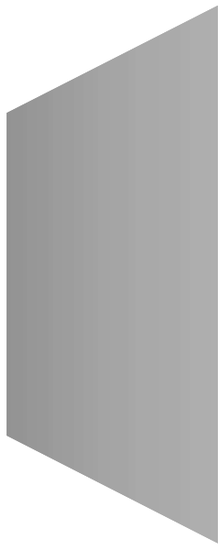
Retail Store Online| Demonstrator

Presentation

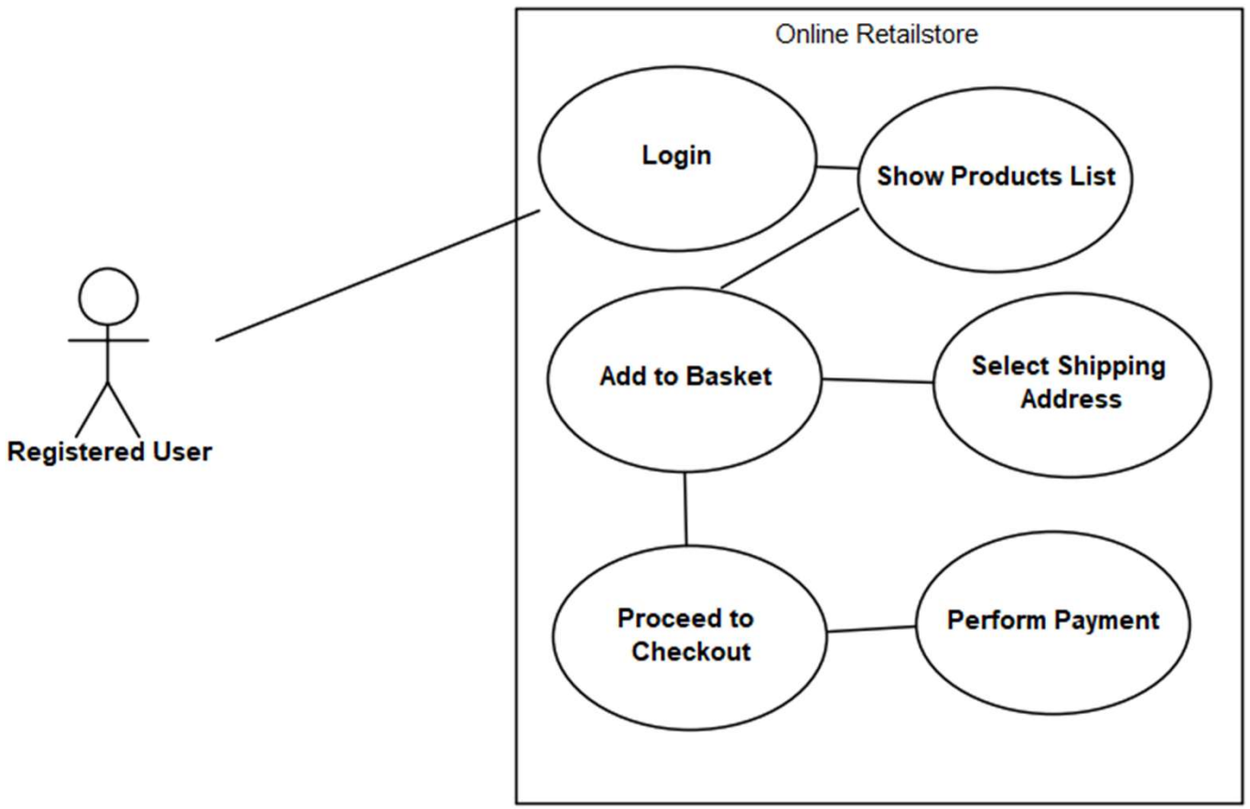


Agenda

Overview| Architecture| Getting Started -API | ReactJS | Summary

Source: All Icons courtesy: https://thenounproject.com/

Overview| Use case/Context



First set of Features

1. *User should be able to **login and see a list of products**, select from there, select one of the defined addresses and do payment and see the order in order list.*
2. *Delivery happens separate as Shipped by the Seller and once Delivered it is updated in the System.*

