INTRODUCTION:

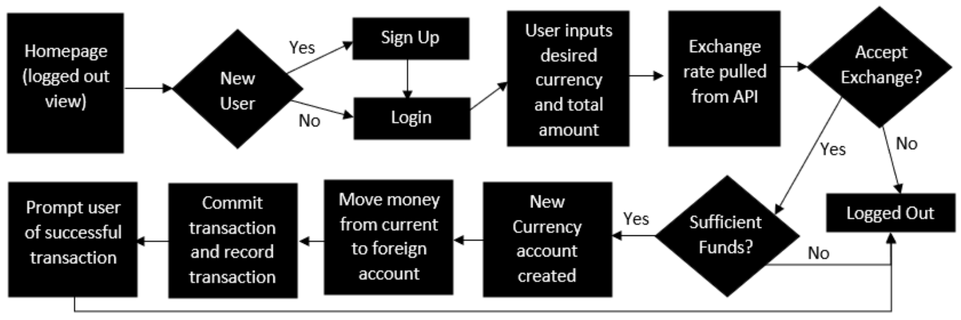
Have you ever been on holiday and treated foreign money like monopoly money? Or been frustrated by the surcharges faced when converting currency? Us too. Thus, 'Xchange' was born - a web-based banking app, which aims to solve these problems by allowing users to sign up to create an online banking account and then exchange GBP into any currency desired! Just like that a linked sub-account is created at no extra cost and allows the user to feel in control of their own money and know what to expect when using the new currency abroad. As a group, we created this app using Python, Flask, SQL, HTML and CSS (Bootstrap) and this report aims to detail the journey we went on to create it; from background and design to execution to finally discuss our main learnings.

BACKGROUND:

When considering the need for a product such as Xchange, we identified that approximately 80% of people prefer to use a card over cash when spending*, this also applies when abroad. Therefore, we created something that simulates the ability to easily spend money abroad whilst still feeling in control.

An Agile Framework was adopted early on in the project (the reason for this will be discussed in more detail later) and we decided on the key functionality we would like Xchange to include within our first sprint planning session. This included concepts we had learned in class such as creating bank account related classes and functions, linking to an API for exchange rates*, linking to an SQL database and extracting queries and building a flask structure to host multiple pages using app routes. Additional concepts which required further research such as login functionality, creation of forms for user input such as registering a new user and using front-end languages such as HTML, CSS and elements of Bootstrap to create a user-friendly interface.

For our app, the user can sign up to create a new account using a unique email, or login if an account already exists. Upon creation of an account, a user and a bank account containing £1000 are simultaneously created. Hashed passwords are posted to the database, therefore hashed passwords are compared when users log in - if there is no match, the user will not be logged in. After login, the user can select any amount and any currency they wish to exchange. After this, the user will be shown the exchange rate and amount at that time of request; this can be accepted or declined. Declining the exchange will log the user out. Should the user accept, the money will only be exchanged if they have enough funds in their account to support the request, if they do not an error is raised and they are logged out. If they do have enough funds, the transaction is successful, money is deducted from their current account and sub-foreign account is created holding the new balance. See Diagram 1 below.
<u>Diagram 1: Core Components & Functionality*</u>


Homepage (logged out view)
New User
Yes
No
Sign Up
Login
User inputs desired currency and total amount
Exchange rate pulled from API
Accept Exchange?
Yes
No
Logged Out
Sufficient Funds?
Yes
No
New Currency account created
Move money from current to foreign account
Commit transaction and record transaction
Prompt user of successful transaction

