ASCENDION

Capstone Project

Personal BACKGROUND

(Name, Past Experience, Qualification, Career Summary)

Name : Valan Antony A

Experience : Fresher

Qualification : Bachelor of Engineering-CSE

Career Summary :

- Completed BE Computer Science at Engineering at Sri Krishna College of Engineering and Technology
- Completed Schooling at Carmel Garden

ASCENDION

Key Takeaways/Learnings from the Program (HTD)

The key takeaways from the program:

- **OOP using Core Java:** Master OOP principles to build modular Java applications.
- **SQL Fundamentals:** Design and manage databases; write complex SQL queries.
- **Advanced Java:** Handle multithreading, collections, and concurrency.
- **SDLC & Agile:** Understand software development processes; work effectively in Agile teams.
- **JUnit:** Write unit tests; ensure code reliability with TDD.
- **REST API:** Design, develop, and test RESTful services.
- **Postman:** Test and automate API validations.
- **Spring Boot & Microservices:** Build and manage scalable microservices.
- **Generative AI Intro:** Gain a basic understanding of AI applications.
- **AWS Cloud Practitioner:** Understand AWS services and cloud deployment.

ASCENDION

Problem Statement of the Capstone Project

- With the rapid growth of e-commerce, businesses need robust, scalable, and efficient systems to manage online shopping experiences. Traditional monolithic architectures often struggle to meet these demands due to their limitations in scalability, flexibility, and maintainability. This project addresses these challenges by developing an online shopping application using a microservices architecture with Java Spring and API gateways.

- The application is designed to enhance the user experience and operational efficiency by dividing functionalities into distinct microservices: customer management, product management, shopping cart management, and order processing. Each microservice operates independently, ensuring scalability and resilience. The API gateway acts as a single entry point for client requests, simplifying client interactions and improving system performance.

ASCENDION

Project Description:

This capstone project involved developing an online shopping application using Java, Spring Microservices, MySQL, and API gateways.

- **Java:**
    - Used for backend development, implementing the core logic and data models for customer, product, cart, and order microservices.

- **Spring Microservices:**
    - Built each service as a Spring Boot application for independent deployment. Spring Cloud was used for inter-service communication, service discovery, and load balancing.

