introduction

Often times when a student finishes writing the form five external examination they tend to aspire or even project themselves as university students. Unfortunately, between this fantasy and the reality lies a hurdle which is the application process. This process has so many problems which are: unawareness of application dates, misinformation about programs they want to pursue, lots of money spent, to name a few. This information gap can lead to uninformed decision, impacting their future careers and overall satisfaction during the course of their study. The need for a centralized system to assist student in making well-informed decision is apparent.

* 1. Problem Identification or Statement

High school graduates encounter several critical issues during the university application process and those includes the following.

1. Limited understanding of available programs: Students often select programs based on advice from others, media influence, or long-held aspiration without fully understanding the program content, admission requirements, or career prospects. This can result in dissatisfaction and hinders career progression. In often times, students end up dropping out because they are required to do things they are not passionate about.
2. Costly application procedures: The traditional application process requires students to physically visit multiple universities and colleges to gather information and submit applications. This methods is time-consuming, expensive, and inefficient, especially for those with limited financial resources. Also, the universities need more human labor to handle this process and that is ultimately costlier.
3. Unawareness of application timeline and costs: Many students are not informed about application opening and closing dates, acceptance periods, and costs associated with the entire process in different institutions. Many miss opportunities due to this.

These challenges not only necessitate a centralized system but also highlight a systematic problem affecting students’ successful transition into tertiary education. The lack of accessible, comprehensive information and streamlined processes necessitate a solution to support students during this critical phase of their lives.

* 1. Project objective

The proposed system aim to alleviate the mentioned issues by building a very informative, user-friendly, and integrated system. The primary objective is to provide comprehensive program information to applicants. Detailed descriptions, prerequisites, admission criteria, curriculum details and potential career paths associated with each program will be provided. Further, we aim to simplify the application process by creating centralized, one-size-fits-all online platform which allows students to apply from the comfort of their homes to reduce physical visits and travelling costs. Lastly, to mention a few of our objectives, we aim to provide the awareness of application timelines and costs associated with applying.

* 1. Contribution (significance of proposed solution)
  2. Stakeholders

Project stakeholders are individuals or organizations who are actively involved in the project, or whose interests may be positively or negatively affected as a result of project execution or successful project completion. The stakeholders involved in this project include:

* Tertiary Institutions: Universities, colleges and technical institutions who are responsible for managing admissions.
* Applicants: Students applying for admission to undergraduate programs.
* Government bodies: Agencies seeing education in Eswatini, responsible for policy formulation and data collection.
* Sponsors and funders: Private organizations or individuals looking to sponsor students based on specific criteria.
* SAS development team: The team responsible for designing, implementing and maintaining the system.

# Requirement Gathering and Feasibility Study

* 1. Requirement gathering techniques
  2. Sample interview questions
  3. Sample questionnaire
  4. Functional and non-functional requirements
  5. Feasibility Study
     1. Technical feasibility
     2. Economic feasibility
     3. Schedule feasibility and scope

