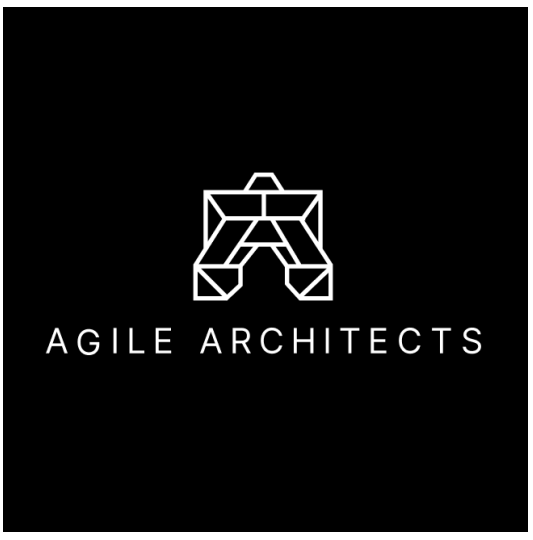
PRONTO

SRS Document- Demo 4

Name	Student Number
Shashin Gounden	u21458686
Tyrone Sutherland-MacLeod	u21578878
Bandisa Masilela	u19028182
Andile Ngwenya	u20612894
Jonel Albuquerque	u21598267



Last Edit: 2023/09/25

Table of Contents

Contents

Github Repo: https://github.com/COS301-SE-2023/Pronto

1.1 Introduction

A. Vision and Objectives

Pronto is a multi-purpose application that will be used by universities, lecturers, and students. Students will be able to create, edit and view custom timetables. Lecturers will be able to schedule reminders for important events like exams and project due dates, as well as notify them of assignment extensions. Institutions will be able to upload their university lecture schedules and campus maps. This application aims to help students organise their schedules and also bring together the features of reminders, navigation, and lecture timetables into a single application.

B. Business Need

Currently, there is no single consolidated application that allows students from any higher education institution to create a timetable based on the modules they are taking. Features for different parts of the application exist in isolation, and we intend to design a system that brings these features into a single place for any student of any university.

C. Project Scope

Defining deliverables

The scope of the project involves developing a proof of concept web/mobile application using Amazon Web Services (AWS) as the underlying infrastructure. The core functionalities and features that we need to develop include:

I. User registration and authentication
- Users, including students, institutions, and lecturers, will be able to create accounts and authenticate themselves to access the application.
- Users will be assigned different roles that will enable them to use the system differently based on their user group.

II. Timetable generation:
- Students will be able to create schedules based on their module choices, and also view assignment deadlines, upcoming tests and examination schedules, and other important events.
- They can also set reminders and define how they would like to receive notifications for upcoming tasks. (E-mail, SMS, or push notifications)

III. Institution pages:
- Institutions will have the ability to create and maintain their own pages within the application.
- They can load and update student schedules, campus maps, assignment deadlines, test and examination schedules, and other relevant information. Students will be able to access these documents and relevant information by visiting their page.

IV. Processing lecture schedules:

The application will allow institutions to upload their lecture schedules in the form of a .csv or .xls file, which will then be processed and stored so that a student can search for a module and have its information generated for their timetable.

V. Notifications:

The application will allow for three notification types that the user can select :

- Push
- Sms
- Email

Students can set which type of notification they would like to receive from these options.

VI. Mobile support

The application should be accessible and responsive on mobile devices, allowing students to conveniently access their tasks and receive notifications on the go.

VII. Web support

Users that are universities or lecturers will use an application with a web view to create their accounts, manage their pages, and set reminders respectively.

VIII. Reminders

The application will provide reminders to students for upcoming deadlines, events, or any updates from their institutions. These will be presented to the user via the notifications subsystem

Additionally, the optional functionalities and features that we would like to develop include:

IX. Geo-location services

Students should be able to use a map service within the application that will give them directions to a lecture hall from their current location. This service will use a walking distance calculator to work out the time required to navigate between the lecture venues.

X. Extracting and transforming the lecture schedule

When an institution uploads its lecture schedule, we would like to take the provided format and transform it into a standard format for storage and consistency purposes. With this, inconsistencies with naming formats can be addressed.

As it stands, we plan to be finished with the core deliverables by the 24th of July 2023. Thereafter, development on the optional deliverables will begin and finish by the 20th of September 2023.

1.2 User Stories/Characteristics

A. Students

1. As a student, I want to be able to create a custom timetable based on the lectures I have, so that I can stay organised and have an easy way to make a timetable.
2. As a student, I want to be able to view assignment deadlines, upcoming tests and examination schedules, and other important events, so that I can plan my schedule accordingly.
3. As a student, I would like to define how I would like to receive reminders for my upcoming tasks so that I can receive notifications in a way that works best for me.
4. As a student, I want to be able to receive notifications about upcoming deadlines or events, so that I never miss important information.
5. As a student, I want to be able to access my institution's page, so that I can easily find relevant information and documents.
6. As a student, I want to be able to use a map service within the application that will give me directions to a lecture hall from my current location, so that I can easily navigate around campus.

B. Lecturers

1. As a lecturer, I want to be able to schedule reminders for important events like exams and assignment due dates so that my students are aware of upcoming deadlines.

2. As a lecturer, I want to be able to change assignment deadlines or test dates as the need arises, and for my students to be notified of these changes.

3. As a lecturer, I want to be able to send out important announcements regarding tests or assignments to my students so that they can be notified immediately of important changes.

C. Institution

1. As an institution, I want to be able to upload files like our university lecture schedules, campus maps, and other relevant information so that students can easily access it.

2. As an institution, I want to be able to update lecture schedules, and other relevant information about the institution, so that students always have access to the most up-to-date information.

D. Any user

1. As a user, I want all of my data stored safely and for the privacy of my data to be taken into account so that my personal information is protected

1.3 Functional Requirements

Requirement 1: User Registration and Authentication

Sub-Requirement 1.1

Pronto will allow users to create a new account by providing their name, email, and password. Accounts will be divided into student accounts and lecturer accounts.

Sub-Requirement 1.2

Pronto shall provide the capability for users to log in to the application using their email and password. Users can log in as either students or lecturers with different privileges.

Sub-Requirement 1.3

Pronto will allow students to pick which university they attend after signing up to get content related to their university.

Sub-Requirement 1.4

Pronto will allow all users to edit their account details, like their name, phone number, and password.

Requirement 2: Timetable Generation

Sub-Requirement 2.1

Pronto shall provide a search capability for modules to students.

Sub-Requirement 2.2

Pronto will allow students to delete modules from their subject list.

Sub-Requirement 2.3

Pronto will allow students to add modules to their subject list.

Sub-Requirement 2.4

Pronto will allow students to edit their timetable by selecting activities for each module.

Requirement 3: File Upload/Retrieval

Sub-Requirement 3.1

Institutions can update student schedules, campus maps, assignment deadlines, test and examination schedules, and other relevant information.

Sub-Requirement 3.2

Students can access these documents and relevant information by visiting their institution's page.

Sub-Requirement 3.3

Institutions can upload their lecture schedules in the form of a .csv or .xls file.

Requirement 4: Notifications

Sub-Requirement 4.1

The application will allow students to select from three notification types (push, SMS, email).

Sub-Requirement 4.2

The students will be able to receive reminders for upcoming deadlines, events, or any updates from their lecturers.

Requirement 5: Reminders

Sub-Requirement 5.1

Lecturers can set reminders for due dates, test dates and any important events that will be sent to students.

Requirement 6: Geo-Location Services

Sub-Requirement 6.1

Students can use a map service within the application that will give them directions to a lecture hall from their current location.

Sub-Requirement 6.2

Students should be able to see the time required to navigate between the lecture venues.

Sub-Requirement 6.3

Pronto will allow lecturers to interactively pin locations of lecture venues.

1.4 Design patterns

Service	Provider	Role
Amazon Amplify	Amazon Web Services	It will be used to build, deploy and host both the web and mobile application. It will also be used to instantiate and deploy new resources as deemed fit.
Amazon Cognito	Amazon Web Services	It will be used for sign-up,authorization, authentication and user grouping.
Amazon Simple Storage Service	Amazon Web Services	It will be the blob storage database that will store any files uploaded by the institution admin such as campus maps and academic year schedules.
Amazon DynamoDB	Amazon Web Services	A noSQL database that will be used to store courses, activities and any future scheduled activities and/or reminders.
Amazon AppSync	Amazon Web Services	Used to build a graphQL API, parse and resolve all graphQL queries/Subscriptions and Mutations. Connect users to other deployed services such as enabling admins to upload files directly to S3.
Amazon Simple Notification Service	Amazon Web Services	Used to send push notifications to the application and other platforms such as emails, according to the user's preferences with push notifications being the default.
Amazon Event Bridge	Amazon Web Services	It will be used to schedule and trigger notifications for new activities created by lectures. The notifications will be sent using Amazon SNS,
Amazon Glue	Amazon Web Services	Used to extract the needed data from the academic schedule uploaded by the institution. It will transform the data to the expected standardised schema and load it onto the course information on our Course Info database (dynamoDB).
Amazon Lambda	Amazon Web Services	It will be used as an API Handler for our graphQL API and datastore resources. Multiple Cloud functions for executing our requirements will be built and hosted on AWS Lambda.