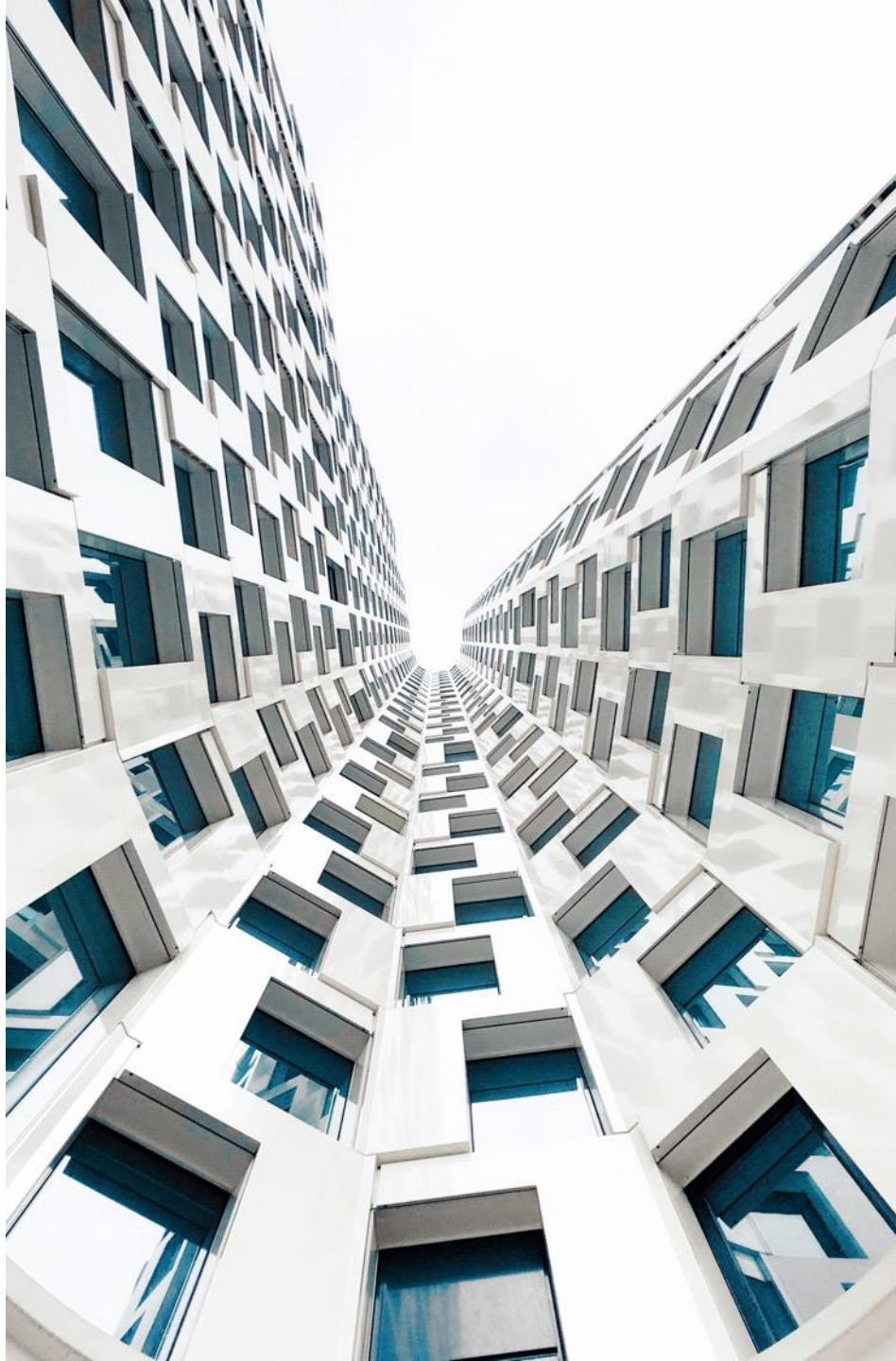
ASCENDION

- **MySQL:**
  - Managed data persistence for each microservice using separate schemas. JPA and Hibernate were used to map Java objects to MySQL tables, facilitating easy data operations.
- **API Gateways:**
  - A Spring Cloud Gateway served as the single entry point for API requests, handling routing, security, and monitoring.
- **API Calls:**
  - RESTful APIs were developed for client-server and inter-service interactions, enabling CRUD operations across the microservices.

ASCENDION

Spring Framework:

Spring is a robust Java framework that streamlines the development of enterprise-level applications. It emphasizes dependency injection to promote loose coupling and modular design. Spring Boot, a key part of the framework, accelerates the development process by providing pre-configured setups and embedded servers, allowing developers to quickly create and deploy standalone, production-ready applications. Additionally, Spring offers a variety of modules like Spring MVC for building web applications, Spring Data for simplified data access, and Spring Security for managing authentication and authorization.



ASCENDION

Microservices:

Microservices is an architectural style that breaks down an application into small, independent services, each responsible for a distinct business capability. These services can be developed, deployed, and scaled independently, which increases flexibility and speeds up development cycles. Each microservice typically manages its own database, enhancing data integrity and isolation. Services communicate via APIs, often using RESTful HTTP, enabling seamless integration. This architecture enhances scalability, resilience, and allows teams to work autonomously on different parts of the application.

