

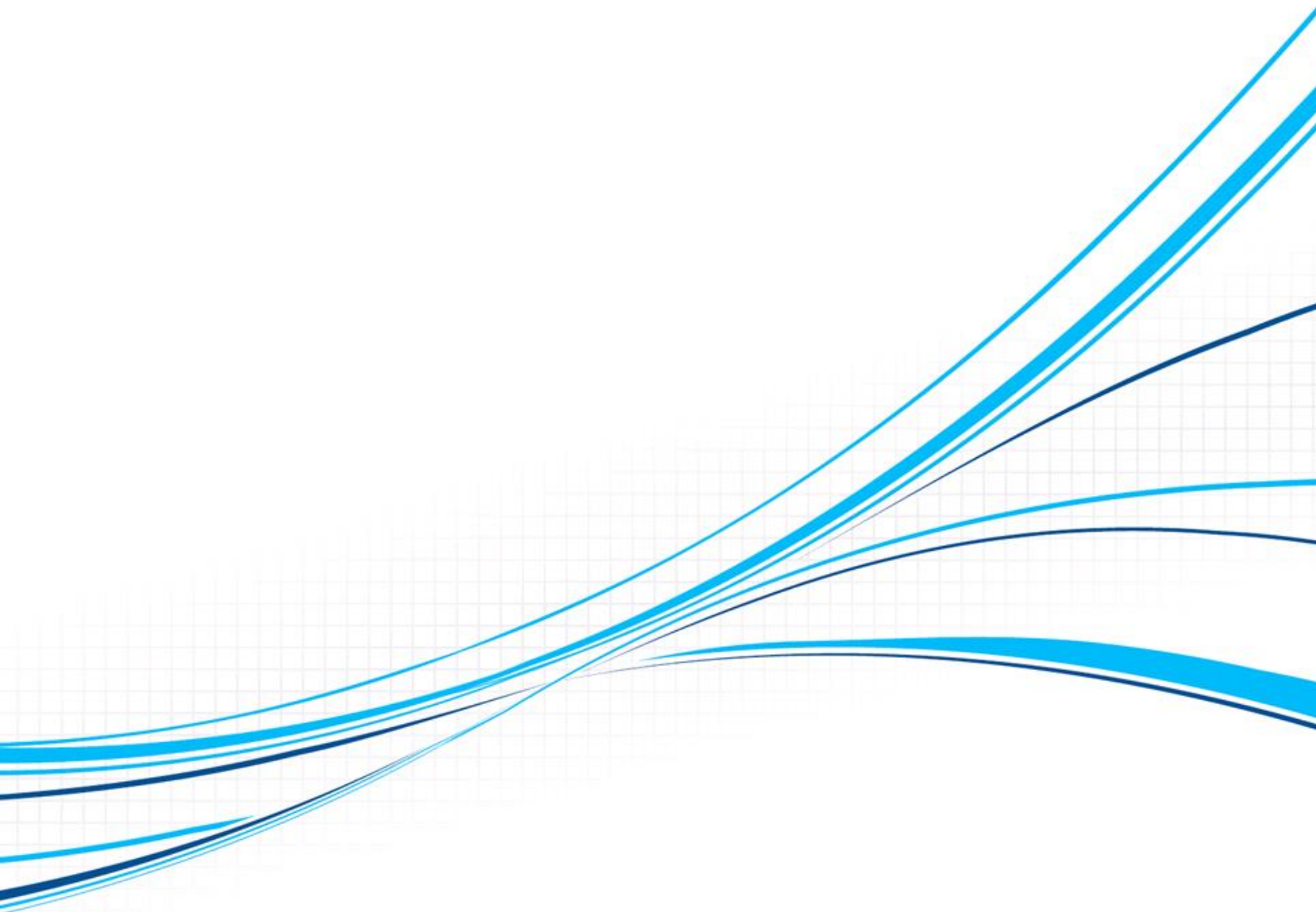
Software Requirements Specification

Shop Project

Created by Antoine Ratat

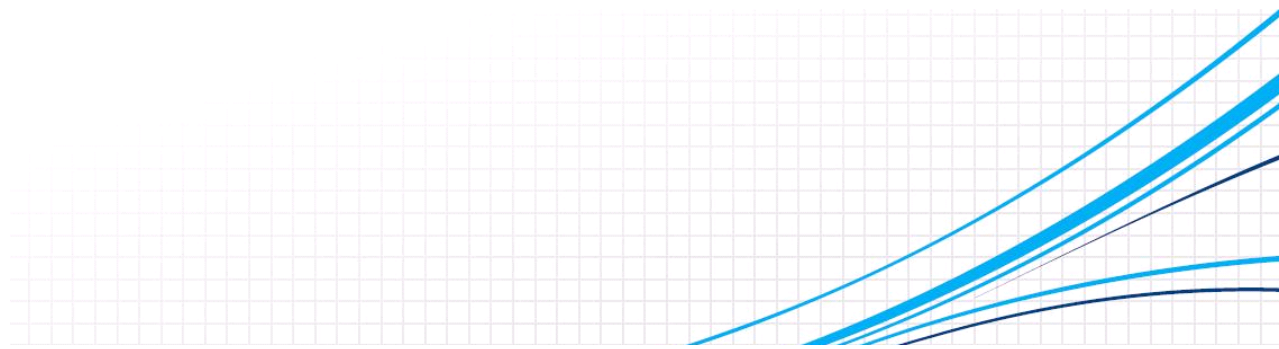
Version 1.0 - Issued February 04, 2021

This requirements specification is used to record the user requirements.



Contents

Software Requirements Specification	1
1 VERSIONS	4
1.1 VERSIONS	4
2 INTRODUCTION	4
2.1 PURPOSE	4
2.2 PROJECT SCOPE	5
2.3 REFERENCES	6
3 DESCRIPTION	6
3.1 PRODUCT PERSPECTIVE	6
3.2 FEATURES	6
3.3 USER OVERVIEW	8
3.4 OPERATING ENVIRONMENT	8
3.5 CONSTRAINTS : IMPLEMENTATION / DESIGN.....	8
3.5.1 Application Communication Schema.....	8
3.5.2 Client-side rendering (frontend)	9
3.5.1 Server-side rendering (backend)	9
SYSTEM FEATURES	11
3.6 SYSTEM FEATURE FRONT END	11
3.6.1 Not Authenticated Features.....	11
3.6.2 Authenticated Features.....	12
3.7 SYSTEM FEATURE BACK END.....	12
3.7.1 User features	12
3.7.2 Admin features.....	13
3.8 SYSTEM FEATURE DATABASE.....	13
3.8.1 User Table	13
3.8.2 User Details Table	13
3.8.3 Cart Details Table.....	13
3.8.4 Category Table	13
3.8.5 Product Table	13
3.8.6 Payment Table	13
3.8.7 Order Table	13
3.8.8 Order Details Table	14
3.8.9 Delivery Table	14
4 REQUIREMENTS OF EXTERNAL INTERFACE	14
4.1 USER INTERFACES	14
4.2 SOFTWARE INTERFACES	16
4.3 COMMUNICATION INTERFACES	17
4.3.1 Admin Operations.....	17
4.3.2 User Operations	20
5 ADDITIONAL NONFUNCTIONAL REQUIREMENTS.....	21
5.1 PERFORMANCE	21
5.2 AVAILABILITY	21
5.3 SECURITY	21
5.4 USABILITY	22



