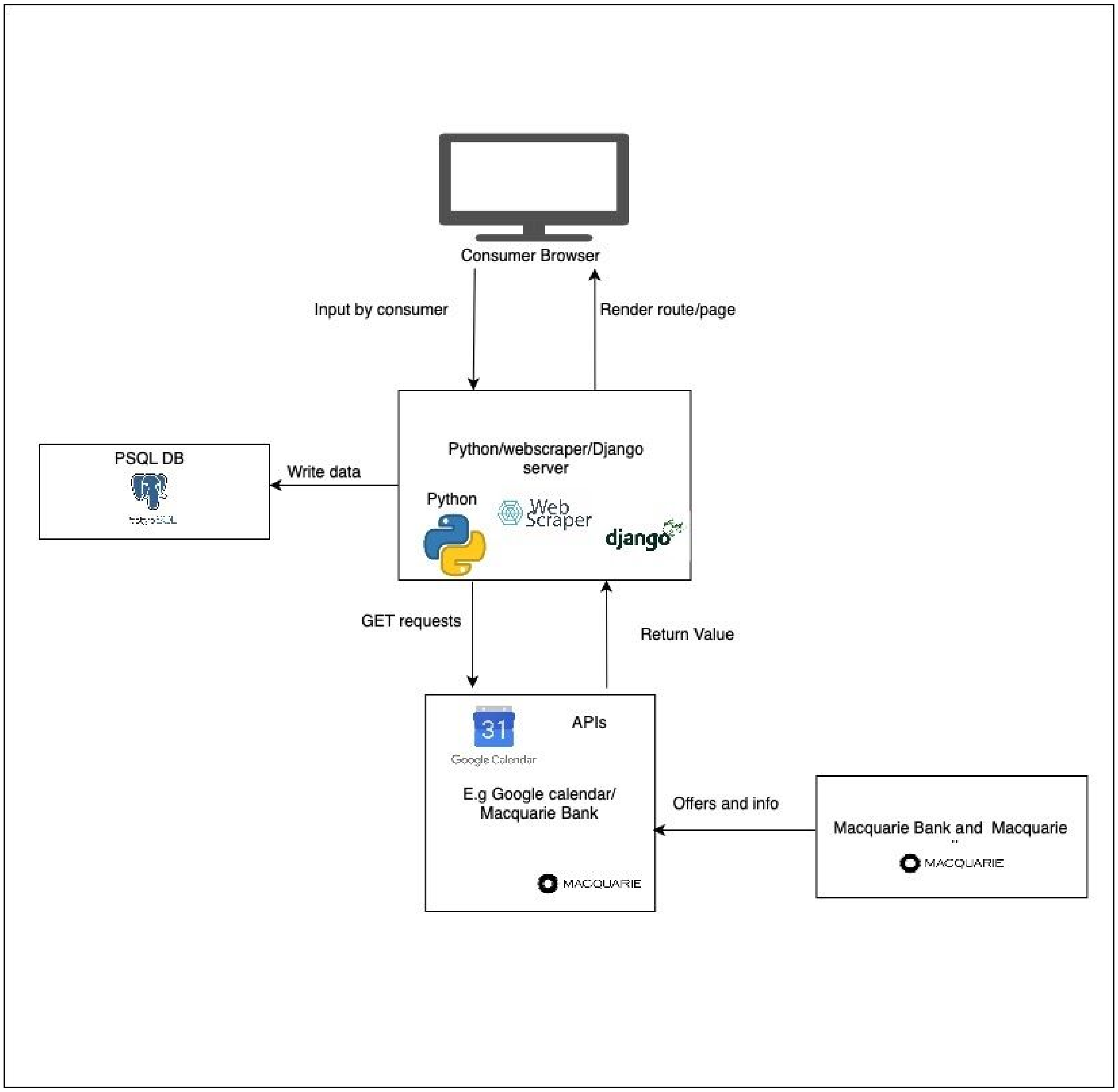
Deliverable 2- SENG2021

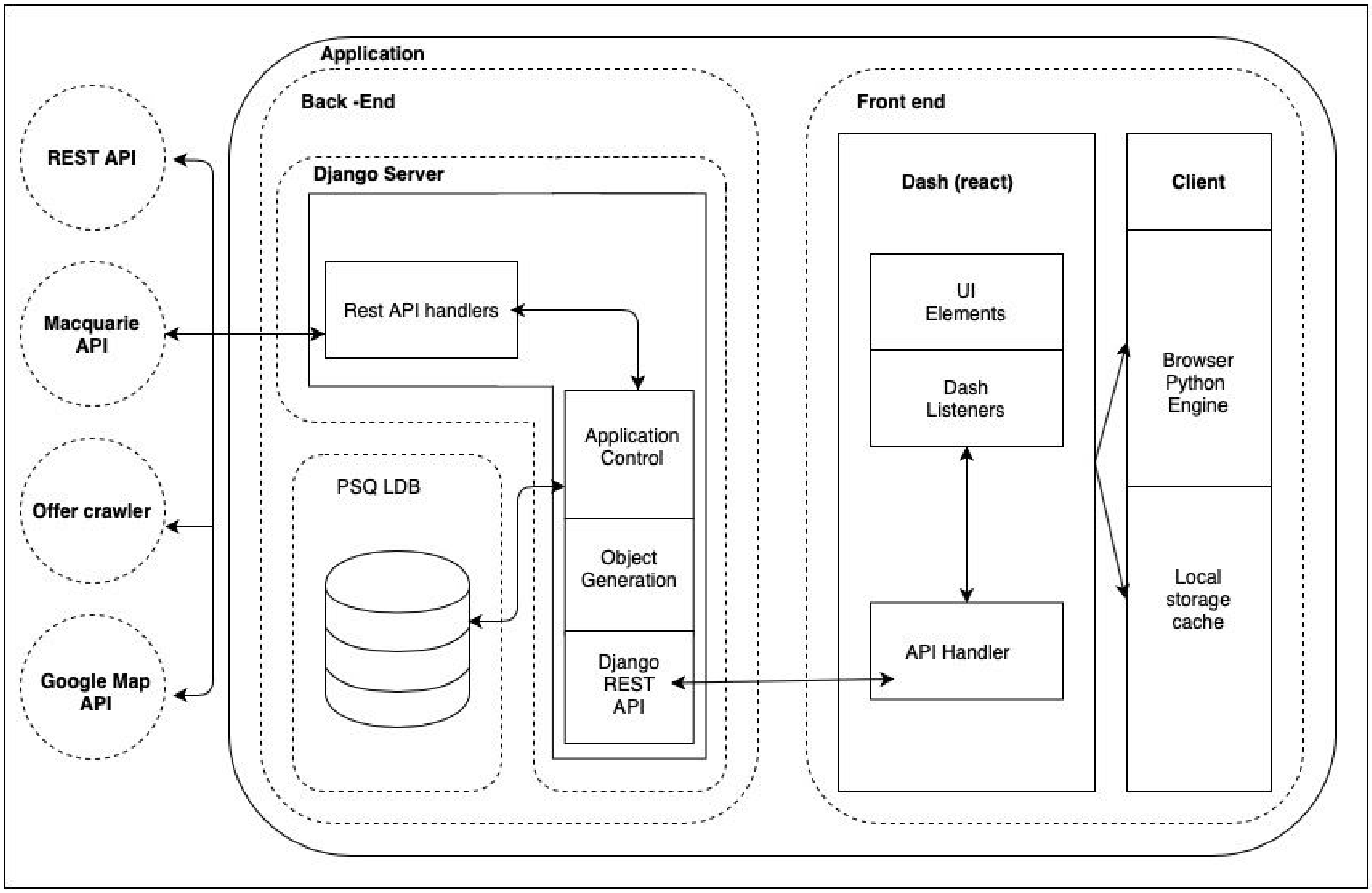
PART 1: SOFTWARE ARCHITECTURE

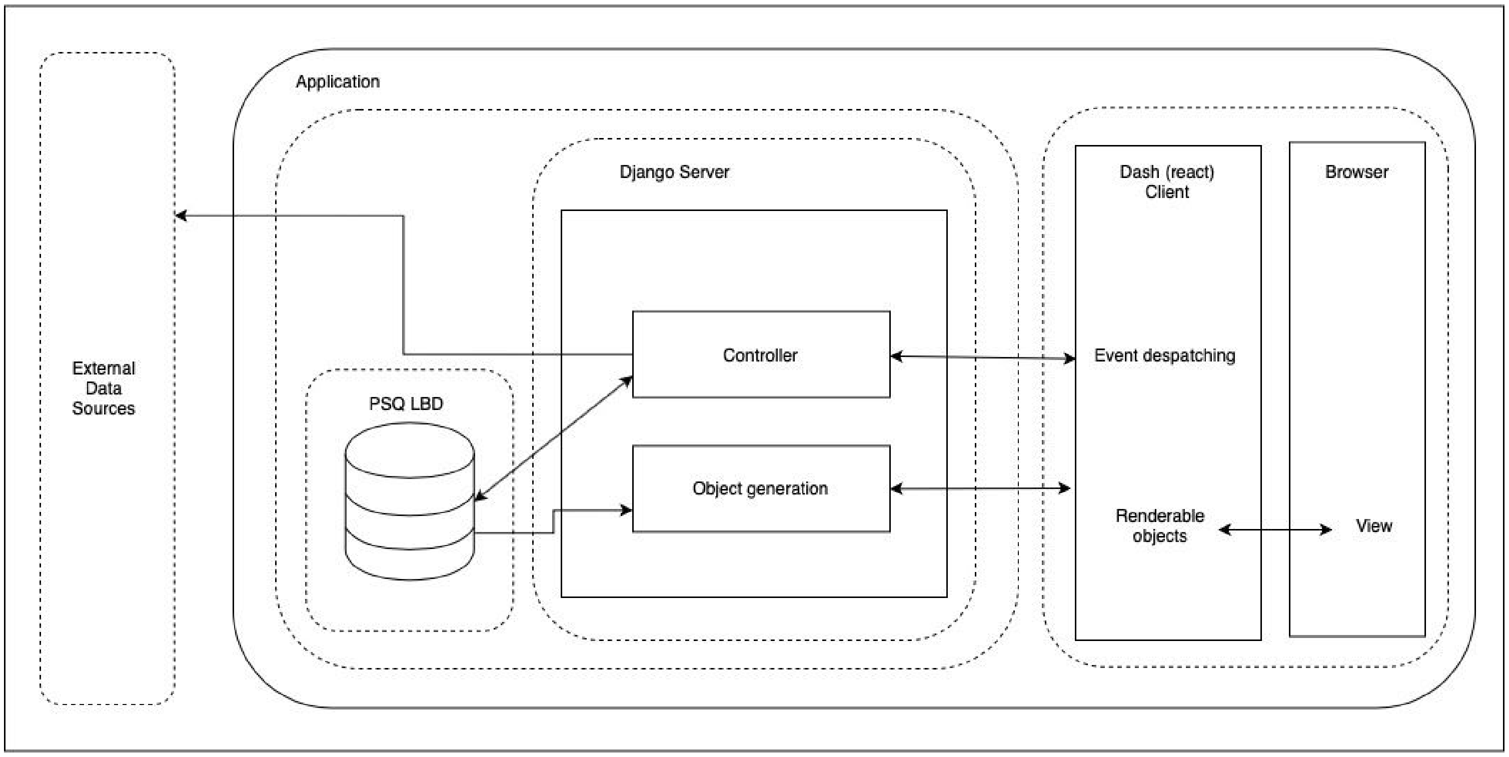
Software Components (Diagram):

The chart above depicts the major software components that comprise our solution.



A more visual and detailed view of our solution and the components that comprise it.





The diagram above is a more simplified version which explores the architure of the stack alone.

External Data Sources:

The backbone of this application is based off a MQ API Sandbox which will provide all necessary clientele banking data, ultimately allowing the website to serve its purpose: “Analysis of customer data to ensure all MQ clients have the ability to maximise the benefits reaped from the wide range of services MQ bank has on offer.” Further, to increase the number of features available the use of Google Calendar API and REST API will be used to allow appropriate responses from HTTP requests.

MQ API Sandbox:

- Provide necessary client banking data which will enable the creation of other features on the application i.e. “Expenses/Projections pages” to become available
- Returns a JSON object which the Django backend and Dash(react) client will simultaneously render into the subsequent page, complete with graphs and interactivity (features and analysis displayed will be based on the relevant page chosen)

REST API:

- Allow for user interaction with the application
- Using HTTP methods, API will allow users to make requests to the server in order to interact with the features available on the website

Google Calendar:

- Used in conjunction with the ‘Reminders page’ to allow users to interact with a calendar and hence set reminders
- API returns a list of events which Django pushes as a series of notifications to ultimately render a reminders page (through the help of a React client)

Language for Components:

Both the frontend and backend are crucial to web development with the Frontend referring to the client side in which data is converted to a graphical interface and the backend focusing on the server side (data access layer) of the application.

- The frontend of the web application will be developed entirely using HTML, CSS and Javascript for interactivity as all 3 languages are supported by most web browsers and operating systems.
