INSTITUTO DE ENSEÑANZA SECUNDARIA CASTELAR

FAMILIA PROFESIONAL DE

TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES WEB

MEMORIA DEL MÓDULO PROFESIONAL DE PROYECTO

BARHUB

ALUMNO: Juan Parejo García

TUTOR: Javier Arostegui

CURSO ACADÉMICO: DAW

Memoria

1.- Análisis de contexto

Detección de necesidades

En el sector hostelero, especialmente en bares y cafeterías, existe una necesidad creciente de digitalizar la gestión de pedidos, reservas y clientes. Muchos establecimientos aún dependen de métodos tradicionales (papel, llamadas telefónicas), lo que dificulta la organización y puede provocar errores o pérdida de información. BarHub surge como solución a esta problemática, ofreciendo una plataforma web que centraliza y automatiza estos procesos.

Estudio de mercado y potenciales clientes

- Tipo de clientes o empresas donde se implantará el proyecto:

BarHub está dirigida a los siguientes tipos de empresas:

· Bares y cafeterías independientes que buscan mejorar la gestión de pedidos y reservas.

· Cadenas de hostelería que necesitan una solución escalable para varios establecimientos.

· Restaurantes pequeños que desean digitalizar sus procesos de atención al cliente.

- Utilidad o beneficio para los clientes:

Los clientes que adquieran BarHub podrán beneficiarse de:

· Automatización de pedidos y reservas: Reducción de errores y optimización del tiempo.

· Centralización de la información: Acceso fácil y rápido a datos de clientes y pedidos.

· Mejora de la experiencia del cliente: Mayor rapidez y profesionalidad en la atención.

· Análisis de datos: Informes sobre tendencias y preferencias de los clientes.

2.- Diseño del proyecto

Coste de implementación

El coste aproximado del estudio, programación e implantación del proyecto puede variar según el tamaño del establecimiento y las funcionalidades requeridas. Para una pequeña empresa, el coste estimado puede rondar los 1.000-3.000 €, incluyendo desarrollo, personalización y formación inicial.

Plataforma y requisitos técnicos

· Plataforma: Web (accesible desde cualquier navegador moderno)

· S.O. compatibles: cualquier dispositivo con navegador web(Windows, macOS, Linux…) .

· Requisitos mínimos: 2GB RAM, conexión a internet, un disco duro para guardar los tickets del pago de los pedidos.

- Utilidad o beneficio para los clientes:

Los clientes que adquieran BarHub podrán beneficiarse de:

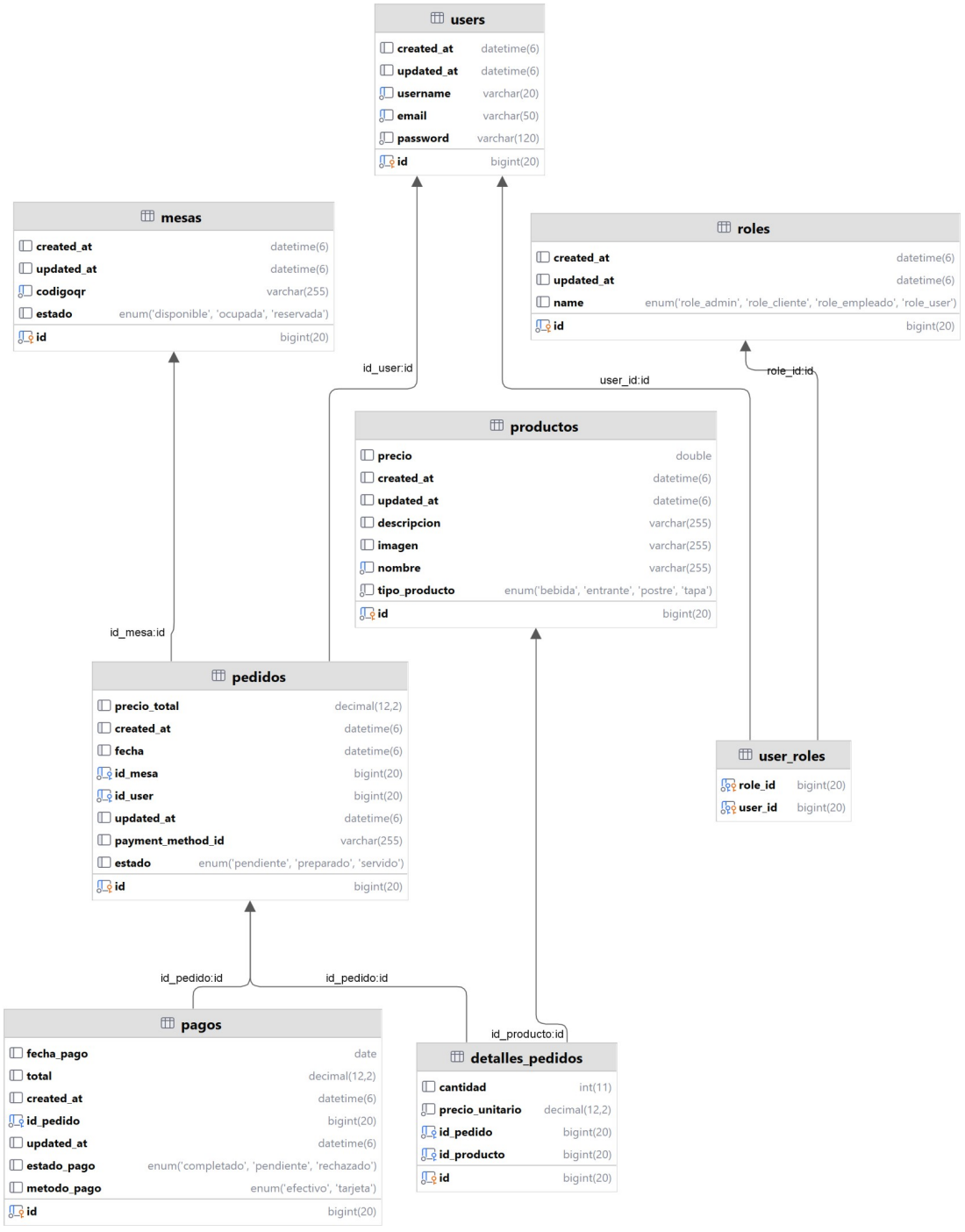
· Automatización de pedidos y reservas: Reducción de errores y optimización del tiempo.

