Chandrasekaran Akash

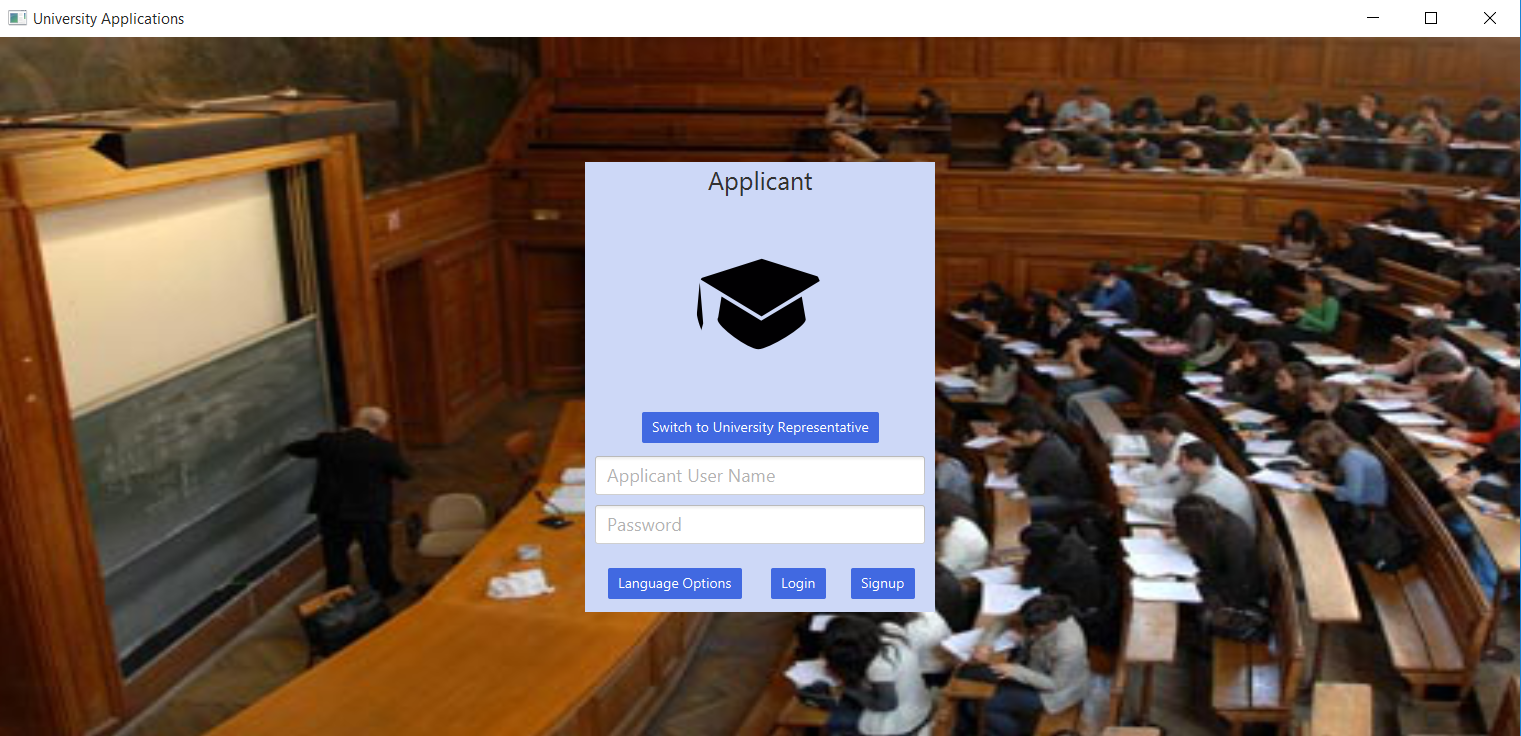
CS Project Report

Semester 2 : UniversityApplicant App

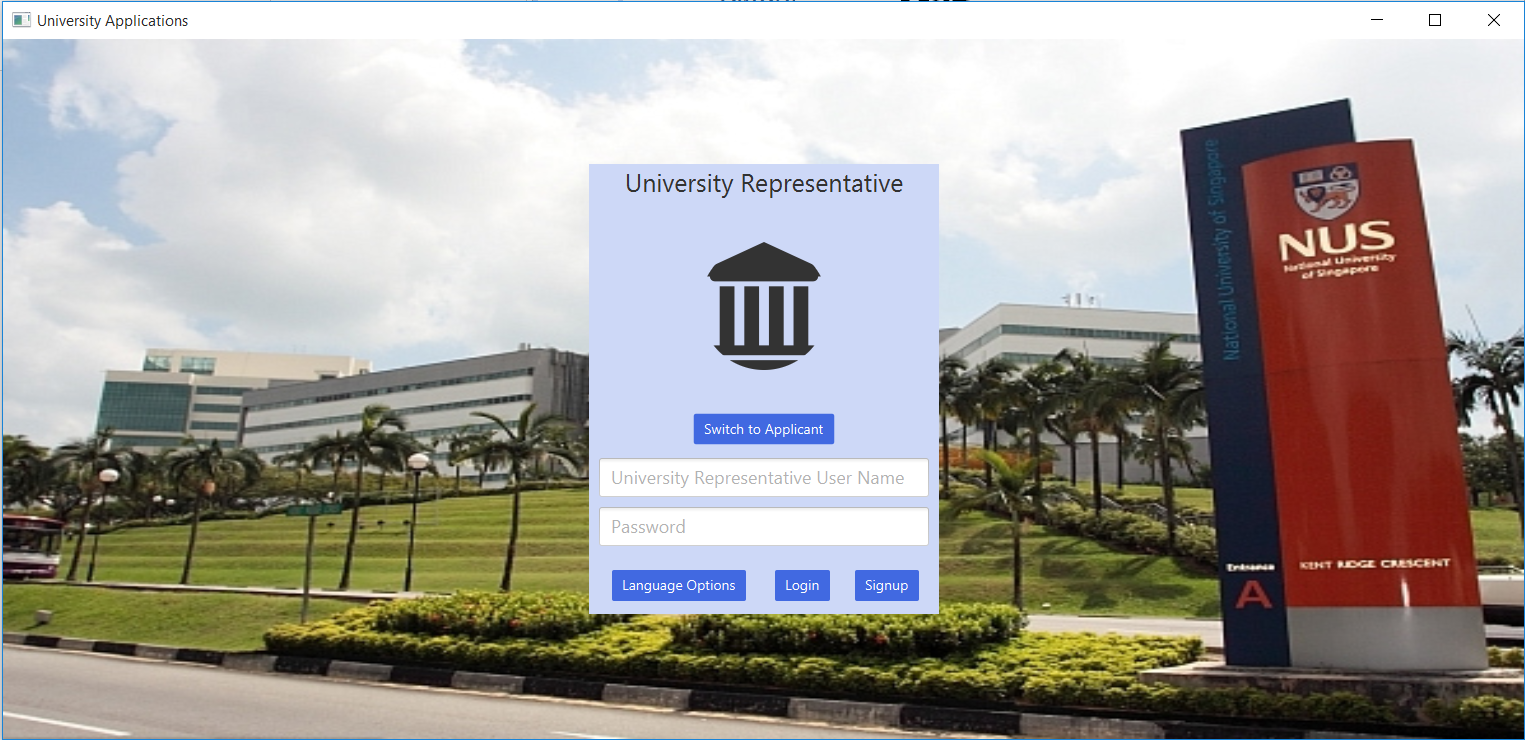
# Overview

The UniversityApplicant App is an app meant for both Applicants and University Representatives. The main aim is to allow applicants to easily sign up for courses from different universities, and chose which university to go to depending on their acceptance into the courses. For the university representatives, it is meant as an easy tool to allow choosing the students for the course depending on their qualifications and the prerequisites met. It supports French and English languages.

