**Project with Microservices Architecture**

**Project Start Date: 18 Jan 2020**

**http://sushantc.com**

***Legends****: COMPLETED, IN-PROGRESS*

***Application running on registered Domain****:* [*http://sushantc.com*](http://sushantc.com)

***Other Applications:***

*Jenkins =>* [*http://sushantc.com:8080*](http://sushantc.com:8080)

*Kibana =>* [*http://sushantc.com:5601*](http://sushantc.com:5601)

*Grafana =>* [*http://sushantc.com:3000*](http://sushantc.com:3000)

*Prometheus =>* [*http://sushantc.com:9090*](http://sushantc.com:9090)

*Eureka =>* [*http://sushantc.com:8012*](http://sushantc.com:8012)

**Progress:**

Spring 1 - Implement Basic Services using H2 database with basic REST endpoints

Added user, product, cart service *(**HATEOAS not for now)*

Sprint 2 - Configure service discovery, config server

Configured

Spring 3 - Implement basic UI using angular 8

*In progress...*

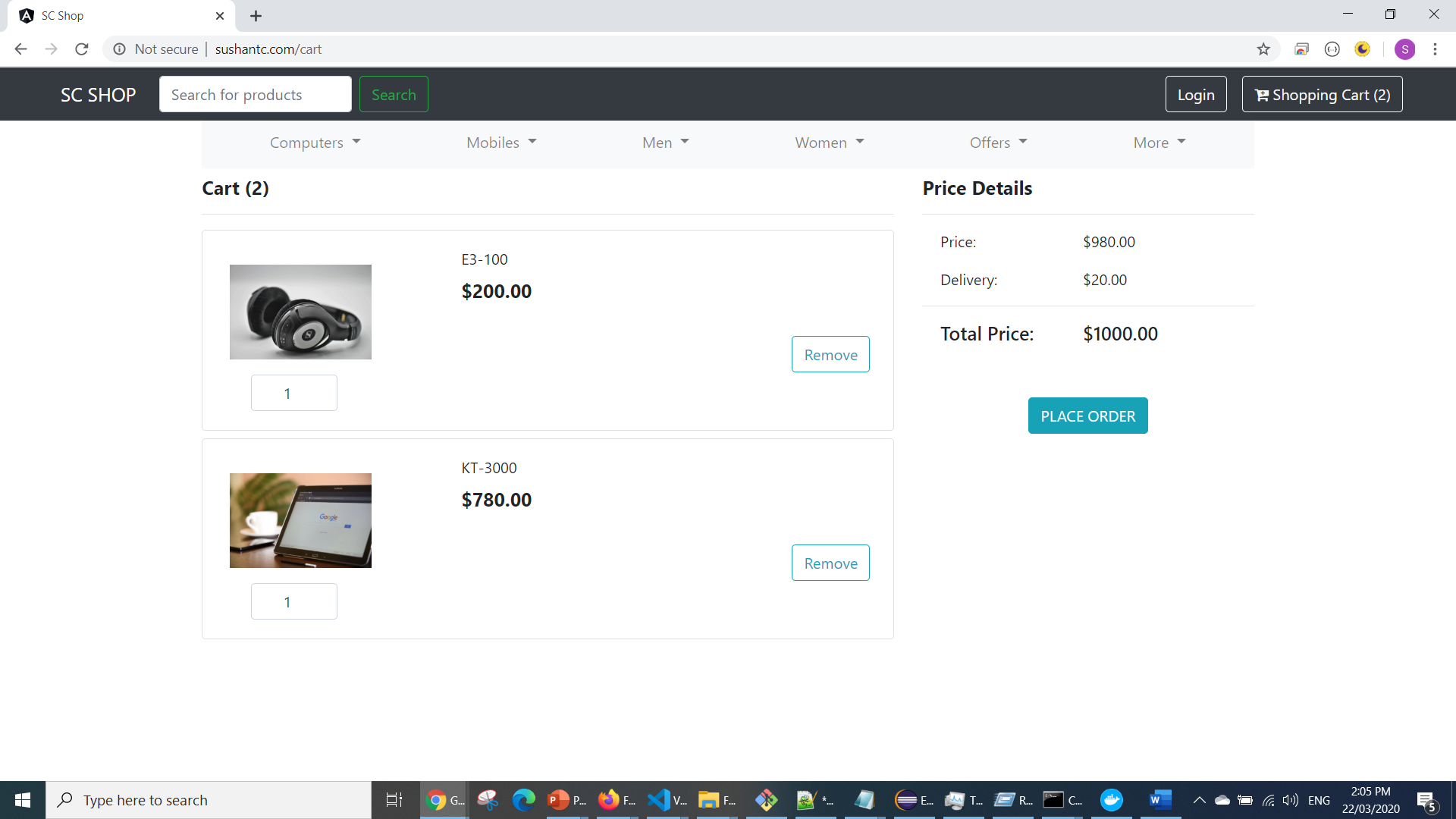
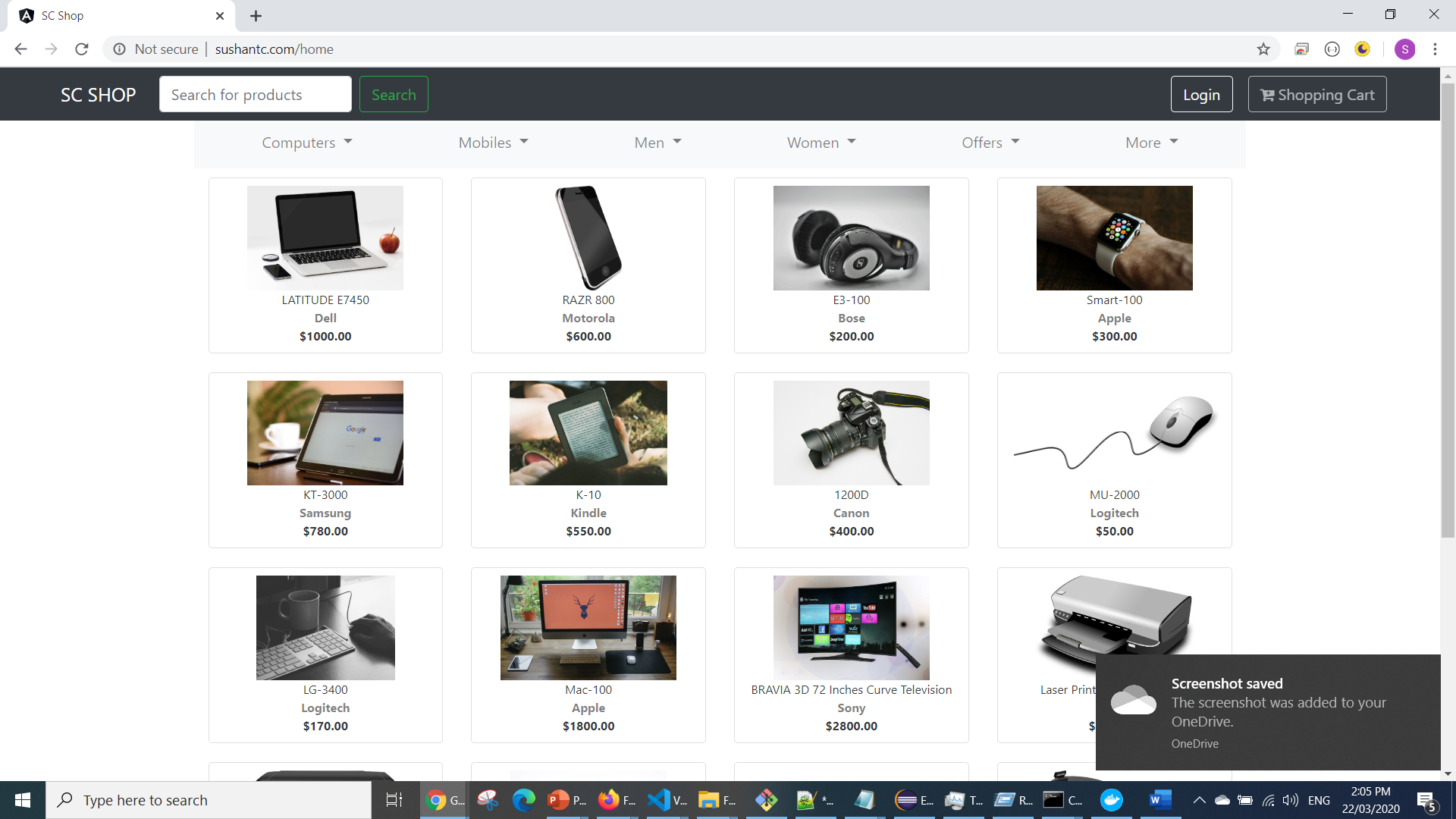
Created angular project

