FEMINSTANT

PROJECT MEMBERS : AMMAARAH, AMINA, FAHMIDA, NANA AKUA

This project report will discuss the approach by which our CFG software specialism team undertook the project. It is split up into the following sections:

CONTENTS

AIM : AS PART OF THE CFG SOFTWARE SPECIALISM OUR PROJECT AIMED TO DEVELOP AN E-COMMERCE WEB APPLICATION TARGETED AT A FEMALE POPULATION TO INSTANTLY SHOP FOR FEMININE ESSENTIALS (COSMETICS, MAKEUP, FEMININE HYGEINE).

Focusing on the London area, our app concept aimed to cater to the needs of females to be able to easily browse, purchase and instantly receive their items within a short time frame straight to their door.

Our project focused on women who may have run out of their favourite ite mascara right before an event or require sanitary products late at night when most stores are closed.

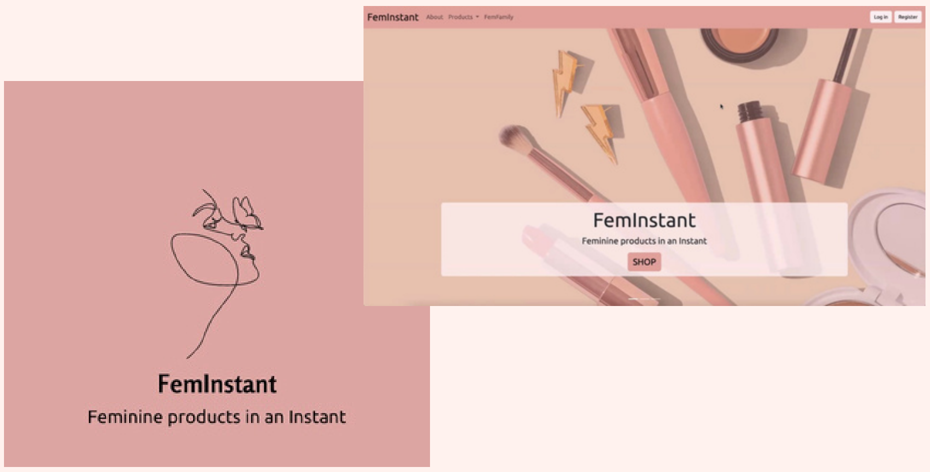
We believe there is a gap in the market for a concise marketplace that focuses solely on providing female products under one convenient instant delivery app - thus creating the opportunity for FemInstant to fill this gap.

2. Background and Concept

Looking at similar competitor markets such as BeautyBay, Getir and Amazon, Flo and Deliveroo. We then performed an Insight analysis.

We observed that each e-commerce type platform catered to the concept of bringing many diverse products under one hood, but few integrated the concept of feminine hygiene products, and feminine essentials as a sole focus, as well as the more commonly used skincare brands, with some retailers catering either heavily towards luxury beauty, or not having a more ordinary essential range. We noted that few offered same day delivery services, and whereas Getir App best demonstrated a same day delivery concept of under 2 hours, Getir lacked cosmetic / skincare range altogether, thus demonstrating a clear gap and opportunity for our product.

We knew that our application would require research into what a user would expect out of a beauty / skincare e-commerce website so we asked amongst our network of friends and family, what sort of expectations or wishes they would typically have out of such a service. We used the data gathered from here as well as exploring popular and highly sought essential skincare / makeup products as well as feminine sanitary products to allow us to develop a small portfolio of products to input into our database and thereby present within our product catalogue



Feminstant therefore aims at being an e-commerce website selling makeup, skincare and female hygiene products. The delivery service would initially beta-launch exclusively in London.
Feminstant would operate utilising delivery drivers to dispatch the customer placed orders from a warehouse. The concept of the website is that the customer can order whichever item(s) they require, the system will then search to see if the product is available in the closest warehouse to which the customer requires the item(s) to be delivered to. If the item(s) is available, a driver will collect the item and deliver it straight to the address.

The customer will be given an estimated time of arrival and would also be able to track the status of the delivery. The customer will have the option to pay via debit / credit card payment.

3. SPECIFICATION AND DESIGN

3.1 TECHNICAL AND NON-TECHNICAL REQUIREMENTS

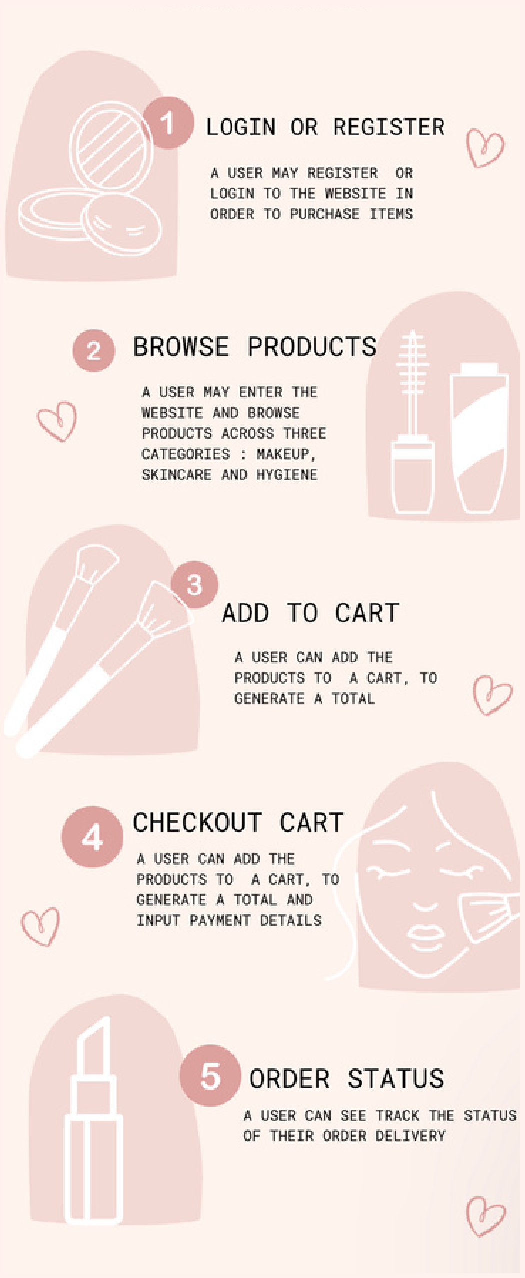
ESSENTIAL REQUIREMENTS:

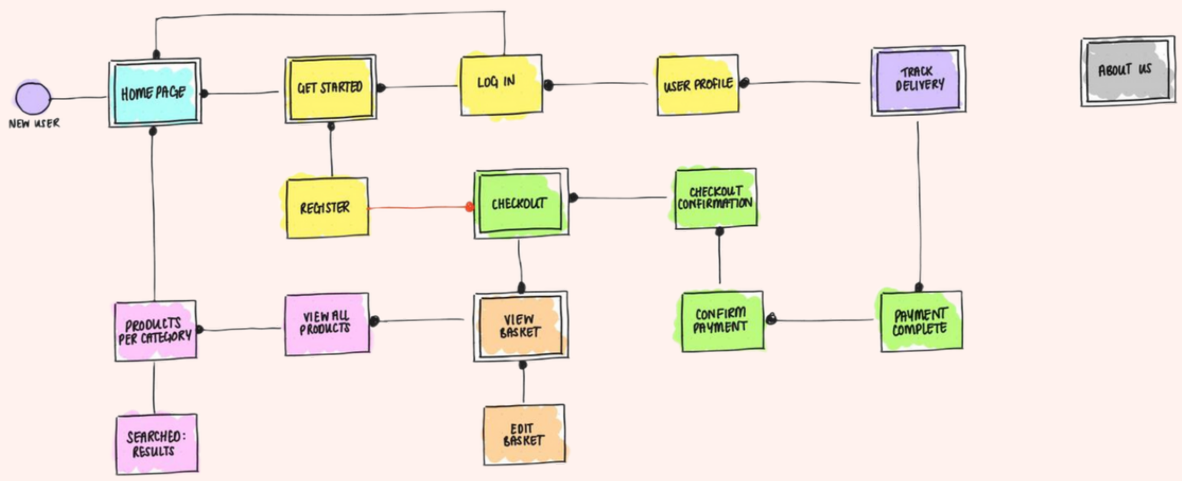
- **A homepage that presents the website and has a navbar that is clearly dispayed**
- **About page that gives a rough overview of the scope of the Feminstant app**
- **Product page which contains products in an overall directory**
- **Users should find the user experience to be easily accessible and visually pleasing**
- **Users should be able to retrieve information about product including the name, product description.**
- **Users should be able to add items and alter quantity of items from product directory to cart**
- **Users should be able to register to the website by inputting personal information**
- **Users should be able to login once an account has been created**
- **Users should be able to checkout a basket without registering**
- **Payments should be secure**
- **The program should be able to pass queries to the database and also retrieve information from the database**
- **Users should be able to use a functional frontend.**

DESIRABLE REQUIREMENTS:

- **Users should be able to filter the products page using a drop-down option**
- **Users could pay through a payment methods Stripe API**
- **Existing users that have an account may be able to recieve a discount upon purchase of items**
- **The customer may be able to see which delivery driver is closest to them based on their postcode**
- **Users may input their postcode to auto fill the borough and street name into the form by utilising a postcodes API which can retrieve the street name and borough**

To help us see the bigger picture of our application, we created a functions infographic which clearly displays the functions in a non-static way, enabling us to concisely present the features and have a visual to work from prior to development. This fed into our user flow designing process and our trello board planning stages.


1 LOGIN OR REGISTER
A USER MAY REGISTER OR LOGIN TO THE WEBSITE IN ORDER TO PURCHASE ITEMS
2 BROWSE PRODUCTS
A USER MAY ENTER THE WEBSITE AND BROWSE PRODUCTS ACROSS THREE CATEGORIES : MAKEUP, SKINCARE AND HYGIENE
3 ADD TO CART
A USER CAN ADD THE PRODUCTS TO A CART, TO GENERATE A TOTAL
4 CHECKOUT CART
A USER CAN ADD THE PRODUCTS TO A CART, TO GENERATE A TOTAL AND INPUT PAYMENT DETAILS
5 ORDER STATUS
A USER CAN SEE TRACK THE STATUS OF THEIR ORDER DELIVERY



To help us see the bigger picture of our product, we designed a user-flow to demonstrate how the individual components of our website interact together in a non-static way. The diagram illustrates a user's journey from the stage of Onboarding and Authentication, to Payment & Checkout. By mapping out all of the possible paths, behaviours and edge cases (such as error states and success states), we were able to develop an app which provides our user with the most effective user experience.

When drafting our user flow and laying the foundations of our website, we asked ourselves three key questions:

1. Who is our user?
2. What is their goal?
3. How might our user progress through our website to achieve their goal?

After gathering these insights, we narrowed down our findings into hypothetical user stories; eventually defining our problem statement, and forming our question for design: "How might we make the process of an on-demand service quicker, easier and more efficient for users?". We aimed to consolidate this by utilising a Trello board which would highlight the vision for the functions we would like to incorporate in our application.

Our first approach was to ascertain our users' call to action which, put simply, is: to make an instant purchase of beauty, hygiene or skincare products. With this in mind, we decided that the best landing page for our site would be the 'Products' page. The prevailing goal of this decision was to place the user right in the centre of their need as soon as they land on our website. From here, users are able to view a catalogue of our products using our carousel feature or navigation bar.

