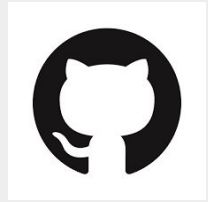
DTAP

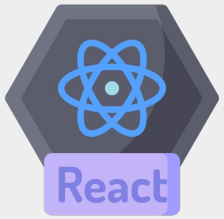
Java Bootcamp

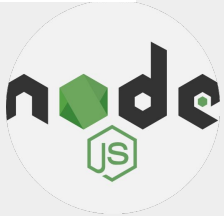
Spring 2023 Cohort

Shirley, Joshua, Jay,
Moise & Ahmed



Java Bootcamp





Capstone Project:

Food Delivery App

Spring 2023 Java Cohort

Breakdown of Tasks

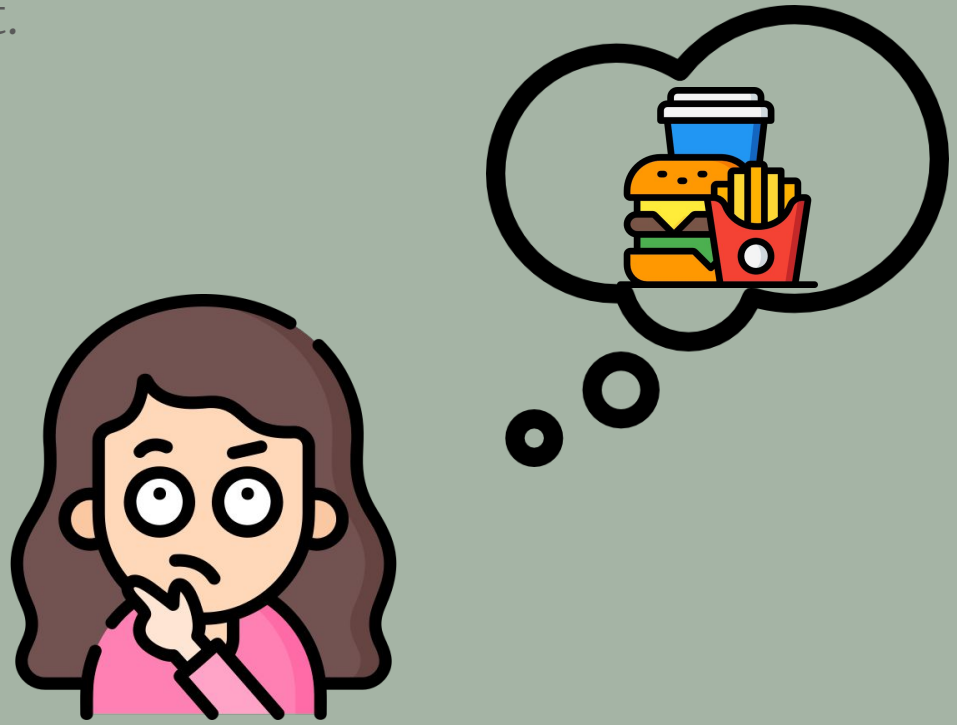
Project Phases				
Team Member	Phase I: Planning	Phase II: Implementation	Phase III: Testing	Presentation Prep
Shirley (Project Lead)	Lead Flow Diagrams Creation - Backend Diagrams - Frontend Diagrams - UI Design Design User Related Pages	Backend: Writing Feedback: - Business Logic - Exceptions - Advice Frontend: -Manage Github Repository - Set up Angular Project - Work on page routing - Writing Cart Services & component		Lead creation/revision of slides Set up Presentation and Slides Flow Work on Backend Planning Slides
Jay	Designing Admin Dashboard page	Backend: Setting Up Microservices Writing AppDelegate.java Writing Offer: - Business Logic - Exceptions - Advice Frontend: Admin, Feedback, & Offers: - pages - components -services		Work on Microservices slides
Joshua		Backend: Writing AppController.java Writing Vendor: - Business Logic - Exceptions - Advice Frontend: Login, Register, Authentication, & Cart: - pages - components -services	Junit Testing Sonar Code Quality	Work on testing slides
Moise	Designing Vendor Pages	Backend: Writing FoodItem: - Business Logic - Exceptions - Advice Frontend: Vendor: - pages - components -services	Postman Testing	Work on frontend development slides
Ahmed	Revising Flow Diagrams Designing Login/Register Pages	Backend: Writing User: - Business Logic - Exceptions - Advice Frontend: FoodItem: - components -services		Prepare for Demo of web application

Project Overview

I. Problem Statement
II. Objectives
III. Phase 1: Planning
IV. Phase 2: Implementation
V. Phase 3: Testing
VI. Technologies Used
VII. Future Improvements

I. Problem Statement

- People want to place food orders at their favorite restaurants without going out.



II. Objectives

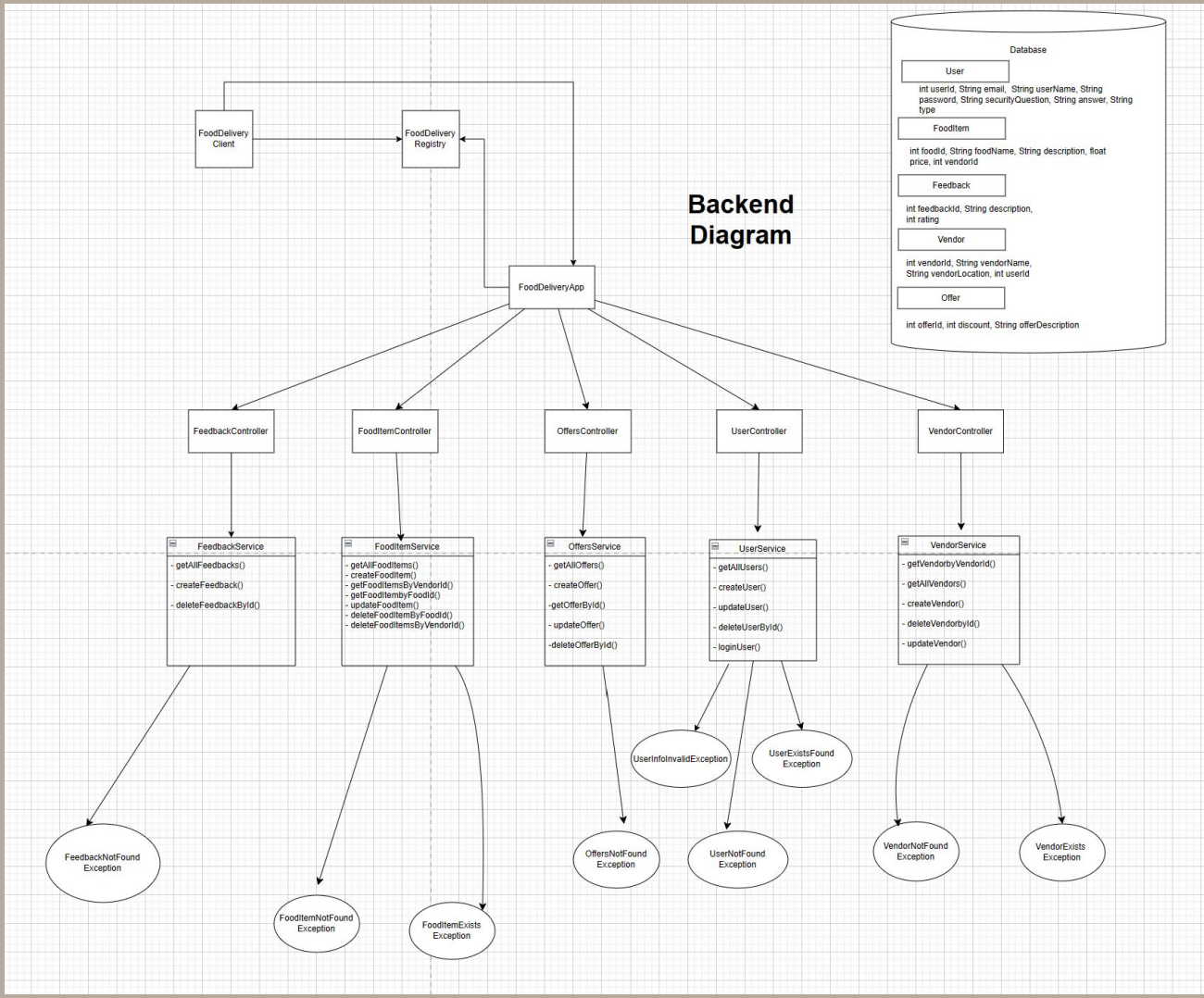
User Stories:

- As a user, I want to order food online, so I don’t have to leave my home.
- As a vendor, I want to market my restaurant online, so I can increase my business’ visibility.

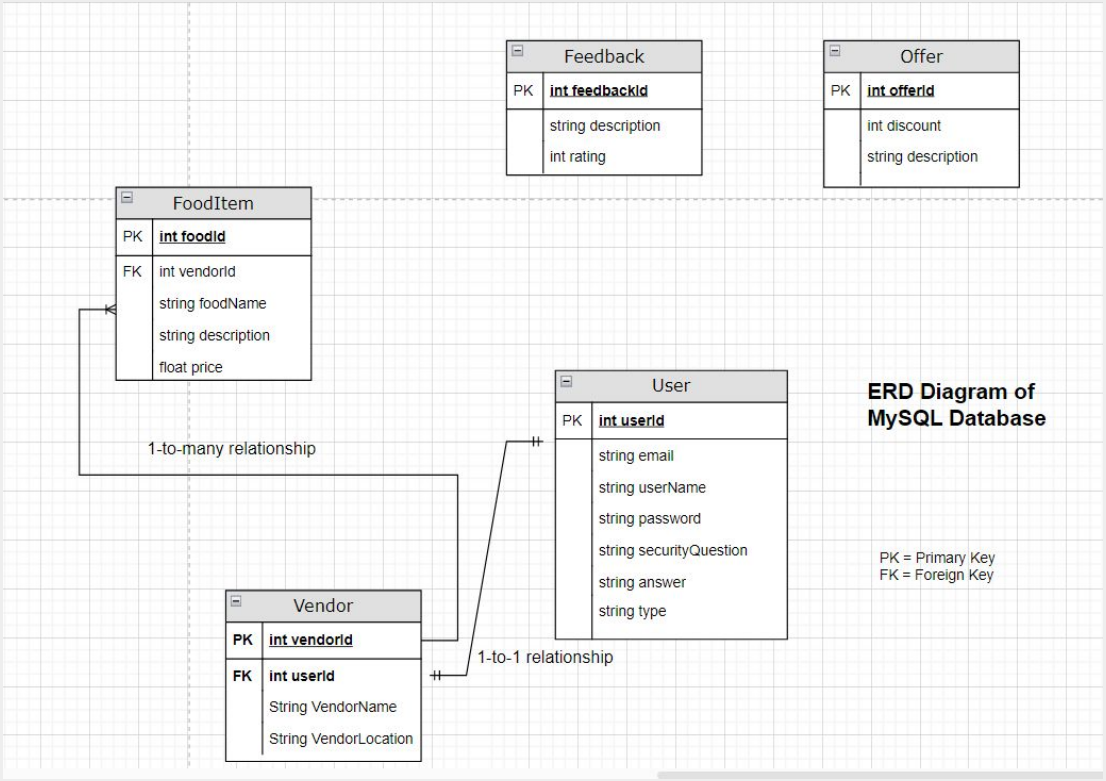
Phase I: Planning

Phase I: Backend Planning

- Written in Java



Entity Relationship Diagram (ERD)



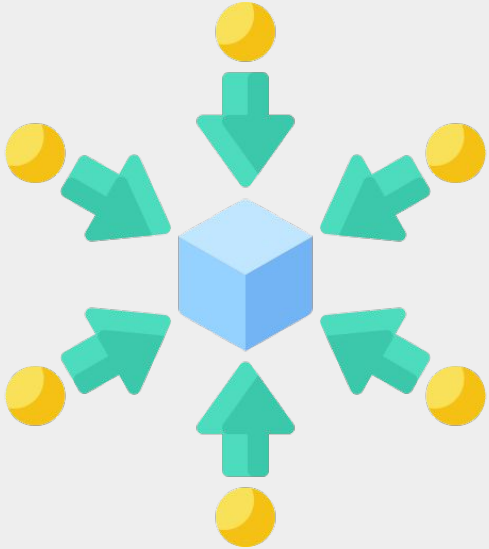
Why MySQL?:

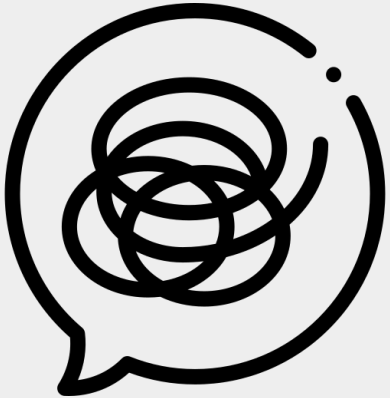
- Supports foreign keys
- Optimized for high performance
- Vertical scalability