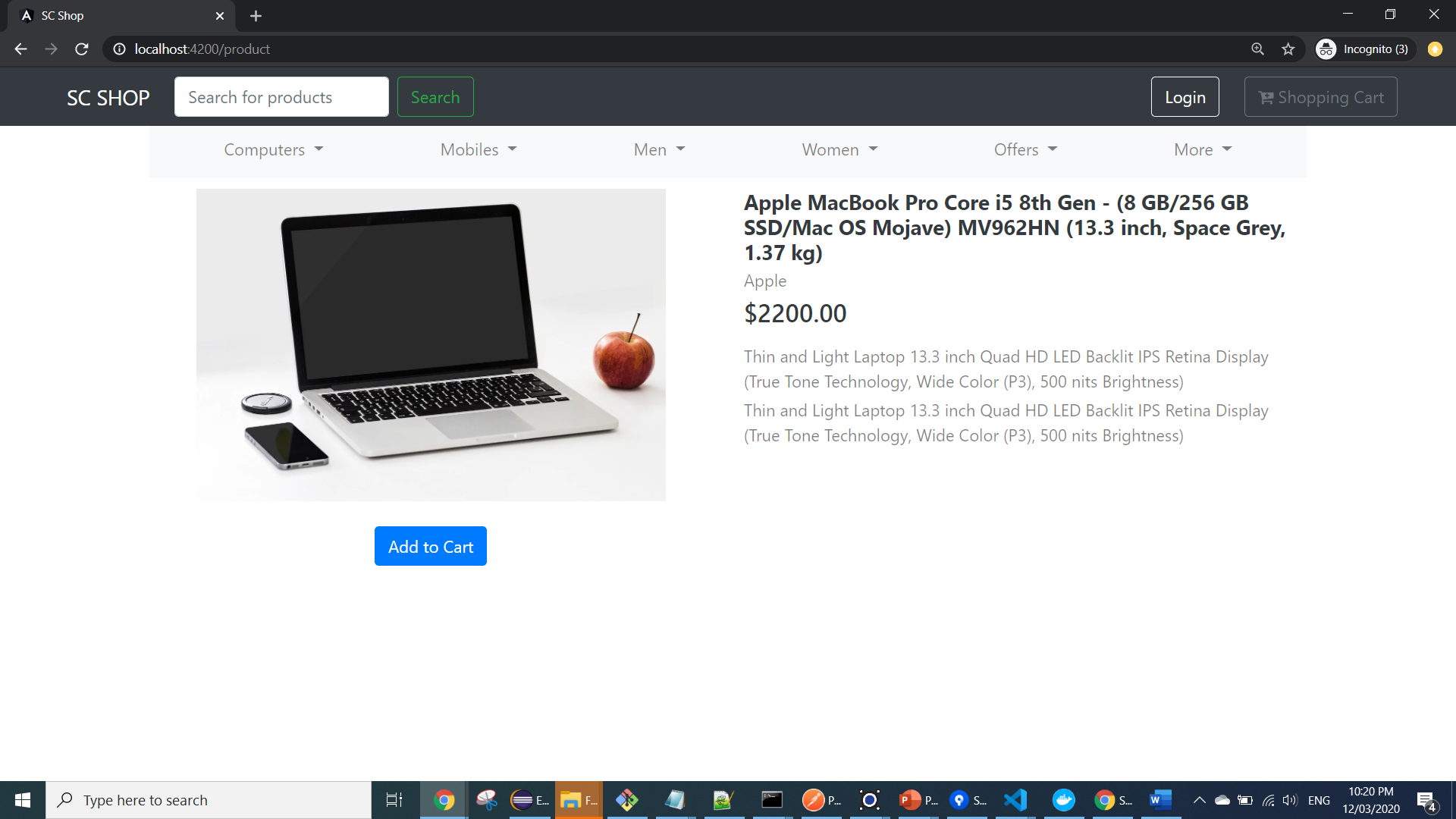
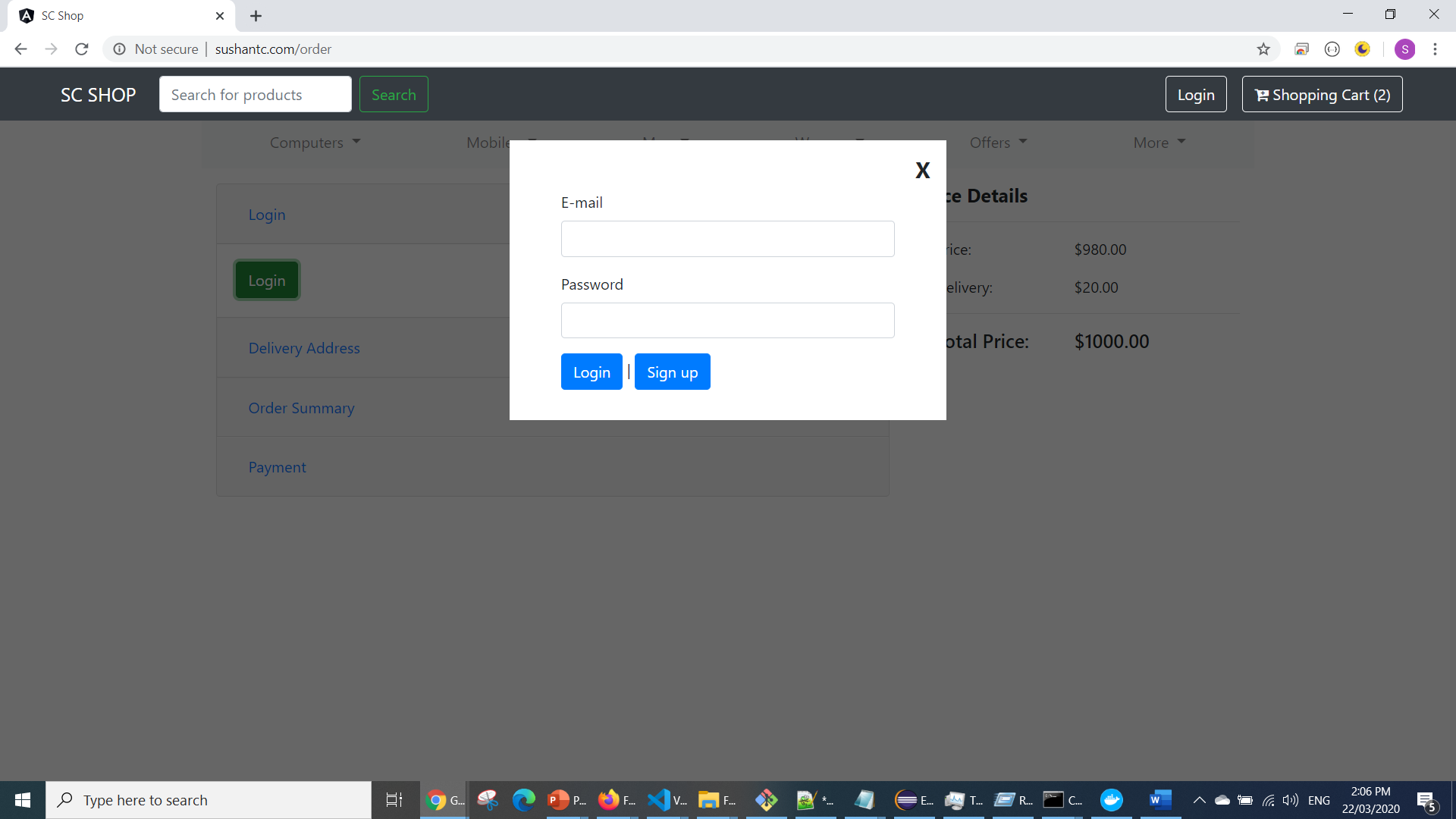
Added initial components