Initially, we restricted the user flow at the point of purchase by directing users to log in or create an account after adding items to their basket, and before proceeding to checkout. We eventually iterated our design by eliminating the immediate Onboarding and Authentication process as found in the design architecture of many of our competitors. By doing this, users are able to seamlessly progress to checkout without any in-app restrictions, such as creating an account or logging in. These validations are made accessible to a user at any stage within their journey, thus allowing them to explore deeply into our website before deciding to create an account or log in. Upon checkout, users are able to view the quantity, description, and total cost of their items before confirming payment, thus adhering to the UX principles of clarity, ease and user-focused products.

Our user-flow was pivotal in showing the series of actions that a user will take to navigate their way through our product and achieve their goal, i.e. choosing a product and making a payment. We leaned heavily on user-centered design thinking by understanding how the user may experience our product, specifying their needs, and configuring solutions to address their pain points.

4. IMPLEMENTATION AND EXECUTION

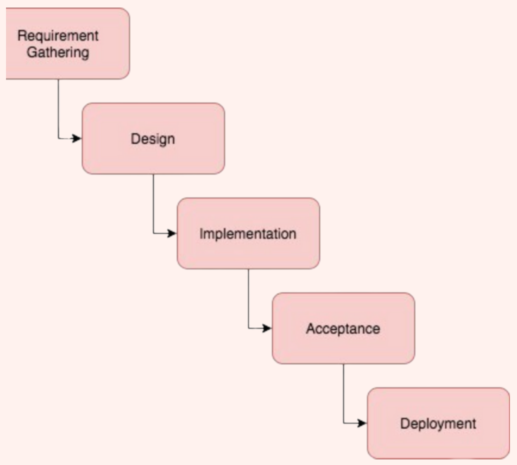
As a team we appointed Fahmida and Ammaarah to be our team leaders ensuring we got together to have weekly meetings to form a sense of direction and bring our visition to life and to ensure we all have our own appointed tasks. We conducted a swot analysis to analyse each member's strength and weaknesses and allocate tasks accordingly as well as opportunities and threats we may find whilst building our programme. After discussing strengths and weaknesses of each team member the following tasks were allocated:

Our Team worked in an Agile development manner, setting ourselves short “sprints” with a goal of implementing a few features at each sprint review, testing these before deploying and ensuring that we continuously undertook test driven development, merging any changes to a development branch and retaining a production environment to compare along the process As each feature was tested. Towards the end of our project lifecycle we worked using one team members code editor and collaboratively worked through any remaining issues (such as making a functional checkout page, and integrating API)

TEAM ROLES

- Ammaarah focused on SQL and databases, connecting database, creation of basket, and researching API’s to be included in to the program.
- Fahmida - Writing the backend python code e.g flask, front end design, creating pages, checkout and basket work as well as implementing Stripe API.
- Nana - User wireframes, user research, exception handling, unit testing as well as working on python code and API research for tracking and checkout.
- Amina - Project report and presentation writing, python unit testing, assisting Ammaarah in creation of python-SQL Alchemy database, and initial mock front end.

AGILE APPROACH



(1)

4.1 THREATS AND OPPORTUNITIES

We found that initially no members were comfortable with flask.
Unsure of how to connect the front-end with the back-end.
Due to other commitments finding the time to collaborate more often.

In week 2 we were faced with major obstacles with two members of our group bereaving the loss of family members, shortly followed by another member of the group being hospitalised for several days, resulting in an overall drop in work rate altogether and setting us back significantly

We aimed to overcame these issues towards the end of the project by seeking assistance from Emily (Instructor) and alsoutilising YouTube tutorials, collaborating and test driven development To trial and error different components until functionality was achieved

We also created a Whatsapp group and had daily scrum meetings over Whatsapp. In addition to this we had weekly meetings on Google Meet. We also collaborated occasionally during the weekdays

4.2 IMPLEMENTATION CHALLENGES

- Amongst challenges that we faced initially was integrating our python code with MySQL database, in a manner that would enable the python to alter the database. Researching this issue allowed us to take advantage of Flask-SQLAlchemy which provides the Object Relational Mapper (ORM) which allows the queries to be written using simple python objects / methods - thus being a far more efficient approach at building our database.

- Having limited knowledge of FRONTEND meant that many of us attempted to render several versions of a FRONTEND UI for our website in order to actually test drive the development. many of our functions would not be clearly visible without utilising html and css, such as form data input and adding products to a basket, in order to clearly visualise these elements and thereby simulate user experience in order to test our application, we each spent a significant time also self-teaching html and css / Bootstrap
- We wanted to implement a discount feature for members, whereby any member whowas signed up to the website with an account, would receive a 5% discount on checkout, however this was not feasible due to time consrraints

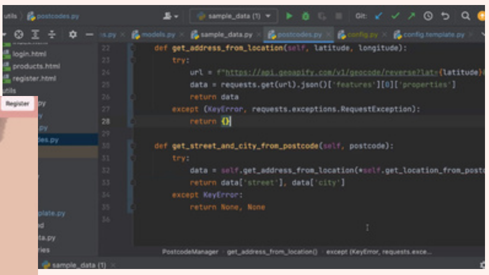
- COOKIES were a concept that we had not fully grasped nor known how to take advantage of, later understanding this under the guidance of our instructor enabled us to enhance functionality of the basket within our application

- FRONTEND BOOTSTRAP helped us to bridge some gaps in frontend knowledge and streamline HTML/CSS elements and integrate these into our base.html

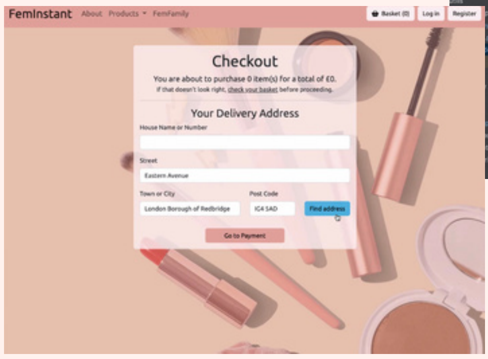
- WTFORMS helped us further enable us to render the templates for login and registration forms which otherwise would've proved difficult to code from limited front end knowledge, as well as handle some validations using in built validators, which otherwise building from scratch would be somewhat challenging.