1.2. Service Contracts

Pronto makes use of GraphQL.Thus the front end uses queries and mutations to communicate with the backend. The format of these
is described below

```
const resp =await API.graphql( { queries: myQuery,
                                 variables: { input : { id :”XXX”,name:”John”}},
                                 authMode:"AMAZON_COGNITO_USER_POOLS" })
```

Below are the different queries/mutations ,their inputs and the structure of the JSON objects they return

Queries

```
query getLecturer{
        getLecturer(id:String){
          id
          name
          firstname
          email
    }

{
      "data" : {
            "getLecturer" : {
                  "id":String,
                  "name":String,
                  "firstname":String,
                  "email":String
                  }
            }
}

query getStudent{
        getStudent(id:String){
          id
          name
          firstname
          email
}

{
      "data" : {
            "getStudent" : {
                  "id":String,
```

```
                        "name":String,
                        "firstname":String,
                        "email":String
                        }
                }
}

query getAdmin{
        getAdmin(id:String){
          id
          name
          firstname
          email
      }


{
        "data" : {
              "getAdmin" : {
                        "id":String,
                        "name":String,
                        "firstname":String,
                        "email":String
                        }
                }
}


query getInstitution{
                getInstitution(id :String){
                id
                adminId
                name
                campusMapUrl
                openingTime
                closingTime
                minimumDuration
              }
}

{
        "data" : {
                  "getInstitution" : {
                                "id" :String,
                                "adminId" :String,
                                "name" :String,
                                "campusMapUrl" :String,
                                "openingTime" :String,
```

```
                                    "closingTime" :String,
                                    "minimumDuration" :number
                                  }
                }
        }


query getCourse{
                getCourse(id :String){
                  id
                  name
                  code
                }
        }

{
          "data" : {
                  "getCourse" : {
                                  "id" : String,
                                  "name" : String,
                                  "code":String
                                }
                }
        }



query getActivity{
              getActivity(id :String ) {
                  id
                  name
                  group
                  day
                  start
                  end
                  venue
                  frequency
              }
        }

{

              "data" : {
                      "getActivity" : {
                                      "id" : String,
                                      "name" :String,
                                      "group" :String,
                                      "day" :String,
```

```
                    "start": String,
                "end" :String,
                "venue" :String,
                "frequency" :number
            }
        }
    }
```

Mutations

```
mutation createAdmin{
    createAdmin( input : {
                    firstname:String,
                    lastname:String,
                    email:String,
                    userRole:String
                }
            )
     {
      id
      }

}

{
    "data" : {
        "createAdmin": {
                "id":String
            }
        }
    }


mutation createStudent{
     createStudent( input : {
                    firstname:String,
                    lastname:String,
                    email:String,
                    userRole:String
                }
            )
     {
      id
      }

}
```

```
{
        "data" : {
                "createStudent": {
                        "id":String
                    }
            }
}


mutation createLecturer{
         createLecturer( input : {
                        firstname:String,
                        lastname:String,
                        email:String,
                        userRole:String
                    }
              )
          {
           id
           }

}


{
        "data" : {
                "createLecturer": {
                        "id":String
                    }
            }
      }


mutation createInstitution{
        createInstitution( input : {
                        name: String,
                        adminId:String,
                        campusMap:String,
                        openingTime:String,
                        closingTime:String,
                        minimumDuration:String,
                        domains:[String]
                       }
                 )
          {
            id
            adminId
```

```
            }
}

{
            "data" : {
                    "createInstitution ": {
                                        "id" :String,
                                        "adminId" : String,
                                       }
                    }
}



mutation createCourse{
            createCourse( input : {
                                name : String,
                                institutionId: String,
                                code:String
                                   }
                        )
            {
              id
              institutionId
            }
}

{
            "data": {
                    "createCourse" : {
                                        "id" : String,
                                        "institutionId" : String
                                      }
                   }
}

mutation createAnnouncement{
             createAnnouncemnent( input : {
                                        courseId: String,
                                        description: String,
                                        date: String
                                      }
                                )
            {
              id
              courseId
              createdAt
            }
```

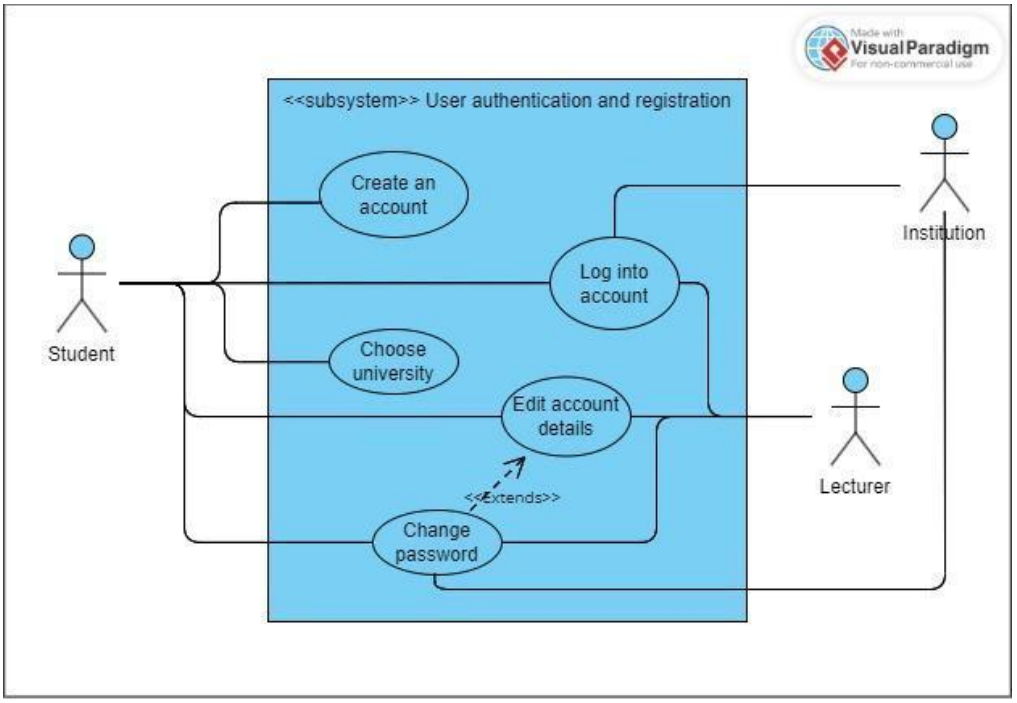
```
}


{
            "data": {
                    "createCourse" : {
                                    "id" : String,
                                    "institutionId" : String,
                                    "createdAt": String
                                   }
                   }
          }

mutation createActivity{
                createActivity(  input: {
                                    name:String,
                                    courseId:String,
                                    day:String,
                                    start:String,
                                    end:String,
                                    group:String,
                                    frequency:number
                                   }
                            )
                 {
                   id
                   courseId
                 }
}

{
            "data" : {
                     "id":String,
                      "courseId" :String
                    }
}


mutation updateAdmin{
            updateAdmin( input : {
                                id:String,
                                institutionId,
                               }
                       )
             {
               institution {
                    adminId
                    closingTime
```

```
                name
                campusMapUrl
                createdAt
                minimumDuration
                location
              }
        }
}

{

        "data" : {
                "updateAdmin": {
                        "institution":{
                                        "adminId":String,
                                        "closingTime" :String,
                                        "name" : String,
                                        "campusMapUrl":String,
                                        "createdAt" :String,
                                        "minimumDuration" : String,
                                        "location" :String
                                                }
                        }
                }
}
```