- For the Backend implementation, the server will be run via Django(a python web framework) along with an Apache server. In addition, in order to effectively search, filter and present information based upon respective web requests from users, the backend database will be run by PostgreSQL.
- To join the python run Backend with the Frontend, Jinga(web template engine for python) will be used as it allows asynchronous execution and because the engine is “inspired” by Django, it is easier to integrate within our web stack.

HTML/CSS/JS vs HAML vs WIX:

HTML/CSS/JS

- Supported by all web browsers
- Integrate easily with other languages
- User friendly
- Display changes instantly

HAML

- Browsers only understand HTML→ HAML is just a templating language that gets transformed into HTML
- Makes CSS and JS integration more difficult
- Not universal among web developers
- Limited number of HAML/SASS templates available for use

Online Website Builders (eg. WIX)

- Lacks ability to create a website that looks and functions professionally
- Limited libraries and templates to choose from
- Shorter longevity and reliability
- Fiddly when dealing with links and media

Python vs Nodejs vs Java vs C++

Python:

- Good flexibility→ several Python implementations integrated with other programming languages
- Many frameworks available
- High level, object orientated
- Dynamically typed, known to be concise and readable → make developing apps faster than using languages like Java
- Open source language → multiple libraries available

Java

- Memory consuming and slower than compiled languages i.e. C++ → run 5PHP sites in equivalence for one java site → expensive hosting

NodeJs

- Inefficient in handling CPU-intensive apps- “generating audio, video, editing graphics are some concurrent requests which cannot be currently managed”

C++

- No security
- Complex to debug
- Bootstrap issues
- C++ is not used on the client side because there isn’t good infrastructure available.
- Easier to create templates using other languages → implementation is more difficult

Python was ultimately chosen as the main language for our web application as given its flexible nature, it is best able to integrate with all the other other frameworks chosen within our web stack. Also, all team members are highly familiar with using Python. Additionally it has multiple libraries available which will be essential given the dynamic range of features our website will have i.e. graphing, charts, other APIS. Given what’s listed above and the security, memory consumption and inefficiencies of the other 3 languages, Python was chosen as it had the greatest appeal.

