UNSW
AUSTRALIA

School of Computer Science and Engineering

The University of New South Wales

SENG2021 – Requirements and Design Workshop

Term 1 – 2019

Raisin

Date of Submission: 26/04/2019

Submitted by: Richard Liu (z5165455)

Rory Madden (z5208738)

Yorke Li (z5207298)

Faiz Ather (z5170340)

Group Info:

Name: digital_invention (formerly NoName3)

Mentor: Christopher Joy

Meeting Day: Tuesday

Meeting Time: 1:20PM – 1:40PM

CONTENTS

1 VISION & MISSION STATEMENT

We envision helping every WebCSM3 user to view all their course assessments and export them to any digital calendar

It is our mission to provide a simple and broad service to the UNSW community

2 CONCEPT INTRODUCTION

This project aims to address the concerns of a typical UNSW student, who is enrolled in courses that use WebCMS3. Under UNSW's new trimester system, courses are accelerated and due dates can arrive quickly, and sometimes surprisingly. Consequently, little time is left available to prepare and students now instead are prompted to learn course content before knowledge of the course requirements, in order to keep up to the pace of the course content.

Our application caters a niche audience of UNSW students, however there currently does not exist any service that scans, consolidates or automatically exports a course outline into a digital calendar. A typical university student then must read the full course outline or use the Find command (Ctrl + F), manually completing the process of finding all their assessments then creating each as an event in their digital calendar.

Therefore our project intends to create a simple web application for students to view a list of all their WebCMS3 due dates (assignments, labs, project milestones and exams, etc) ahead of time, with the option of being able to export them to their Google or Apple calendar/iCal for viewing later. Ultimately this aims to assist students in being prepared for their courses and provide a swift service so more time can be dedicated towards study.

3 PROBLEM STATEMENTS

(1) There is no way for a UNSW student using WebCMS3 to view all their due dates for a specific course without reading the course outline

(2) A UNSW student must thoroughly search the entire course outline on WebCMS3 to verify they have accounted for all due dates for a specific course

(3) A UNSW student cannot view their entire due dates for all assessable content from all their courses on WebCMS3 in a single centralised place

(4) A UNSW WebCMS3 student must manually insert all their courses' due dates into their digital calendar

(5) A UNSW student has to create their own file of calendar assessment due dates which is compatible with all of their digital calendars

4 SOFTWARE DESIGN

In order to fulfil the purpose of our application, the features that our application will need to provide have been deconstructed into user stories. The relationships between the entities within these user stories will be handled treated by our application as Figure 4.1 indicates, while realistically on the back-end, the connections between these entities will operate as viewed in Figure 4.2.

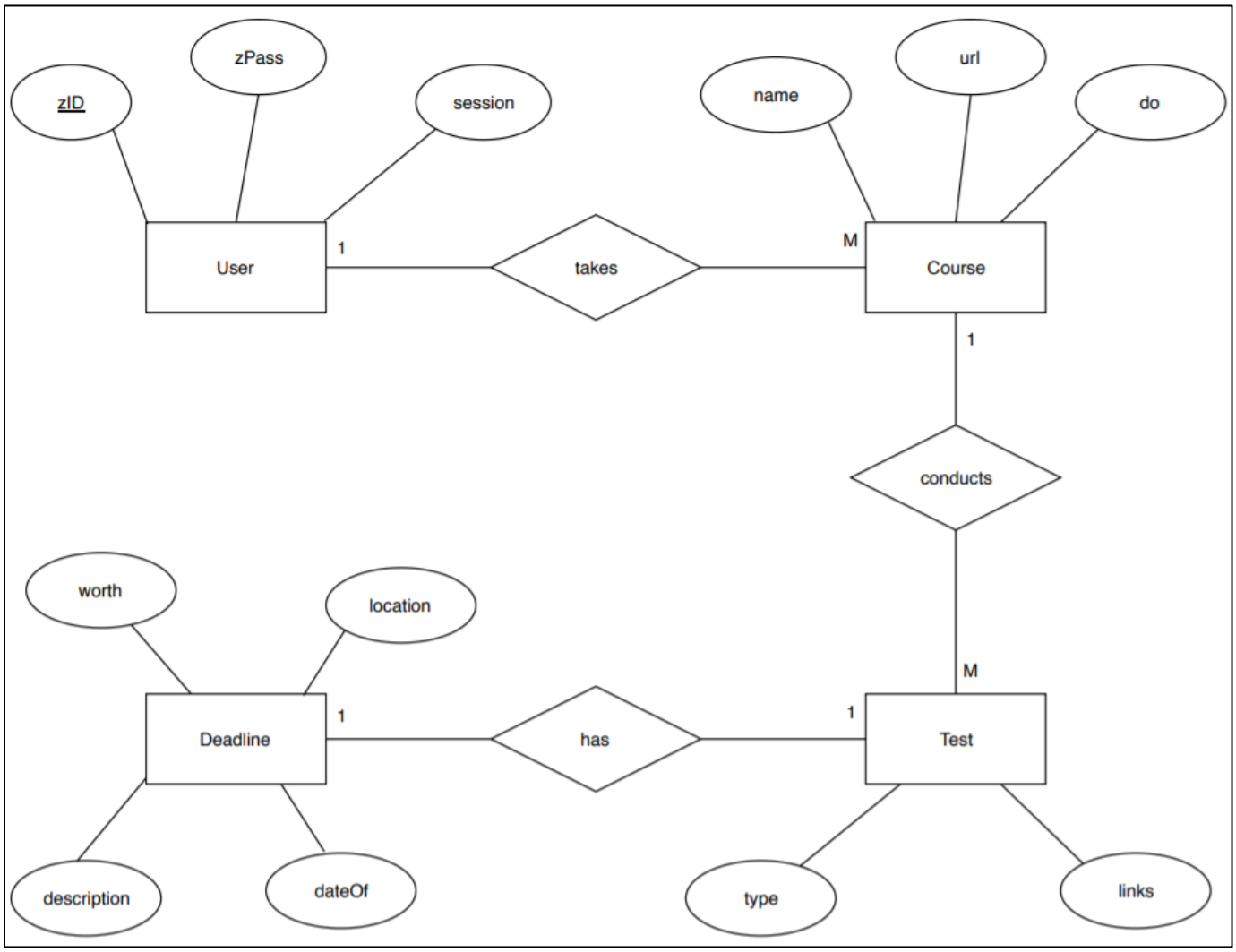


Figure 4.1: System Entity Relationship Diagram

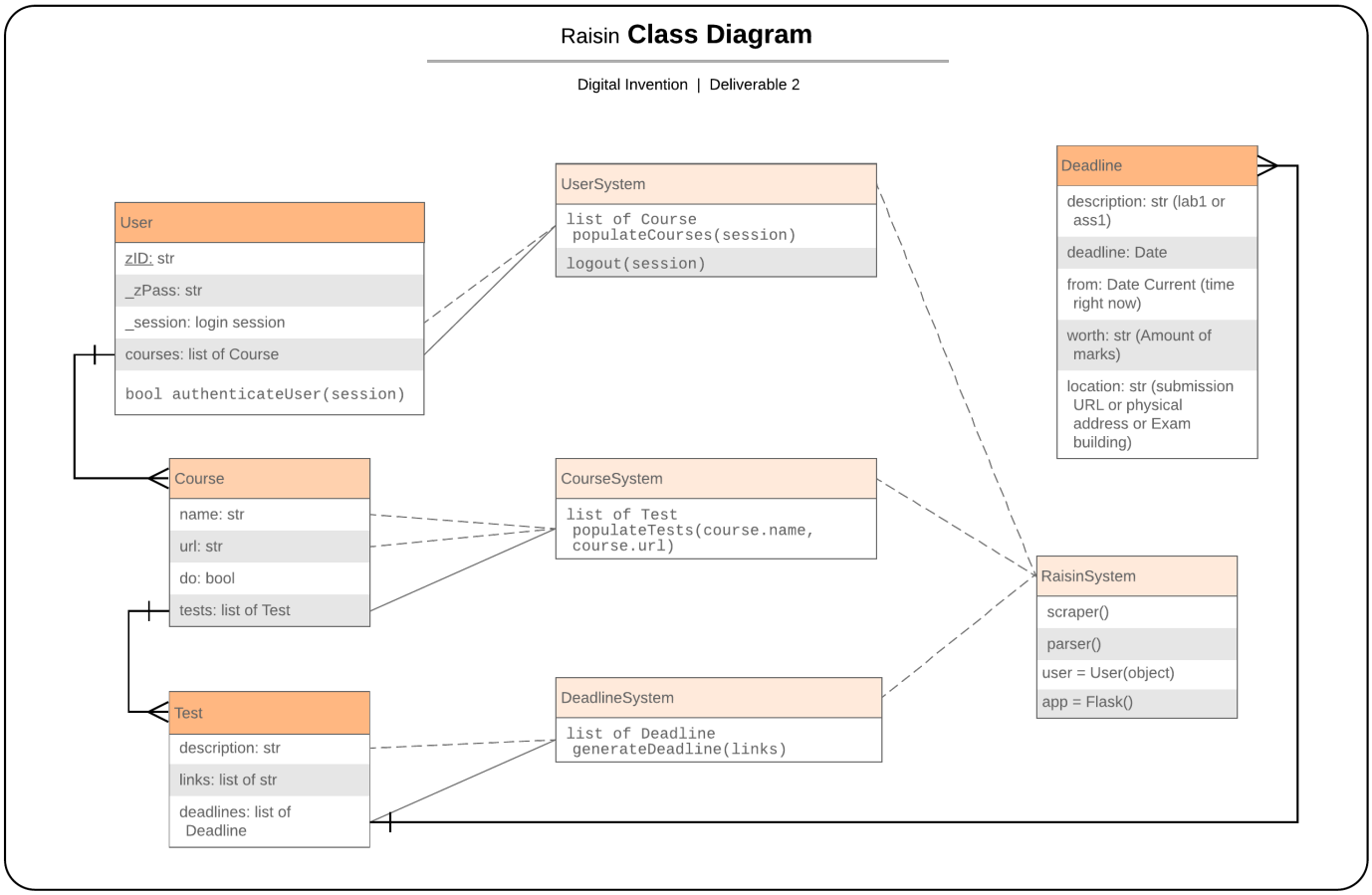


Figure 4.2: Class Diagram of Raisin's System

All of these features have been assessed and prioritised accordingly while also embracing SMART principles so that our project can be realistically visualised to achieve success. Hence the user stories that our application must implement are delineated below.

The figure below explains the priority scale for our project

Priority Level	**Description of feature**	**Urgency**
Priority 1	Essential for the application to function.	Must be completed first.
Priority 2	Improves the quality of the application's necessary functions.	Should be completed directly after all priority 1 tasks are completed.
Priority 3	Improves the quality of the application by providing more useful features.	Can be implemented if time permits, only if all priority 2 tasks are completed.

Figure 4.3: Feature Priority Table

All of the user stories below have been fully implemented and work as intended, except the feature: of being able to view/organise your assessments by week (but the feature of viewing/organising the assessments by course has been implemented). After considering how a typical user may use our product we deduced the feature of sorting assessments by week ultimately achieved nothing. This is because since the assessments themselves will be placed onto a calendar, they will be organised by week. Thus the feature seemed redundant.

Priority Level: 1
Feature
Feature: To login and access all currently enrolled courses **As A** UNSW WebCMS3 student **I Want To** be able to log into my WebCSM3 account **So That** I can use the web application; Raisin
Scenario
Scenario: I just launched the Raisin web application and would like to access its functions/features **GIVEN** I am on Raisin's Login Page **WHEN** I enter my WebCSM3 login details **AND** my account has been authenticated **THEN** I will be redirected to Raisin's features/functions; the Course Assessment Selection page
Estimated Time Required to Integrate Feature: 5 Hours

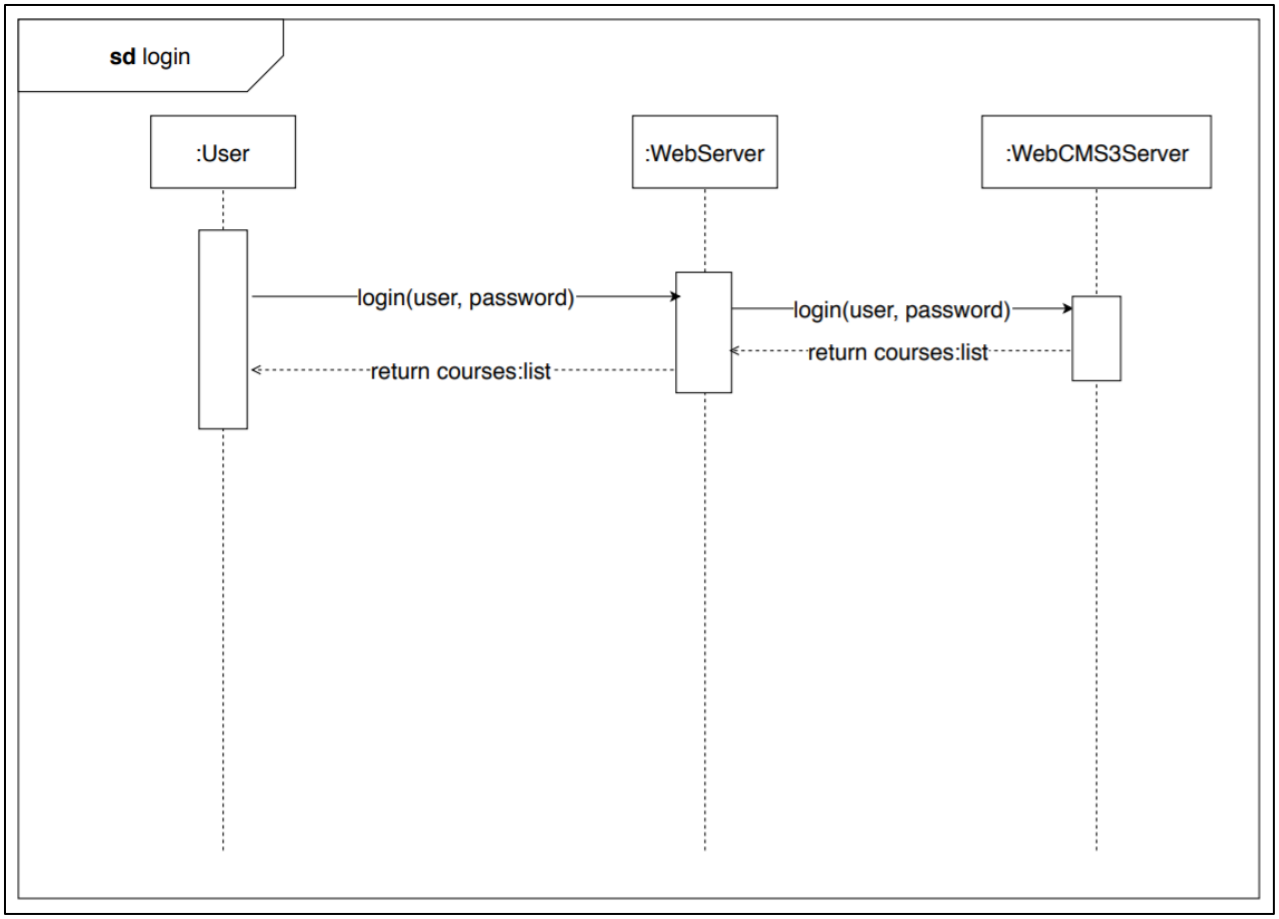


Figure 4.4: Sequence Diagram for Raisin's Login Process

Priority Level: 1
Feature
Feature: View all course commitments **As A** UNSW WebCMS3 student **I Want To** be able to view all my course assessments together on a single page **So That** I am prepared for my entire term
Scenario
Scenario: Course outlines have just been released and I want to find all the assessable tasks & when they are due. **GIVEN** I am on Raisin's Course Selection page **WHEN** I have checked all the boxes for my enrolled courses or clicked the "Select all" button **AND** clicked the "Next" button **THEN** I will be navigated to a Course Assessment Selection page **WHEN** I have checked all the boxes for the course assessments or clicked the "Select all" button **AND** clicked the "Next" button **THEN** I should be navigated to an Assessment Synopsis/"Due Dates" page displaying a summary of all my course assessments and their deadlines
Estimated Time Required to Integrate Feature: 3 Weeks