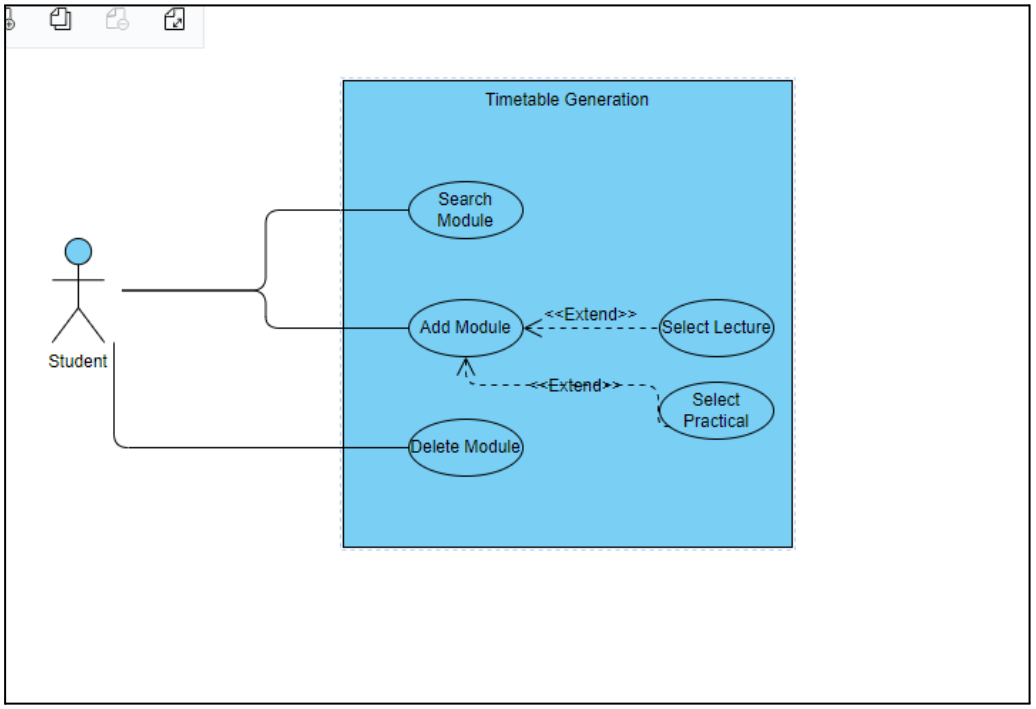
2. Diagrams

A. Use case Diagrams

1. User Registration and Authentication

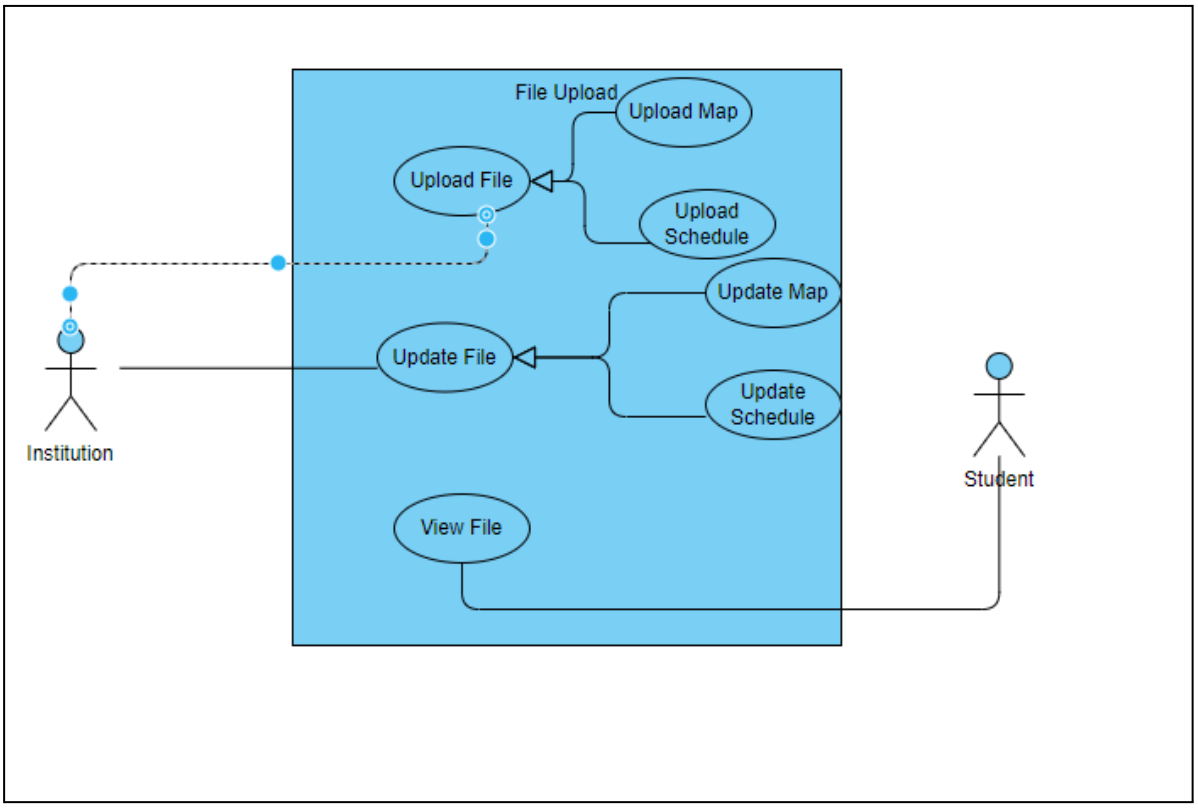


2.Timetable Generation

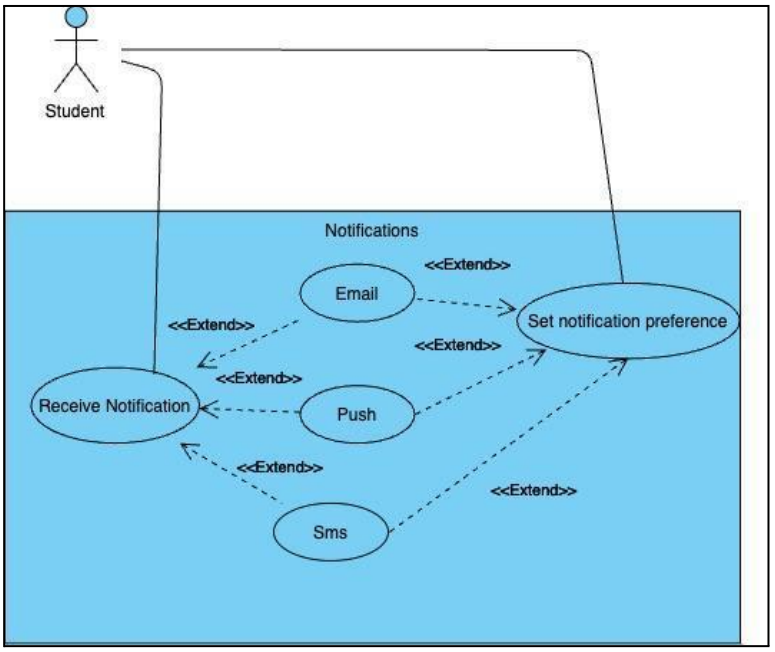


Last Edit: 2023/09/25

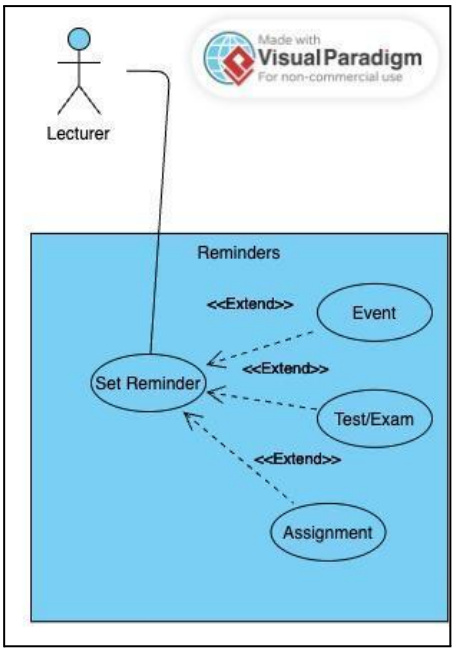
3. File Upload/Retrieval



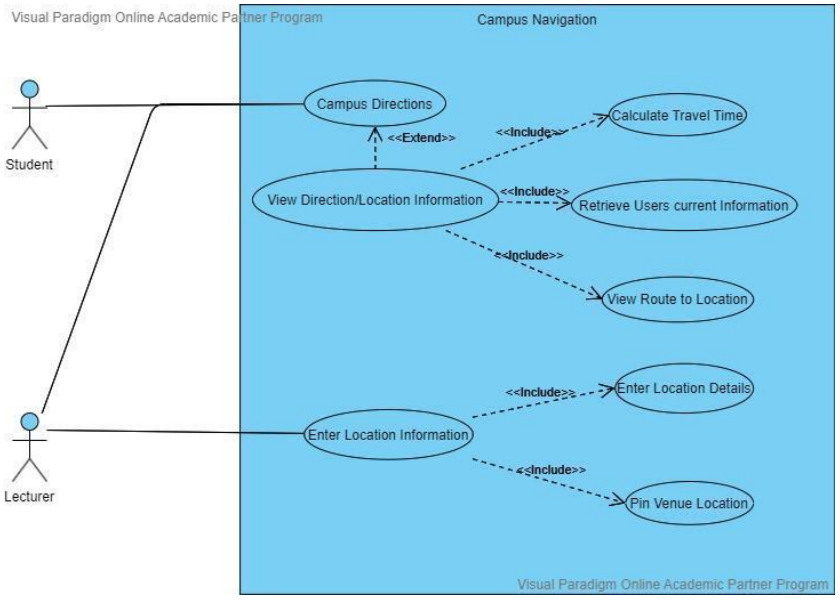
4. Notifications

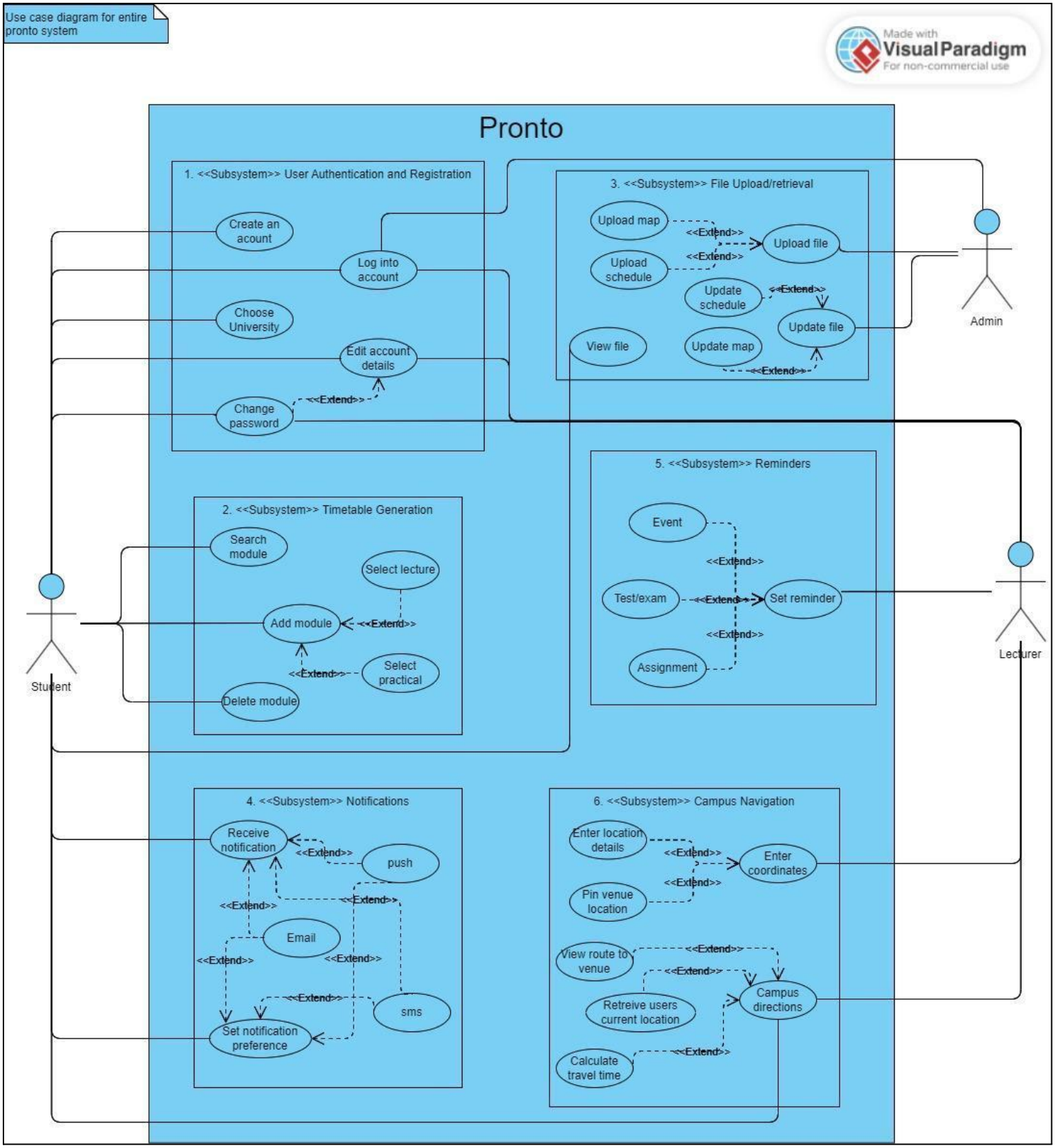


5. Reminders

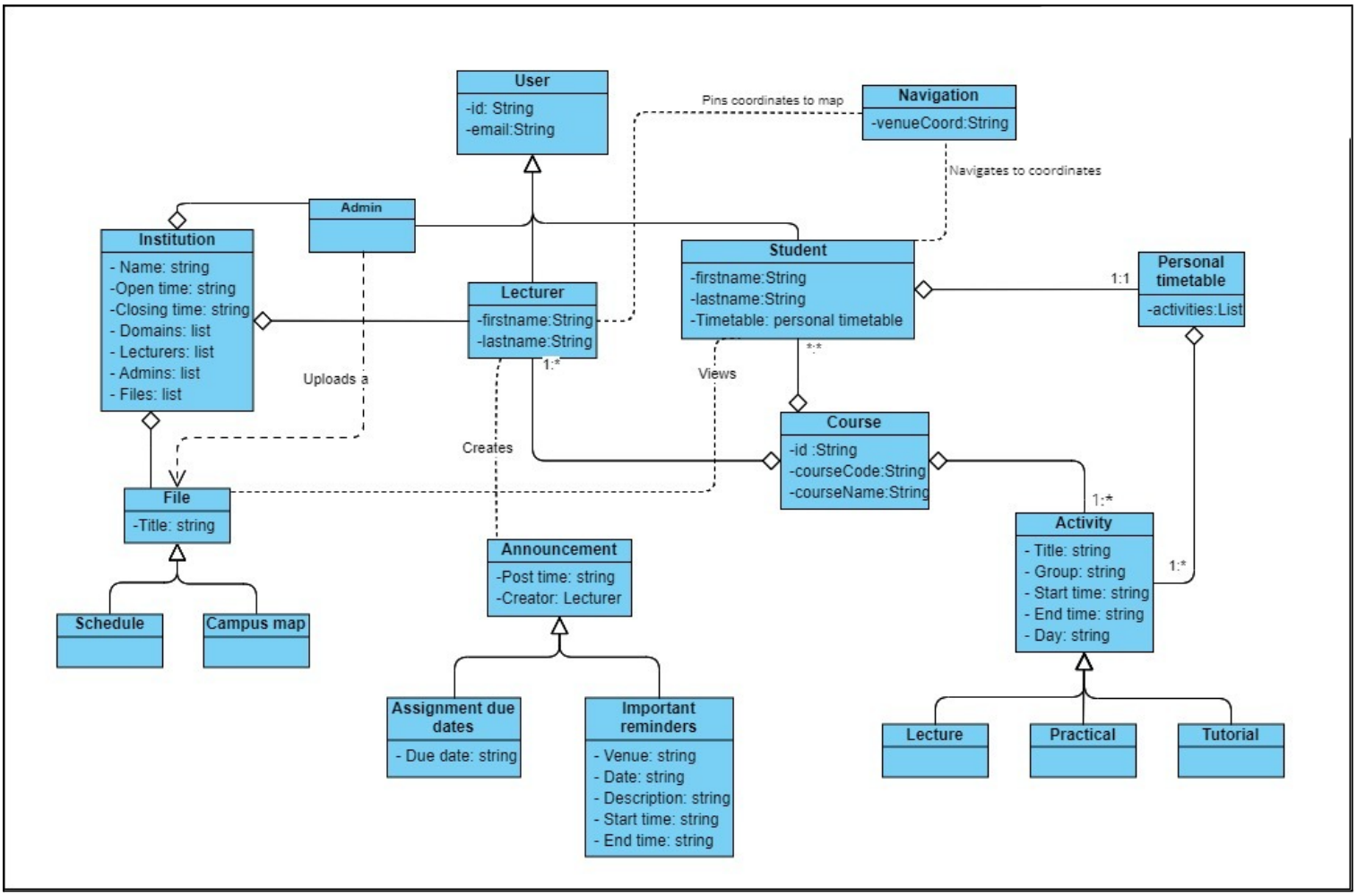


6. Campus Navigation




Use case diagram for entire pronto system
Made with Visual Paradigm For non-commercial use
Pronto
1. <<Subsystem>> User Authentication and Registration
Create an acount
Log into account
Choose University
Edit account details
Change password
<<Extend>>
3. <<Subsystem>> File Upload/retrieval
Upload map
Upload file
Upload schedule
Update schedule
Update file
View file
Update map
<<Extend>>
Admin
2. <<Subsystem>> Timetable Generation
Search module
Select lecture
Add module
Select practical
Delete module
Student
5. <<Subsystem>> Reminders
Event
Test/exam
Set reminder
Assignment
Lecturer
4. <<Subsystem>> Notifications
Receive notification
push
Email
sms
Set notification preference
6. <<Subsystem>> Campus Navigation
Enter location details
Enter coordinates
Pin venue location
View route to venue
Retreive users current location
Campus directions
Calculate travel time