Software Architecture (choice of components):

BACKEND:

Django vs Flask vs Bottle

Django:

- Full-stack Python web framework

- developed based on batteries included approach → easier to accomplish common tasks like user authentication and URL routing
- Provides built-in template engine, ORM system
- Older than the flask framework → wider choice of third party apps, plugins and extensions available
- Provides login and authentication abathat can be used for login purposes
- Provides its own Django ORM (object relational mapping) & uses data models → allow developers to link db tables with classes in a programming language so they can work with models in the same way as database references
- Provides a full-featured Model-View-Controller framework
- Large number of packages available

Flask:
- Lacks some of the built in features of Django
- Doesn't provide authentication/login features
- No default template engine
- No direct way to migrate the database
- Doesn't have ORM

Bottle:
- difficulty of finding documentation or online support.
- challenging to establish projects broader than 1000 lines → suitable for smaller apps, as it is challenging to manage → result of the single-file distribution pattern that lacks some Flask/Django blueprints

Django was chosen considering our web application would be implemented using Python and there is a strong integration between the language and framework. Despite Flask/Bottle having the advantage of the ease with which smaller applications can be built, the large suite of features and functionalities Django has on offer outweighs both web frameworks for this instance.

Dash vs Nodejs

Dash
- Dash uses react and is generated entirely from python
- bind interactive components (dropdowns, graphs, sliders, text inputs) with Python code through reactive "callbacks".
- Easier to integrate with complicated UIs (with multiple inputs, multiple outputs, and inputs that depend on other inputs)
- Multi-user → "state" of the app is entirely in the client → multiple users can view apps and have independent sessions.
- Uses react.js to render components
- Interactive graphing component → able to write applications that respond to hovering, clicking, or selecting points on the graph.

- Component class for every HTML tag as well as keyword arguments for all of the HTML arguments

Node js
- Lacks consistency -> API changes frequently, and the changes are often backward-incompatible.
- Best suitable for small projects
- Not simple to understand and work with
- Documentation of libraries not easily available online
- Does not have interactive components (dropdowns, graphs, sliders, text inputs) like Dash does

The interactivity and graphing libraries within Dash make it appealing for the development of Bankquire as visual depictions of customer data are a large feature of this application. Hence Dash as a framework was preferred over Node js given the requirements being dealt with. Additionally given the framework is generated from python it seamlessly integrated with the rest of the framework. Finally, given you don't have to learn other technologies like Javascript the implementation of Dash is much simpler than that of Node js.

Postgresql vs mongo db vs sqlite vs aws Aurora
Postgres
- Object Relational Database Management System.
- and like Python is cross-platform
- Extensible and supports data scaling
- Relational db
- Diverse indexing techniques
- Simpler to manage data regardless of the size of the dataset

Mongodb:
- Joins on data are not supported
- No default transaction support.
- Data consumption is high.
- No ACID(atomicity, consistency, isolation, durability) guarantee and rollbacks have to be handled by the application

Sqlite
- Lack of multi user capabilities which can be found in relational databases like postgresql i.e. cannot handle concurrent writes
- Not client/server → poor for multi-user applications
- Large datasets cause Sqlite to stop or slow down

Amazon aws Aurora (relational database service)

- The source code is not available
- It's only compatible with mySql-5.6.19 - so we can't use features in newer or older versions of mysql .
- It's not free of cost.

All 4 variations of a database have their benefits and disadvantages however PostgreSQL helps protect data integrity and it is much simpler to manage datasets regardless of the size. Also, all the team members are familiar with PostgreSQL. Considering the website administers large amounts of client data, for banking it is essential that the web frameworks and database procedures used have a high level of data security attached and ability to efficiently manage and manipulate the data, hence we choose PostgreSQL as the prefered RDBMS .

Jinja vs TypeScript

Jinja:

- Python programming language
- Source code compilation→ better run time performance
- Has asynchronous execution
- Flexible in what the templates can contain i.e. support macros and python-like constructs
- Since we're using Django→ jinja is inspired by Django → strong correlation and interdependence
- Security features are tightly integrated → required for a banking app and hence proves useful

TypeScript:

- Long time to compile code
- Needs to be a definition file available if using a third party library (not always available)
- Overly complicated typing system
- Gives false sense of security

The python based language has strong interdependence with the rest of the selected web stack (as it is inspired by Django) and hence jinja proves to be the most suitable language in order to integrate the frontend and backend. Additionally, the security features are highly integrated making the language desirable to ensure the website is built securely and efficiently.

Apache vs NGINX:

Apache:

- can be modified to adjust the code and also to fix errors.
- ability to add more features and modules
- changes made are recorded immediately, even without restarting the server.
- can run on almost any operating systems like Windows, Linux, macOS etc.

- multiple websites can be run from the same server. In other words, it can create virtual hosts on the same server.
- Flexible
- Greater selection of modules

NGINX

- Less extensive list of modules
- Less community support and documentation available as compared to Apache

The selection of modules Apache provides and reliability of this web server as opposed to NGINX because of its longevity meant that Apache was favoured. Also, it is available for free and no license is required hence making this server more favourable.

Deployment Hosts

Below are the comparison of deployment hosts we considered:

Host	Price	*Performance	Security	Suitable for production	Industry relevance
Localhost	Free of cost	Not acceptable, can only be run on one machine	Not secure, For LAN access only	No, only accessible by the person running the machine	N/A- Not used for production deployment
AWS	12 months free, ~4.8USD/month	2vCPU, Memory: 0.5 GB	Highly secure. Prefered by most businesses	Highly suitable for large organisations. Provides great features	Highly used, ranked on top for market share with 61% (as in 2019)

Google Cloud Platform	12 months free trial, ~3.32USD/month (customer friendly pricing)	2vCPU Memory: 0.6GB	Very secure. Google gives security high priority	Not the best option for big businesses. Although there have been significant growth in expansion	Ranked 3rd with 6% market share (as in 2019)
Microsoft Azure	12 months free trial, ~4.7USD/ month	2vCPU, Memory: 0.5GB	Highly secure	A great choice for businesses. Rapidly scalable and feature rich cloud	Used moderately, ranked 2nd with market share of 17.6% (as in 2019)

*Performance depends on what type of machine/instance you are paying for, where the machine is located and the machine specs. The above comparisons are done on a Linux machine when, instance size = cheapest available instance from each provider, location = Sydney and Machine-type for Google: f1-micro, Azure: t3.nano, AWS: B1LS(instance).

Summary:
Given the benefits, flexibility and success Python has delivered over a period of time as a suitable web development language, Python was chosen as the main coding language. The rest of the web frameworks were simultaneously chosen based on the level of interdependence shared with Python (i.e. Python based frameworks) and if they aligned with the chosen features/requirements of Bankquire. Hence, the frontend will be developed using HTML/CSS, JavaScript and Dash which will collectively create a strong framework for the interface of the website. Additionally, Django in conjunction with PostgreSQL were chosen to develop the backend of the website as Python can be used with both and they provide a large number of libraries/formats that can aid with effectively managing a large dataset. Finally, as mentioned before, the modern Jinja framework was selected as it seamlessly integrates both the frontend and backend and an Apache server was chosen as the web server. Given the nature of this assignment and after much consideration, the website will be deployed locally but with further time the benefits of AWS outweigh that of a local host, hence proving as a suitable deployment platform.

PART 2: INITIAL SOFTWARE DESIGN

Updates in user stories:
To ensure the application was completed at a high standard under a timely manner and under the given restrictions, two features/user stories were removed (from our first deliverable).

1. The “Tips” option, which provided a page to easily access common tips provided by Macquarie Bank, was removed so as not to overcomplicate and repeat the offers and schemes feature provided by Bankquire
2. The “More information” page which utilised a GoogleMaps API was removed to ensure the addition of a third API didn’t affect the development of the website as a whole and allowed the team to enhance the details of existing personalised features.

1) **Feature:** 1
As a: Macquarie client
So that: I am aware of all the exciting offers provided by the bank and further able to make use of all of them
I want to: know about the schemes and offers my Bank provides

GIVEN I am logged in on the website and want to know the offers my bank provides
WHEN I click on the “Offers and Schemes” which is the 5th button on the Toggle Menu
THEN A page with all the offers provided by the bank comes up in a tabular format
WHEN I click on a particular offer
THEN I am directed to the Macquarie bank website which gives me more information about that particular offer

2) **Feature**: 3a
As a: Macquarie client
So that: I am able to view and visualise my monthly expenditure and account balance
I want to: view a bar graph of my spendings for the year, categorised monthly

GIVEN I am logged in on the website and want to view a summary of my transaction history
WHEN I click “Profile”, which is the first button on the toggle menu
THEN I will be directed to my profile page which will show two graphs; one for my account balance and one for my expenditure for the last 12 months
WHEN I hover over each bar in the graph
THEN dependent on the graph**,** my exact spending or account balance for that month will appear.