· Centralización de la información: Acceso fácil y rápido a datos de clientes y pedidos.

Protección de datos

Se aplicará la normativa vigente de protección de datos (RGPD), garantizando la seguridad y confidencialidad de la información de los clientes. Se implementarán protocolos de cifrado y acceso seguro.

Esquema de la base de datos



3.- Organización de la ejecución

Planing de ejecución del proyecto Marzo-Junio 2025

- Semana 1-2:

Organización del seguimiento a través de aplicaciones como todoist y evernote.

Desarrollo del modelo Entidad-Relación de la base de datos.

Elección de paleta de colores.

Creación del backend en SpringBoot desde Intellij Idea.

Creación del frontend en angular desde VS Code.

Creación de las entidades(tablas) que posteriormente generará la estructura de la base de datos.

- Semana 3-4:

Implementación de spring-security (auth) en el backend.

Conexión entre angular y springboot.

Creación de métodos login y register en backend.

Creación de los CRUDS en backend.

- Semana 5-8:

Implementación stripe en backend y frontend.

Vistas de carta con descarga de pdf(ticket del pedido).

Páginas de mejoras, seguimiento del pedido, paginas de empleado…

- Semana 9-10:

Eliminé los datos de prueba de la base de datos y cree una carta real.

Añadí todos los estilos de la pagina.

- Semana 11:

Despliegue de la aplicación en la vps.

Documentación en README.md del despliegue.

- Semana 12:

Documentación de la memoria, manual de usuario y manual de desarrollador.

Recursos utilizados para el desarrollo

- VPS de Azure (para el despliegue de la aplicación)

- Una base de datos de mariadb

- Intellij Idea Ulitmate Edition (ide para el backend de Springboot)

- VS Code (ide para el frontend de angular)

- Github

Manual de Despliegue

Este es el manual de despliegue, donde se explica como ha sido desplegada la aplicación en la vps.

URL Página Principal: https://barhub.duckdns.org/

0.- Introducción

0.1.- Requisitos

Una VPS en Azure con Linux y Docker instalado.

Una red Docker ya creada llamada mi_red.

Un repositorio privado en Git; en este ejemplo se usará el repositorio llamado juan.

0.2.- Recursos

Dentro de la carpeta /documentacion/despliegue/recursos se encuentran todos los archivos necesarios para la práctica, incluyendo archivos .yml, archivos .wav y algunas imágenes que se referenciarán en el README.

La configuración de Caddy y los archivos de los contenedores también están en esta carpeta.

0.3.- Estructura de directorios y archivos en la VPS

.

```
├── backend
│   ├── BarHub-0.0.1-SNAPSHOT.jar
│   ├── Dockerfile
│   └── docker-compose.yml
├── caddy
│   ├── Caddyfile
│   └── docker-compose.yml
├── duckdns
│   └── docker-compose.yml
├── frontend
│   ├── BarHub-FrontendAngular
│   ├── Dockerfile
│   ├── apache-extra.conf
│   └── docker-compose.yml
├── get-docker.sh
├── kuma
│   └── docker-compose.yml
├── mysql
│   ├── docker-compose.yml
│   └── my.conf
└── swap.sh
```

1.- Creación de contenedores previos al despliegue de la aplicación

Antes de desplegar la aplicación principal, es necesario levantar varios contenedores, algunos esenciales y otros opcionales, que facilitan la infra-estructura y la monitorización.

1.1.- Contenedor de DuckDNS

Este contenedor gestiona la actualización dinámica del DNS para la URL pública.

mkdir duckdns

cd duckdns

nano .env

nano docker-compose.yml

En el archivo .env se debe colocar el token de DuckDNS y la IP pública que se asociará al dominio.

1.2- Contenedor de MariaDB

Este contenedor aloja la base de datos de la aplicación.

mkdir mariadb

cd mariadb

nano docker-compose.yml

1.3.- Contenedor de Uptime-Kuma

Contenedor opcional para monitorizar la disponibilidad de las URLs que desees.

mkdir kuma

cd kuma

nano docker-compose.yml

1.4.- Contenedor de Caddy

Caddy actuará como proxy inverso, conectando los contenedores internos con la IP pública y el dominio gestionado por DuckDNS.

mkdir caddy

cd caddy

nano docker-compose.yml

nano Caddyfile

2.- Despliegue de la aplicación

Crea un directorio app en la VPS donde se ubicará el docker-compose.yml principal de la aplicación.

2.1.- Despliegue del backend y de la base de datos

2.1.1.- Creación de Dockerfile

Crea el Dockerfile en la raíz del proyecto backend.

En la VPS, crea el directorio para el backend:

mkdir backend

2.1.2.- Despliegue del backend en la VPS

Genera el archivo .jar con Gradle:

.\gradlew clean build -x test

Ubica el .jar generado y transfiérelo a la VPS:

cd C:\Users\User\Documents\GitHub\BarHub\BarHub-BackendSpring\build\libs

scp -i "C:\Users\User\.ssh\id_rsa.pem" .\BarHub-0.0.1-SNAPSHOT.jar juan@20.199.89.174:~/backend/

En la VPS:

cd backend

Crea el Dockerfile y el docker-compose.yml, luego ejecuta:

docker compose up -d --build

2.2.- Despliegue del frontend

2.2.1.- Creación del Dockerfile

Crea el Dockerfile en la raíz del frontend.