Phase II: Implementation

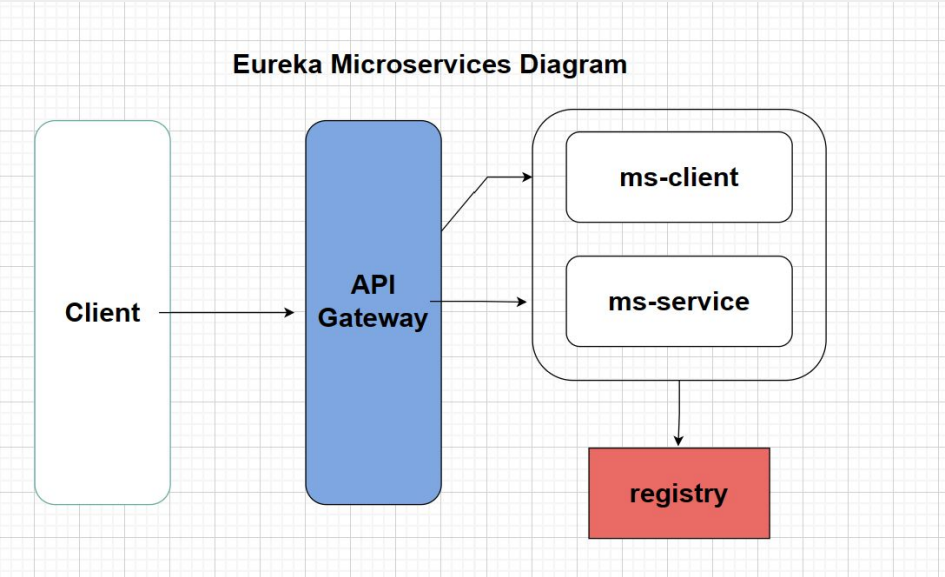
Traditional Approach to Development

- All code is centralized at 1 point
- Disadvantages:
  - Low scalability
  - Low flexibility to changing technologies
  - Harder to read code





Solution: Eureka Microservices



Advantages:

- Better scalability
- Load balancing

Eureka Server Registry with Spring configuration

```java
@SpringBootApplication
@EnableEurekaServer
public class OnlineFoodDeliveryMsRegistryApplication {

    public static void main(String[] args) {
        SpringApplication.run(OnlineFoodDeliveryMsRegistryApplication.class, args);
    }

}
```

```
server.port=7778
eureka.instance.hostname=localhost
eureka.client.registerWithEureka=false
eureka.client.fetchRegistry=false
```

Eureka Service configuration

```java
@SpringBootApplication
@EnableJpaRepositories("com")
@ComponentScan("com")
@EntityScan("com")
@EnableEurekaClient
public class CapstoneProjectApplication {

    public static void main(String[] args) {
        SpringApplication.run(CapstoneProjectApplication.class, args);
    }
```

```
spring.datasource.url=jdbc:mysql://localhost:3306/FoodDeliveryDB?serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root123
spring.datasource.platform=mysql
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.application.name=onlinefooddelivery-ms-service
eureka.client.serviceUrl.defaultZone=http://localhost:7778/eureka
server.port=${PORT:0}
eureka.instance.instance-id=${spring.application.name}:{spring.application.instance_id: ${random.value}}
```

Eureka Client configuration

```java
@SpringBootApplication
@ComponentScan("com")
@EntityScan("com")
@EnableHystrix
@EnableEurekaClient
public class OnlineFoodDeliveryMsClientApplication {

    public static void main(String[] args) {
        SpringApplication.run(OnlineFoodDeliveryMsClientApplication.class, args);
    }
}
```

```
spring.application.name=onlinefooddelivery-ms-client
eureka.client.serviceUrl.defaultZone=http://localhost:7778/eureka
server.port=8055
```

Client consuming a Service registered on Eureka

```java
@Service
public class AppDelegate {
    @LoadBalanced
    @Bean
    public RestTemplate template() {
        return new RestTemplate();
    };

    @Autowired
    public RestTemplate template;

    // USER

    public User createUser(User user) {
        User data = template.exchange("http://onlinefooddelivery-ms-service/foodDeliveryApp/createUser",
                HttpMethod.POST, new HttpEntity<User>(user), User.class).getBody();

        return data;
    }
```

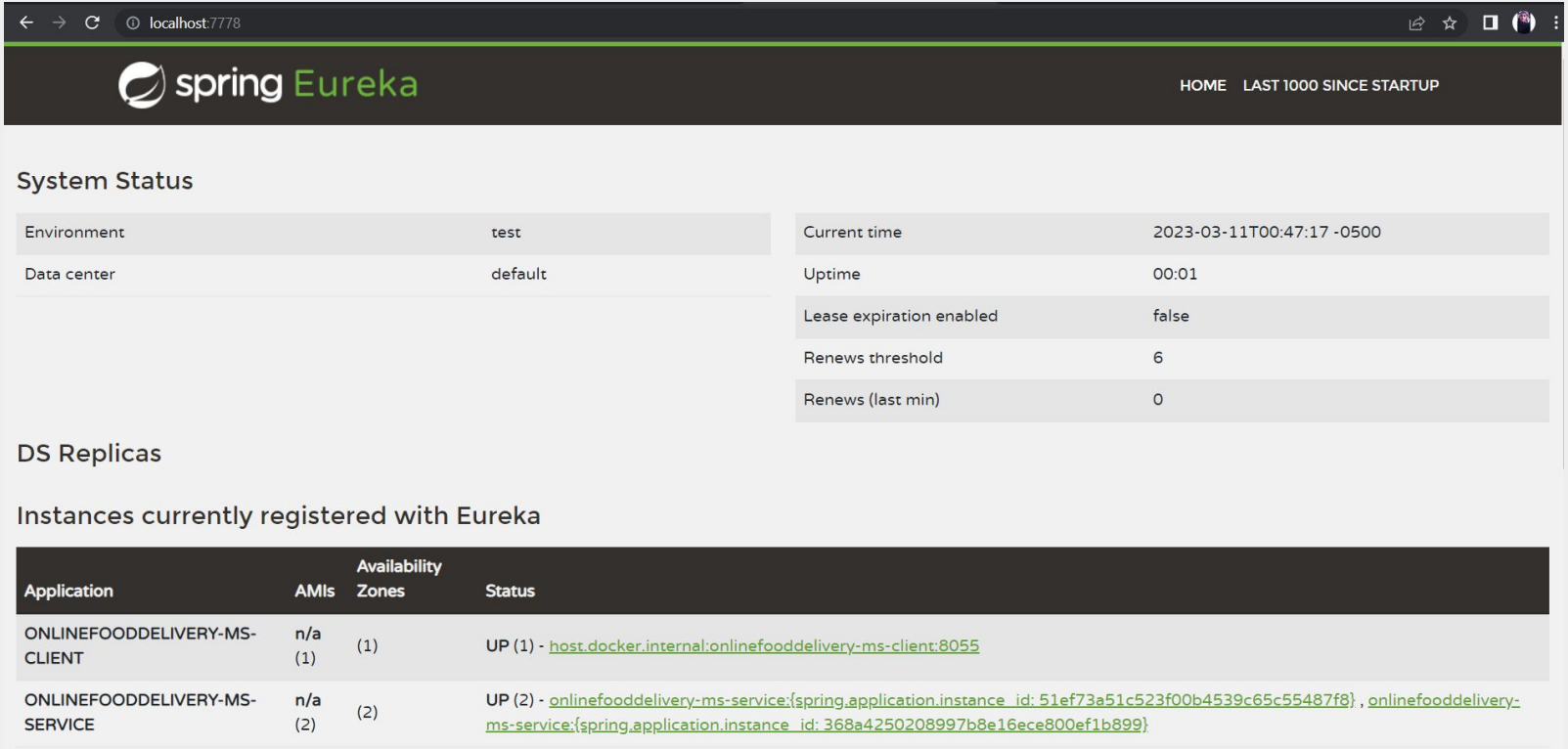
```java
@RestController
@RequestMapping("/foodDeliveryAppClient")
@CrossOrigin("*")
public class AppController {

    @Autowired
    private AppDelegate dao;

    // User

    @PostMapping("/createUser")
    public User createUser(@RequestBody User user) {
        return dao.createUser(user);
    }
```