ASCENDION

Important areas of the Project with screenshots

Customer Implementation

```java
@NoArgsConstructor
@Getter
@Setter
@ToString
@EqualsAndHashCode
public class Customer {
    @Id
    @GeneratedValue
    private int cusId;
    private String cusName;
    private String cusCity;
    private int cusPinCode;
    private String cusGender;
    private String cusEmail;
    private String cusPhone;
    @OneToOne
    @JoinColumn(name = "cartId")
    @JsonManagedReference
    @JsonIgnore
    private Cart cart;
    @ManyToOne
    @JoinColumn(name = "orId")
//    @JsonManagedReference
    @JsonIgnore
    private Order orderTab;
```

```java
@RestController
@RequestMapping("/api/shop/customers")
public class CustomerController {

    @Autowired
    CustomerService customerService;

    @GetMapping
    public List<Customer> findAll(){
        return customerService.findAll();
    }

    @GetMapping("/cusName/{cusName}")
    public List<Customer> findByCusName(@PathVariable("cusName") String cusName){
        return customerService.findByCusName(cusName);
    }
    @GetMapping("/cusCity{cusCity}")
    public List<Customer> findByCusCity(@PathVariable("cusCity")String cusCity){
        return customerService.findByCusCity(cusCity);
    }
    @GetMapping("/find/{cusId}")
    public Customer findById(@PathVariable("cusId")int cusId) throws CustomerNotFoundException {
        return customerService.findById(cusId);
    }

    @DeleteMapping("/delete/{cusId}")
    public void deleteById(@PathVariable("cusId")int cusId){
        customerService.deleteById(cusId);
    }

    @PutMapping("/update/{cusId}")
    public Customer updateCustomer(@PathVariable int cusId,@RequestBody Customer customer){
        return customerService.updateCustomer(cusId,customer);
    }
```

```java
public class CustomerServiceImpl implements CustomerService{

    @Autowired
    CustomerRepo customerRepo;
    @Override 1 usage
    public Customer addCustomer(Customer customer) {
        return customerRepo.save(customer);

    }

    @Override 1 usage
    public Customer updateCustomer(int cusId, Customer customer) {
        if(customerRepo.existsById(cusId)){
            customer.setCusId(cusId);
            return customerRepo.save(customer);
        }
        return null;
    }

    @Override 1 usage
    public void deleteById(int cusId) {
        customerRepo.deleteById(cusId);
    }

    @Override 1 usage
    public Customer findById(int cusId) throws CustomerNotFoundException {
        Optional<Customer> customer = customerRepo.findById(cusId);
        if(customer.isEmpty()){
            throw new CustomerNotFoundException(cusId);
        }
        return customer.get();
    }
```

ASCENDION

Important areas of the Project with screenshots

Cart Implementation

```java
public class CartDaoImpl implements CartDao {  no usages


    @PersistenceContext  16 usages
    private EntityManager em;

    @Override  1 usage
    @Transactional
    public void addToCart(int cusId, int prodId) {

        Customer customer = em.find(Customer.class, cusId);
        Cart cart = customer.getCart();
        if (cart != null) {
            Product p1 = em.find(Product.class, prodId);
            List<CartLine> cp = cart.getCartLineProducts();
            boolean exist = false;
            int cpi = 0;
            for (CartLine ce : cp) {
                if (ce.getProduct() == p1) {
                    cpi = ce.getCartLineId();
                    exist = true;
                    break;
                }
            }
            if (exist) {
                for (CartLine cd : cp) {
                    if (cd.getCartLineId() == cpi) {
                        cd.setTotalPrice(cd.getTotalPrice() + cd.getProdPrice());
                        cd.setProdQuantity(cd.getProdQuantity() + 1);
                    }
                }
            } else {
```

```java
public class CartServiceImpl implements CartService{

    @Autowired
    CartRepo cartRepo;
    @Override  1 usage
    public void addCart(int cusId,int prodId) {
         cartRepo.addToCart(cusId,prodId);
    }



    @Override  no usages
    public void removeFromCart(int cartId) { cartRepo.deleteById(cartId); }

    @Override  no usages
    public List<Cart> removeCart(Cart cart) {
        return null;
    }


    @Override  1 usage
    public List<Cart> getAllCarts() {

        return cartRepo.findAll();
    }

    @Override  1 usage
    public String updateCart(int cusId, int prodId) {
        return cartRepo.updateCart(cusId,prodId);
    }

    @Override  1 usage
    public Cart getCartById(int cusId) {
        return cartRepo.getCartById(cusId);
```

```java
@RestController
@RequestMapping("/api/shop/cart")
public class CartController {
    @Autowired
    CartService cartService;

    @PostMapping("/add")
    public void addToCart(@RequestParam("prodId")int prodId,@RequestParam("cusId")int cusId){
        cartService.addCart(cusId,prodId);
    }

    @PostMapping("/update")
    public String updateCart(@RequestParam("prodId")int prodId,@RequestParam("cusId")int cusId)
        return cartService.updateCart(cusId,prodId);
    }

    @GetMapping("/getCartById")
    public Cart getCartById(@RequestParam("cusId")int cusId){
        return cartService.getCartById(cusId);
    }
    @GetMapping("/getAllCarts")
    public List<Cart> getAllCarts(){
        return cartService.getAllCarts();
    }

    @DeleteMapping("/removeAllCart")
    public String removeAllCart(@RequestParam("cusId")int cusId){
        return cartService.removeAllCart(cusId);

    }
```