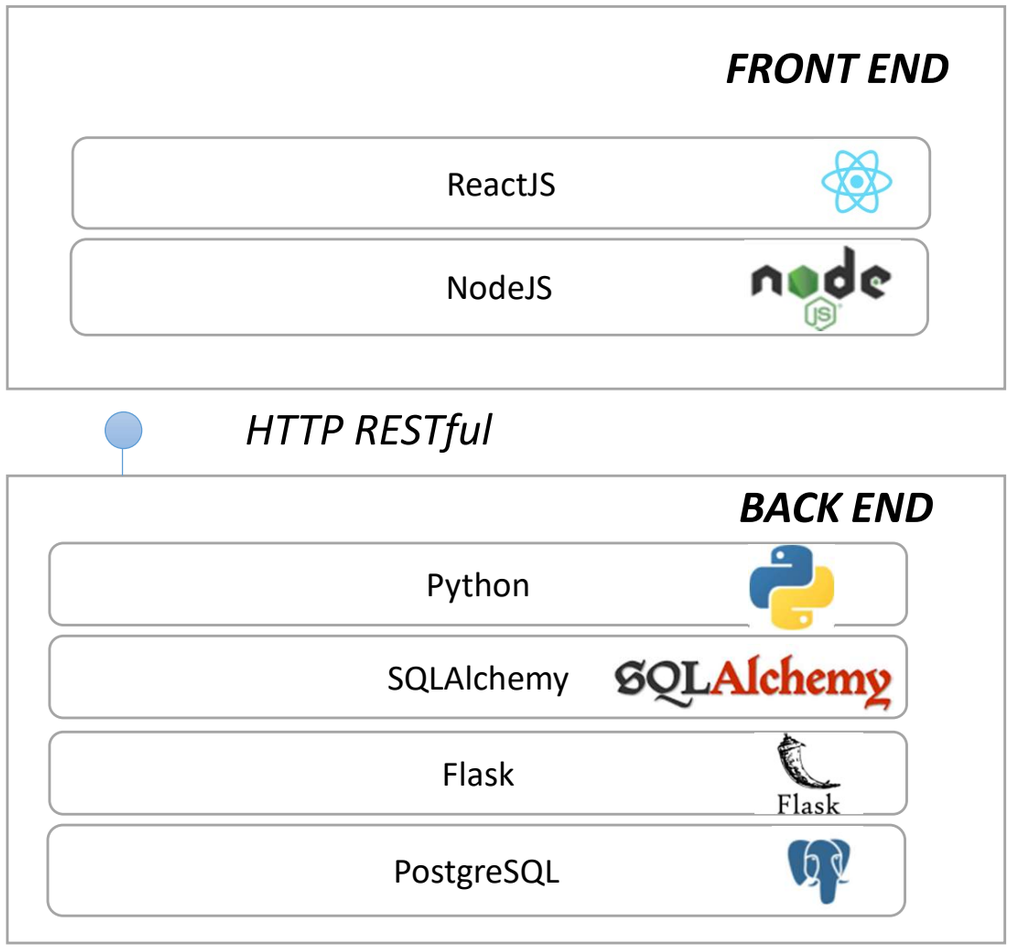
Assumptions:

User information including Address, Product Types, Products List are available. Everything available in Products List is available in Store in infinite numbers (Not checking availability at the moment).

Payment is dummy interface. Inventory and price history is not handled now. Price is fixed at the moment.

Context Setting

Overview| Architecture



The Front End is build using ReactJS

Service/API Testing is done using Postman

The Services/Back End is built using Python with Flask microframework serving HTTP RESTful services.

Architecture – Layered View

Architecture| Back End

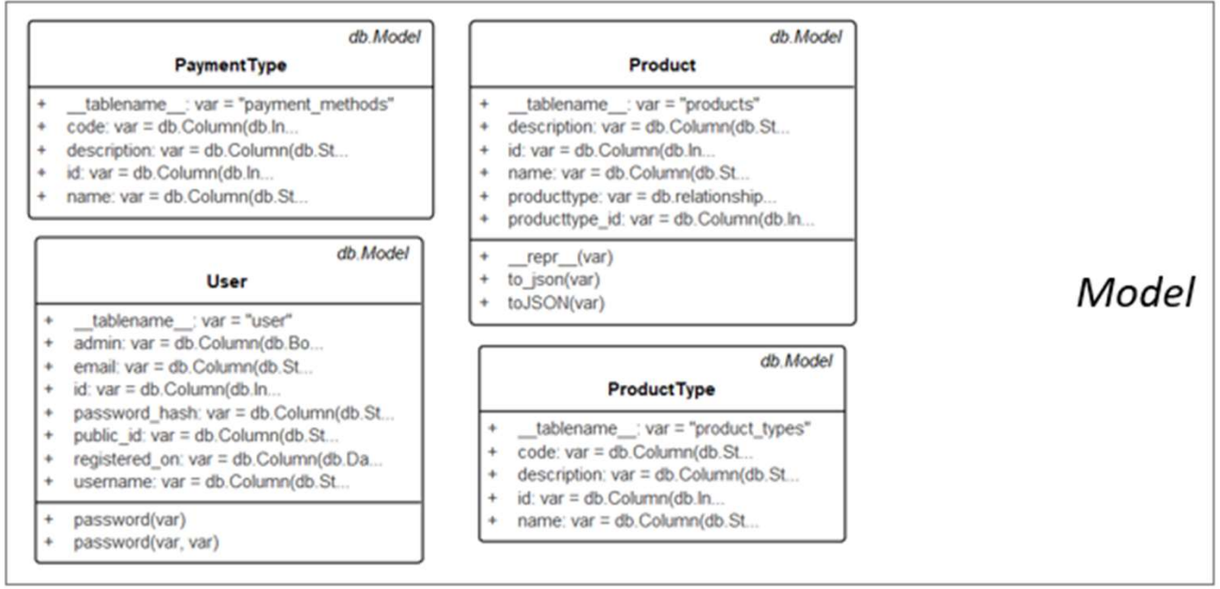
```python
#route returning Products list
@cross_origin(**api_v2_cors_config)
def getProductsList():
    product = ProductController()
    return product.getAllProducts()
```