1 VERSIONS

1.1 VERSIONS

Ver.	Author(s)	Date	Description
1.0	Antoine RATAT	04/02/2021	Creating Document

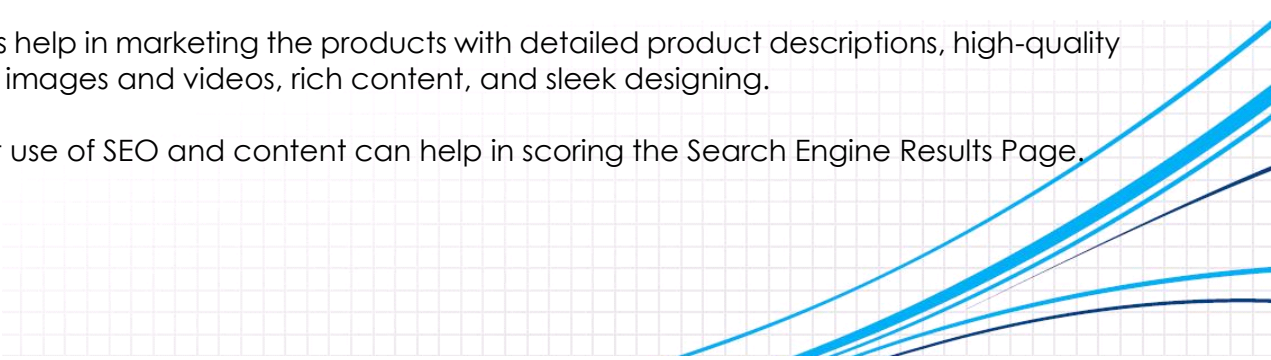
2 INTRODUCTION

2.1 PURPOSE

Shop is an eCommerce solution caters to the exchange of goods for electronic transaction of money. This solution would help prospects in purchasing the products that are displayed in the product lists. People can visit the website, choose the products they like, enter the required details, make a payment, and purchase goods in a short amount of time. An eCommerce website facilitates online transactions with the use of data and fund transfer.

Our solution is Business to Consumer (B2C) oriented, in our case goods and services are sold by one business to a consumer.

This store has for the main purpose to helps in reaching remote customers, that might not know about the existence of physical stores. Later, the application would try to enforce its market by running advertisements and marketing campaigns.

1. We listed down a few points that our eCommerce website aims to achieve:
 2. Online sellers can get a wide range of customers from around the globe.
 3. You can get feedback for your services to improvise and boost user's experience.
 4. The seller and the consumer can interact with each other regarding the products or services.
 5. The consumers can get the details of the product you sell and your brand story
 6. It saves considerable time for the consumers
 7. Consumers can purchase online from anywhere at any time using your eCommerce website.
 8. Websites help in marketing the products with detailed product descriptions, high-quality product images and videos, rich content, and sleek designing.
 9. The right use of SEO and content can help in scoring the Search Engine Results Page.
- 

10. It can collect consumer data including the demographics that can be used for target marketing.
11. You can add unlimited products to your eCommerce website and categorize them to sell efficiently.
12. An eCommerce website would be responsive to provide customer convenience.
13. A good website design along with strong content can help you build online traffic to boost sales.
14. A website can help you in creating brand authenticity and consumer trust.

2.2 PROJECT SCOPE

Shop is targeting small businesses that just start selling online.

We currently identified five essential features that this platform would need to have :

1. The main goal of the platform is to allow the customer to sell exactly what they want, where they want, and how they want. This means the platform should be able to deal with both digital and physical goods. Especially for small businesses with physical premises that plans to sell internationally. The application should be able to be accessible in multiple languages. And can in the near-future evolve with the following versions including ways to handle international sales taxes and currencies.
2. Another important point is the ability to build a good looking, modern store that works on any device. It would need to include customization options that allow the customer to fit their existing brand and design charters.
3. The platform has to be able to manage orders, ship goods, and track inventory. In simple words, handle the running back-end without the hassle of using other apps and spreadsheets.
4. The platform also has to offer ways to integrate with other services, marketplaces, and apps.
5. All these features have to be offered at a reasonable price in a fully customized, done-for-you solution.

This platform charges a little differently than a lot of other services and includes a different kind of expenses :

First, there's the monthly fee it would include hosting, acquiring a domain name, and associated SSL certificates.

Then, there are the payment gateway fees. This platform will rely on Stripe and will charge a flat rate of 2.9% + 30¢ per successful charge on each payment to handle the transaction.

Finally, there are the maintenance fees, that can regroup Domain Registration Cost, Hosting Cost, Website design costs, Technical maintenance on the website, Additional plugins and



tools, SSL license, Backup verifications, Software Updates, Check for broken resources, Check Analytics and rankings.

2.3 REFERENCES

React - <https://reactjs.org/>
React Router - <https://reactrouter.com/web/guides/quick-start>
React Bootstrap - <https://react-bootstrap.github.io/>
React Spinners - <http://www.davidhu.io/react-spinners/>
React Toastify - <https://www.npmjs.com/package/react-toastify/v/1.4.3>
Country List JS - <https://www.npmjs.com/package/country-list-js>
Country Flag Icon - <https://www.npmjs.com/package/country-flag-icons>
Formik - <https://www.npmjs.com/package/formik>
Yup - <https://www.npmjs.com/package/yup>
React Token Auth - <https://www.npmjs.com/package/react-token-auth>
Flask - <https://flask.palletsprojects.com/en/1.1.x/>
Flask Mail - <https://flask-mail.readthedocs.io/en/latest/>
ItsDangerous - <https://pypi.org/project/itsdangerous/>
Flask SQLAlchemy - <https://flask-sqlalchemy.palletsprojects.com/en/2.x/>
Flask JWT Extended - <https://flask-jwt-extended.readthedocs.io/en/stable/>
Flask Bcrypt - <https://flask-bcrypt.readthedocs.io/en/latest/>
Moment - <https://www.npmjs.com/package/moment>

3 DESCRIPTION

3.1 PRODUCT PERSPECTIVE

The product is a full-stack web application implemented on both the front and back end, it is using API calls to operate the database.

3.2 FEATURES

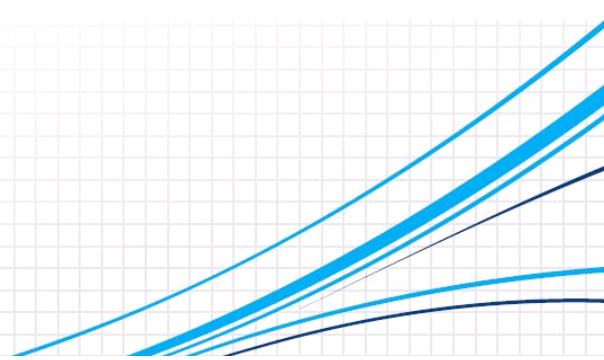
The Shop platform system provides a simple mechanism for users to acquire information.

The following are the main features that are included in the system:

→ Products

- Products Display
- Filter Category
- Price Ordering
- Collection Products Page
- Single Product Display
- Stock Verification
- Select Quantity Modal
- Place in Cart
- Constraint 5 Similar Products

→ Cart

- Add new Product to the Cart
 - Change Quantity for Product in the Cart
 - Delete Product from the Cart
- 

- Recalculate subtotal for each Product on change
- Recalculate total Cart on change

→ **Delivery Management**

- Force Add address if no address
- Add new address during Checkout
- Edit existing address
- Delete existing address
- Select address from customer address list

→ **Payment**

- Summary with address, products list, total Order
- Credit Card verification
- Payment handle by Stripe API

→ **Performance**

- Async Data Loading
- Each page as a loading state to handle API delay
- Each page as an Error state if no data is returned from the API
- Disable Buttons while loading
- Async Image Loading for images components
- IntersectionObserver to load images component within 200px of the current viewport

→ **Security**

- HTTPS use to transmit sensitive data (credit card numbers, login credentials)
- JSON Web Token to Authenticate user and transmit information in payload
- Use remote API with Cross-Origin Resource Sharing (CORS) to allow communication origin
- Data validation in Front-end (Formik & Yup)
- Data validation in Back-end
- Verify user email to activate account