ASCENDION

Important areas of the Project with screenshots

Order Implementation

```java
@Repository
public class OrderDaoImpl implements OrderDao{

    @PersistenceContext  13 usages
    private EntityManager em;
    @Override  1 usage
    @Transactional
    public void buyNow(int cusId, int prodId, int prodQuantity) {
        Customer cust = em.find(Customer.class,cusId);
        Product prod = em.find(Product.class,prodId);
        Order order = new Order();
        order.setCustomer(cust);
        List<OrderLine> orderLineList = new ArrayList<>();
        OrderLine orderLine = new OrderLine();
        orderLine.setProduct(prod);
        orderLine.setOrLinePrice(prod.getProdPrice());
        orderLine.setOrLineProdQuantity(prodQuantity);
        orderLine.setOrLineStatus("Ordered");
        orderLine.setOrLineDate(LocalDate.now());
        orderLine.setOrLineTotPrice(prod.getProdPrice()*prodQuantity);
        orderLineList.add(orderLine);
        em.persist(orderLine);
        order.setOrLineList(orderLineList);
        order.setOrTotPrice(orderLine.getOrLineTotPrice());
        em.persist(order);
        int prodQuan = prod.getProdQuantity()-prodQuantity;
        prod.setProdQuantity(prodQuan);
        em.persist(prod);
        cust.setOrderTab(order);
        em.persist(cust);
    }

    @Override  1 usage
    @Transactional
```