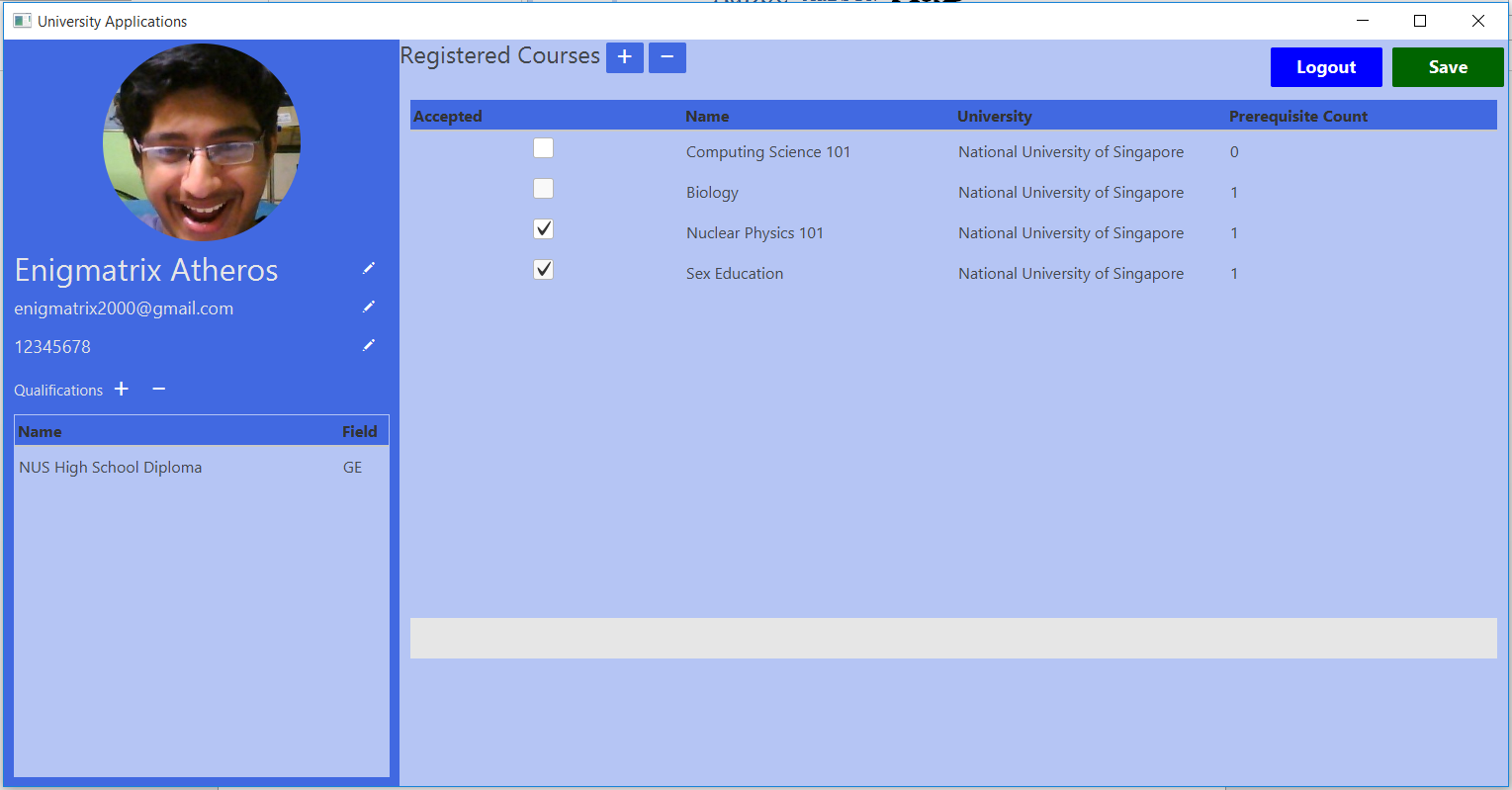
Applicant Login



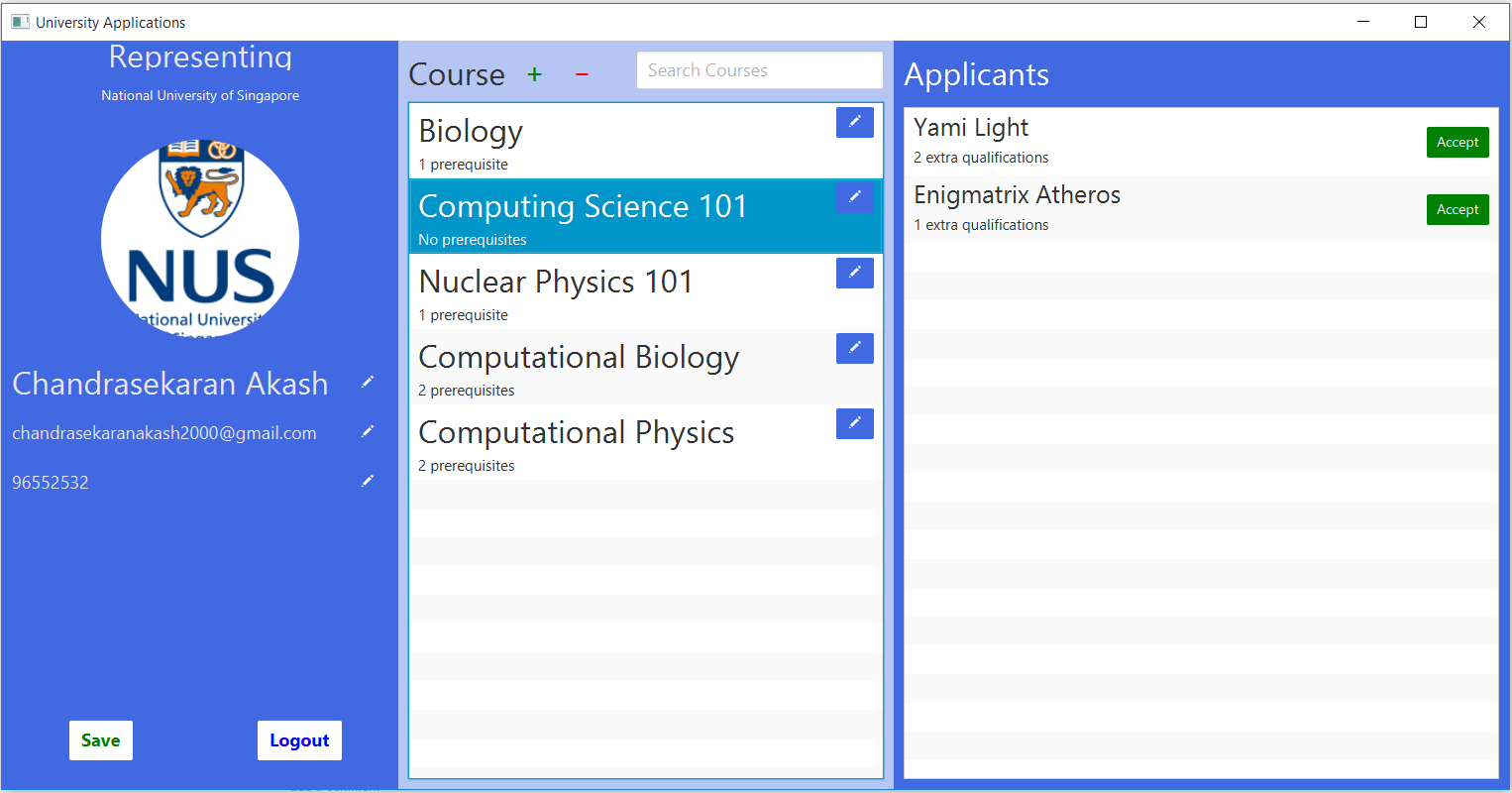
University Representative Login



Applicant Home Page



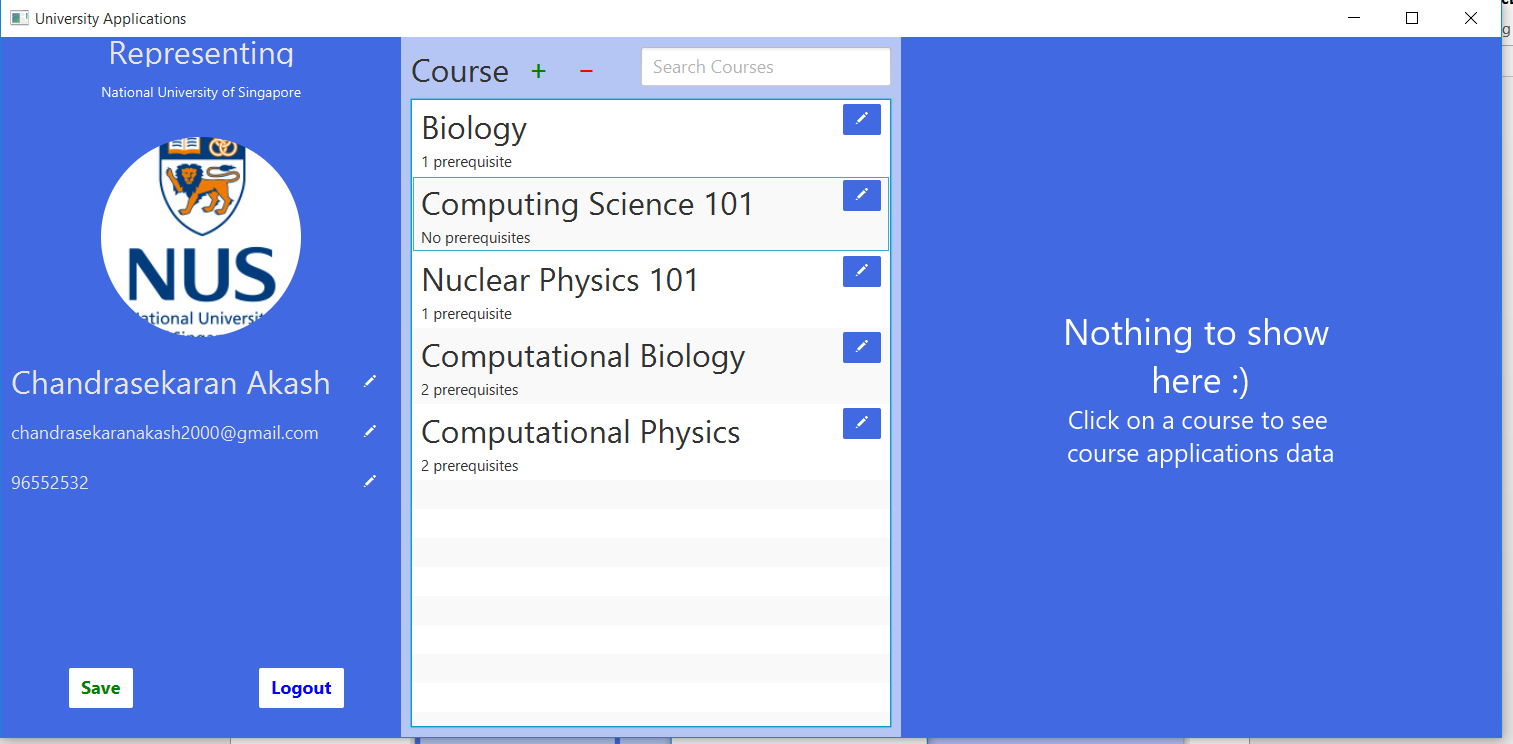
University Representative Home Page

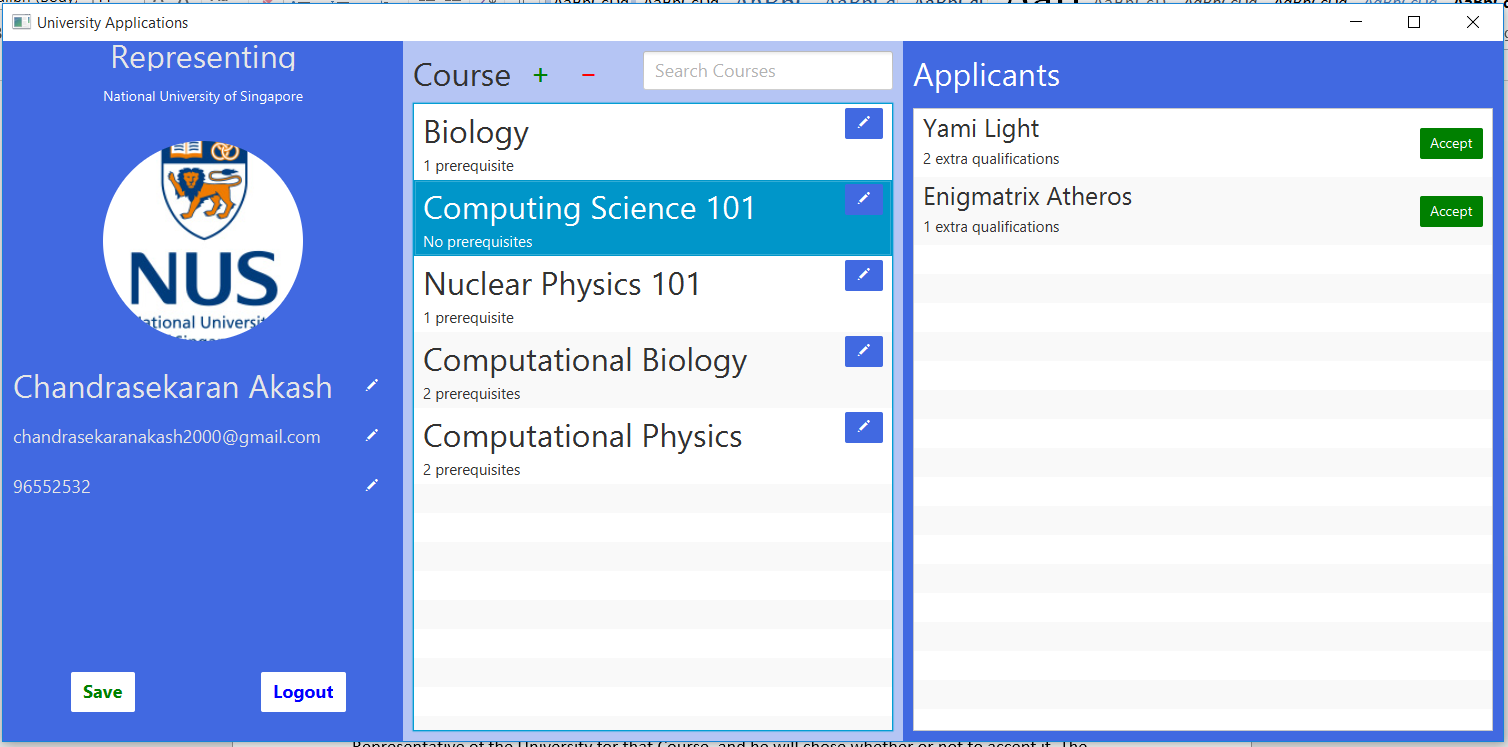
Code explanation of key algorithms and features

## Data Persistence

All data is store inside a .json file, serialized using the Jackson Json library (imported using Maven). The reason for using json instead of csv/txt is that my datamodels are too complex to express in those data formats. Furthermore, my datamodels have circular references, which csv/txt can’t handle very well (while Jackson can, with a few modifications). Furthermore, I did not want to write a serialization method for each of my datamodel classes. When the application loads / saved, all the data is deserialized / serialized, respectively. Since the json file can get large very quickly, loading and saving will become a hassle, and multithreading is used here to avoid a unresponsive client.

