

# Online Food Order application MVP

## CONTENT

- Overview
- Features
- System Architecture
- Application flow
- Layered design
- System Environment
- Test driven development

# Overview

To launch an Online Food Order System that Allows Users , consumers & service providers to interact in real-time and want to support both Mobile App and Web-based applications.

We are concerned about the following:

- Adding/Deleting/Retrieving/update Restaurants, menu and items.
- Effective distribution of load.
- configurable database/s and data access layer to yield high performance and throughput.
- Design for easy Onboarding and search-ability of restaurants

# Features of Users

Adding/Deleting/Retrieving/update Restaurants. Allow users ( Customers , HotelOwners etc..) to perform ..

## Restaurants

Retrieve all Restaurants

Get details of specific Restaurants

Delete a Restaurants

Create a new Restaurant

Update Restaurant details

# Features of Users

Adding/Deleting/Retrieving/update menu. Allow users ( Customers , HotelOwners etc..) to perform ..

## Menu

Retrieve all Menus

Get details of specific Menu

Delete a Menu

Create a new Menu

Update Menu details

# Core Features of Users

Adding/Deleting/Retrieving/update items. Allow users ( Customers , HotelOwners etc..) to perform ..

## Item

Retrieve all Items

Get details of specific Item

Delete a Item

Create a new Item

Update Item details

# Features of Users

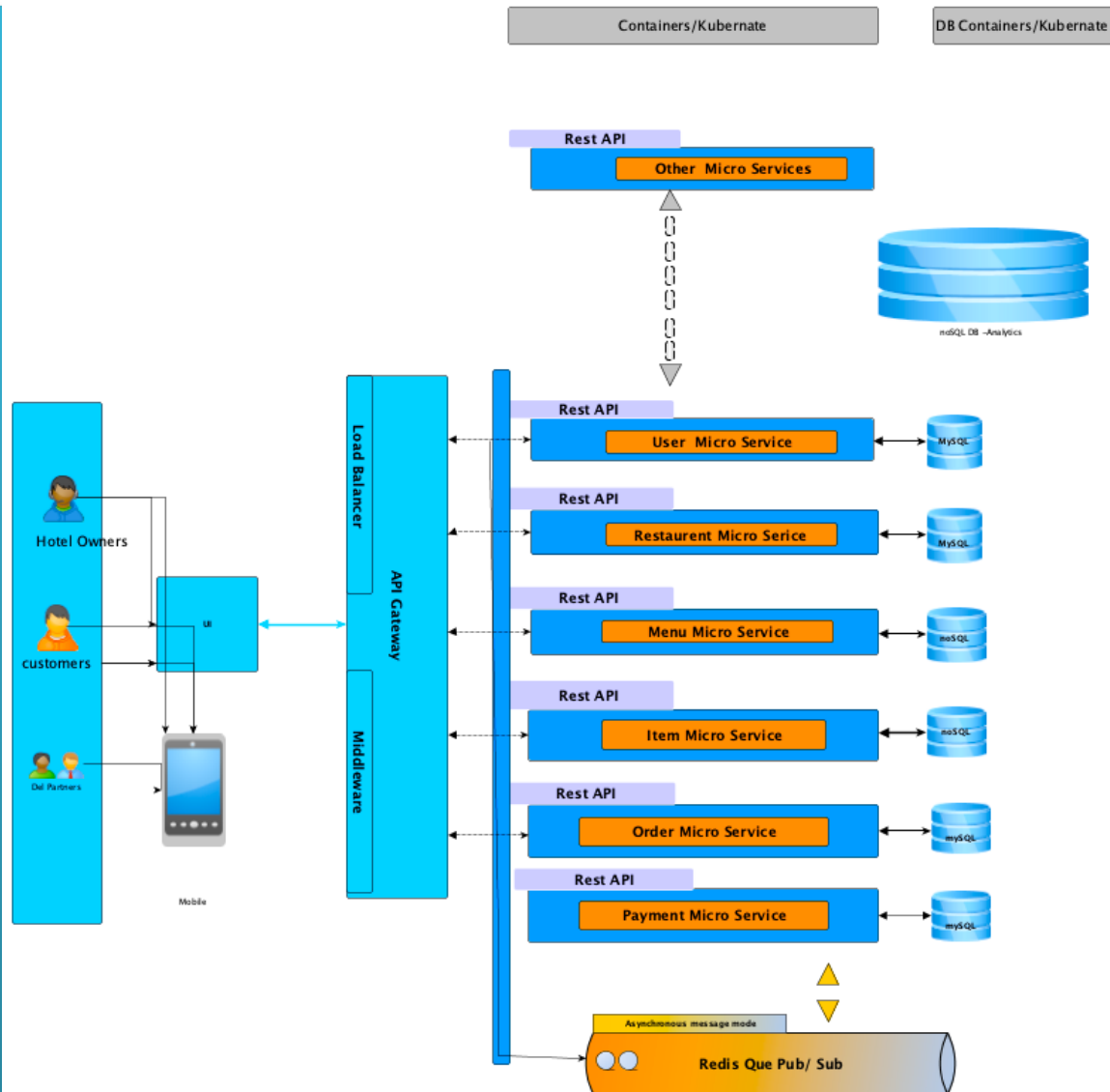
## Order

The user can place an order of selected food items . They just need to cross-verify their preferred dish, and proceed check-out.

## Payment

You provide the users with multiple payment options like credit/debit cards, different wallets like Google Pay, Paytm, Phonepe, UPI, etc

