Projeto de Bloco: Engenharia de Softwares Escaláveis [24E2_5]

ASSESSMENT

Professor: Francisco Benjamim

Aluno: Frederico Flores

KitchenSystem

https://github.com/nagualcode/kithcensystem

Conteúdo desta apresentação

- Motivação
- Objetivo do Sistema
- Motivação Técnica
- Tecnologias Utilizadas
- Arquitetura Geral do Sistema
- Escolhas de design
- Diagrama da Arquitetura
- Menssageria
- Entidades
- Orquestração
- Demonstração ao vivo
- Conclusão

1 Motivação

A eficiência operacional é um fator crítico no setor de restaurantes. Muitos estabelecimentos enfrentam dificuldades para sincronizar a produção da cozinha com os pedidos realizados pelos clientes. Essas dificuldades podem levar a:

- **Atrasos nos pedidos**: Impactando negativamente a experiência do cliente.
- **Erros na preparação dos pratos**: Devido à falta de comunicação eficaz entre a equipe de atendimento e a cozinha.
- **Desperdício de recursos**: Preparação de pratos incorretos ou em duplicidade.

2 Objetivo do Sistema

O **KitchenService** foi desenvolvido para solucionar esses desafios, fornecendo um serviço capaz de integrar menu, criação de pedidos, pagamento e comandos para a cozinha, com o objetivo de facilitar a adoção de:

- **Totens de Autoatendimento**: Permitindo que os clientes façam pedidos diretamente, reduzindo filas e tempo de espera.
- **Aplicativos de Delivery**: Integrando pedidos online diretamente com a cozinha, agilizando o processo de preparação.

3 Motivação Técnica

- **Desacoplamento**: A arquitetura de microsserviços permite que cada componente seja desenvolvido, implantado e escalado independentemente.
- **Escalabilidade**: Serviços críticos podem ser escalados conforme a demanda sem impactar o sistema como um todo.
- **Flexibilidade**: Facilita a integração com novos sistemas e tecnologias, como aplicativos de delivery ou novos métodos de pagamento.
- **Resiliência**: A falha de um serviço não compromete todo o sistema, aumentando a disponibilidade geral.

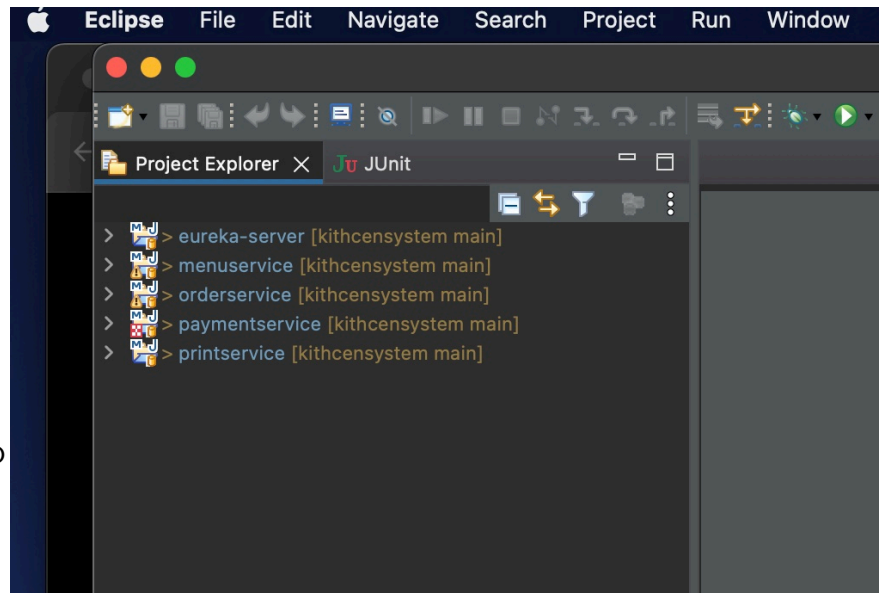
4 Tecnologias Utilizadas

- Spring Boot
- RabbitMQ
- PostgreSQL
- Docker
- Actuator
- Flyway
- Eureka
- Testcontainers
- Github
- React