En la carpeta app, crea el directorio para el frontend:

mkdir frontend

2.2.2.- Despliegue del frontend en la VPS

Compila el frontend Angular:

ng build --configuration production

Transfiere los archivos generados a la VPS:

scp -i "C:\Users\User\.ssh\id_rsa.pem" -r C:\Users\User\Documents\GitHub\BarHub\BarHub-FrontendAngular\dist\barhub juan@20.199.89.174:~/frontend/

2.3.- Despliegue final

Para finalizar, crea el Dockerfile y el docker-compose.yml en el directorio correspondiente y levanta los servicios:

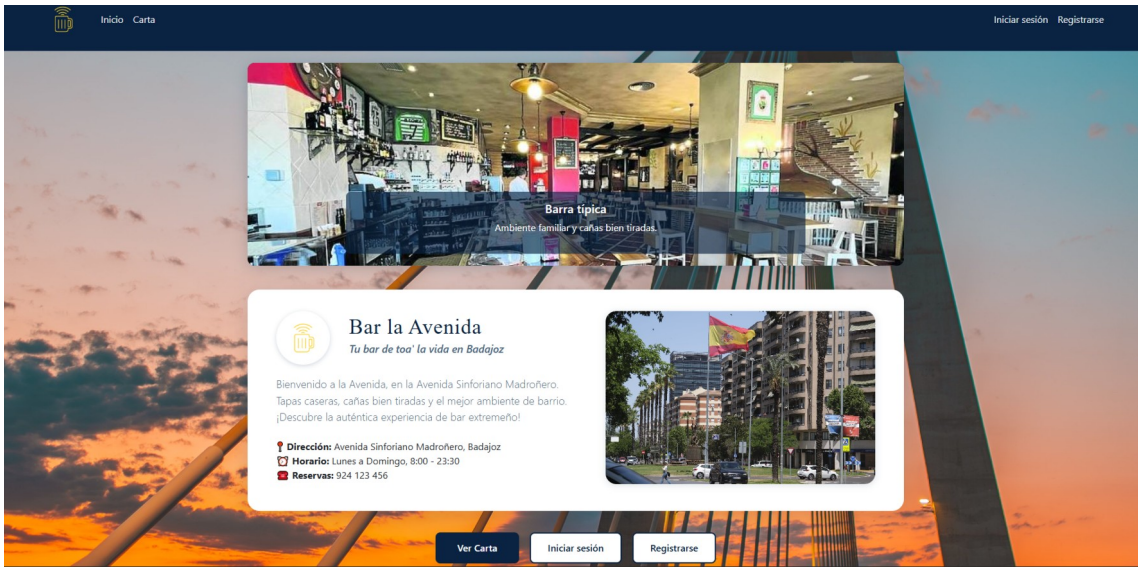
docker compose up -d --build

Manual de Usuario

Este es el manual de usuario, donde cualquier usuario que nunca se haya metido en la pagina web podrá saber acceder a ella y saber moverse por esta.

1.- Pagina Inicio

Para acceder a la página web deberá de dirigirse a cualquier navegador ya sea por el movil o por el ordenador, y insertar la siguiente URL:



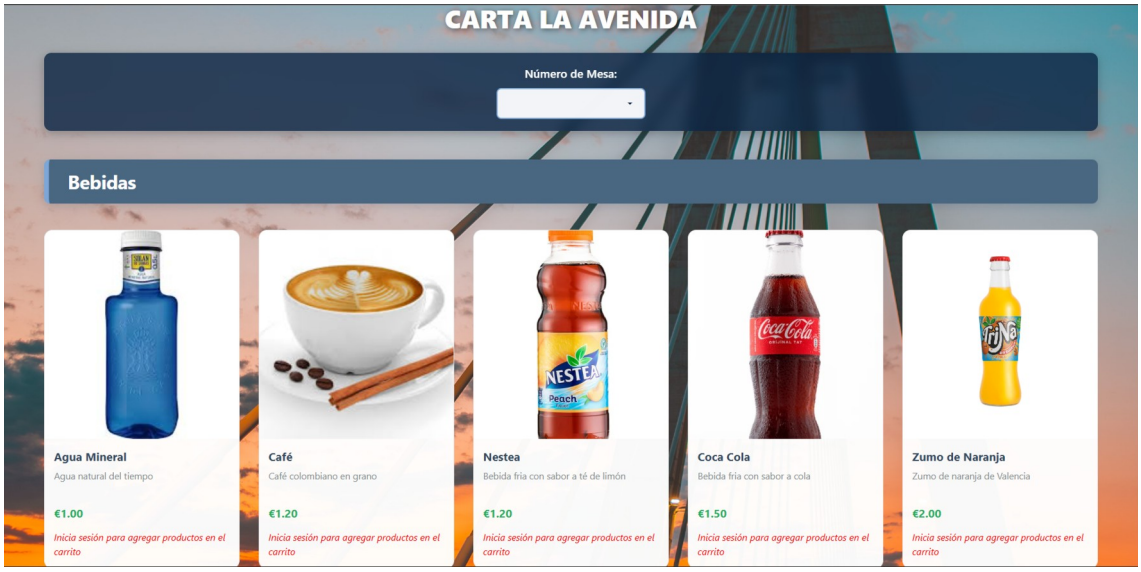
Una vez ya dentro de la web lo primero que ve el usuario es la pagina principal podrás encontrar un carrousel con varias fotos, la información del bar(Dirección, horario de apertura, num de teléfono para reservar…) y 3 botones, uno que te redirige a la carta para pedir pedidos, otro para iniciar sesión y otro para registrarse. Debajo de estos botones se encuentra un mapa donde indica la ubicación del bar.

Si no te has registrado o no has iniciado sesión, tu como usuario solo tendrás acceso a dos páginas, la de inicio y la de carta aunque no podrás hacer pedidos.