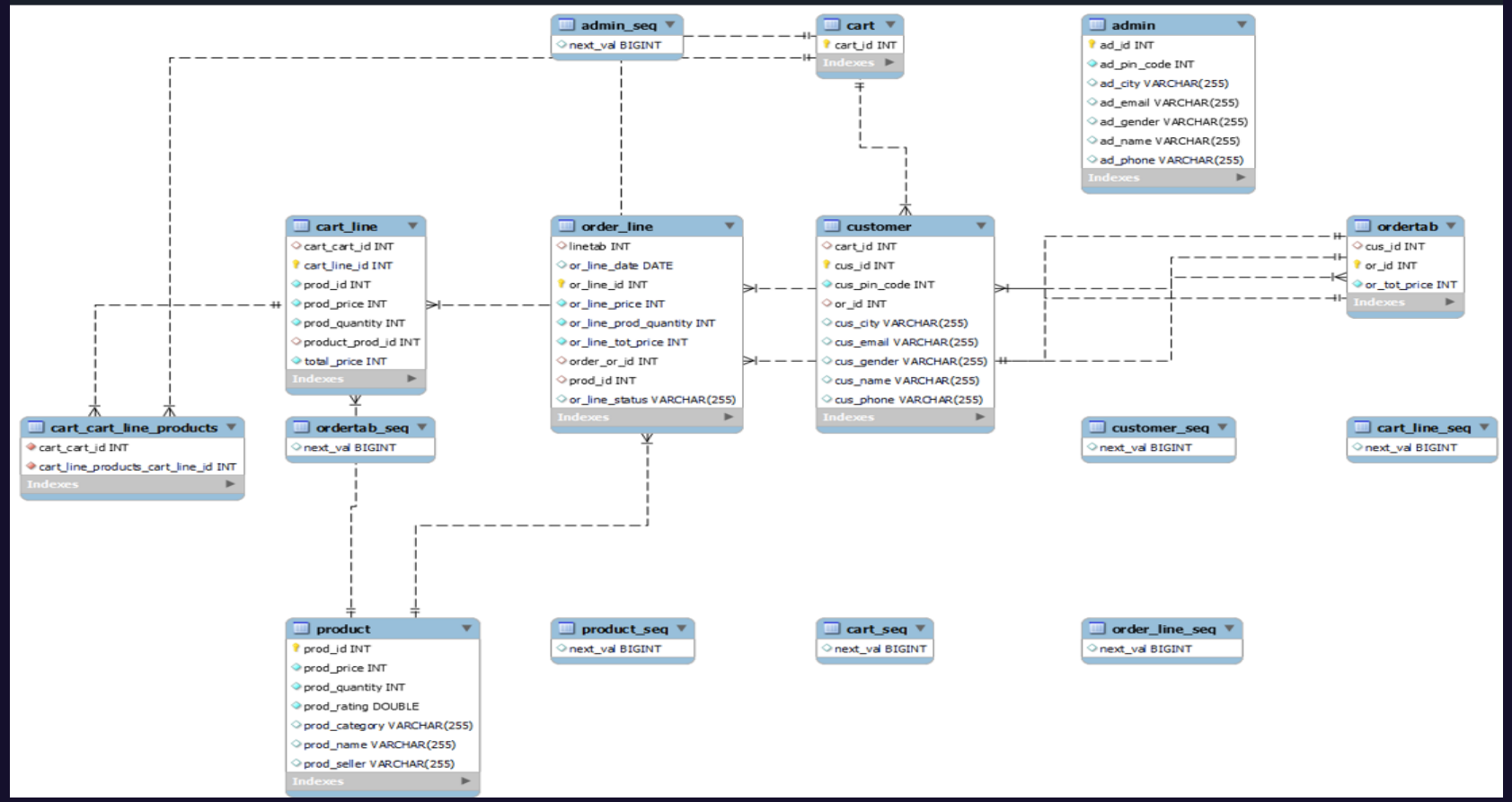
```java
@RestController
@RequestMapping("/api/shop/orders")
public class OrderController {

    @Autowired
    OrderService orderService;
    @PostMapping("/buyNow")
    public void buyNow(@RequestParam("cusId")int cusId,@RequestPara
        orderService.buyNow(cusId,prodId,prodQuantity);
    }
    @PostMapping("orderFromCart")
    public void orderFromCart(@RequestParam("cusId")int cusId){
        orderService.orderFromCart(cusId);
    }
    @DeleteMapping("/cancel")
    public void cancelOrder(@RequestParam("cusId")int cusId){
        orderService.cancelOrder(cusId);
    }
    @GetMapping("/displayOrder")
    public List<Order> findAll(){
        return orderService.findAll();
    }
    @GetMapping("/findOrderById")
    public Order findOrderById(@RequestParam("cusId")int cusId){
        return orderService.getOrderById(cusId);
    }

}
```

```java
@Service
public class OrderServiceImpl implements OrderService{

    @Autowired
    private EntityManager em;
    @Autowired
    private OrderRepo orderRepo;

    @Override  1 usage
    public void buyNow(int cusId, int prodId, int prodQuantity) {
        orderRepo.buyNow(cusId, prodId, prodQuantity);
    }

    @Override  1 usage
    public void orderFromCart(int cusId) {
        orderRepo.orderFromCart(cusId);
    }

    @Override  1 usage
    public void cancelOrder(int cusId) {
        orderRepo.cancelOrder(cusId);
    }

    @Override
    public List<Order> findAll() {
        return orderRepo.findAll();
    }

    @Override  1 usage
    public Order getOrderById(int cusId) {
        return orderRepo.getOrderById(cusId);
    }
}
```

