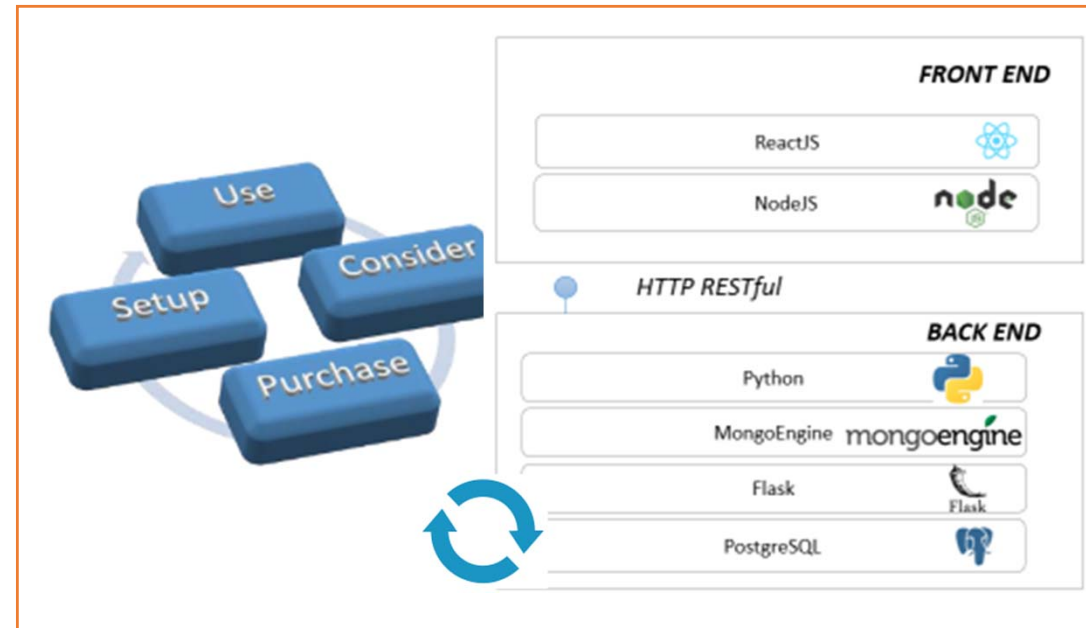


# Retail Store Online | Demonstrator

## Presentation



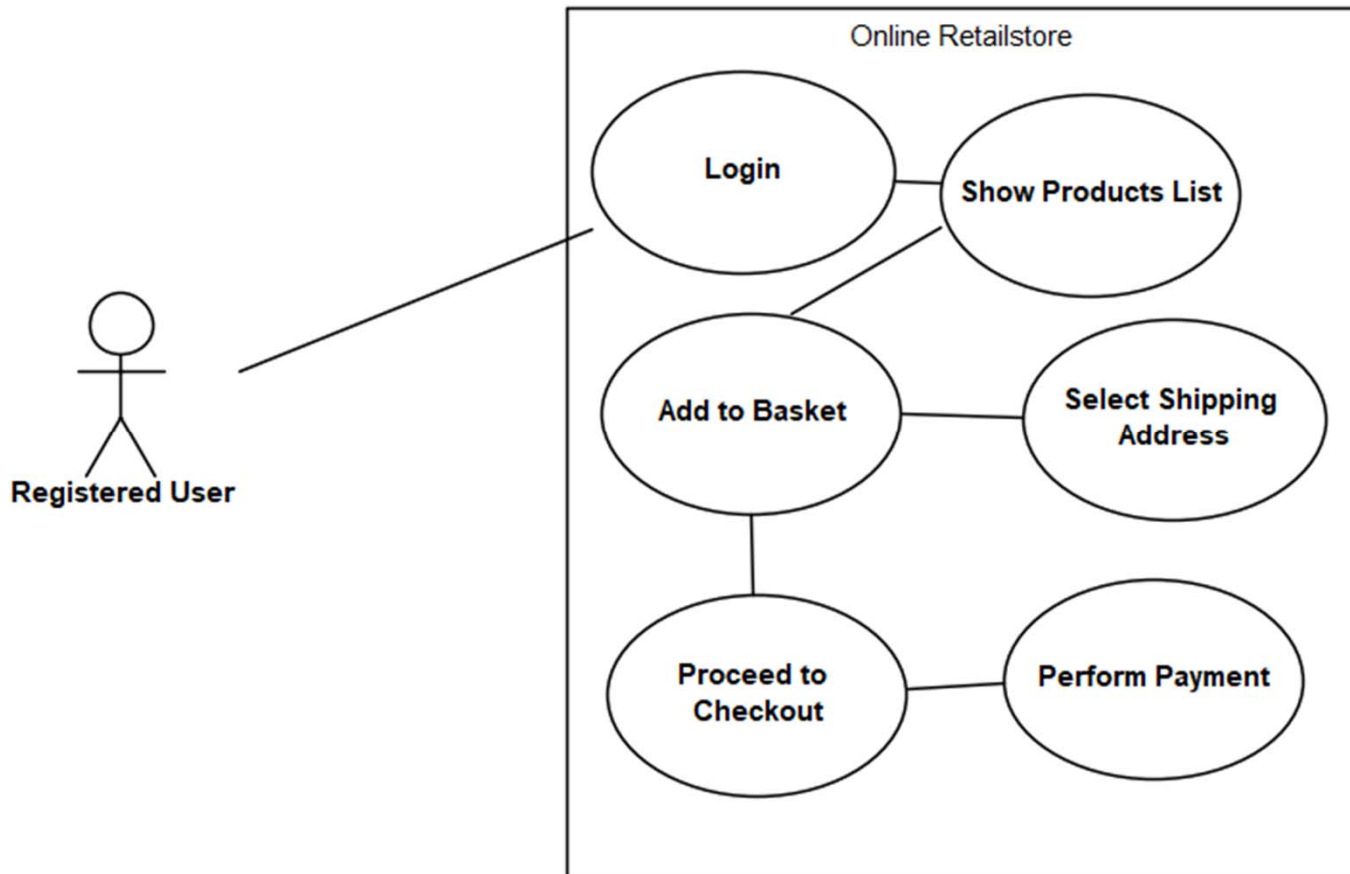
*Using MongoDB (Document Database)*

## Agenda

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

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