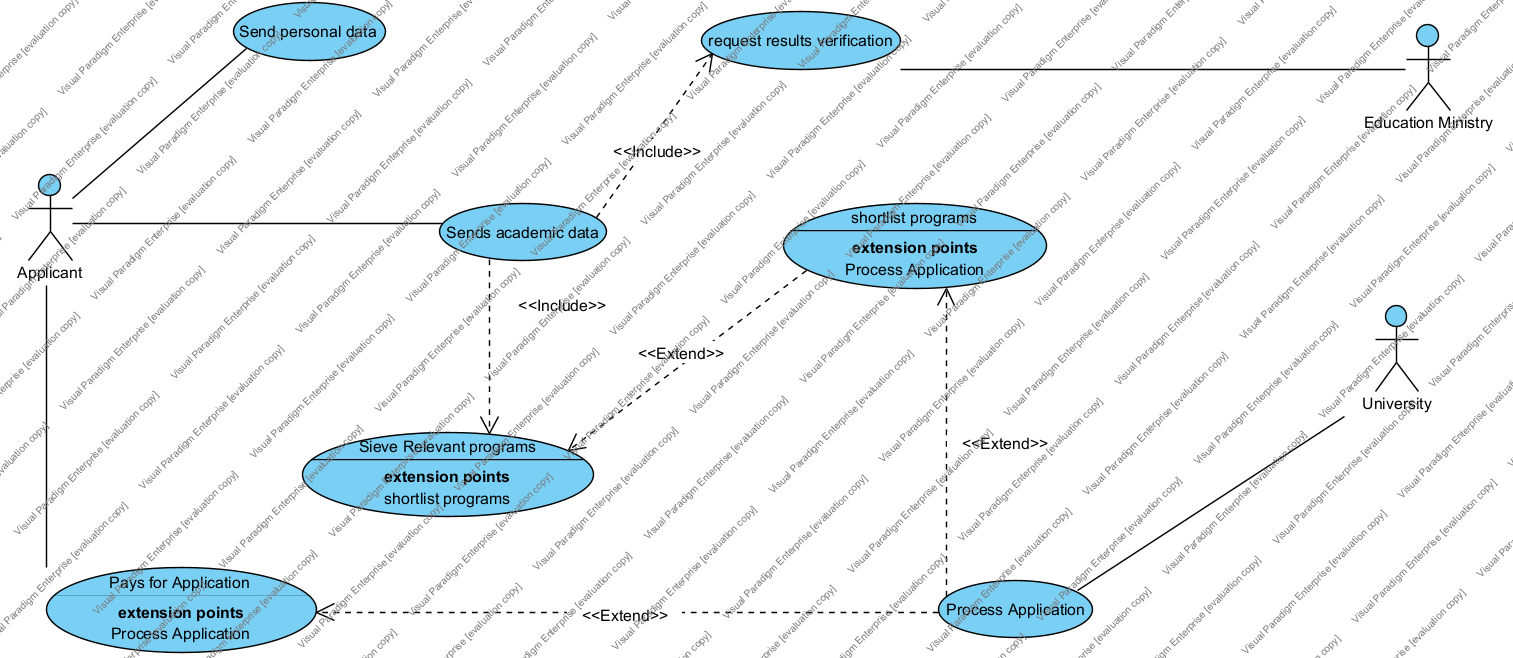
# Solution Modeling (DFD)

3.1 Use case diagrams

The use case diagram below is a high-level depiction of the entire platform. It is based on the scope defined for this project and it is independent of the implementation details. The applicant, university, and education ministry are the key actors in the platform, for now, and their interactions with the system are represented in the use cases.

The relationship between the use cases, use cases and actors were only defined using the communication/ association, includes, and extends relationships. A breakdown of key relationships is shown below.

* The *applicant* communicates with the **send personal** and **send academic data** use cases
* When the applicant sends academic data, we need to verify them in the Ministry of Education so the use case **send academic information** has an include relationship with the **request results verification** use case.
* Sending academic data also triggers the need to get the programs that the students can qualify for depending on their total points and their cut of point. Therefore, **send academic data** has an include relationship with **sieve relevant programs**
* The *applicant* communicates with **pay for application**
* We only process the application based on whether the student has paid and we have the programs that the students qualify for shortlisted to the top three. Hence, we have the following relationships:
  + An extend relationship exists between **sieve programs** and **shortlist programs**
  + An extend relationship exists between **pay for application** and **process application**
  + An extend relationship exists between **shortlist programs** and **process application**
* The *university* communicates with **process application**