- DATA SECURITY - ensuring data security was a challenging task, we moved our private keys into config.py and then utilized the git ignore command so that only we would be able to protect secret keys

- API - finding accessible free APIs was challenging and limited our options. Though we later tried to integrate Stripes Payment API. We also integrated a postcodes api from Geoapify to retrieve the address details from postcode entry . We secured this process by ensuring our code removed the address after checkout has occurred for the user.



```
    def get_address_from_location(self, latitude, longitude):
        try:
            url = f"https://api.geoapify.com/v1/geocode/reverse?lat={latitude}
            data = requests.get(url).json()['features'][0]['properties']
            return data
        except (KeyError, requests.exceptions.RequestException):
            return {}

    def get_street_and_city_from_postcode(self, postcode):
        try:
            data = self.get_address_from_location(*self.get_location_from_postc
            return data['street'], data['city']
        except KeyError:
            return None, None
```



Checkout

You are about to purchase 0 item(s) for a total of £0.
If that doesn't look right, check your basket before proceeding.

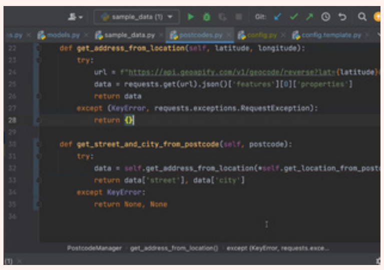
Your Delivery Address

House Name or Number:

Street: Eastern Avenue

Town or City: London Borough of Redbridge

Post Code: IG4 5AD



```
    def get_address_from_location(self, latitude, longitude):
        try:
            url = f"https://api.geoapify.com/v1/geocode/reverse?lat={latitude}
            data = requests.get(url).json()['features'][0]['properties']
            return data
        except (KeyError, requests.exceptions.RequestException):
            return {}

    def get_street_and_city_from_postcode(self, postcode):
        try:
            data = self.get_address_from_location(*self.get_location_from_post
            return data['street'], data['city']
        except KeyError:
            return None, None
```

5. DESIGN AND ARCHITECTURE

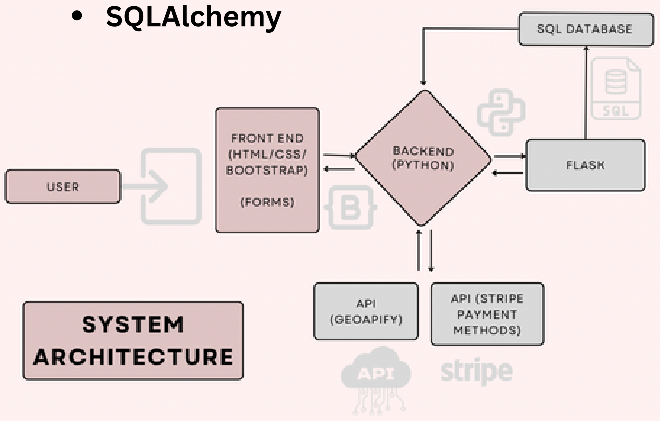
5.1 TOOLS AND LIBRARIES

COMMUNICATION
- **Slack**
- **Whatsapp Chat**
- **Google Meets/Zoom**

BACKEND
- **Python**
- **WTForms**
- **Flask**
- **Flask-Bcrypt**
- **PyMySQL**
- **Flask-SQLAlchemy**
- **SQLAlchemy**

API
- **Stripe**

FRONTEND
- **Bootstrap**
- **Jinja**



SYSTEM ARCHITECTURE

We used Google/Meets and Zoom for screenshare collaborative meetings and regular update messages were sent via Whatsapp and Slack chat

Flask is considered the best framework for lightweight web applications, and as it enables the same user interface to be used for numerous pages, it seemed that Flask was the most appropriate choice for a shopping website such as Feminstant.

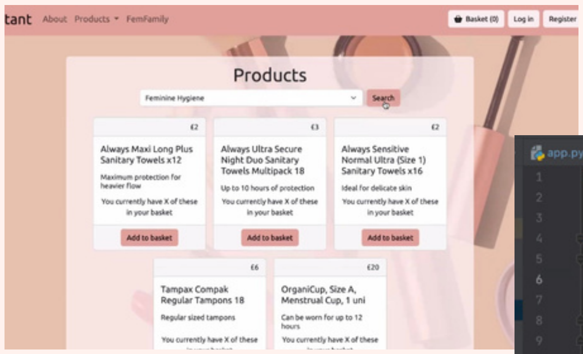
PyMySQL is advantageous in that it is platform independent, and the mySQL server protocol is implemented from Python alone.

BCrypt was implemented to ensure hashing and increase encryption for our application.

WTForms is flexible forms rendering and validation library which enabled us to efficiently include forms within our application

Bootstrap and Jinja were utilized to bring our front end together enabling us to efficiently use template inheritance to form a base, and then child html files.