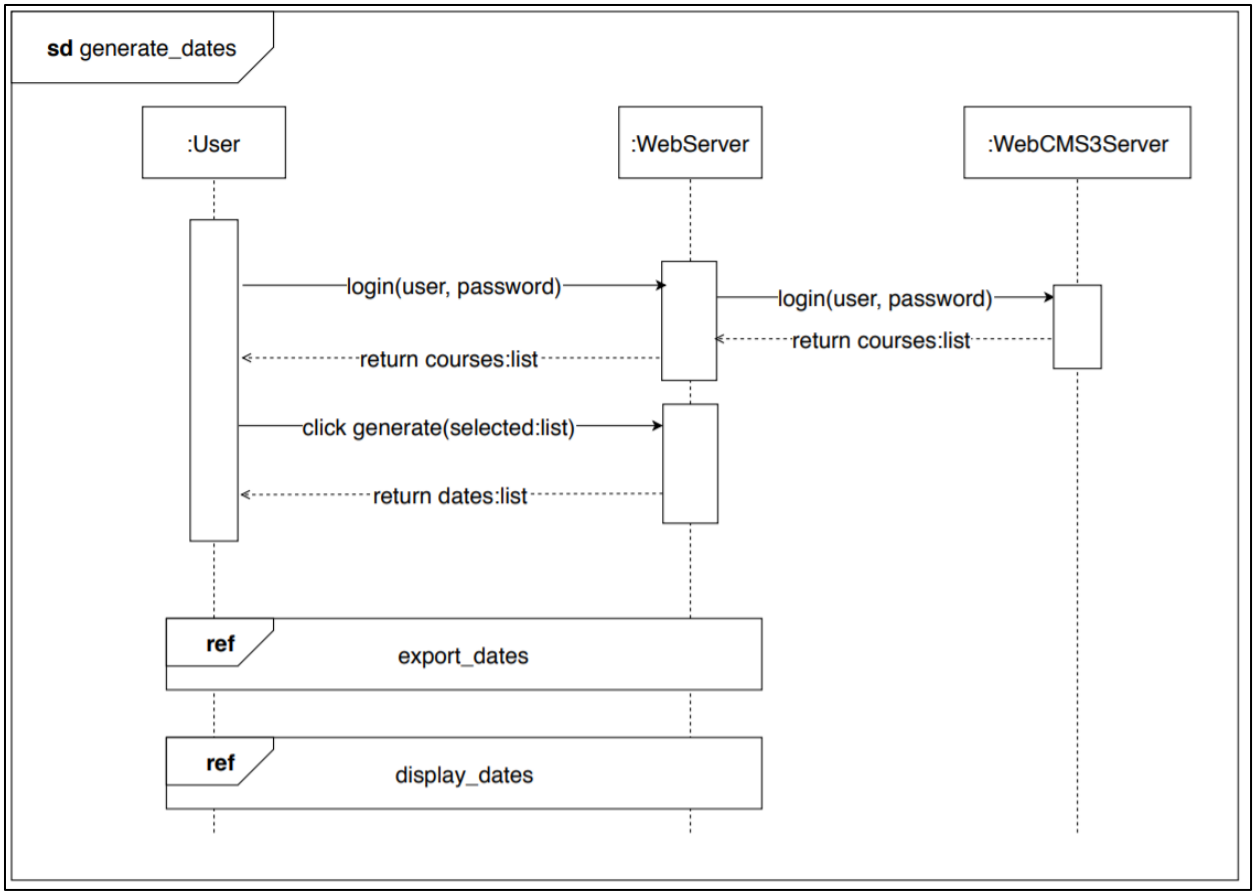


Figure 4.5: Sequence Diagram for Viewing Course Commitments

Priority Level: 2
Feature
Feature: All course assessments can be transferred onto a digital calendar **As A** UNSW WebCMS3 student **I Want To** be able to export all my course assessments to a digital calendar; Google Calendar **So That** I can complete my assessments on time
Scenario
Scenario: Raisin has just completed gathering all of the course assessments and I wish to export them into my Google Calendar **GIVEN** I am on Raisin's Assessment Synopsis/"Due Dates" page **WHEN** I click on the "Google Calendar" **AND** I log into my Google account **THEN** I will be asked to allow permission **AND** a pop up will appear verifying that the assessments have successfully been imported into my Google Calendar **WHEN** I check my Google Calendar, the assessments will be there
Estimated Time Required to Integrate Feature: 1 Week

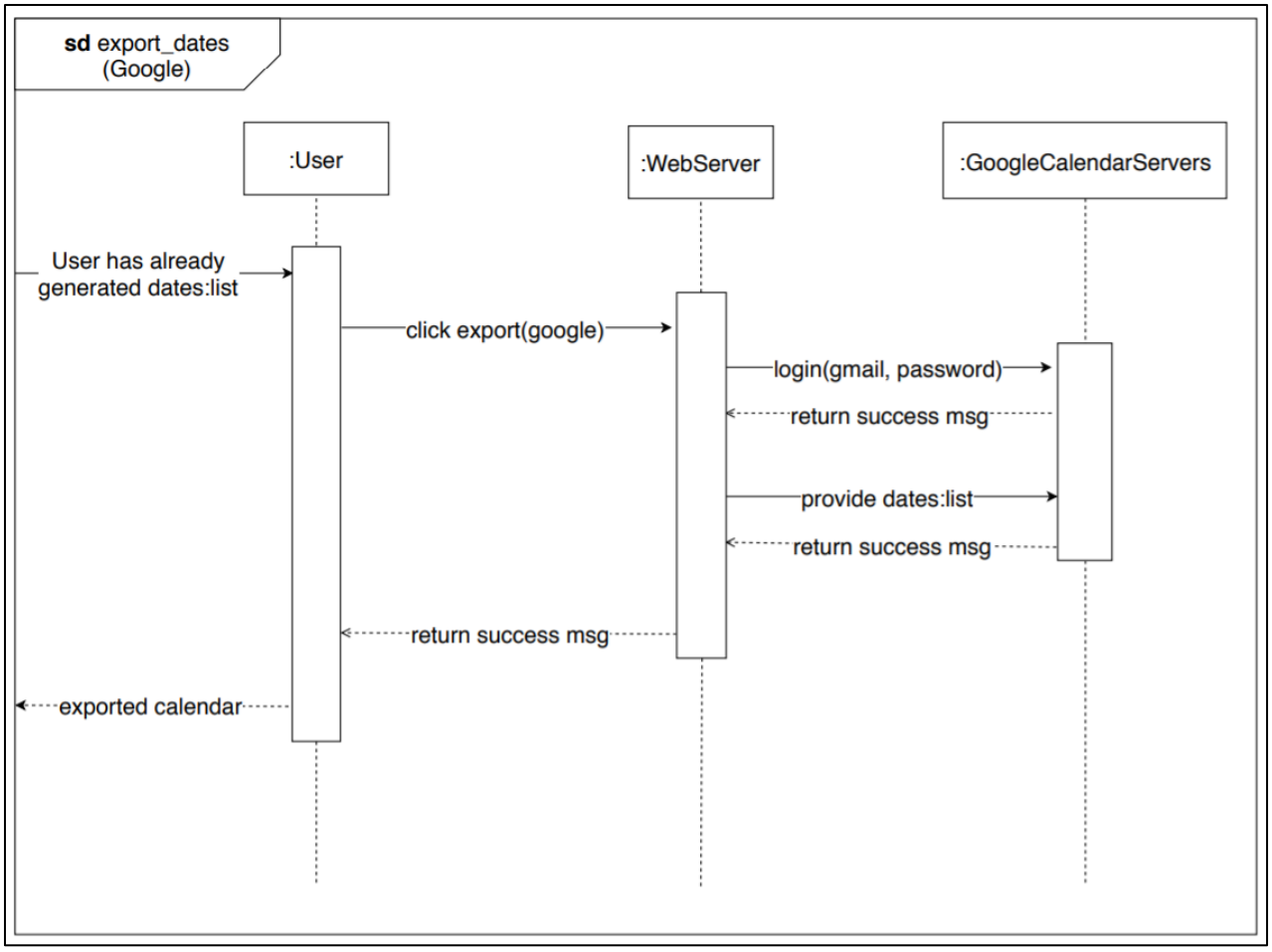


Figure 4.6: Sequence Diagram for Exporting to Google Calendar

Priority Level: 2
Feature
Feature: All course assessments can be transferred onto a digital calendar **As A** UNSW WebCMS3 student **I Want To** be able to export all my course assessments to a digital calendar; Apple's Calendar application/iCal **So That** I can complete my assessments on time
Scenario
Scenario: Raisin has just completed gathering all of the course assessments and I wish to export them into my iCalendar/Apple Calendar **GIVEN** I am on Raisin's Assessment Synopsis/"Due Dates" page **WHEN** I click on the "Apple Calendar" **AND** my course assessments have been compiled and converted into an ".ics" file **THEN** a downloadable file of my course assessments will appear in the "Downloads" of my web browser
Estimated Time Required to Integrate Feature: 1 Week

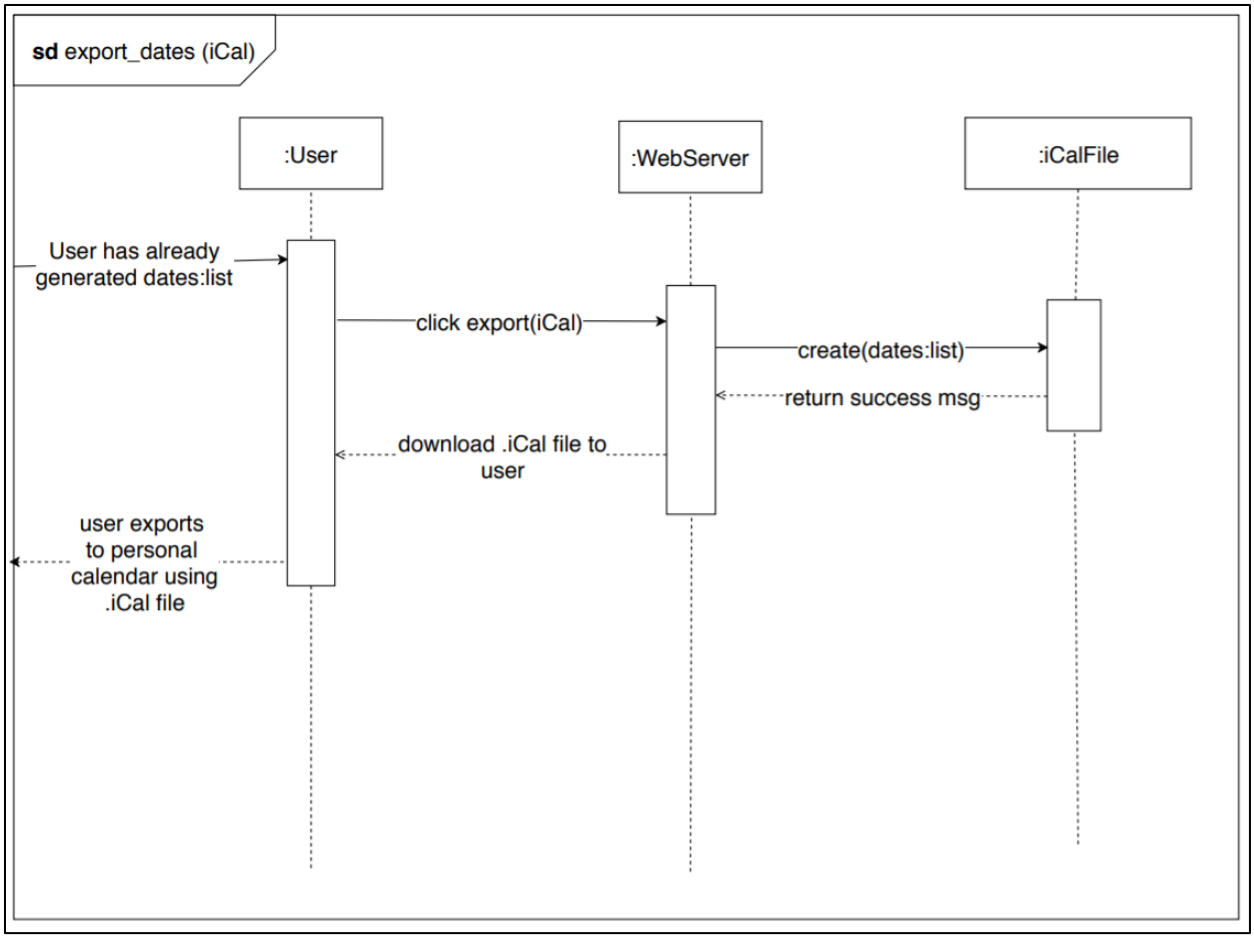


Figure 4.7: Sequence Diagram for Exporting to iCal

Priority Level: 2
Feature
Feature: .csv file is downloadable for students protective of their security **As A** UNSW WebCMS3 student **I Want To** be able to manually import events into my digital calendar; Google Calendar **So That** I don't have to enter my Google account login information through the Raisin website
Scenario
Scenario: Raisin has just completed gathering all of the course assessments and I wish to download a .csv file **GIVEN** I am on Raisin's Assessment Synopsis/"Due Dates" page **WHEN** I click on the ".CSV FILE" **AND** my course assessments have been compiled and converted into an ".csv" file **THEN** a downloadable file of my course assessments will appear in the "Downloads" of my web browser
Estimated Time Required to Integrate Feature: 1 Week

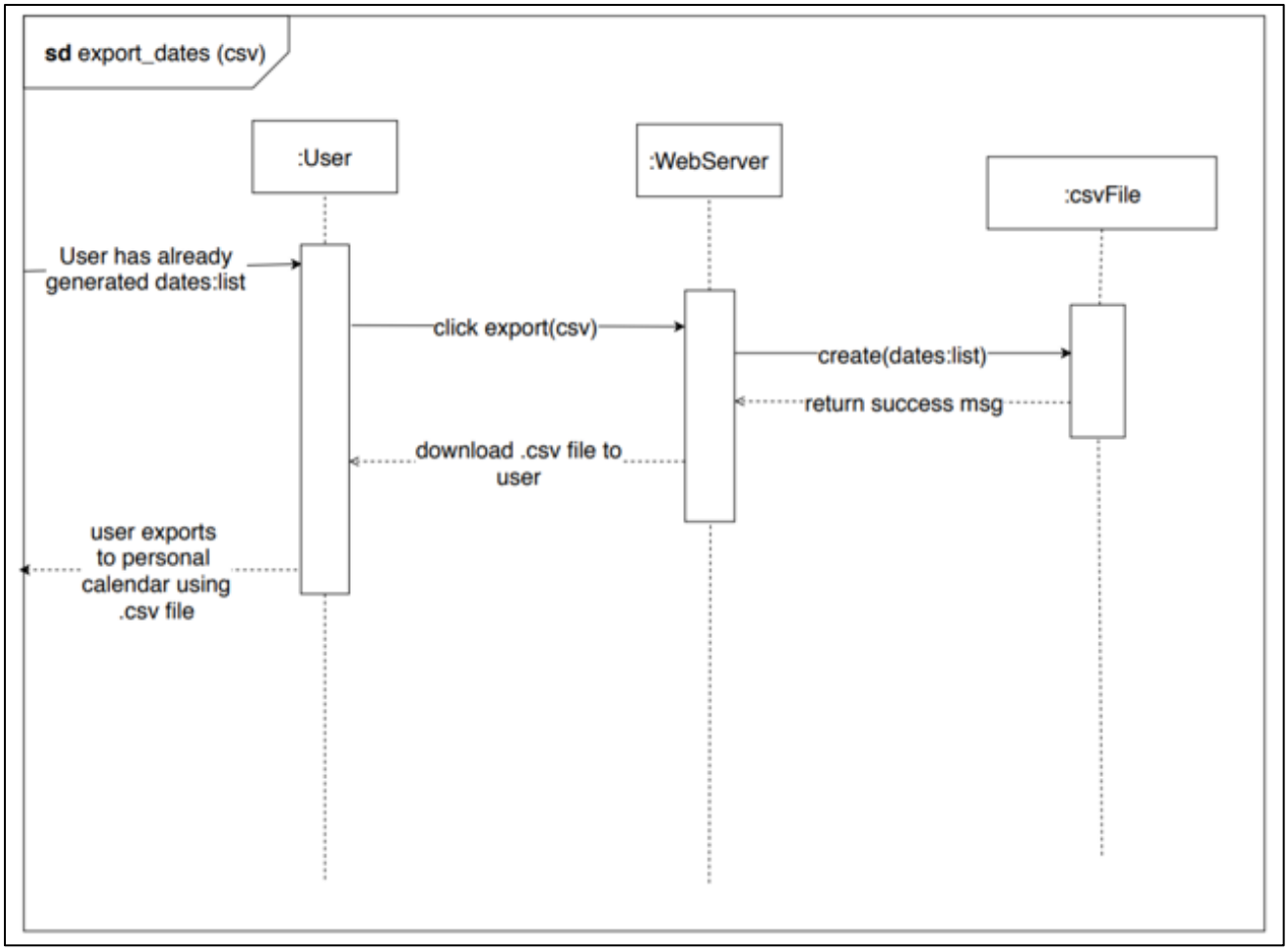


Figure 4.8: Sequence Diagram for Downloading a .csv file

Priority Level: 2
Feature
Feature: All course assessments can be transferred onto a digital calendar **As A** UNSW WebCMS3 student **I Want To** be able to export all my course assessments to any digital calendar **So That** I can complete my assessments on time
Scenario
Scenario: I am using my phone to access the Raisin webpage, and it has just completed gathering all of the course assessments. I wish to send myself an email of the files so I can manually import them to all the digital calendars I use. **GIVEN** I am on Raisin's Assessment Synopsis/"Due Dates" page **WHEN** I click on the "Email" **AND** I have entered my email address into the pop up that appears **THEN** systemraisin@gmail.com will send an email containing an .ics file and a .csv file to the designed email address
Estimated Time Required to Integrate Feature: 1 Week