# Overview | Use case/Context



## First set of Features

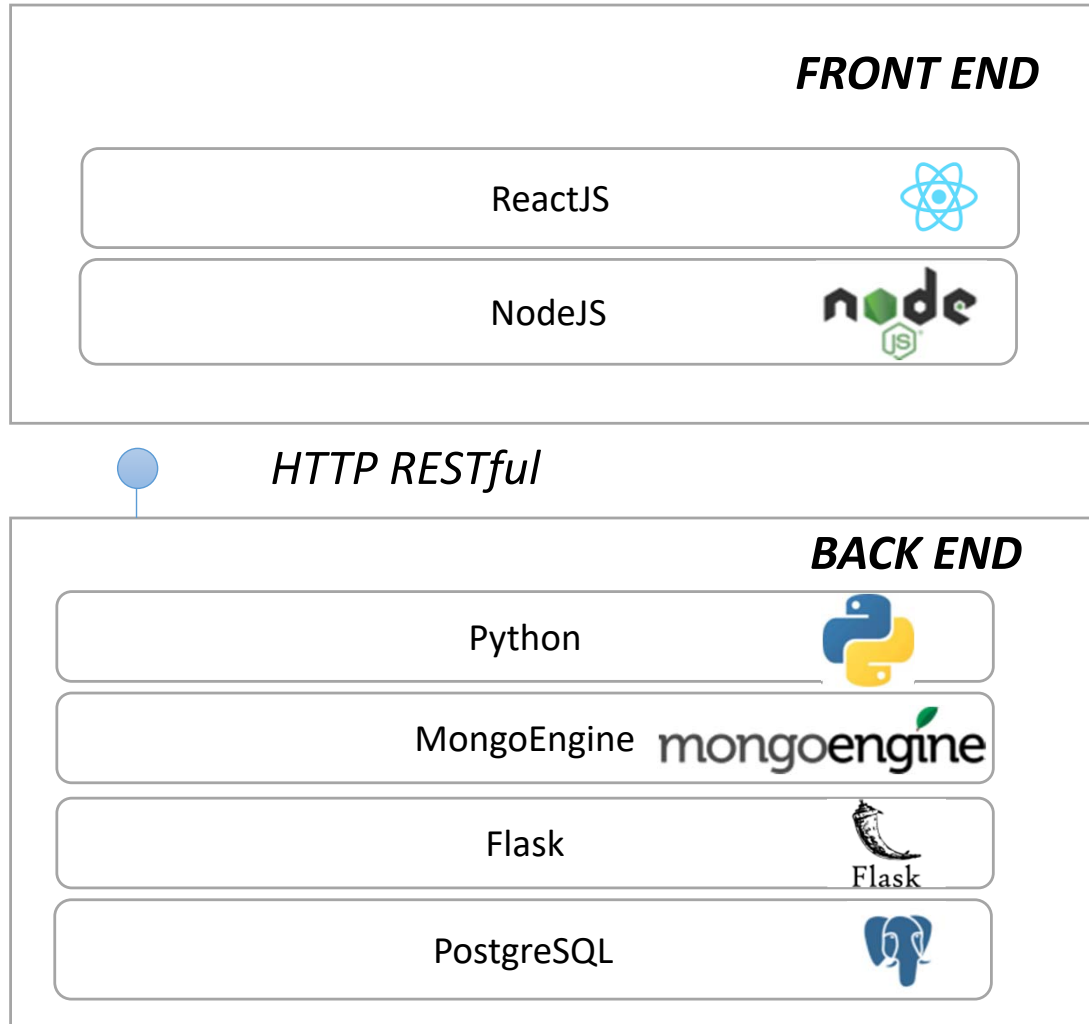
1. User should be able to **login and see a list of products**, select from there, select one of the defined addresses and do payment and see the order in order list.
2. Delivery happens separate as Shipped by the Seller and once Delivered it is updated in the System.