API Geoapify was used to generate the users Street name and borough based on postcode entry in checkout form



Products

Feminine Hygiene

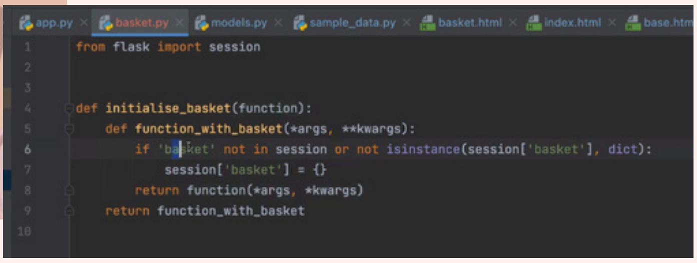
€2
Always Maxi Long Plus Sanitary Towels x12
Maximum protection for heavier flow
You currently have X of these in your basket

€3
Always Ultra Secure Night Duo Sanitary Towels Multipack 18
Up to 10 hours of protection
You currently have X of these in your basket

€2
Always Sensitive Normal Ultra (Size 1) Sanitary Towels x16
Ideal for delicate skin
You currently have X of these in your basket

€6
Tampax Compak Regular Tampons 18
Regular sized tampons
You currently have X of these

€20
OrganiCup, Size A, Menstrual Cup, 1 uni
Can be worn for up to 12 hours
You currently have X of these



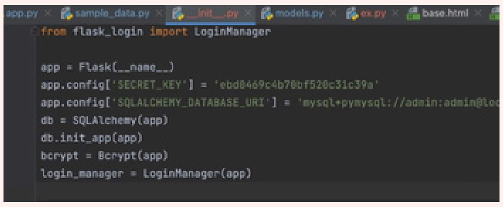
```python
from flask import session


def initialise_basket(function):
    def function_with_basket(*args, **kwargs):
        if 'basket' not in session or not isinstance(session['basket'], dict):
            session['basket'] = {}
        return function(*args, *kwargs)
    return function_with_basket
```

In order to enable any user to checkout a basket - we ensured that in our base.html the basket is accessible by all users (whether registered or not) so that a user may checkout without registering if they should chose to. For every page within our app.py file we can then use a decorator and we must import from basket file to ensure the basket is present on every page.

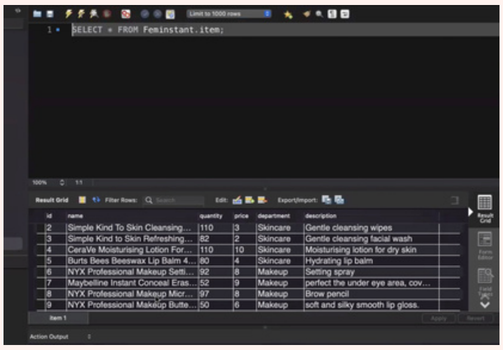
Flask provides you with a session each time and if we utilize cookies which retain the memory from that session, we can restore the users basket using this - and restore the items from its ID, thereby enabling the whole item to be restored and its quantity restored. Even if the user refreshes, this should ensure the Python would not be overwritten.

5. DESIGN AND ARCHITECHTURE

SQLALCHEMY DATABASE



```
from flask_login import LoginManager

app = Flask(__name__)
app.config['SECRET_KEY'] = 'ebd0469c4b70bf520c31c39a'
app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://admin:admin@loc
db = SQLAlchemy(app)
db.init_app(app)
bcrypt = Bcrypt(app)
login_manager = LoginManager(app)
```



```
SELECT * FROM Feminstant.item;
```

id	name	quantity	price	department	description
2	Simple Kind To Skin Cleansing...	110	3	Skincare	Gentle cleansing wipes
3	Simple Kind to Skin Refreshing...	82	2	Skincare	Gentle cleansing facial wash
4	CeraVe Moisturising Lotion For...	110	10	Skincare	Moisturising lotion for dry skin
5	Burts Bees Beeswax Lip Balm 4...	80	4	Skincare	Hydrating lip balm
6	NYX Professional Makeup Setti...	92	8	Makeup	Setting spray
7	Maybelline Instant Conceal Eras...	52	9	Makeup	perfect the under eye area, cov...
8	NYX Professional Makeup Micr...	97	8	Makeup	Brow pencil
9	NYX Professional Makeup Butte...	50	6	Makeup	soft and silky smooth lip gloss.

Flask was our choice of web framework as it is able to use open source toolkits such as SQLAlchemy, it is flexible, scaleable lightweight and easy to build APIS with.
All the private keys for our database were secured by storing in config.py.

Our @app.route decorators would bind our routes to the functions - then mapping to the specific page associated with the products page information.

To begin with we wrote our databsee file and set up a database connection by urilising flask-sql alchemy in conjunction with our app.py and config.py to generate our database for FemInstant. The database contained tables for items, customers, employees.

Utilising Flask-SQLAlchemy enabled us to run function-based query constructions enabling us to work with our database to modify it or retrieve information from our sql Database. The advantages of which is that Flask-SQLAlchemy has its own API. This adds complexity by having its different methods for ORM queries and models

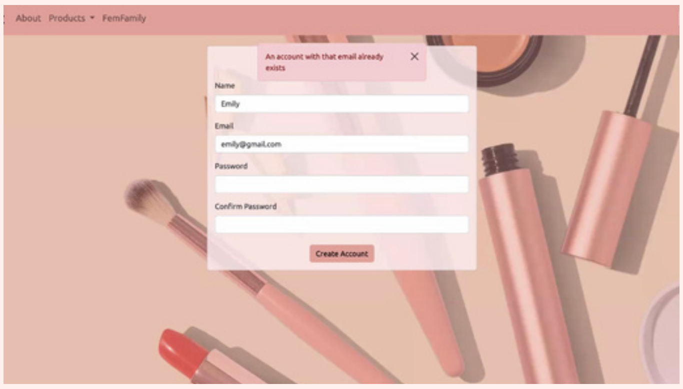
REGISTER / LOGIN

WTForms- was used to create a form class, which gave us a registration form object, which subsequently passed on to our template.

Form class, containing username, email, password, and confirm password fields.
 Using form.validate_on_submit():. This essentially states 'if the request method is POST and if the form field(s) are valid, then the user can proceed and If input passes our validation criteria, on the home page a Welcome “User” button will appear with the user's name.