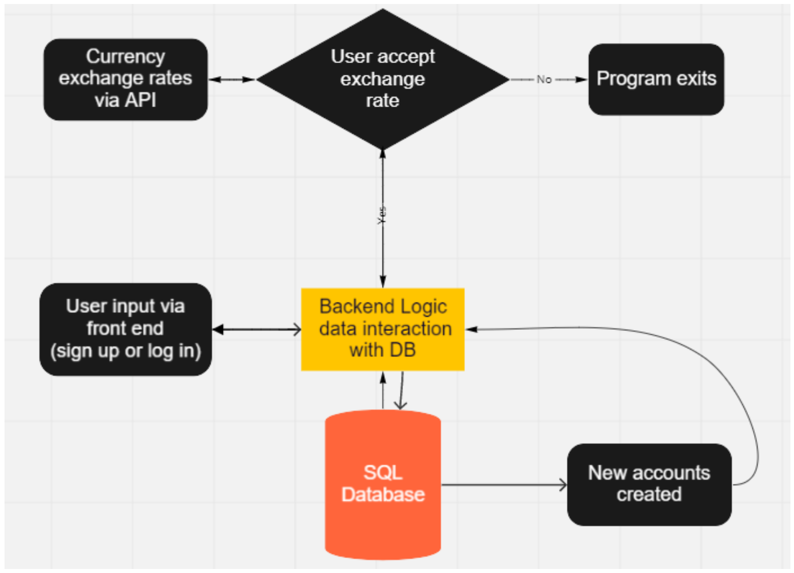
SPECIFICATIONS & DESIGN:

We spoke about designing our application with mobile-first design in mind, as this would likely be how users would prefer to interact with the product while travelling. However as we were using Flask, which is primarily used for web applications, we decided to incorporate a responsive design, meaning that it works across all devices. In terms of design, we wanted something that was simple and intuitive to use.

The following specifications were identified: user login with password verification, bank account balance check, transfer of monies from one account to another, linking accounts and user information, recording transactions, user acceptance of exchange rate and having a live API interaction.

Project deadline was one of our biggest considerations as well as the ability to link all components together. Therefore, back-end, front-end, API interaction and the database structure were all worked on simultaneously. This was done to ensure that all elements connected together properly, and so that we could make iterative changes throughout the building process of the web app in a timely manner. This proved essential later throughout the project as the tools we used were not sufficient to meet our specifications, this will be discussed in more detail later on.

Diagram 2: Project Workflow


Currency exchange rates via API
User accept exchange rate
No
Program exits
Yes
User input via front end (sign up or log in)
Backend Logic data interaction with DB
SQL Database
New accounts created

Our final project does take the majority of its shape from Diagram 2 above, which is our initial product workflow. We intended for the user input to be via flask forms which feed into our database, enabling users to convert their money into foreign currency using a combination of forms, API integration and Python classes and functions which again interact with the database. Working in an 'Agile' way was definitely beneficial as it allowed us to assess what was left within our backlog during each sprint process and prioritise the parts which were most essential to creating a minimum viable product. From this we were then able to again make prioritisation decisions on some of the additional features included within our backlog and decide not to go ahead with others.

IMPLEMENTATION AND EXECUTION

Agile methodology and team member roles:

An agile methodology was chosen for this project, as we knew our product development would be influenced by customer experience and therefore would be susceptible to change. Furthermore, we were aware that given our experience with these tools, we were not expecting a right first time execution so it was important we had a framework that would allow us to go back and revisit tasks. After mapping and fleshing out our initial concept on our Miro board during our sprint planning session, we then decided to structure how best to utilise the remaining time we had to work on our project. We split out each week into a separate sprint - the first involved looking at all the separate elements of functionality we would like to include - such as setting up the initial flask structure with app routes, integrating the foreign exchange API, building the initial database with dummy data, developing the bank account classes and functions, along with sketching out our initial design of what we would like the app to look like. These were all elements from our backlog which we had prioritised for sprint 1 as we recognised that these would all likely require iterations to be made later on.

The second sprint was focused around creating all the pages required for the app, building flask forms in both the front and back end, making changes to the iterations to the database in relation to the data which was being collected by the forms and implementing the initial front-end design using mostly HTML and CSS (we also utilised bootstrap and a tiny amount of Javascript), alongside iterating some of the elements from the first sprint which needed to be amended after reviewing these in our sprint retro. The third and final sprint was spent really honing in on any remaining elements which needed extra iterations and attention, most of which was centred around linking the front and back end - particularly linking the forms to the bank account functions and database. We also spent this time attempting to simplify our code to read better, along with testing various parts of the project to ensure everything worked the way we wanted it to, whilst also setting aside time to work on the project report.