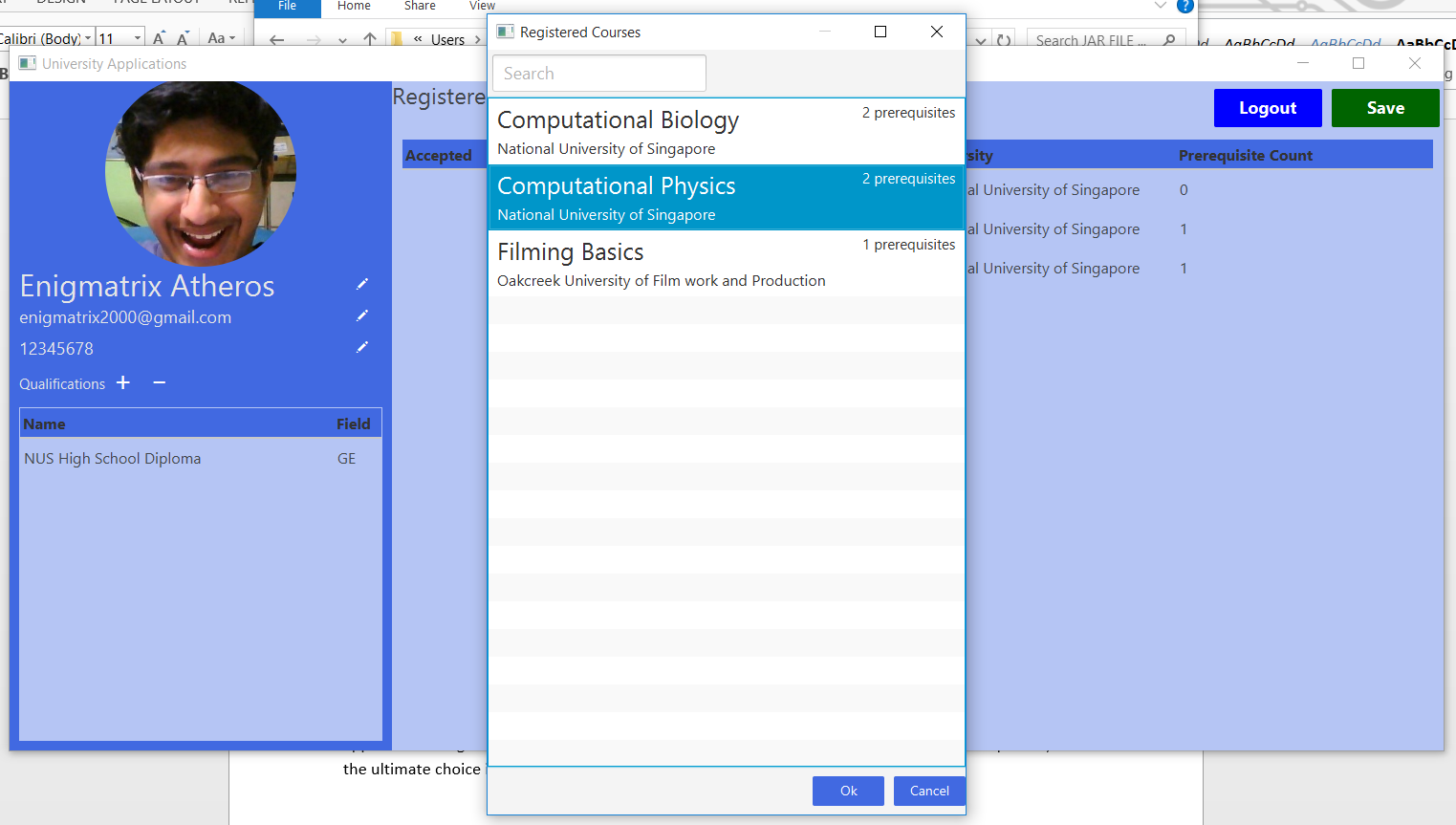
## Aesthetics

The color for the application is royal blue. I am using some material design themes from a maven package (not in maven core repository, but can be installed) called sdkfx from com.guigarage. Elements such as the RoundImageView, button style and table view coloring are from that library. Furthermore, I am using custom controls (e.g. the course is represented on a list view by a custom listviewcell, the text with the edit icon). This is done by using a User Control class that is extensible, from GitHub: [Javafx-UserControl.](https://github.com/BenjaminGale/JavaFX-UserControl/blob/master/Core/src/com/github/benjamingale/usercontrol/UserControl.java) For animations, there are subtle fade transitions on the University Representative View. 