ASCENDION

Important areas of the Project with screenshots

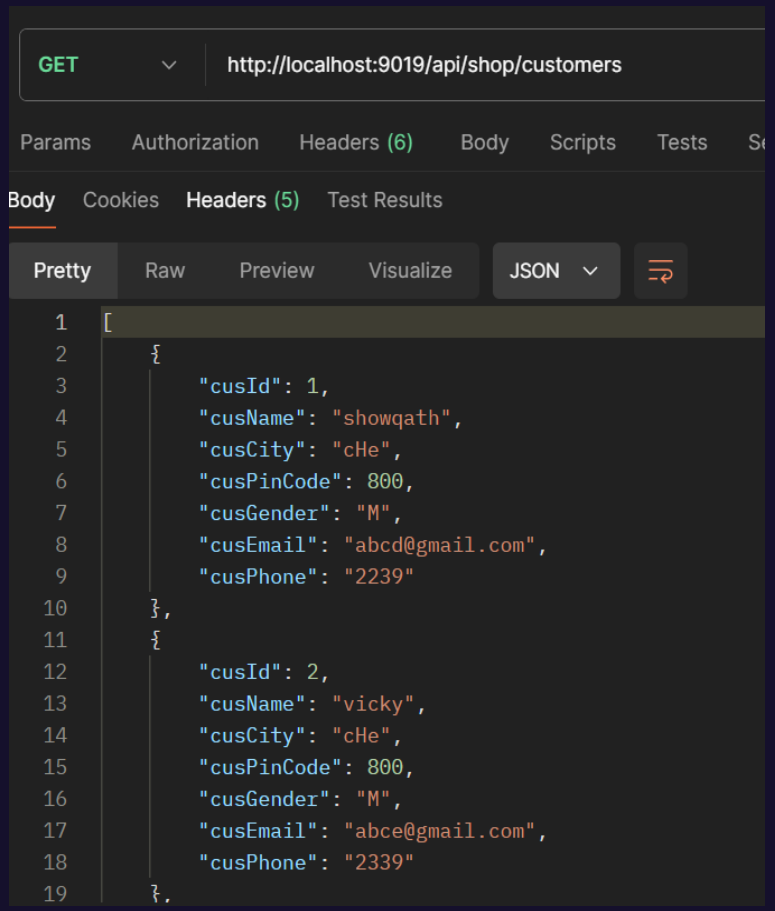
Entity Diagram

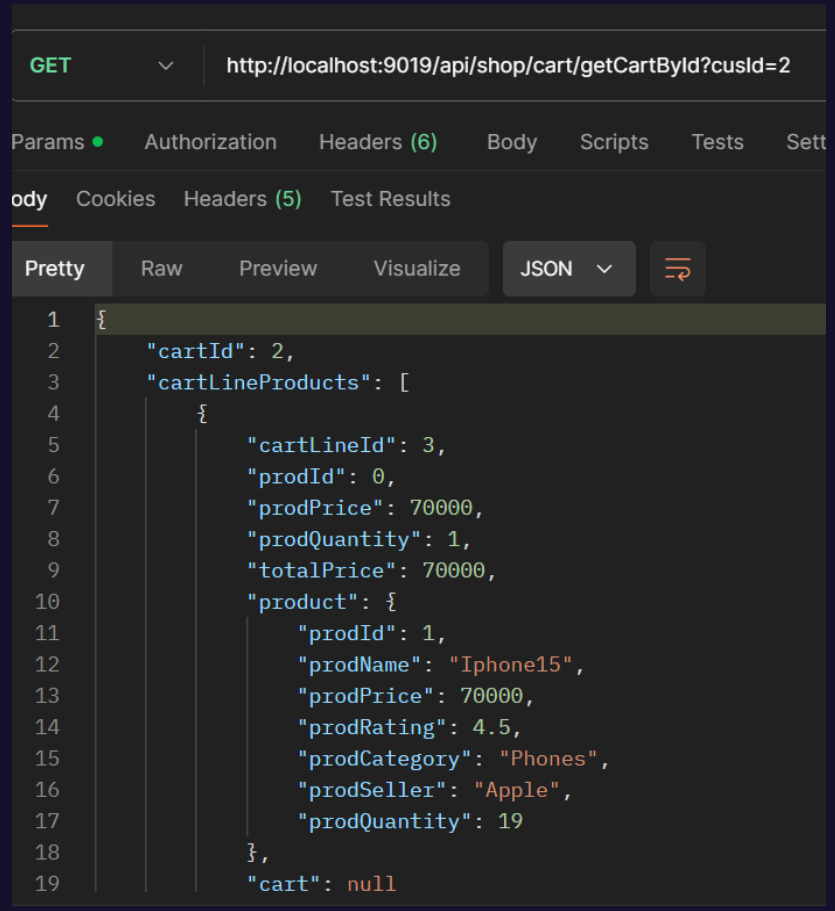


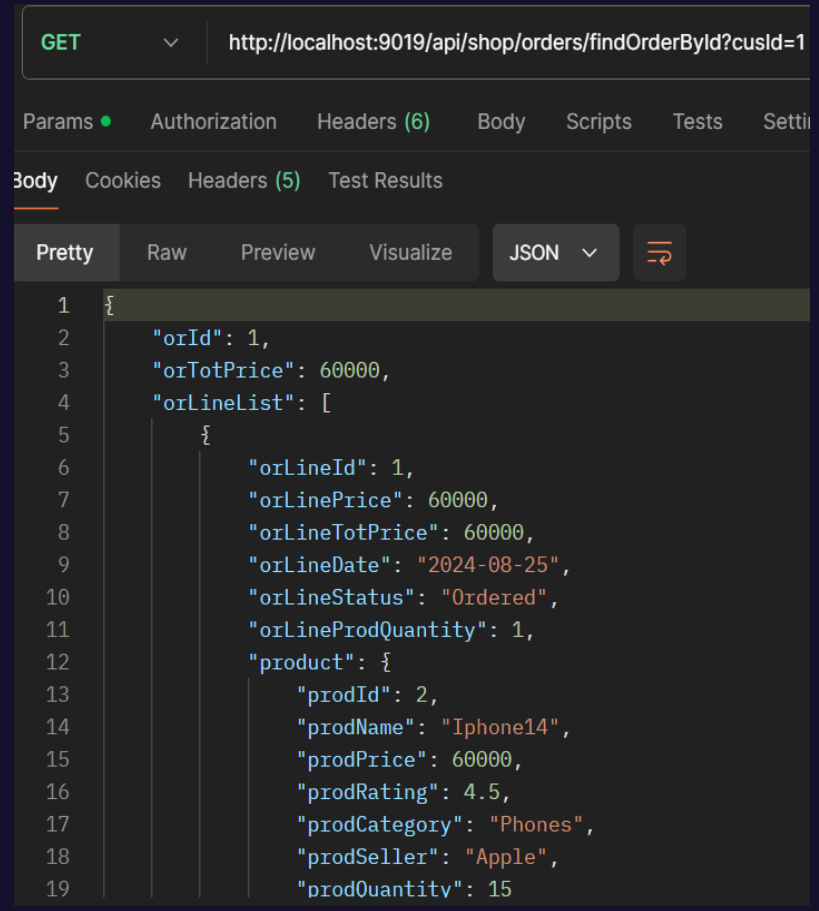
ASCENDION

Important areas of the Project with screenshots

Postman outcomes







ASCENDION

Conclusion

In conclusion, this capstone project successfully demonstrated the implementation of a scalable and maintainable online shopping application using Java Spring and microservices architecture. By modularizing the application into distinct services for customer, products, cart, and order management, and integrating them through an API gateway, the project showcases the benefits of flexibility, independent scaling, and enhanced resilience in modern application development.


ASCENDION
Thank You