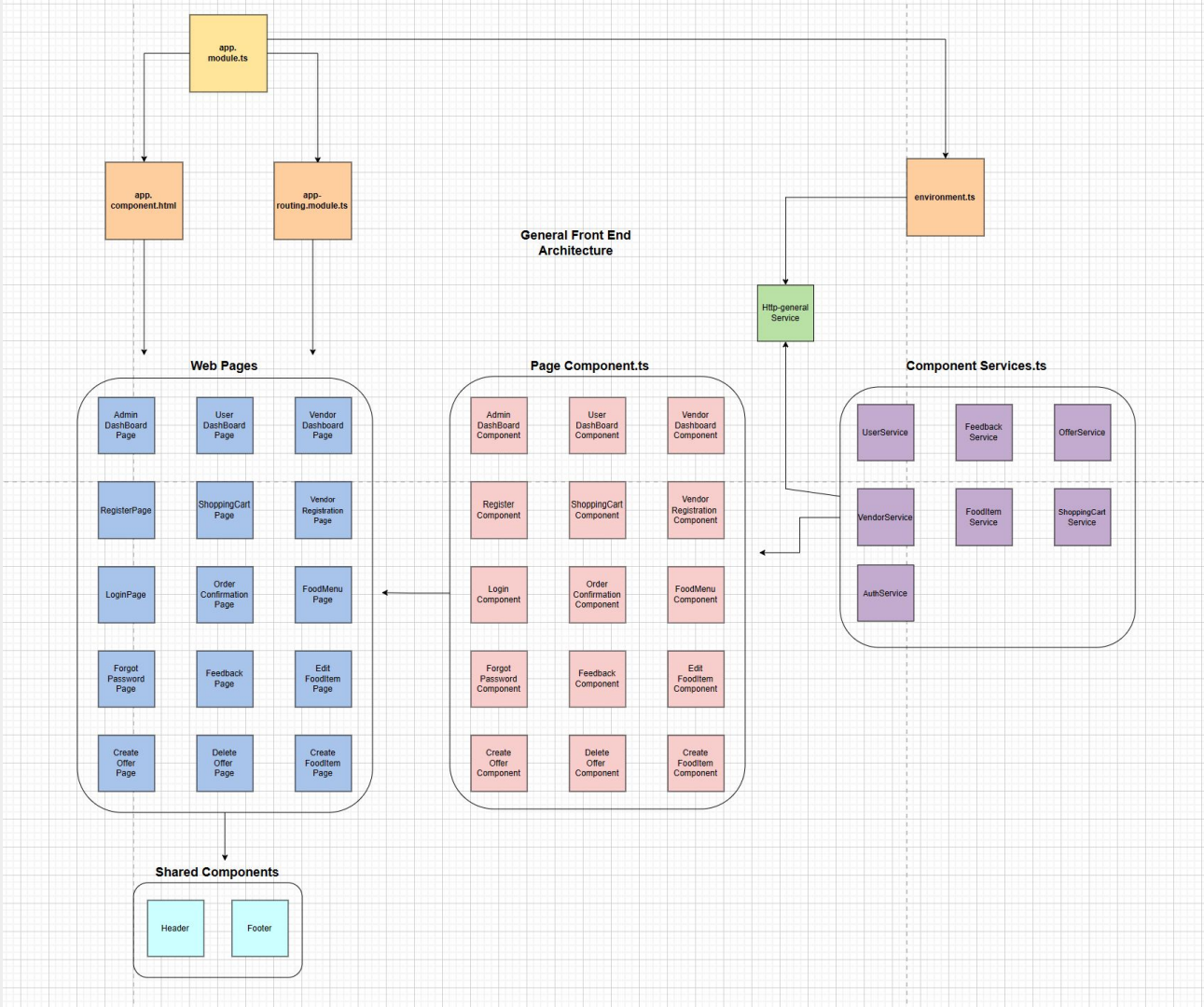
Eureka Server Dashboard



All instances registered on localhost

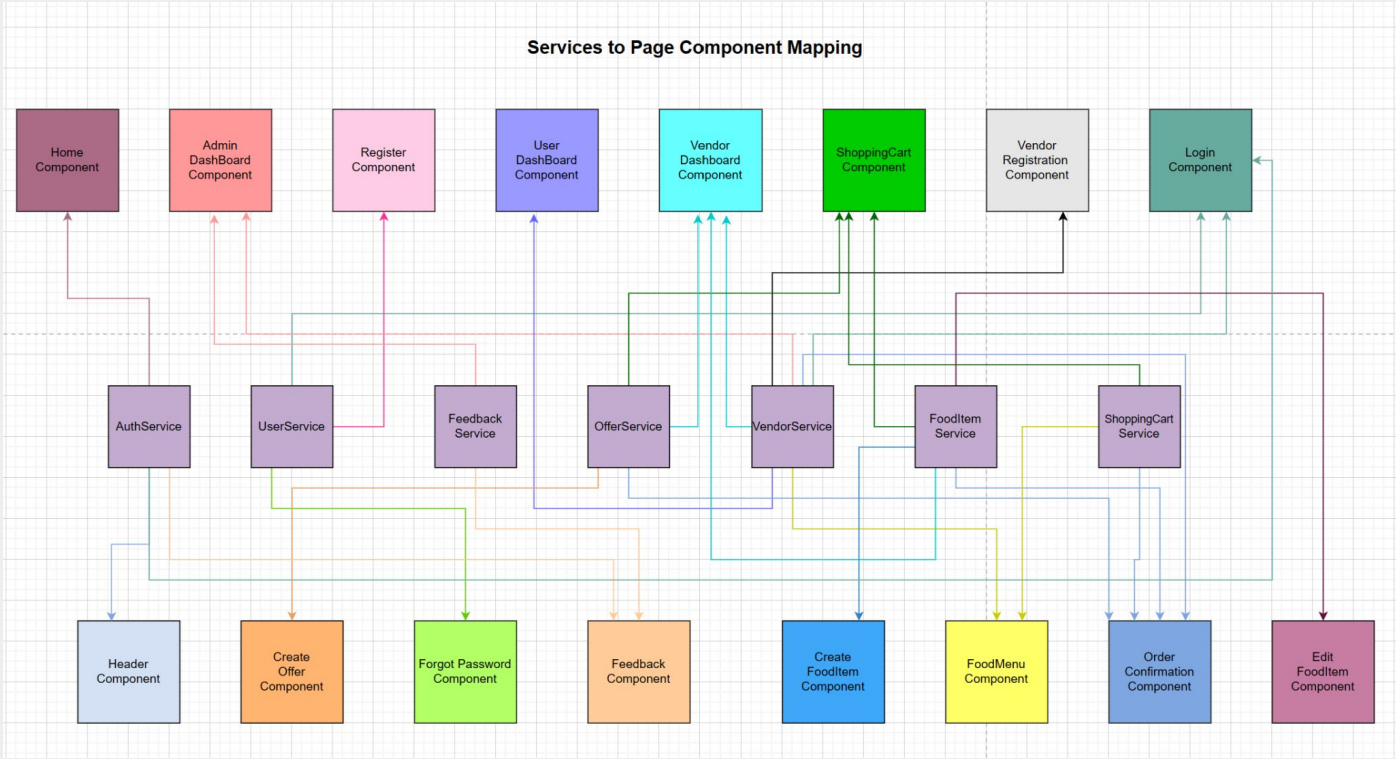
Front End Diagrams

- Angular
- Visual aid for code flow



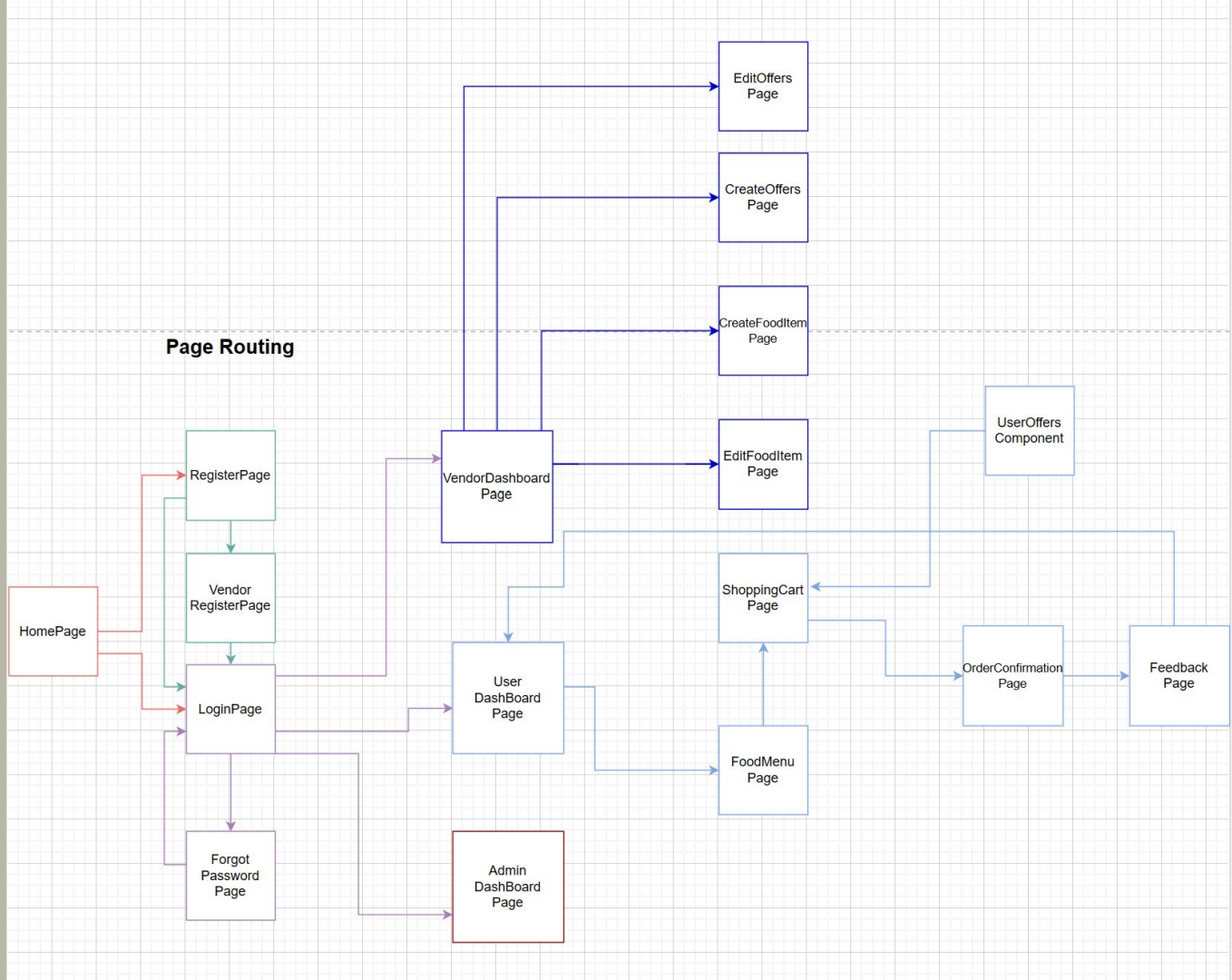
