

DTAP

Java Bootcamp

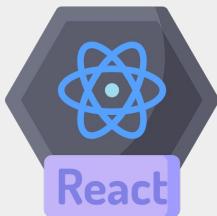
Spring 2023 Cohort

Shirley, Joshua, Jay,
Moise & Ahmed





Java Bootcamp



Firebase



Express  JS



Capstone Project:

Food Delivery App

Spring 2023 Java Cohort



Breakdown of Tasks

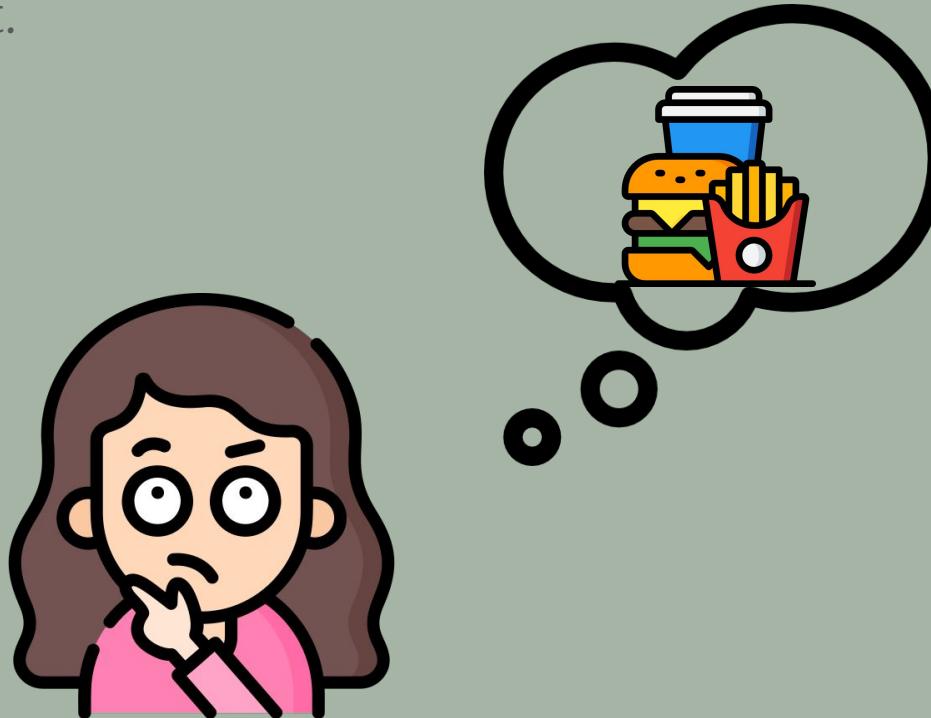
	Project Phases				
Team Member	Phase I: Planning	Phase II: Implementation	Phase III: Testing	Presentation Prep	
Shirley (Project Lead)	Design User Related Pages	<p>Backend:</p> <p>Lead Flow Diagrams Creation - Backend Diagrams - Frontend Diagrams - UI Design</p> <p>Frontend:</p> <ul style="list-style-type: none"> - Manage Github Repository - Set up Angular Project - Work on page routing - Writing Cart Services & component 		<p>Lead creation/revision of slides</p> <p>Set up Presentation and Slides Flow</p> <p>Work on Backend Planning Slides</p>	<p>Backend:</p> <p>Writing FoodItem: - Business Logic - Exceptions - Advice</p> <p>Frontend: Vendor: - pages - components - services</p>
	Designing Admin Dashboard page	<p>Backend:</p> <p>Setting Up Microservices</p> <p>Writing AppDelegate.java</p> <p>Frontend: Admin, Feedback, & Offers: - pages - components - services</p>		Work on Microservices slides	
Jay	Designing Admin Dashboard page	<p>Backend:</p> <p>Writing AppController.java</p> <p>Frontend: Login, Register, Authentication, & Cart: - pages - components - services</p>			<p>Backend:</p> <p>Writing User: - Business Logic - Exceptions - Advice</p> <p>Frontend: FoodItem: - components - services</p>
		Junit Testing Sonar Code Quality	Work on testing slides		
Joshua					Prepare for Demo of web application
<p>Moise</p> <p>Designing Vendor Pages</p> <p>Ahmed</p> <p>Revising Flow Diagrams</p> <p>Designing Login/Register Pages</p>					

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.