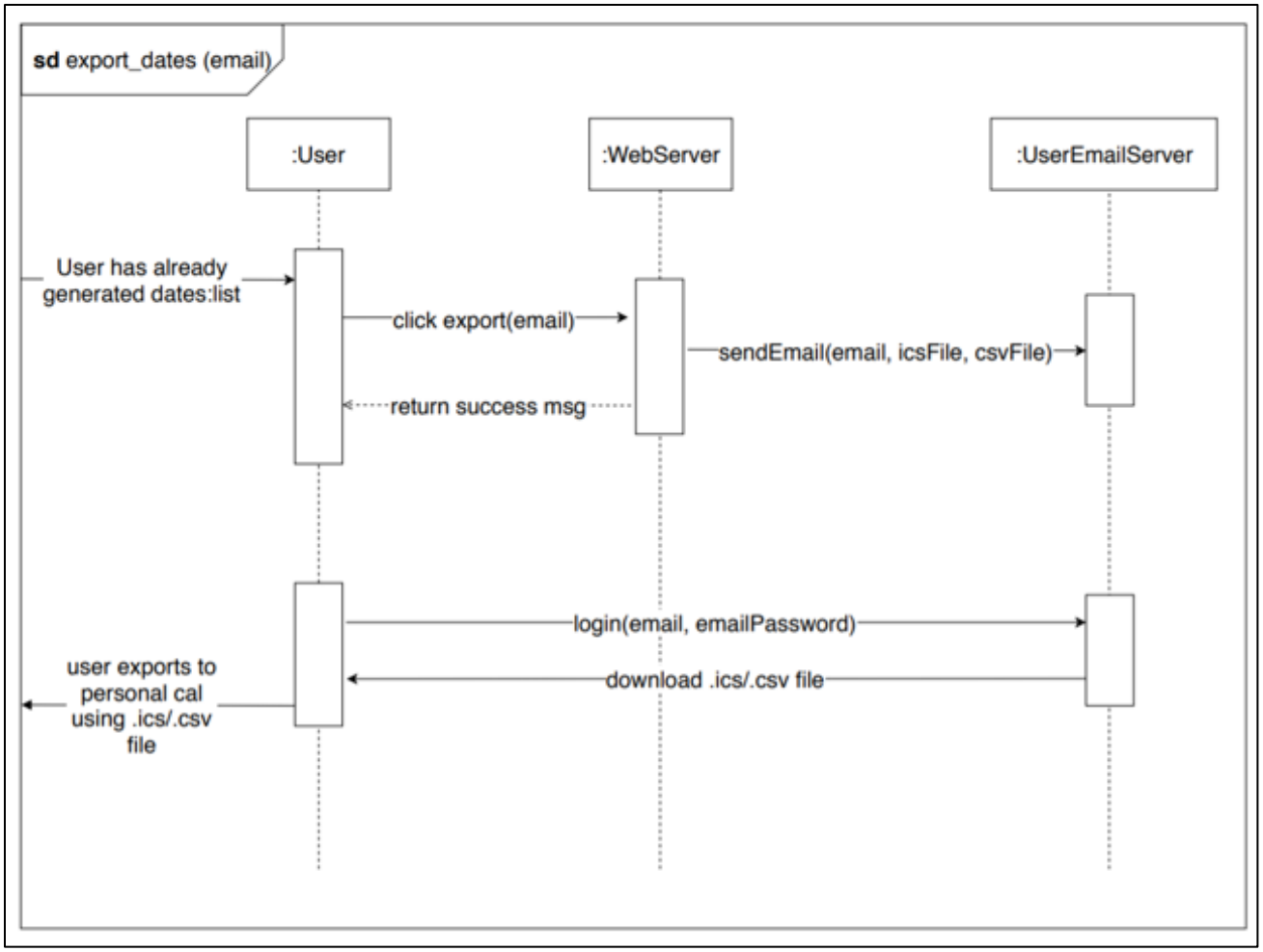


Figure 4.9: Sequence Diagram for Email Service

Priority Level: 3
Feature
Feature: To alternate how I can view the organisation of my course assessments **As A** UNSW WebCMS3 student **I Want To** have the ability to toggle how my course assessments are displayed **So That** I can organise my commitments each week or to each course accordingly
Scenario
Scenario: Raisin has just completed gathering all of the course assessments, and I want to see a summary of all my assessments tasks by course and/or by week **GIVEN** I am on Raisin's Assessment Synopsis/"Due Dates" page **WHEN** I click on a button called "Courses" **THEN** the course assessments should be arranged into separate sections for each individual course, which is organised by each week in these sections **WHEN** I click on a button called "Week" **THEN** all course assessments should be organised together into their respective weeks
Estimated Time Required to Integrate Feature: 1 Day

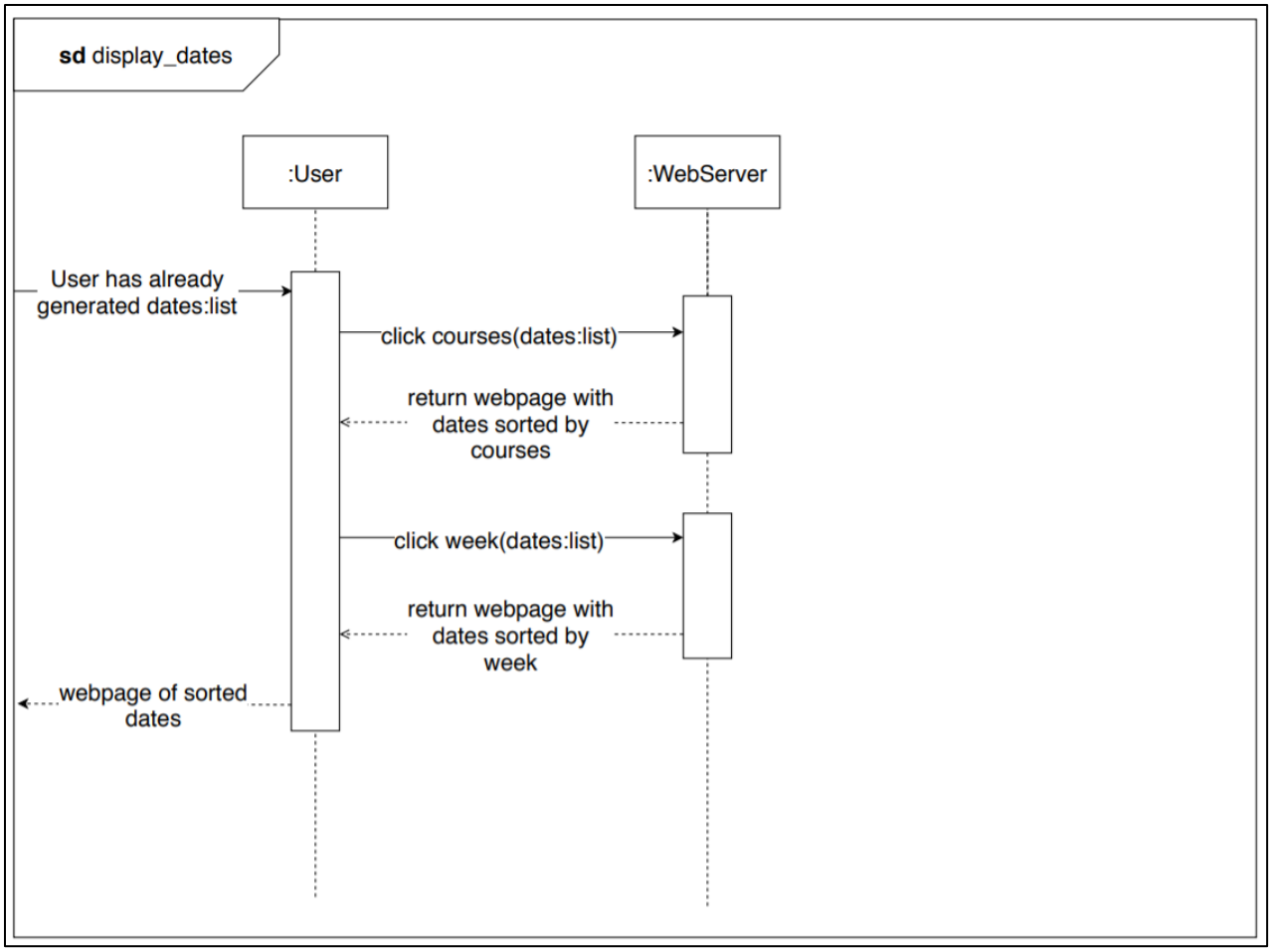


Figure 4.10: Sequence Diagram for Viewing Course Assessments Tasks by Group

5 LO-FIDELITY PROTOTYPE

This area of the report shows our initial design concepts illustrated in both sketches (lo-fidelity prototypes) and actual HTML webpages (high-fidelity prototypes). These are used to visualise how our application may appear so that we can determine how we would implement all the features.

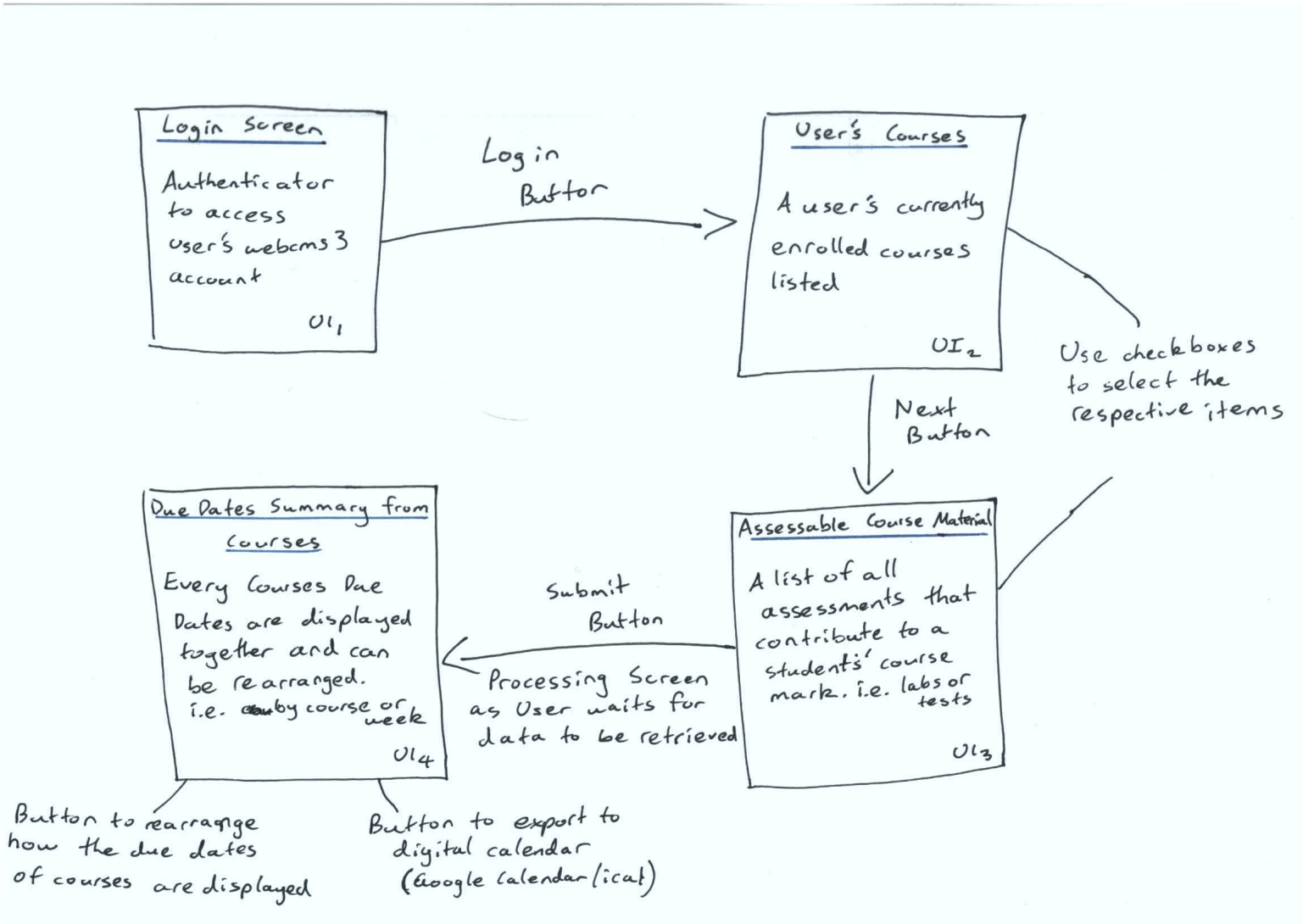


Figure 5.1: Storyboard Interactions Graphs

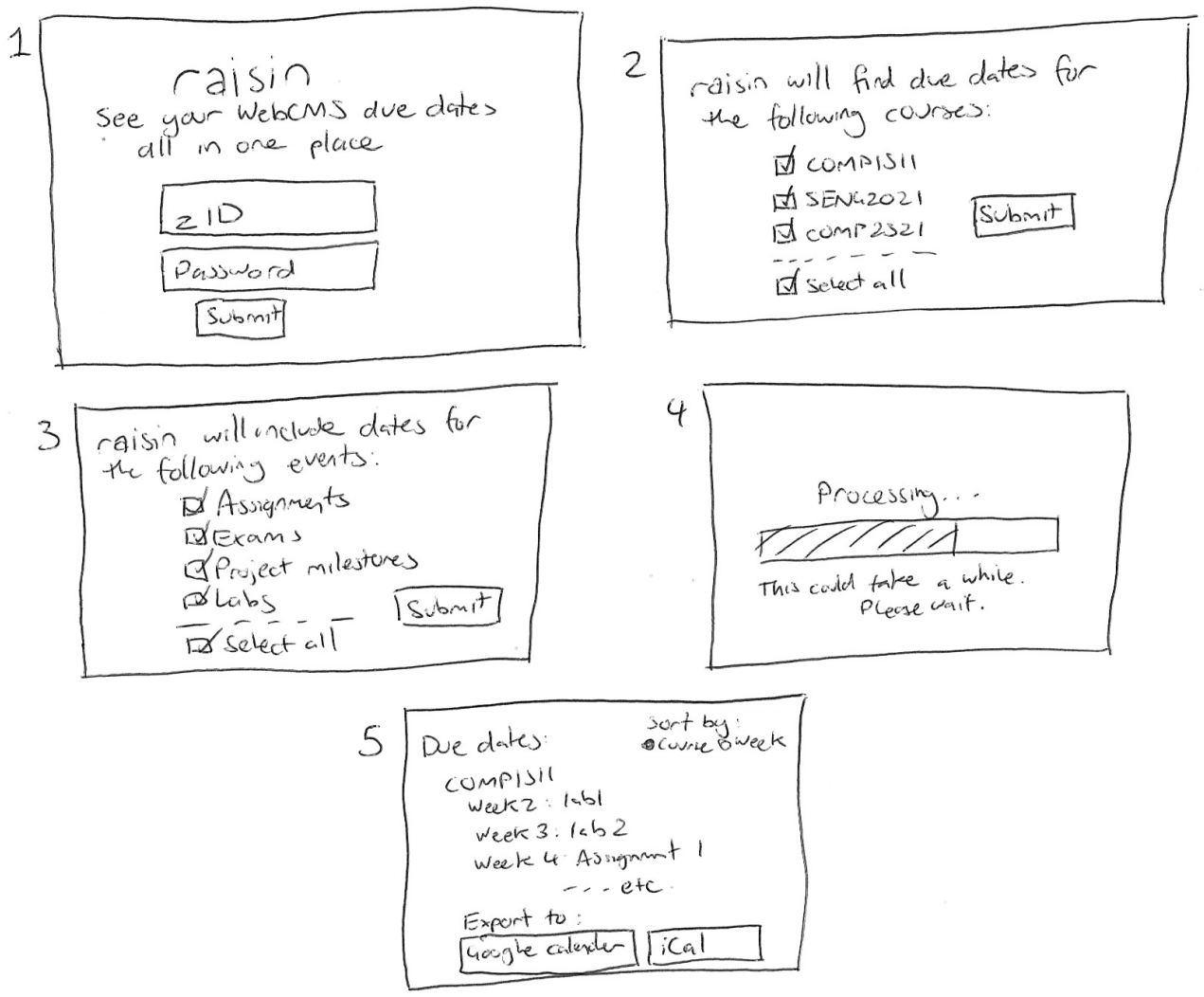


Figure 5.2: Storyboard interactions + UI Component Sketch

6 HIGH-FIDELITY PROTOTYPE

A simple interactive prototype of our website can be viewed through opening the HTML files in any web-browser. Hence the buttons allow navigation to the next webpage. This initial user interface of our website provides an insight as to how a UNSW student using WebCMS3 can operate our application.

These HTML files can be found in the 'master' branch, in the 'html' directory, and the user interface files begins with the 'index.html'. These user interface webpages of our website can also be viewed through the figures below.