Front End Diagrams

- Shows services utilized by each page component

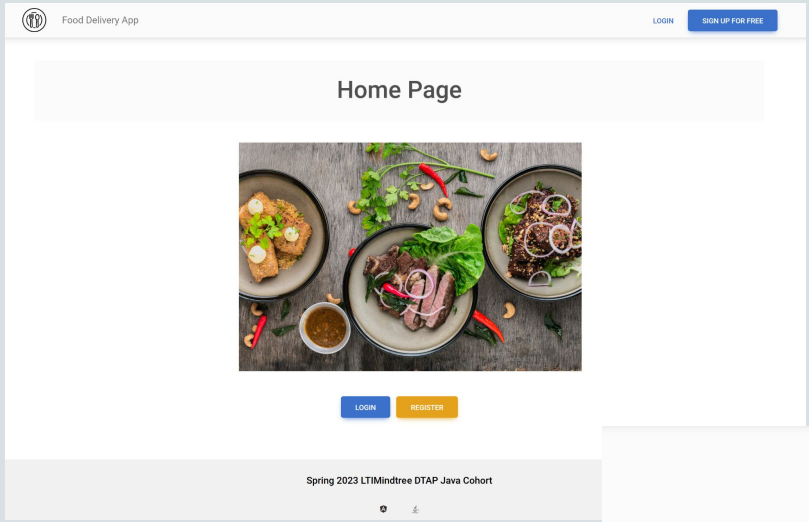


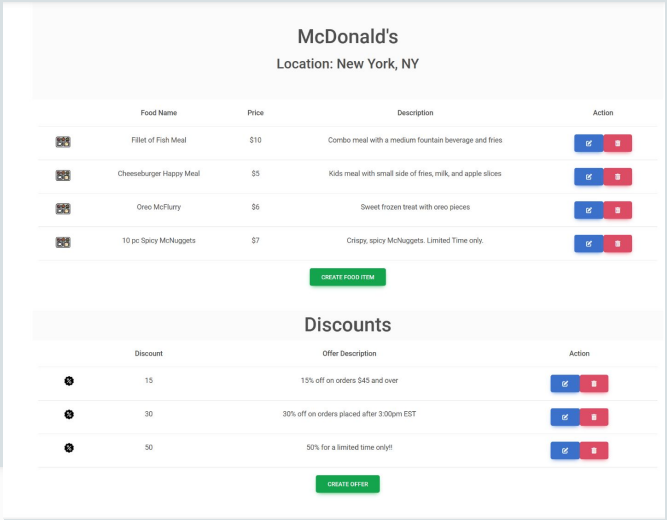
Front End Diagrams

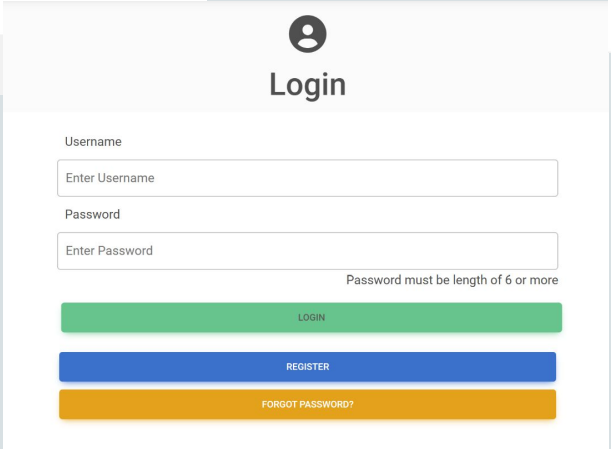
- Provides clarity of website navigation