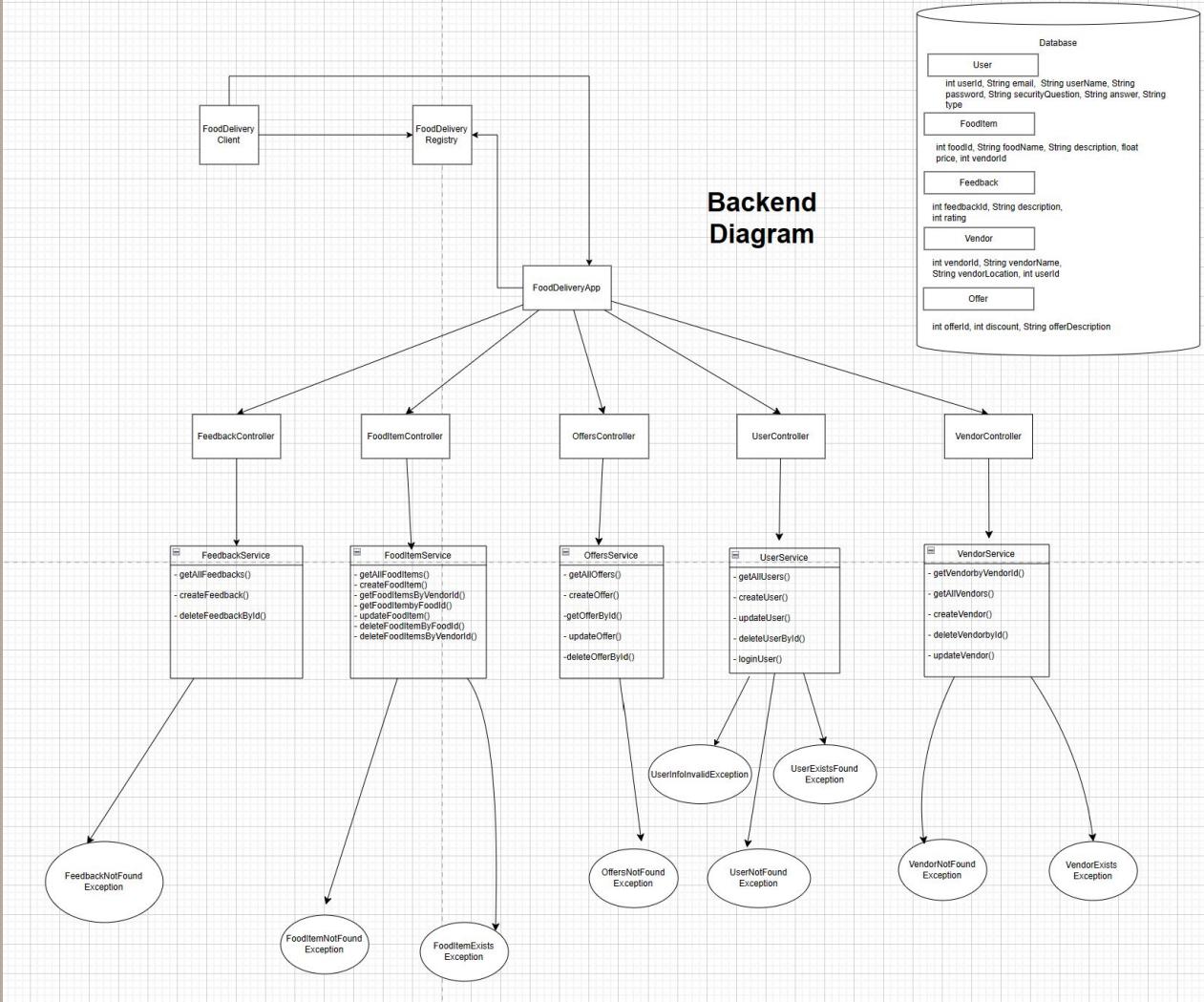




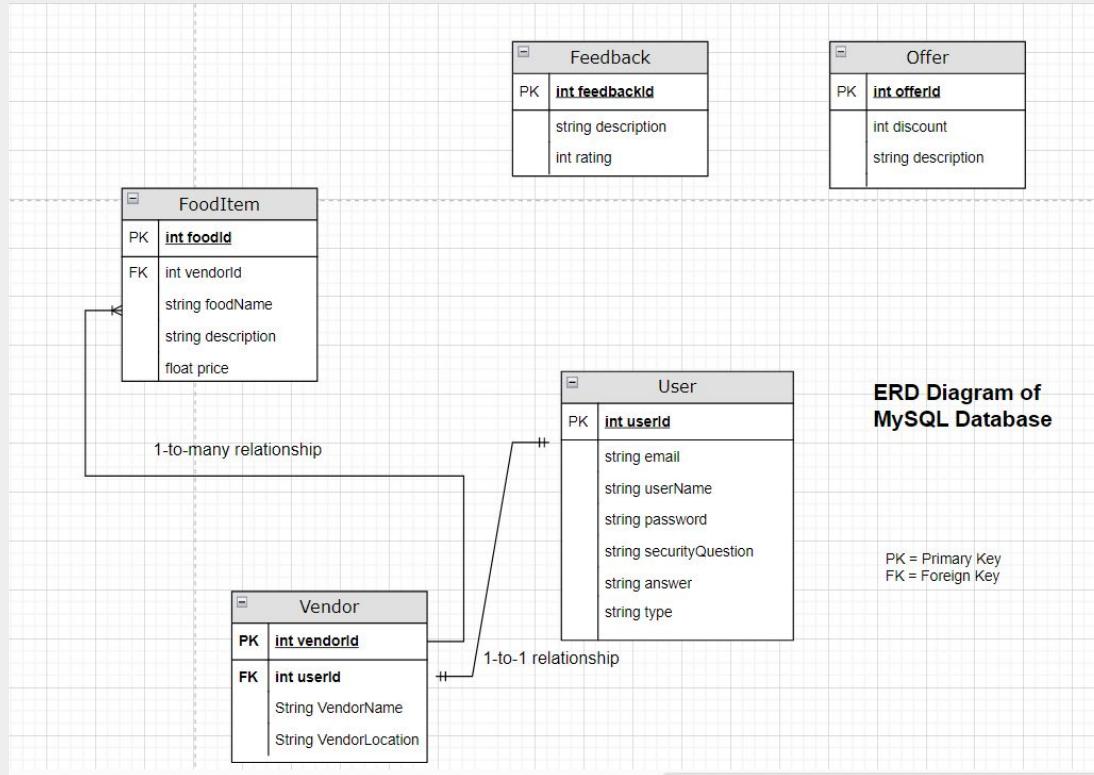
Phase I: Planning

Phase I: Backend Planning

- Written in Java



Entity Relationship Diagram (ERD)



Why MySQL?:

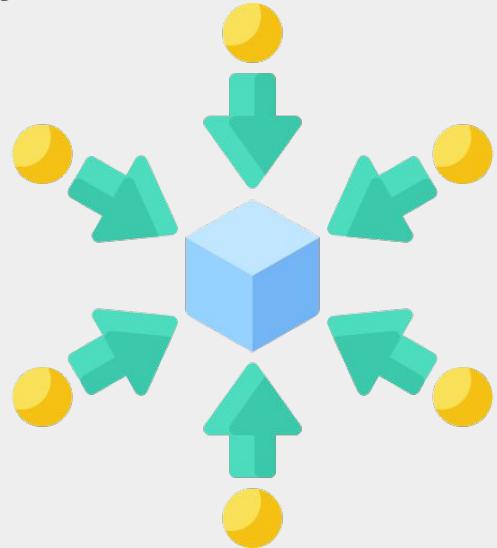
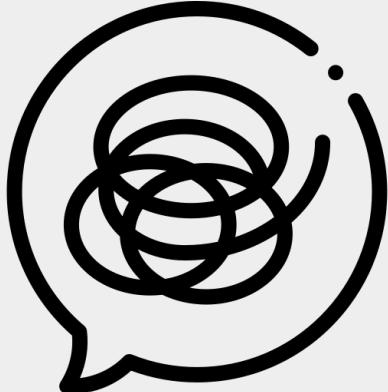
- Supports foreign keys
- Optimized for high performance
- Vertical scalability



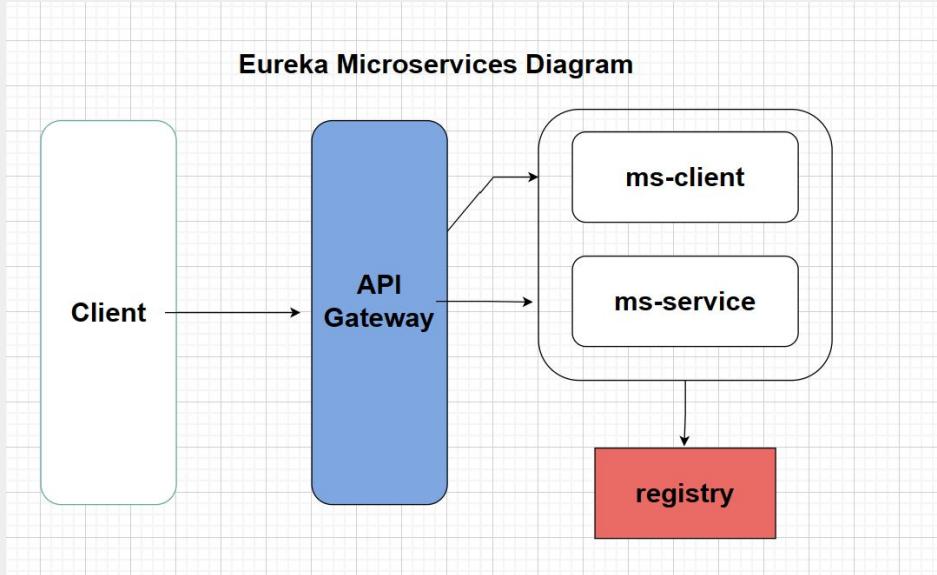
Phase II: Implementation

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

```
7 @SpringBootApplication  
8 @EnableEurekaServer  
9 public class OnlineFoodDeliveryMsRegistryApplication {  
10  
11     public static void main(String[] args) {  
12         SpringApplication.run(OnlineFoodDeliveryMsRegistryApplication.class, args);  
13     }  
14  
15 }
```

```
1 server.port=7778  
2 eureka.instance.hostname=localhost  
3 eureka.client.registerWithEureka=false  
4 eureka.client.fetchRegistry=false
```