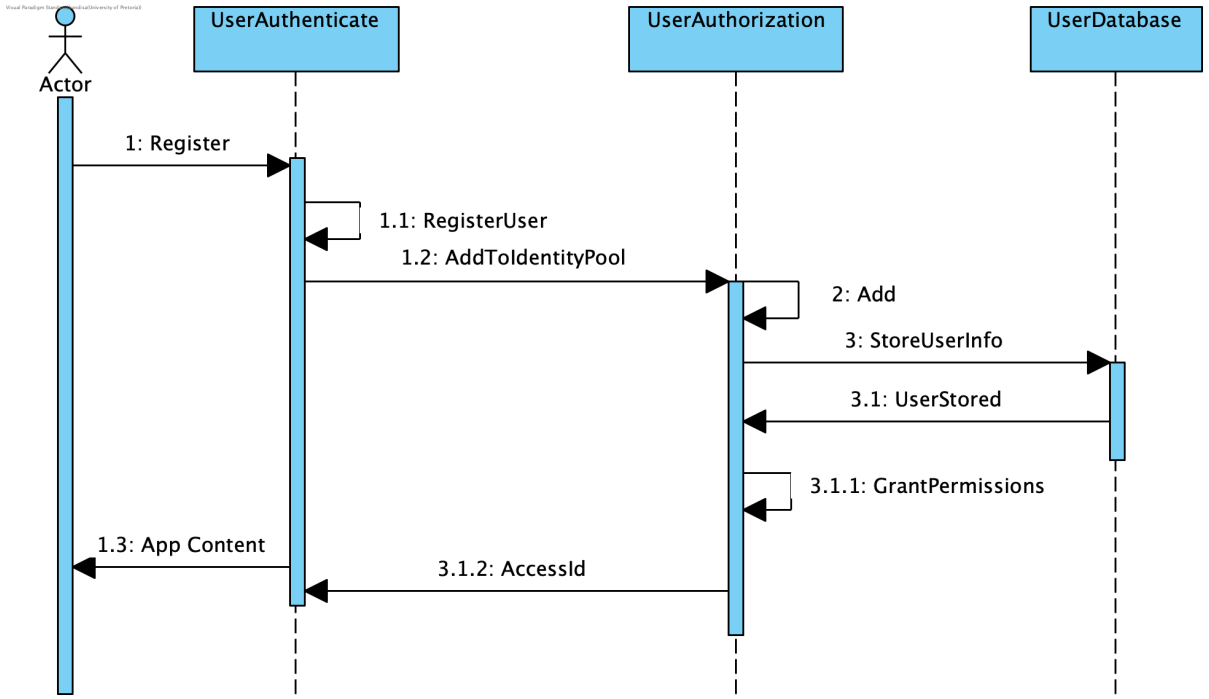
B. Class Diagram



C. Sequence Diagrams

1. Authentication and Authorization

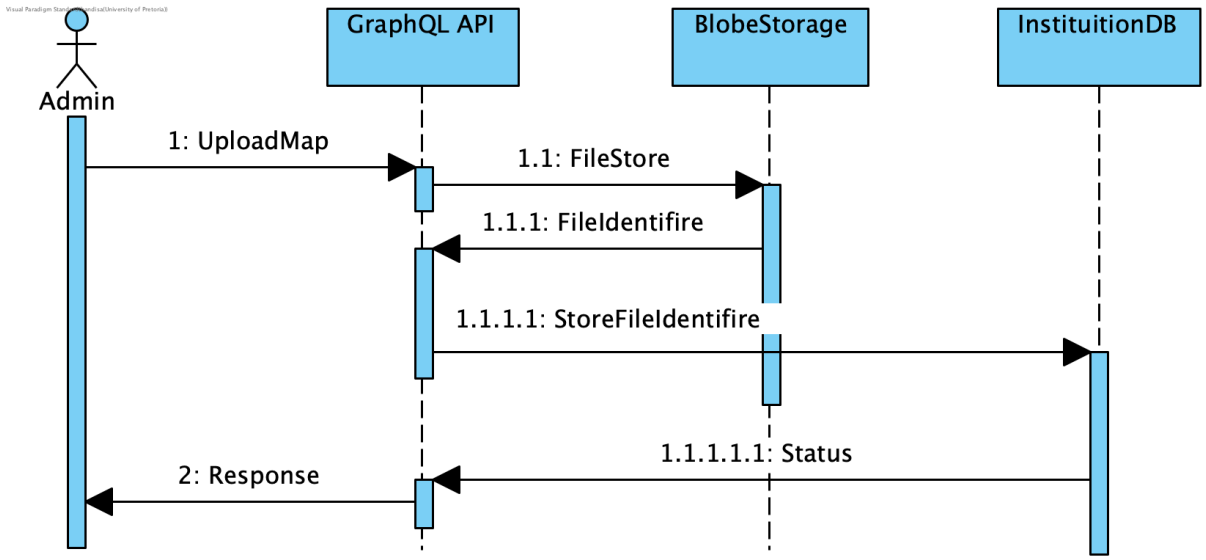
Sequence Diagram



2. File Upload

A. Campus Map

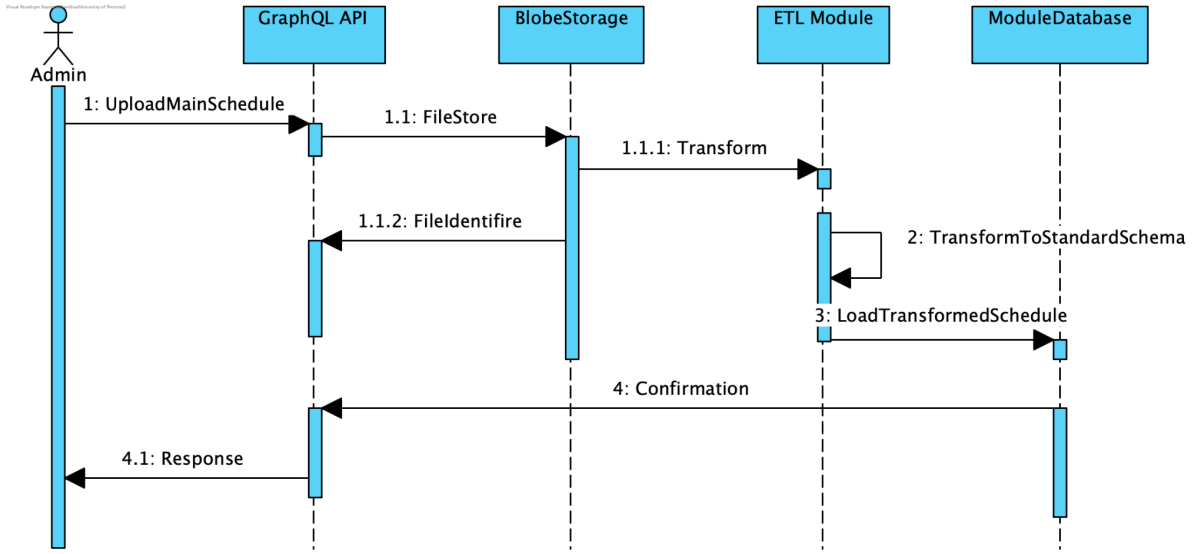
Sequence Diagram



Last Edit: 2023/09/25

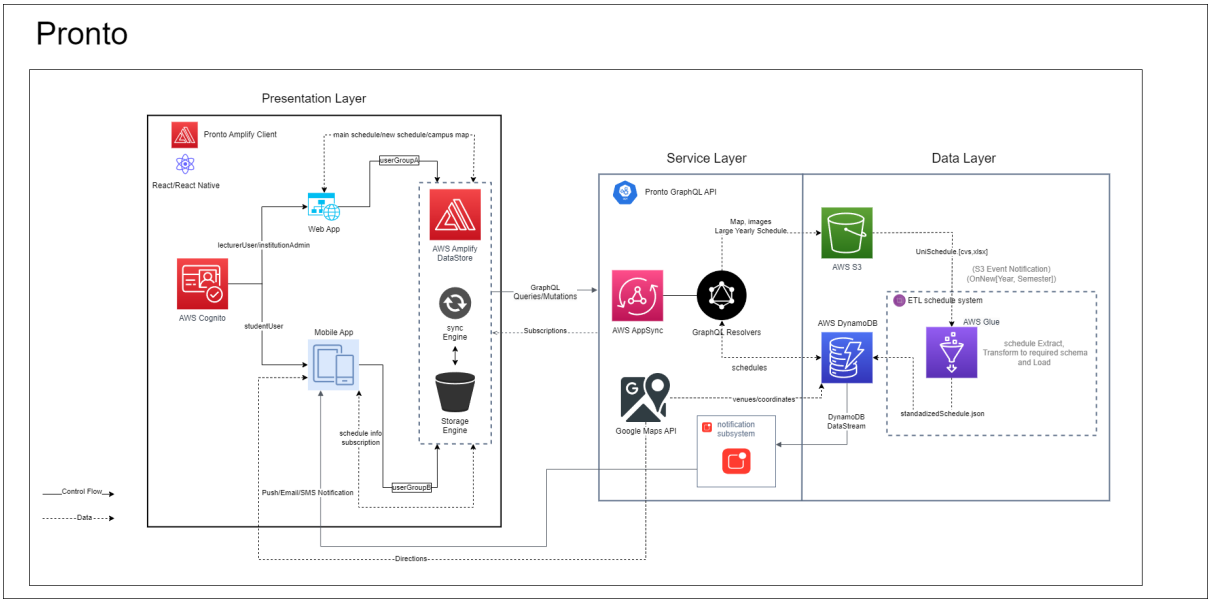
B. Transform Schedule to Standard Schema

Sequence Diagram



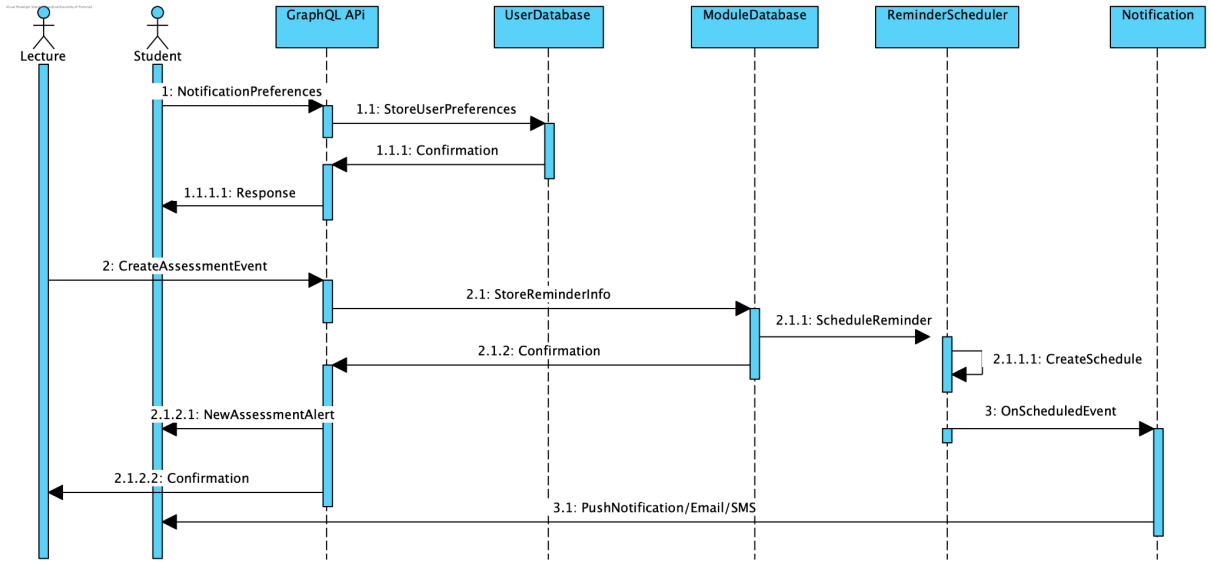
D. High-Level Architecture

High-level architecture of the entire system with the microservices used



3. Notifications and Reminders

Sequence Diagram



3. Architectural Requirements

3.1 Architectural Design Strategy

Strategy: Design based on quality requirements

Our strategy emphasises the importance of the system's quality requirement. This strategy ensures that our design decisions will lead us to deliver a system that upholds our quality requirements and guarantees that the system operates as expected.

3.2 Architectural Styles

1. Three Tier Architecture:

- Security
- Simplifies design
- Maintainability and portability

This architecture partitions our system into three different layers, namely the presentation layer, service layer, and database layer. By separating concerns of the user interface from our backend services and database, we can provide the users with a nice, easy-to-use interface that will also provide added security since the users cannot access the database directly. This separation of layers also makes the application more maintainable and scalable since changes to one layer should not affect another.

2. Component-Based Architecture:

- Robustness
- Reliability (via thorough testing)
- Reusability

By composing our UI of several components, we can encapsulate related data and functions in each component, which will aid in testing as well as organise backend functionality to improve the overall quality of the app. This architecture also allows us to reuse certain useful components in multiple instances, reducing our production time and increasing overall cost/time efficiency.

3. Serverless Computing Architecture:

- Reliability
- Scalability
- No server management

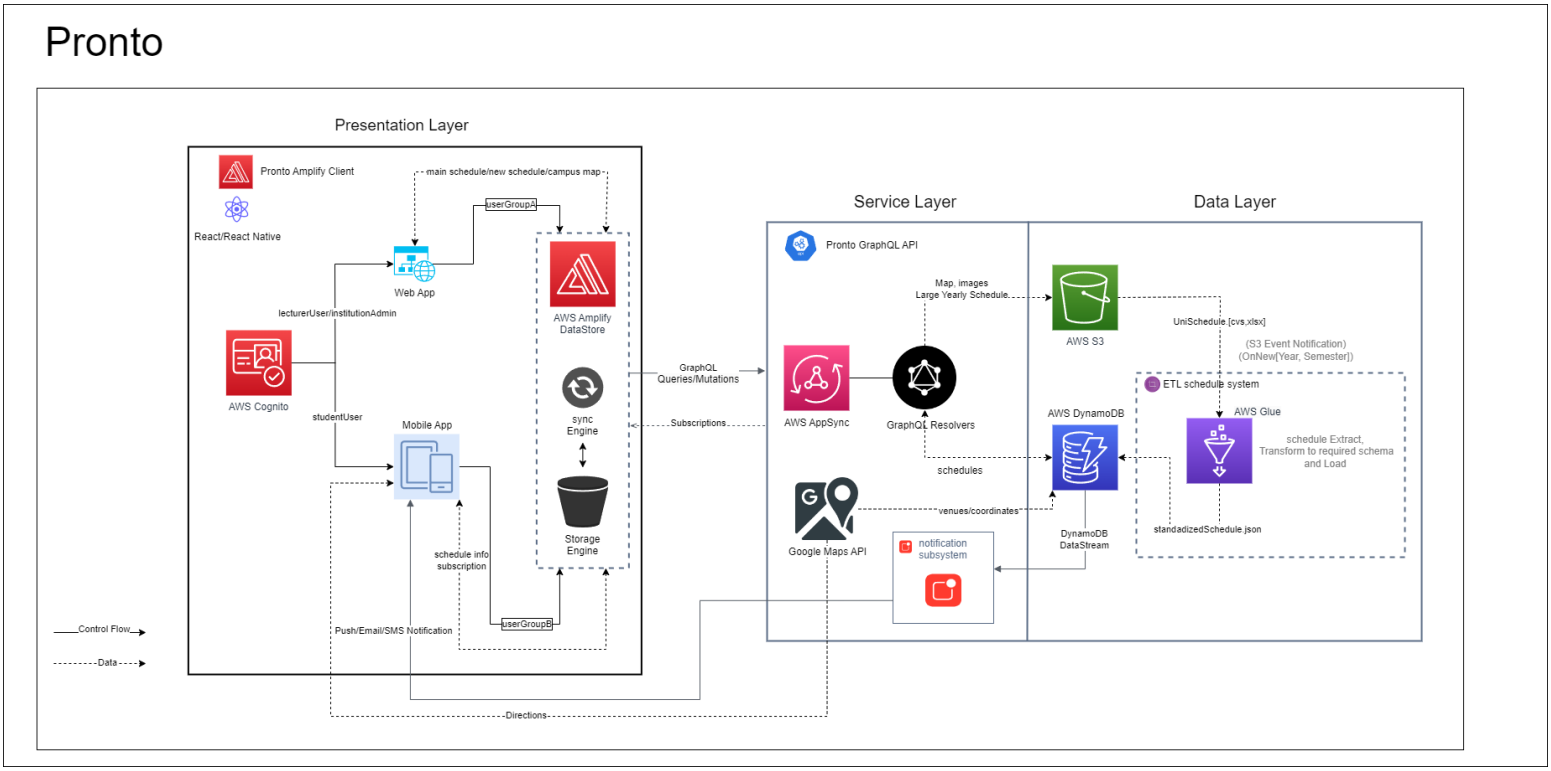
This architecture will allow us to achieve our required uptime/reliability guarantee without having to worry about infrastructure maintenance or scaling. The AWS platforms we are using will allow us to automatically scale our app depending on the incoming workload, and make integration between several components and services simpler, improving overall app quality, maintainability and production time.