5 Arquitetura Geral do Sistema

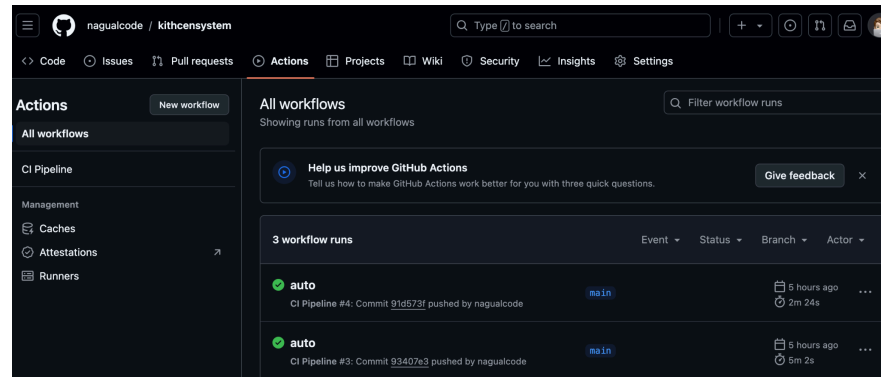
O **KitchenService** utiliza uma arquitetura de microsserviços, garantindo flexibilidade, escalabilidade e manutenção facilitada. Os principais componentes incluem:

- **OrderService**: Gerencia os pedidos dos clientes.
- **MenuService**: Administra o catálogo de pratos disponíveis.
- **PaymentService**: Processa pagamentos e atualiza o status dos pedidos.
- **PrintService**: Informa a cozinha dos pratos a serem preparados.