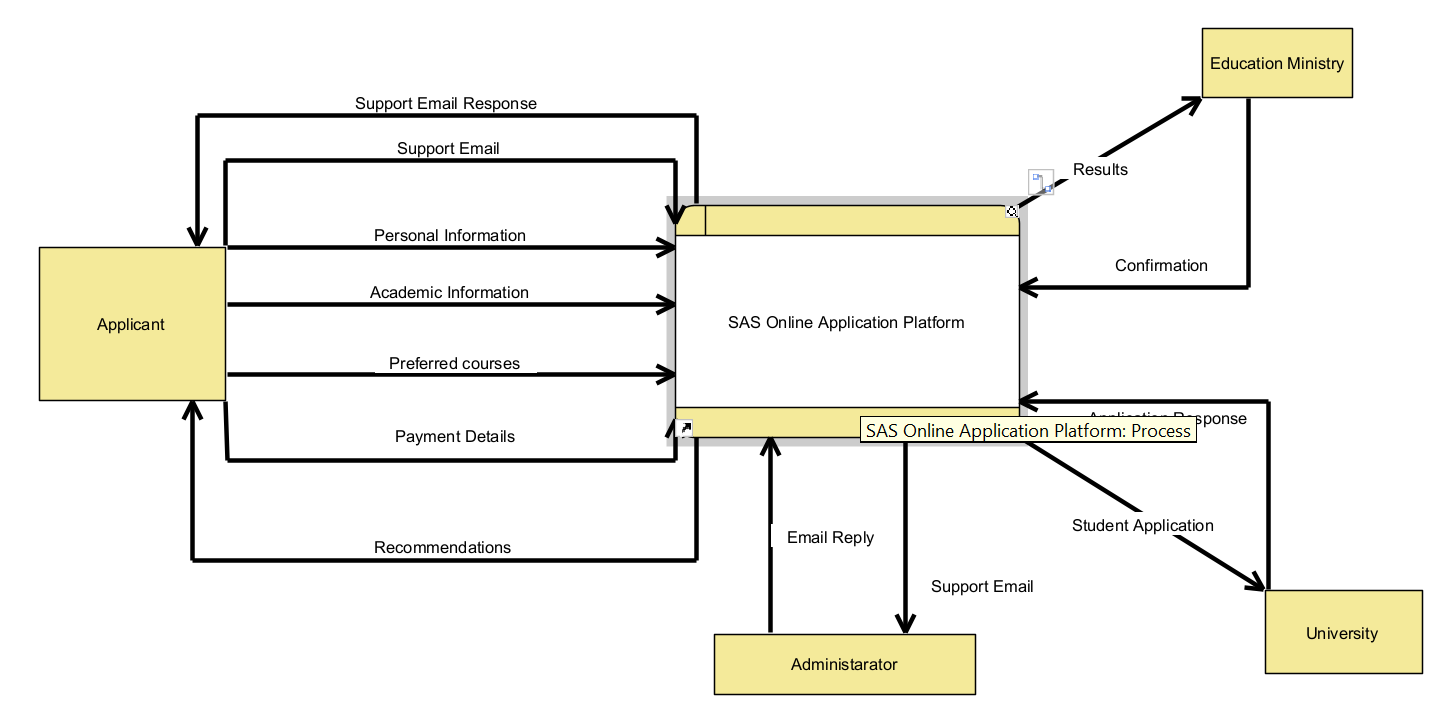


* 1. Context Diagram

The context or level 0 data flow diagram depicts the entire system as one big process. This process is given the number 0 and in other levels of the DFD, it is exploded into many processes. The major thing in the context diagram are the external entities and the major data flows. As we can see, external entities are:

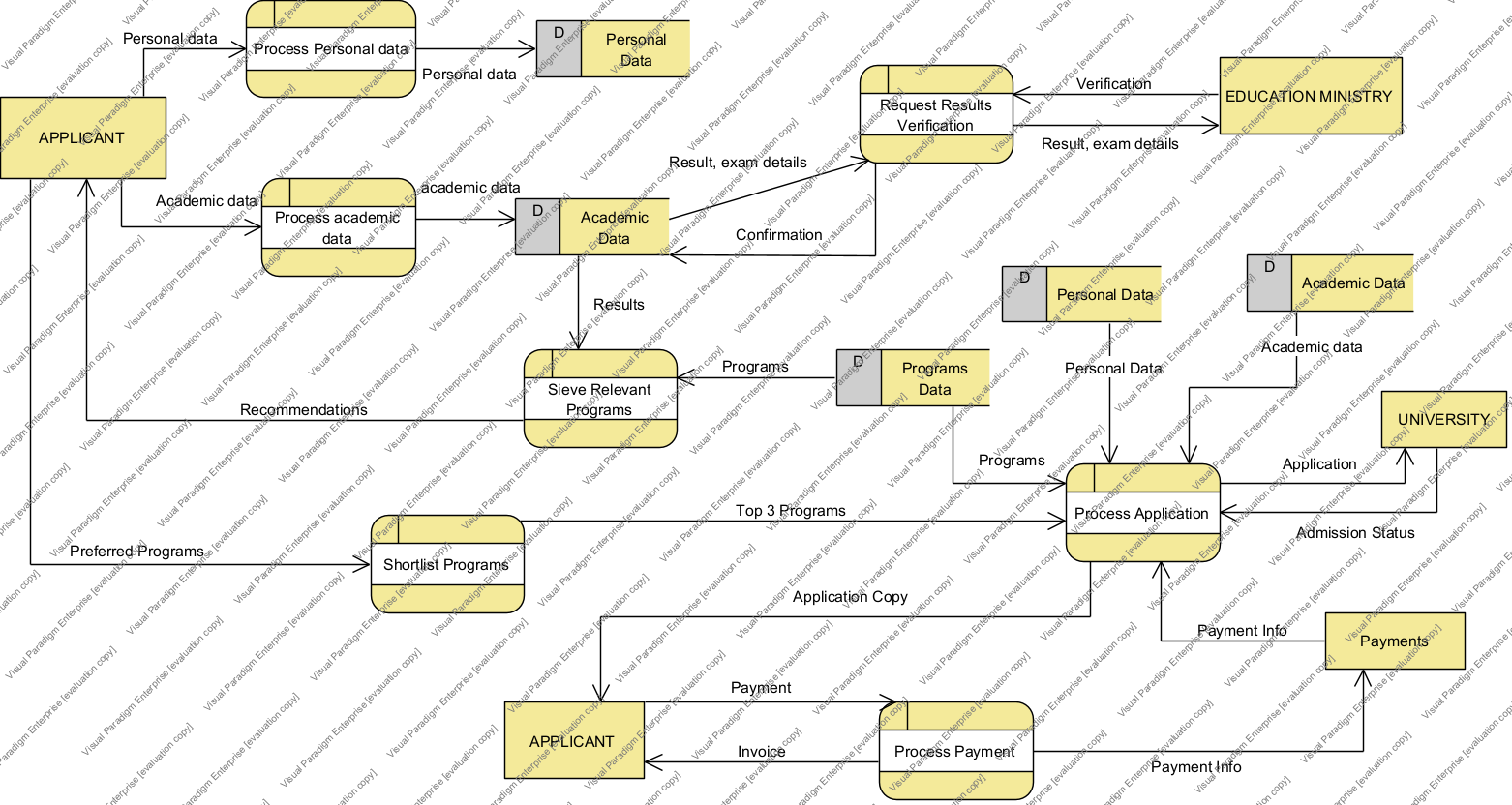
* Applicant,
* Education Ministry
* Administrator, and
* The university we’re to apply to

This list resembles the key stakeholders on the platform, in as far as primary actors are concerned.



* 1. Level 1 data flow diagram

When exploding the number 0 process in the context diagram we had a level one data flow diagram which is shown below.



* 1. Level 2 data flow diagram
  2. Data dictionary

# Database and Normalization

* 1. Databases used

For the purposes of this project we initially projected to use a not only structured query language (NOSQL) database to store the information about our users. After careful realization, we thought of including a structured query language (SQL) database management system (DBMS) like MySQL. We have a wide variety of information needed by this platform so we’re going to use the two database management systems to store our information.

The advantage with NOSQL is that in as much as it incorporates SQL, it is not only that. It further goes to implementing a flexible schema which is not strict during enforcement and when new data is added. Particularly, the choice here is the excelling Mongo DB. It allows for changes and storing non-structured textual data which is widely used in this platform.

For storing user information, we need a strict database management system and therefore, we opted for MySQL. First, we need to define the schemas for each and every relation, the relationships between relations, enforced or initiated by the primary and foreign key.

* 1. Schemas

We used the following relations to and their schemas are defined below

APPLICANT (name, surname, pin, address, postal, cell-number, next-of-kin, center-number, candidate-number, school, symbols, next-of-kin-cell-number,)

UNIVERSITY (Code, Full name, location, campuses, telephone, email, website, addresses)

PAYMENTS (Pin, date, reference, amount, commission, proof, application-amount)

PROGRAMS (Prog-id, name, description, courses, cut-of-point, year-of-study,)

Note: The underlined attribute is the primary key of the relation.

* 1. Progressive normalization (0NF, 1NF, 2NF, 3NF, BCNF)

The schemas listed above are un-normalized, or in 0 normal form so we need to apply the progressive normalization process to ensure that they are normalized. In this stage, the schemas pose a great deal of update anomalies which can be categorized as deletion, modification, and insertion anomalies. Also, we have redundancy spanning around the entire database, hence consuming a lot of disk space.