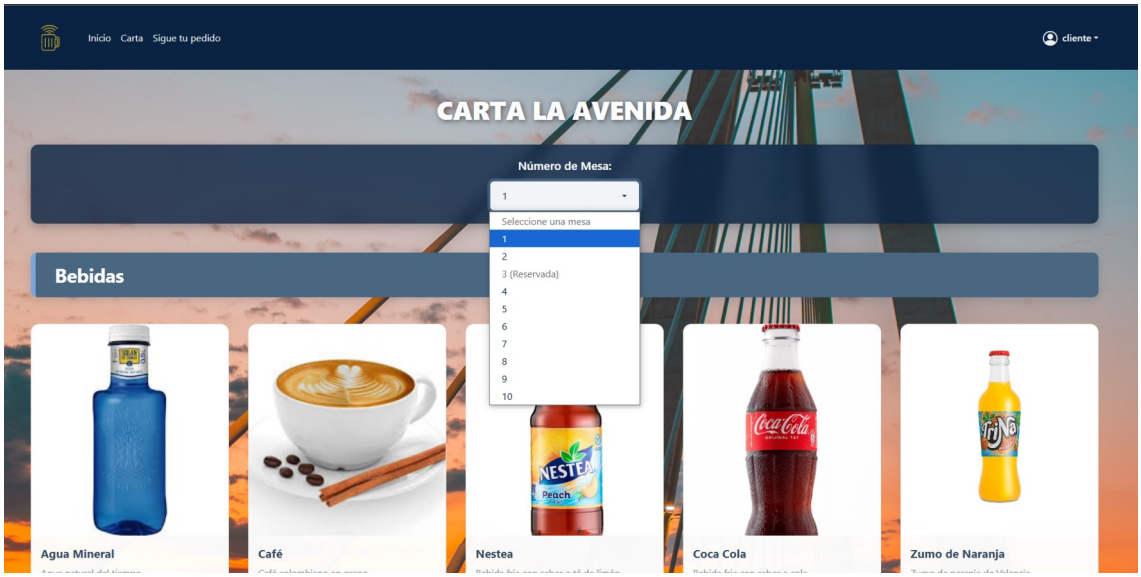
Lo que se le recomienda al usuario es que inicie sesión o se registre sino no podrá ver el resto de paginas ni interactuar con ellas.

2.- Pagina Carta

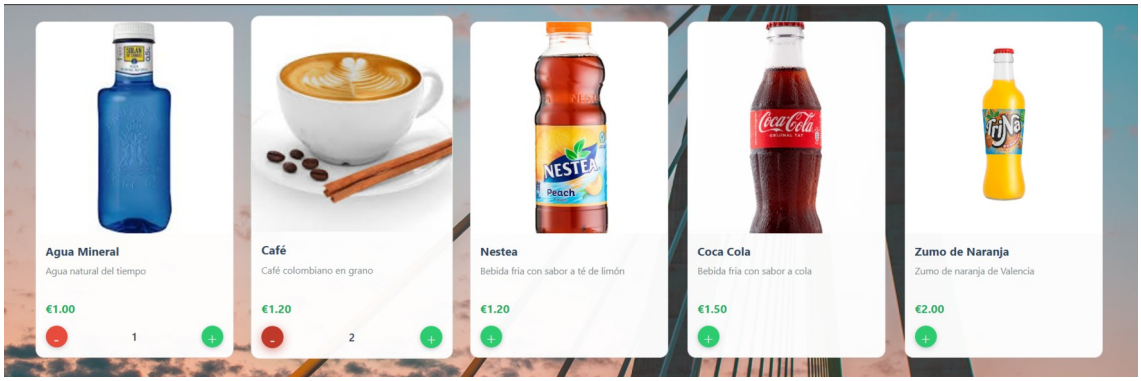
Esta pagina es para realizar pedidos, pero si no tienes la sesión iniciada no podrás realizar pedidos.



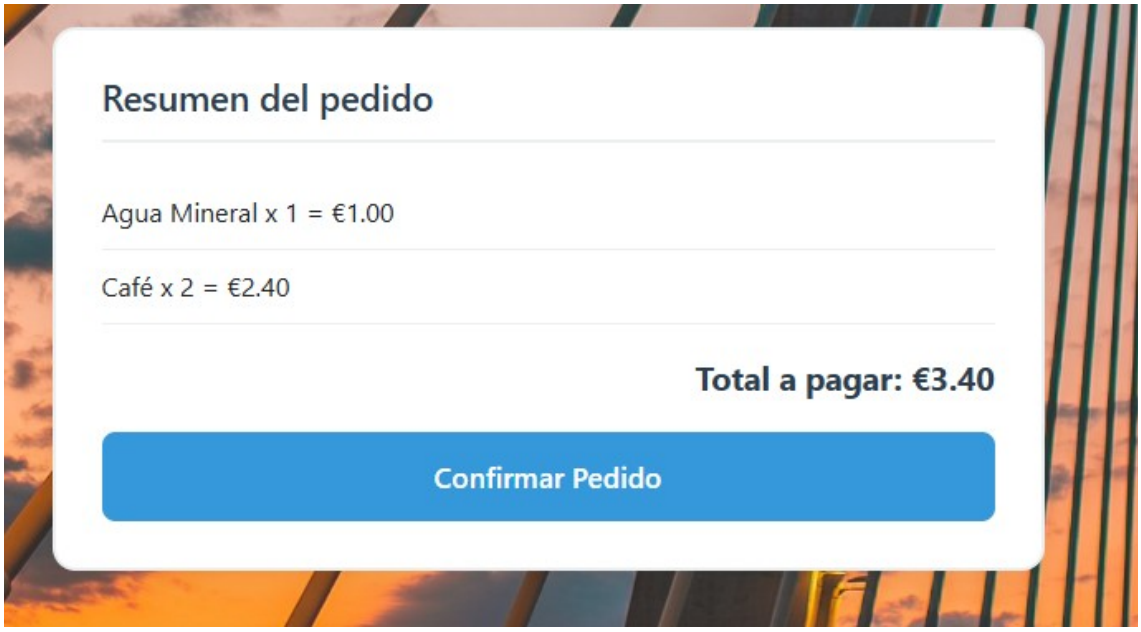
Con la sesión ya iniciada ya si podremos realizar pedidos, lo primero que haremos, será elegir la mesa en donde estamos, que fisicamente lo pondrá en la mesa.



Una vez tengas la mesa escogida puedes elegir los productos que quieras o eliminarlos, a gusto de consumidor.