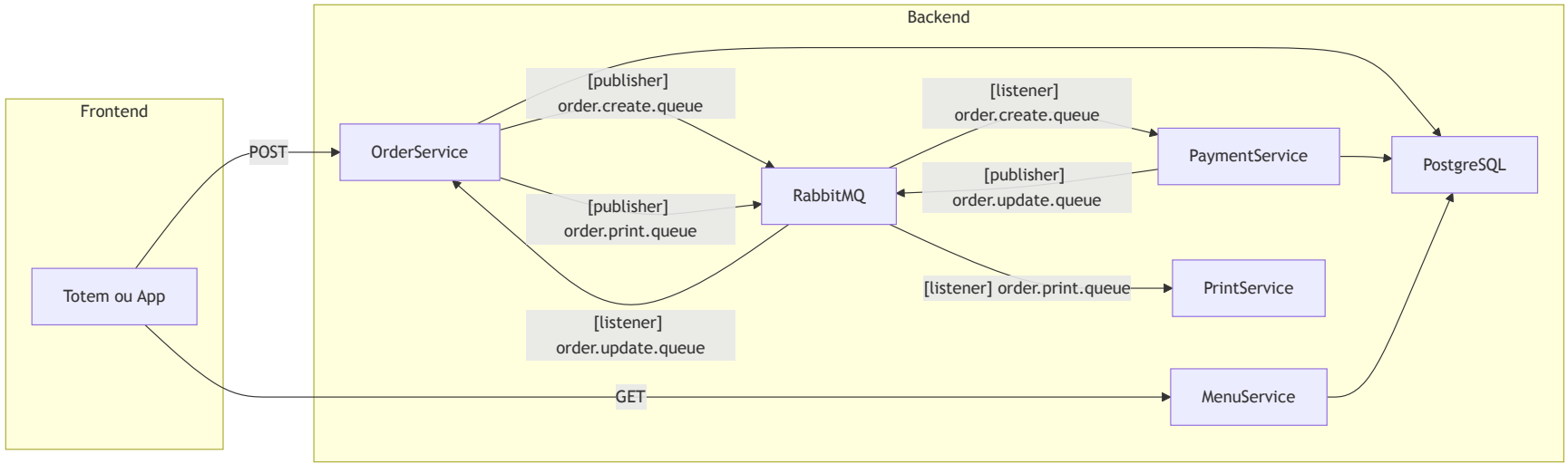


6 Escolhas de design

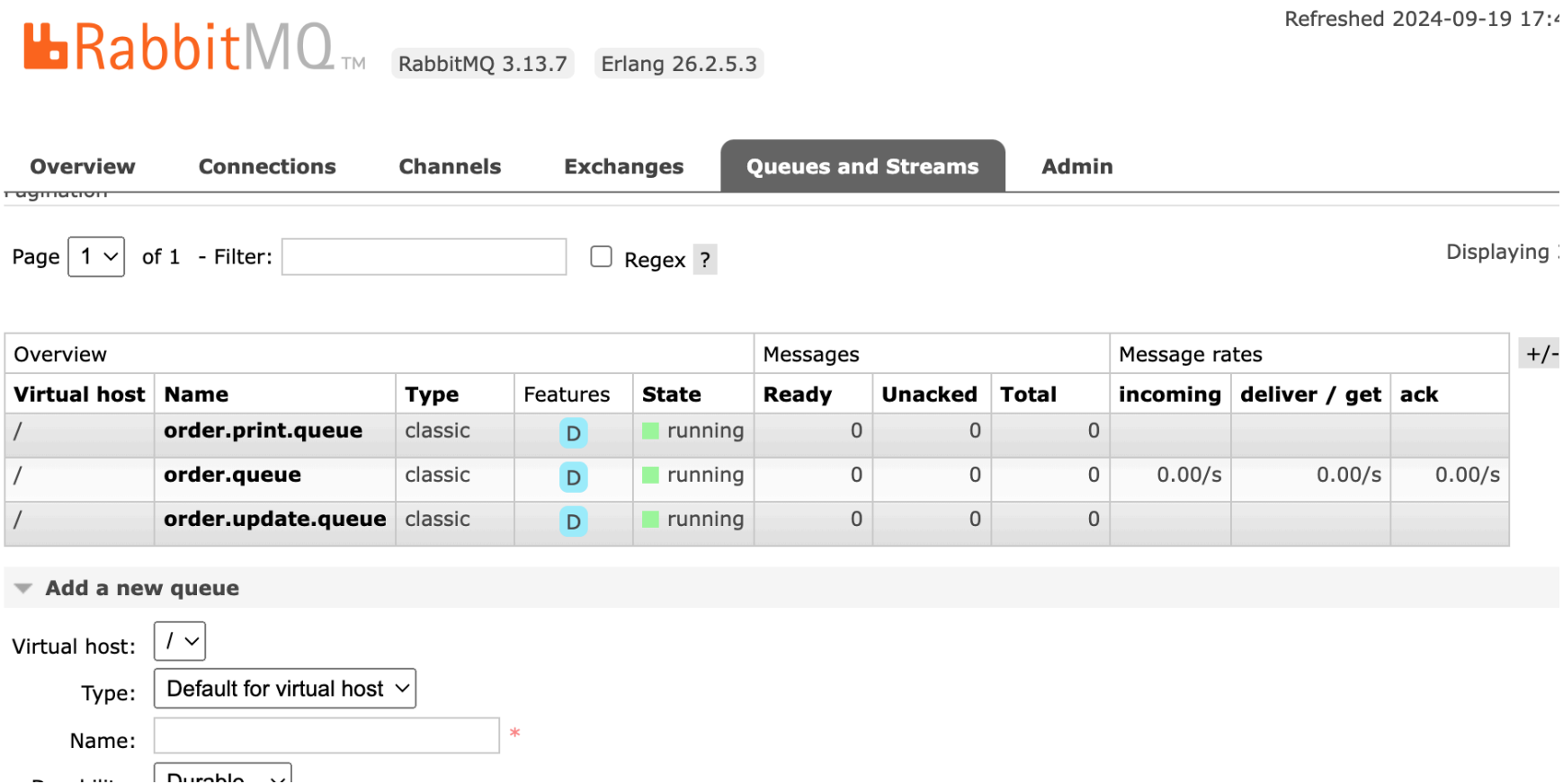
- **Separação de Responsabilidades:** Cada microsserviço cuida de um domínio específico, o que facilita a manutenção e escalabilidade.
- **Mensageria Assíncrona:** O uso de RabbitMQ elimina a necessidade de que os serviços esperem uns pelos outros, melhorando o desempenho e a tolerância a falhas.
- **Dockerização:** Cada serviço é executado em seu próprio container Docker, garantindo consistência nas implantações em diferentes ambientes.
- **Testabilidade:** JUnit tests em todos os controlers a services, realizados com testcointeiners, e mock, para garantir a independencia, e implementados no github Actions.

