

**WORKSHOP N° 4**  
**SOFTWARE ENGINEERING II**

**Presented by:**

Michael Stiven Betancourt Gelves

Cristhian Yamith Cely Oliveros

**Professor:**

Carlos Andres Sierra Virguez

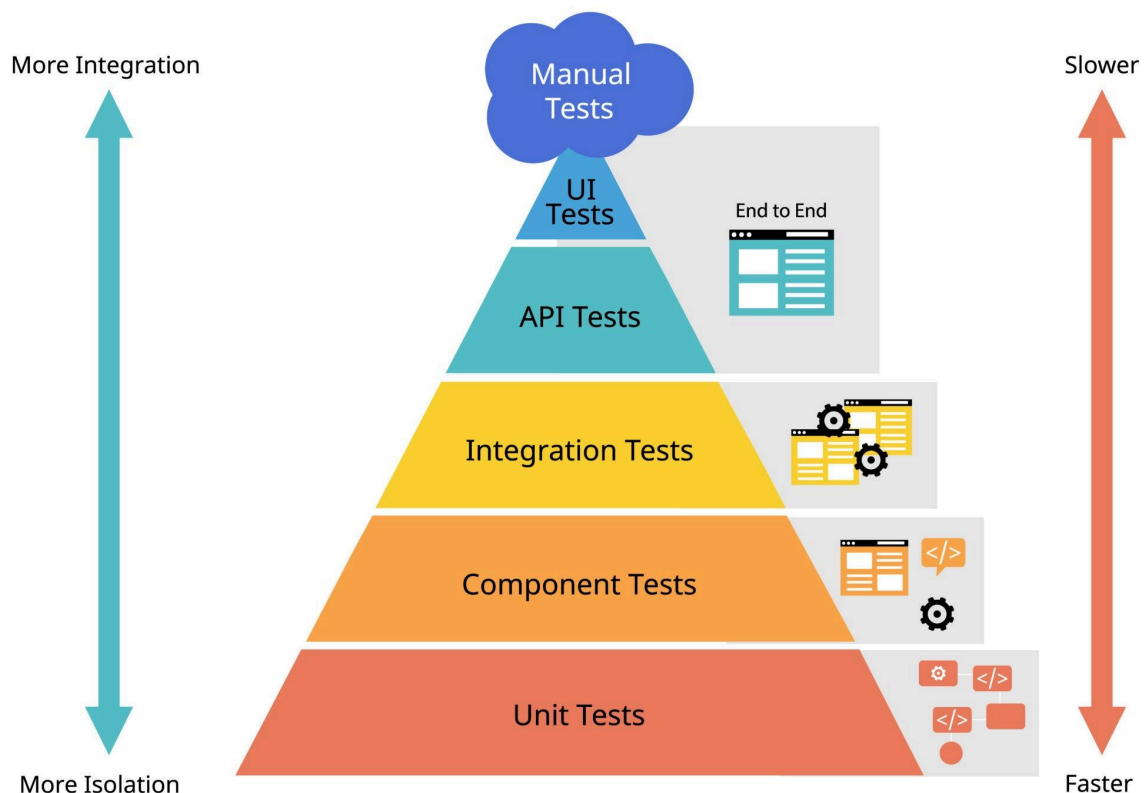
November 30th 2025



**Universidad Nacional de Colombia**  
**Engineering Faculty**  
**2025**

# Workshop 4: Methodology and Deliverables

## 1. Testing Strategy & Execution



### 1.1 Unit Tests

The library that was used for testing was Mockito, which is a proven to work library for unit testing. All endpoints had unit tests as per requirements by the project leader to guarantee that each endpoint was fully functional. For both front-end applications, we used Jest as our testing framework alongside `@testing-library/react-native` to validate component rendering, behavior, and user interactions. In the Expo-based project, we also integrated `jest-expo` to ensure full compatibility with Expo modules and native features. This setup allowed us to detect issues early, maintain UI consistency, and ensure reliable component behavior across both applications.