## 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.

# Overview | Architecture



*The Front End is built using ReactJS*



*Service/API Testing is done using Postman*

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

*Object Document Mapping MongoEngine with MongoDB <http://mongoengine.org/>*

# Architecture | Back End

```
#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

## Customer

db.Document

```
+ address_line1: var = db.StringField(...)
+ address_line2: var = db.StringField(...)
+ city: var = db.StringField(...)
+ country: var = db.StringField(...)
+ email: var = db.StringField(...)
+ firstname: var = db.StringField(...)
+ id: var = db.IntField(primary_key=True)
+ lastname: var = db.StringField(...)
+ meta: var = {'collection': ...}
+ paymenttype_id: var = db.IntField()
+ phone: var = db.StringField(...)
+ profile: var = db.StringField(...)
```

## PaymentType

db.Document

```
+ code: var = db.IntField()
+ description: var = db.StringField(...)
+ id: var = db.IntField(primary_key=True)
+ meta: var = {'collection': ...}
+ name: var = db.StringField(...)
```

db.Document  
ProductType

db.Document  
Product

db.Document  
User

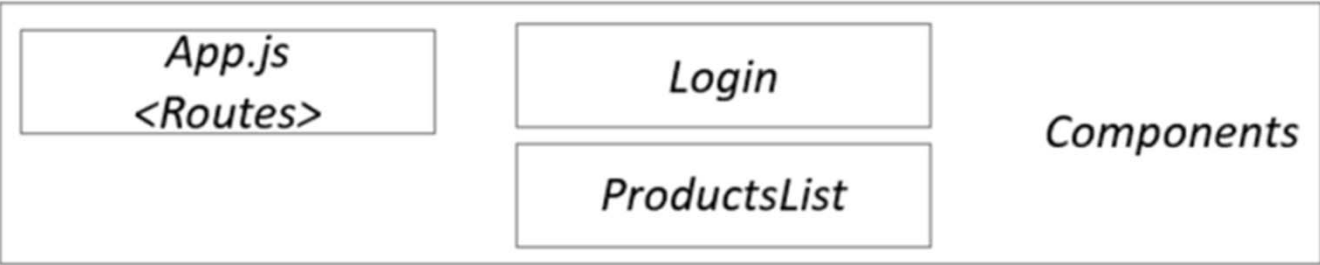
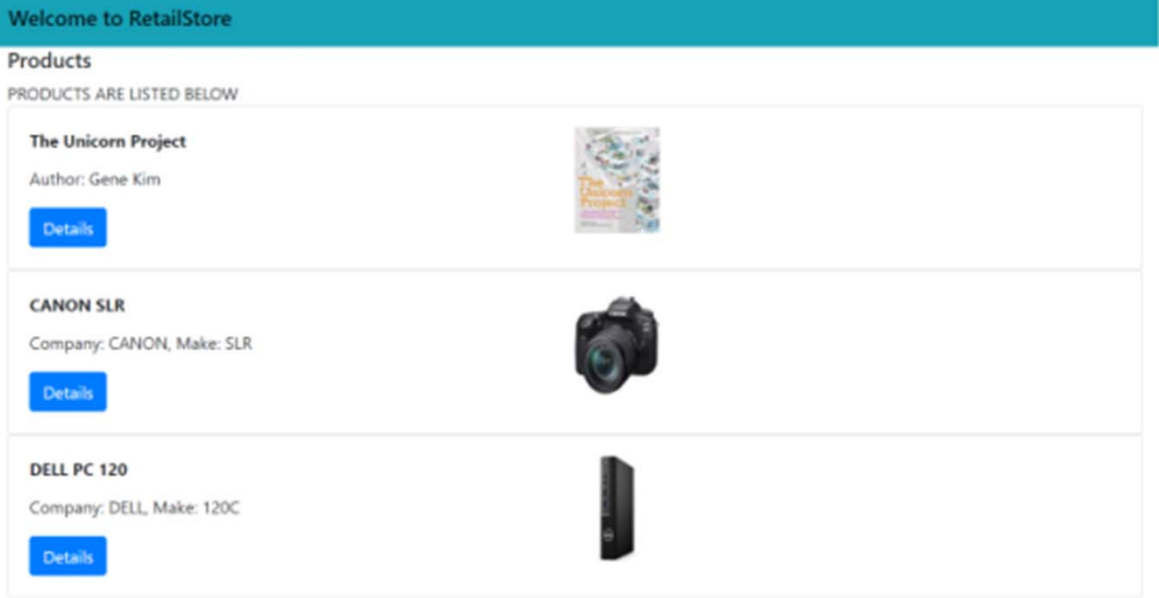
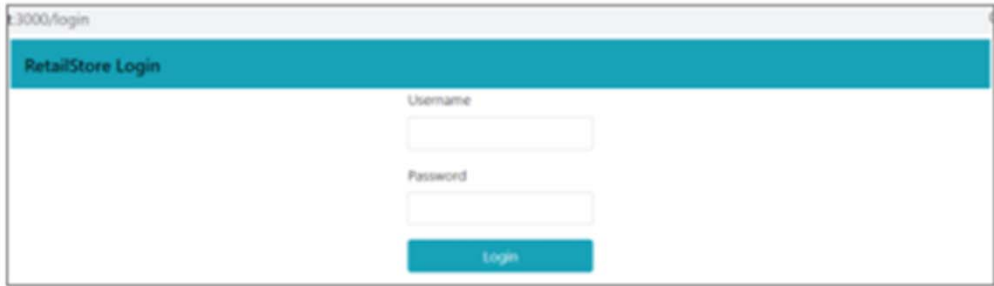
Model

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

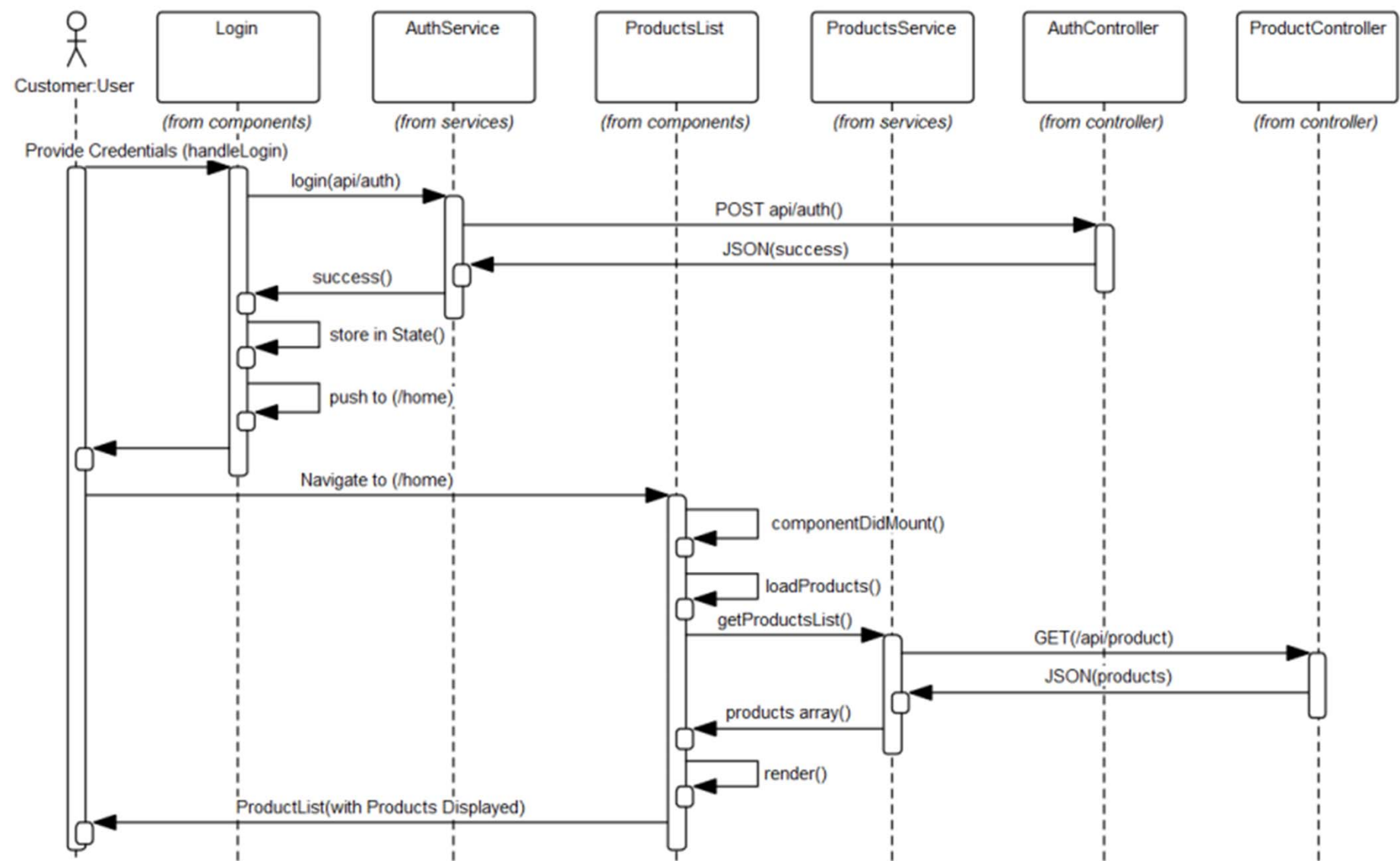
# Architecture | Front End



*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 | Interactions

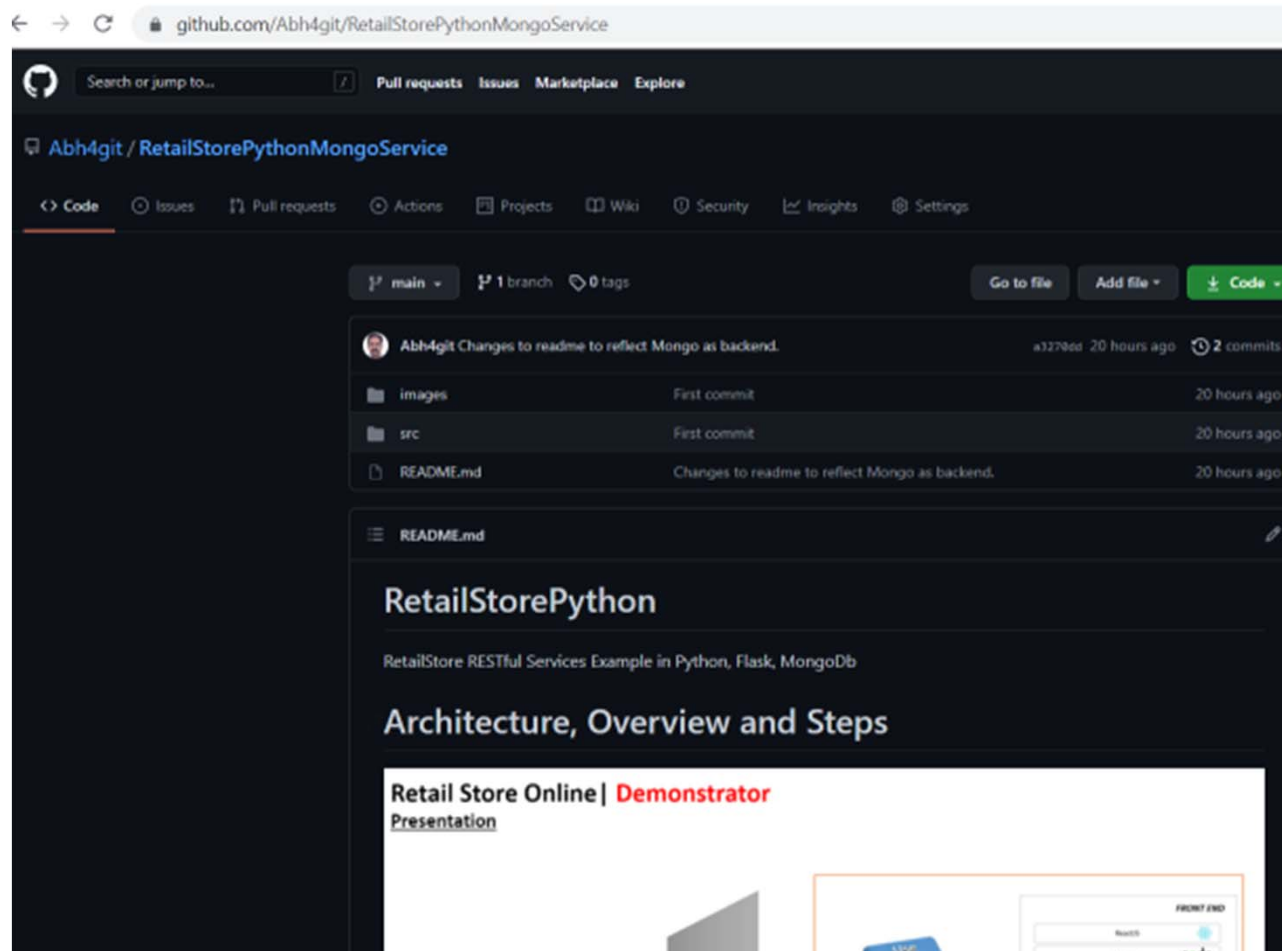


*the way react organizes elements  