3) **Feature:** 3b
As a: Macquarie client
So that: I am able to visualise a categorised summary of my transaction/spending history
I want to: view a pie chart of my spendings for the year (or month) categorised by essentials such as food, travel, petrol etc.

GIVEN I am logged in on the website and want to view my transactions
WHEN I click “Expenditure” which is the 2nd button on the toggle bar at the top of the page
THEN a pie chart depicting my spendings, over a set period of time, and a list of transactions will appear
WHEN I change the length of time for my transactions
THEN my pie chart will change to illustrate my transactions for the duration chosen.

4) **Feature:** 3c
As a: Macquarie client
So that: I am able to understand the details (what and how much) of my expenditure within a particular category
I want to: view my transaction history by category

GIVEN I am logged in on the website and want to view my transactions
WHEN I click the “Expenditure” button which is the 2nd button on the toggle menu at the top of the page
THEN a pie chart depicting my spendings in the year and a list of transactions will appear
WHEN I click on a particular category on the pie chart
THEN my transaction history for that particular category in the last year will appear.
WHEN I click the monthly button on the top left of the screen
THEN my transaction history for that particular category in the last month will appear.
WHEN I click the yearly button on the top left of the screen

THEN my transaction history for that particular category in the last year will appear.

5) **Feature:** 4
As a: Macquarie client
So that: I am able to better understand my future account balance and expenses
I want to: View my projected savings and expenses based on my transaction history.

GIVEN I am logged in on the website and want to view my calendar
WHEN I click the "Projections" button which is the 4th button on the toggle menu at the top of the page
THEN two graphs will appear, depicting the predicted account balance and expenditure for the next 12 months

6) **Feature:** 6
As a: Macquarie client
So that: I am able to compare and reflect on my expenses
I want to: view the differences between my categorised expenses and that of the average person in the same age bracket

GIVEN I am logged in on the website and want to view the comparisons
WHEN I click the "Expenditure" button which is the 2nd button on the toggle menu at the top of the page
THEN a table will appear showing the differences between my expenses and that of the average person in the same age bracket per category.

7) **Feature:** 7
As a: Macquarie client
So that: My Bank and Transaction details are secured
I want to: Login into my account using Client_Id and password

GIVEN that I am on the login page of the website
WHEN I click on the login field I am directed to the Macquarie Bank login page
THEN I enter my login details
WHEN I authorise the website to access my account
THEN I should be able to gain access to my personalised account within the website

8) **Feature:** 8
As a: Macquarie client
So that: No one else can get access to my account details
I want to: Logout of my account

GIVEN that I want to end the session
WHEN I click on the logout field

THEN All my data is securely removed from the website
AND I am logged out of my Macquarie bank account

9) **Feature:** 9a
As a: Macquarie client
So that: I am able to see any upcoming payment reminders and plan accordingly
I want to: View all upcoming payments in a calendar format

GIVEN I am logged in on the website and want to view my calendar
WHEN I click the "Payment Reminders" button which is the 3rd button on the toggle menu at the top of the page
THEN a calendar, which will give me the ability to create new reminders, and a list of all my reminders will appear
WHEN I hover and scroll on the google calendar
THEN my foregone and scheduled payment reminders will appear

10) **Feature:** 9b
As a: Macquarie client
So that: I am able to see any upcoming payment reminders and plan accordingly
I want to: Create scheduled payments

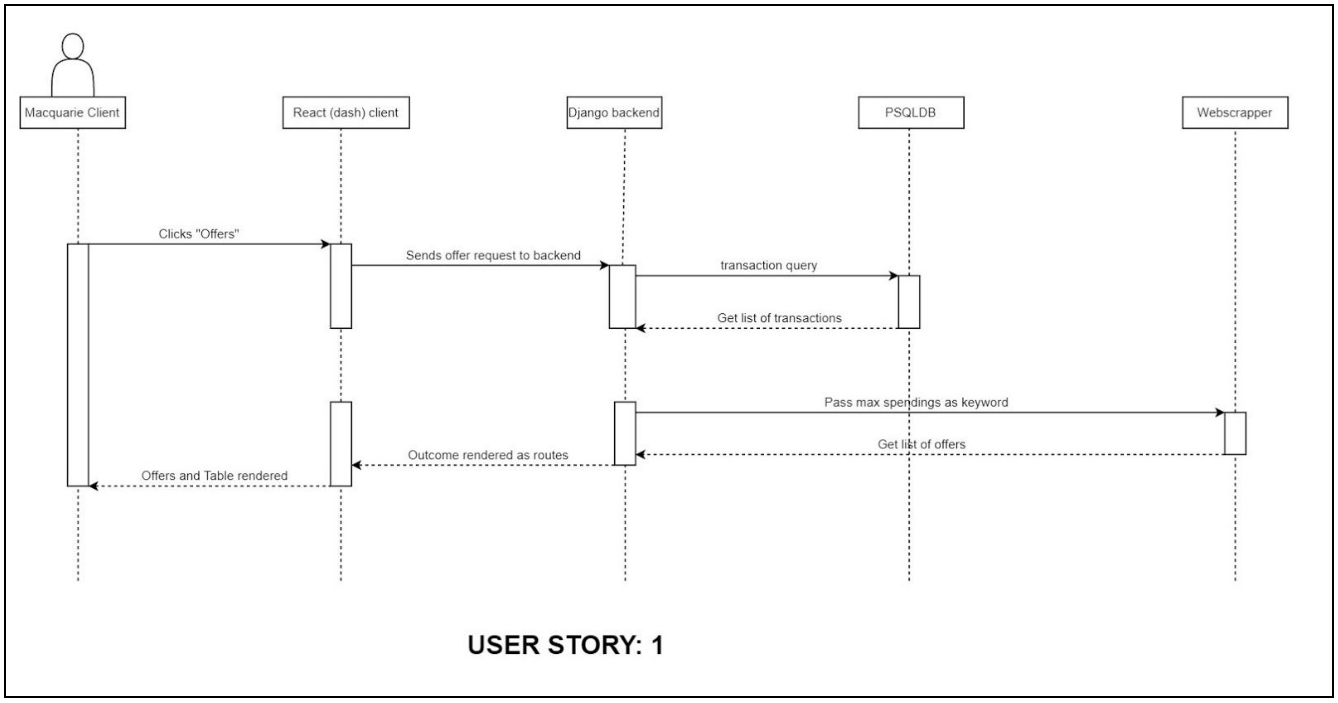
GIVEN I am logged in on the website and want to view my calendar
WHEN I click the "Payment Reminders" button which is the 3rd button on the toggle menu at the top of the page
THEN a calendar, box to create new reminder and a list of reminders will appear
WHEN I fill in the company's name, description and frequency and click send at the bottom right of the box
THEN I am able to create a scheduled reminder.
WHEN I click on the edit button on the right
THEN I am able to change the name, description or frequency of the reminder