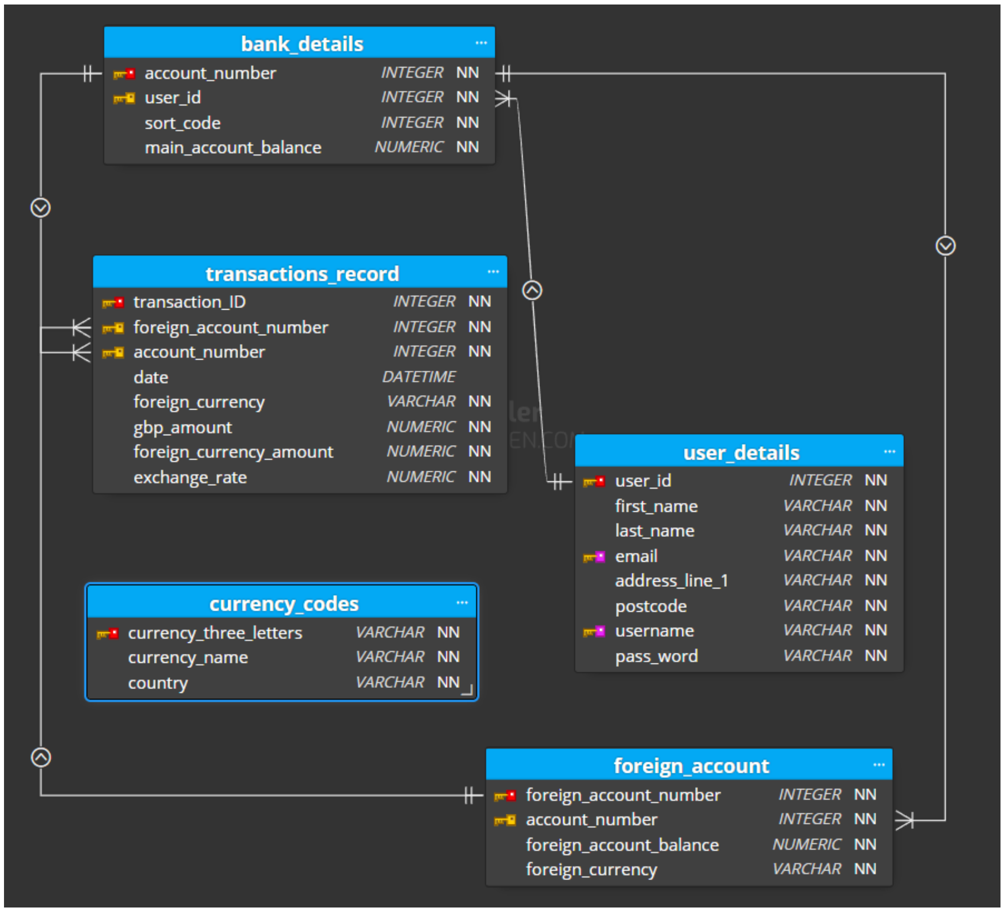
During our initial sprint planning session at the start of the project and sprint retros at the end of each week, we decided to delegate tasks for the following sprint based on a combination of each team member's strengths, their interest in delving more into learning about a specific or new concept, the capacity to take on tasks during each sprint (given that we all had additional commitments such as work taking up a significant portion of our time). Throughout the week we would then touch base on each other's progress and swap roles or decide to work together on parts based on what was remaining from our sprint objectives. Each sprint we also swapped roles so we had the opportunity to work across multiple parts of the project - database, API, bank account creation, Flask and design.

Developing our database:

The first iteration of our database only included four tables, however by the second sprint we realised this needed to be amended. Looking at existing banking platforms identified address and postcode fields as imperative for new user registration so these needed to be added into the user_details table. After having added these fields, our registration forms and our database are aligned closely and all the data our potential users would provide ends up being stored, ready to be analysed and interpreted.

When attempting to link the API to a user-input form, we found it would be a lot more efficient if the currency names and codes were being pulled from the database which led to the need to build a currency_codes table. We did discuss splitting our user details and login information however, this was later scrapped due to ease of importing information from the database. Our final database structure is down in Diagram 3 below.