Routes

ProductController

- __init__(var)
- getAllProducts(var)
- getAllProductsByType(var, var)
- obj_dict(var, var)

Controllers



Model

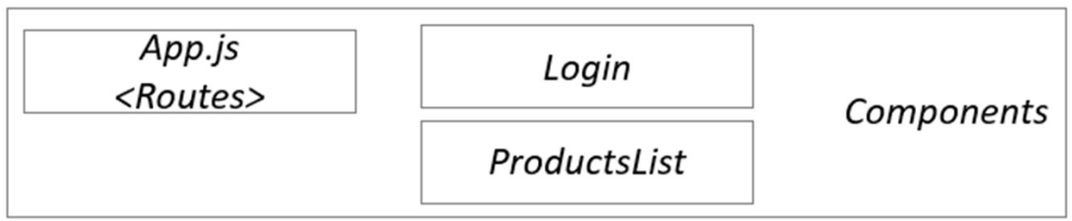
Routes define the service endpoints. They are linked to a URL in manage.py using add_url_rule app.add_url_rule('/api/producttypes', view_func=routes.getProductTypesList)

Controllers expose the methods which can be exposed as service end points. They are invoked from route.

Model holds Object Relational Mapping to Database. These are made serializable to JSON using @dataclass attribute and defining the attributes (like id:int in ProductType)