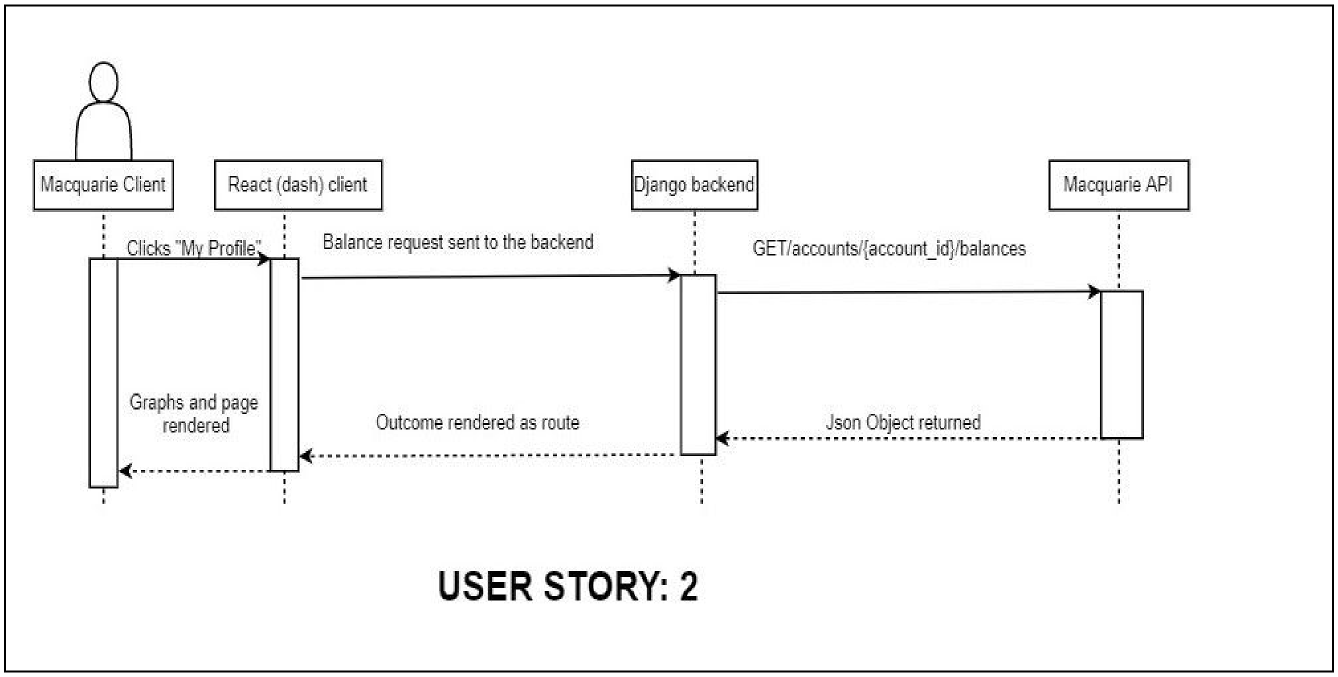
11) **Feature:** 9c
As a: Macquarie client
So that: I am able to know and plan for my upcoming expenses
I want to: Edit and delete scheduled payments in a calendar.

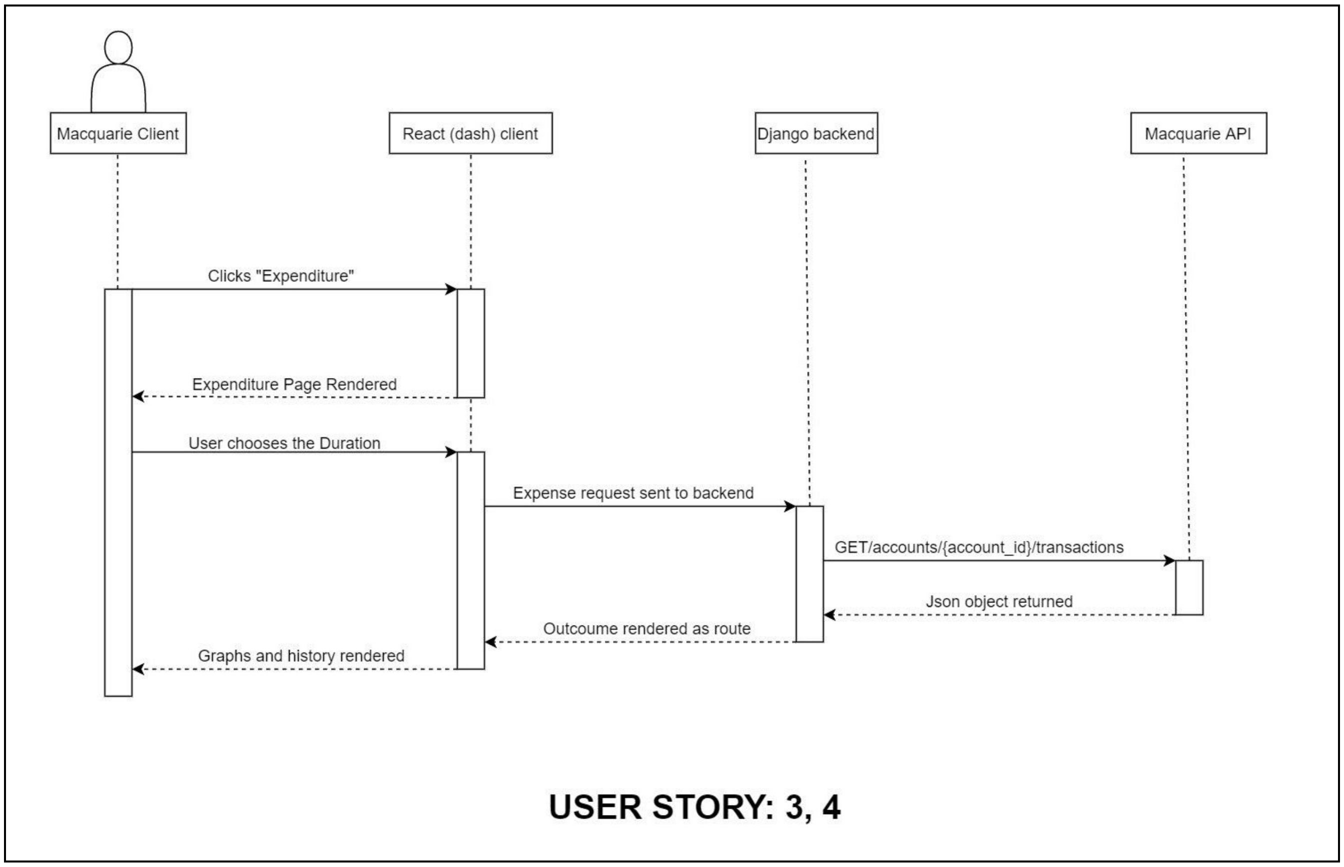
GIVEN I am logged in on the website and want to view my calendar
WHEN I click the "Payment Reminders" button which is the 3rd button on the toggle menu at the top of the page
THEN a calendar, a form that allows you to create new reminders, and a list of current reminders will all simultaneously appear
WHEN I scroll down to the bottom of the page
THEN I am able to view all my scheduled reminders.