Diagram 3: Database Structure

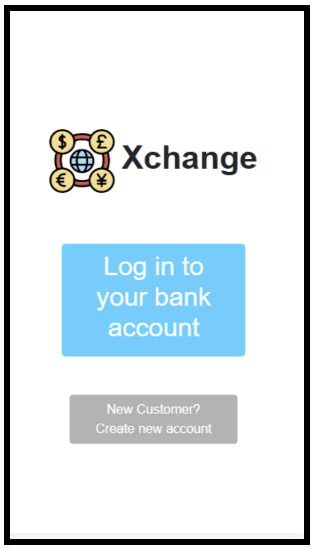

bank_details
account_number INTEGER NN
user_id INTEGER NN
sort_code INTEGER NN
main_account_balance NUMERIC NN
transactions_record
transaction_ID INTEGER NN
foreign_account_number INTEGER NN
account_number INTEGER NN
date DATETIME
foreign_currency VARCHAR NN
gbp_amount NUMERIC NN
foreign_currency_amount NUMERIC NN
exchange_rate NUMERIC NN
user_details
user_id INTEGER NN
first_name VARCHAR NN
last_name VARCHAR NN
email VARCHAR NN
address_line_1 VARCHAR NN
postcode VARCHAR NN
username VARCHAR NN
pass_word VARCHAR NN
currency_codes
currency_three_letters VARCHAR NN
currency_name VARCHAR NN
country VARCHAR NN
foreign_account
foreign_account_number INTEGER NN
account_number INTEGER NN
foreign_account_balance NUMERIC NN
foreign_currency VARCHAR NN

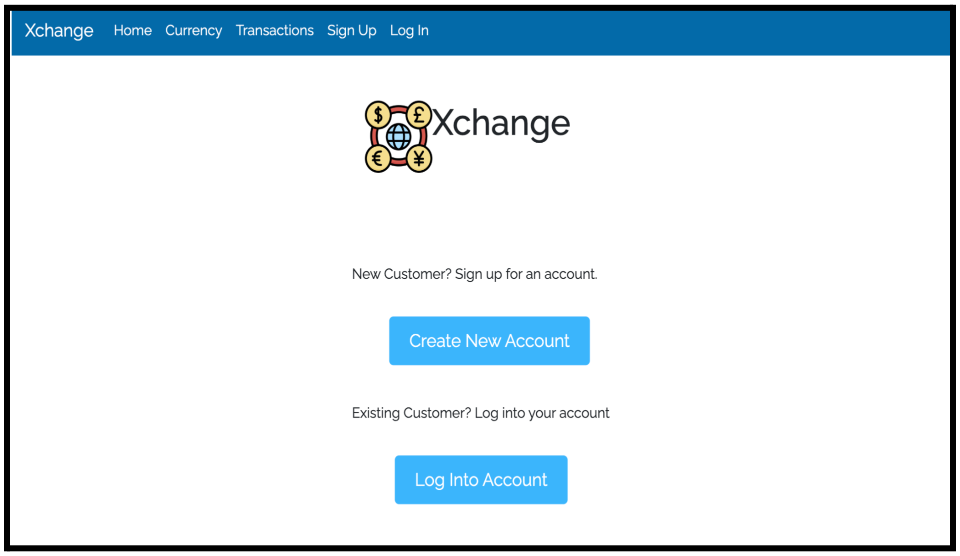
Design concepting vs implementation:

During the brainstorming and sprint planning period at the beginning of the project timeline, we had conducted some research into existing bank and foreign currency platforms to not only get a sense of their functionality but also how they looked in regard to aesthetics and design. We found many of these platforms were extremely simplistic in design and used colours such as navy blue, white and grey which helped us build our initial concept and brand colour palette.

As a group, we all started out with little to no background knowledge of front-end development whilst this was also not a concept we had previously covered in class. To begin with we had selected Tkinter to create the front-end element of Xchange with one team member working on the development of a Tkinter file for the currency conversion page, however towards the end of the first sprint we realised that this would not be the most efficient way to build the entire website - taking all the app routes which had been set up for multiple pages into account - we decided as a team we would switch to HTML and CSS for our front-end development. Consequently, multiple team members pitched in to meet our sprint objective of styling the front end homepage and other html pages by the end of the sprint, making tweaks to look more user-friendly through different iterations and learning new skills as we went along, this design is shown in Diagram 4 below.

Diagram 4: Initial Homepage build vs iterative execution:





TESTING & EVALUATION:

Given that our app focussed strongly on user experience and required frontend and backend connection, testing throughout our project was essential to its success.