list are examples. They are  
in App.js*

*// to APIs and return response*

Architecture – Sequence Diagram

# Getting Started | **Back End**

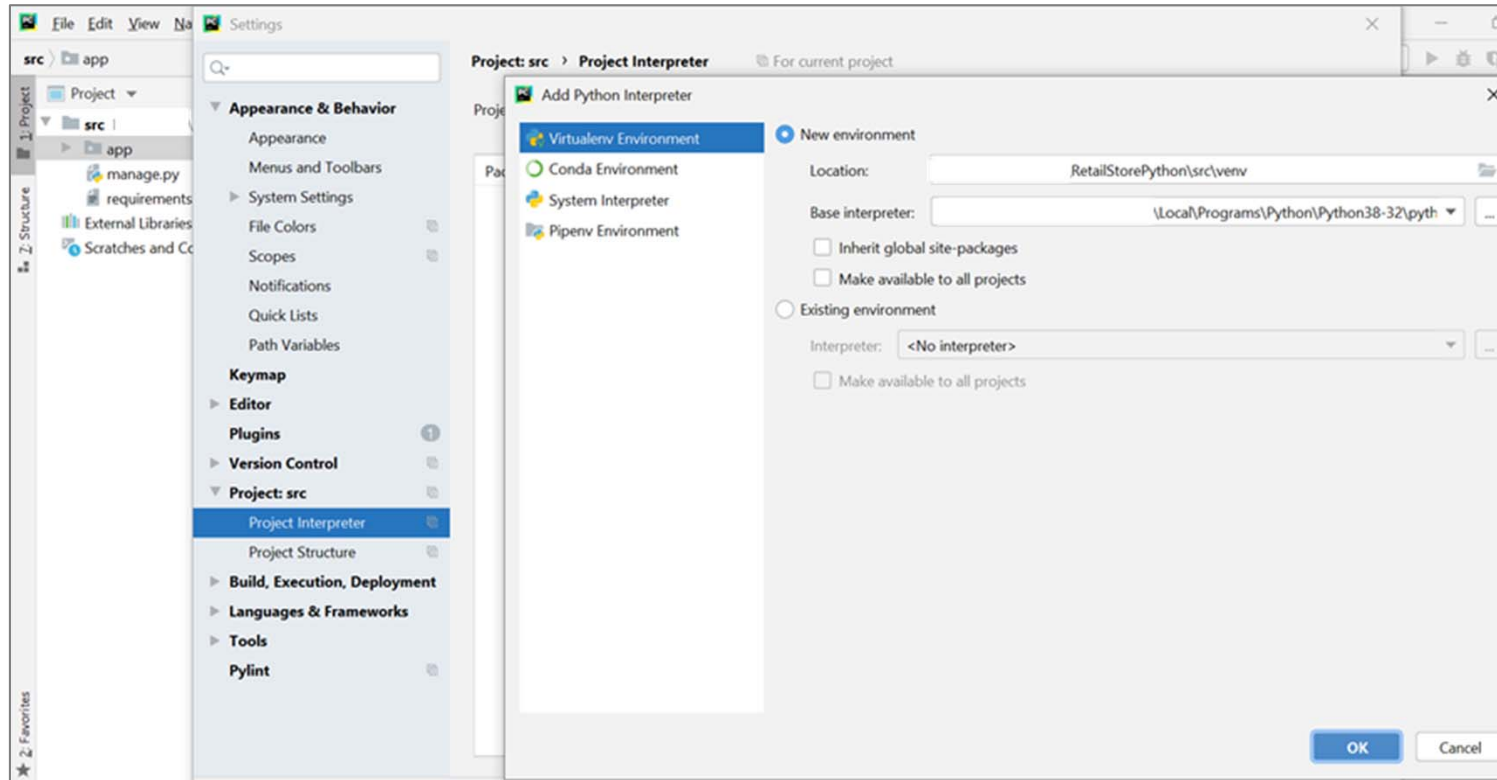


*GitHub URL:*

<https://github.com/Abh4git/RetailStorePythonMongoService>

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

1. *If the database already exists, drop it*

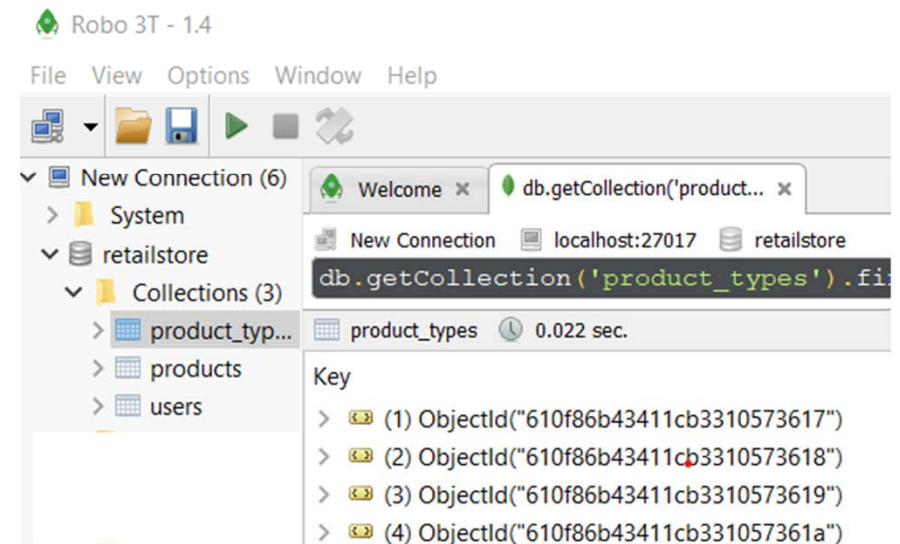
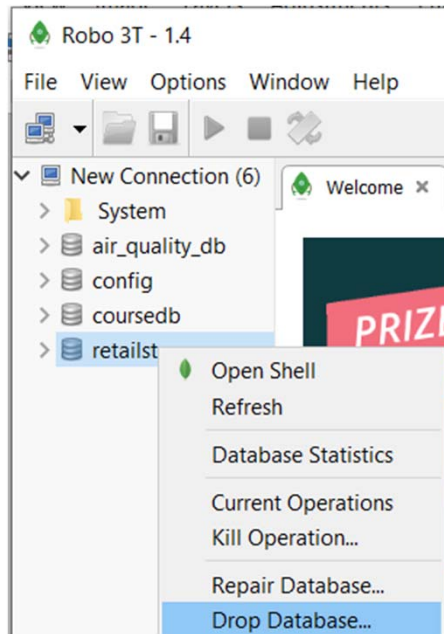
2. *Execute the script in src folder*

*mongo localhost:27017 mongodbinit.js*