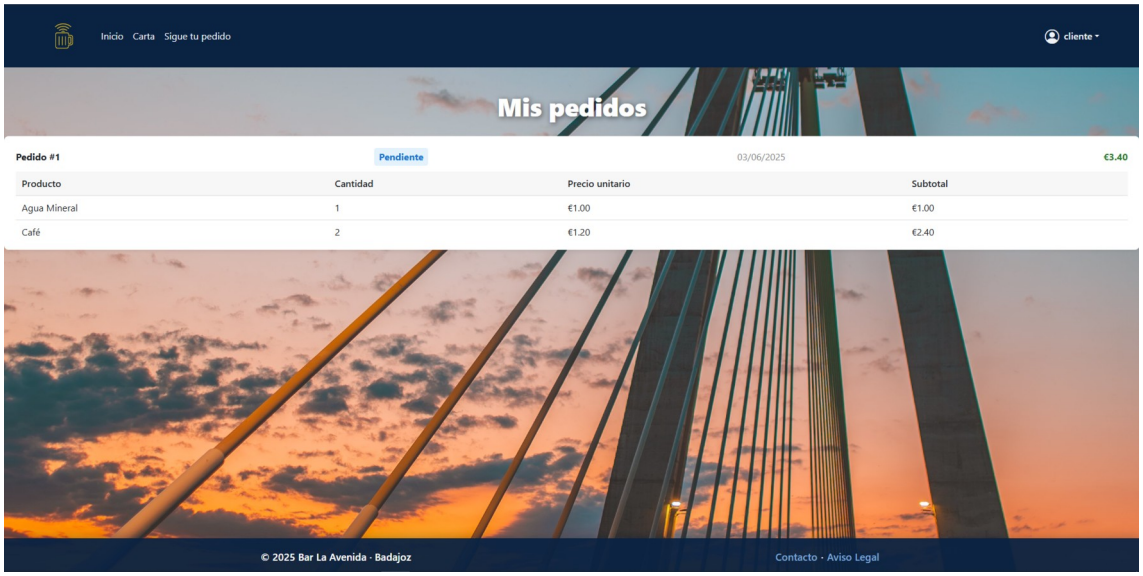


Y cuando el usuario haya elegido todos los productos que desea al final de la página aparecerá un resumen con los productos que ha pedido y el total a pagar.



Si le das al botón de confirmar pedido te redireccionará a una pagina para pagar el pedido. El usuario podrá pagar en efectivo o tarjeta.

Una vez ya finalizado el pedido, se le descargará al usuario un pdf con el ticket del pedido. A parte el usuario puede ir a través de la barra de navegación a la página de Sigue tu pedido, para ver el seguimiento del pedido, que lo irá actualizando los empleados a través de la página.



Manual de Empleado

Este es el manual que debería seguir un empleado para utilizar la página web y actualizar los datos de esta, y se vean reflejados para el cliente.

Los empleados pueden acceder a 3 páginas más que el cliente:

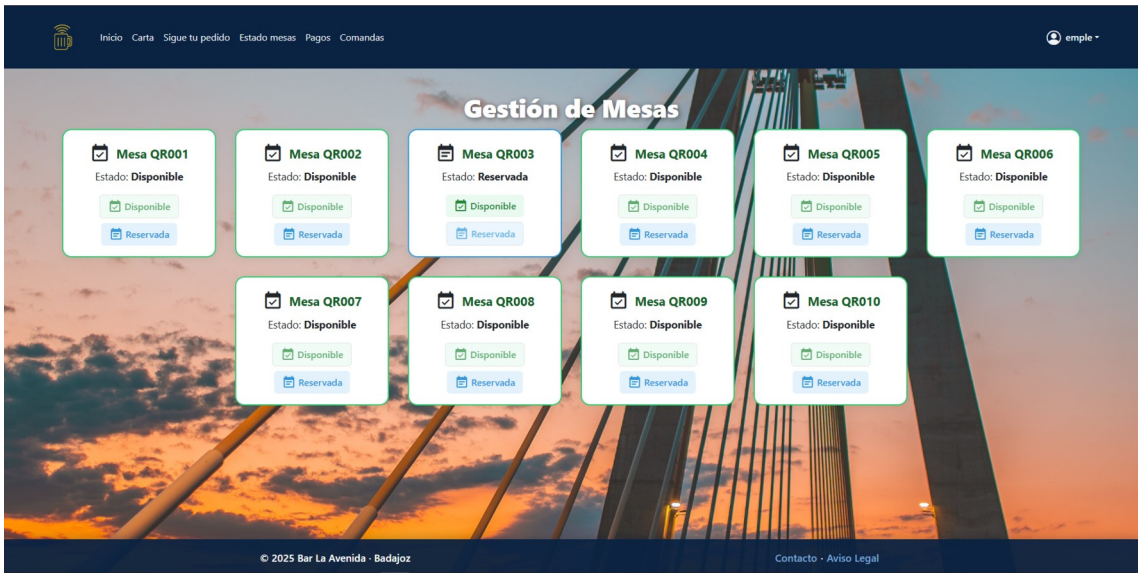
- Estado Mesas

- Pagos

- Comandas

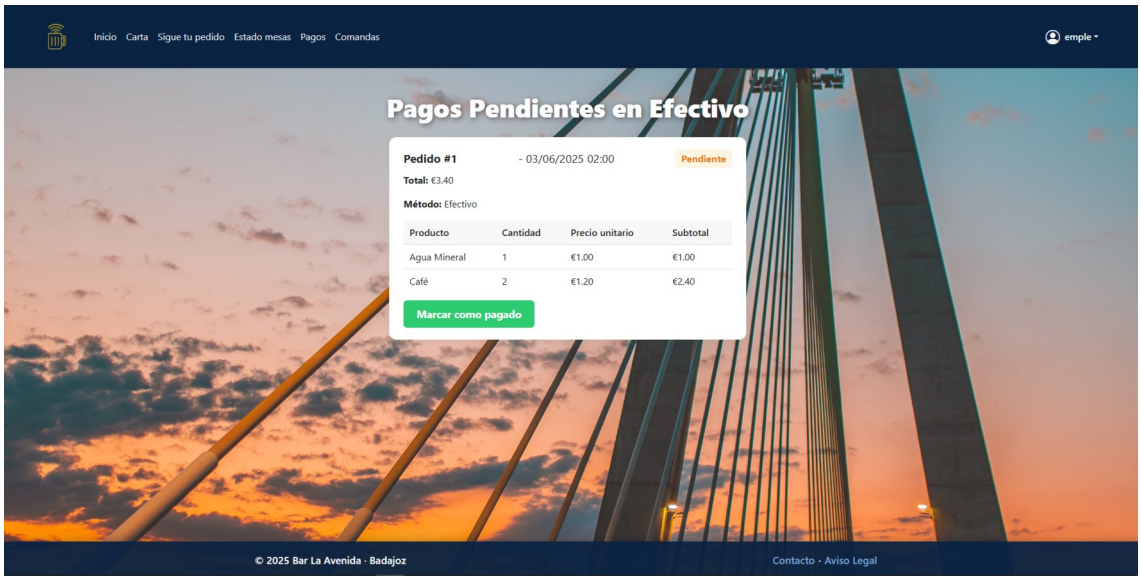
1.- Pagina Estado Mesas

Esta página sirve para que los empleados cambien el estado de las mesas a Disponible o Reservada. Esto es por si los clientes reservan mesa a través de llamada, que los empleados reserven la mesa y una vez esos clientes se sienten ya se pondría la mesa libre para realizar pedidos.



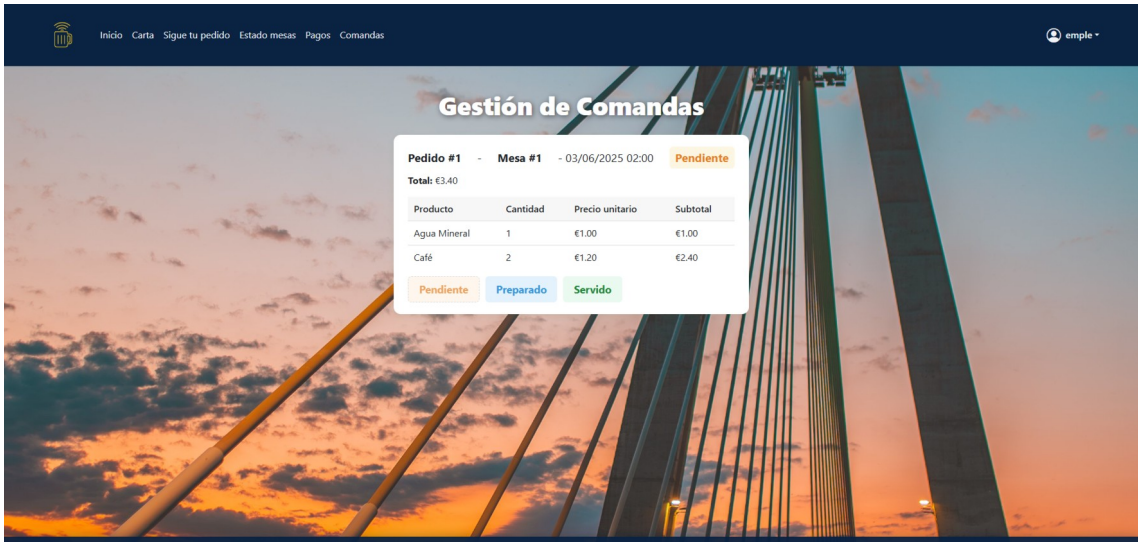
2.- Pagina Pagos

Esta página sirve para que los empleados cambien el estado del pago del pedido del cliente que se haya hecho en efectivo. El funcionamiento sería simple el cliente se acerca a la barra a pagar o el camarero se acerca a la mesa a cobrarles, y una vez ya se haya realizado el pago, el empleado manualmente selecciona que el pedido ha sido pagado.



3.- Pagina Comanda

Esta página ayuda a los cocineros/camareros a la hora de preparar un plato o de llevar un pedido a la mesa y cambiarán el estado del pedido de si ya está preparado o de si ya se ha servido a la mesa, manualmente. El estado del pedido le aparecerá al cliente en sus respectivos pedidos.



Manual de Programador

BarHub es una aplicación web integral para la gestión de pedidos y reservas en establecimientos hosteleros. Desarrollada con Spring Boot (backend) y Angular (frontend), este manual proporciona una guía detallada para entender, configurar y extender el sistema.

1. Arquitectura del Sistema

1.1 Stack Tecnológico

· Backend:

- Java 17 + Spring Boot 3.2: Base del servidor, con soporte para inyección de dependencias y configuración modular.

- Spring Security + JWT: Autenticación basada en tokens JWT (JSON Web Tokens) para seguridad de endpoints.

- MariaDB: Sistema gestor de bases de datos relacional, alojado en Azure.

- Stripe API: Integración para procesamiento de pagos en línea.

· Frontend:

- Angular 14: Framework para construcción de interfaces dinámicas con TypeScript.

- PrimeNG: Biblioteca de componentes UI para Angular (tablas, formularios, modales).

- PDFMake: Generación de tickets en formato PDF directamente desde el navegador.

· Infraestructura:

- VPS Azure: Despliegue de la aplicación en máquinas virtuales de Azure.

- GitHub Actions: CI/CD para integración continua y despliegues automatizados.

2. Configuración del Entorno

2.1 Requisitos

- JDK 17+: Para compilar y ejecutar el backend.

- Node.js 18+ y npm 9+: Instalación de dependencias del frontend.

- Docker 24+: Empaquetado y despliegue en contenedores.

- IDEs Recomendados:

- IntelliJ IDEA (backend)

- VS Code (frontend) con extensiones Angular.