4. Service Oriented Architecture:

- Flexibility
- Enhanced security
- Scalability

This Architectural style mainly pertains to the service layer. It consists of AWS services such as App sync, SNS, Event bridge and GraphQL resolvers to allow us to query the data layer and respond to user's requests whilst adhering to our scalability and security requirements (due to the nature of the AWS services being used). This loosely coupled, modular architecture makes our code secure, easily maintainable and also reusable, greatly aiding us in the development, upgrade and testing of the system.

3.3 Architectural Design: Diagram



3.4 Architectural constraints

- **Scalability:** The system needs to be capable of seamlessly scaling to accommodate a large number of students and lecturers accessing our resources automatically.

- **Security:** The system must ensure the security of user credentials, such as passwords used for creating and signing into accounts, by implementing robust protective measures.
- **Maintainability:** The system should be well-documented to facilitate easy maintenance, and its design should follow modular principles, allowing for efficient updates and modifications.
- **Portability/Tech stack:** Since the project requires both web and mobile applications, it is important to select the appropriate frameworks that meet the project requirements. Additionally, the system should have limited reliance on libraries and AWS services to ensure flexibility and reduce dependencies.
- **Testability:** The architecture should support automated unit testing and integration testing, enabling comprehensive and efficient testing of system components.

4. Quality Requirements

4.1. Performance Requirements

A. Scalability

Definition

The product should be at first able to scale horizontally to accommodate for typical traffic that would be expected by an institution during peak times, then this should extend out to multiple institutions per region.

Acceptance Criteria

- Autoscaling of the services used will be able to accommodate changing traffic on a regular basis.

B. Performance

Definition

The different clients of the product should be fast and highly responsive.

Acceptance Criteria

- End-to-End testing will be conducted regularly to test server response times and ensure that a user can quickly perform each use case or multiple use cases in an acceptable time frame.

- Different instances of the back-end will be deployed per major region to decouple scaling, allowing for different regions and resources to scale up accordingly.

4.2. Availability

Definition

The tool should always be available for all users and any unavailability communicated.

Acceptance Criteria

- The website and application should have an uptime of at least 99% as it is critical for a user to be able to access and use their timetables at any given time.
- Any planned maintenance will occur outside of normal university hours or schedules
- Planned downtime will be indicated or communicated with clear UI changes.

4.3. Security

Definition

User data should be secured at rest and in transition in order to prevent unauthorised access and comply with local laws and regulations (POPI).

Acceptance Criteria

- User data will be Encrypted in transition and Encrypted at rest. (AES 256 is used by Cognito, with salt added)
- Users will be grouped into user pools during sign up
- This will provide limited access to data modification and access as per use case.
- System must be tested and compared against OWASP list of vulnerabilities.

4.4. Maintainability

Definition

The software should be easily maintainable by both internal, or new external party(s), with clear concise documentation and code.

Acceptance Criteria

- Code linters and standard linting rules will be applied to ensure code uniformity.

- A 3-person review procedure will be used to ensure that all solutions provided are efficient and employ the best procedures.
- New features implemented will be explained thoroughly with each Pull Request, which will also be tested and evaluated by at least 3 parties prior to deployment.
- The application will be serverless at it's core, which means that other than optimising implementation, there won't be a server to maintain.
- Code coverage must be no less than 75% by project completion.

4.5. Useability

Definition

The software should be easy to use and navigate, even for users who are not technologically inclined/ experienced.

Acceptance Criteria

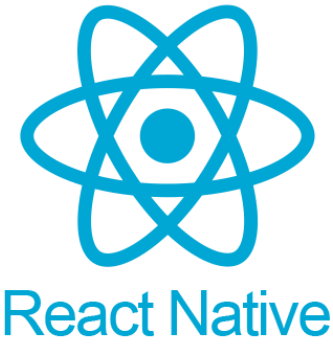
- UI must be as easy to use and clean as possible and participants should have positive feedback about the app’s useability.
- Upon useability testing any given user should be able to carry out a task within 20 seconds or less.
- Users should not fail to complete more than one of the activities asked to carry out.

Useability interviews and questionnaires will be carried out for our assessment.

5. Technology Choices

Front end - **Mobile Application**:

We chose React Native as our mobile front-end technology. It's an open-source framework that allows JavaScript code to create native-like applications for iOS and Android platforms. React Native's use of native components and APIs ensures a rich and performant user experience.



Pros and cons of React Native:

- Code Reusability: React Native enables code reuse, reducing development time and cost by sharing code between web and mobile apps.
- Time and Cost Efficiency: With React Native, one codebase serves both iOS and Android platforms, saving development time and cost.
- Native-like Performance: React Native achieves smooth and responsive user interfaces, although it may have performance limitations in complex animations or graphics-intensive tasks. However, these limitations don't affect Pronto as it doesn't rely on heavy animations or graphics.

Alignment with the Selected Architecture:

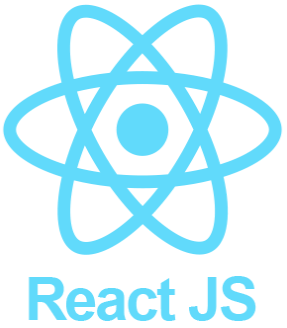
- 3-Tier Architecture: React Native fits the presentation layer, separating the user interface and backend services with clear boundaries.
- Component-Based Architecture: React Native's component-based approach aligns with the chosen architecture, promoting code organisation, modularity, and reusability.
- Serverless Computing Architecture: React Native integrates well with serverless computing, enabling secure and scalable communication with selected AWS backend services like Lambda, DynamoDB, and GraphQL.

Other similar technologies:

While Flutter and Xamarin can achieve cross-platform, native applications, React Native has advantages:

- Larger Ecosystem: React Native has a broader ecosystem and community support, providing more resources for development.
- JavaScript: React Native's use of JavaScript aligns with our team's existing expertise, making it more accessible compared to Flutter, which requires learning Dart.
- Platform Maturity: React Native's longer existence has resulted in a more mature and stable platform compared to Flutter.

Front-end - **Web Application**:



ReactJS is the choice we made for the web application of the Pronto project. This is due to its component-based architecture, virtual DOM, rich ecosystem, reusable components, and JSX syntax.

Pros, cons, and alignment with architecture:

With ReactJS, UIs can be broken down into reusable components that efficiently update and render when data changes.

The creation of reusable components in ReactJS improves productivity and code quality, reducing duplication and making maintenance and extension easier. This aligns with our chosen component-based architecture.

ReactJS aligns well with the selected 3-tier architecture by separating the user interface from backend services, maintaining a clear separation of concerns. It is also compatible with serverless computing architecture, consuming APIs efficiently.

Other similar technologies:

Compared to similar technologies, ReactJS is considered lightweight with a smaller learning curve than Angular, and it has a larger community and ecosystem. When compared to Vue.js, ReactJS shares similar architectural principles and performance but benefits from a larger community and more available resources.

For styling and components, we opted to use material UI in combination with Bootstrap. This is because these technologies are well integrated with ReactJS and have a visually appealing and consistent UI. Bootstrap offers a comprehensive design system with a wide range of customizable components and a responsive grid system. Both frameworks facilitate rapid development with ready-to-use components.

Back-end - **(Database)**:

Pronto has chosen Amazon DynamoDB as the backend database for scheduling data storage and retrieval. DynamoDB's scalability, performance, alignment with the chosen architecture, and comparisons to similar technologies are the main reasons for this decision.

Pros, cons, and alignment with architecture:

DynamoDB's scalability enables handling large volumes of scheduling data and adapting to workload demands. It provides seamless scaling in terms of storage and throughput, ensuring performance and avoiding downtime. DynamoDB aligns well with Pronto's chosen architecture, especially in a serverless computing environment, integrating smoothly with AWS Lambda and GraphQL.

DynamoDB's performance is excellent, delivering low-latency, high-throughput access for real-time scheduling applications. Its distributed nature and data replication enhance reliability and availability.

However, DynamoDB has limitations, such as the lack of support for complex relational queries. Data modelling and query design require careful consideration.

Other similar technologies:

Compared to Apache Cassandra, a competitor, DynamoDB stands out due to its managed service approach, eliminating database administration tasks. DynamoDB's integration with AWS services like CloudFormation and IAM enhances ease of use and manageability.

Back-end - **(API)**:

Pronto has decided to implement a GraphQL API using AWS AppSync or AWS API Gateway, as it believes that GraphQL is the ideal technology to enable efficient data retrieval and manipulation. GraphQL's flexible and precise query language provides several advantages that align with Pronto's chosen architecture and communication requirements between the frontend and backend components of the application.

Pros, cons, and alignment with architecture:

One of the key benefits of GraphQL is its ability to allow clients to request only the specific data they need. With GraphQL, clients can send queries that define the exact data requirements, avoiding over-fetching or under-fetching of data. This minimises network payload and reduces the number of round trips between the client and server, resulting in improved performance and reduced bandwidth consumption.