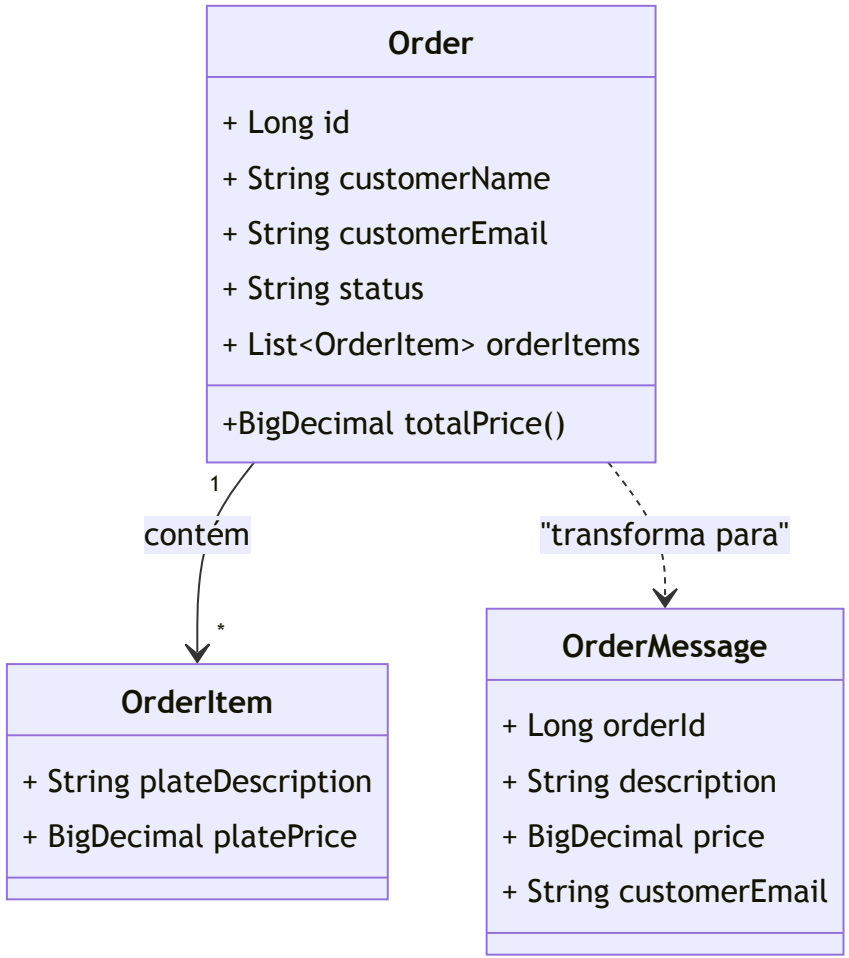


7 Diagrama da Arquitetura do Sistema



8 Queues


RabbitMQ™
RabbitMQ 3.13.7
Erlang 26.2.5.3
Refreshed 2024-09-19 17:
Overview
Connections
Channels
Exchanges
Queues and Streams
Admin
Page 1 of 1 - Filter:
Regex ?
Displaying
Overview
Messages
Message rates
+/-
Virtual host
Name
Type
Features
State
Ready
Unacked
Total
incoming
deliver / get
ack
/ order.print.queue classic D running 0 0 0
/ order.queue classic D running 0 0 0 0.00/s 0.00/s 0.00/s
/ order.update.queue classic D running 0 0 0
Add a new queue
Virtual host: /
Type: Default for virtual host
Name:



Entidades do Orderservice

- desacoplamento entre persistência e comunicação,
- controle de dados expostos
- possibilita evolução independente
- simplifica testes e debugging.