Eureka Service configuration

```
13 @SpringBootApplication .  
14 @EnableJpaRepositories("com")  
15 @ComponentScan("com")  
16 @EntityScan("com")  
17 @EnableEurekaClient  
18 public class CapstoneProjectApplication {  
19  
20     public static void main(String[] args) {  
21         SpringApplication.run(CapstoneProjectApplication.class, args)  
22     }  
23  
24         1spring.datasource.url=jdbc:mysql://localhost:3306/FoodDeliveryDB?serverTimezone=UTC  
2spring.datasource.username=root  
3spring.datasource.password=root123  
4spring.datasource.platform=mysql  
5spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
6spring.jpa.hibernate.ddl-auto=update  
7spring.jpa.show-sql=true  
8spring.application.name=onlinefooddelivery-ms-service  
9eureka.client.serviceUrl.defaultZone=http://localhost:7778/eureka  
10server.port=${PORT:0}  
11eureka.instance.instance-id=${spring.application.name}:{spring.application.instance_id: ${random.value}}
```

Eureka Client configuration

```
10 @SpringBootApplication
11 @ComponentScan("com")
12 @EntityScan("com")
13 @EnableHystrix
14 @EnableEurekaClient
15 public class OnlineFoodDeliveryMsClientApplication {
16
17     public static void main(String[] args) {
18         SpringApplication.run(OnlineFoodDeliveryMsClientApplication.class, args);
19     }
20 }
```

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

Client consuming a Service registered on Eureka

```
19 @Service
20 public class AppDelegate {
21     @LoadBalanced
22     @Bean
23     public RestTemplate template() {
24         return new RestTemplate();
25     };
26
27     @Autowired
28     public RestTemplate template;
29
30     // USER
31
32     public User createUser(User user) {
33         User data = template.exchange("http://onlinefooddelivery-ms-service/foodDeliveryApp/createUser",
34             HttpMethod.POST, new HttpEntity<User>(user), User.class).getBody();
35
36     return data;
37 }
```

```
-->
23     @RestController
24     @RequestMapping("/foodDeliveryAppClient")
25     @CrossOrigin("*")
26     public class AppController {
27
28     @Autowired
29     private AppDelegate dao;
30
31     // User
32
33     @PostMapping("/createUser")
34     public User createUser(@RequestBody User user) {
35         return dao.createUser(user);
36     }
```

Eureka Server Dashboard

The screenshot shows the Spring Eureka Server Dashboard running at `localhost:7778`. The top navigation bar includes links for `HOME` and `LAST 1000 SINCE STARTUP`.

System Status

Environment	test	Current time	2023-03-11T00:47:17 -0500
Data center	default	Uptime	00:01
		Lease expiration enabled	false
		Renews threshold	6
		Renews (last min)	0

DS Replicas

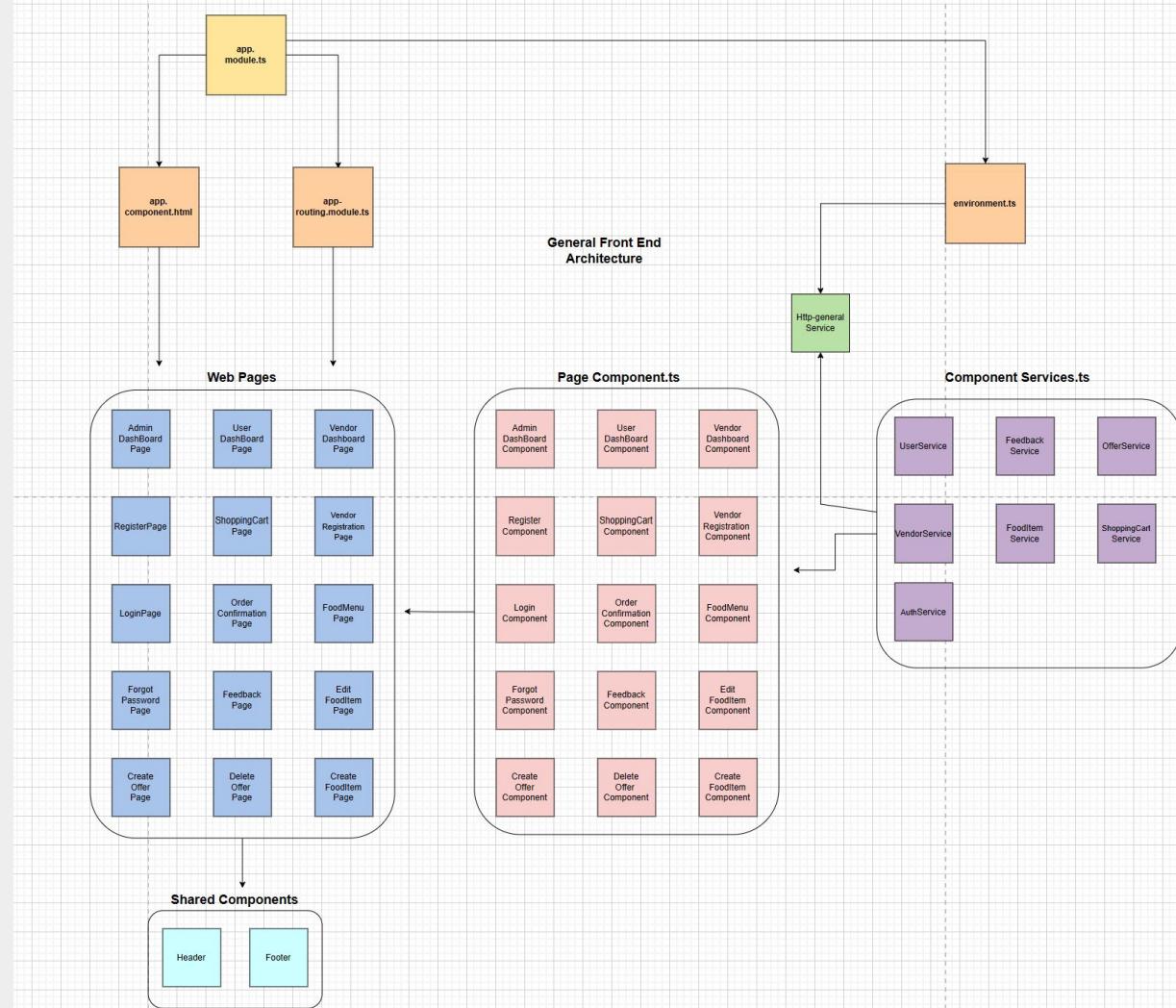
Instances currently registered with Eureka