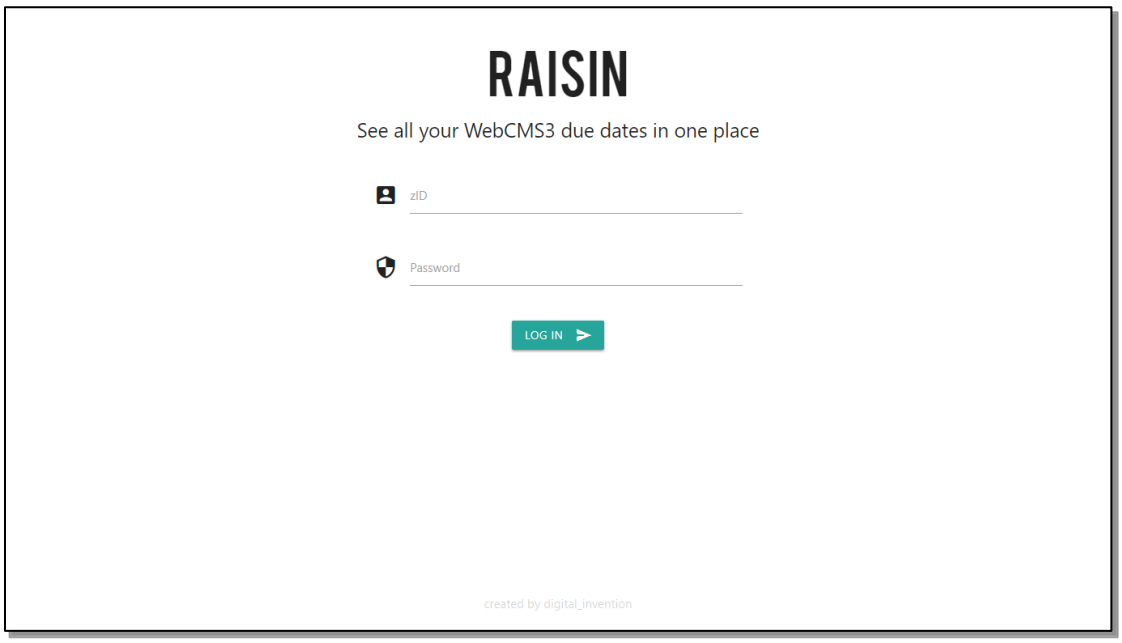


Figure 6.1: Raisin Login Interface

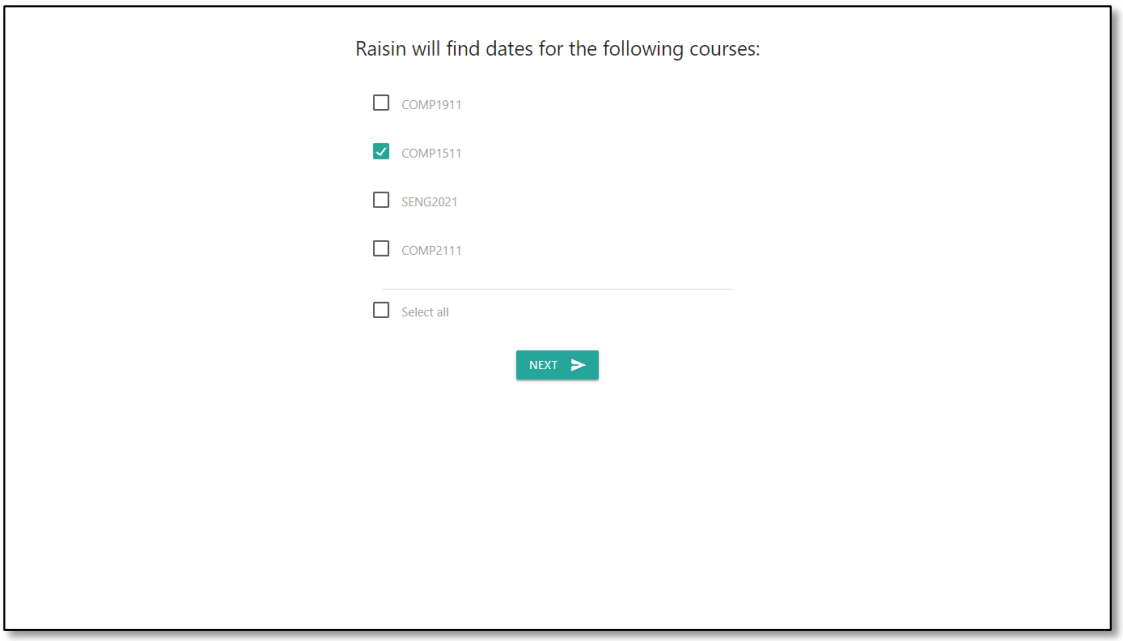


Figure 6.2: Raisin Course Selection Interface

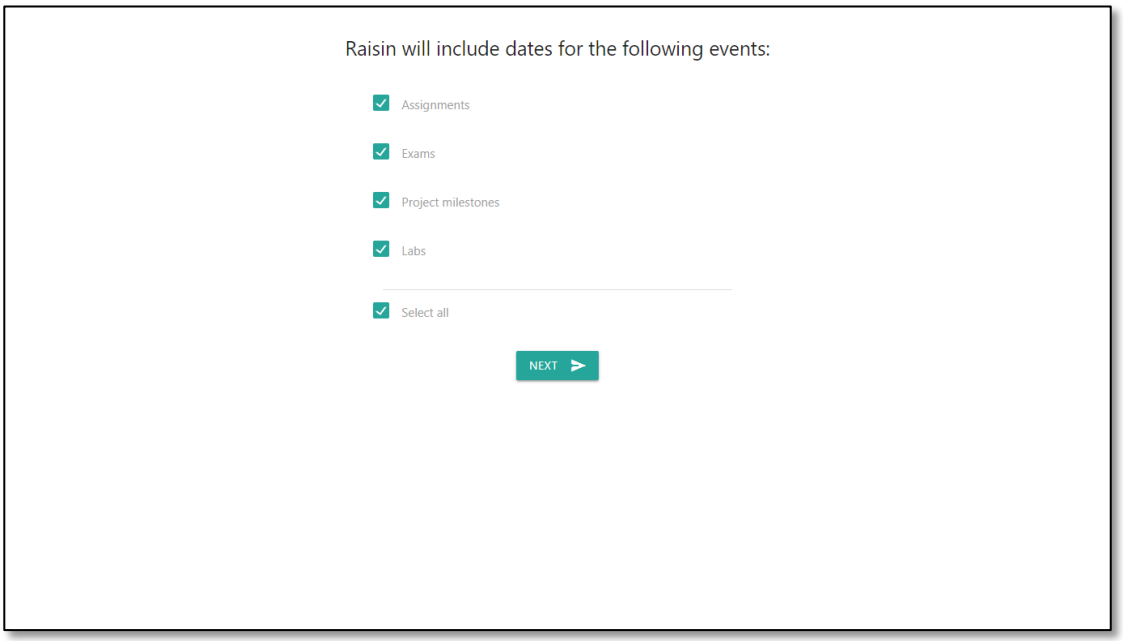


Figure 6.3: Raisin Course Deadlines Interface

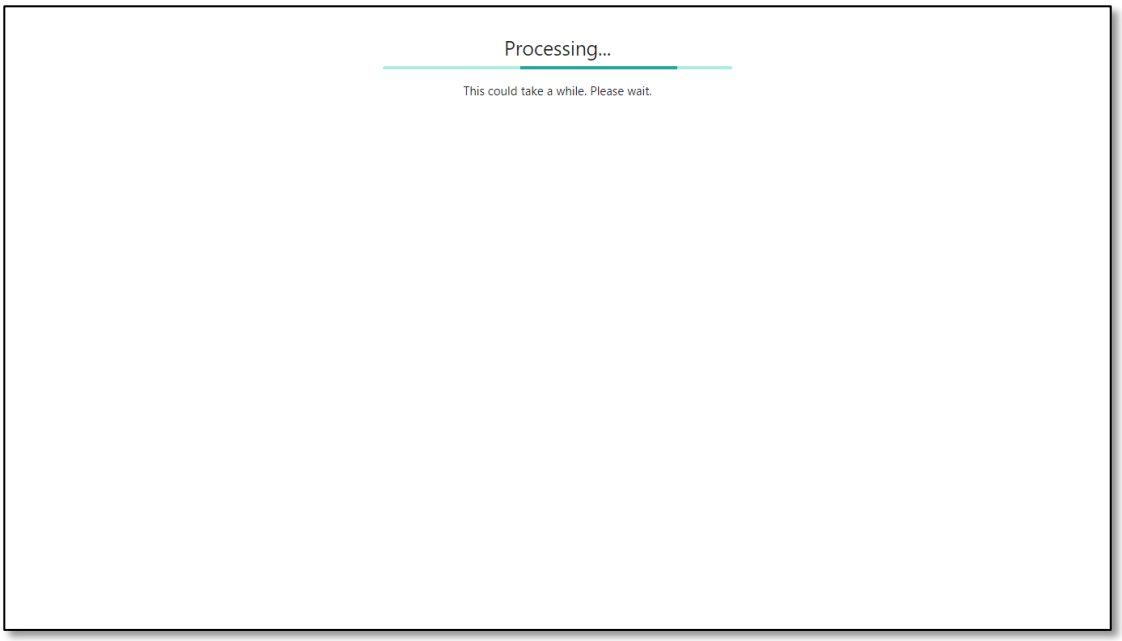


Figure 6.4: Raisin Loading Screen Interface

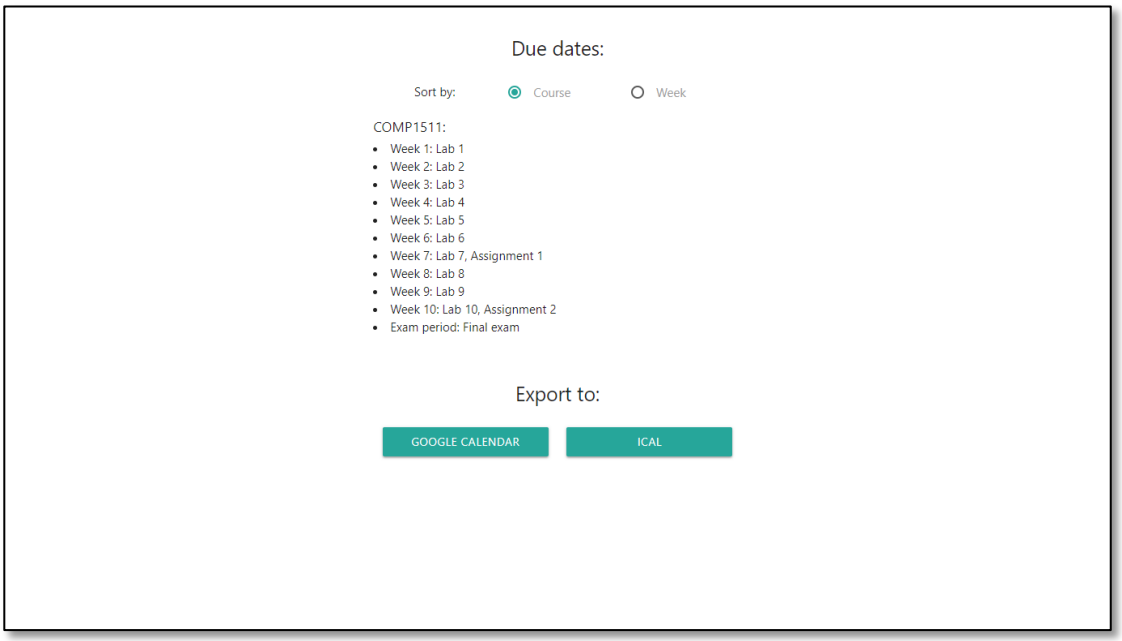


Figure 6.5: Raisin Assessment Synopsis Interface

7 OVERVIEW OF RAISIN'S FEATURES

This segment details a comprehensive summary of all of the completed features in the final iteration of our application Raisin. Thus demonstrating how the user stories have been accomplished by our application

- Login authenticator

Raisin's first webpage requires a user to input their WebCSM3 account details to access our service. This guarantees only UNSW students can use our application and Raisin can automatically access a student's current courses for a quick service

- Automatically knows a user's currently enrolled courses

Our application will instantaneously scrape a user's currently enrolled courses so the user does not have to waste time remembering their course codes and selecting them.

- Instantly collects all course assessable content from every course outline

Raisin will immediately scrape each course outline and find their respective course assessments easing the user from having to navigate to & scroll through each course outline.

- Displays all course assessments from each course on a single webpage

All of a user's course assessments and the week (based on the UNSW academic calendar) they will be due in is presented to the user on an easy to read single webpage. For example, if they are currently enrolled in 3 courses, this single webpage will have each course and their respective assessments with due dates in their individual box.

- Capable of exporting these due dates into Google Calendar

Instead of a user having to now open a new tab and log into their Google account and manually create events for their assessments, they can do this through our application. The complete process only requires a user to log into their Google account, allow permission and their calendar will automatically be updated.

- Provides a downloadable .ics file

For iOS/Apple users, our application will automatically provide an .ics file of the events so they can manually import them into their Apple Calendar (as we cannot access a user's Apple Calendar).

- Provides a downloadable .csv file

Another feature Raisin provides is providing a .csv file which can be imported to any digital calendar service that provides it. This also caters to protective users who may want to use our service, but not log into their Google account through our application.

- An email service is offered, emailing the user an .ics file and a .csv file

Our application can also be accessed by mobile devices, so Raisin offers the user the ability to receive an email of an .ics and .csv file. This only requires a user to enter their email address, and often can serve as a reminder for students to update their digital calendar.

8 RAISIN'S SECURITY PROTOCOLS

The biggest concern a UNSW student would have when using a 3rd party web application would be revealing their account credentials and personal information. Thus we have incorporated security measures and various features as alternatives for those users. However to use and access our application at all, a student must enter their WebCMS3 login details.

- Our website (http://raisinplanner.com/) has a Transport Layer Security certificate which officially informs the consumer that their personal data is protected. This is because a TLS ensures the connection between our web server and a browser is encrypted, which means a user's webCMS3 login details are protected.

- A feature our application provides is emailing a .csv and .ics file to a designated email address. This is to alleviate concerns of logging into their Google account through our application, so instead they can manually import it themselves on Google personal webpage.

- Each time a user wishes to export their assessments to their Google calendar, they must grant read/write permission every time so that for systemraisin@gmail.com to edit their Google Calendar. This is because the user's credentials are not saved, so systemraisin@gmail.com access to is limited as it does not always have read/write permissions of a user's Google calendar.

- Internally our system does not have a permanent database, which means a student's login credentials and a student's currently enrolled courses are unknown each time our application is used. When handling a user's login details, our server just transfers the sensitive data across to WebCMS3. This means only verified UNSW students can use our application as the authentication process is done by the university's security system. Additionally when a user is created, they are assigned a specific number and only exist as long as the user is logged in. This ensures a user's courses are not linked to a specific UNSW zid. Thus at all times our system only has access to the least amount of information on a user.

9 SOFTWARE ARCHITECTURE: INTRODUCTION

Since our application will be website based, it will require a graphical user interface (front-end), and a data processing server (back-end). The necessary interactions between our application's front-end and back-end can be seen in Figure 9.1. Thus, the application will use a client-server architecture, which will allocate the workload accordingly through a centralised server as clients request services. The entire software architecture of our application can be seen in Figure 9.2.

Primarily our application is targeted towards students wanting to summarise their WebCMS3 course due dates ahead of time. Although many unique users would be served, the application provides a single-use service for each user: requiring no persistence or permanent storage of data. Thus data will be temporarily stored during a session, and once finished, it will be handed over to a respective calendar API and then deleted from our system.

Since our application is being hosted on actual domain, anyone can access the website at https://raisinplanner.com/. However only UNSW students with a WebCMS3 account can naturally only login and access Raisin's features.

Alternatively Raisin can be run on a local computer address any hardware that can use a Linux machine. The process to do this is explained below

1. Create a new virtual environment, and start it up
2. Install all modules from requirements.txt (pip install -r requirements.txt)
3. Install Java (https://www.java.com/en/download/) - needed for the PDF scraper
4. You may also have to add Java to your PATH if there's a Java error (instructions - https://community.akamai.com/customers/s/article/Adding-JDK-Path-in-Mac-OS-X-Linux-or-Windows)
5. Run the server (python run.py) - this will work on localhost, and will also broadcast externally from ports 80 and 443 if you have them open

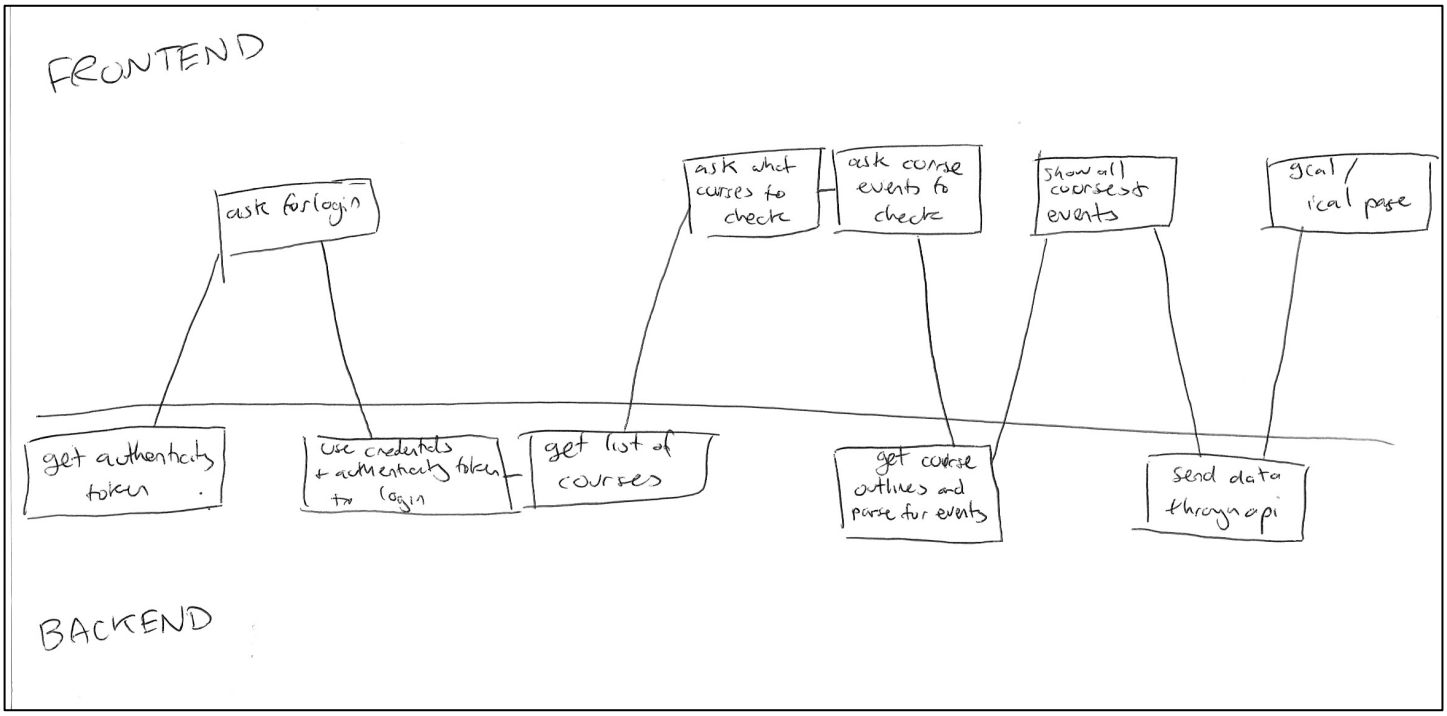


Figure 9.1: The front end & back end process of using the web application

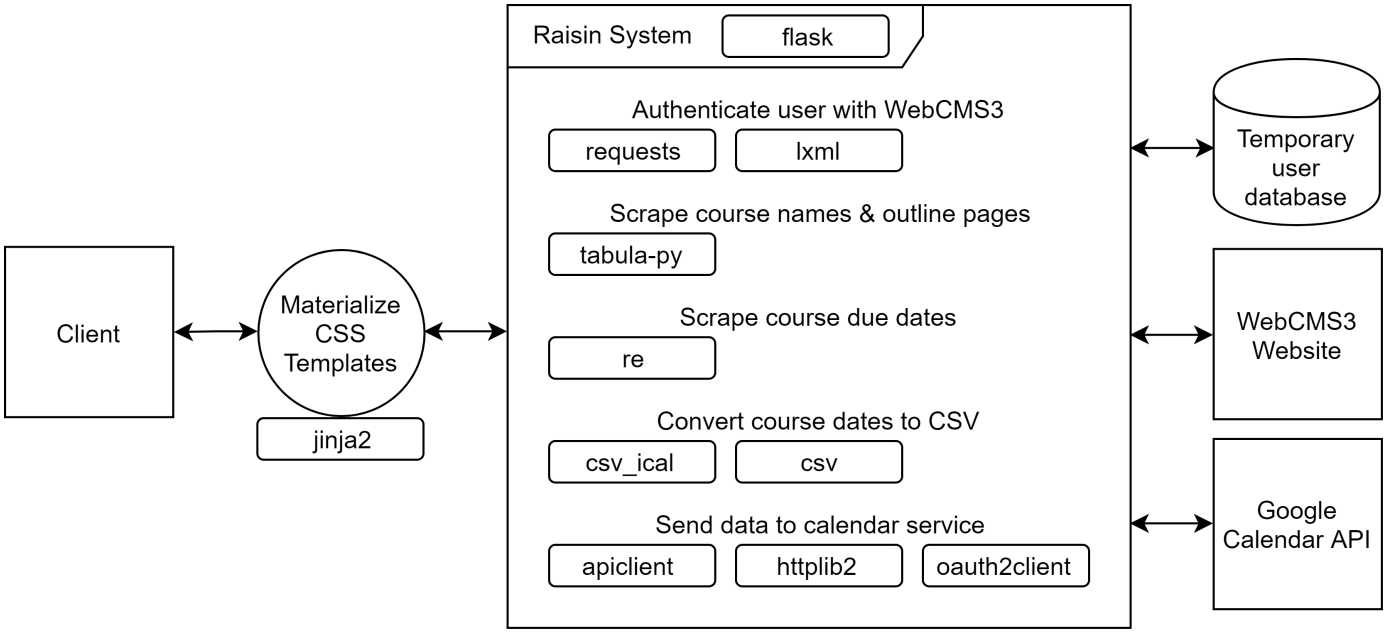


Figure 9.2: System Architecture Diagram of Raisin.