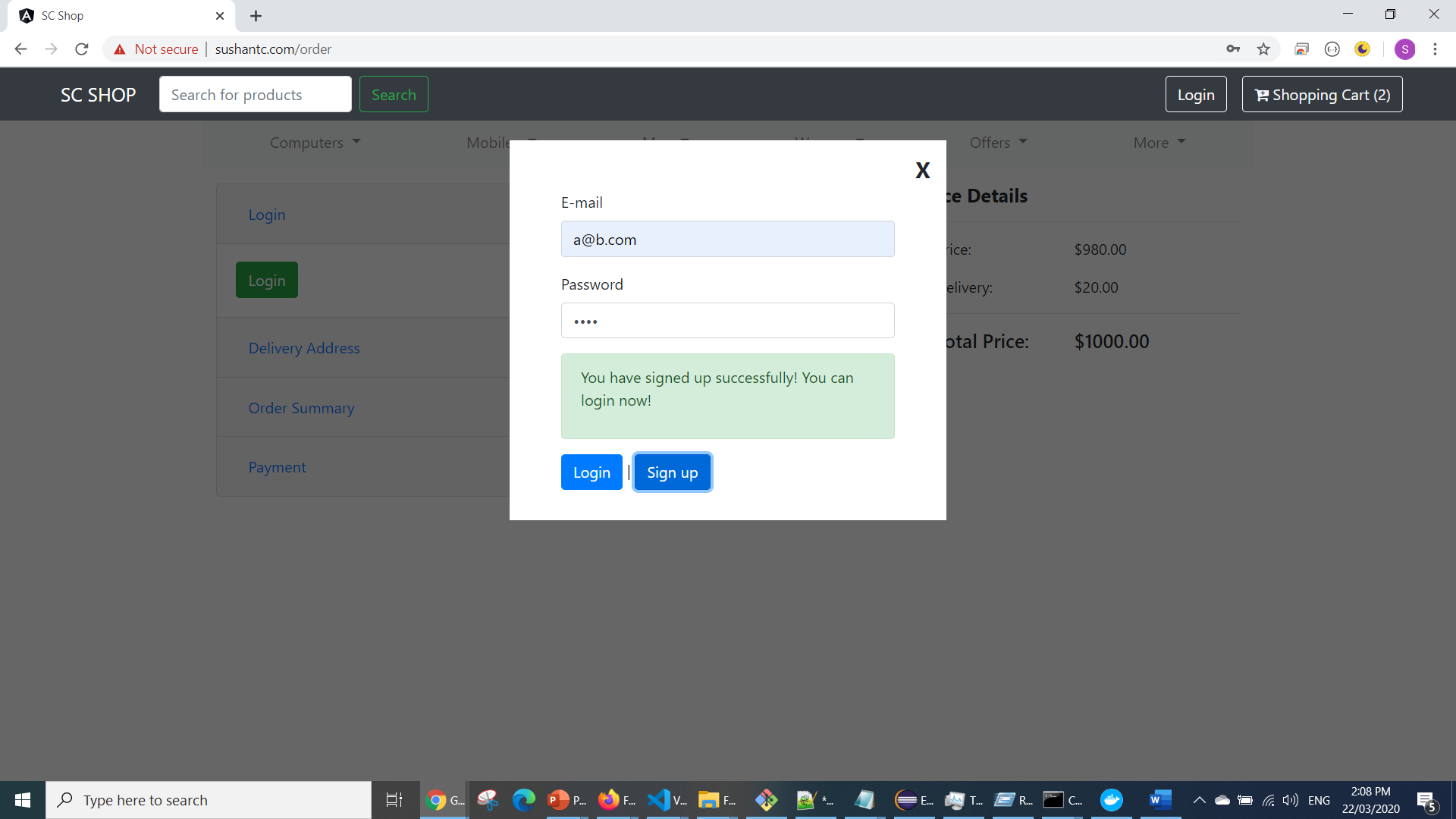
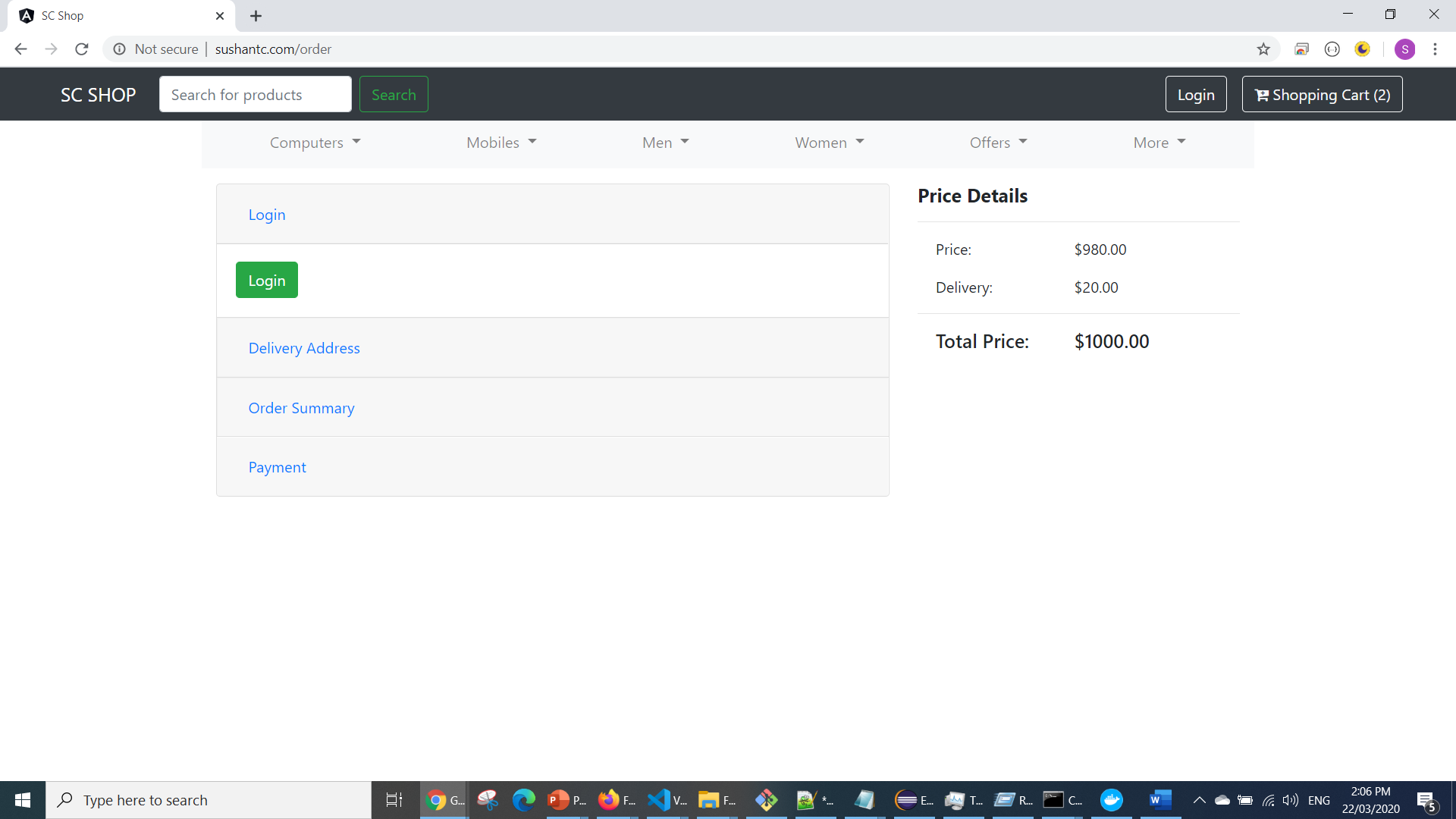
Added components – header, product-list, product pages