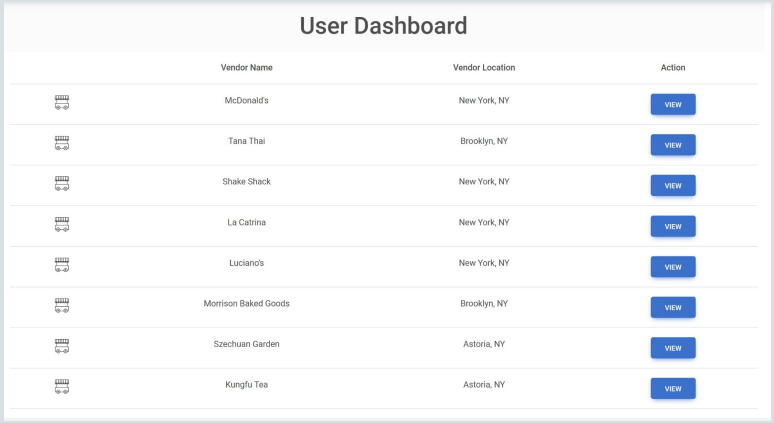
Web Application: User Authentication

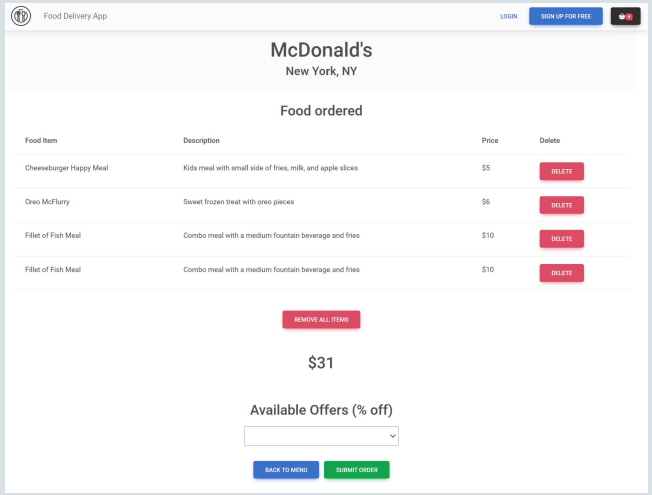


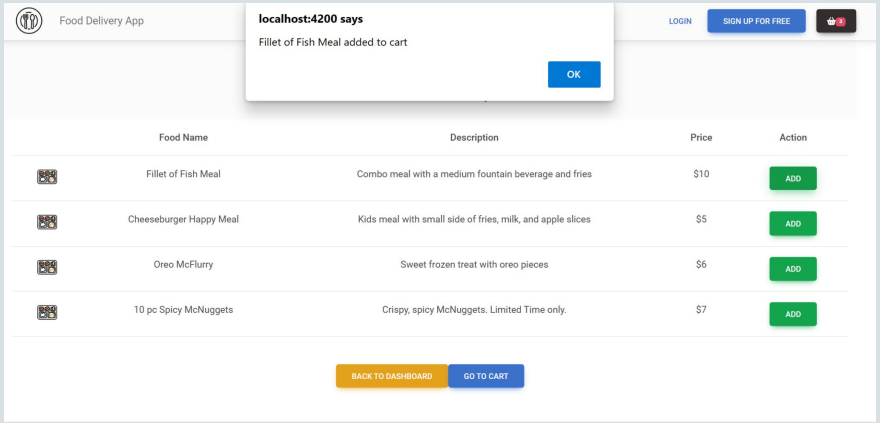




Web Application: Ordering Food



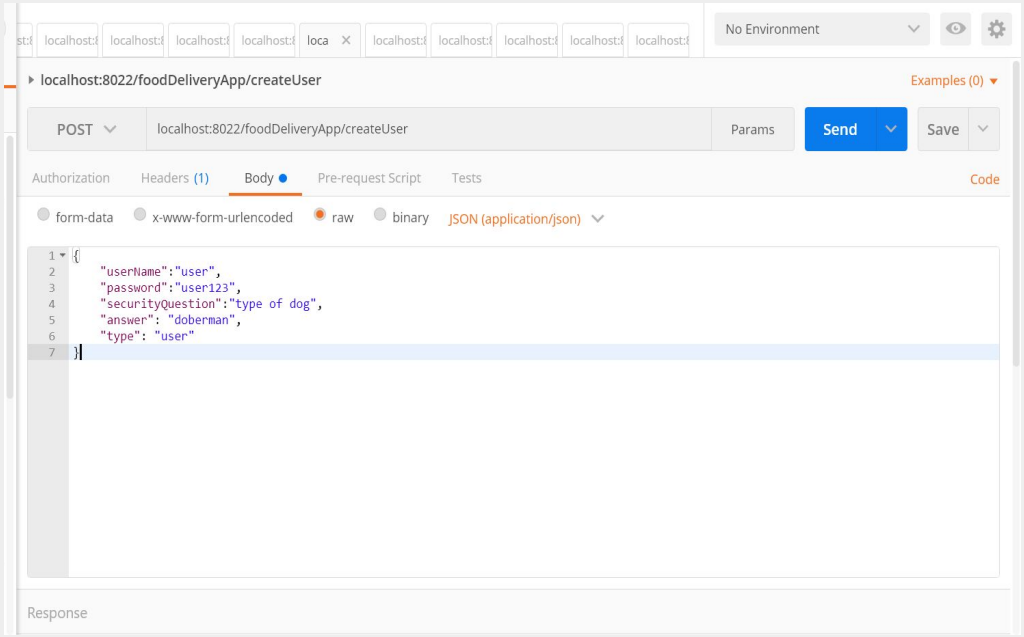




Phase III: Testing