→ **User Management**

- Login, Register, Logout
- Forgot Password Functionality
- Set Password Page from Email verification Link
- Profile Management
- Customer Graphical Personal Card
- Edit Information functionality
- Update Password functionality
- Delete Account functionality

→ **Language**

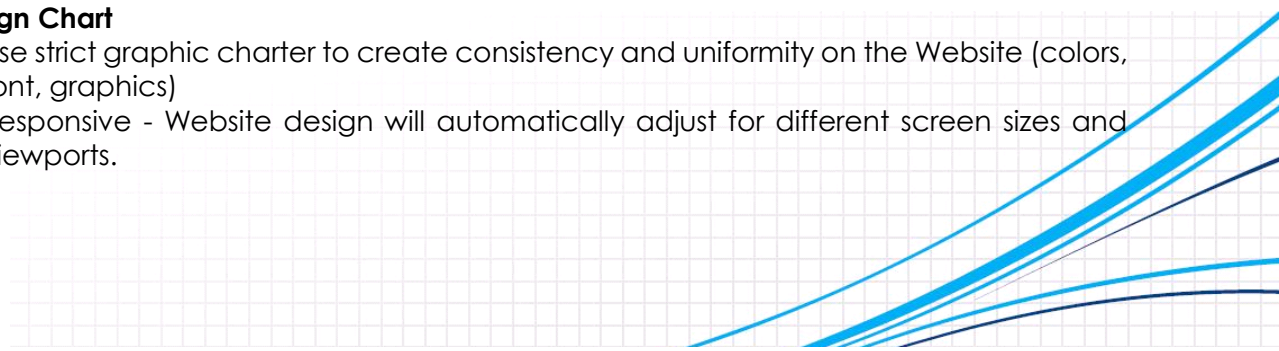
- Website is completely translatable into English or Mandarin Chinese
- Use Context to handle Language Management
- Store Language Preferences in User browser (localStorage)
- Store Translation in JSON object

→ **Order Tracking**

- Order tracking functionality
- Transporter information summary
- Order Number for customer service

→ **Design Chart**

- Use strict graphic charter to create consistency and uniformity on the Website (colors, font, graphics)
- Responsive - Website design will automatically adjust for different screen sizes and viewports.



- Using CSS to hide, shrink or enlarge Website elements
- Using Bootstrap for element positioning

3.3 USER OVERVIEW

It is considered that the user does have the basic knowledge of operating the internet and to have access to it.

The administrator is expected to be familiar with HTML, CSS, JavaScript, React library, AJAX (Asynchronous JavaScript and XML), React Router, React Pagination, React Token Authentication, React ChartJS2, React Bootstrap, JWT Decode, Formik, Yup, and Styled components to handle Front-End's side. Regarding the Back-End perspective the administrator is expected to be familiar with Python, Flask, Flask-SQLAlchemy, Flask-Mail, SMTP Satellite Configuration, CORS configuration, JWT Tokens, Authentication, and Password hashing mechanisms.

3.4 OPERATING ENVIRONMENT

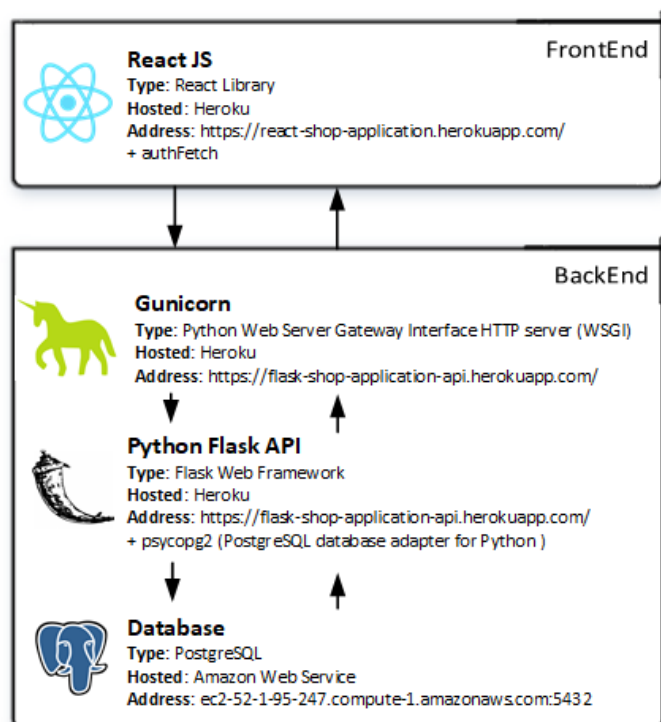
This is a web-based system and hence will require the operating environment for a client and server GUI.

Dependencies

- This software highly depends on the type and version of the browser being installed in the system i.e. browser version should be used which has HTML5 support.

3.5 CONSTRAINTS : IMPLEMENTATION / DESIGN

3.5.1 Application Communication Schema



3.5.2 Client-side rendering (frontend)

This system is provisioned to be built in JavaScript using React library which is highly flexible.

The browser will be in charge of rendering this application in its final form, HTML. Some of the logic involved in creating the web page, especially the one in charge of dealing with presentation logic is handled on the client-side.

List of frontend dependencies and version used:

```
@fortawesome/fontawesome-svg-core v1.2.32
@fortawesome/free-regular-svg-icons v5.15.1
@fortawesome/free-solid-svg-icons v5.15.1
@fortawesome/react-fontawesome v0.1.14
@stripe/react-stripe-js v1.1.2
@stripe/stripe-js v1.11.0
@testing-library/jest-dom v5.11.4
@testing-library/react v11.1.0
@testing-library/user-event v12.1.10
animate.css v4.1.1
country-flag-icons v1.2.8
country-list-js v3.1.7
formik v2.2.5
jwt-decode v3.1.2
moment v1.1.0
react v17.0.1
react-bootstrap v1.4.0
react-dom v17.0.1
react-helmet v6.1.0
react-router-dom v5.2.0
react-script v4.0.1
react-select v3.1.1
react-spinners v0.9.0
react-toastify v6.2.0
react-token-auth v1.1.7
web-vitals v0.2.4
yup v0.32
```

3.5.1 Server-side rendering (backend)

The system is also provisioned to connect and contact a custom made API. This API handles CRUD request and manage operations on the database.

The API is build using Python with Flask web framework. The application server would be served with Gunicorn (Python WSGI HTTP Server for UNIX)

The API will interface the database using SQLAlchemy. SQLAlchemy is an object-relational mapper (ORM) and provides the data mapper pattern, where classes can be mapped to the database in open-ended, multiple ways - allowing the object model and database schema to develop in a cleanly decoupled way from the beginning.

Authentication will be handled using Flask-JWT-Extended which adds support for using JSON Web Tokens (JWT) to Flask for protecting views.