Application	Availability		
	AMIs	Zones	Status
ONLINEFOODDELIVERY-MS-CLIENT	n/a (1)	(1)	UP (1) - host.docker.internal:onlinefooddelivery-ms-client:8055
ONLINEFOODDELIVERY-MS-SERVICE	n/a (2)	(2)	UP (2) - onlinefooddelivery-ms-service:(spring.application.instance_id: 51ef73a51c523f00b4539c65c55487f8) , onlinefooddelivery-ms-service:(spring.application.instance_id: 368a4250208997b8e16ece800ef1b899)

All instances registered on localhost

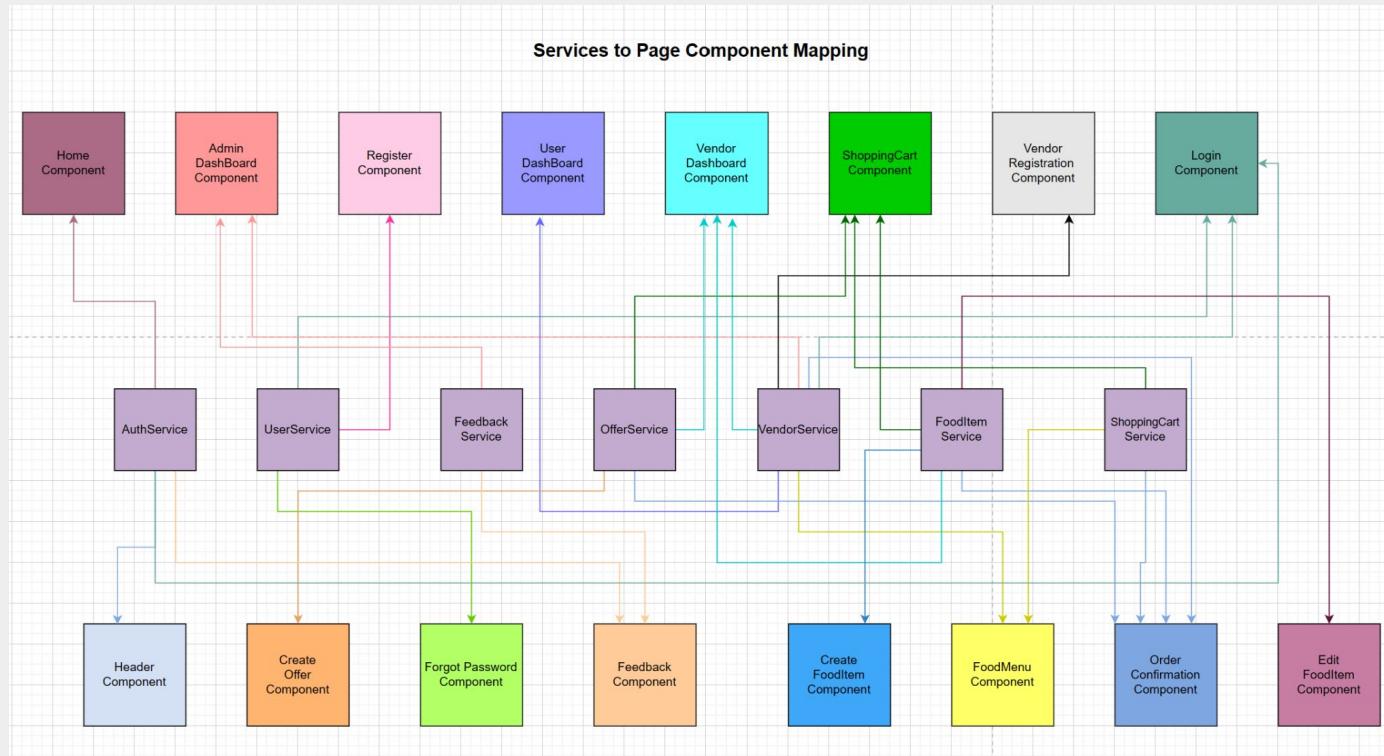
Front End Diagrams

- Angular
- Visual aid for code flow



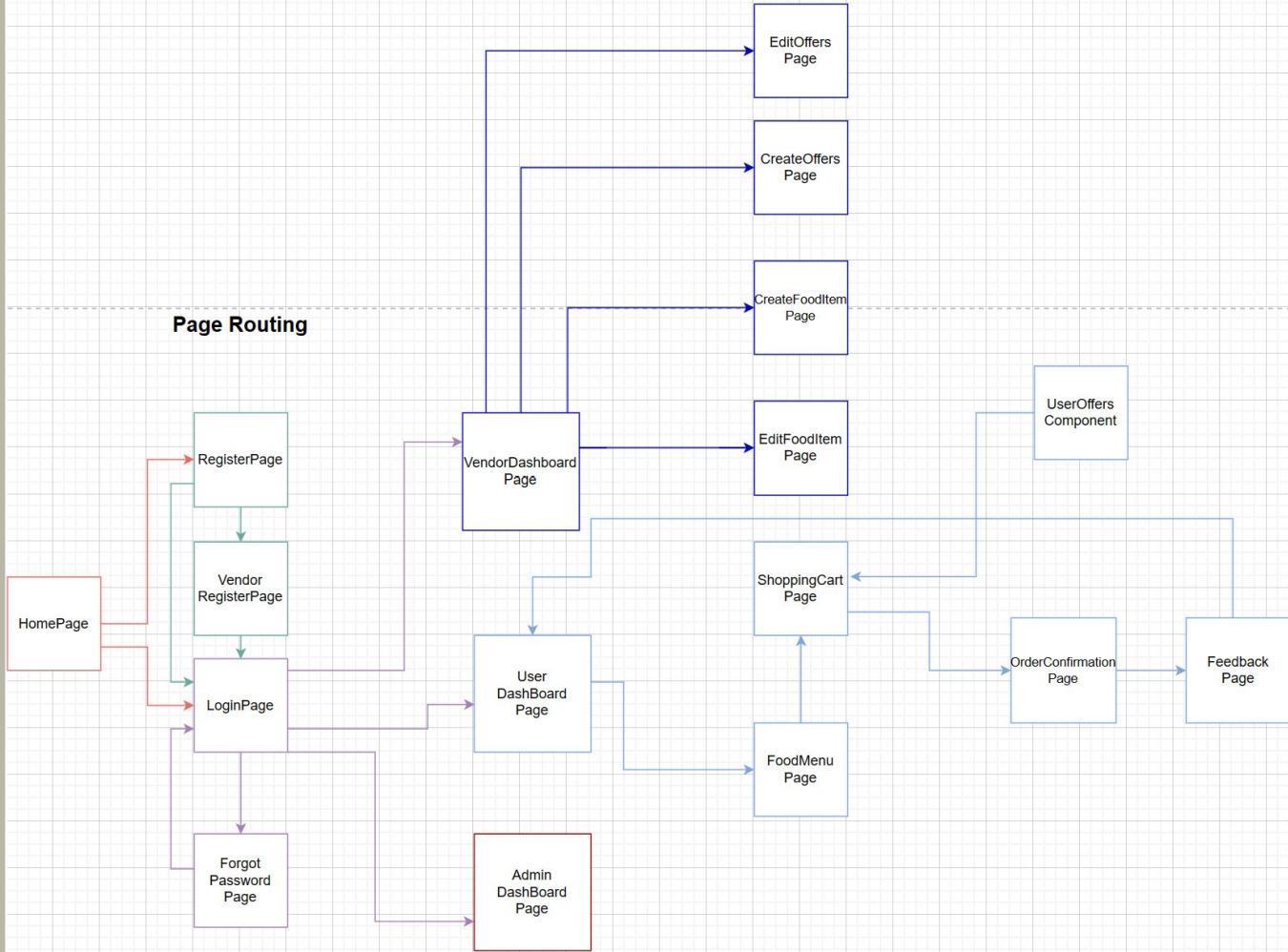
Front End Diagrams

- Shows services utilized by each page component



Front End Diagrams

- Provides clarity of website navigation



Web Application: User Authentication

Food Delivery App

Home Page



Spring 2023 LTIMindtree DTAP Java Cohort

LOGIN SIGN UP FOR FREE

LOGIN REGISTER

McDonald's

Location: New York, NY

Food Name	Price	Description	Action
Fillet of Fish Meal	\$10	Combo meal with a medium fountain beverage and fries	[Edit] [Delete]
Cheeseburger Happy Meal	\$5	Kids meal with small side of fries, milk, and apple slices	[Edit] [Delete]
Oreo McFlurry	\$6	Sweet frozen treat with oreo pieces	[Edit] [Delete]
10 pc Spicy McNuggets	\$7	Crispy, spicy McNuggets. Limited Time only.	[Edit] [Delete]

CREATE FOOD ITEM

Discounts

Discount	Offer Description	Action
15	15% off on orders \$45 and over	[Edit] [Delete]
30	30% off on orders placed after 3:00pm EST	[Edit] [Delete]
50	50% for a limited time only!	[Edit] [Delete]

CREATE OFFER

Login

Username

Password

Password must be length of 6 or more

LOGIN

REGISTER

FORGOT PASSWORD?