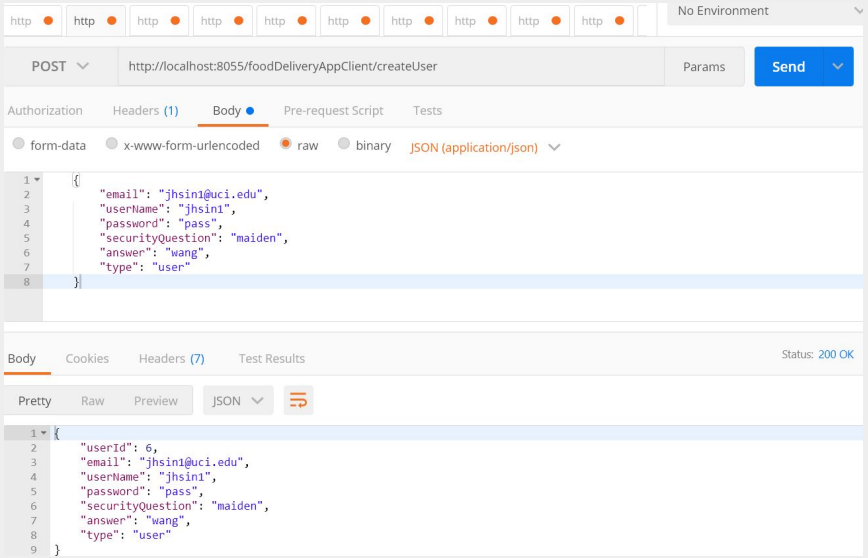
Postman

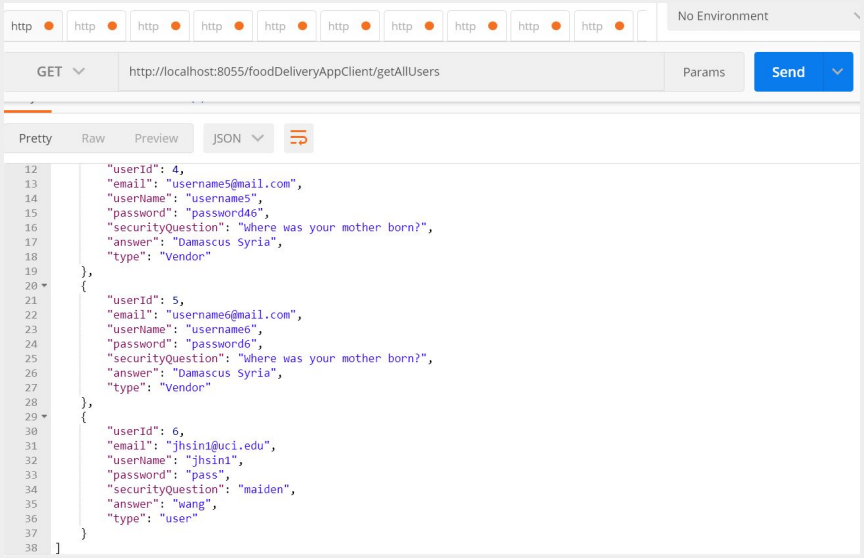
- Postman was used for API and functionality testing.



Postman

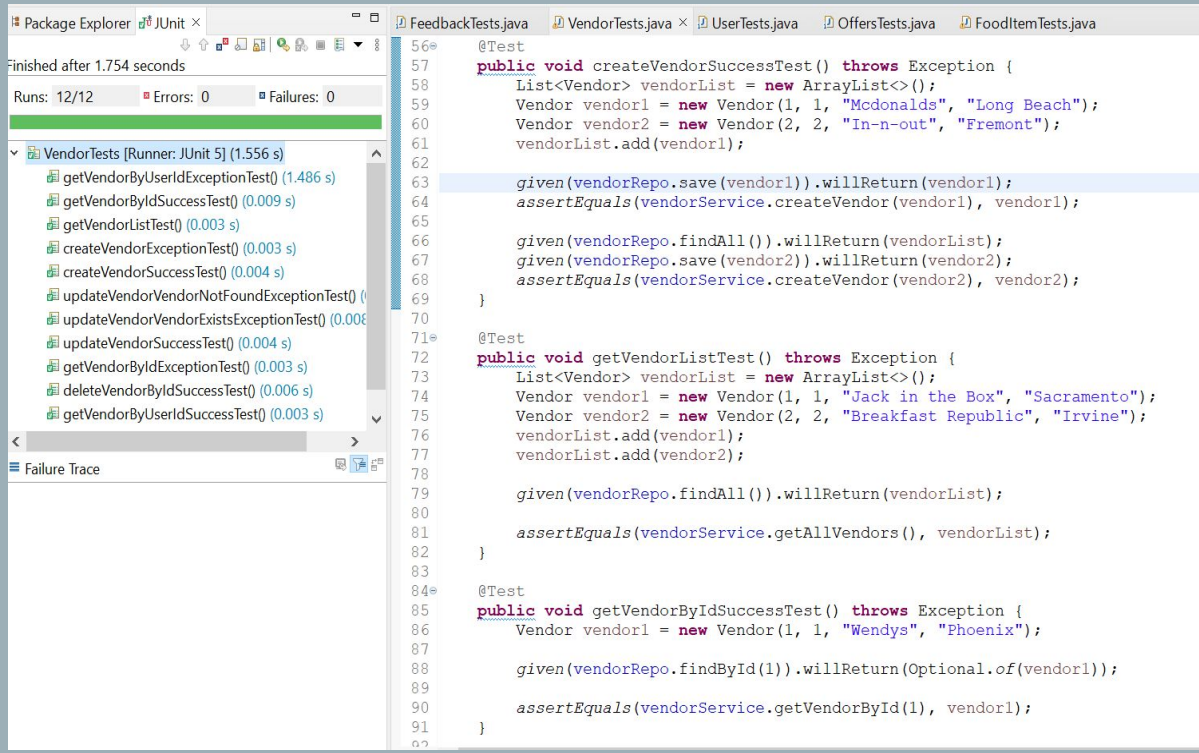
- Tested CRUD methods independently and in conjunction, in our service and client