The normalization process starts with the un-normalized form, to first normal form, to second normal form, third normal form, and Boyce-Codd normal form. Even though some of the relations showed to have fully normalized even before we go for BCNF.

* 1. Query optimization

To optimize queries we used indexing

# Testing and Verification

* 1. Unit tests and CI/CD

To test if each function works, we require unit tests. The code base might be huge so we made a strategy for our testing, especially the backend. We employed continuous integration, continuous development, which is a term that is frequently used in DevOps to refer to continuous pushes into the code base and contribution to deployment into production. This means that after each contribution which could be adding functionality of a feature, the developer will push the changes and create a pull request to the deploying branches. The second part of CI/CD means the new changes will be released to deployment.

But before this can be done we need to ensure that the system will continue to function as intended. Now this is where our strategic unit tests comes into play. Here we want to ensure that certain components of the system work and those are the breakable strategic points.

When these unit tests pass, it ultimately means that the system is functioning properly and will not break in production. So since we were using GitHub as our version control, we exploited the GitHub action feature and made a workflow to test the functionality of our breakable parts. When the workflow passes, the system goes to production and when it fails, the pull request is not accepted and we’re then prompted to fix the issues until the unit tests are functioning correctly.

* 1. User acceptance testing (Automation)

To perform user acceptance testing, which is validation of system behavior against user requirements, we employed two strategies. The first strategy involves giving the software to the users who helped us refine the system requirements in the beginning of the planning phase and the second option is to write an automation bot which will have to test the system behavior automatically.

We ensured strict reliance on the requirements by making sure that we implement prototyping into our UAT testing. That is, we first give the system to users to work with, next, we collect their feedback and then we implement or change the system behavior accordingly. From there we continue this loop until we attain or achieve all the requirements. The developers, knowing the requirements themselves, were also required to test the system to ensure that it relies on what was initially planned and nothing diverged during the system development. When issues are encountered in this phase, the developer, who takes the position of a tester, needs to write a bug report and throw it back to the developers to resolve the bug and ensure system compliance.

The automation part is the fastest and probably the most reliable one as it has no assumptions and will not result in scope creep. Here we just test what happens when the user clicks some buttons, scrolls, and if we get the expected result when we make some API calls. The expected, or actual behavior, which is in accord to the requirements, is matched against the response we get from doing an action in the front-end. The user interacts with the front-end and therefore we are using this bot in our front-end.

* 1. Validation

Validation is a crucial process in web and development and it ensures that he data input by users meet the required standards before being process or stored. This drastically enhance our security, improve data quality, and provides a better experience by catching errors early. We avoid possible breaks in our backend by ensuring that the data in the front-end is going to be compatible with the backend, ensuring smooth serving. We implement validation from three perspectives which are form, field, and page validation.

* + 1. Form validation

Form validation is the practice of checking data entered into an HTML form before it is submitted to the server. This kind of validation has two options which are client-side and server-side form validation. Client-side provides immediate feedback to users before the data is sent, reducing server load and enhancing user experience. On the other hand, server-side validation ensures that the data is validated even if the user tried to bypass client-side checks. It is essential for maintaining data integrity and security.

In form validation what we focused on were the following aspects.

* Ensuring required or mandatory fields are not left empty
* Check if inputs matched the expected data formats. For instance, ensure that an ID is eleven digits
* Verify data types
* Check ranges for values. For instance, there’s no person alive who was port in the 16th century
* Cross field validation. Validating related fields. For user authentication we can validate the password and username separately and then the combination of the two to see if the username corresponds to the password give.
  + 1. Field validation

Field validation focuses on individual form fields, ensuring each one contains acceptable data before the form is submitted. Immediate feedback helps the user correct the mistake instantly. The error messages need to be clear and specific when the validation fails, for instance, when the user should enter a value between 1 and 9 we need to put the range in the error message to ensure we guide the user. We also have to make accessibility considerations to ensure that validation messages are accessible to all users, including those using assistive technologies.

* + 1. Page validation

We were not able to do page validation given this time constraint of the project but going forward we will implement it. Pardon.

# Development Tools

* 1. Front-end tools

Just like any typical website, the front-end tools we used to develop our front-end are Hyper Text Markup language, HTML, cascading style sheet, CSS, and JavaScript. Each one of these have a pivotal role in the development of and interactive, intelligent, and responsive website.