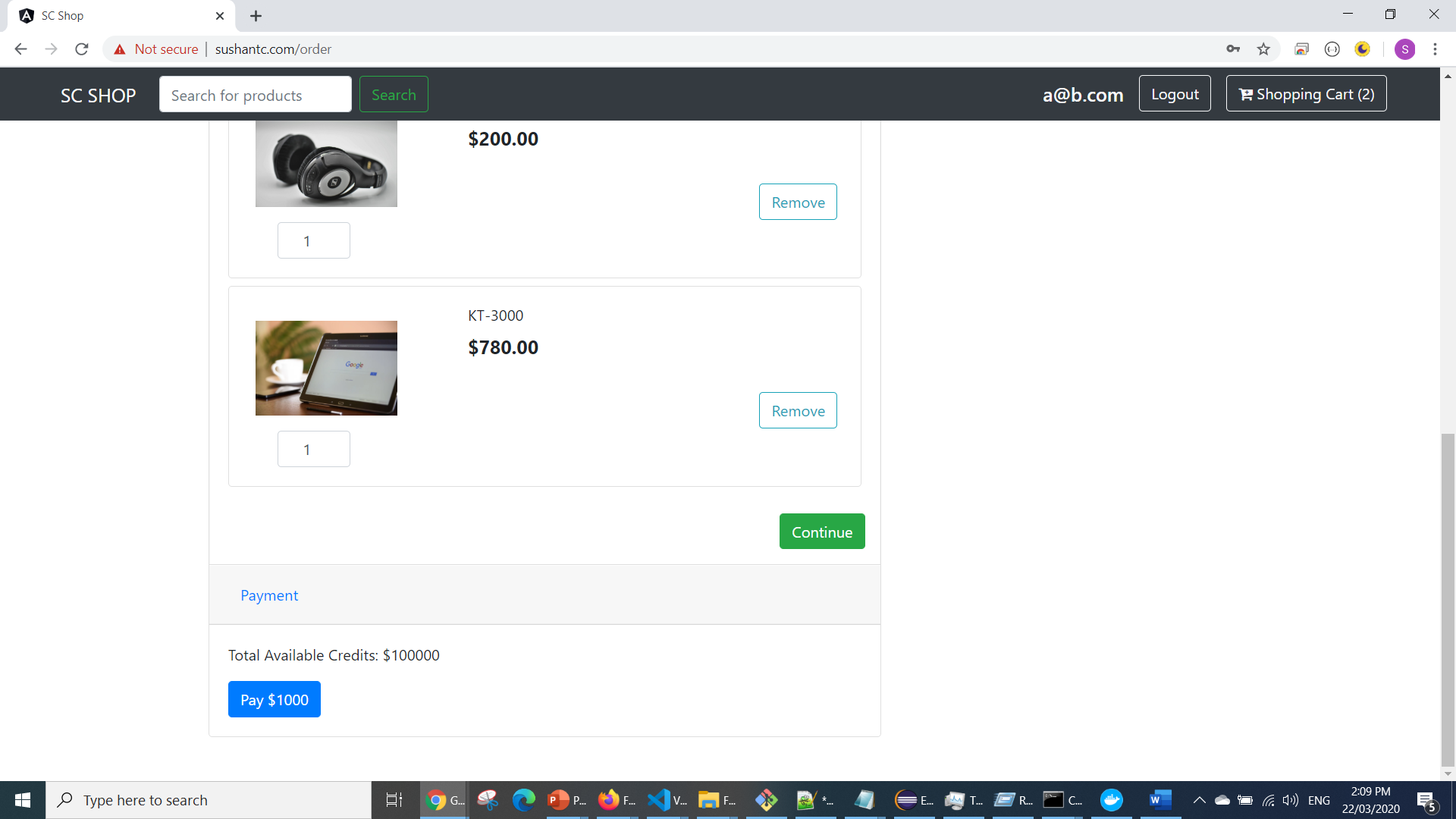
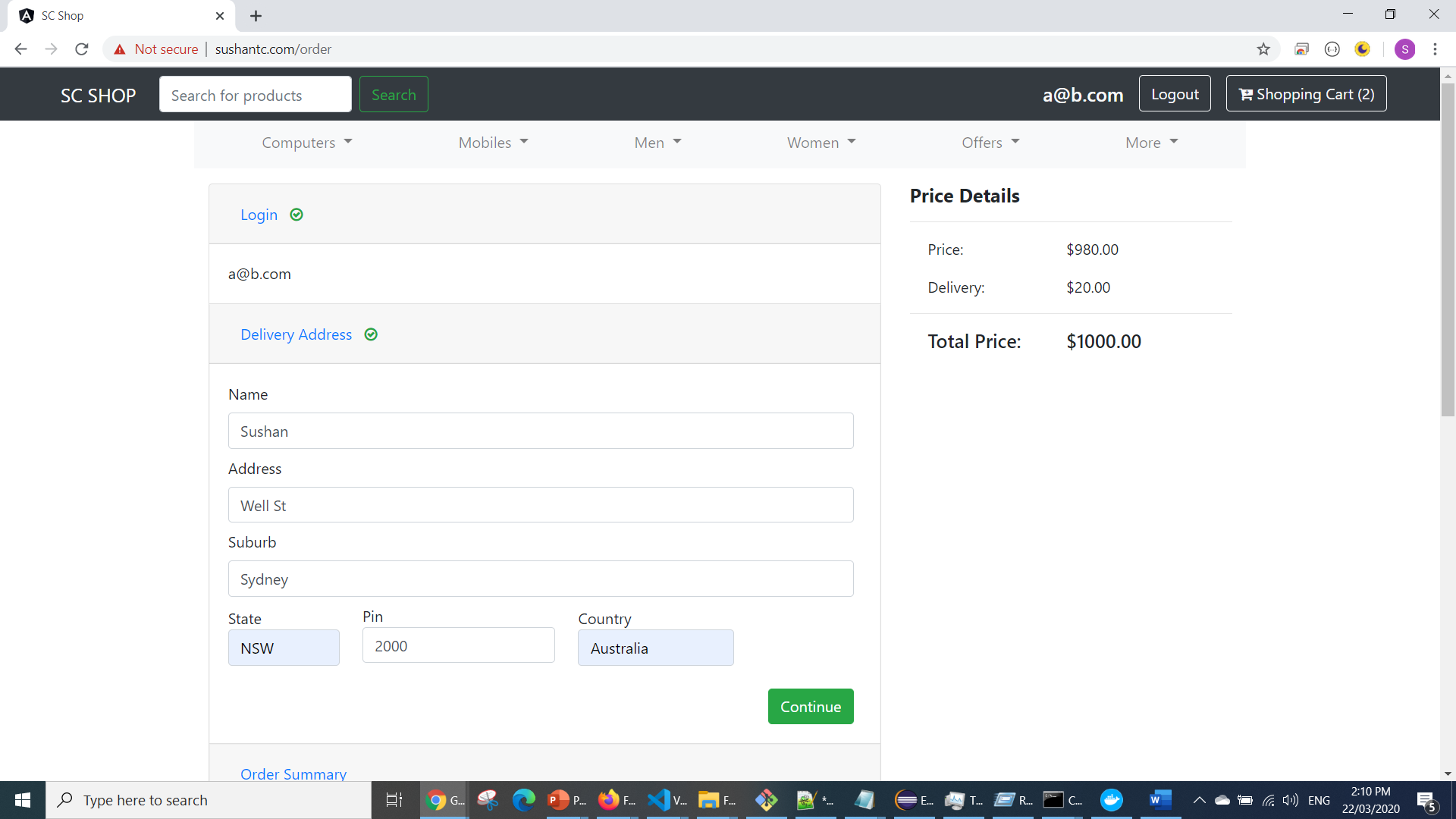
Added components –product, cart, order, login, order-success