Architecture – Layered View

Architecture| Front End

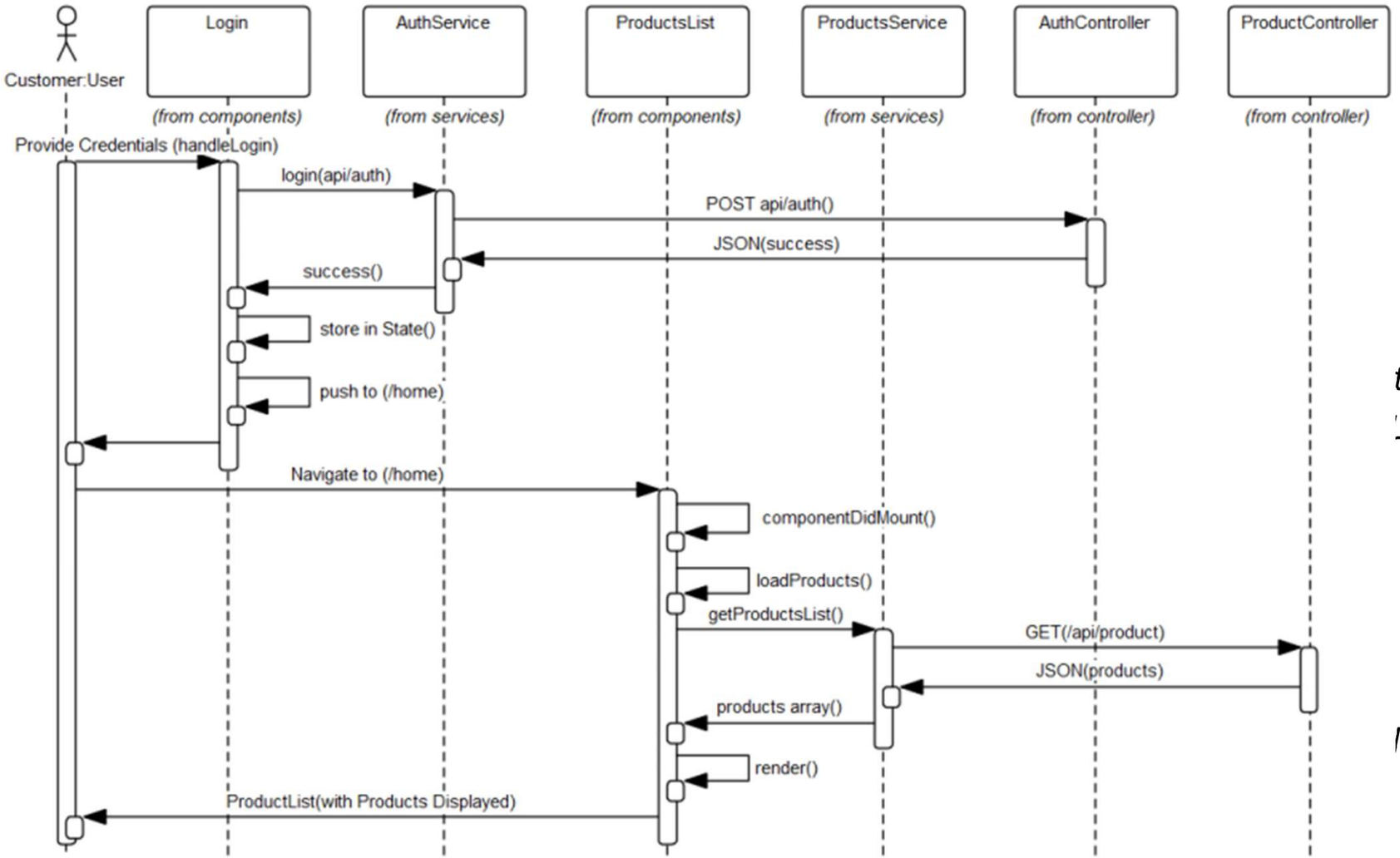


Components are the way react organizes elements –Login, ProductsList are examples. They are defined as routes in App.js