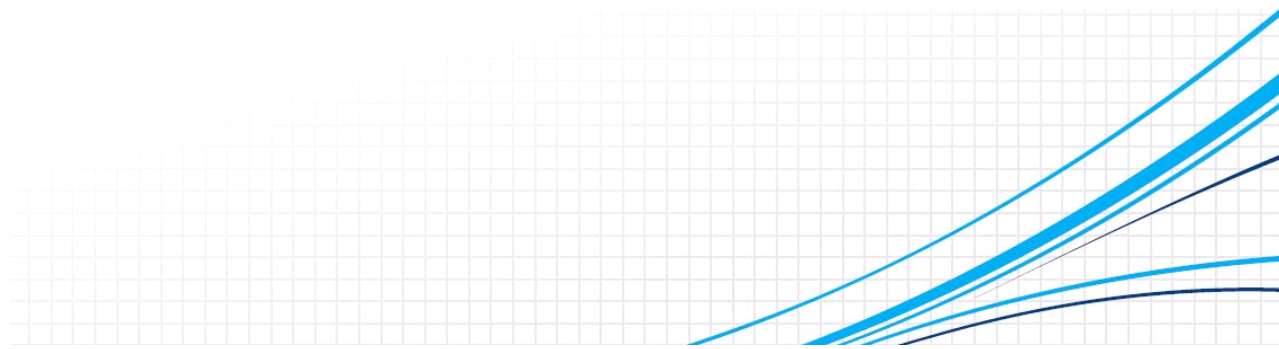


List of used dependencies and version used:

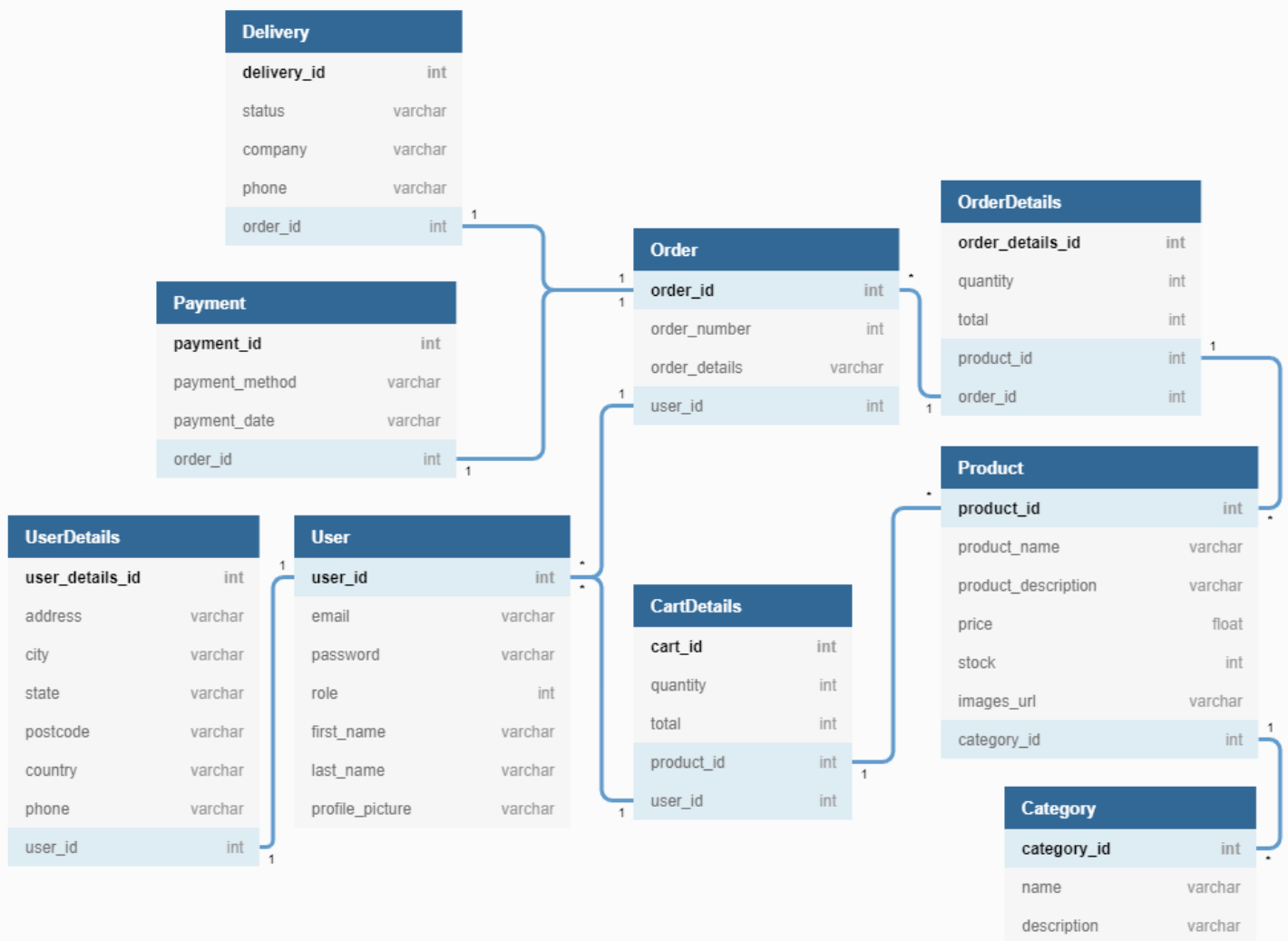
astroid V2.4.2
bcrypt V3.2.0
blinker V1.4
certifi V2020.12.5
cffi V1.14.4
chardet V4.0.0
click V7.1.2
colorama V0.4.4
Flask V1.1.2
Flask-Bcrypt V0.7.1
Flask-Cors V3.0.9
Flask-JWT-Extended V3.25.0
Flask-Mail V0.9.1
Flask-SQLAlchemy V2.4.4
gunicorn V20.0.4
idna V2.10
isort V5.6.4
itsdangerous V1.1.0
Jinja2 V2.11.2
lazy-object-proxy V1.4.3
MarkupSafe V1.1.1
mccabe V0.6.1
psycpg2 V2.8.6
pyparser V2.20
PyJWT V1.7.1
pylint V2.6.0
requests V2.25.1
six V1.15.0
SQLAlchemy V1.3.20
stripe V2.55.1
toml V0.10.2
urllib3 V1.26.2
Werkzeug V1.0.1
wrap V1.12.1

The database itself is using PostgreSQL and is accessible for UI management tools like PgAdmin. It would be hosted on AWS (Amazon Web Service) and available on port 5432.

The database would contain the following tables: User, UserDetails, Order, Delivery, Payment, Category, Product, OrderDetails, CartDetails.



You can find an ER Entity Relationship Diagram to understand its structure:



Hardware Interface

The device should be enabled with the Internet.

Software interface

The user's browser should be HTML5 compatible for a satisfactory user experience.

SYSTEM FEATURES

3.6 SYSTEM FEATURE FRONT END

3.6.1 Not Authenticated Features

- Create an account
- Login
- Forget Password
- Reset Password
- Browse Products
- Use Filter on Products
- Open single product for details
- Change Language

3.6.2 Authenticated Features

Navigation Experience

- Browse Products
- Apply Filter on Products
- Open single product for details
- Change Language

User Information

- Edit Profile Information
- Update Password
- Delete Account
- Logout

Cart

- Add Products to Cart
- Edit Products in Cart
- Delete Products in Cart
- Increase quantity Products in Cart
- Decrease quantity Products in Cart
- Preview Product in Cart

Delivery Information

- Consult existing address
- Add new address
- Edit existing address
- Delete existing address
- Select existing address from list

Payment


- Consult Order Delivery summary
- Consult Order Product summary
- Consult Order Product SubTotal
- Consult Order Total
- Pay

Orders

- Consult previous orders
- Track order status
- Consult order advanced details

3.7 SYSTEM FEATURE BACK END

3.7.1 User features

- Allow user to create an account
 - Allow user to login
 - Allow user to disconnect
 - Allow the user to edit profile
 - Allow user to delete their account
 - Allow user to reset their password through Email
- 

- Allow user to handle their cart (add, edit quantity or delete item in cart)
- Allow user to make a payment
- Allow user to choose delivery
- Allow user to create order

User account is only available after the first login with User account authenticated with User JWT.

3.7.2 Admin features

- Allow admin to access, create, edit or delete categories
- Allow admin to access, create, edit or delete products
- Allow admin to access, create, edit or delete users
- Allow admin to access, create, edit or delete userdetails
- Allow admin to access, create, edit or delete order
- Allow admin to access, create, edit or delete orderdetails
- Allow admin to access, create, edit or delete cartdetails
- Allow admin to access, create, edit or delete delivery
- Allow admin to access, create, edit or delete payment

Admin features are only available to Admin account authenticated with Admin JWT.