- **Backend Component:**

AuthService:

```

✓ Test Results 419 ms ✓ 1 test passed 1 test total, 419 ms
> Task :compileJava UP-TO-DATE
> Task :processResources UP-TO-DATE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses
20:30:32.295 [Test worker] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not
20:30:32.355 [Test worker] INFO org.springframework.boot.test.context.SpringBootTestContextBootstrapper -- Found @SpringB

  ____
 /  __ \  /  __ \  /  __ \  /  __ \
( (  ) /  ( (  ) /  ( (  ) /  ( (  ) /
 \ \_/ /  \ \_/ /  \ \_/ /  \ \_/ /
  _/_/___/_/_/___/_/_/___/_/_/___/_/_/
=====|_|=====|_|_/___/_/_/___/_/_/

:: Spring Boot ::                (v4.0.0-RC2)

2025-11-30T20:30:32.613-05:00 INFO 88641 --- [GeoService] [   Test worker] c.c.G.GeoServiceApplicationTests : S
2025-11-30T20:30:32.615-05:00 INFO 88641 --- [GeoService] [   Test worker] c.c.G.GeoServiceApplicationTests : M
2025-11-30T20:30:33.032-05:00 INFO 88641 --- [GeoService] [   Test worker] s.d.r.c.RepositoryConfigurationDelegate : B

```

- **Frontend Component:**

themed-button.test.tsx:

```

import React from 'react';
import { render } from '@testing-library/react-native';
import { test, expect } from '@jest/globals';
import ThemedButton from './themed-button';

test('renders label', () => {
  const { getByText } = render(<ThemedButton>Tap</ThemedButton>);
  expect(getByText('Tap')).toBeTruthy();
});

```

```

_01_2492-debug-0.10g
PS C:\Users\yamit\OneDrive\Documents\College\IngesoftII\Click&MunchApp\frontend\mobile> npm run test --silent
FAIL presentation/theme/components/themed-button.test.tsx
  ● Test suite failed to run

    TypeError: jest: failed to cache transform results in: C:/Users/yamit/AppData/Local/Temp/jest-jest-transform-cache-d3e6b17766ddc2eb23a795f3d089eeb6-12533232bd0f05f65688e7a7764bf3fb/2e/setup_2e1859b39cdcc6917162e35091a46c499.map
    Failure message: onExit is not a function

      at writeFileSync (node_modules/write-file-atomic/lib/index.js:212:31)
      at writeCacheFile (node_modules/@jest/transform/build/index.js:711:33)
      at ScriptTransformer._buildTransformResult (node_modules/@jest/transform/build/index.js:387:7)
      at ScriptTransformer.transformSource (node_modules/@jest/transform/build/index.js:431:17)
      at ScriptTransformer._transformAndBuildScript (node_modules/@jest/transform/build/index.js:519:40)
      at ScriptTransformer.transform (node_modules/@jest/transform/build/index.js:558:19)

Test Suites: 1 failed, 1 total
Tests:      0 total
Snapshots:  0 total
Tim
Ran
Focus folder in explorer (ctrl + click)

```

## 1.2 Integration Tests

Once the Unit tests were validated, we proceeded to the integration tests, where the backend microservices and the frontend controllers were tested to work together to make sure that all requests were valid.

- **Integration Strategy:** Prior to continuing to deploy, the front and backend were executed in single machines. Tests were passed as per team metrics.
- **Key Test Snippet:**

One of the main and most important tests for integration to be valid was the communication between services. Here, there are several test snippets that were developed to make it work only within the backend services, but Frontend vs Backend were the most interesting to see:

When integrating Backend and Frontend, the “Find Nearby Restaurants” request was sent using a GET method:

GET localhost:8082/api/restaurants/nearby:

The body must have these components:

```
{
  "latitude": 52.75214,
  "longitude": -72.886547,
  "radiusInKm": 5
}
```