Junito & Mockito

- Additional tests were made with JUnit testing using Mockito.
- 44 tests in total were made to test all functionality and exceptions.



Junito & Mockito

```java
@Override
public Vendor createVendor(Vendor vendor) {
    List<Vendor> vendorList = vendorRepo.findAll();
    for(Vendor v : vendorList) {
        if(v.getVendorName().equals(vendor.getVendorName()) && v.getVendorLocation().equals(vendor.getVendorLocation())) {
            throw new VendorExistsException(vendor.getVendorName(), vendor.getVendorLocation());
        }
    }
    return vendorRepo.save(vendor);
}
```

```java
@Test
public void createVendorExceptionTest() throws Exception {
    List<Vendor> vendorList = new ArrayList<>();
    Vendor vendor1 = new Vendor(1, 1, "Mcdonalds", "Long Beach");
    Vendor vendor2 = new Vendor(2, 2, "Mcdonalds", "Long Beach");
    vendorList.add(vendor1);

    given(vendorRepo.findAll()).willReturn(vendorList);

    VendorExistsException except = assertThrows(VendorExistsException.class, () -> vendorService.createVendor(vendor2));
    assertTrue(except.getMessage().contentEquals("Vendor with name Mcdonalds and location Long Beach already exists"));
}
```

```java
    }
```

Junito & Mockito

```java
@Override
@Transactional
public boolean deleteVendorById(int vendorId) {
    Optional<Vendor> vendor = vendorRepo.findById(vendorId);
    if(vendor.isPresent()) {
        vendorRepo.deleteById(vendorId);
        userRepo.deleteById(vendor.get().getUserId());
        foodItemRepo.deleteByVendorId(vendorId);
        return true;
    }
    throw new VendorNotFoundException(vendorId);
}
```

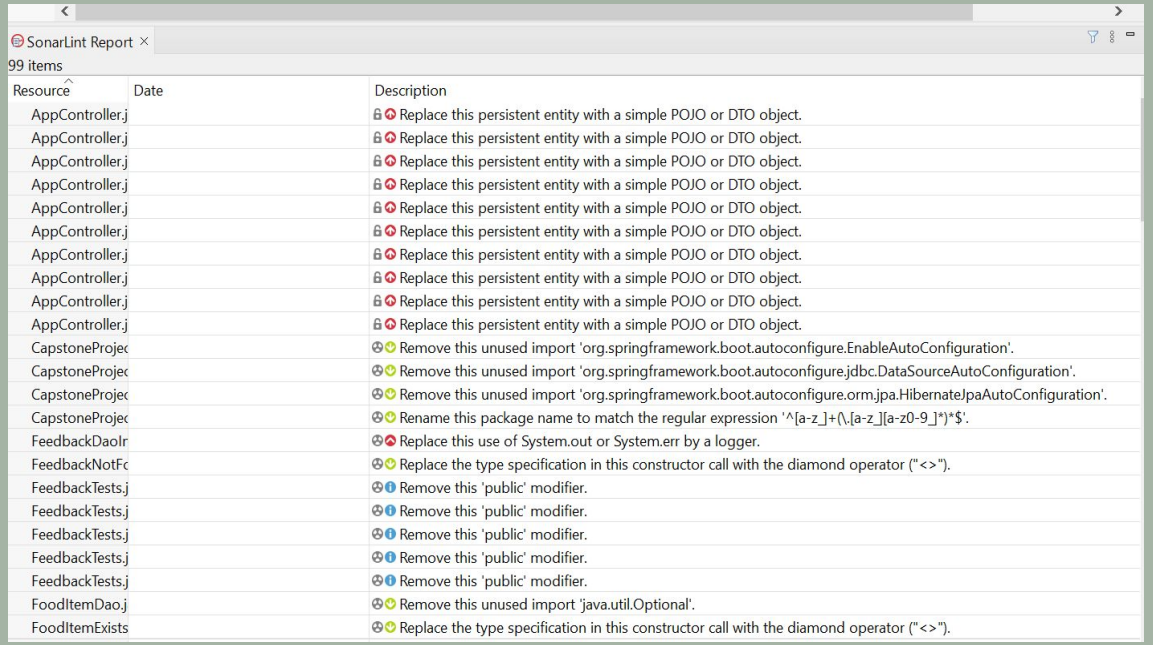
```java
@Test
public void deleteVendorByIdSuccessTest() throws Exception {
    Vendor vendor1 = new Vendor(1, 1, "Sonic", "Seattle");

    given(vendorRepo.findById(1)).willReturn(Optional.of(vendor1));
    doNothing().when(userRepo).deleteById(1);
    given(foodRepo.deleteByVendorId(1)).willReturn(1);

    assertTrue(vendorService.deleteVendorById(1));
}
```

Code Quality: SonarLint

- Used to receive a report of quality code.
- Helped to clean up code by improving naming conventions, removing unnecessary portions of code



VI. Technologies

Frontend

AngularJS

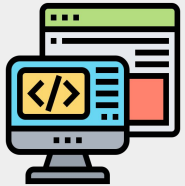
HTML

Typescript

CSS

Bootstrap

Backend



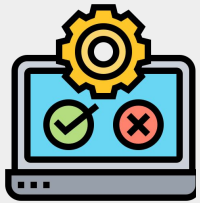
Spring

Java 11

Eureka

Maven

Backend Testing



Sonar

Postman

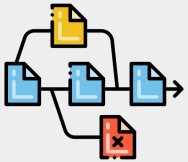
Junit

Mockito

JPA

Hibernate

Version Control



Git/Github

Database

MySQL

MySQL CLC

Comms

MS Teams

VII. Future Improvements

1. Optimize Data through Request and Response classes
2. Improve UI Design
3. Add Payment System
4. Add Location API
5. Rating System for food & vendors
6. Add offers for each vendor

Web Application Demonstration

Overall Takeaways

Q & A