2.2 Clonación del Repositorio

git clone https://github.com/PapiAlca/BarHub.git

cd BarHub

3. Estructura del Proyecto

3.1 Backend (Spring Boot)

src/main/java/

├── auth/ # Configuraciones de SpringSecurity

├── config/ # Configuraciones de Spring (CORS, seguridad)

├── controllers/ # REST endpoints (PedidoController, UsuarioController)

├── dtos/ # Objetos de transferencia (ProductoDto, PedidoDto)

├── entities/ # Entidades JPA (Usuario, Pedido, Mesa)

├── repositories/ # Interfaces Spring Data JPA

└── services/ # Lógica de negocio (PedidoService, UsuarioService)

Ejemplo de Entidad JPA:

```
package com.daw2.barhub.model.entity;


import com.daw2.barhub.model.Enum.EstadoMesa;
import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;


import java.time.Instant;


@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
@Data
@Entity
@Table(name="mesas")
public class Mesa {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;


    @Column(unique=true, nullable=false)
    private String codigoqr;


    @Column
    @Enumerated(EnumType.STRING)
    private EstadoMesa estado;


    @CreationTimestamp
    private Instant createdAt;


    @UpdateTimestamp
    private Instant updatedAt;
}
```

3.2 Frontend (Angular)

src/app/

├── admin/ # Módulo de administración(Página admin CRUDS backend)

├── auth/ # Módulo de autenticación (login, registro)

├── public/ # Vistas principales (carta, seguimiento-pedido)

├── shared/ # Componentes reutilizables (toast, footer)

└── views/ # Vistas principales (carta, seguimiento-pedido)

Ejemplo de Servicio Angular:

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { JwtHelperService } from '@auth0/angular-jwt';
import { BehaviorSubject } from 'rxjs';
import { Router } from '@angular/router';

@Injectable({
      providedIn: 'root'
      })
export class AuthService {
   private apiUrl = 'http://localhost:8080/api/auth';
   private roles: string[] = [];
   private rolesSubject = new BehaviorSubject<string[]>(this.getRoles());
   private isAuthenticatedSubject = new BehaviorSubject<boolean>(this.isAuthenticated());

   roles$ = this.rolesSubject.asObservable();
   isAuthenticated$ = this.isAuthenticatedSubject.asObservable();

   private tokenKey = 'token';
   private jwtHelper = new JwtHelperService();

   constructor(private http: HttpClient, private router: Router) {}

   login(credentials: { username: string; password: string }): Observable<{ token: string }> {
      this.logout();
      return new Observable(observer => {
            this.http.post<{ token: string }>(`${this.apiUrl}/login`, credentials).subscribe({
            next: (res) => {
      if (!res.token) {
         observer.error('El servidor no devolvió un token');
         return;
      }
```

```
    localStorage.setItem(this.tokenKey, res.token);

    try {
      const decodedToken = this.jwtHelper.decodeToken(res.token);
      const roles = decodedToken.roles || []; // 👈 Extrae roles del token

      // Guarda roles en localStorage
      localStorage.setItem('roles', JSON.stringify(roles));

      localStorage.setItem('user', JSON.stringify({
            id: decodedToken.id,
            username: decodedToken.sub
      }));

      // Actualiza los subjects
      this.rolesSubject.next(roles);
      this.isAuthenticatedSubject.next(true);

      this.router.navigate(['/carta']);
      observer.next(res);
    } catch (error) {
      localStorage.removeItem(this.tokenKey);
      observer.error('Token inválido: ' + error);
    }
    observer.complete();
    },
    error: (err) => observer.error(err)
  });
});
}

register(data: { username: string; email: string; password: string }): Observable<any> {
    return this.http.post(`${this.apiUrl}/register`, data);
}

logout(): void {
    localStorage.clear();
    this.rolesSubject.next([]);
    this.isAuthenticatedSubject.next(false);
    this.router.navigate(['/']);
}

isAuthenticated(): boolean {
    return !!localStorage.getItem('token');
}

verifyEmail(token: string): Observable<any> {
    return this.http.get(`${this.apiUrl}/verify?token=${token}`);
}

setRoles(roles: string[]) {
```

```
    this.roles = roles;
  }

  getRoles(): string[] {
  const roles = localStorage.getItem('roles');
    return roles ? JSON.parse(roles) : [];
  }

  hasRole(requiredRoles: string[]): boolean {
  const userRoles = this.getRoles();
    return requiredRoles.some(role => userRoles.includes(role));
  }



  getToken(): string | null {
    return localStorage.getItem(this.tokenKey);
  }

  setToken(token: string): void {
    localStorage.setItem(this.tokenKey, token);
  }

  isLoggedIn(): boolean {
    return localStorage.getItem('user') !== null;
  }

  getCurrentUser(): any {
  const user = localStorage.getItem('user');
    return user ? JSON.parse(user) : null;
  }

  getNombreUsuario(): string {
  const user = JSON.parse(localStorage.getItem('user') || '{}');
    return user.nombre || user.username || '';
  }
}
```

4. Base de Datos y Modelo de Datos

4.1 Esquema Principal

```
-- Tabla de usuarios
CREATE TABLE users (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  created_at DATETIME(6),
  updated_at DATETIME(6),
  username VARCHAR(20) NOT NULL,
  email VARCHAR(50) UNIQUE NOT NULL,
  password VARCHAR(120) NOT NULL
);

-- Tabla de roles
CREATE TABLE roles (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  created_at DATETIME(6),
  updated_at DATETIME(6),
  name ENUM('role_admin', 'role_cliente', 'role_empleado', 'role_user')
NOT NULL
);

-- Tabla intermedia para relación muchos a muchos entre usuarios y roles
CREATE TABLE user_roles (
  role_id BIGINT NOT NULL,
```

```
  user_id BIGINT NOT NULL,
  PRIMARY KEY (role_id, user_id),
  FOREIGN KEY (role_id) REFERENCES roles(id),
  FOREIGN KEY (user_id) REFERENCES users(id)
);

-- Tabla de mesas
CREATE TABLE mesas (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  created_at DATETIME(6),
  updated_at DATETIME(6),
  codigoqr VARCHAR(25) UNIQUE,
  estado ENUM('disponible', 'ocupada', 'reservada') NOT NULL
);

-- Tabla de productos
CREATE TABLE productos (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  precio DOUBLE NOT NULL,
  created_at DATETIME(6),
  updated_at DATETIME(6),
  descripcion VARCHAR(255),
  imagen VARCHAR(255),
  nombre VARCHAR(255) NOT NULL,
```

```
  tipo_producto ENUM('bebida', 'entrante', 'postre', 'tapa') NOT NULL
);