And the response was

```
{
  "id": 1,
  "name": "Burger Station",
  "description": "Best burgers in town",
  "phone": "300123456",
  "email": "contact@burger.com",
  "locationId": 2
}
```

```
},  
  
{  
  
  "id": 2,  
  
  "name": "Burger Station",  
  
  "description": "Best burgers in town",  
  
  "phone": "300123456",  
  
  "email": "contact@burger.com",  
  
  "locationId": 3  
  
},  
  
{  
  
  "id": 3,  
  
  "name": "Burger Station",  
  
  "description": "Best burgers in town",  
  
  "phone": "300123456",  
  
  "email": "contact@burger.com",  
  
  "locationId": 4  
  
},  
  
{  
  
  "id": 4,  
  
  "name": "Burger Station",  
  
  "description": "Best burgers in town",  
  
  "phone": "300123456",  
  
  "email": "contact@burger.com",  
  
  "locationId": 5  
  
},  
  
{
```

```
"id": 5,

"name": "Burger Station",

"description": "Best burgers in town",

"phone": "300123456",

"email": "contact@burger.com",

"locationId": 6
},

{

  "id": 6,

  "name": "Burger Station",

  "description": "Best burgers in town",

  "phone": "300123456",

  "email": "contact@burger.com",

  "locationId": 7
},

{

  "id": 7,

  "name": "Burger Station",

  "description": "Best burgers in town",

  "phone": "300123456",

  "email": "contact@burger.com",

  "locationId": 8
},

{

  "id": 8,

  "name": "Burger Station",
```

```
[{"description": "Best burgers in town",  
  
  "phone": "300123456",  
  
  "email": "contact@burger.com",  
  
  "locationId": 9  
}]
```

In the integration tests.

### 1.3 Acceptance Tests (User Stories)

Some of the key user stories that were tested are:

- **User Story 1:**

US-01: As a Customer, I want to register and log in so I can use the platform.

The acceptance criteria here was based on the following:

Given the registration screen, when I enter valid details, then my account is created.

Given the login screen, when I enter valid credentials, then I am authenticated.

Given invalid credentials, when I try to log in, then I receive an error message.

- **User Story 2:**

US-02: As a Restaurant Manager, I want to create and manage my restaurant account.

The acceptance criteria here was based on the following:

Given the registration form, when I submit valid business data, then the account is created and pending approval.

Given an approved restaurant account, when I update information, then it is saved.

## 2. Deployment Configuration

### 2.1 Dockerfiles

The following dockerfile was created to orchestrate the creation of all databases at the same time using a single host but several microservices:

This is the Docker-compose.yml file used to compose the entire database on a per-microservice basis.

version: '3.8'

```
services:

  auth-db:

    image: postgres:16

    container_name: auth-db

    restart: always

    environment:

      POSTGRES_DB: auth_db

      POSTGRES_USER: mike

      POSTGRES_PASSWORD: secret

    ports:

      - "5433:5432"

    volumes:

      - auth_data:/var/lib/postgresql/data

  restaurant-db:

    image: postgres:16

    container_name: restaurant-db

    restart: always
```



environment:

POSTGRES\_DB: restaurant\_db

POSTGRES\_USER: mike

POSTGRES\_PASSWORD: secret

ports:

- "5434:5432"

volumes:

- restaurant\_data:/var/lib/postgresql/data

geo-db:

image: postgis/postgis:16-3.4

container\_name: geo-db

restart: always

environment:

POSTGRES\_DB: geo\_db

POSTGRES\_USER: mike

POSTGRES\_PASSWORD: secret

ports:

- "5435:5432"

volumes:

- geo\_data:/var/lib/postgresql/data

menu-db:

image: postgres:16

container\_name: menu-db

```
restart: always

environment:

  POSTGRES_DB: menu_db

  POSTGRES_USER: mike

  POSTGRES_PASSWORD: secret

ports:

  - "5436:5432"

volumes:

  - auth_data:/var/lib/postgresql/data

volumes:

auth_data:

restaurant_data:

geo_data:

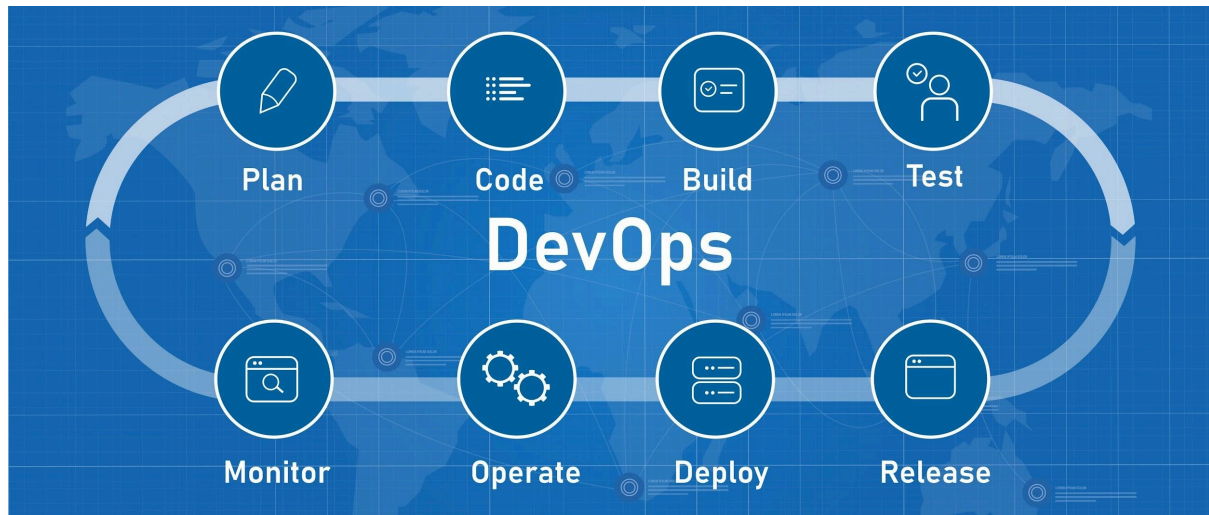
menu_data:
```

## 2.2 Azure Environment Setup

Below is the configuration used for the VM in azure:

- **VM Specs:** Azure B1s, Ubuntu 20.04 LTS
- **Prerequisites Installed:** Docker, Docker Compose, Git.
- **Port Configuration:** Ports were opened in Azure Network Security Group: 80, 443, 8081, 8082, 8083, 8084.

## 3. CI/CD Pipeline Implementation



### 3.1 Pipeline Workflow

The tools leveraged for CI/CD implementation were Github Actions + Azure VM with Linux as a host. The referenced workflow is mentioned in the [Readme.md](#) file located in the repository with the instructions to deploy the app and manage microservices.

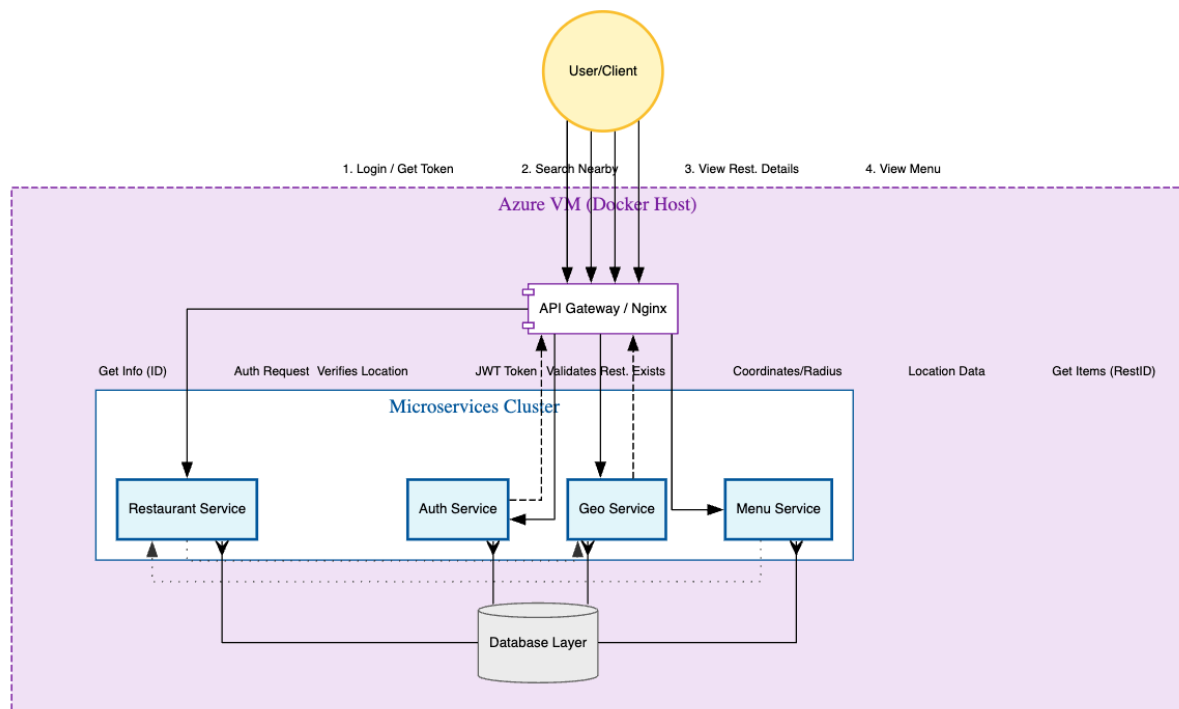
This is the basic workflow:

1. **Build:** GitHub Actions builds Docker images for all 4 services.
2. **Test:** Runs unit tests inside the containers.
3. **Push:** Pushes images to Docker Hub.
4. **Deploy:** SSH into Azure VM, pulls new images, and restarts containers.

## 4. MVP Delivery

### 4.1 Architecture Overview

Below is the high-level interaction diagram of the reservation system.



## 4.2 Functional Features

- **Authentication:** Users can register/login (Auth Service).

20:36

100

Crear cuenta

Por favor crea una cuenta para continuar

Nombre completo

Correo electronico

Nombre de usuario

Contraseña

Crear cuenta

Ya tienes cuenta?

Ingresar

20:35

100

Crear cuenta

Por favor crea una cuenta para continuar

Nombre completo

Correo electronico

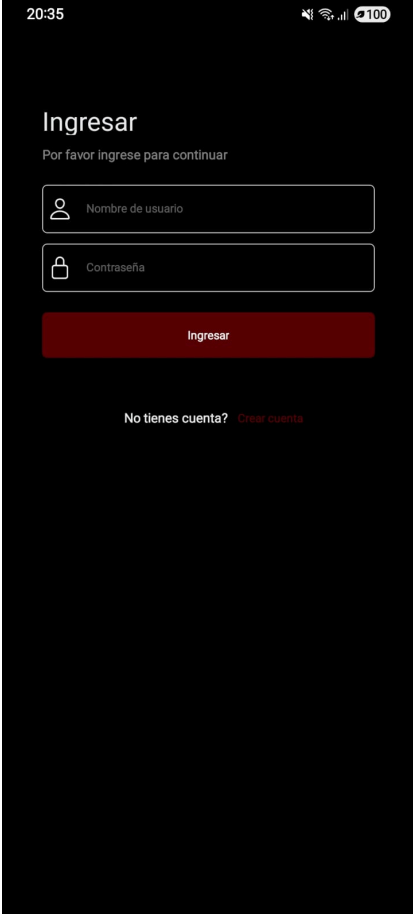
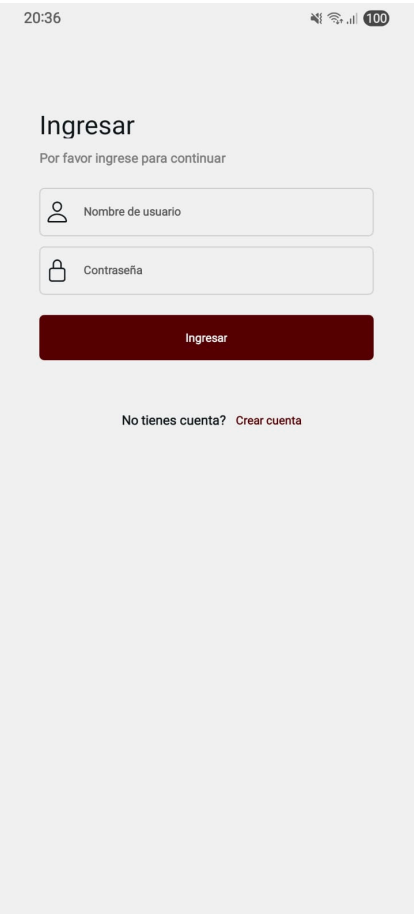
Nombre de usuario

Contraseña

Crear cuenta

Ya tienes cuenta?

Ingresar



[Ingresa a Click&Munch](#)


m@ejemplo.com

Contraseña

[O ingresa con](#)



By clicking continue, you agree to our [Terms of Service](#) and [Privacy Policy](#).