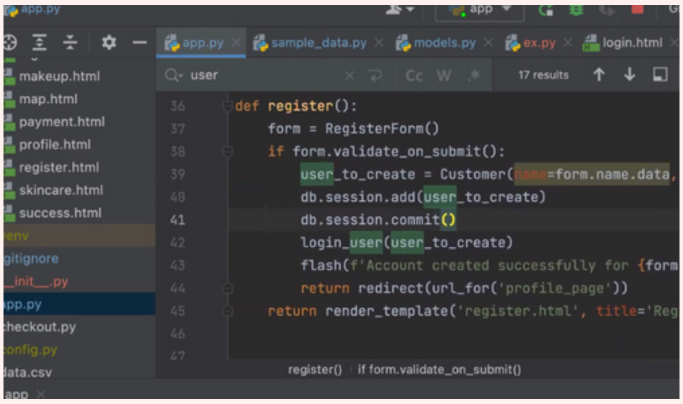
The SECRET_KEY is commonly used for encryption with database connections and browser sessions.

Upon input of information into this registration form the customer would then be signed up, with an account entry that inputs into our Customers table on the backend, and on the front end, the customer would be able to
 access the Welcome profile page.



An account with that email already exists

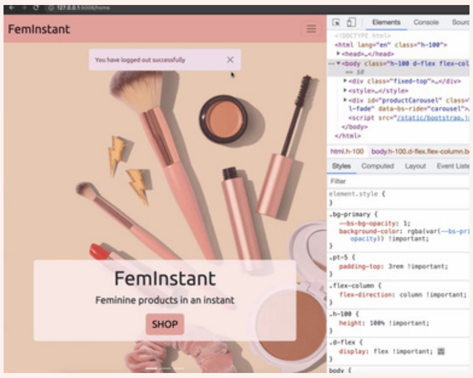
Name: Emily
Email: emily@gmail.com
Password:
Confirm Password:



```
def register():
    form = RegisterForm()
    if form.validate_on_submit():
        user_to_create = Customer(name=form.name.data,
        db.session.add(user_to_create)
        db.session.commit()
        login_user(user_to_create)
        flash(f'Account created successfully for {form
        return redirect(url_for('profile_page'))
    return render_template('register.html', title='Reg
```

5. DESIGN AND ARCHITECHTURE

FRONT END ARCHITECTURE CSS BOOTSTRAP

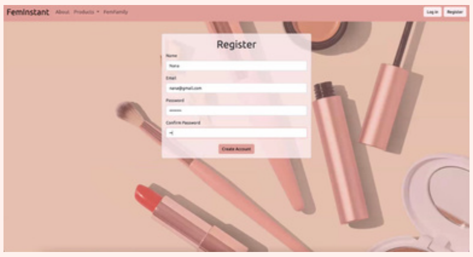


```
<!DOCTYPE html>
<html lang="en" class="h-100">
 ▸<head>…</head>
…▾<body class="h-100 d-flex flex-col
   == $0
   ▸<div class="fixed-top">…</div>
   ▸<style>…</style>
   ▸<div id="productCarousel" class="
    l-fade" data-bs-ride="carousel">
    <script src="/static/bootstrap.j
  </body>
</html>
```

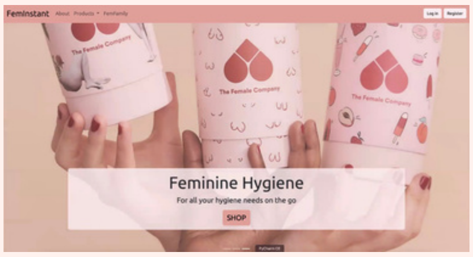
```
element.style {
}
.bg-primary {
  --bs-bg-opacity: 1;
  background-color: rgba(var(--bs-pri
    opacity)) !important;
}
.pt-5 {
  padding-top: 3rem !important;
}
.flex-column {
  flex-direction: column !important;
}
.h-100 {
  height: 100% !important;
}
.d-flex {
  display: flex !important;
}
body {
```

Bootstrap is a popular CSS framework that provides several pre-built CSS classes, which we utilised by incorporating into our HTML code in order to help create the navbar for our page. (2)

Our website follows a theme and has similar elements (like a navbar) on every page. We utilised template inheritance by creating a new HTML document called "base.html". This will contain all of the HTML code that will be present throughout our application.

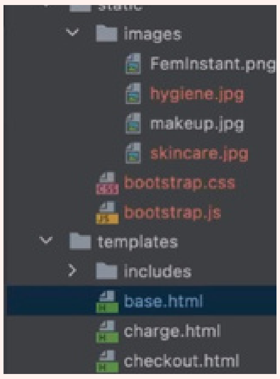


Within each of our webpages we created blocks to make use of Jinja template inheritance - enabling certain components of the base.html to be overwritten by blocks depending on which child template is used - we did this using {% extends "base.html" %}{% block title %}Home Page{% endblock %}{% block content %}<h1>Test</h1>{% endblock %} tags in the beginning of our child files.



$light: #FFEFF1 ⬜;

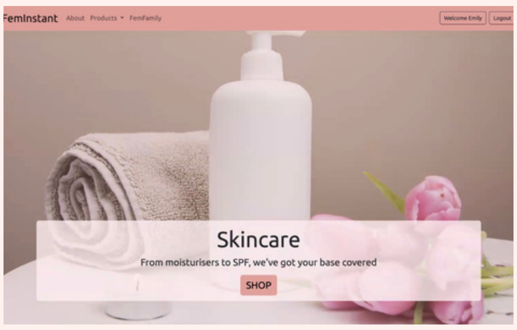
$primary: #EE9D98 🟥;
$secondary: #B2675A 🟫;
$success: #00B2E3 🟦;
$info: #BA5B84 🟪;
$warning: #FAB86B 🟧;
$danger: #E30413 🟥;



```
  ⌄ images
      FemInstant.png
      hygiene.jpg
      makeup.jpg
      skincare.jpg
    bootstrap.css
    bootstrap.js
⌄ templates
  › includes
    base.html
    charge.html
    checkout.html
```

An advantage of Flask-Bootstrap in this context is in rendering of forms. Instead of having to style the form fields one by one, Flask-Bootstrap utilises a macro whixh accepts a Flask-WTF form object as an argument and renders the complete form using Bootstrap styles.