3.8 SYSTEM FEATURE DATABASE

3.8.1 User Table

For each subscribed user store information such as email, hashed password, role, first and last name

3.8.2 User Details Table

For each user details store information such as user corresponding user id, address, city, state, postcode, country, phone.

3.8.3 Cart Details Table

For each cart details store information such as user corresponding user id, cart item quantity, matching product id, total product.

3.8.4 Category Table

For each category store information such as category name, category description and category gender.

3.8.5 Product Table

For each product store information such as user corresponding category id, product name, product description, product price, product stock and product image URL.

3.8.6 Payment Table

For each payment store information regarding matching order id, payment status, currency used, amount paid, payment date, payment method, payment method number and payment stripe number.

3.8.7 Order Table

For each order has information with corresponding user id, order number and order details.



3.8.8 Order Details Table

For each order details store information such as id, product quantity, product subtotal and corresponding product id.

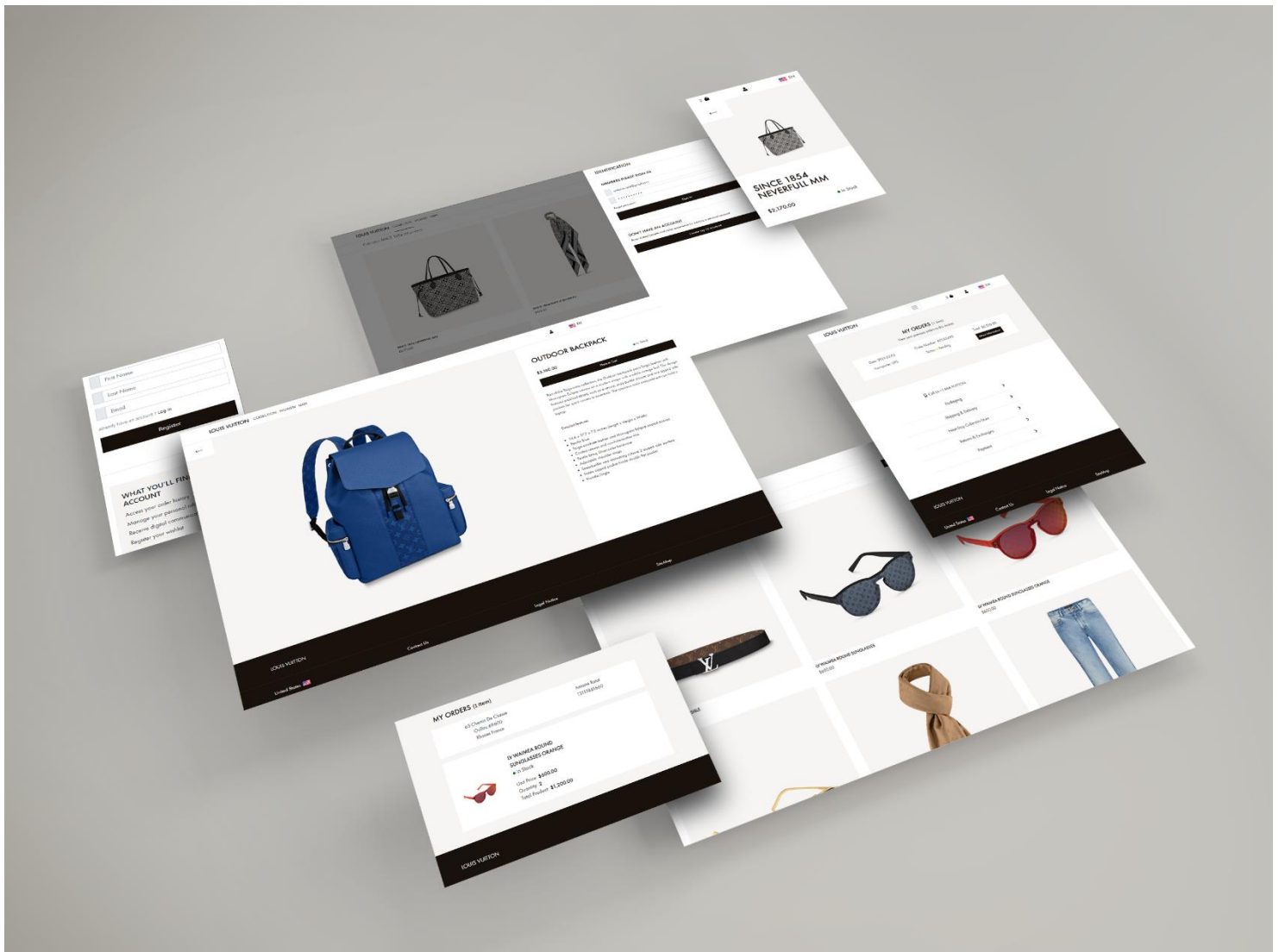
3.8.9 Delivery Table

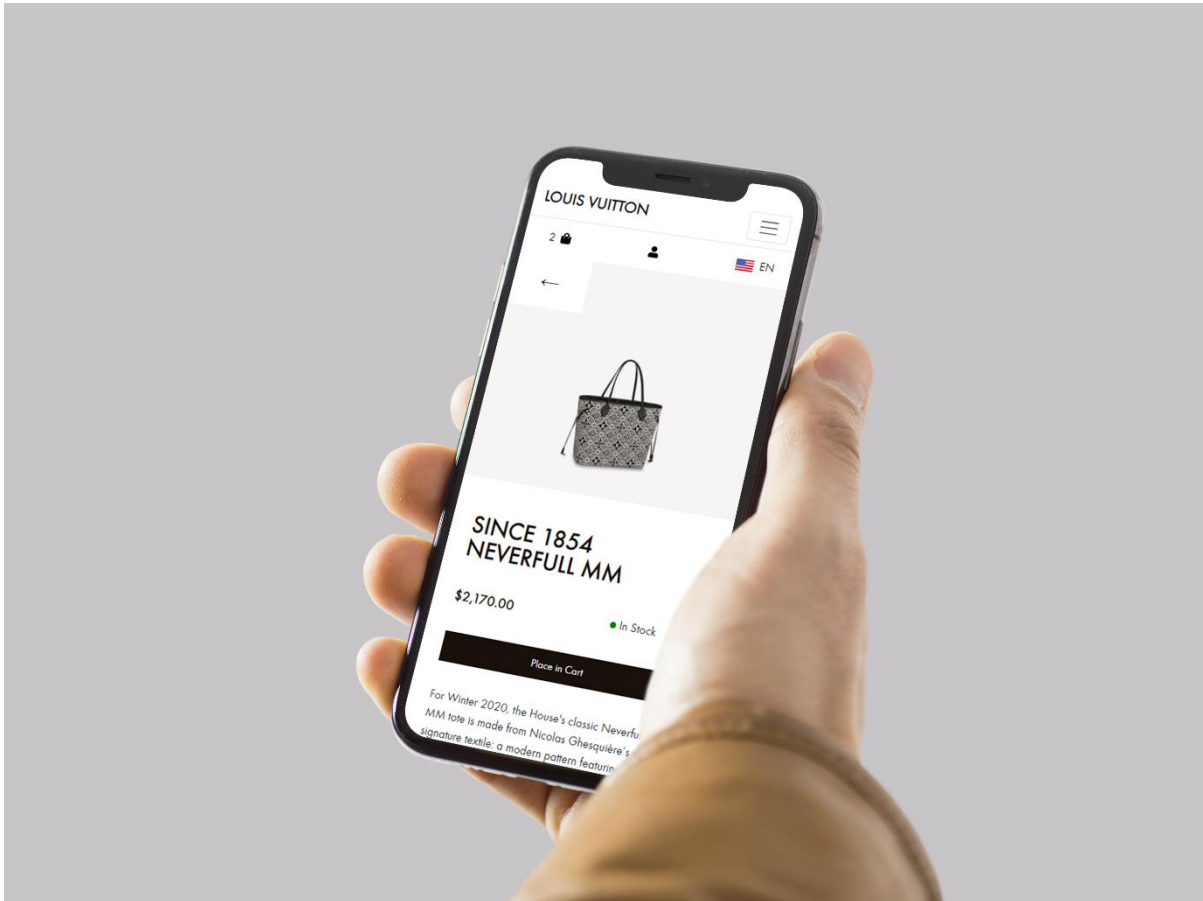
For each delivery store information such as user corresponding order id, delivery status, transporter company, and transporter phone