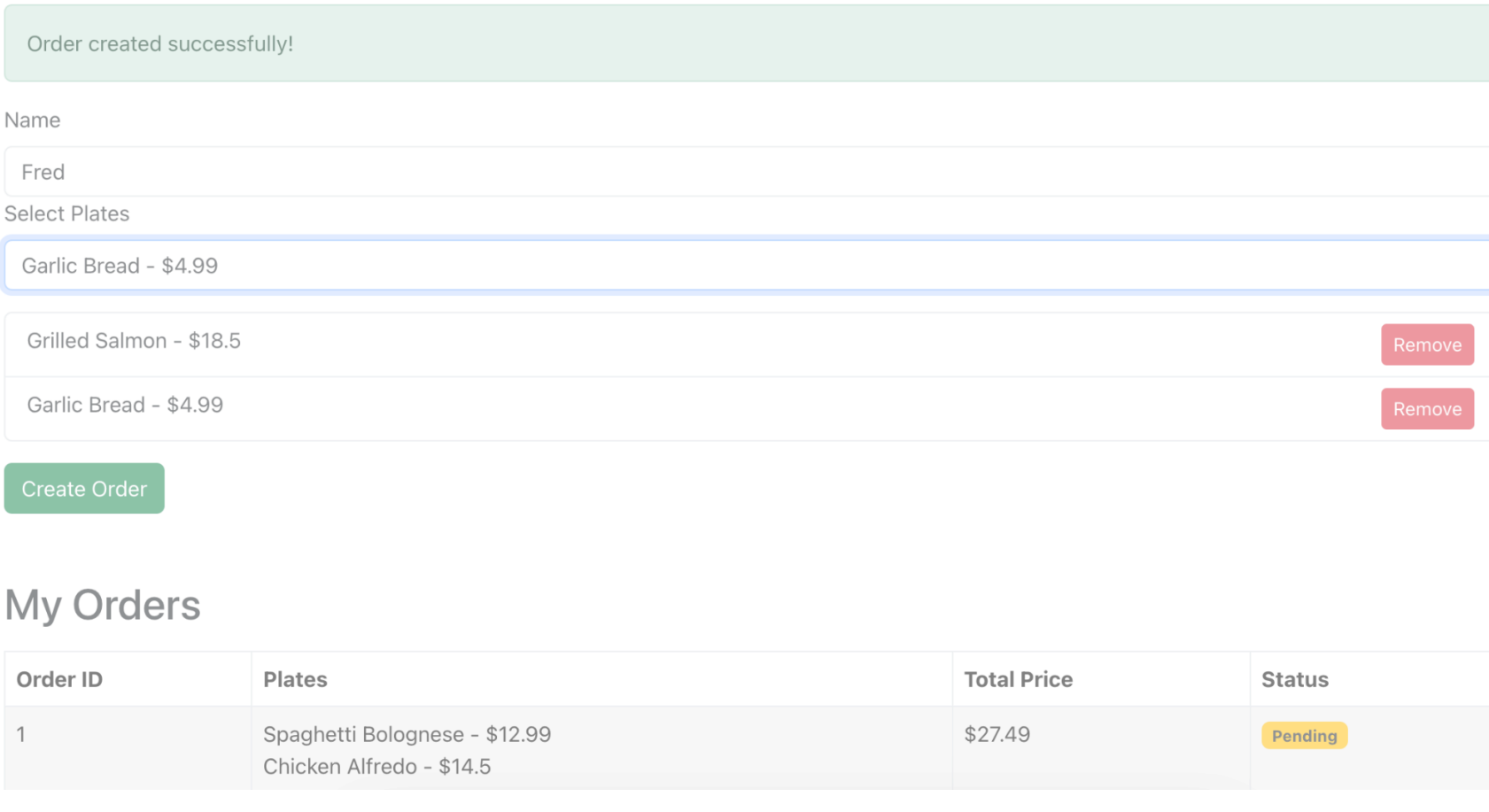
10 Orquestração

```
services:
  eureka-server:
    build: ./eureka-server
    ports:
      - "8761:8761"
    networks:
      - spring-boot-network

  flyway:
    image: flyway/flyway:9.22.0
    command: -url=jdbc:postgresql://postgres:5432/test_db -schemas=public -user=postgres -password=password -connectRet
    volumes:
      - ./V1__Create_All_Tables.sql:/flyway/sql/V1__Create_All_Tables.sql
    depends_on:
      postgres:
        condition: service_healthy
    networks:
      - spring-boot-network
```

```
paymentservice:
  build: ./paymentservice
  ports:
    - "8082:8082"
  environment:
    SPRING_PROFILES_ACTIVE: docker
    EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server:8761/eureka/
    SPRING_DATASOURCE_URL: jdbc:postgresql://postgres:5432/test_db
    SPRING_RABBITMQ_HOST: rabbitmq
  depends_on:
    - eureka-server
    - postgres
    - rabbitmq
    - flyway
  networks:
    - spring-boot-network
```