Legend:

Straight Edges = External Application/Entities

Rounded Edges = Python Modules

Circle = Web Interfaces/Templates

Cylinder = Database

10 SOFTWARE ARCHITECTURE: BACK-END

The server for our application will be written entirely in Python as its countless powerful packages and frameworks are suited to efficiently create our simple website model. Python also suits our group as we all are experienced with the language and its simplicity and easy readability facilitates a cohesive group working environment. Additionally, the classes that we will use to run our server will incorporate the structural software design pattern "*façade*", so our server maintains high cohesion and low coupling. **To add in more code principles we follow** These principles will allow the server to simplify its subsystem and make it less dependent upon them. Specifically, for our server, it will streamline the process of using the objects that represent an assessment and simplifies the other features that use the object. Consequently, Python will be used to construct a back end for our application so the tools and data sources can be used effectively.

The main data sources our project will use are WebCMS3 and Google Calendar's API:

- The application uses "web scraping" to obtain a student's current courses, as well as their respective course outline documents, which are found on WebCMS3. Accessing the course outlines requires a student's zID and password, which are sent directly to WebCMS3 and are not withheld by our application.
- In order to export a course's due dates into a calendar form, the application will require access to Google Calendar's API, where a student can log into their Google account and through the use of the Python "oauth2client" library, the calendar will automatically request & update.
- Alternatively, if a student would prefer to use Apple's calendar, our application will then convert the current data into an ".csv" file then into an ".ics" file, which can then be manually added to a calendar.

However, the process of extracting due dates from a course outline document is no trivial task and is essentially the core functionality of our application. This will require our group to develop a specialised web scraper & PDF scraper in order to extract the assessments and their due dates from a course's outline. The web scraper will use the Python "lxml" library to extract data from webpages while the PDF scraper will use the Python "tabula-py" module. Overall the intended process to achieve this is, follows:

(Bolded portions represent front-end interactions on Raisin's web browser)

1. Server scrapes the CSRF authenticity token from the WebCMS3 login form (this is necessary for login)
2. **User manually inputs their zID and password**
3. User's login details are passed to WebCMS3 for authentication, along with the CSRF authenticity token from before
4. WebCMS3's top navigation bar (which contains currently enrolled courses) is scraped for course names and URLs
5. **User is asked what courses and event types (assignments, labs, project milestones and exams) they want**
6. Course outline documents are scraped for each course selected
   - If the course outline is a PDF, the PDF scraper is initiated, extracting all tables from the PDF
7. Points of interest are extracted from the course outlines, namely table rows and bullet points. These are each searched for instances of "Assignment X" or "X due" or "X exam", etc.
8. If any of these terms are found, their immediate surroundings are searched for text such as "Week X" or "Exam Period", as well as a "% weighting". If found, these values are added to a dictionary
9. **Due dates for each course are formatted and presented to the user**

Raisin's back-end is also responsible for handling all of the export features our application provides. The process that each of these features goes through internally on our server are chronologically ordered below. All of the interactions involving Raisin's website use the Python library "Flask".

Export to Google Calendar

1. Once the course assessments have been created as objects within our server, a server connection to Google is arranged
2. Raisin's email systemraisin@gmail.com is then connected to the server
3. A new tab opens, representing the connection to this server, where the user must accept to allow permission for systemraisin@gmail.com to have read/write privileges of their Google Calendar
4. A unique code of current user's Google account is generated (this code represents their specific account)
5. The user inputs this code on Raisin's application which then finally grants systemraisin@gmail.com read/write access of their Google Calendar
6. A processing message "Please wait…" appear on Raisin's interface as systemraisin@gmail.com updates the user's calendar with the events/course assessments. (The course assessment objects are passed through to a specific library function which automatically will format it and create the events)
7. Then a success message will appear notifying the user "Added X events to your Google Calendar!"

The Google API primarily supports the "oauth2client" library in Python and the "apiclient" library, thus this entire feature revolves around using functions from these libraries.

Export to Apple Calendar (providing a downloadable .ics file)

1. As the course assessments are being scraped from the course outline, they are being made into objects in Python. Specifically a “Course” object is created which has due dates list. This list will then contain the objects “Deadline” which represent a specific course assessment and its weight/%, description, deadline and a summary of it.
2. Then the deadline objects are made into a specific event object from the ics library
3. These events are then added onto a calendar, then this calendar is written to an .ics file
4. The flask function “send_file” then takes the path of the .ics file and sends it to the downloads section of the user’s web browser

Since creating an .ics file has a specific process, functions from the Python library “ics” are used in order to create the events and calendar.

Export as .csv file

5. As the course assessments are being scraped from the course outline, they are being made into objects in Python. Specifically a “Course” object is created which has due dates list. This list will then contain the objects “Deadline” which represent a specific course assessment and its weight/%, description, deadline and a summary of it.
6. These deadline objects are then created as events on a calendar.
7. This calendar is then written to a .csv file as strings.
8. The flask function “send_file” then takes the path of the .csv file and sends it to the downloads section of the user’s web browser

This feature uses the Python library “csv” to do all of these operations.

Email service (Emailing the user with an .ics file and a .csv file)

1. Once the .ics and .csv files are created, a server connection to Gmail is requested, using SSL for security reasons.
2. Then an email is generated with a preloaded message, attaching the files.
3. This email is then specifically encoded so that it can be sent using this SSL connection to the Gmail server (The server connection only accepts bitrate strings, so the email itself is passed through as a string)
4. The server connection is closed and an confirmation message appears on the Raisin interface if no errors occur

The email service uses the Python library “smtplib” to be able to initialise and create a server connection then send an email while the Python libraries “email.mime” handle encoding the email.

11 SOFTWARE ARCHITECTURE: FRONT-END

The graphical user interface will use Flask, Jinja2 and Materialize in order to provide a simplistic and easy-to-use experience. This type of user experience is desired so that any UNSW student who uses our website will instantly understand how to use our application and immediately receive service.

Flask

Flask is a powerful, yet relatively straightforward web server, with enough features to cover all the needs of our application. Due to the limited timeframe, this was chosen as all members of our group are experienced with its use.

Jinja2

Jinja2 allows for seamless integration between a client and server in the form of modular HTML templates. These allow for data to be presented effectively and efficiently, in order to serve multiple clients at once. Again, this has also been chosen due to our group's prior experience.

Materialize

These templates are styled using Materialize CSS, allowing for simplicity and ease of use, all while maintaining a modern aesthetic. While no group members have had any prior experience with it, the documentation is simple enough to understand, making it a suitable choice for a comfortable user interface.

Through the use of the above libraries, our web application should achieve a high-quality user experience.

12 Demonstration of Final Product

This section provides screenshots of all of the features and functions of our final product. The first group of screenshots below represent the core functionality of our product.