We chose to utilise a soft theme with a font “Ubuntu”, and a color palaette as shown below (right):



We believe the use ofbeauty aeesthetic imagery, along with a complimentary palaette of colors help to not only streamline the website according to a theme, but also entice the customer to browse each section and easily access the various features, overall providing a pleasant user experience

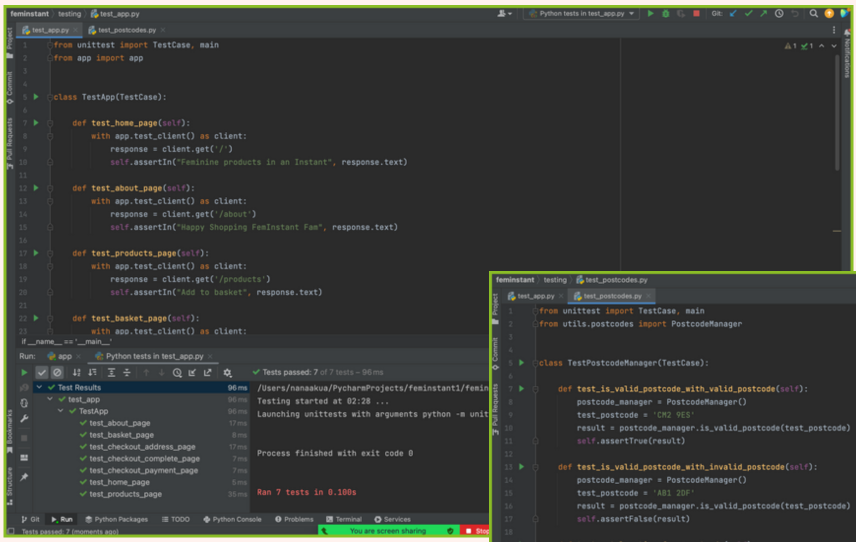
6. TESTING

USER TESTING

- Testing the User Interface and Experience was a crucial component of developing our application, so we simulated acceptance tests to test against the technical requirements as outlined in Section 3.
- We ensured that our user Tests were able to navigate through every section of our website, from browsing products through to adding these to cart, checking out as well as signing up / registering and logging out.
- Through feedback obtained from our simulated acceptance testing we were able to drive continuous improvement into our working approach and thereby improve the quality of the overall user experience and interface