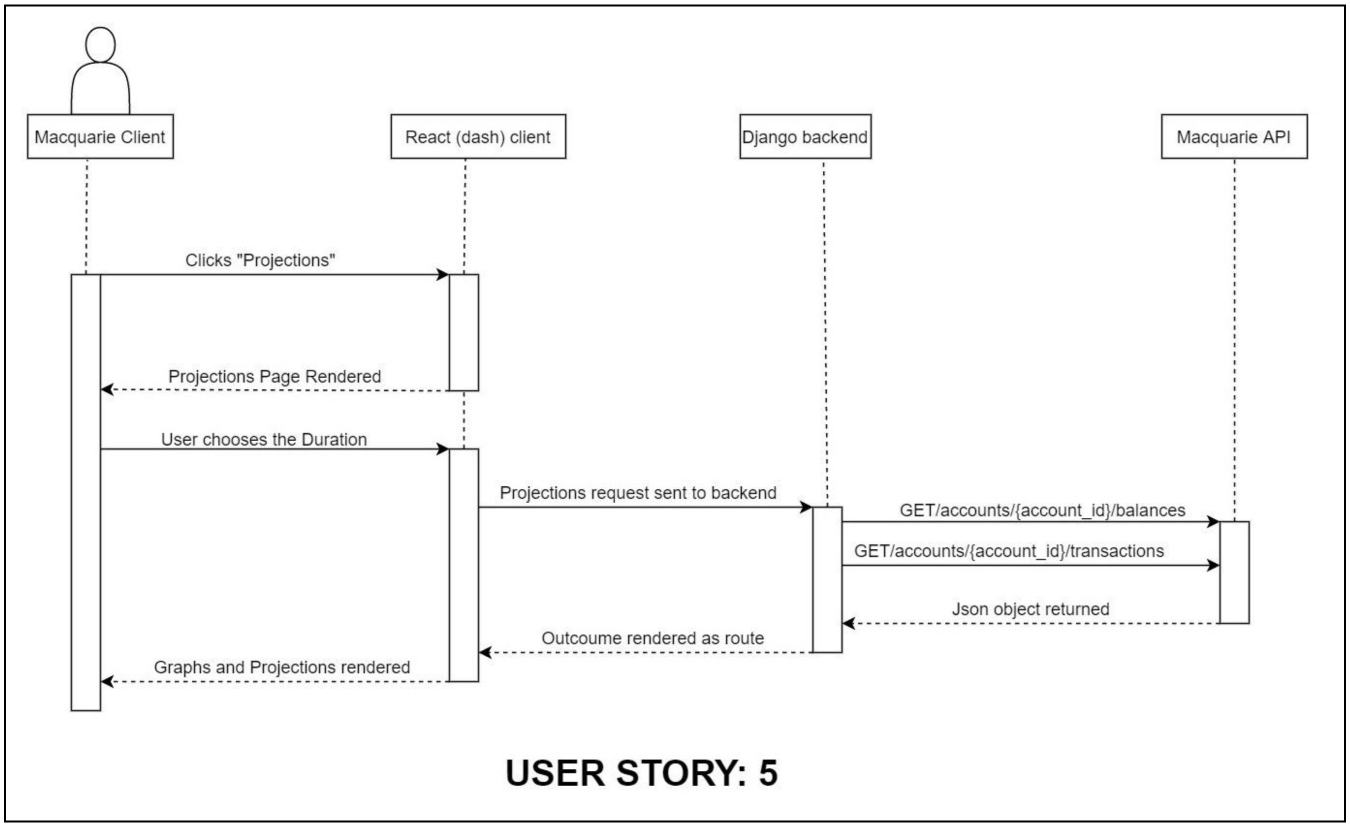
WHEN I click on the edit button on the right of the chosen reminder
THEN I am able to change the name, description or frequency of the reminder
WHEN I click the delete button on the right of the chosen reminder
THEN the reminder will be deleted

UML Diagrams:

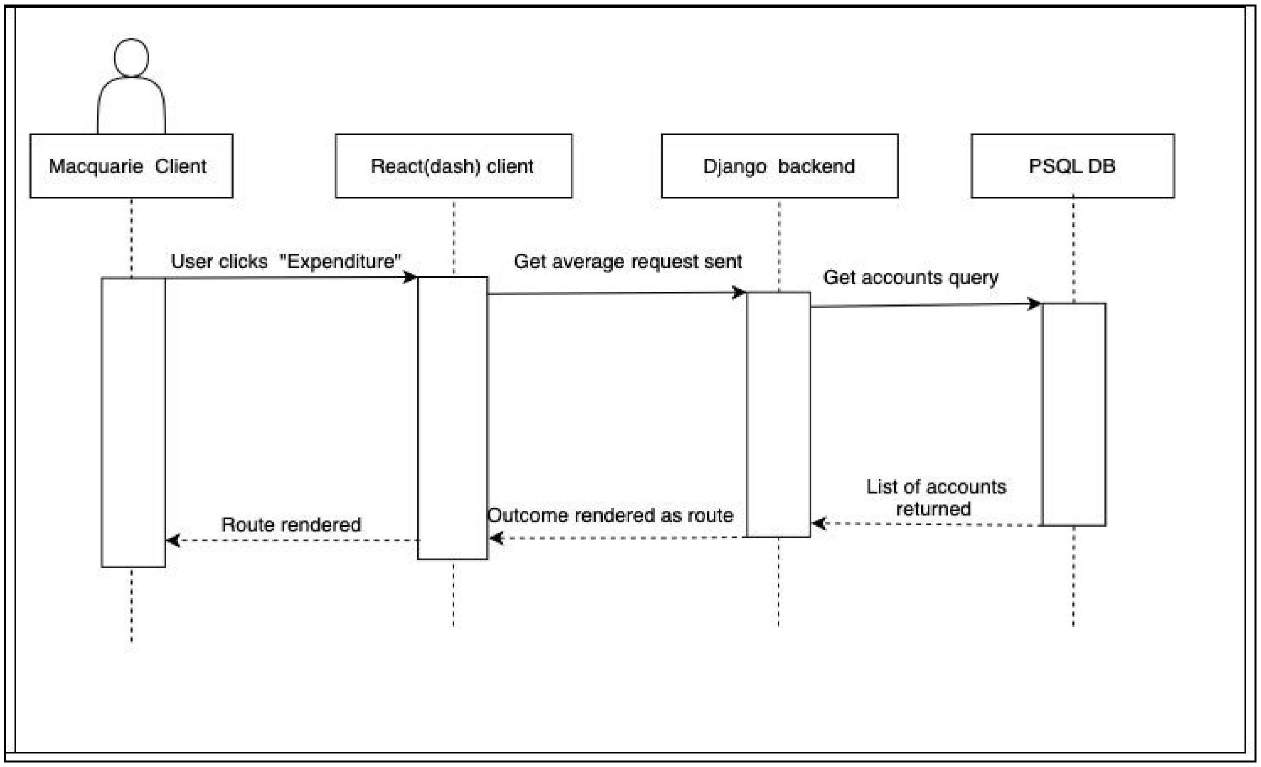




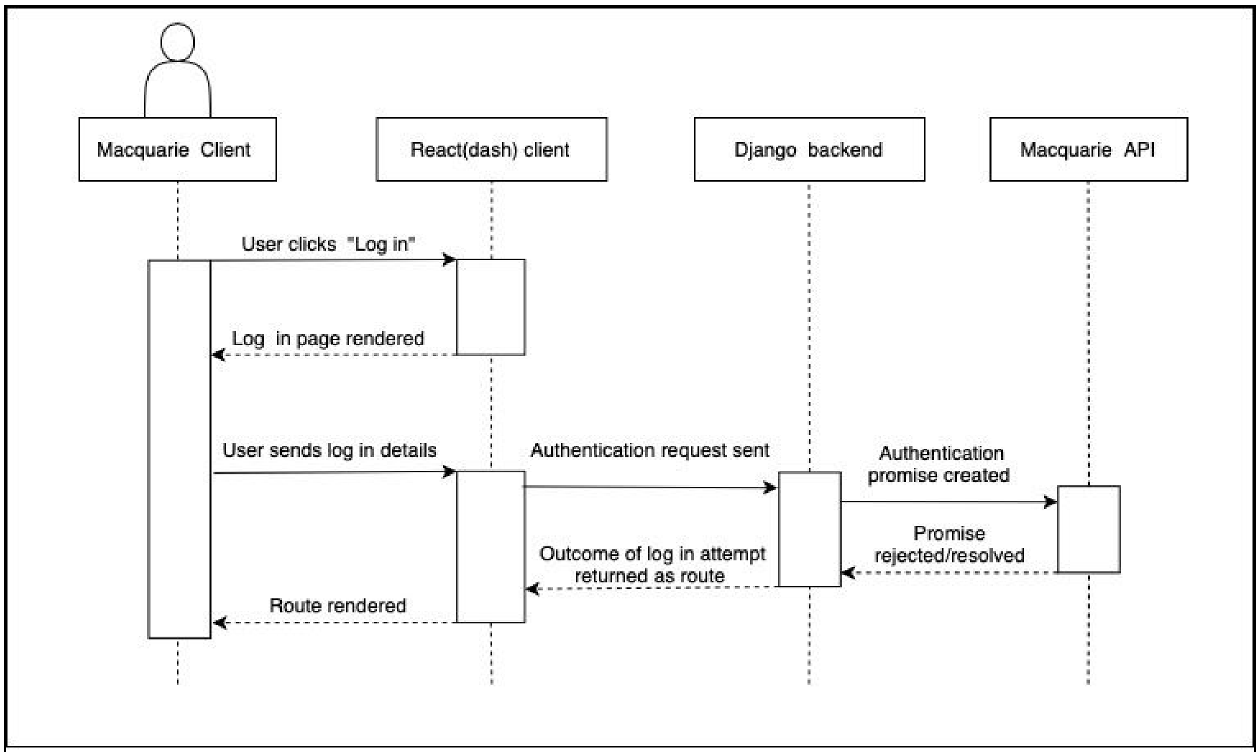

Macquarie Client
React (dash) client
Django backend
Macquarie API
Clicks "Expenditure"
Expenditure Page Rendered
User chooses the Duration
Expense request sent to backend
GET/accounts/{account_id}/transactions
Json object returned
Outcoume rendered as route
Graphs and history rendered
USER STORY: 3, 4

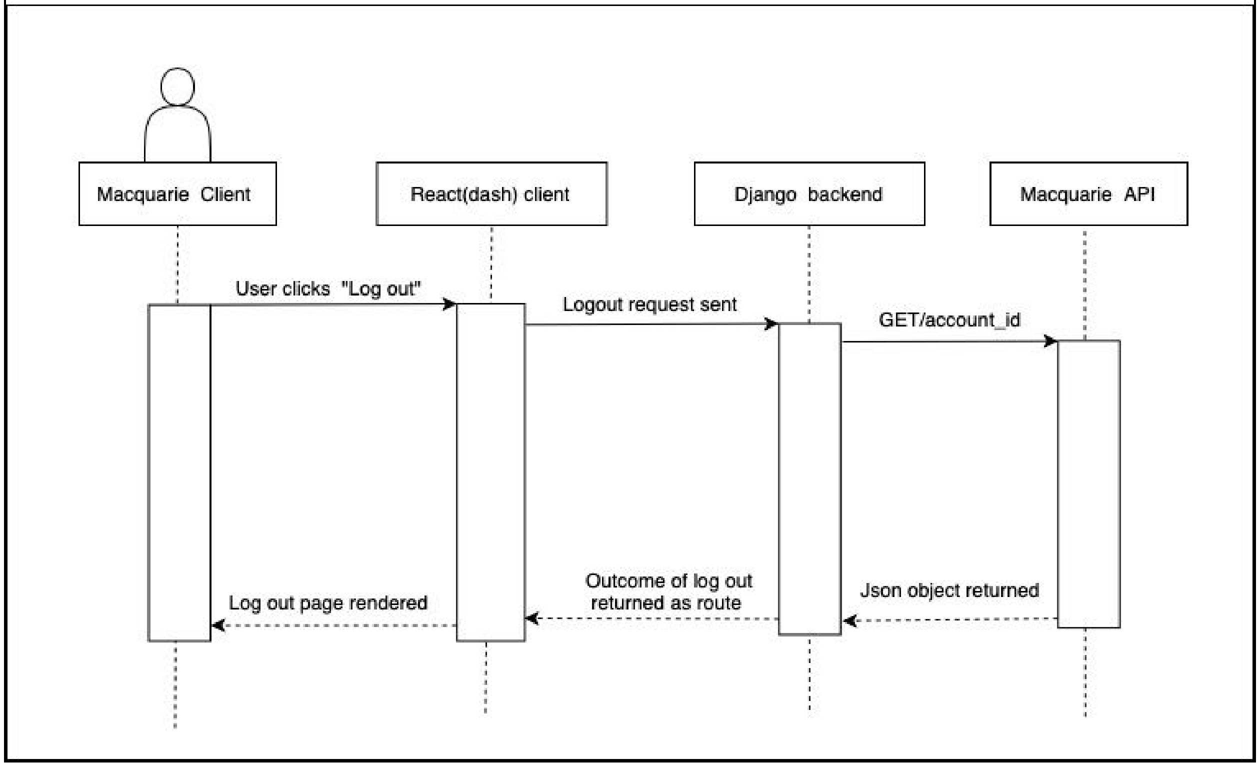

Macquarie Client
React (dash) client
Django backend
Macquarie API
Clicks "Projections"
Projections Page Rendered
User chooses the Duration
Projections request sent to backend
GET/accounts/{account_id}/balances
GET/accounts/{account_id}/transactions
Json object returned
Outcoume rendered as route
Graphs and Projections rendered
USER STORY: 5

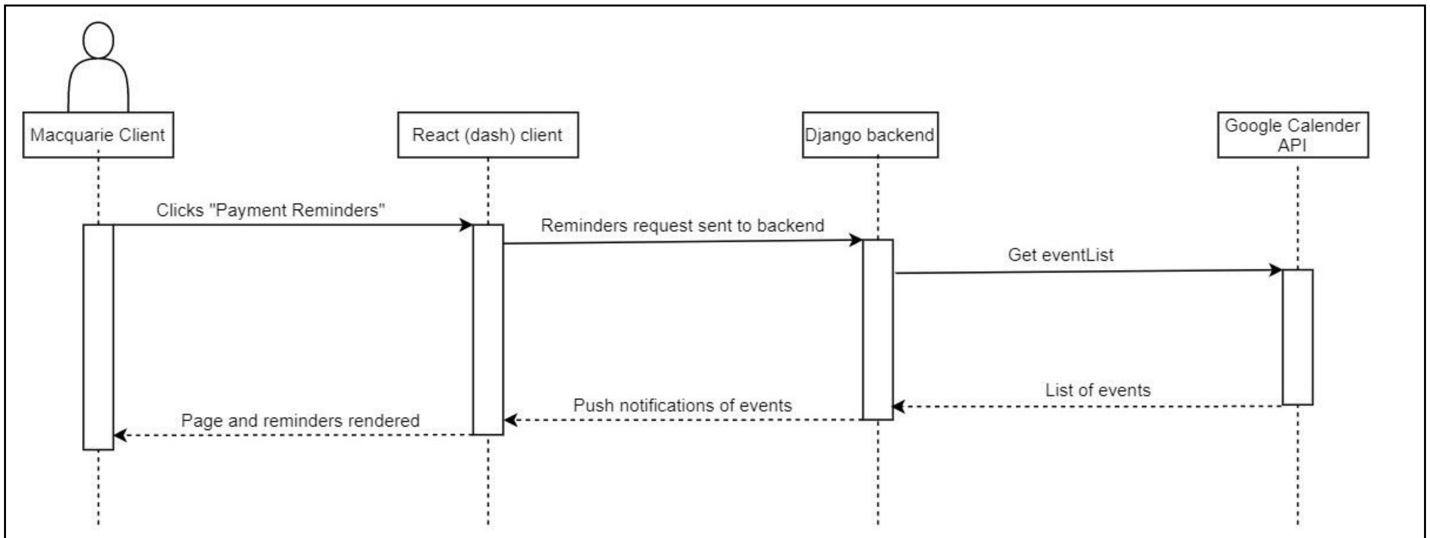
User story: 6


Macquarie Client
React(dash) client
Django backend
PSQL DB
User clicks "Expenditure"
Get average request sent
Get accounts query
List of accounts returned
Outcome rendered as route
Route rendered