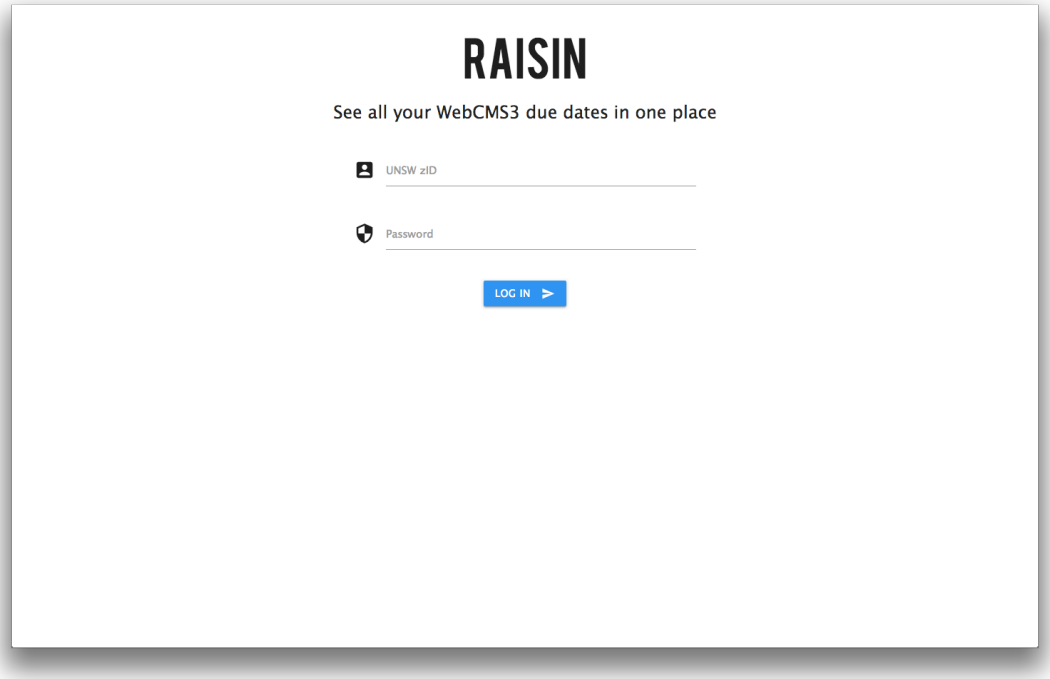


Figure 12.1: Raisin's user login interface

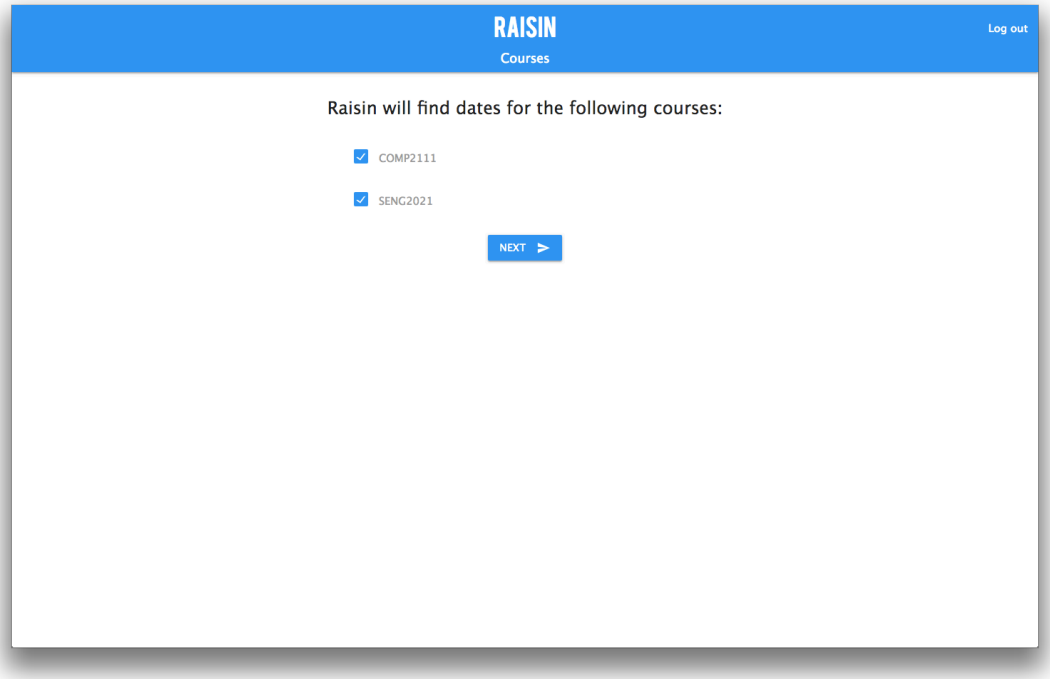


Figure 12.2: Raisin's Course Selection page

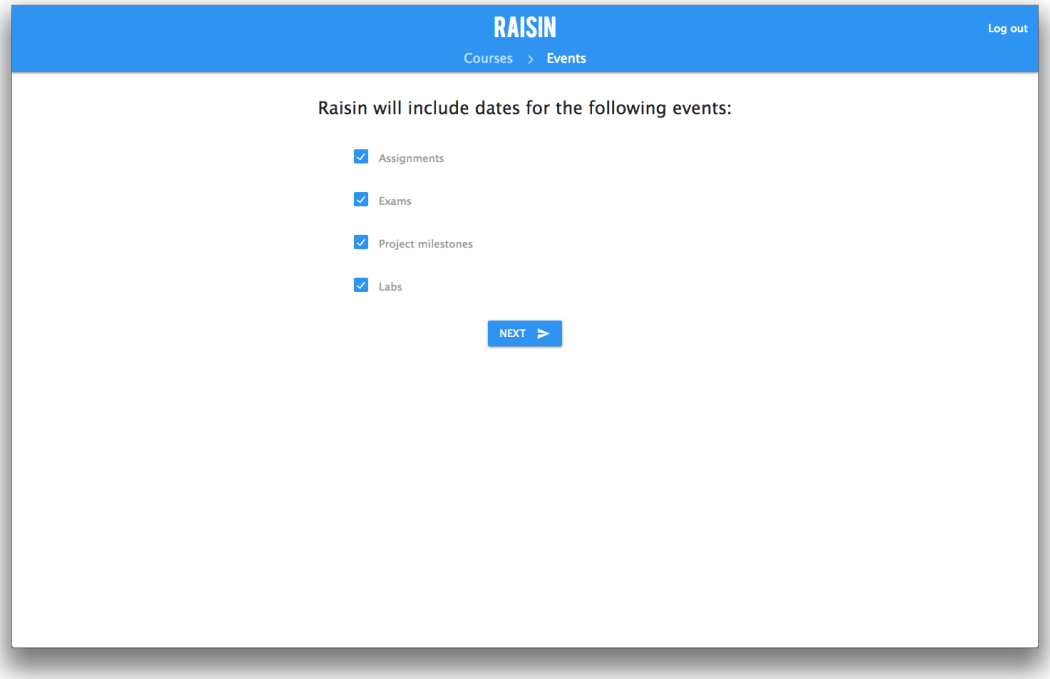


Figure 12.3: Raisin's Assessment Selection page

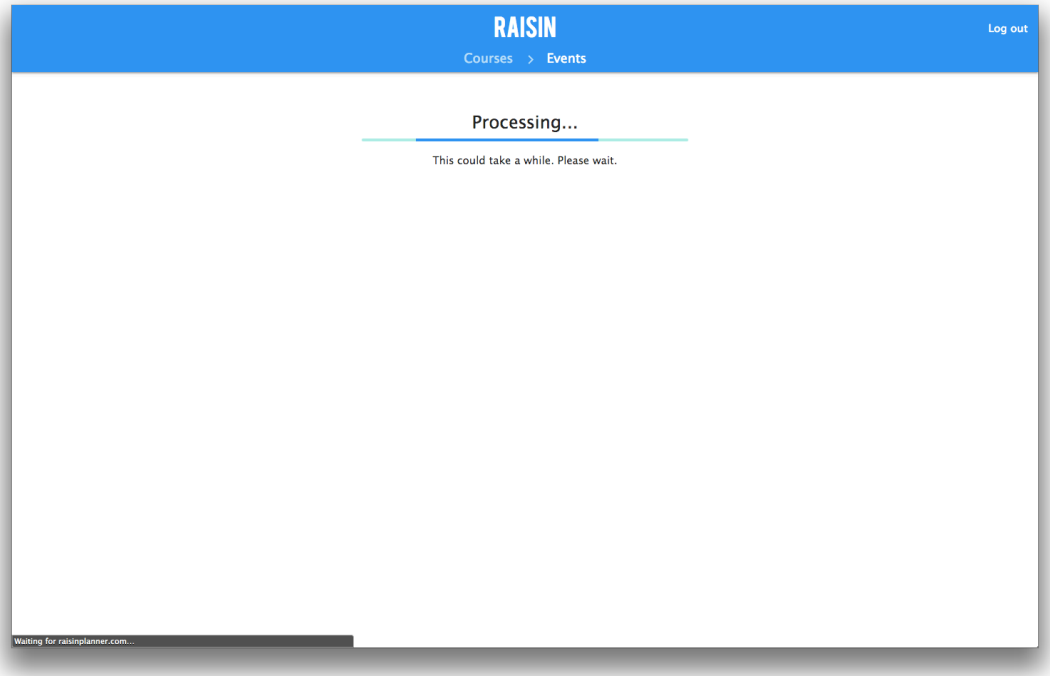


Figure 12.4: Loading screen as user waits for courses to be scraped

Figure 12.5: Raisin's Assessment Synopsis/"Due Dates" page

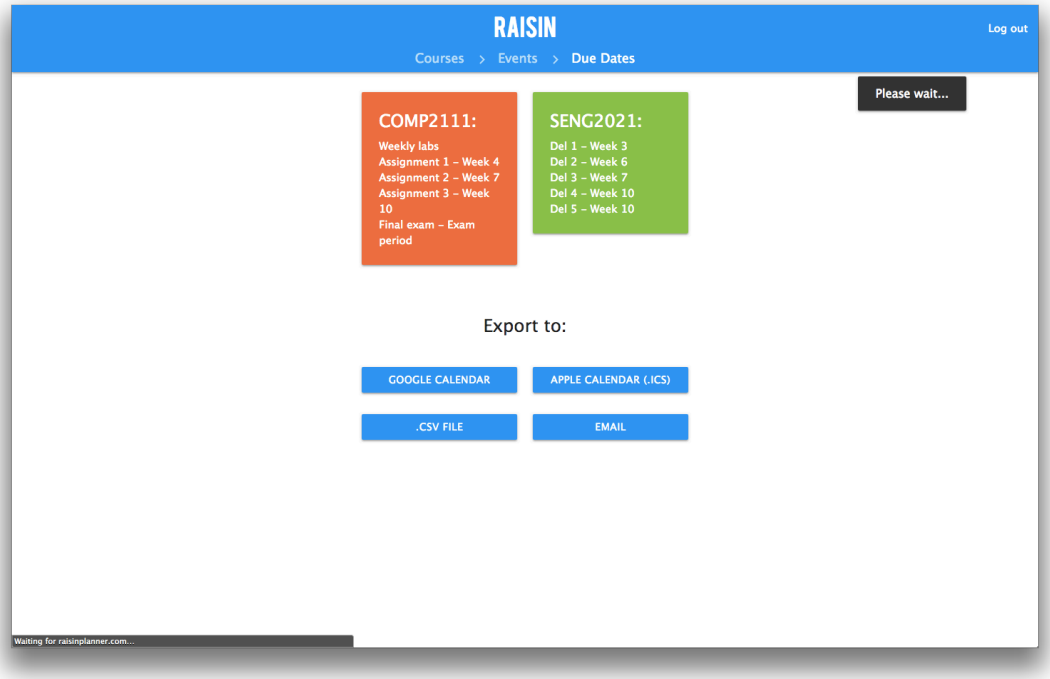
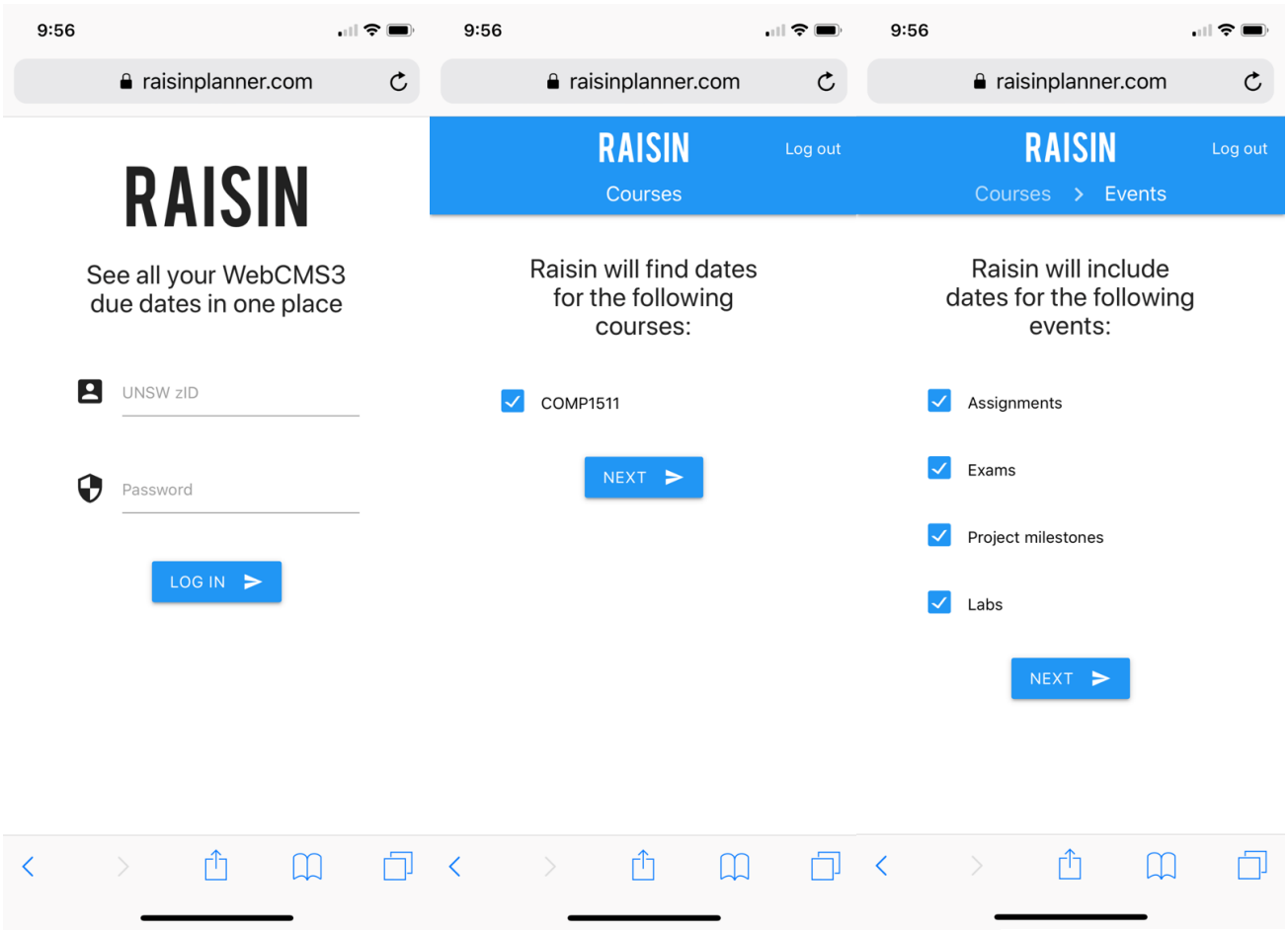
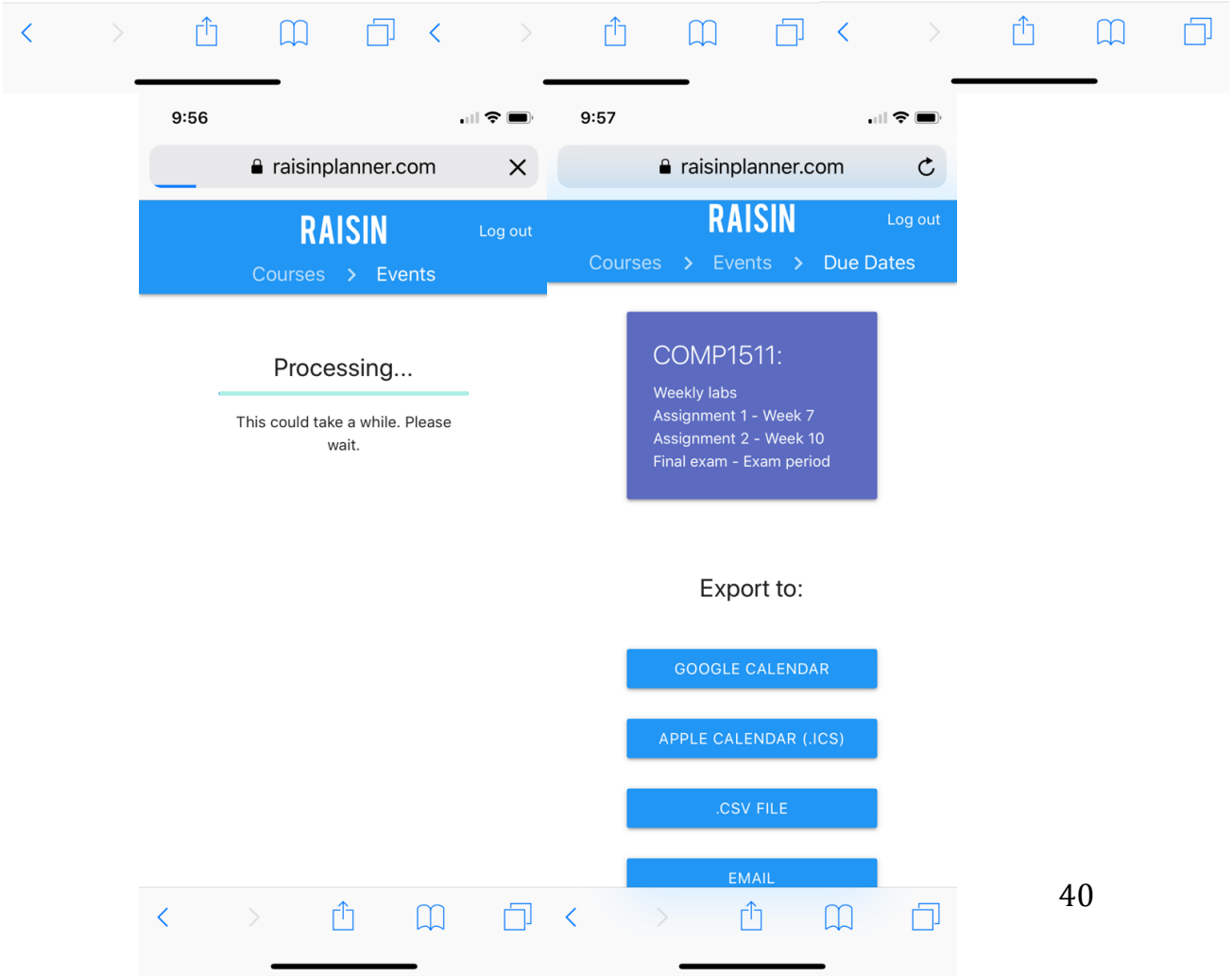


Figure 12.6: Pop up indicating a function is in the process of being completed

Since the website can be accessed on any device at http://raisinplanner.com/ these screenshots highlight what Raisin would look like on a mobile device.





The next group of screenshots outline the process of a user exporting their assessments to their Google Calendar

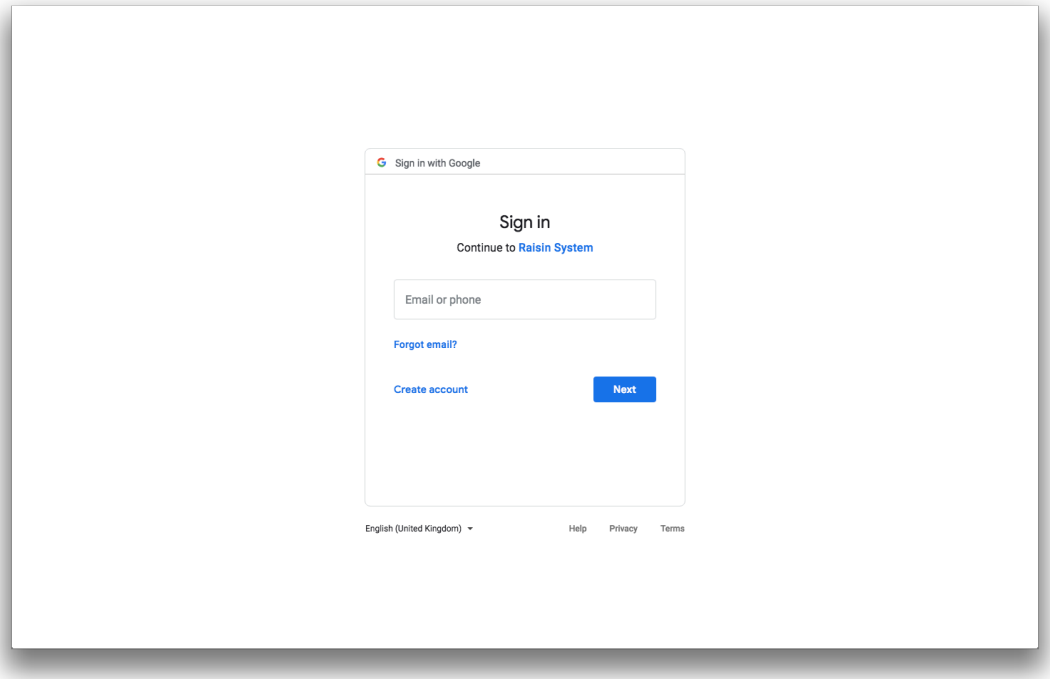


Figure 12.7: Clicking on ‘Google Calendar’ redirects you to a Google login page

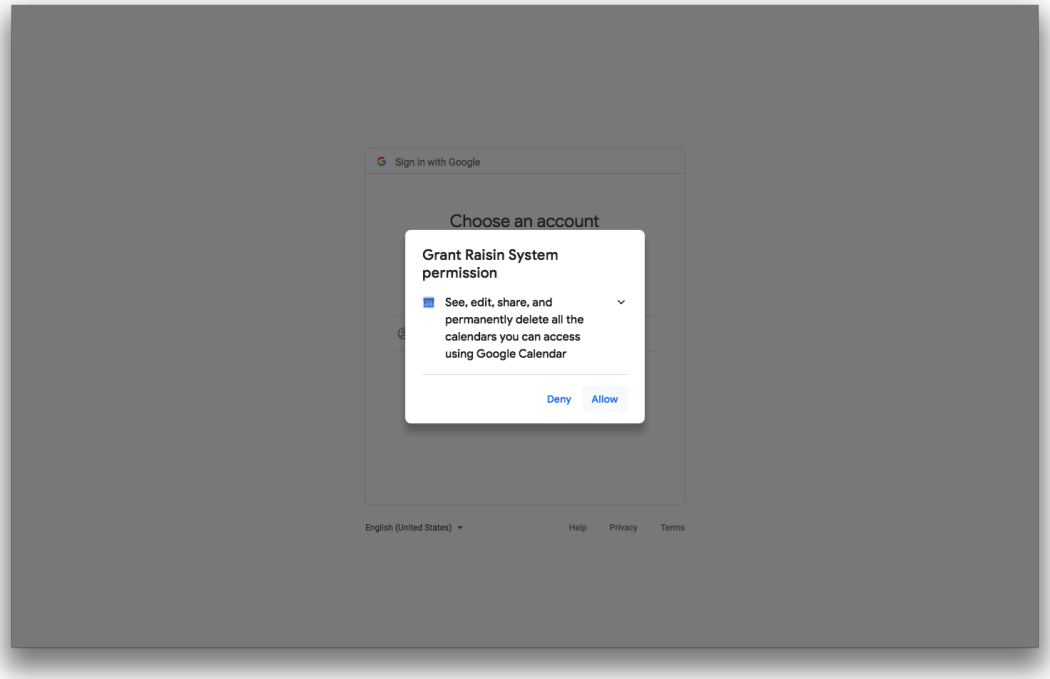


Figure 12.8: Google services then require the user to allow permissions so Raisin can export the events into the user’s Google Calendar

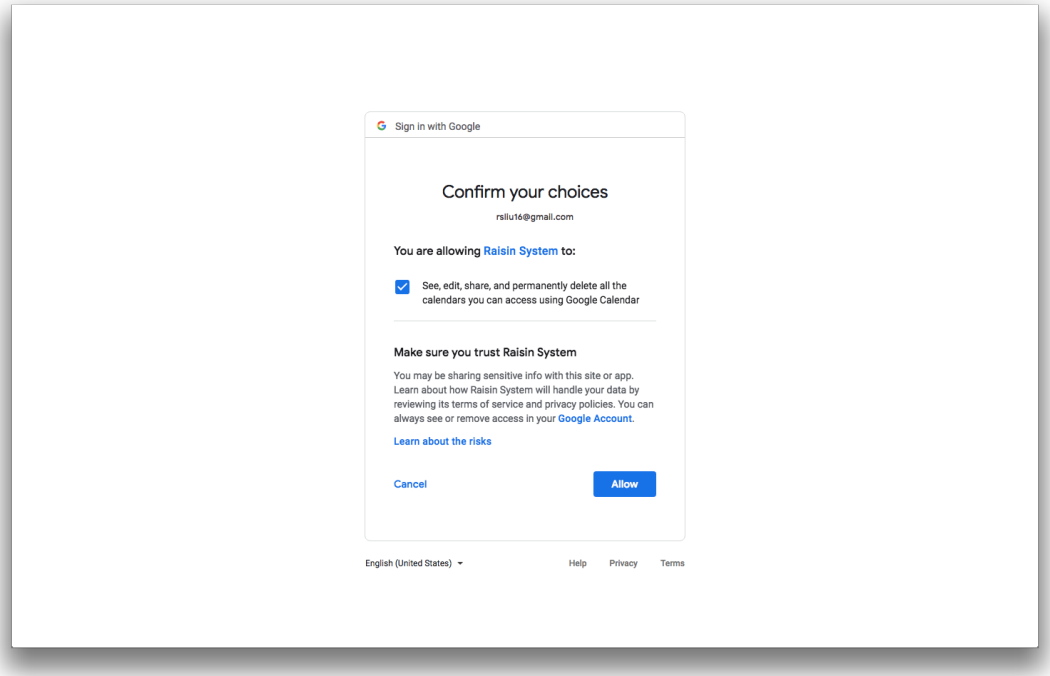


Figure 12.9: ????