Services make call to APIs and return response

Architecture – Layered View

Architecture| Interactions

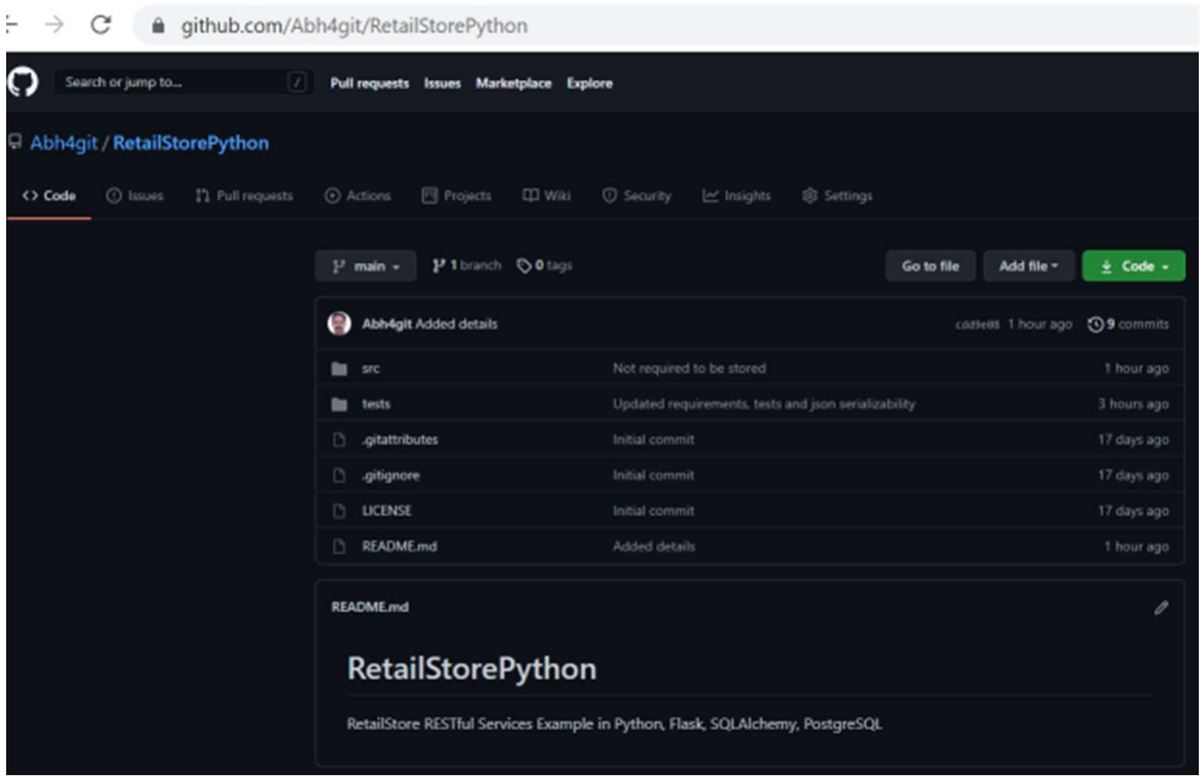


the way react organizes elements
List are examples. They are
in App.js

ll to APIs and return response

Architecture – Sequence Diagram

Getting Started| Back End

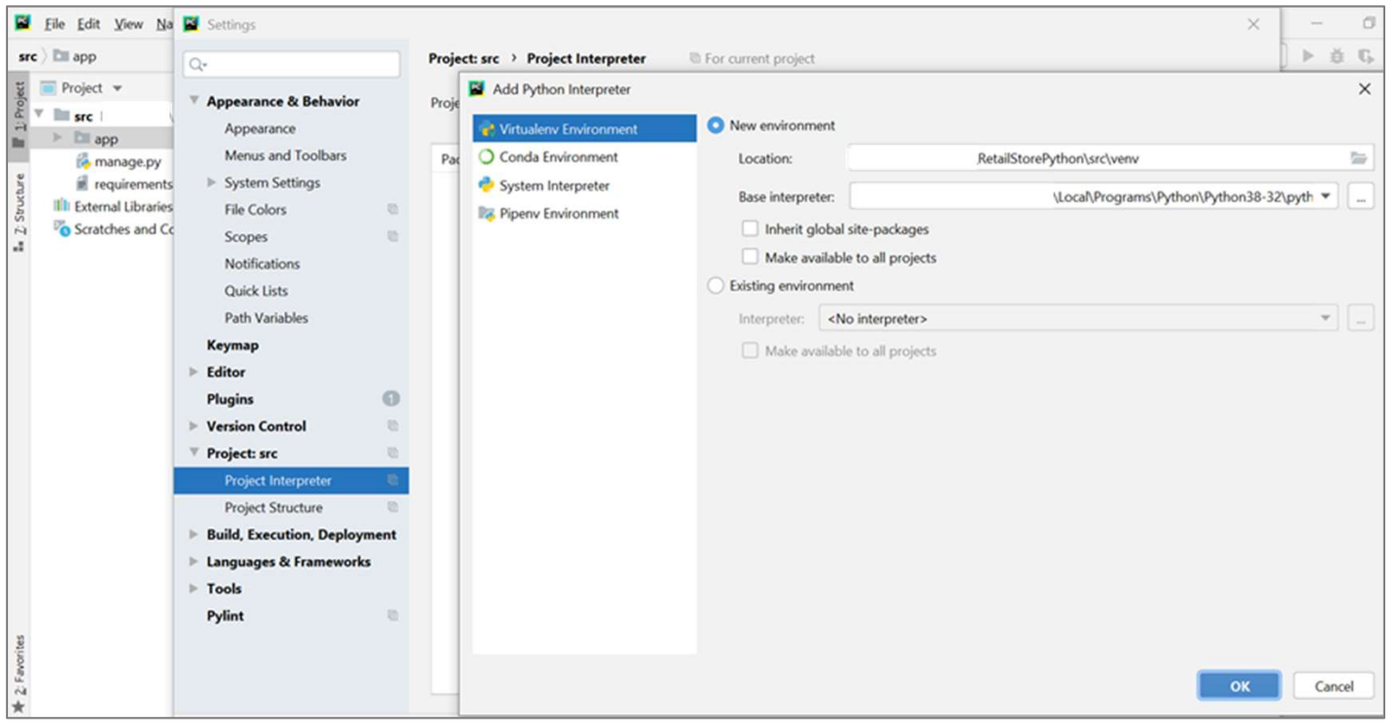


GitHub URL:

https://github.com/Abh4git/RetailStorePython

GitHub Repo for Back End

Getting Started| Back End



1. *Open PyCharm in src folder and setup Interpreter and select Virtual Environment*
2. *Pip install as shown below:*

```
src>pip install -r requirements.txt
```

3. Set FLASK_APP environment variable like below

src> **set FLASK_APP=manage.py** *(In Linux use export)*

Back End running

Getting Started| Back End - Database

```
basedir = os.path.abspath(os.path.dirname(__file__))
class Config:
    SECRET_KEY = os.getenv('SECRET_KEY', 'my_precious_secret_key')
    DEBUG = False
class DevelopmentConfig(Config):
    # uncomment the line below to use postgres
    SQLALCHEMY_DATABASE_URI ='postgresql://localhost/retailstoredb?user=postgres&password=[redacted]'
    SQLALCHEMY_TRACK_MODIFICATIONS = True
```

```
Terminal:  Local  +
(venv) D:\Abhilash\GitHub\RetailStorePython\src>flask db migrate -m "Added table ProductTypes"
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.autogenerate.compare] Detected added table 'product_types'
INFO  [alembic.ddl.postgresql] Detected sequence named 'user_id_seq' as owned by integer column 'user(id)', assur
Generating D:\Abhilash\GitHub\RetailStorePython\src\migrations\versions\be483fd19b64_added_table_producttypes.py
```

```
(venv) D:\Abhilash\GitHub\RetailStorePython\src>flask db upgrade
INFO  [alembic.runtime.migration] Context impl PostgresqlImpl.
INFO  [alembic.runtime.migration] Will assume transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade 28619cbdb4d0 -> be483fd19b64, Added table ProductTypes
```

1. *Check your database URI is updated with your username and password*
2. *Run Db Init to start migration like below:*