Additionally, GraphQL enables clients to retrieve multiple resources and related data in a single request. Through its concept of "resolvers," GraphQL consolidates data from various sources into a single response, enhancing efficiency and reducing the number of API calls. This capability is particularly valuable for Pronto, as it aims to retrieve and manipulate scheduling data efficiently.
One potential downside is the increased complexity of setting up and managing the GraphQL server and schema. It requires defining and maintaining the schema, resolvers, and relationships between data entities.

Other similar technologies:

When comparing GraphQL to similar technologies, such as RESTful APIs, GraphQL stands out in several aspects. RESTful APIs often require multiple endpoints to retrieve specific data, leading to over-fetching or chaining of requests. In contrast, GraphQL allows for more precise querying, eliminating the need for multiple round trips and reducing unnecessary data transfer. Additionally, with RESTful APIs, versioning and managing different versions of the API can become complex, whereas GraphQL's schema evolution capabilities make it easier to introduce changes without breaking existing clients.

Other:

Blob Storage:

The scheduling application shall utilise Amazon S3 for storing and serving files such as campus maps, logos, or any other blob-like data associated with the scheduling system. S3's durability and scalability will ensure reliable storage and retrieval of files while providing a secure access mechanism.

Authentication:

Pronto shall utilise Amazon Cognito for user authentication and authorization. This service will handle user sign-up, sign-in, and management of user credentials, ensuring secure access to the application's functionalities.

Testing:

Unit testing:

We carry out unit testing for all components in the mobile application, as well as the web application using jest. This testing framework is very easy to use and works well with react and react native applications which makes it perfect for our purposes.

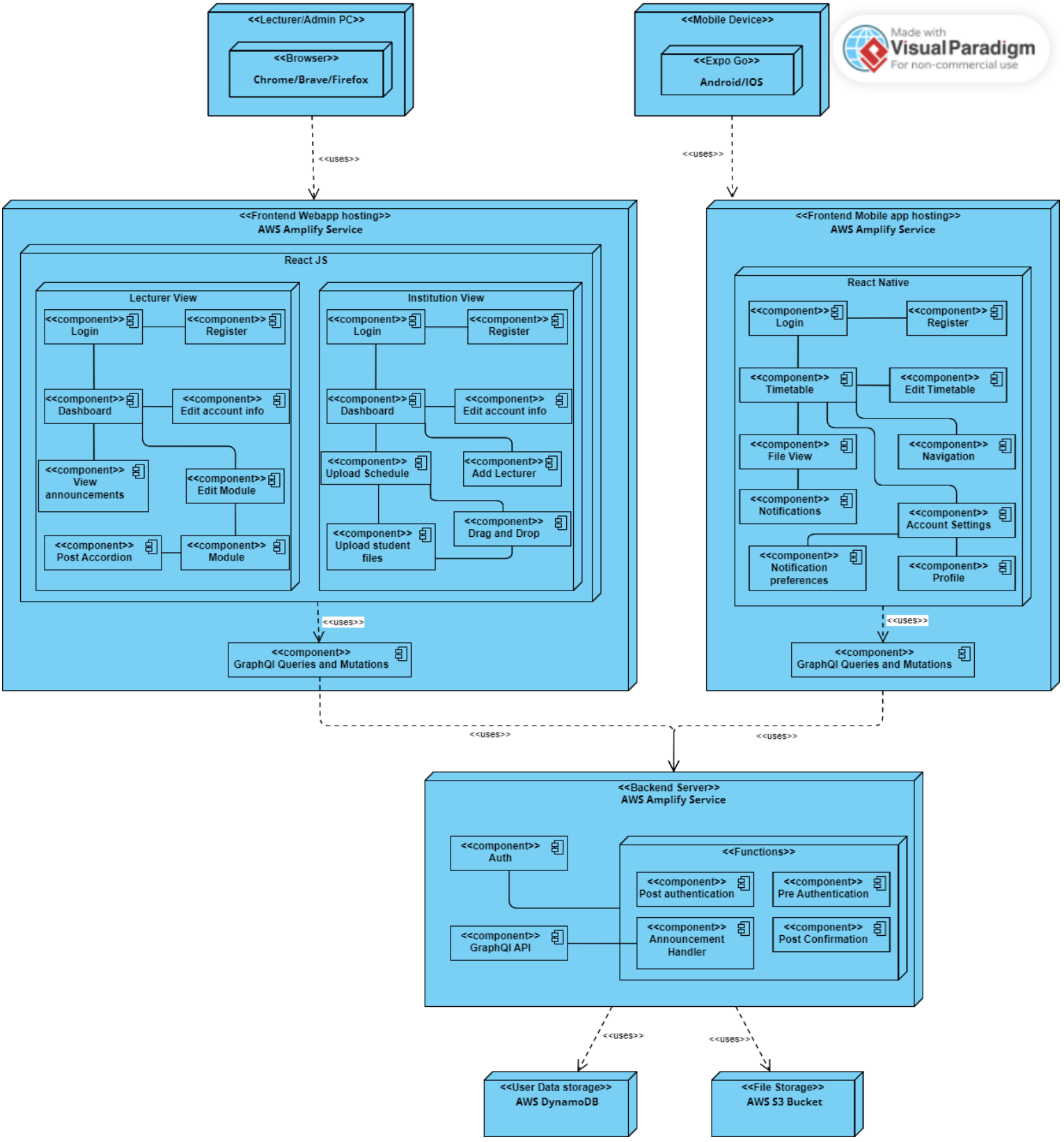
Integration/end-to-end testing

Cypress is our “tool” of choice for end-to-end testing as it allows us to simulate the users interaction with the app and all combinations of possible actions that they may take, and thus, thoroughly test the integration of all components of the application. It is a quick and easy to use tool that has greatly helped us in testing and improving the quality of our application.

6. Code Quality Standards

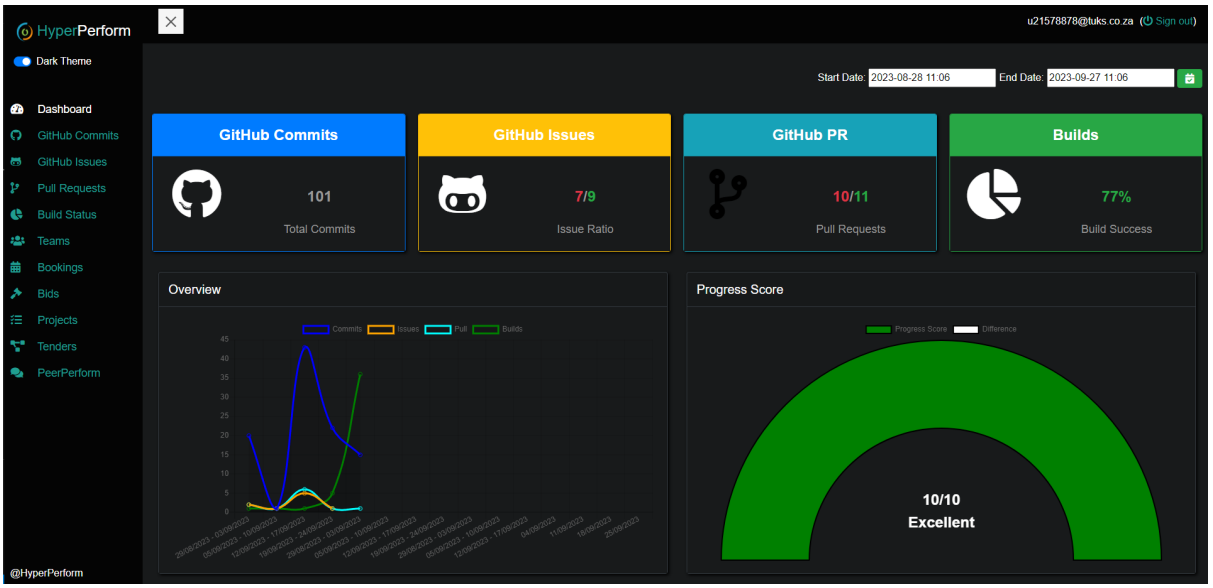
Readability: We coded using meaningful variable and function names, writing clear comments, and organising code into logical sections.
Formatting: We followed consistent use of indentation, line length, spacing, and the placement of braces. We used GitHub Super Linter to ensure that the code is formatted uniformly.
Documentation: We documented new additions of code in pull requests, explained its purpose and any important considerations. In conjunction with comments within the code, we included separate documentation files in our pull requests.
Modularity: We structured our code into reusable functions and components that perform specific tasks. This helps improve maintainability, reusability, and testability of the code.
Error Handling: We use appropriate error messages in the UI, and use try catch statements to handle exceptions as needed. The error messages can be shown to users in the form of alerts or error boxes.
Testing: We have frontend and backend unit tests, integration tests to validate the code's functionality and ensure that code is reliable.
Security: Care has been taken and common secure coding practices such as input validation, have been used to ensure that areas of vulnerability (e.g input from a user) are secured against common attacks.
Version Control: We used GitHub for version control of our code, with clear commit messages, a GitHub flow branching strategy, and creating pull requests which must be reviewed by 3 members before being merged.
Code Reviews: The team participates in code reviews, providing constructive feedback in the comments of pull requests and addressing any identified issues. At least three members need to approve a pull request for it to be merged.