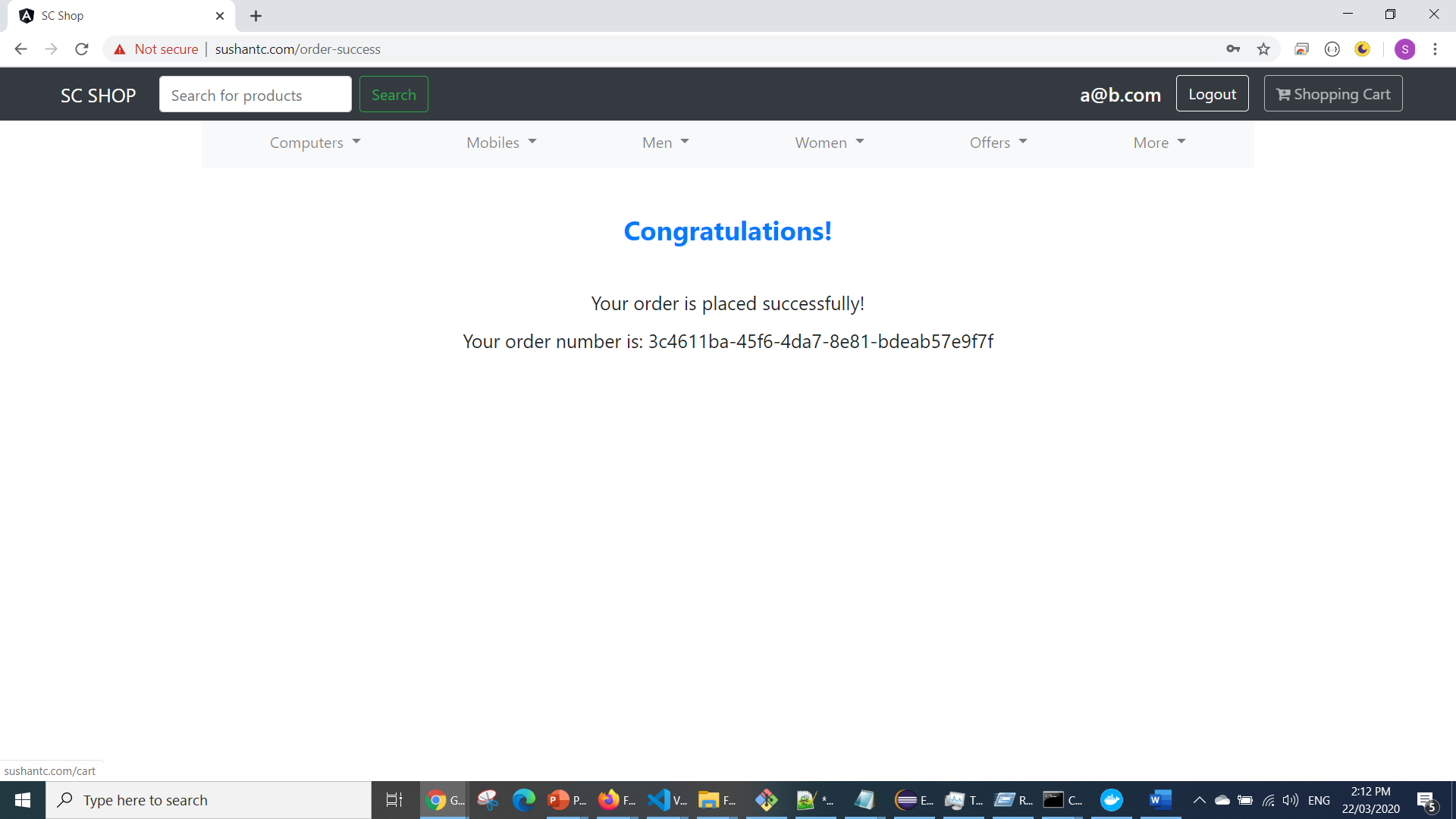
Integrated backend apis – auth, user, product, order