4 REQUIREMENTS OF EXTERNAL INTERFACE

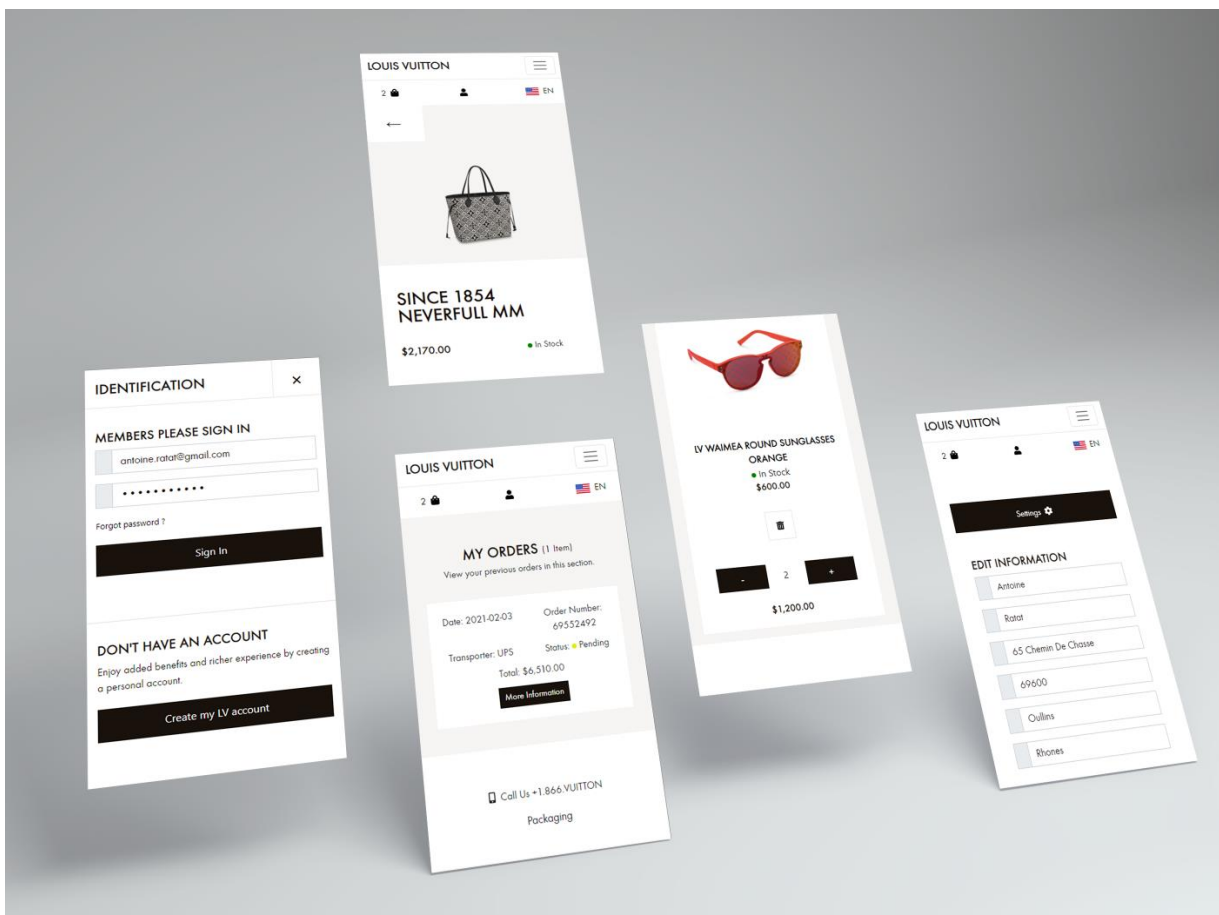
4.1 USER INTERFACES

Main Pages Full-Screen Desktop Mockup





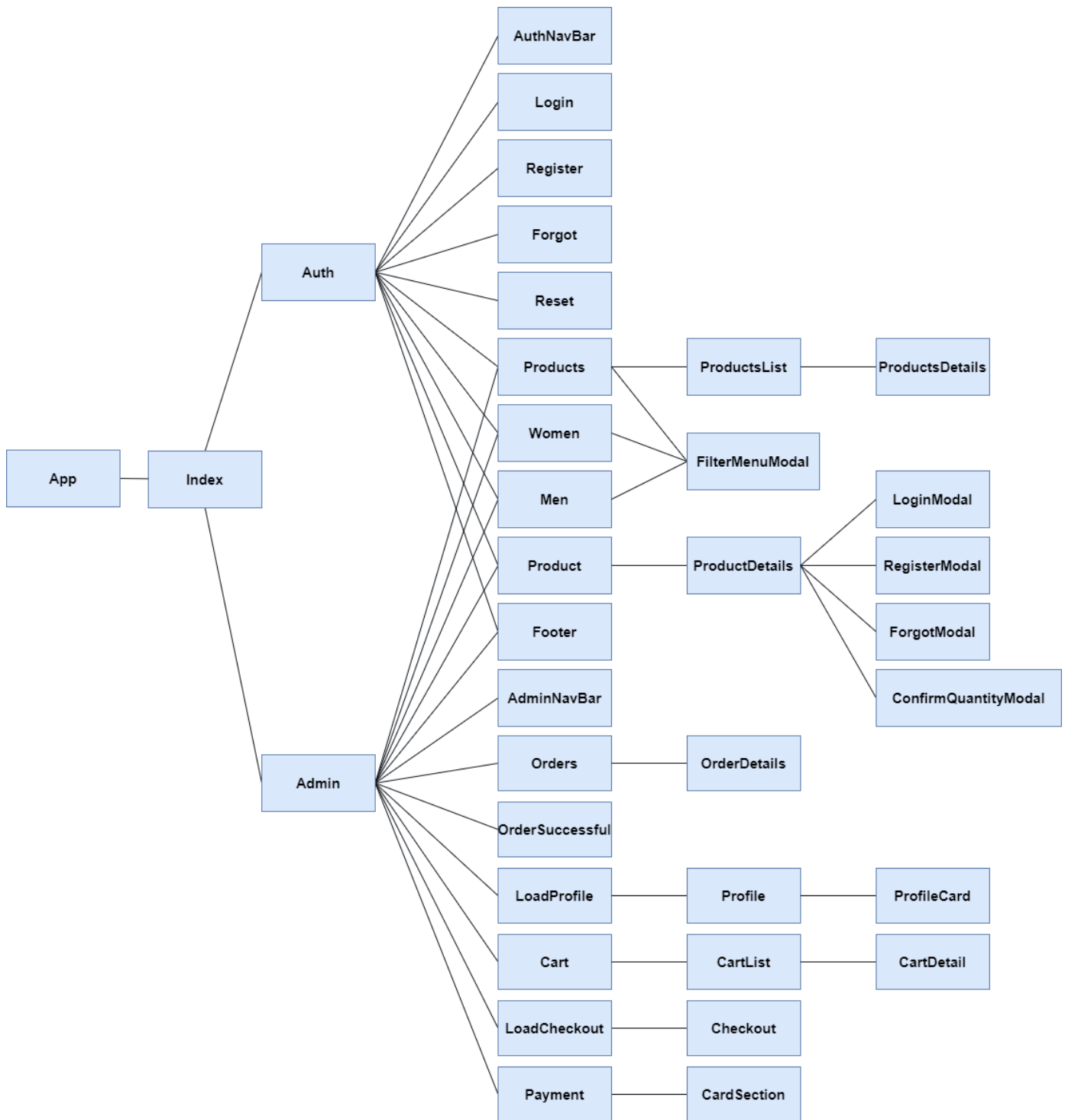
Login Modal – Product Browsing – My Order – My Cart – My Delivery Information - Mobile.