Web Application: Ordering Food

User Dashboard

Vendor Name	Vendor Location	Action
McDonald's	New York, NY	<button>VIEW</button>
Tana Thai	Brooklyn, NY	<button>VIEW</button>
Shake Shack	New York, NY	<button>VIEW</button>
La Catrina	New York, NY	<button>VIEW</button>
Luciano's	New York, NY	<button>VIEW</button>
Morrison Baked Goods	Brooklyn, NY	<button>VIEW</button>
Szechuan Garden	Astoria, NY	<button>VIEW</button>
Kungfu Tea	Astoria, NY	<button>VIEW</button>

Food Delivery App

LOGIN SIGN UP FOR FREE

McDonald's
New York, NY

Food ordered

Food Item	Description	Price	Delete
Cheeseburger Happy Meal	Kids meal with small side of fries, milk, and apple slices	\$5	<button>DELETE</button>
Oreo McFlurry	Sweet frozen treat with oreo pieces	\$6	<button>DELETE</button>
Fillet of Fish Meal	Combo meal with a medium fountain beverage and fries	\$10	<button>DELETE</button>
Fillet of Fish Meal	Combo meal with a medium fountain beverage and fries	\$10	<button>DELETE</button>

REMOVE ALL ITEMS

\$31

Available Offers (% off)

BACK TO MENU SUBMIT ORDER

Food Delivery App

localhost:4200 says

Fillet of Fish Meal added to cart

OK

Food Name	Description	Price	Action
Fillet of Fish Meal	Combo meal with a medium fountain beverage and fries	\$10	<button>ADD</button>
Cheeseburger Happy Meal	Kids meal with small side of fries, milk, and apple slices	\$5	<button>ADD</button>
Oreo McFlurry	Sweet frozen treat with oreo pieces	\$6	<button>ADD</button>
10 pc Spicy McNuggets	Crispy, spicy McNuggets. Limited Time only.	\$7	<button>ADD</button>

BACK TO DASHBOARD GO TO CART



Phase III: Testing

Postman

- Postman was used for API and functionality testing.

The screenshot shows the Postman application interface. At the top, there is a header bar with several tabs labeled 'localhost' and a dropdown menu 'No Environment'. On the right side of the header are icons for 'Examples (0)', a gear for settings, and a user profile.

The main area displays a request configuration for a 'POST' method to the endpoint 'localhost:8022/foodDeliveryApp/createUser'. The 'Body' tab is selected, showing the following JSON payload:

```
1 [ {  
2   "userName": "user",  
3   "password": "user123",  
4   "securityQuestion": "type of dog",  
5   "answer": "dberman",  
6   "type": "user"  
7 } ]
```

Below the body, there is a large empty box labeled 'Response' where the API response will be displayed.

Postman

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

The screenshot shows a POST request to `http://localhost:8055/foodDeliveryAppClient/createUser`. The request body is a JSON object:

```
1 {  
2   "email": "jhsini@uci.edu",  
3   "userName": "jhsini",  
4   "password": "pass",  
5   "securityQuestion": "maiden",  
6   "answer": "wang",  
7   "type": "user"  
8 }
```

The response status is 200 OK, and the response body is identical to the request body.

The screenshot shows a GET request to `http://localhost:8055/foodDeliveryAppClient/getAllUsers`. The response body is a JSON array of three user objects:

```
12 [  
13   {  
14     "userId": 4,  
15     "email": "username5@mail.com",  
16     "userName": "username5",  
17     "password": "password46",  
18     "securityQuestion": "Where was your mother born?",  
19     "answer": "Damascus Syria",  
20     "type": "Vendor"  
21   },  
22   {  
23     "userId": 5,  
24     "email": "username6@mail.com",  
25     "userName": "username6",  
26     "password": "password6",  
27     "securityQuestion": "Where was your mother born?",  
28     "answer": "Damascus Syria",  
29     "type": "Vendor"  
30   },  
31   {  
32     "userId": 6,  
33     "email": "jhsini@uci.edu",  
34     "userName": "jhsini",  
35     "password": "pass",  
36     "securityQuestion": "maiden",  
37     "answer": "wang",  
38     "type": "user"  
39   ]
```