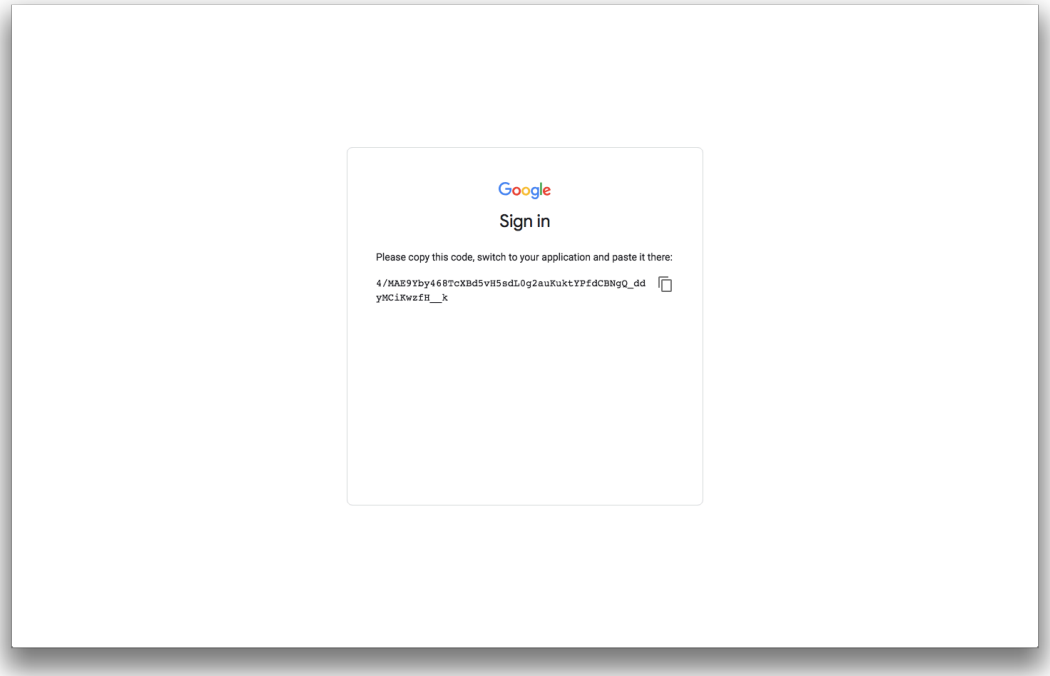


Figure 12.10: Google provides the user with a verification code for systemraisin@gmail.com to edit the user's Google calendar

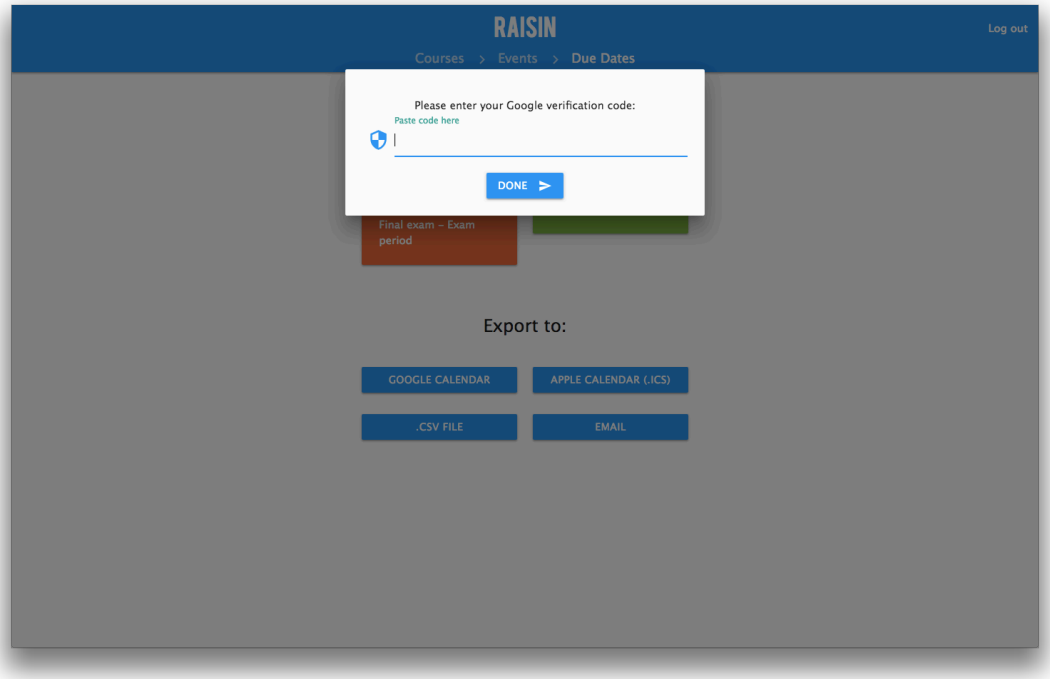


Figure 12.11: Raisin prompts the user to enter the verification code

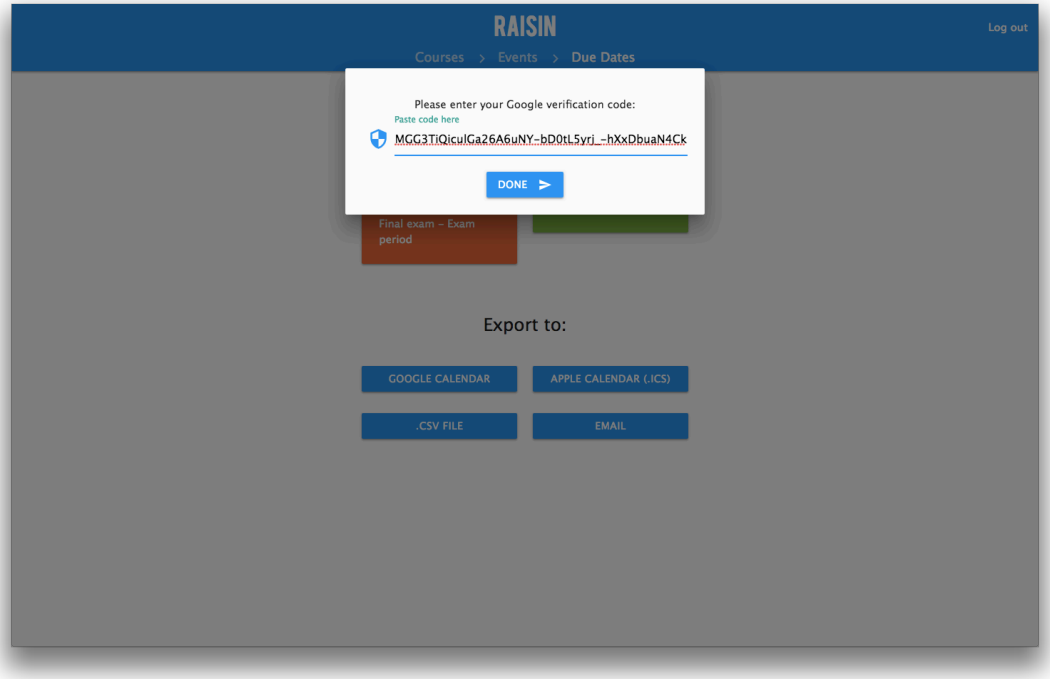


Figure 12.12: User enters the verification code on Raisin, then prompting systemraisin@gmail.com to edit the user's Google calendar

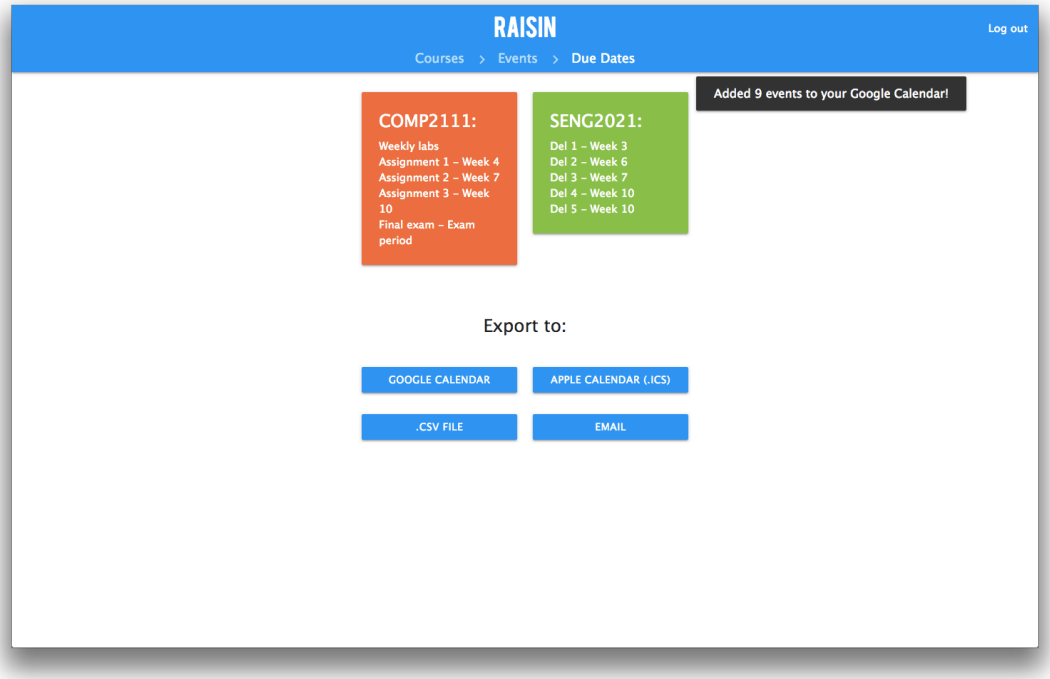


Figure 12.13: Pop up appears confirming the events have been successfully added to the users' calendar

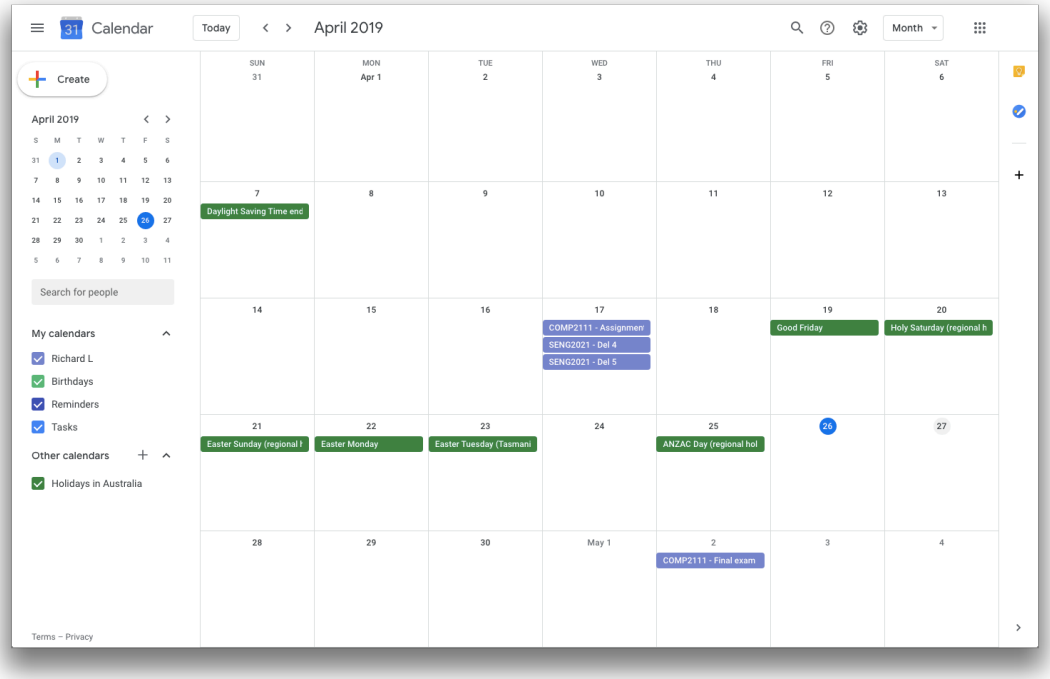


Figure 12.14: This is an example of what the user's Google Calendar would look after using Raisin

These screenshots represent the process of a user performing the export “Apple Calendar (.ics)” feature.

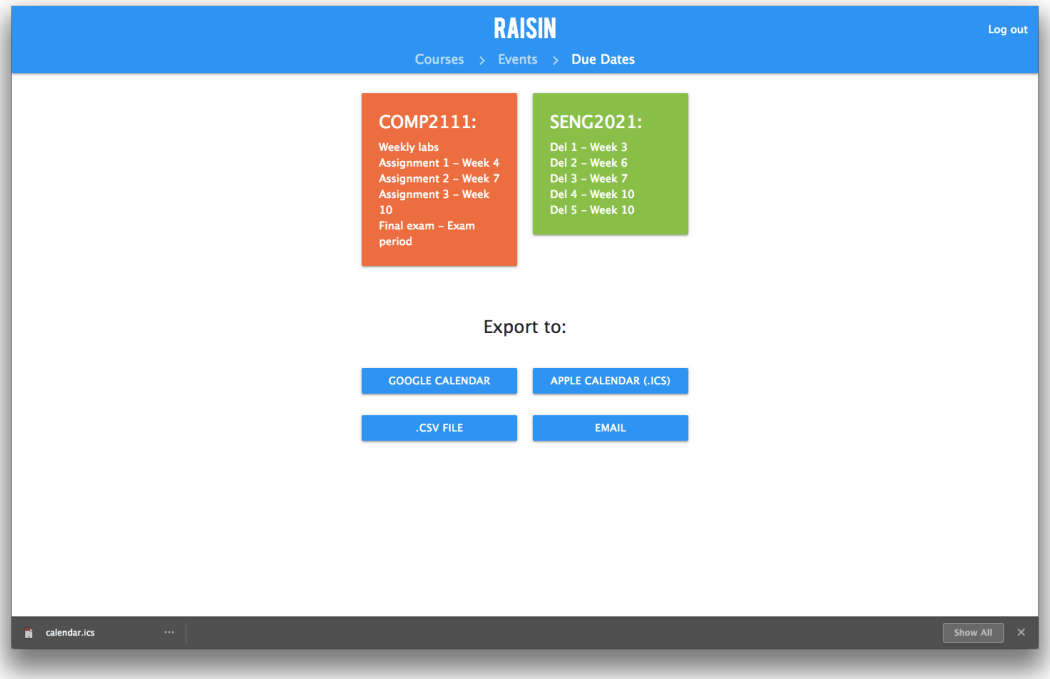


Figure 12.15: The user clicks on ‘Apple Calendar’ and is provided an .ics file in the web browser’s downloads folder (as seen in bottom left corner)

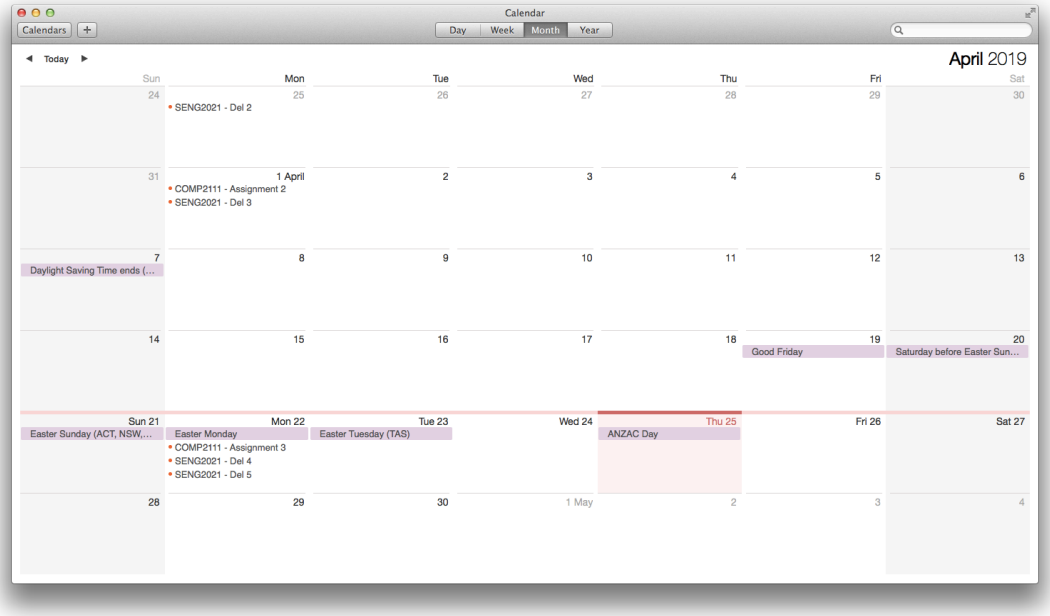


Figure 12.16: The .ics file is then imported into the user’s Apple Calendar, with events displayed as such

Next this screenshot represent how the export ".CSV file" feature functions. This could be used to import a calendar into Google Calendar or some other digital calendar.

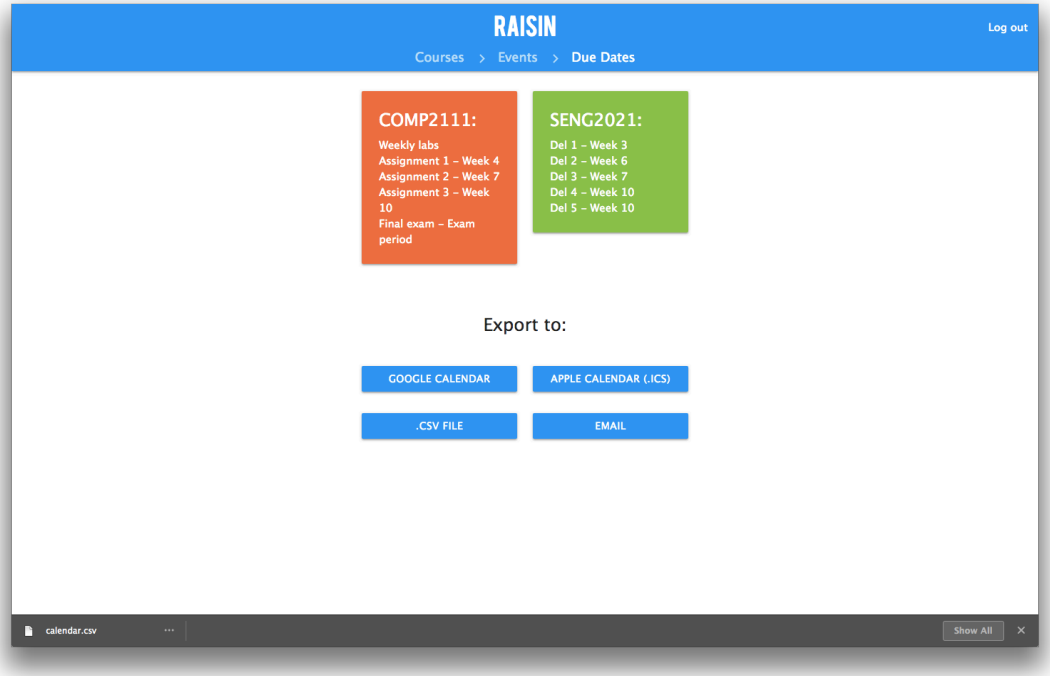


Figure 12.17: The user clicks '.CSV File' and is provided an .csv file in the web browser's downloads folder (as seen in bottom left corner)

Finally these screenshots demonstrate how Raisin's email service would operate.