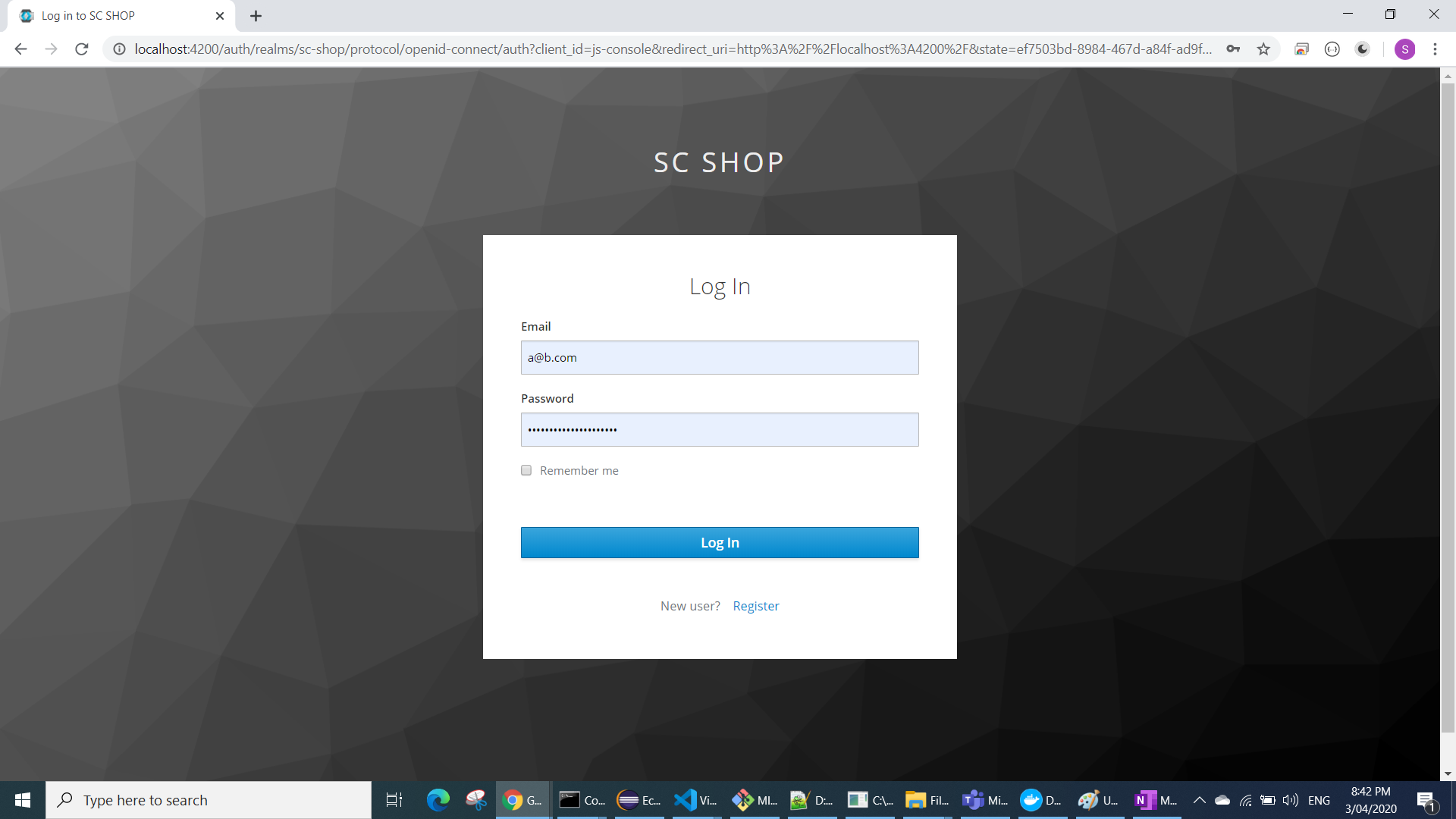
 







Using Keycloak- OAuth2.0 for application security now



Sprint 4 - Add order service and required communication with other services

***In progress...***

Added basic APIs

Added validation before initiating order

Configured Kafka on local windows machine and tried pub-sub with it

Configured Kafka with bitnami docker containers - zookeeper and 1 broker *(2 brokers causing some never-ending error logs -solve later)*

Using Conduktor to administer Kafka

*Kafka event notifications for*

*Order Service: ORDER\_INITITATED -> payment, product inventory, email, shipping*

*Payment Service: ORDER\_PAYMENT\_FAILED -> order, product inventory, email, shipping*

*Product Service: ORDER\_ITEM\_OUTOFSTOCK -> order, payment, email, shipping*

*Order Service: ORDER\_CONFIRMED -> email, shipping*

*TODO – Revisit and fine tune Kafka settings - commit strategy, etc.*

*Set retention.ms = 5 minutes at Topic-level (should be around a day – but should be enough for experimental purpose)*

Sprint 5 - Add application security using **OAuth 2.0 -** **Keycloak** and **Resource Server config for resources** APIsecurity

Added Keycloack docker image

Configured angular client to connect to Keycloak for user registration and authentication

*Added Token Exchange Stateless Authorization security pattern on Spring Cloud Gateway. UI will have a restricted access\_token which is sent to Gateway, Gateway will exchange it with a JWT token(from Keycloak server) with wider scope (scope=‘product’) to be able to access the products. Similar token exchange will happen for other service call e.g. placing order.*

*For inter-service communication, made the requesting service as oauth2-client and used client\_credentials grant type to be able to securely call other internal micro-service apis.*

**Need to add similar interservice security and Token exchange for call from UI for remaining services.**

***In progress...***

Sprint 6 - Add API gateway service – *Spring Cloud Gateway*

Added service with spring cloud gateway and configured routes

*Added Token Exchange Stateless Authorization security pattern on Spring Cloud Gateway*

*(To Do: Need to see how swagger api docs can be exposed through gateway routes)*

Sprint 7 – Enable CI-CD

(\*Configured Jenkins pipeline script for CI-CD for user-service and tried deployment using docker container, docker-hub.