Junit & Mockito

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

The screenshot shows the Eclipse IDE interface with the Package Explorer and JUnit view. The JUnit view displays test results: 'Finished after 1.754 seconds', 'Runs: 12/12', 'Errors: 0', and 'Failures: 0'. Below this, a tree view shows the 'VendorTests [Runner: JUnit 5] (1.556 s)' suite with its individual test cases and execution times. To the right, the code editor displays the 'VendorTests.java' file, which contains Mockito annotations and test methods for vendor operations like creation, update, and retrieval.

```
56  @Test
57  public void createVendorSuccessTest() throws Exception {
58      List<Vendor> vendorList = new ArrayList<>();
59      Vendor vendor1 = new Vendor(1, 1, "McDonalds", "Long Beach");
60      Vendor vendor2 = new Vendor(2, 2, "In-n-out", "Fremont");
61      vendorList.add(vendor1);
62
63      given(vendorRepo.save(vendor1)).willReturn(vendor1);
64      assertEquals(vendorService.createVendor(vendor1), vendor1);
65
66      given(vendorRepo.findAll()).willReturn(vendorList);
67      given(vendorRepo.save(vendor2)).willReturn(vendor2);
68      assertEquals(vendorService.createVendor(vendor2), vendor2);
69  }
70
71  @Test
72  public void getVendorListTest() throws Exception {
73      List<Vendor> vendorList = new ArrayList<>();
74      Vendor vendor1 = new Vendor(1, 1, "Jack in the Box", "Sacramento");
75      Vendor vendor2 = new Vendor(2, 2, "Breakfast Republic", "Irvine");
76      vendorList.add(vendor1);
77      vendorList.add(vendor2);
78
79      given(vendorRepo.findAll()).willReturn(vendorList);
80
81      assertEquals(vendorService.getAllVendors(), vendorList);
82  }
83
84  @Test
85  public void getVendorByIdSuccessTest() throws Exception {
86      Vendor vendor1 = new Vendor(1, 1, "Wendys", "Phoenix");
87
88      given(vendorRepo.findById(1)).willReturn(Optional.of(vendor1));
89
90      assertEquals(vendorService.getVendorById(1), vendor1);
91  }
92}
```

Junit & Mockito

```
@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);  
}
```

```
@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"));  
}
```

Junito & Mockito

```
@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);  
}
```

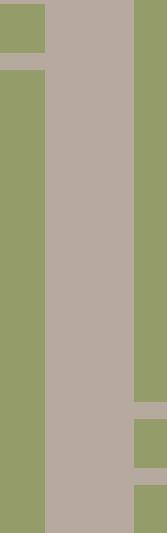
```
@Test  
public void deleteVendorByIdSuccessTest() throws Exception {  
    Vendor vendor1 = new Vendor(1, 1, "Sonic", "Seattle");  
  
    given(vendorRepo.findById(1)).willReturn(Optional.of(vendor1));  
    doNothing().when(userRepo).deleteById(1);  
    given(foodRepo.deleteByVendorId(1)).willReturn(1);  
  
    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

Resource	Date	Description
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
AppController.j		ⓘ ⚡ Replace this persistent entity with a simple POJO or DTO object.
CapstoneProject		ⓘ ⚡ Remove this unused import 'org.springframework.boot.autoconfigure.EnableAutoConfiguration'.
CapstoneProject		ⓘ ⚡ Remove this unused import 'org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration'.
CapstoneProject		ⓘ ⚡ Remove this unused import 'org.springframework.boot.autoconfigure.orm.jpa.HibernateJpaAutoConfiguration'.
CapstoneProject		ⓘ ⚡ Rename this package name to match the regular expression '^[a-z]+([a-z][a-zA-Z]*\$)'.
FeedbackDaorl		ⓘ ⚡ Replace this use of System.out or System.err by a logger.
FeedbackNotFc		ⓘ ⚡ Replace the type specification in this constructor call with the diamond operator ("<>").
FeedbackTests.j		ⓘ ⚡ Remove this 'public' modifier.
FeedbackTests.j		ⓘ ⚡ Remove this 'public' modifier.
FeedbackTests.j		ⓘ ⚡ Remove this 'public' modifier.
FeedbackTests.j		ⓘ ⚡ Remove this 'public' modifier.
FoodItemDao.j		ⓘ ⚡ Remove this unused import 'java.util.Optional'.
FoodItemExist		ⓘ ⚡ Replace the type specification in this constructor call with the diamond operator ("<>").

VI. Technologies

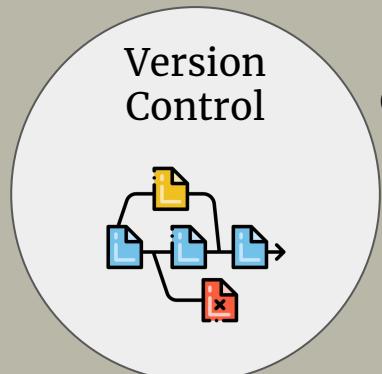




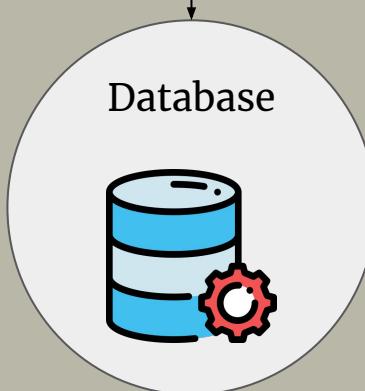
- Spring
- Java 11
- Eureka
- Maven



- Sonar
- Postman
- Junit
- Mockito



Git/Github



- Hibernate
- JPA
- MySQL
- MySQL CLC



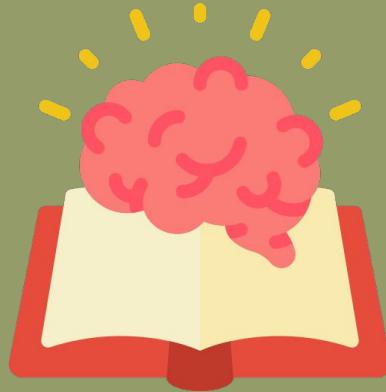
MS Teams

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