```
D:\GitHub\PythonMongoService\src>mongo localhost:27017 mongodbinit.js
MongoDB shell version v4.4.6
connecting to: mongodb://localhost:27017/test?compressors=disabled&gssapiServiceName=
Implicit session: session { "id" : UUID("47e4cdb8-6ab1-4602-83bc-d7ef754b4e0a") }
MongoDB server version: 4.4.6
```

*No errors, then database, tables and data should be created*

app  
venv  
manage.py  
mongodbinit.js



Back End Database

# 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 screenshot displays the Postman application interface. On the left, the 'Test' tab is active, showing a collection named 'Get All Products' with two requests: 'GET Get All Products' and 'GET Get Product Types'. The main panel shows the 'GET' request to 'http://127.0.0.1:5000/api/product'. The 'Tests' tab is selected, displaying the following JavaScript test scripts:

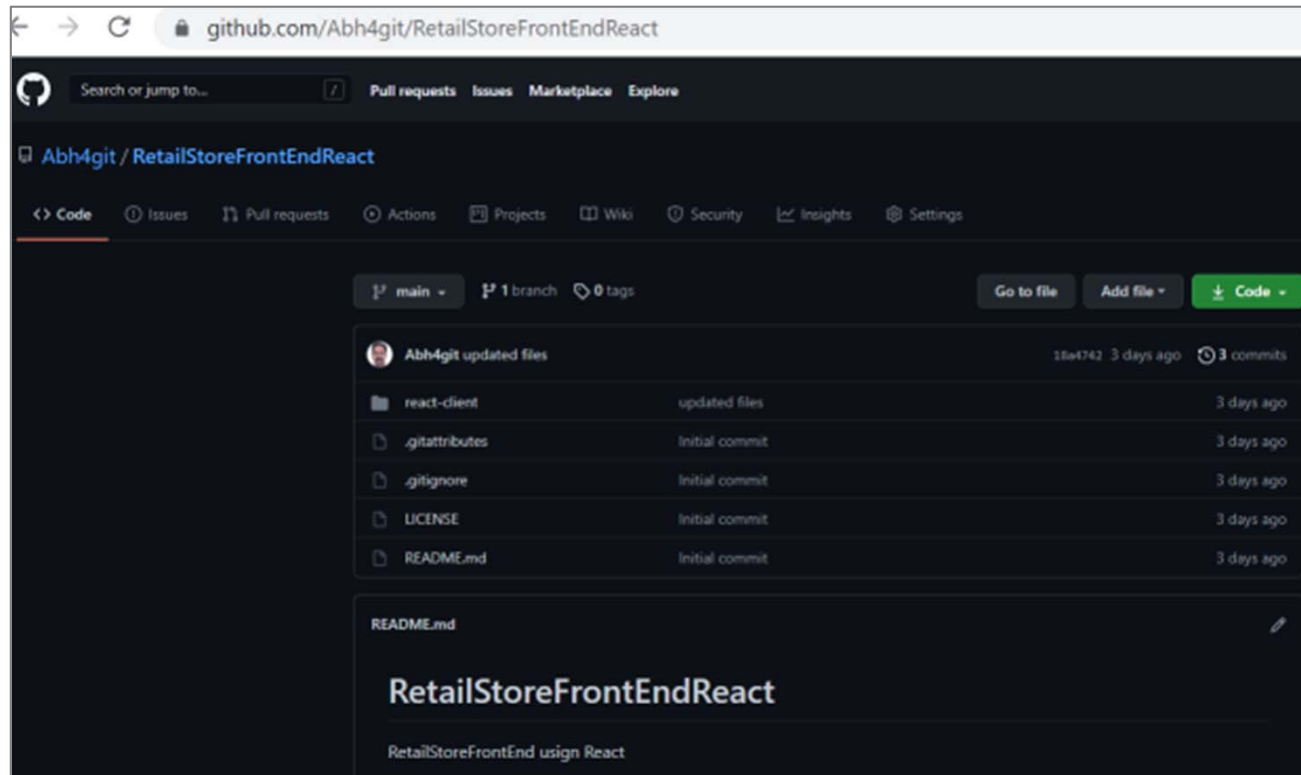
```
1 pm.test("Status code is 200", function () {
2   pm.response.to.have.status(200);
3 });
4 pm.test("The response has all properties", () => {
5   //parse the response json and test three properties
6   const responseJson = pm.response.json();
7   pm.expect(responseJson.products).to.have.lengthOf(3);
8 })
```

The 'Send' button has been clicked, and the response is displayed in the 'Body' tab. The status is '200 OK' with a response time of '353 ms' and a size of '426 B'. The JSON response is:

```
{
  "products": [
    {
      "description": "Author: Gene Kim",
      "id": 1,
      "name": "The Unicorn Project"
    }
  ]
}
```

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***

st:3000/login

**RetailStore Login**

Username

Password

Login

st:3000/home

**Welcome to RetailStore**

**Products**

#	NAME	DESCRIPTION	
1	The Unicorn Project	Author: Gene Kim	
2	CANON SLR	Company: CANON, Make: SLR	
3	DELL PC 120	Company: DELL, Make: 120C	

Executing the Front End

# Summary | Architecture, Design and Getting it running

github.com/Abh4git/RetailStorePythonMongoService

README.md

## RetailStorePython

RetailStore RESTful Services Example in Python, Flask, MongoDB

### Architecture, Overview and Steps

#### Retail Store Online | Demonstrator Presentation

Using MongoDB (Document Database)

#### Agenda

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

For Details, click on Link below  
!Architecture Overview  
[https://github.com/Abh4git/RetailStorePythonMongoService/blob/main/docs/Retail\\_Store\\_Online\\_Demo.pdf](https://github.com/Abh4git/RetailStorePythonMongoService/blob/main/docs/Retail_Store_Online_Demo.pdf)  
Data is already as part of mongodbninit.js script

Packages

No packages published  
Publish your first package

Languages

Python 93.0%

JavaScript 7.0%

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

Thank you!