# Bienvenido

Regístrate en Click&Munch




**Nombre Completo**

**Email**

**Contraseña**

**Registrarse**

O ingresa con

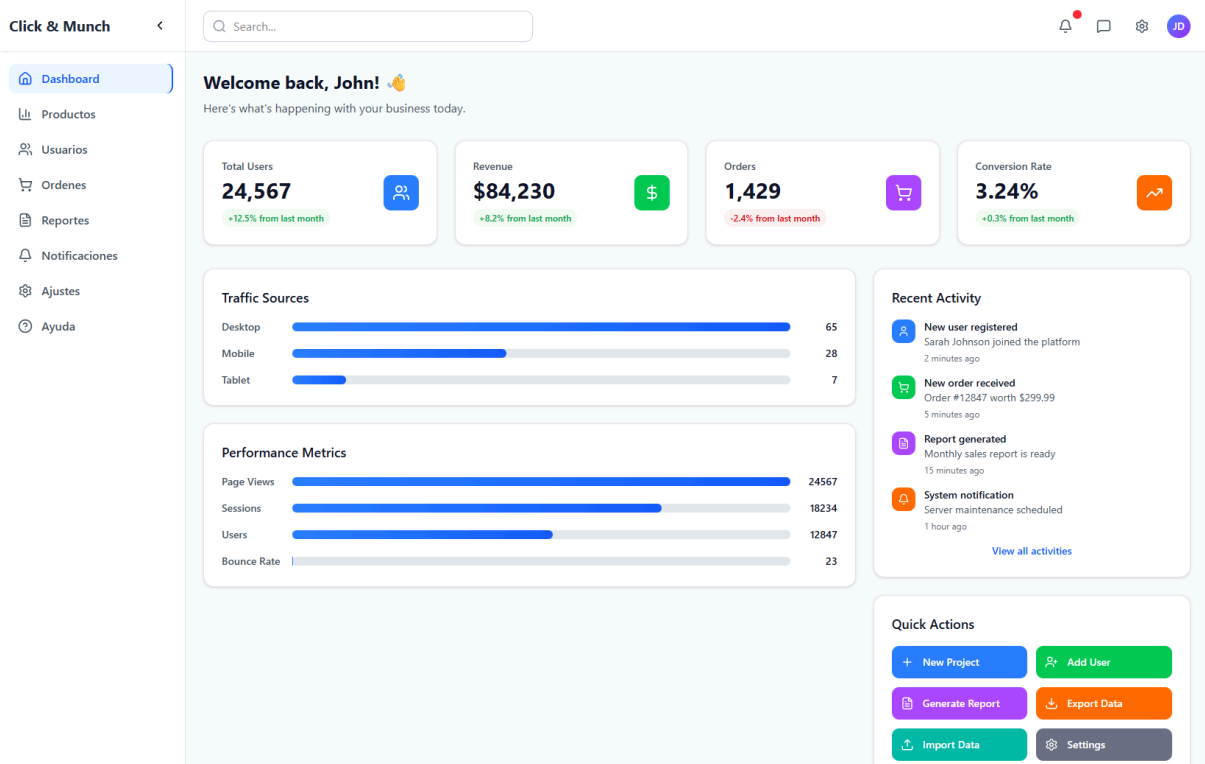


Ya tienes cuenta? [Ingresar](#)

By clicking continue, you agree to our [Terms of Service](#) and [Privacy Policy](#).

- **Discovery:** Users can find restaurants by location (Geo Service).
- **Browsing:** Users can view restaurant details (Restaurant Service).





Click & Munch

Search...

Dashboard

Productos

Usuarios

Ordenes

Reportes

Notificaciones

Ajustes

Ayuda

John Doe

john@company.com

ID	Imagen	Nombre	Precio	Inventario	Categoria	Acciones
001		Producto 1	\$250.00	100	Bebidas	Editar

A list of your recent invoices.

- **Selection:** Users can view specific menus for a restaurant (Menu Service).

# References

EngAndres. (n.d.). *unal\_public* [Folder: Software Engineering 2\_Morning (G3)/slides].  
GitHub.

[https://github.com/EngAndres/unal\\_public/tree/main/Software%20Engineering%202\\_Morning%20\(G3\)/slides](https://github.com/EngAndres/unal_public/tree/main/Software%20Engineering%202_Morning%20(G3)/slides)