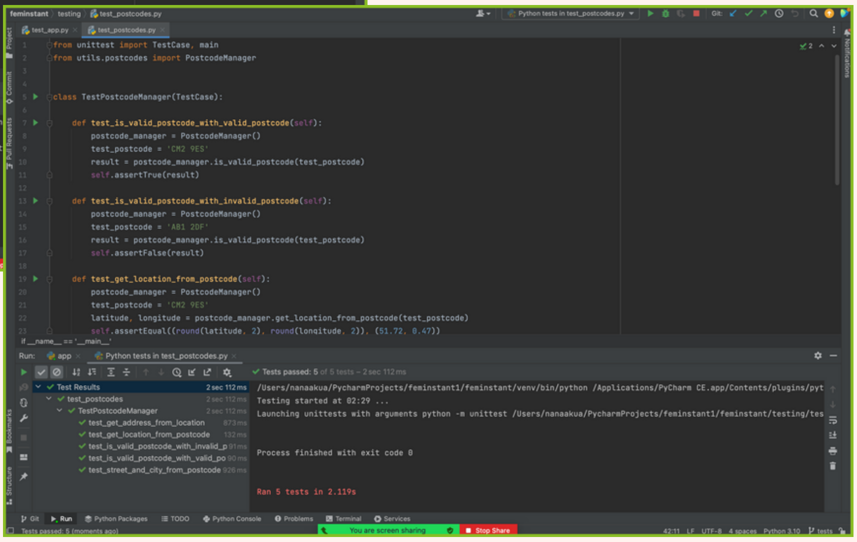
UNIT TESTING

- Unit tests were written whilst developing the code to ensure functions such as app routing were working and that items could be added or dropped from the database using a mock database.



```python
from unittest import TestCase, main
from app import app


class TestApp(TestCase):

    def test_home_page(self):
        with app.test_client() as client:
            response = client.get('/')
            self.assertIn("Feminine products in an Instant", response.text)

    def test_about_page(self):
        with app.test_client() as client:
            response = client.get('/about')
            self.assertIn("Happy Shopping FemInstant Fam", response.text)

    def test_products_page(self):
        with app.test_client() as client:
            response = client.get('/products')
            self.assertIn("Add to basket", response.text)

    def test_basket_page(self):
        with app.test_client() as client:
```

```
Tests passed: 7 of 7 tests – 96 ms

Test Results                      96 ms
  test_app                        96 ms
    TestApp                       96 ms
      test_about_page             17 ms
      test_basket_page             8 ms
      test_checkout_address_page  17 ms
      test_checkout_complete_page  7 ms
      test_checkout_payment_page   7 ms
      test_home_page               5 ms
      test_products_page          35 ms

/Users/nanaakua/PycharmProjects/feminstant1/femin
Testing started at 02:28 ...
Launching unittests with arguments python -m unit


Process finished with exit code 0


Ran 7 tests in 0.100s
```



```python
from unittest import TestCase, main
from utils.postcodes import PostcodeManager


class TestPostcodeManager(TestCase):

    def test_is_valid_postcode_with_valid_postcode(self):
        postcode_manager = PostcodeManager()
        test_postcode = 'CM2 9ES'
        result = postcode_manager.is_valid_postcode(test_postcode)
        self.assertTrue(result)

    def test_is_valid_postcode_with_invalid_postcode(self):
        postcode_manager = PostcodeManager()
        test_postcode = 'AB1 2DF'
        result = postcode_manager.is_valid_postcode(test_postcode)
        self.assertFalse(result)

    def test_get_location_from_postcode(self):
        postcode_manager = PostcodeManager()
        test_postcode = 'CM2 9ES'
        latitude, longitude = postcode_manager.get_location_from_postcode(test_postcode)
        self.assertEqual((round(latitude, 2), round(longitude, 2)), (51.72, 0.47))
```

```
Tests passed: 5 of 5 tests – 2 sec 112 ms

Test Results                                  2 sec 112 ms
  test_postcodes                              2 sec 112 ms
    TestPostcodeManager                       2 sec 112 ms
      test_get_address_from_location               873 ms
      test_get_location_from_postcode              132 ms
      test_is_valid_postcode_with_invalid_p         91 ms
      test_is_valid_postcode_with_valid_po          90 ms
      test_street_and_city_from_postcode           926 ms

/Users/nanaakua/PycharmProjects/feminstant1/feminstant/venv/bin/python /Applications/PyCharm CE.app/Contents/plugins/pyt
Testing started at 02:29 ...
Launching unittests with arguments python -m unittest /Users/nanaakua/PycharmProjects/feminstant1/feminstant/testing/tes


Process finished with exit code 0


Ran 5 tests in 2.119s
```

feminstant › testing › test_postcodes.py
test_app.py
test_postcodes.py
if __name__ == '__main__'
Run: app
Python tests in test_postcodes.py
Git
Run
Python Packages
TODO
Python Console
Problems
Terminal
Services
Tests passed: 5 (moments ago)
You are screen sharing
Stop Share
42:11 LF UTF-8 4 spaces Python 3.10 tests

CONCLUSIONS

For our software specialism project we were asked to create a programme which included OOP's, with effective algorithms and a working API.

We were provided a time allowance of 4 weeks to bring this project to completion and we attempted to adhere to agile working ways through daily scrum which enhanced teamwork and understanding tasks being tackled by each member of the group. Though at times we implemented more collaborative approaches to work, conflict between personal and professional schedules and commitments to external jobs which limited us in terms of time.

Nonetheless, we distributed tasks in accordance to individual strengths and regularly convened with Zoom calls, WhatsApp and slack messages. We then essentially followed through the process of developing this application as displayed by the diagram above

Although we faced challenges in incorporating every feature that we initially intended to include; (due to issues as outlined under Opportunities and Threats (section 4)), we still managed to create a functional and interactive e-commerce platform displaying the basic requirements of our overall vision for our FemInstant application.

One main limitation we faced was finding a working API to do live order tracking and thereby this feature was not implemented. We discovered quite late into the process that many of the APIs were not accessible or free to use, and this meant which later hindered us to some extent. In future, we would overcome this challenge by dedicating more API research time earlier on in the project planning stages in accordance to the “API-first” approach. (3)

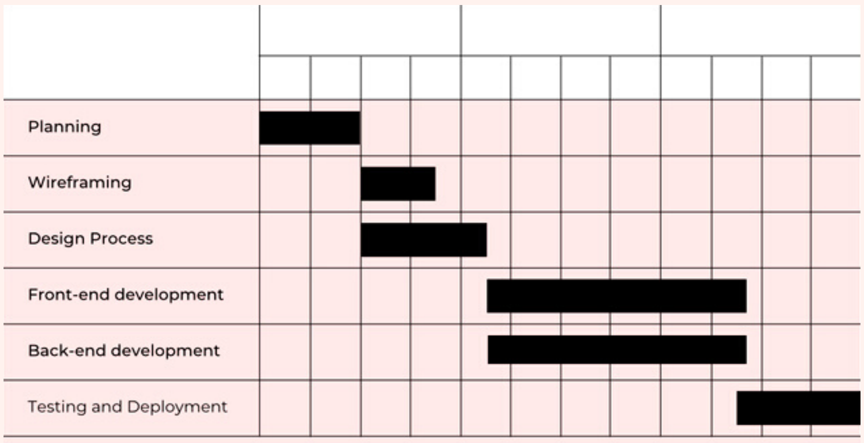
However we implemented two APIs in our programme, Stripe Payment and our postcodes API enabled payment methods and the user’s address details to be filled in based on postcode entry, an effective feature that suited our users needs, thereby allowing swifter checkout process.

We overcame the onset of obstacles that we faced over the past 4 weeks; through pragmatically adapting our work approach to check in with each other through daily scrums. We liaised amongst ourselves and made full use of tutorials and self guided teaching - specifically on areas such as frontend. As more frontend knowledge became apparent we were able to better understand how to visualise our site and the functionality of it, thereby improving our ability to perform test driven development.

Retrospectively, though we felt our application met a lot of our initially outlined requirements, in future we would adapt our way of working to definitely include an API-first approach, and better plan and schedule our meetings to suit the time schedules of all members of the team, thereby maximising our efficiency in working towards deadline of the project.

REFERENCES:

1. Agile Diagram:
2. Bootstrap: https://www.w3schools.com/whatis/whatis_bootstrap.asp
3. API-First Design Approach: https://blog.developer.adobe.com/three-principles-of-api-first-design-fa6666d9f694


Planning
Wireframing
Design Process
Front-end development
Back-end development
Testing and Deployment