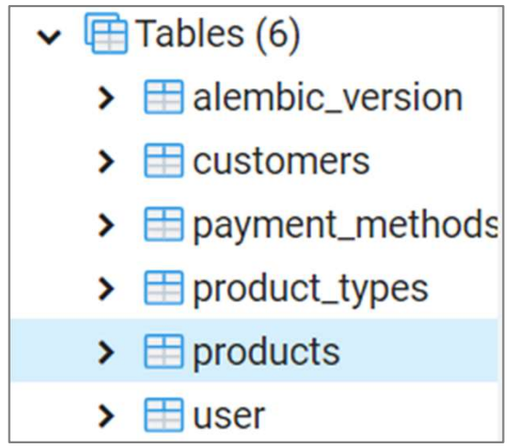
src> **flask db init**

3. *Run DB Migrate*

src> **flask db migrate –m “Added tables”**

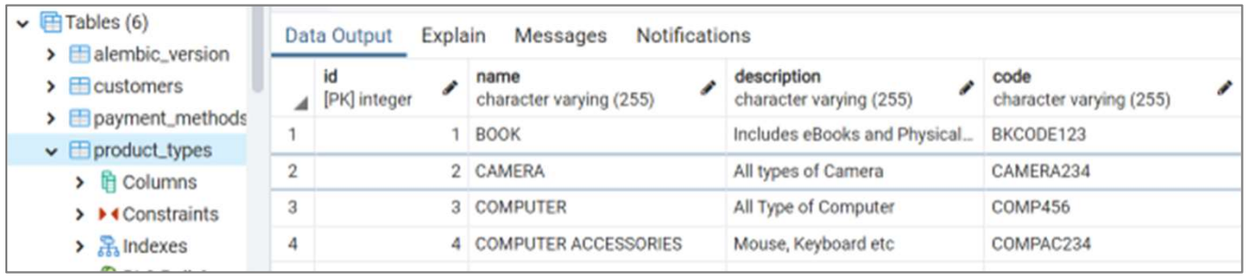
4. *Run DB Upgrade*

src> **flask db upgrade**

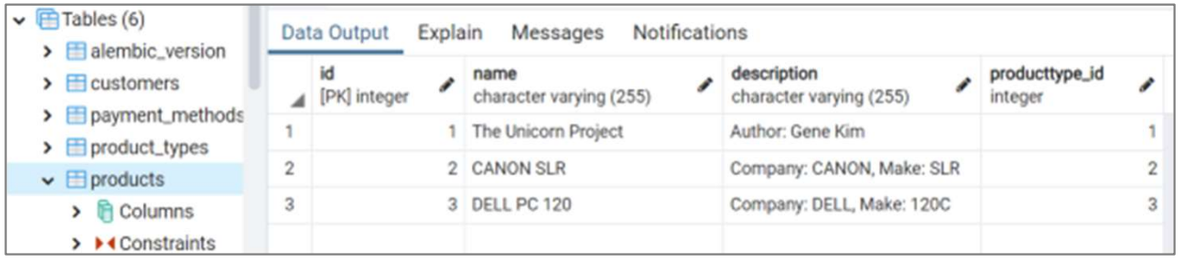


After step4, you should find all tables in database

Getting Started| Back End - Database



1. *Add entries in product_types table*



2. *Add entries in products table*

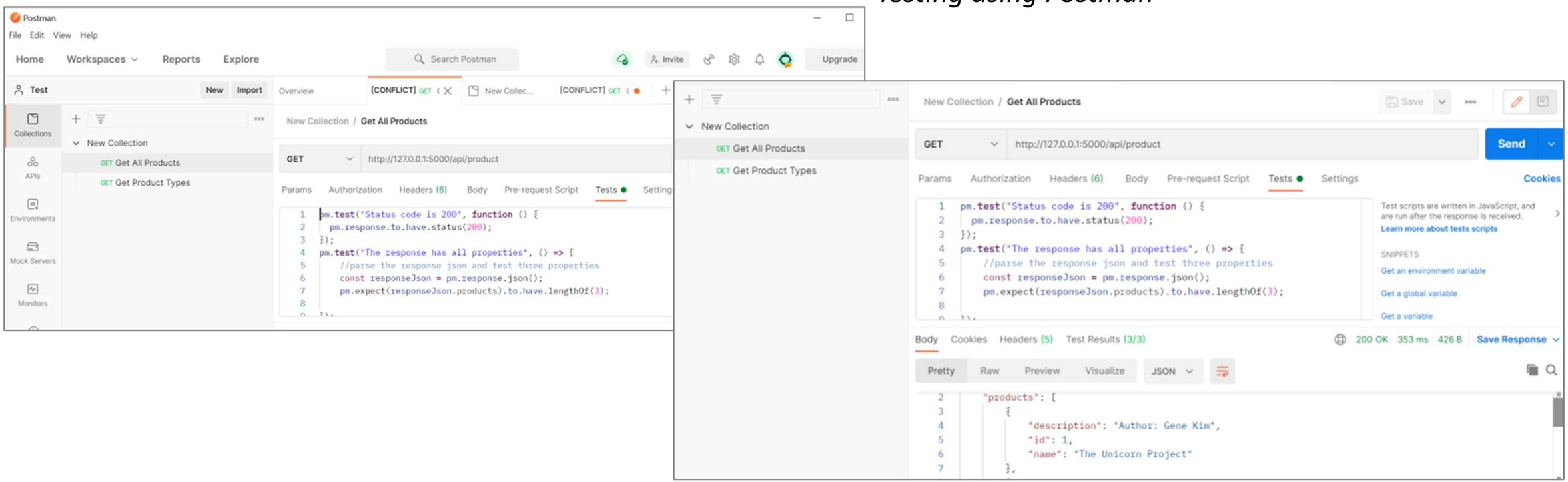