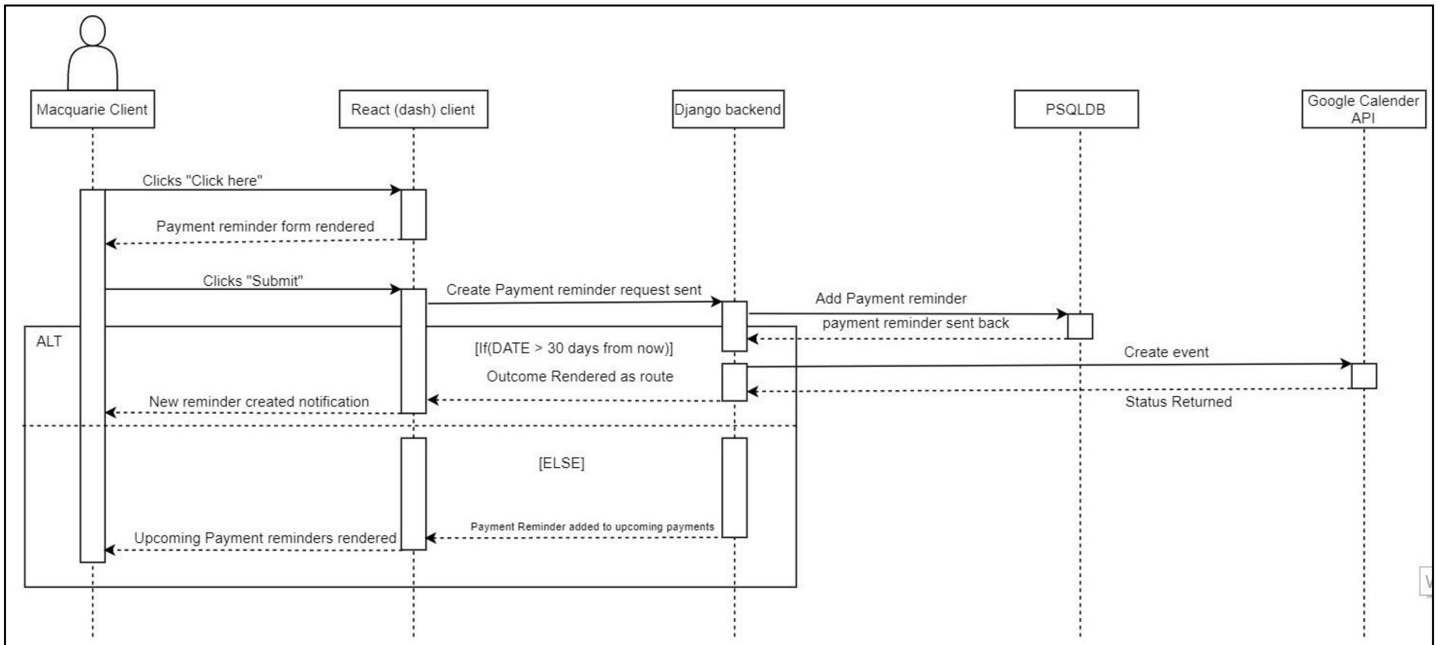
(User story : 7 and then 8)


Macquarie Client
React(dash) client
Django backend
Macquarie API
User clicks "Log in"
Log in page rendered
User sends log in details
Authentication request sent
Authentication promise created
Promise rejected/resolved
Outcome of log in attempt returned as route
Route rendered

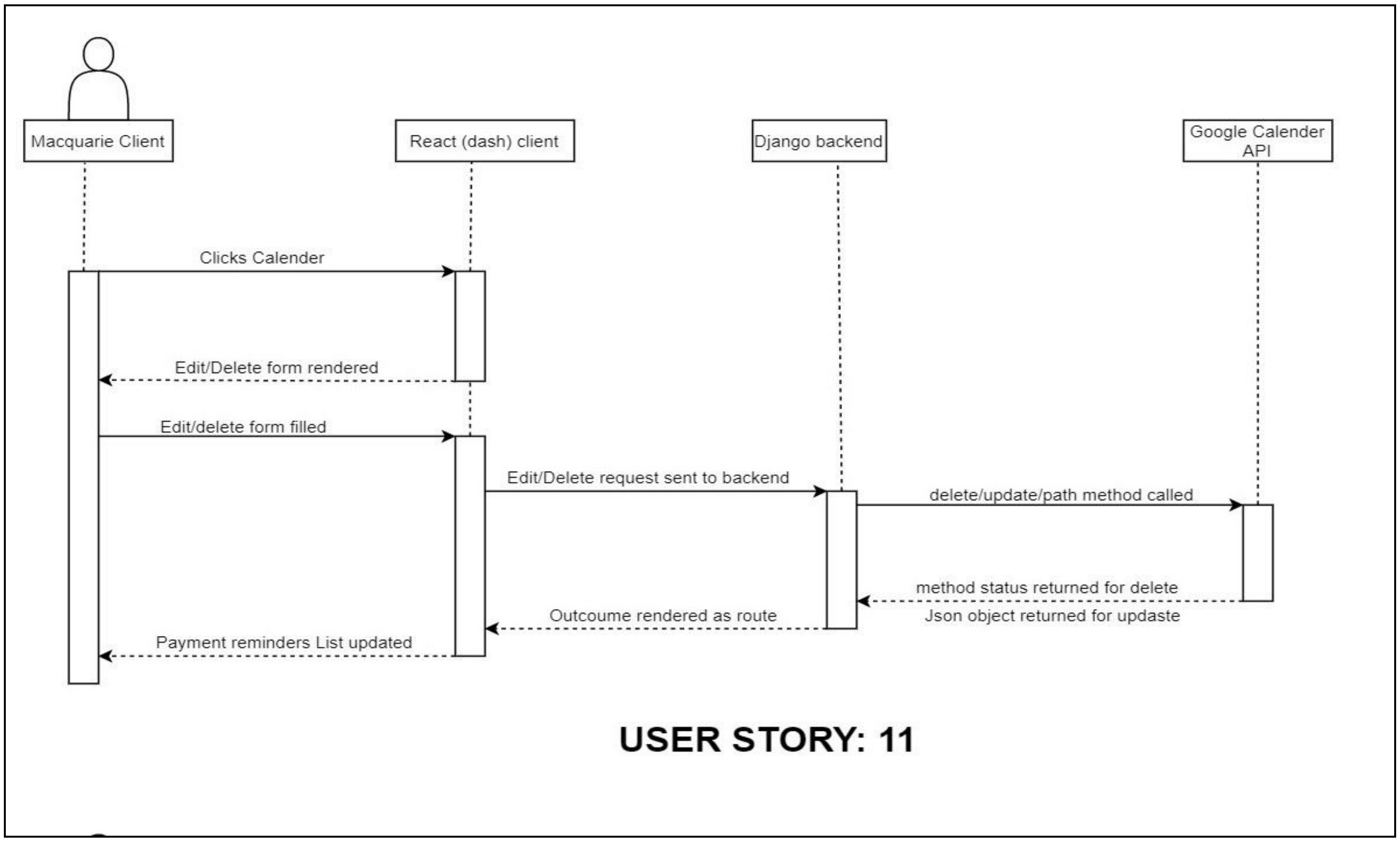



Macquarie Client
React (dash) client
Django backend
Google Calender API
Clicks "Payment Reminders"
Reminders request sent to backend
Get eventList
List of events
Push notifications of events
Page and reminders rendered

USER STORY: 9


Macquarie Client
React (dash) client
Django backend
PSQLDB
Google Calender API
Clicks "Click here"
Payment reminder form rendered
Clicks "Submit"
Create Payment reminder request sent
Add Payment reminder
payment reminder sent back
ALT
[If(DATE > 30 days from now)]
Create event
Outcome Rendered as route
Status Returned
New reminder created notification
[ELSE]
Payment Reminder added to upcoming payments
Upcoming Payment reminders rendered

USER STORY: 10


Macquarie Client
React (dash) client
Django backend
Google Calender API
Clicks Calender
Edit/Delete form rendered
Edit/delete form filled
Edit/Delete request sent to backend
delete/update/path method called
method status returned for delete
Json object returned for updaste
Outcoume rendered as route
Payment reminders List updated
USER STORY: 11