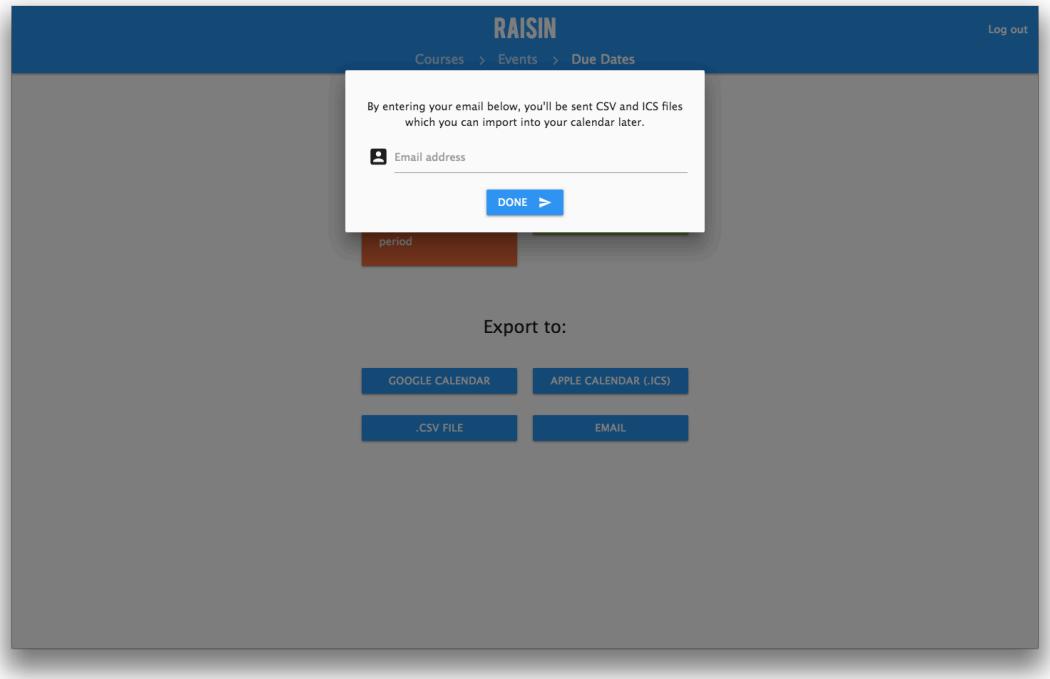


Figure 12.18: The user clicks 'Email' and the user is asked to enter an email address

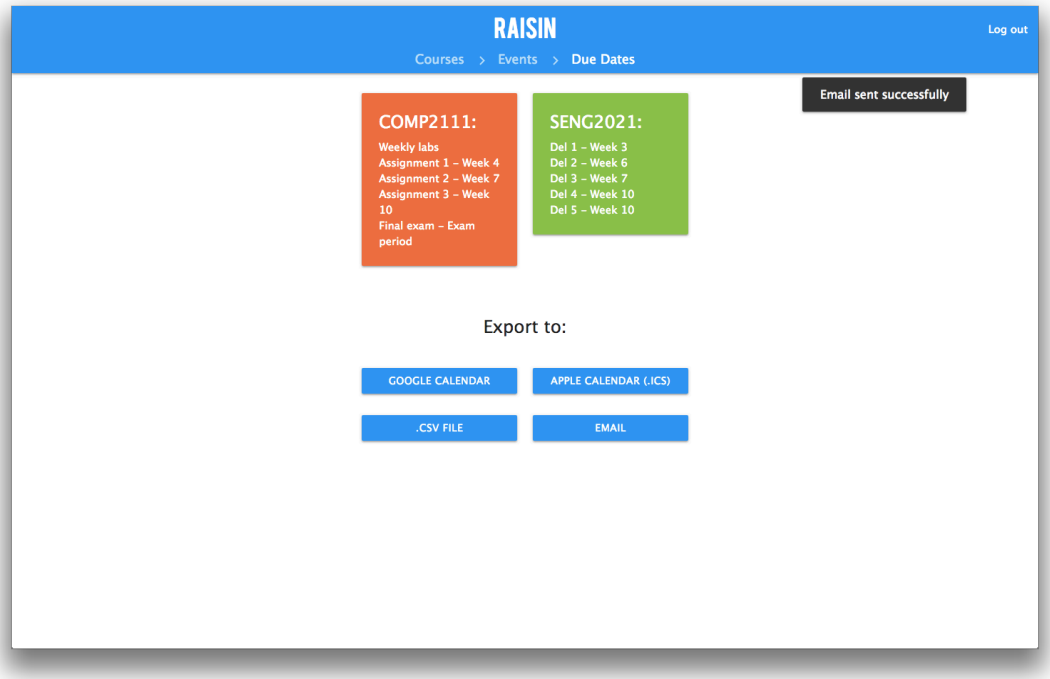


Figure 12.19: Pop up appears confirming the email has successfully been sent

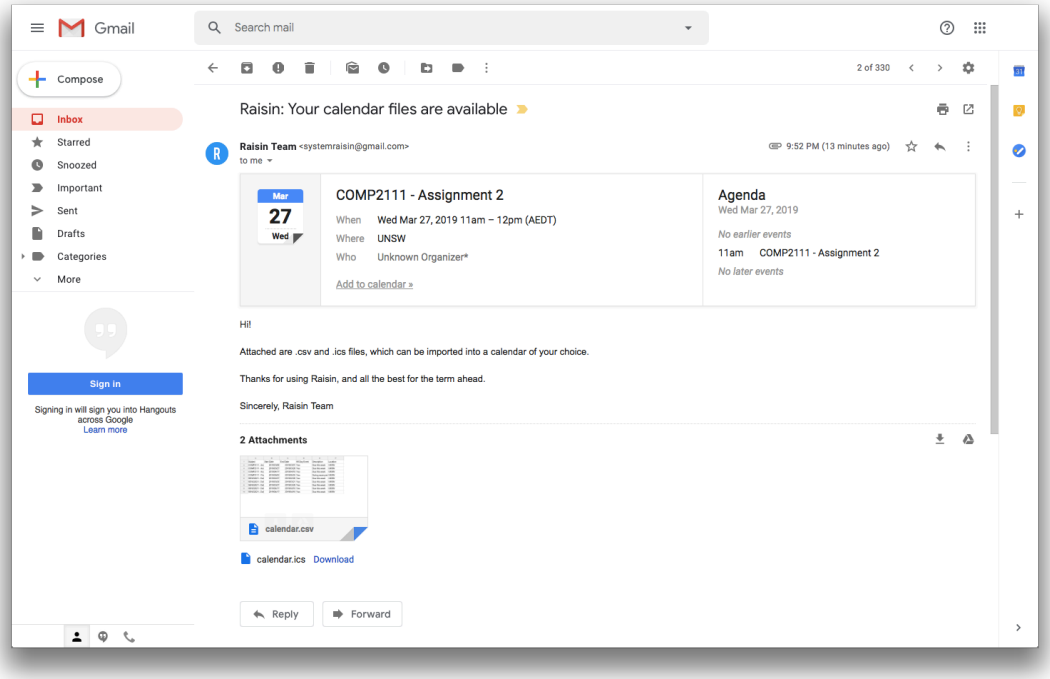


Figure 12.20: The email that systemraisin@gmail.com sends the user

13 RAISIN'S FUTURE POSSIBILITIES

The general problem that Raisin aims to solve can be applied to any university or college and if further improved upon, Raisin could potentially be capable of serving all students in Australia or across the world. Currently Raisin's initial system and user interface has the opportunity for scalability since most of our code has been refactored. Therefore since the general process of creating calendars with events will stay consistent the main improvements that can be applied to Raisin, are improving the process of web scraping course outlines, the parser to filter course assessments from the outline and a new web crawler to be compatible with any online course outline. The branch "feature1" has an alpha version of a potential future iteration of Raisin. Explained below is a how this alpha version can be built upon to first more efficiently handle collecting course assessments for WebCMS3 students at UNSW.

The new system will take in a University URL and start crawling the website creating a graph as seen in Figure 13.1, as the parser filters it and scraper will analyze the HTML saving a tree alongside the URL. Figure 13.2 illustrates this process. This tree can then be further processed using key words such as calendar to get the current Term. Within this tree, a "datetime" object will also be saved to detect changes in the HTML.

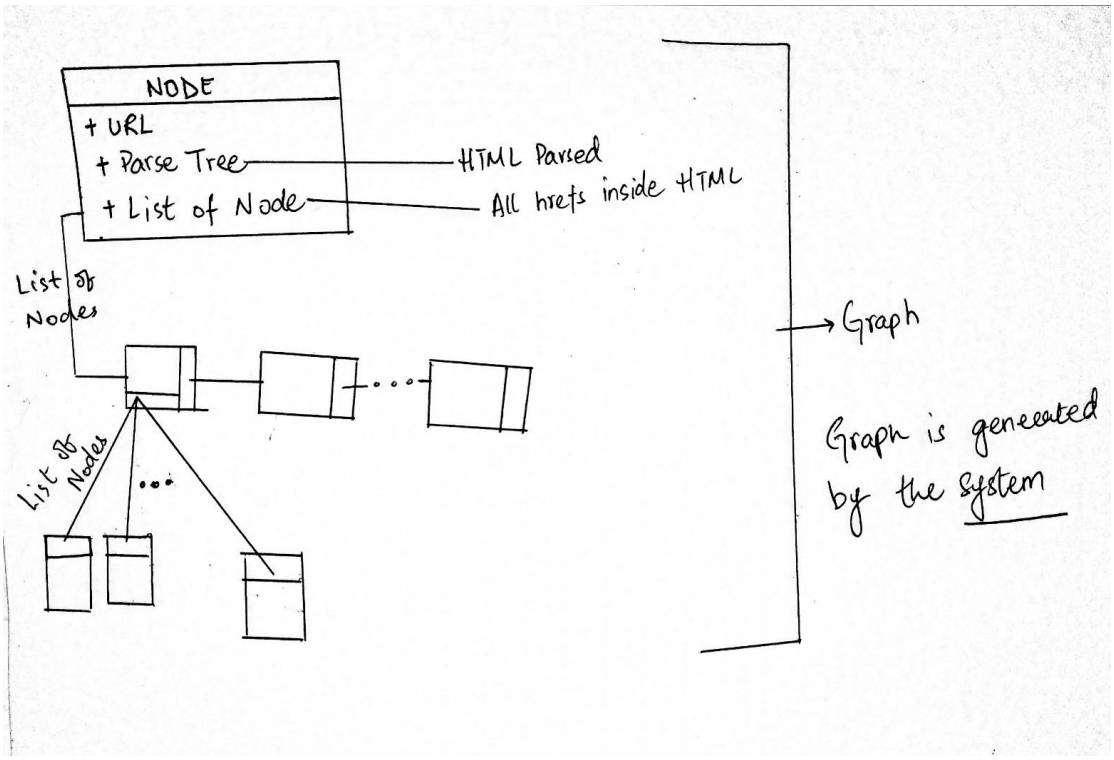


Figure 13.1: Potential Tree Data Structure

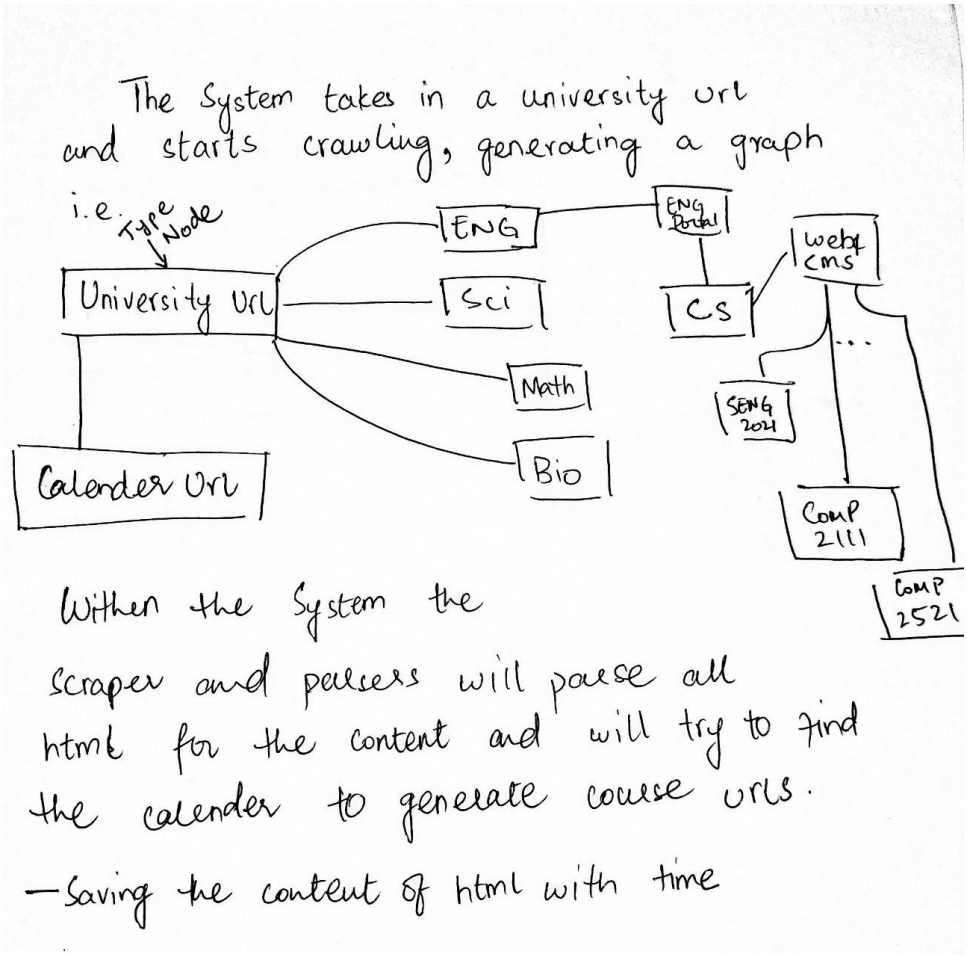


Figure 13.2: Webcrawling Process

This type of data structure would then require the future iteration of Raisin will make use of a permanent database to store the data regarding courses so that whenever the system is restarted, the URLs just need to be re-scraped in order to see if the course outline has been updated. The due date of an assessment can also be re-scraped to check if the deadline of an event has been changed by course admin. Then all the data can be written to CSV files to restore whenever the system shuts down. This would make use of the Python library "SQLalchemy".

Consequently by having a permanent database, Raisin could then store courses and their course assessments. This would then lessen the time a user has to wait as a the timestamp of the course would be cross checked with the online course outline for any changes, and act appropriately if necessary. **Then on Raisin's course selection webpage a message would appear indicating "X course has been updated!".** Raisin could then have a new feature allowing users to search for courses, which could serve as a barebones replacement than looking for the course outline themselves. Alternatively it could serve as a method for students to view the previous course commitments from past semesters.

Internally the database would be implemented in the following way.

System.ID = UNSW

CourseSystem.courses : List of Courses

Course.ID = UNSW_COMP2521

EventSystem.events : List of Events

Event.ID = UNSW_COMP2521_Ass

DeadlineSystem.deadlines : List of Deadlines

Deadline.ID = UNSW_COMP2521_Ass1

These changes on the back end would also have positive impacts upon the front end. A new security measure Raisin could implement as a result of now webcrawling would be being able to distinguish whether courses webpages are public or not, which would then minimise when a user's personal

information is required. Similarly Raisin could look to implement a subscription service, which could then notify the user by email whenever these courses are updated.

Overall following this plan for Raisin, the user interface on the front end would undergo minimal updates while the back end would become more sophisticated. Despite needing to refactor code, more time, potentially more workers and definitely more experience would be necessary to incorporate all of the new changes proposed.

14 PROJECT EVALUATION: SOFTWARE/PRODUCT

The main challenges our group faced when developing our application was inexperience in using the Google API/new libraries, creating a scraping tool and creating an engaging user interface. Thus this required time and experimentation in order to implement these tools properly so that they could achieve our specific function. We also had difficulty in creating an effective parser tool, since each course outline is unique: tutorial & laboratories are not always listed, there are a plethora of different assessment types, and assessment dates are not always listed, etc. Therefore with the limited time and experience our group had, we could only make a simple parsing tool, which naturally will not be able to correctly filter the assessments from every course outline. Another problem between our team was having two different team members working on the back end, on their own branches. This resulted in having two completely different systems/servers and classes which made merging impossible and required time to be wasted transferring features from one server to the other.

Taking this project in consideration, our group would look to make minimal changes if we were to redevelop our application again. Our team would look to better manage our time, by ensuring one server/back end is being worked on by team members so more time could be spent on refactoring code and good code principles. We could also look to challenge ourselves by setting more features or making our application compatible with “Moodle”. Furthermore, now having experience of what worked succeeded, we would attempt to further refine our front end/user interface and develop a more sophisticated parsing tool.

Although after fully designing a working application which has full functionality, our group believes we were able to solve all of the problem statements. Hence we are satisfied with our final product.

15 PROJECT EVALUATION: TEAM

Our group delegated responsibilities in accordance to an individual's strengths.

1. Richard was responsible for the reports and oversaw the responsibilities of the group
2. Rory was responsible for creating the web/PDF scrapers, the front end/user interfaces and revisions for the report. Rory and Faiz both worked on developing the system.
3. Faiz was responsible for implementing the process of exporting to a digital calendar, using the Google API and set up the alternative downloading methods. Faiz and Rory both worked on developing the system.
4. Yorke was responsible for creating diagrams and researching for the report

The primary issues our team faced was communication and role delegation as it often took time to receive clear responses from members leading to other members taking on the workload themselves. Subsequently our team faced commitment issues as not everyone was punctual or turned up to meetings which meant valuable face to face communication was lost.

Otherwise now completing the project and with hindsight, we would not make any drastic changes in terms of our application, focusing more on improving the group environment. Internally our group would make it mandatory that people are responding with their current status frequently to keep involvement high and track progress. Similarly we would use a task delegation application such as “Trello” to ensure each team member is aware and accountable for their responsibilities. Then we would design a set up consequences for anyone was not contributing or being as involved, as constant excuses or laziness had led to duties being neglected. We believe these changes would make everyone feel more committed to the project and accountable for their own actions, which would create a better working environment and a more interpersonal atmosphere.

Ultimately our project went smoothly and as expected. This is largely attributable to the early forecasting and planning during our idea generation process, which allowed our group to settle on a concept which would satisfy the requirements of the project. We were even able to implement more features than we initially decided upon since we had spare time. Thus our group was able to complete our project to a high standard and addressed all of the problems we outlined within the 10 week time frame.

16 PROJECT CONCLUSION

In conclusion, our group worked cohesively in order to finish the project, focusing on a real issue within our student lives and creating a web application that solves it.