Getting Started| Running Back End and Testing it

```
(venv) D:\Abhilash\GitHub\RetailStorePython\src>flask run
 * Serving Flask app "manage.py"
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

1. *Start execution from terminal*

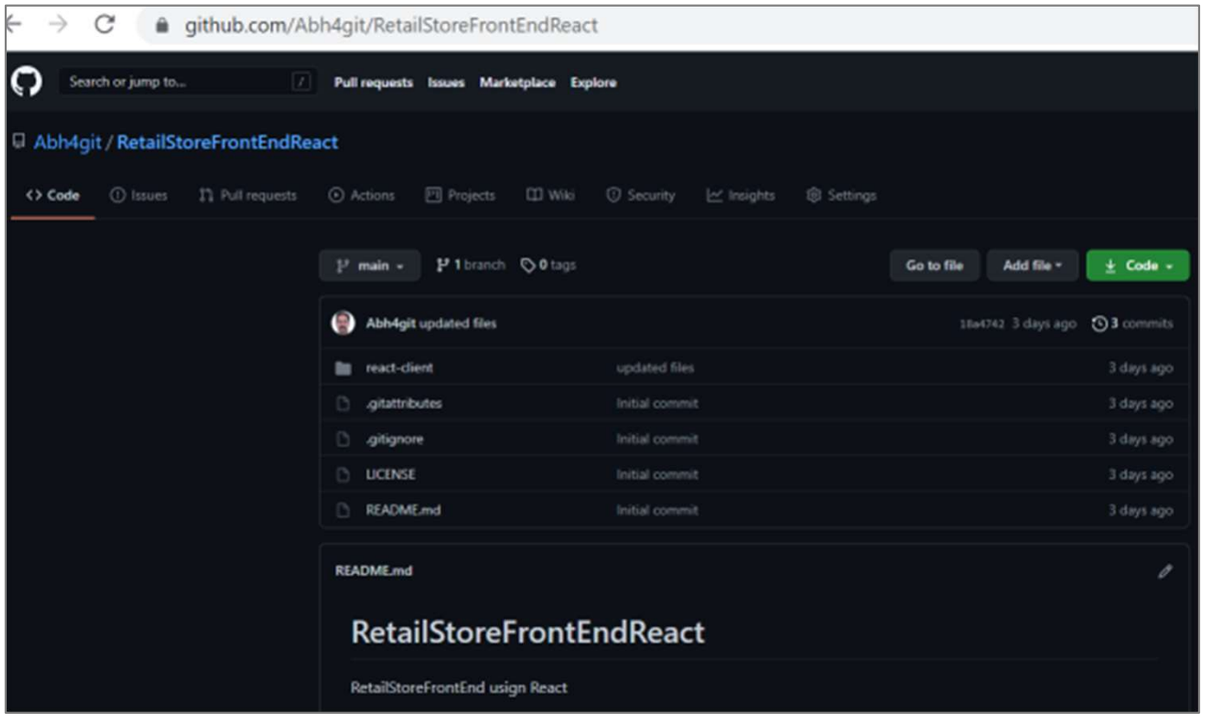
src> ***flask run***

Testing using Postman



The Test results show if the service is running fine.

Getting Started| Front End

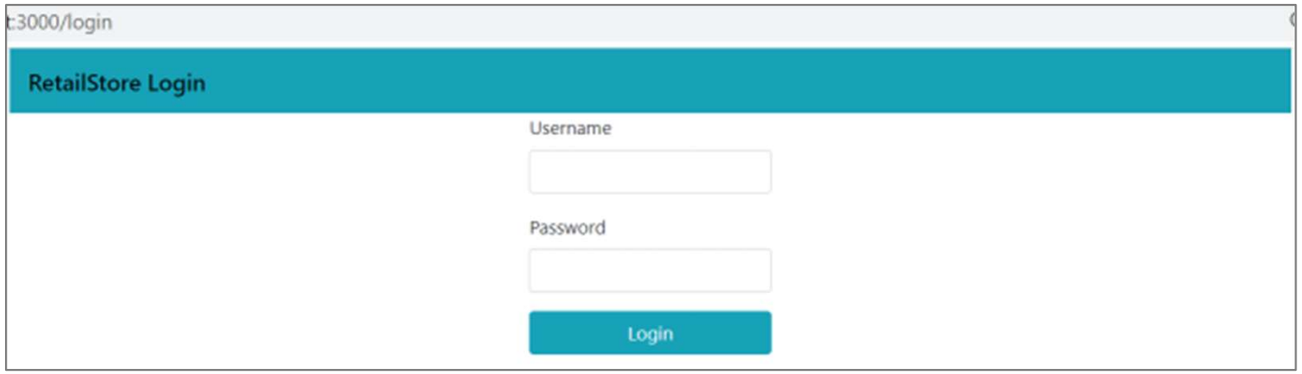


GitHub URL:

https://github.com/Abh4git/RetailStoreFrontEndReact