HTML was used for defining the structure and content of all our webpages. Within HTML, there are tags which have special meaning which are used to define the content from text, to images, videos, and any kind of content. Mostly, we made use of text, forms, input, and anchor tags and so on. The output at this stage is not so good looking. Mind you, this is mainly was what they interacted with during the Web 1.0 evolution stage.

To transform the basic HTML structure into a pleasing interface, we employed CSS. CSS allows us to style the HTML elements by applying properties such as colors, fonts, layouts, and spacing. We ensured that the website adapts seamlessly to different screen sizes and devices. To enhance user experience, we made use of transitions and so on.

JavaScript was used to add interactivity and dynamic behavior to the website. Ever since Web 2.0, we have been witness a great deal of dynamic pages. By manipulating the Document Object Model, DOM, we created interactive elements like dropdown menus, sliders and form validation. We also used JavaScript to fetch data asynchronously using APIs, allowing for dynamic content updates without reloading the pages.

To set up a local development server and manage front-end routing, we used Node.js along with the Express framework. Node.js is a JavaScript runtime that allows us to run JavaScript code on the server-side. Express simplifies the process of handling HTTP requests, serving static pages, and setting up middleware. This setup enabled efficient testing and development of our front-end application.

* 1. Backend tools

The backend was main written in the Python programming language. This is a fast, high-level language known for its readability and extensive libraries, which accelerates the development process and reduce complexity.

* 1. Application programming interfaces (APIs)

The APIs we made using the Fast API framework which is designed for building high-performance APIs for python. It also comes with automatic API documentation and the incredible SwaggerUI Documentation which allows you to have an interface where you can call the endpoints.

By using FastAPI, we adhere to RESTful API principles, creating stateless endpoints. Our APIs follows the REST architecture which uses HTTP methods like GET, POST, PUT, PATCH, DELETE, and so on. Each endpoint corresponds to a specific resources. The input data, from the request body, is checked against a request schema which uses type hints to enforce some validation. This is another layer of security built to complement the form and field validation done on the client-side. Also, the response is also checked against a response schema to ensure that the output of the endpoint complies which the expected data formats making it compatible which other parts of the software.

Alternatively, we explored using GraphQL as it allows clients to request exactly the data they need, nothing more, nothing less. GraphQL serves as an efficient alternative to RESTful APIs, especially in complex systems with interconnected resources like ours. It has a number of advantages such as: fetch specific data, single endpoint which servers everything, strong typing through the schema, and real-time data supported by subscription, and efficient network usage by minimizing requests and data transferred.

Despite these incredible advantages of GraphQL, we chose to stick with FastAPI for now since it is a framework which we were already comfortable working with but in the near future it will be best to stick to it. As we also make use of a chat system, we’d also consider adding Web Sockets for real-time communication.

* 1. SCRUM framework

To manage our project efficiently, we adopted the SCRUM framework which is an agile methodology that emphasizes iterative development, collaboration, and adaptability. We had a number of benefits adopting this framework which are: enhanced collaboration, increased transparency, improved adaptability, and continuous improvement.

We had the development team, the project manager, the SCRUM master, and the product owner. We did not have enough personnel so we had to split these roles among ourselves and the responsibilities were assumed by all of us.

From the SCRUM principles we were able to proactively engage in these.

* Product Backlog. A list of prioritized features, enhancements, and bug fixes required to be done by the development team based on stakeholder engagement and requirements.
* Sprints. We used iterations which we a week long, during which specific backlog items were developed and delivered.
* Sprint Planning. Before delving into a sprint, we had to plan, set a goal, select the items to work on and decide on how to evaluate the development process and we did all of that in the sprint planning meeting.
* Sprint Reviews. At the end of each sprint, we evaluated if we met our goals and review how the sprint went.
* Sprint Retrospective. Infrequently, we meet to discuss what went well, want did not go well and decide on how to improve.
* Daily stand-ups. On a daily basis, we meet on campus to see what we did the previous day, what we’re to do on the current day and help resolve issues that any of us might be encountering.

# Wireframe

* 1. Tools
  2. System wireframe

# Conclusion

As we conclude, we’d like to first acknowledge the amount of change to be brought by the proposed solution, its contribution to the society and the entire education spectrum.