Tried it on an AWS free instance, but free instances are too small(1GB) and hangs frequently)

Configured it on GCP Instance – with 3.75GB RAM 10G HDD for now – will increase capacity if required

*Not keeping Instance always ON – Saving on free credits – will keep it on once UI is ready*

Reserved a Google Cloud VM instance

Installed and configured Jenkins to link with GitHub repos

Created pipeline to build, test and create docker image and then push them to docker-hub

Tried mono-repo as well as repo-per-microservice – will continue with mono-repo

Tried Declarative and Scripted pipeline –

Using Scripted pipeline for now

*Need to refine Jenkins files further – Find services with changes and build-deploy only those – will do it later*

*(faced issues with deployment steps in Declarative pipeline - will try again later to save time now)*

Using docker-compose within Jenkins to start/stop all docker containers for now *(will try orchestration later)*

*Register a domain and expose application with domain name – will do when UI is ready*

Sprint 8 – Add monitoring

Configured Micrometer, Prometheus, Grafana

Some actuator exposed metrics can be monitored with imported JVM dashboard on (Grafana)<http://35.244.121.244:3000/>,

(Prometheus) <http://35.244.121.244:9090/>

Currently only actuator exposed metrics are recorded *(Will add custom metrics later)*

**Sprint 9** – Enable logging and configure ELK - ElasticSearch, LogStash, and Kibana

*Configured using docker containers - elastic stack (TODO: Format logs further, Add Beats-> Filebeat to configuration)*

Sprint 11 – Configure Distributed Request Tracing

Spring 12 – Configure Circuit Breaker for Fault Tolerance

Use Cases –

Sprint 13 – Add and Integrate Relational Database - Postgres on GCP

Sprint 14 – Add Unit and Integration tests for Order, Payment service (remaining later)–

*Should have added them alongside development (TDD) but first concentrating on delivering basic working skeleton asap*

Revisit each use case with tests and improve code alongside

Sprint 15 – Add few performance tests to simulate load

Sprint 16 – Add application caching using Redis

Sprint 17 – Configure Kubernetes and deploy all containers

Sprint 18 – Configure scaling in Kubernetes and test with existing performance tests

Sprint 19 – Improve product search – MongoDB, ElasticSearch

Sprint 20 – Improve it further with WebFlux

**Initial Features Scope:**

User profile (details, addresses) management

Product search, inventory management

Cart management

Order management

Payment management (only shopping credits mode for now)

**GOAL:**

Host it on: ~~AWS~~ or Google Cloud

Architecture: Microservices

Explore: *domain-driven-design, TDD, event storming, event-driven architecture, (event sourcing – Kafka is covering this), CQRS*

Service communication: REST APIs, send events over Kafka ~~(try gRPC LATER)~~

Database: In-memory H2 for now, integrate Postgres & NoSQL(MongoDB/~~DynamoDB~~) later

UI: Angular 8, TypeScript, Bootstrap

Event Streaming: Kafka

Caching: Redis

Also (*try* *later*): Reactive stream - [WebFlux](https://docs.spring.io/spring/docs/current/spring-framework-reference/web-reactive.html)

Microservice tools -

Service Discovery: [**Spring Cloud Netflix Eureka**](https://spring.io/projects/spring-cloud-netflix)

Configuration Server: [**Spring Cloud Config**](https://spring.io/projects/spring-cloud-config)

Security: [**Spring Security**](https://spring.io/projects/spring-security) – **OAuth 2.0**(Explore [Spring Cloud Security](https://spring.io/projects/spring-cloud-security) later)

Monitoring: [**Micrometer**](https://micrometer.io/) **+** [**Prometheus**](https://prometheus.io/) **+** [**Grafana**](https://grafana.com/)

Request [Tracing](https://medium.com/swlh/distributed-tracing-in-micoservices-using-spring-zipkin-sleuth-and-elk-stack-5665c5fbecf): [**Spring Cloud Sleuth**](https://spring.io/projects/spring-cloud-sleuth) **+** [**Zipkin**](https://zipkin.io/)

Fault Tolerance: [**Spring Cloud Circuit Breaker**](https://spring.io/projects/spring-cloud-circuitbreaker) **+** [**Resilience4J**](https://github.com/resilience4j/resilience4j)

Load Balancing: [**Spring Cloud LoadBalancer**](https://spring.io/guides/gs/spring-cloud-loadbalancer/)

For microservices to discover each other with service name, find working endpoint and communicate

API Gateway: [**Spring Cloud Gateway**](https://spring.io/projects/spring-cloud-gateway)

API gateway – single point of entry (port 8000) for all client requests which will then redirect it to respective to services

Other –

JPA, Swagger2 (for API docs), DevTools, Actuator*(TODO: Decide on what to expose)*

Logging –

ElasticStack– Beats, ElasticSearch, LogStash, and Kibana

Testing:

Unit: Junit, Mockito, Hamcrest - @DataJpaTest, @WebMvcTest, @SpringBootTest

Deployment:

CI/CD: Do with Jenkins configured on GCP, ~~Bamboo~~ and then maybe Jenkins X for kubernetes

Path: => Bare OS Spring Boot services deployment

=> Bare OS docker containers deployment – manage using docker-compose

=> Deploy containers on Kubernetes

=> *If possible, try on OpenStack/CloudFoundry*

SCM: GitHub

Other: Integrate Code coverage tools- Cobertura, PMD in Jenkins CI

Other Editor Tools: Git Bash, Eclipse/IntelliJ Idea, Visual Studio Code, Postman, SourceTree, Conduktor for Kafka administration, Connect Prometheus/Grafana to monitor Kafka



*TODO: update further…*

User Service

Order Service

Email Service

Payment Service

Shipping Service

Eureka Service

UI Client

Config Service

Cart Service

Gateway Service

Kafka – ORDER\_TOPIC

Product Inventory Service

Product *Search* Service

Config Repo on GitHub

*Token exchange Stateless Authorization*

Keycloak Authorization Server

*Authentication*

TO DO

**Prometheus**

**Kibana**

**Grafana**

Monitoring Tools

*Secured*