4.2 SOFTWARE INTERFACES

Software is designed in small fragmented atomic components. Each component has specific functionality and assembled creates our application.

This is easier to maintain, replace, and re-use. The component organization of the Shop app is available in the scheme below:



4.3 COMMUNICATION INTERFACES

Communication is assured to external interfaces. The system is connected to several APIs using REST (representational state transfer), The payload is defined in the request itself and is formatted in JSON. Most of the operations are directed to a custom-made API. This API is a CRUD API created with Python and Flask technology to handle the Front-End requests and communicate safely with a PostgreSQL database.

4.3.1 Admin Operations

Category CRUD

Requires JWT Token auth as admin

GET /api/admin/categories Return JSON with all categories.
<https://flask-shop-application-api.herokuapp.com/api/admin/categories>

POST /api/admin/categories Post new category from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/categories>

GET /api/admin/category/id Return JSON selected category.
<https://flask-shop-application-api.herokuapp.com/api/admin/category/id>

PUT /api/admin/category/id Update selected category from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/category/id>

DELETE /api/admin/category/id Delete selected category.
<https://flask-shop-application-api.herokuapp.com/api/admin/category/id>

Product CRUD

Requires JWT Token auth as admin

GET /api/admin/products Return JSON with all products.
<https://flask-shop-application-api.herokuapp.com/api/admin/products>

POST /api/admin/products Post new product from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/products>

GET /api/admin/product/id Return JSON selected product.
<https://flask-shop-application-api.herokuapp.com/api/admin/product/id>

PUT /api/admin/product/id Update selected product from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/product/id>

DELETE /api/admin/product/id Delete selected product.
<https://flask-shop-application-api.herokuapp.com/api/admin/product/id>

User CRUD

Requires JWT Token auth as admin

GET /api/admin/users Return JSON with all users.
<https://flask-shop-application-api.herokuapp.com/api/admin/users>

POST /api/admin/users Post new user and userdetail from JSON

<https://flask-shop-application-api.herokuapp.com/api/admin/users>

GET /api/admin/user/id Return JSON selected user.

<https://flask-shop-application-api.herokuapp.com/api/admin/user/id>

PUT /api/admin/user/id Update selected user and userdetail from JSON

<https://flask-shop-application-api.herokuapp.com/api/admin/user/id>

DELETE /api/admin/user/id Delete selected user.

<https://flask-shop-application-api.herokuapp.com/api/admin/user/id>

UserDetails CRUD

Requires JWT Token auth as admin

GET /api/admin/userdetails Return JSON with all userdetails.

<https://flask-shop-application-api.herokuapp.com/api/admin/userdetails>

GET /api/admin/userdetail/id Return JSON selected userdetail.

<https://flask-shop-application-api.herokuapp.com/api/admin/userdetail/id>

Order CRUD

Requires JWT Token auth as admin

GET /api/admin/orderdetails Return JSON with all orders.

<https://flask-shop-application-api.herokuapp.com/api/admin/orders>

POST /api/admin/orders Post new order from JSON

<https://flask-shop-application-api.herokuapp.com/api/admin/orders>

GET /api/admin/order/id Return JSON selected order.

<https://flask-shop-application-api.herokuapp.com/api/admin/order/id>

PUT /api/admin/order/id Update selected order from JSON

<https://flask-shop-application-api.herokuapp.com/api/admin/order/id>

DELETE /api/admin/order/id Delete selected order.

<https://flask-shop-application-api.herokuapp.com/api/admin/order/id>

OrderDetails CRUD

Requires JWT Token auth as admin

GET /api/admin/orderdetails Return JSON with all orderdetails.

<https://flask-shop-application-api.herokuapp.com/api/admin/orderdetails>

POST /api/admin/orderdetails Post new orderdetail from JSON

<https://flask-shop-application-api.herokuapp.com/api/admin/orderdetails>

GET /api/admin/orderdetail/id Return JSON selected orderdetail.

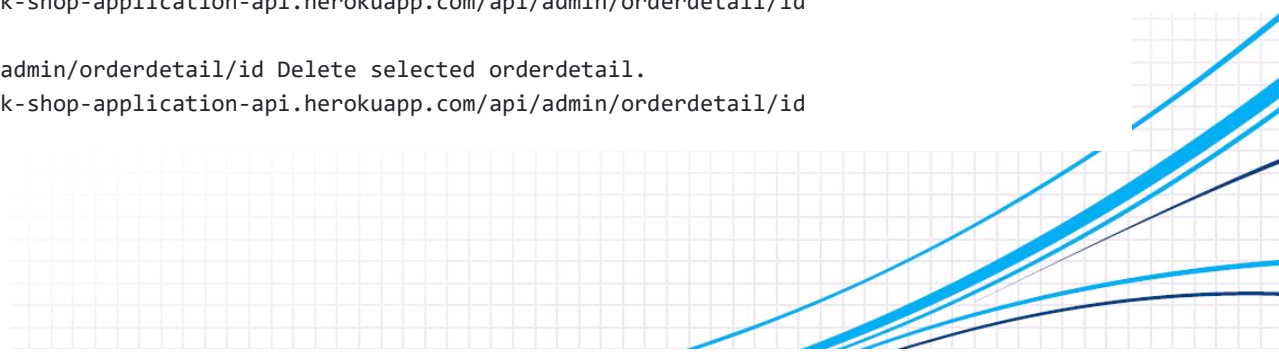
<https://flask-shop-application-api.herokuapp.com/api/admin/orderdetail/id>

PUT /api/admin/orderdetail/id Update selected orderdetail from JSON

<https://flask-shop-application-api.herokuapp.com/api/admin/orderdetail/id>

DELETE /api/admin/orderdetail/id Delete selected orderdetail.

<https://flask-shop-application-api.herokuapp.com/api/admin/orderdetail/id>



CartDetails CRUD

Requires JWT Token auth as admin

GET /api/admin/cartdetails Return JSON with all cartdetails.
<https://flask-shop-application-api.herokuapp.com/api/admin/cartdetails>

POST /api/admin/cartdetails Post new cartdetail from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/cartdetails>

GET /api/admin/cartdetail/id Return JSON selected cartdetail.
<https://flask-shop-application-api.herokuapp.com/api/admin/cartdetail/id>

PUT /api/admin/cartdetail/id Update selected cartdetail from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/cartdetail/id>

DELETE /api/admin/cartdetail/id Delete selected cartdetail.
<https://flask-shop-application-api.herokuapp.com/api/admin/cartdetail/id>

Delivery CRUD

Requires JWT Token auth as admin

GET /api/admin/deliveries Return JSON with all deliveries.
<https://flask-shop-application-api.herokuapp.com/api/admin/deliveries>

POST /api/admin/deliveries Post new delivery from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/deliveries>

GET /api/admin/delivery/id Return JSON selected delivery.
<https://flask-shop-application-api.herokuapp.com/api/admin/delivery/id>

PUT /api/admin/delivery/id Update selected delivery from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/delivery/id>

DELETE /api/admin/delivery/id Delete selected delivery.
<https://flask-shop-application-api.herokuapp.com/api/admin/delivery/id>