7. Deployment Overview Diagram

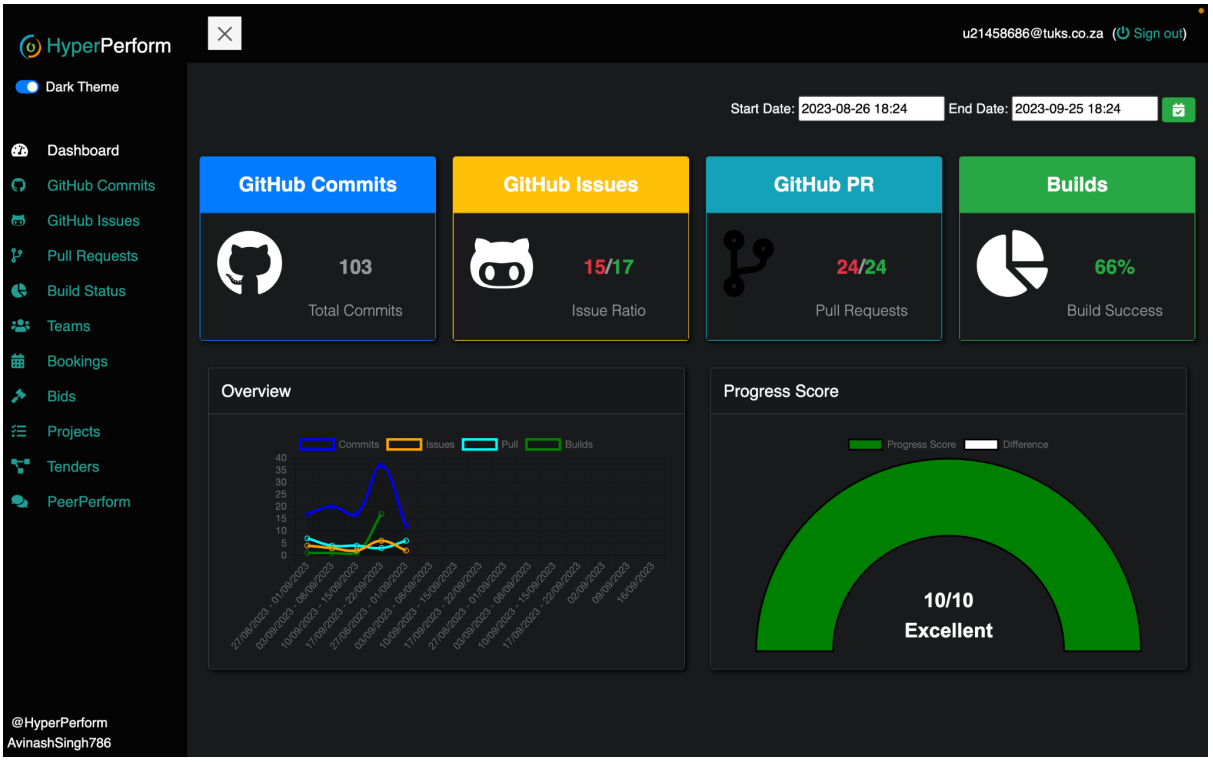


8. Contributions

Tyrone

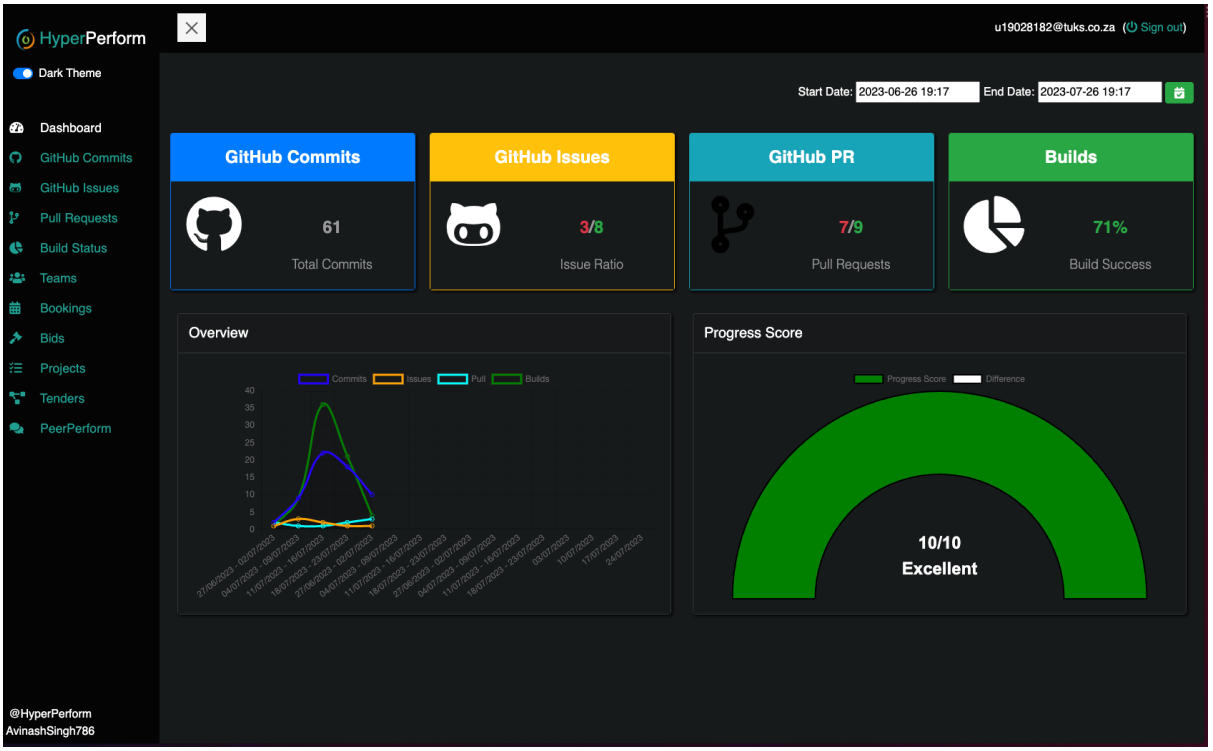


Shashin

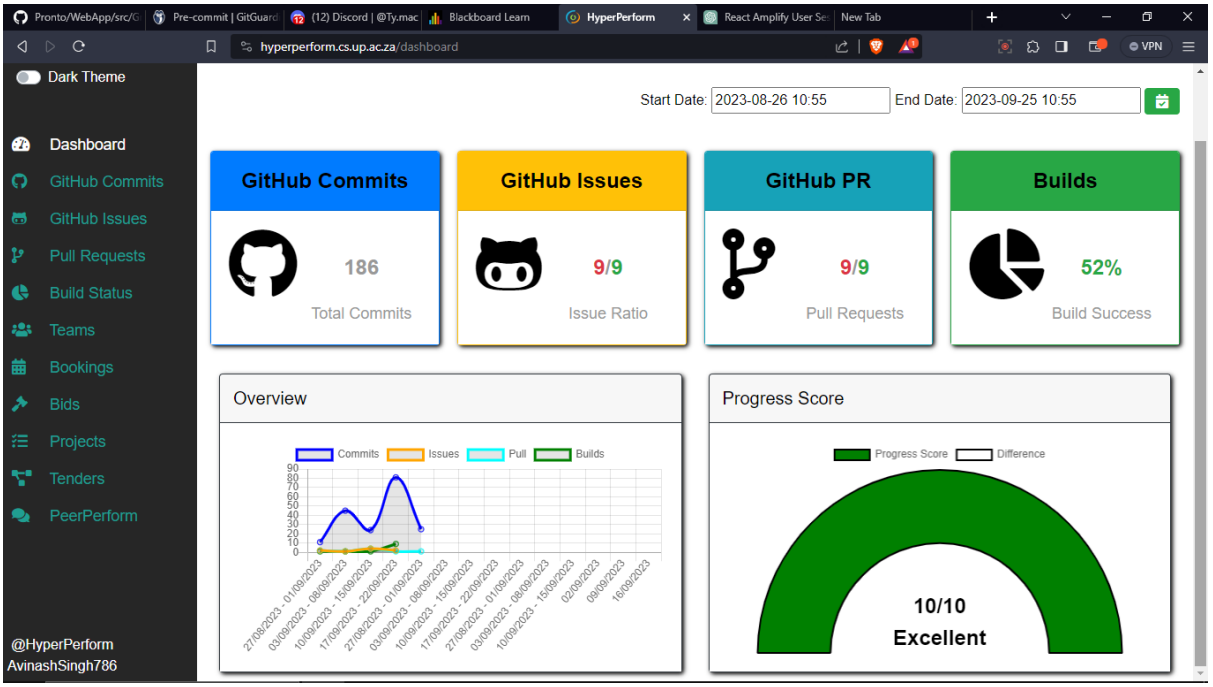


Last Edit: 2023/09/25

Bandisa

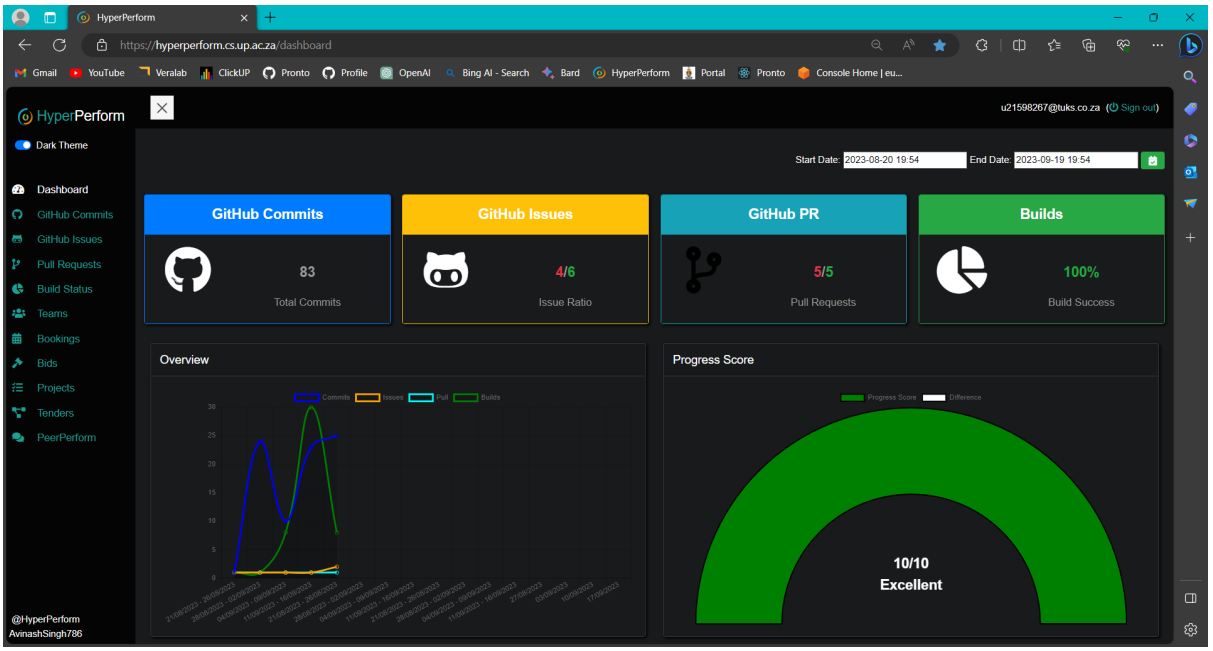


Andile



Last Edit: 2023/09/25

Jonel



Last Edit: 2023/09/25