Working collaboratively whilst using Git, whenever a new pull request was made, changes were reviewed by other team members as part of regression testing. This was especially important for us to learn what one another was doing as well as being key to ensuring function integrity any time changes were made. Unit tests were also performed on a number of our functions as well as the API class function, this was done to verify the functions in isolation were running as planned when imported into our app.

As previously mentioned, parts of the program were initially worked on in isolation simultaneously before integrating them together. Therefore, it was natural that when integrations were made integration testing was executed. This involved testing whether the existing functions still performed as expected once and no new and unwanted errors were introduced. Should there have been an issue with integration this was assessed as a team in order to prioritise next steps. This testing proved to be our most important, as it identified the biggest hurdle faced in the project: getting to integrate Flask with mySQL connector. Trying to integrate the two interfaces proved troublesome and identified that the connection manager, mySQL connector, would not be sufficient for our programme and we were compelled to change to using SQLAlchemy.

Without the ability to do User Acceptance Testing, system testing was our final form of testing. Thorough testing was performed by the team to ensure the system operated to the right standard. With a number of exception handling scripts and validation criteria throughout the application, rigorous system testing was essential. We are simulating a banking app so we needed to be certain no data was committed to the database incorrectly.

CONCLUSION:

<u>What could we do next? :</u>

Thinking about the project retrospectively, as a group we have considered additional features and functionality we would have liked to incorporate if we had more time. These include allowing users to update their password and personal details which in turn would update in the database, allowing functionality for mobile (possibly using SWIFT to create an additional iPhone app - since Flask is significantly for web-applications), incorporating graphs using data visualisation tools such as Tableau or Pandas, allowing users to have their current accounts (base currency for the conversions) in currencies other than GBP and allowing users to update the initial value of their accounts (adding in more funds). Additionally, we would have liked to include more interactive front-end functionality - as members of the team picked up basic skills in front-end design throughout the course of the project, whilst we also would have liked to consider usability factors within the design process - thinking about the overall user-experience such as choosing accessible fonts and colours to ensure Xchange could be used by all potential users.

<u>Key Learnings :</u>

The aspect we found the most difficult was the process of joining up all of our individual elements, particularly linking the back end to the front end. We struggled with this from early on, specifically with our user forms interacting with the database, and tried many different approaches to resolve this issue. With one week before the project deadline, we learnt that the way we had joined the Python and SQL elements using mySQL Connector would not work on our login form, as it did not allow data to be fetched from the DB and passed through for verification. Due to this, our database had to be rewritten using SQLAlchemy and working with SQLite, and our overall files restructured taking this change into account. Amazingly we worked well as a team within the space of a weekend to rebuild our project, along with getting the forms to interact with the newly built database and adjusted routes and functions, meaning we still had a week left for testing and final tweaks to our code. Unfortunately, due to this setback we had less time to work on certain “nice to have'' features that we had been hoping to include, such as a front-end web page showing the user’s transaction history, and displaying their various foreign currency accounts.

We are also proud that another of our initial stumbling blocks which we found challenging- being version control via git, actually turned into a team strength through practice uploading and merging versions of our code into our shared repository.

Ending on a positive note, some of the things which we felt went extremely well for us included maintaining agile methodology throughout the entirety of the project - as a team we worked iteratively to ensure we were spending our time efficiently based on sprint objectives and provided feedback on each other's work in a productive and encouraging format. On the non-technical side, communication was also key to the success of our group, keeping up with our daily sprints and slack messages to check in on each other's progress and help each other out with any issues we were facing. We also feel like we have a much stronger handle of utilising APIs, classes and functions, flask structures with app routes and database integration within a real-life scenario which could potentially be similar to what we may get the opportunity to work on at NatWest. Overall, the project was a success as we have a functioning app that satisfies our key requirements.

REFERENCES:

1) Miro Board used for planning out activity:
   https://miro.com/app/board/uXjVOn84ZXw=/?share_link_id=364853334485
2) Fundera, Cash vs Credit Card Spending Statistics (2021)
   https://www.fundera.com/resources/cash-vs-credit-card-spending-statistics
3) Exchange Rate API Documentation
   https://www.exchangerate-api.com/docs/overview