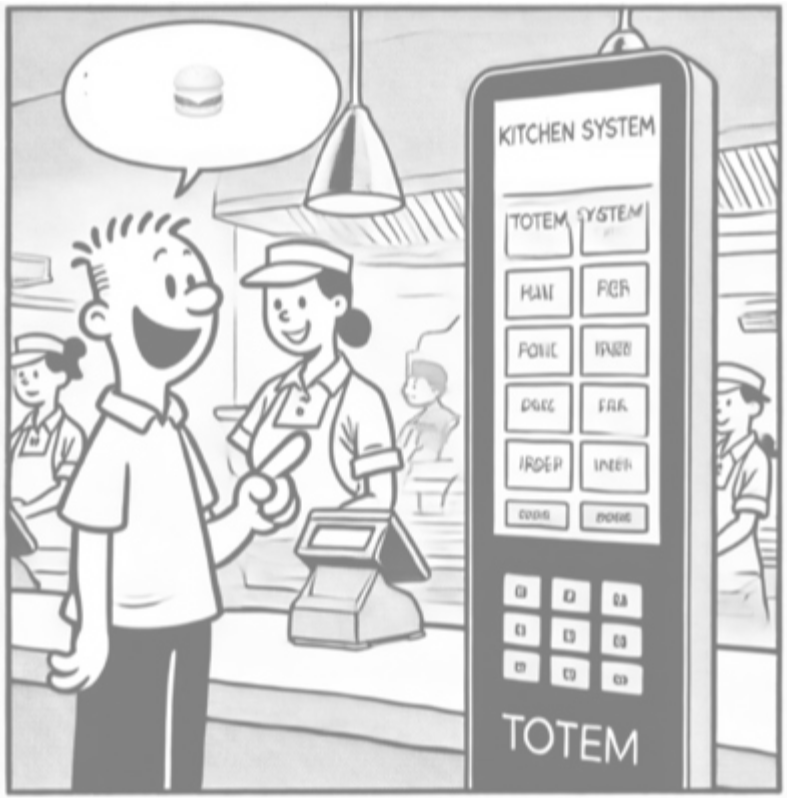
11 Demonstração



12 To Do’s

- Spring Security em Todos os Endpoints
- Segurança do RabbitMQ
- Controle de Estoque Integrado
- KitchenService Interativo

13 Obrigado!



frederico.silva@al.infnet.edu.br