GitHub Repo –Front End

Getting Started| Front End

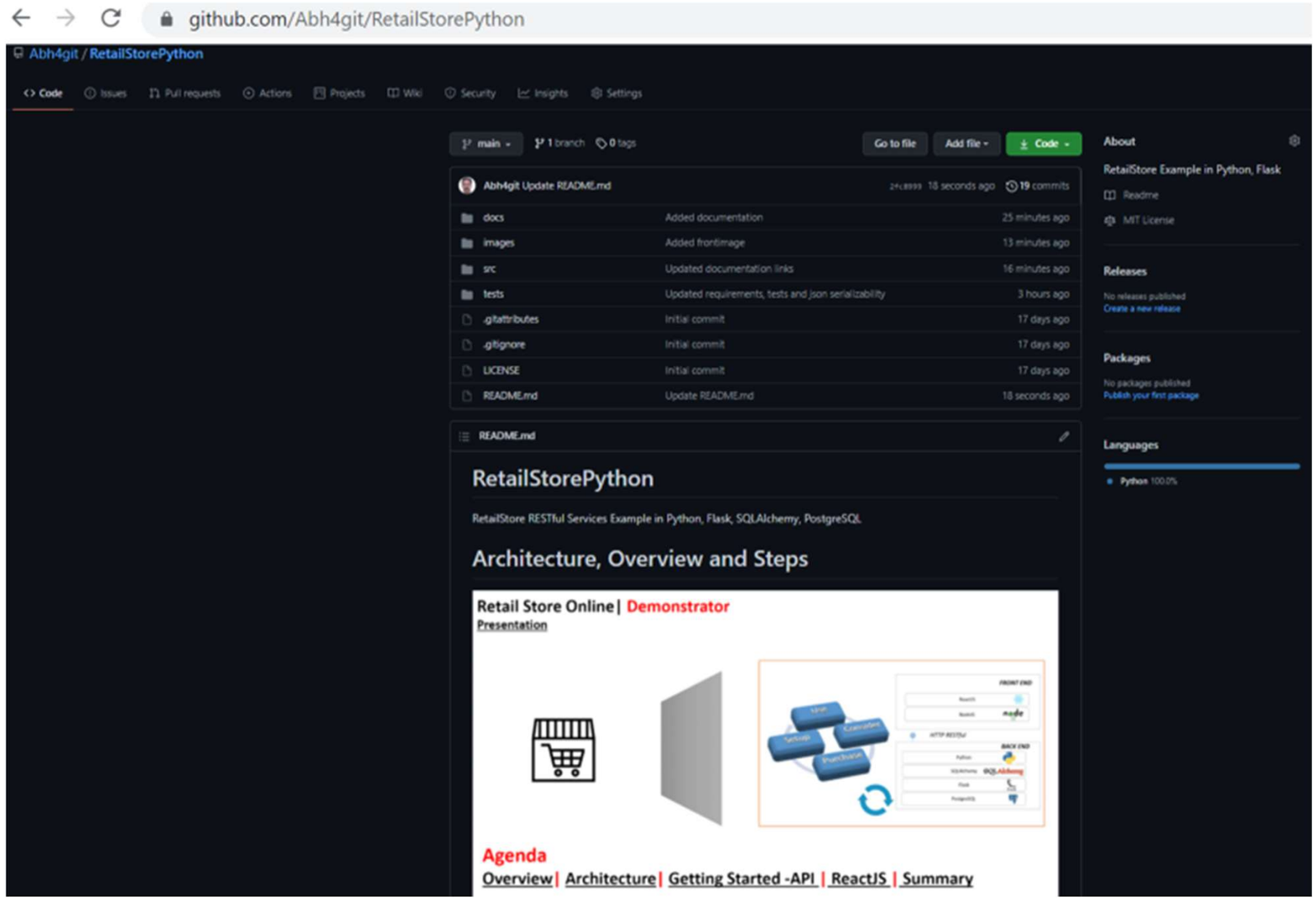


Pre-requisite: NodeJs need to be installed:

1. Install dependencies
*src> **npm install***
2. Start program
*src> **npm start***

Executing the Front End

Summary| Architecture, Design and Getting it running



1. *Back End using Python, Flask, SQLAlchemy and PostgreSQL*
2. *Front End using NodeJS, ReactJS*
3. *Testing using Postman*
4. *Step by step approach explained.*

Thank you!