## Course Application (Applicant View)

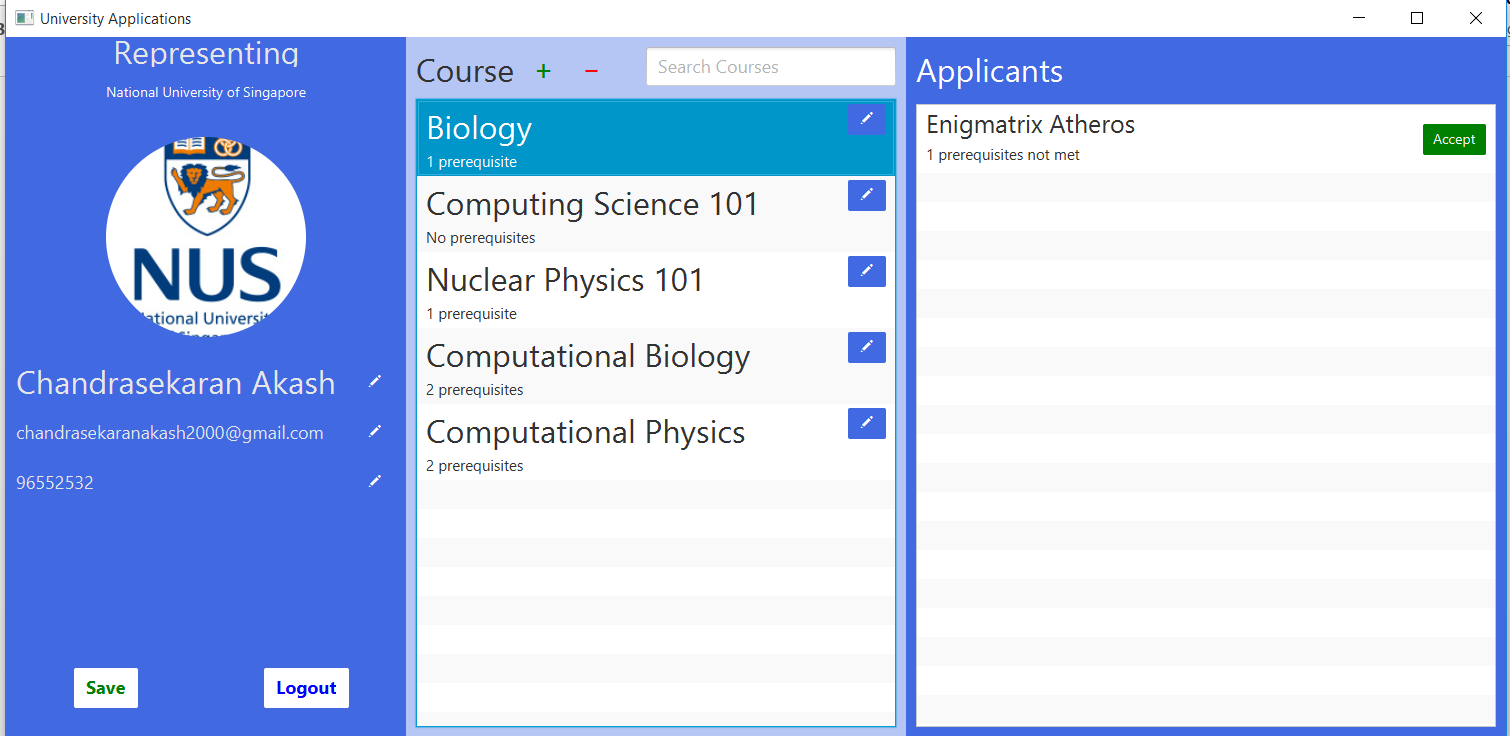
When the applicant send an application for a course, that application is sent to the University Representative of the University for that Course, and he will chose whether or not to accept it. The applicant can register for any course from any university, even if he doesn’t meet the prerequisites, but the ultimate choice is up to the University Representative.