# System Architecture





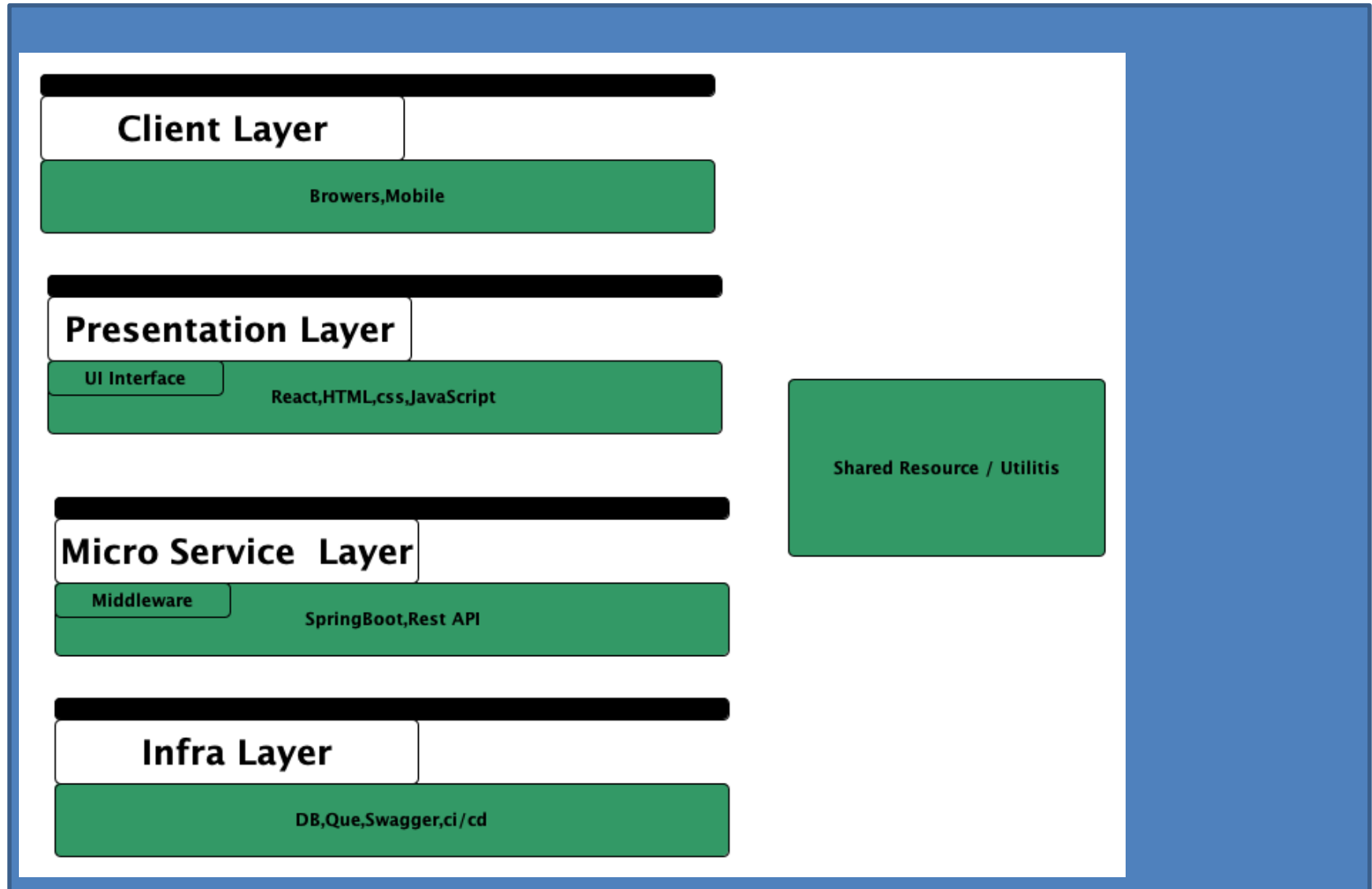
# Application Flow

Proposed above is a high-level architecture for Online Food Delivery systems. This proposed architecture is generic and it can be deployed to any of cloud provider like AWS/GCP/Azure.

We are considering here micro-services-based architecture. Different services are listed in the architecture diagram

- All requests made from a mobile app or UI will go to different services via the API gateway. API gateway will take care of load balancing and routing requests to services. This will authenticate and authorise the user and send back the token ID. This token is used for further communication
- Different services like, user, Restaurant info , MenuService and item service, order service, payment service will use transactional databases. We will use the MySQL relational database. This is a highly scalable database service to manage users and concurrent orders etc.
- Information about different restaurants, their menu, price, offers, etc will be stored in JSON format ( noSQL DB preferred )

# Layered Design and Technologies



# System Environment

The Online Food Order Web application is built on the standard cloud platform structure as described in the following table.

Specification	Description
Windows Browsers	Internet Explorer 10 / 11, Firefox 62 Chrome on Window69.0.x
Mac Browsers	Safari 12, Firefox 62.x.x, Chrome (69.0.x)
DB	MySQL
WebServer	Apache Tomcat ( Inbuilt Tomcat SpringBoot )
Programming and Web	Java8, SpringBoot, SpringCore
I D E , A P I , D a t a format, Testing	Eclipse, JSON, Swagger, JUnit Mockito, Apache Maven, Redis
HTTP Server and Authentication	OAuth2.0, NGINX, Redis Messaging (Pub/Sub)
UI	HTML, CSS, JavaScript, React

# Test Driven Development

Automated testing is implemented using JUnit , which provides functionality for setting up tests, verifying data in the MicroServices, DB and other useful re-usable components. JUnit framework tool helps developers to perform unit testing and practice test-driven development.

Below are some of the JUnit assertions helps to write tests:

*Assert.assertEquals()*

*Assert.assertTrue()*

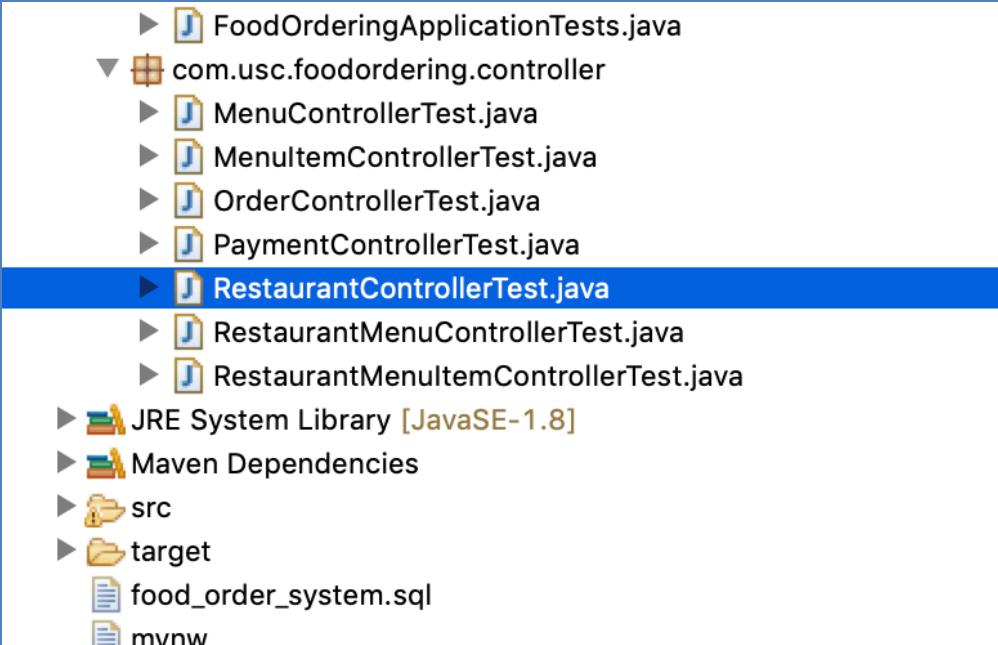















*Assert.assertFalse()*

*Assert.assertNull()*

# Test Driven Development

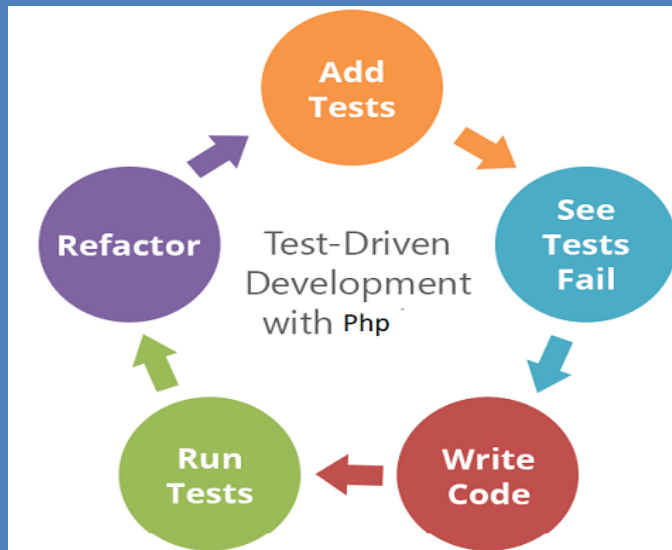
Below is the Sample project layout (Composer project layout)

As per the above src/tests, Development Teams separate the test code from the source code.

- 
- ▶  FoodOrderingApplicationTests.java
  - ▼  com.usc.foodordering.controller
    - ▶  MenuControllerTest.java
    - ▶  MenuItemControllerTest.java
    - ▶  OrderControllerTest.java
    - ▶  PaymentControllerTest.java
    - ▶  RestaurantControllerTest.java
    - ▶  RestaurantMenuControllerTest.java
    - ▶  RestaurantMenuItemControllerTest.java
  - ▶  JRE System Library [JavaSE-1.8]
  - ▶  Maven Dependencies
  - ▶  src
  - ▶  target
  - ▶  food\_order\_system.sql
  - ▶  mvnw

# Test Driven Development Lifecycle

- Add tests for the functionality.
- Run tests to fail.
- Write code according to the identified errors.
- Run the tests again to see if the test fails or passes
- Re-factor the code and follow the process again.



# Deployment CI/CD

GITHUB, Sonar , DOCKER ,KUBERNATES etc....

GitHub Location

<https://github.com/intellect82/foodapp.git>

# Companies which may benefit of this Architecture