Payment CRUD

Requires JWT Token auth as admin

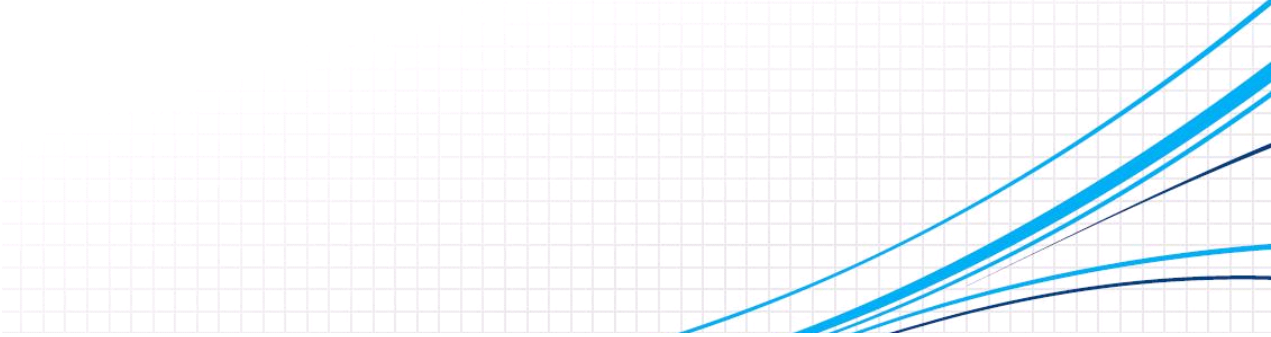
GET /api/admin/payments Return JSON with all payments.
<https://flask-shop-application-api.herokuapp.com/api/admin/payments>

POST /api/admin/payments Post new payment from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/payments>

GET /api/admin/payment/id Return JSON selected payment.
<https://flask-shop-application-api.herokuapp.com/api/admin/payment/id>

PUT /api/admin/payment/id Update selected payment from JSON
<https://flask-shop-application-api.herokuapp.com/api/admin/payment/id>

DELETE /api/admin/payment/id Delete selected payment.
<https://flask-shop-application-api.herokuapp.com/api/admin/payment/id>



4.3.2 User Operations

User Auth Management

POST /api/login Return JSON with Access TOKEN.

<https://flask-shop-application-api.herokuapp.com/api/login>

POST /api/register Send Email set Password.

<https://flask-shop-application-api.herokuapp.com/api/register>

POST /api/forgot Send Email set Password.

<https://flask-shop-application-api.herokuapp.com/api/forgot>

POST /api/set_password Set/Update password.

https://flask-shop-application-api.herokuapp.com/api/set_password

User CRUD

Requires JWT Token auth as user

GET /api/user Return JSON with user and userdetails .

<https://flask-shop-application-api.herokuapp.com/api/user>

PUT /api/user Update selected user from JSON

<https://flask-shop-application-api.herokuapp.com/api/user>

DELETE /api/user Delete selected user.

<https://flask-shop-application-api.herokuapp.com/api/user>

UserDetails CRUD

Requires JWT Token auth as user

GET /api/userdetails Return JSON with all userdetails for user.

<https://flask-shop-application-api.herokuapp.com/api/userdetails>

POST /api/userdetails Add userdetails to user.

<https://flask-shop-application-api.herokuapp.com/api/userdetails>

GET /api/userdetail/user_details_id Get selected userdetail.

https://flask-shop-application-api.herokuapp.com/api/userdetail/user_details_id

PUT /api/userdetail/user_details_id Update selected userdetail from JSON

https://flask-shop-application-api.herokuapp.com/api/userdetail/user_details_id

DELETE /api/userdetail/user_details_id Delete selected userdetail.

https://flask-shop-application-api.herokuapp.com/api/userdetail/user_details_id

Orders / Carts / Products / Category CRUD

Requires JWT Token auth as user

GET /api/categories Return JSON with categories.

<https://flask-shop-application-api.herokuapp.com/api/categories>

GET /api/products Return JSON with products.

<https://flask-shop-application-api.herokuapp.com/api/products>

GET /api/product/id Return JSON with single product.

<https://flask-shop-application-api.herokuapp.com/api/product/id>



GET /api/orders Return JSON with orders for user.
<https://flask-shop-application-api.herokuapp.com/api/orders>

GET /api/cart Return JSON with cart for user.
<https://flask-shop-application-api.herokuapp.com/api/cart>

POST /api/cart Add to Cart for user.
<https://flask-shop-application-api.herokuapp.com/api/cart>

5 ADDITIONAL NONFUNCTIONAL REQUIREMENTS

5.1 PERFORMANCE

The system must be interactive and the delays involved must be fewer. So, in every action-response of the system, there are no immediate delays. In the case of opening new windows, popping error messages, and saving the settings or sessions there is a delay depending on the API response time and availability.

In the case of calling external APIs, the delay is based on editing on the distance of the secondary system and the configuration between them so there is a high probability that there will be or not a successful connection in less than 5 seconds for sake of good communication.

The following also needs to be implemented :

- Async Data Loading: To Load prior fast elements
- Each page has a loading state to handle API delay
- Each page has an Error state if no data is returned from the API
- Disable Buttons while loading
- Async Image Loading for images components
- IntersectionObserver to load images component within 200px of the current viewport

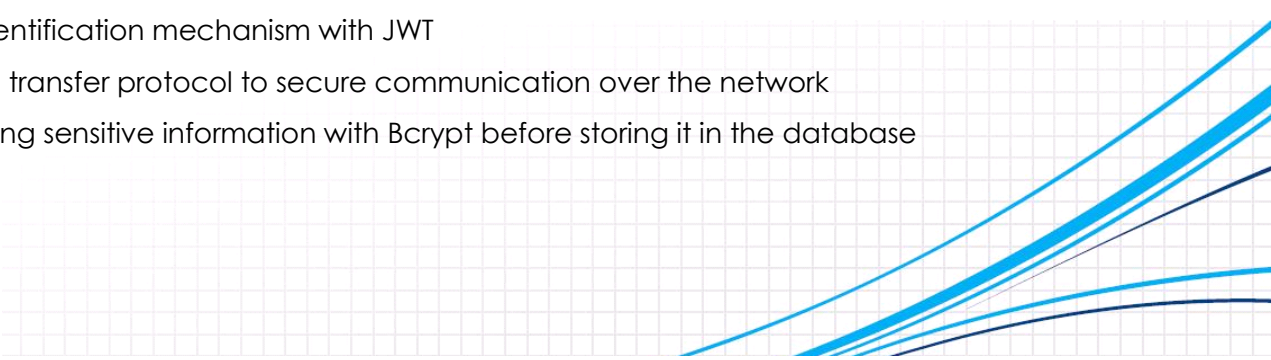
5.2 AVAILABILITY

If the internet service gets disrupted while fetching information from the external APIs, the information can be fetched again later.

The system is highly dependant on external APIs, if the communication between the client and the API is disrupted, the information could be requested again.

API contacted : <https://flask-shop-application-api.herokuapp.com/api/>*

5.3 SECURITY

- Check input data with Form Validation using Formik and Yup
 - Back end verification before any database operation
 - Authentification mechanism with JWT
 - HTTPS transfer protocol to secure communication over the network
 - Hashing sensitive information with Bcrypt before storing it in the database
- 

- Storing an application's sensitive information in an environment variable on the server
- JSON Web Token to Authenticate user and transmit information in the payload
- Use remote API with Cross-Origin Resource Sharing (CORS) to allow communication origin
- Verify user email to activate account

5.4 USABILITY

As the system is easy to handle and navigates in the most expected way with no delays. In that case, the system program reacts accordingly and transverses quickly between its states.