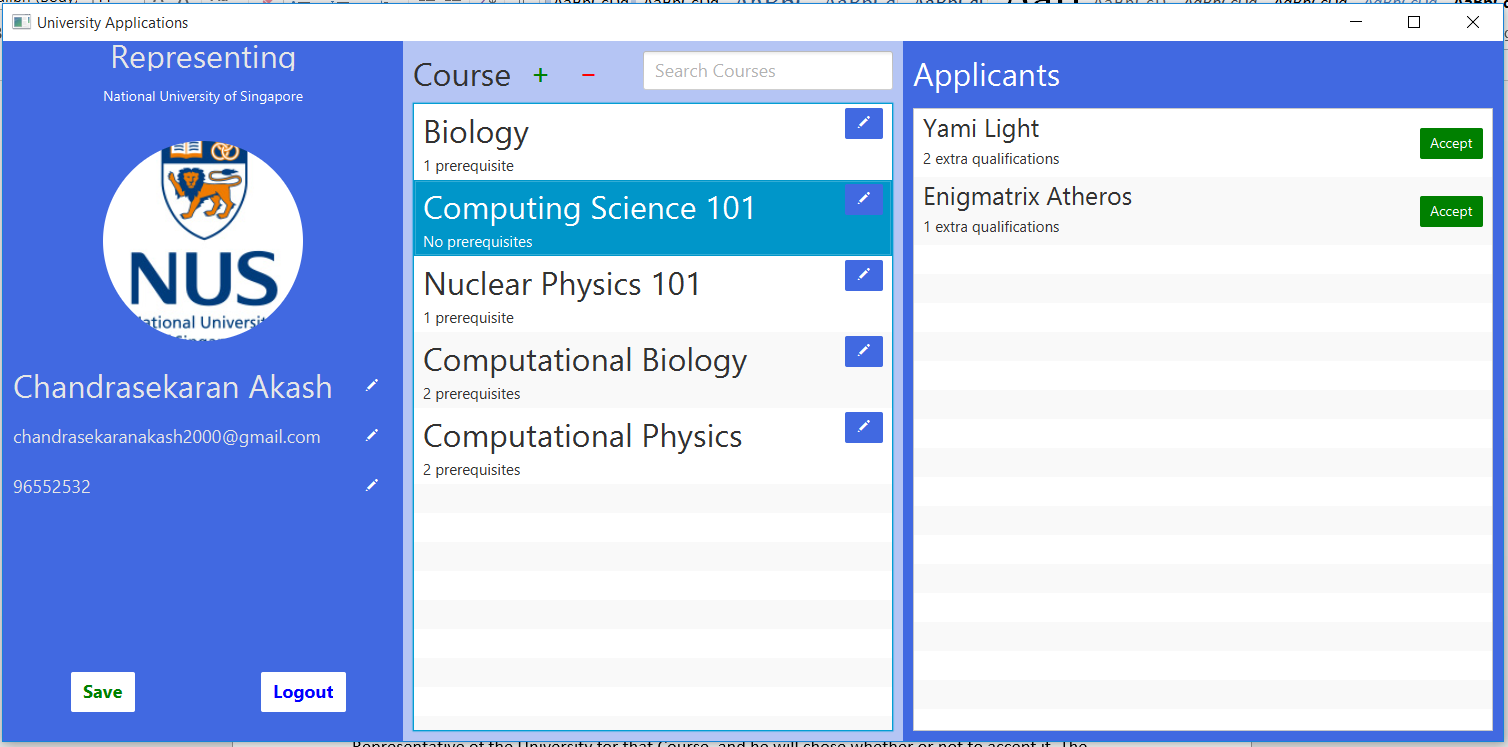


More details can be seen by double clicking on the Course

## Course Application (University View)

For the University Representative, he can add / remove courses and accept / reject applicants. Data about course applicants include whether or not he fulfills all prerequisites, whether he has extra qualifications and how many prerequisites he has not achieved.





# Testing Strategy

There is nothing much to say here. Some of my tests include remnant data testing (e.g. if a course is deleted, then all course applications for it should also be deleted, but the applicants who signed up shouldn`t be deleted.), which is done by manually checking the dataStore.json file after saves (at least the json is pretty printed else my eyes would hurt!). Furthermore, I tested the performance for the application in terms of its largest bottleneck, data persistence. Results are below

|  |  |  |
| --- | --- | --- |
| Test Number | Data Comments | Result (in terms of millisecond for saving) |
| 1 | 2 uni rep, 2 applicants, both have images | 1206 |
| 2 | 2 uni rep, 4 applicants, all have images | 1533 |

As you can see, data persistence is a pretty large bottleneck that affects application responsiveness, so I have multithreaded it.

# Reflections

## Obstacles faced

* Javafx`s CSS has a habit of being completely misleading
* Javafx`s color transparency is a bit buggy
* Maven build / Maven package commands have a tendency to be extremely slow, and packaging to a jar file was a hassle
* By using Custom User Controls, I cannot use scene builder since it cannot detect the controls and ends up not loading the entire file. I had to resort to manually editing the FXML file.
* Centering elements is hard.
* Decoding / encoding and storing images is a hassle. Furthermore, they take up a lot of space

## Learning

I have learnt how to use JavaFX from FXML, its CSS styling, and how to include Resource Bundle Strings to the FXML. Furthermore, I have learnt the typical pattern of JavaFX GUI Apps, User Experience and Animations. Also, I have learnt about the different libraries available and the Maven ecosystem.

## Possible Improvements

Storing the data as json is quite primitive. Real life applications store the data into a database. Options exist for this task, such as Hibernate ORM (Object-Relation Mapper) and the Oracle databases. This will improve save time as now only the changes will be considered, while before I had deserialized all data (including unaffected data) into json.

In a real life scenario, the data will be stored in a web server and there can be dynamic updates on the data that will be reported to the user. I felt it was out of scope of this project and I did not like the Java Server side technologies (too much cruft), while coding in ASP.NET and C# / NodeJS were not Java options.

I also think that I could have made several user experience improvements by leveraging the material design library, sdkfx / material-design-fx. I had also forgotten to include data validation -\_-.

## How could the task be improved

During the demos, the application used were our baseline, and this motivated us to make applications that were as good as those. I fell it would be better if we could increase the baseline by demoing more complex applications