-- Tabla de pedidos
CREATE TABLE pedidos (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  precio_total DECIMAL(12,2) NOT NULL,
  created_at DATETIME(6),
  fecha DATETIME(6),
  id_mesa BIGINT NOT NULL,
  id_user BIGINT NOT NULL,
  updated_at DATETIME(6),
  payment_method_id VARCHAR(255),
  estado ENUM('pendiente', 'preparado', 'servido') NOT NULL,
  FOREIGN KEY (id_mesa) REFERENCES mesas(id),
  FOREIGN KEY (id_user) REFERENCES users(id)
);

-- Tabla de detalles de pedidos
CREATE TABLE detalles_pedidos (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  cantidad INT(11) NOT NULL,
  precio_unitario DECIMAL(12,2) NOT NULL,
  id_pedido BIGINT NOT NULL,
```

```
  id_producto BIGINT NOT NULL,
  FOREIGN KEY (id_pedido) REFERENCES pedidos(id),
  FOREIGN KEY (id_producto) REFERENCES productos(id)
);

-- Tabla de pagos
CREATE TABLE pagos (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  fecha_pago DATE,
  total DECIMAL(12,2) NOT NULL,
  created_at DATETIME(6),
  id_pedido BIGINT NOT NULL,
  updated_at DATETIME(6),
  estado_pago ENUM('completado', 'pendiente', 'rechazado') NOT NULL,
  metodo_pago ENUM('efectivo', 'tarjeta') NOT NULL,
  FOREIGN KEY (id_pedido) REFERENCES pedidos(id)
);
```

4.2 Relaciones Clave

- *Relaciones Muchos a Muchos:*

User-Role: N:N (un usuario puede tener múltiples roles y un rol puede ser asignado a múltiples usuarios, gestionado a través de la tabla user_roles).

- *Relaciones Uno a Muchos:*

User-Pedido: 1:N (un usuario puede realizar múltiples pedidos).

Mesa-Pedido: 1:N (una mesa puede tener múltiples pedidos a lo largo del tiempo).

Pedido-DetallePedido: 1:N (un pedido puede contener múltiples productos).

Producto-DetallePedido: 1:N (un producto puede aparecer en múltiples detalles de pedidos).

Pedido-Pago: 1:N (un pedido puede tener múltiples intentos de pago o pagos parciales).

-Campos de Auditoría:

Todas las entidades principales incluyen campos created_at y updated_at para tracking de cambios y auditoría temporal.

- Campos Enumerados:

roles.name: role_admin, role_cliente, role_empleado, role_user

mesas.estado: disponible, ocupada, reservada

productos.tipo_producto: bebida, entrante, postre, tapa

pedidos.estado: pendiente, preparado, servido

pagos.estado_pago: completado, pendiente, rechazado

pagos.metodo_pago: efectivo, tarjeta

5. Seguridad y Autenticación

5.1 Configuración de Spring Security

```
@Configuration
@EnableWebSecurity
public class SecurityConfig {


   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http
            .cors(cors -> cors.configurationSource(corsConfigurationSource()))
            .csrf(AbstractHttpConfigurer::disable)
```

```
            .authorizeHttpRequests(auth -> auth
                .requestMatchers(HttpMethod.OPTIONS, "/**").permitAll()
                .requestMatchers(
                    "/api/auth/**",
                    "/usuarios/**",
                    "/roles/**",
                    "/detalles_pedidos/**",
                    "/detalles_pedidos/pedido/**",
                    "/mesas/**",
                    "/pagos/**",
                    "/pedidos/**",
                    "/productos/**",
                    "/seguimientos/**"
                ).permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            );
        return http.build();
    }
}
```

5.2 Flujo JWT

Cliente envía credenciales a /api/auth/login.

Servidor valida y devuelve un JWT firmado.

Cliente incluye el JWT en el header Authorization de solicitudes subsiguientes.

6. Integración con Stripe

6.1 Service de Stripe

```
@Service
public class StripeService {

    @Value("${stripe.secret-key}")
    private String secretKey;

    @PostConstruct
    public void init() {
        Stripe.apiKey = secretKey;
```

```
    }

    public PaymentIntent createPaymentIntent(Long amount, String currency, String paymentMethodId)
            throws StripeException {

        PaymentIntentCreateParams params = PaymentIntentCreateParams.builder()
                .setAmount(amount)
                .setCurrency(currency)
                .setPaymentMethod(paymentMethodId)
                .setConfirm(true)
                .build();

        return PaymentIntent.create(params);
    }
}
```

7. Estilo de Código y Buenas Prácticas

7.1 Backend (Java)

- Google Java Style Guide:

- Indentación de 4 espacios.

- Nombres de clases en PascalCase, métodos en camelCase.

- Documentación Javadoc en clases públicas.

7.2 Frontend (TypeScript)

- Prefixes para componentes (app-producto-card).

- Uso de interfaces fuertemente tipadas.

- Inyección de dependencias mediante constructor.

Bibliografia

1.- Spring Boot 3.2 - Migración y documentación

OpenRewrite Docs. (2025). Migrate to Spring Boot 3.2. Recuperado de: https://docs.openrewrite.org/recipes/java/spring/boot3/upgradespringboot_3_2

Spring.io. (2023). Spring Boot Reference Documentation. Recuperado de: https://spring.io/projects/spring-boot

2.- Angular v14 - Novedades y guías oficiales

Angular Blog. (2023). Angular v16 is here! Recuperado de: https://blog.angular.dev/angular-v16-is-here-4d7a28ec680d

Angular.io. (2023). Angular Documentation. Recuperado de: https://angular.io/docs

3.- MariaDB - Documentación y administración

MariaDB Foundation. (2025). MariaDB Server Documentation. Recuperado de: https://mariadb.com/kb/en/documentation/

4.- Stripe API - Referencia y guía de integración

Stripe Documentation. (2025). API Reference. Recuperado de: https://docs.stripe.com/api

5.- Despliegue en Microsoft Azure

DeployHQ. (2016). Deploying to Microsoft Azure VM. Recuperado de: https://www.deployhq.com/guides/microsoft-azure

Microsoft Azure. (2023). Azure Virtual Machines documentation. Recuperado de: https://docs.microsoft.com/en-us/azure/virtual-machines/

6.- Control de versiones y CI/CD

GitHub. (2023). GitHub Actions Documentation. Recuperado de: https://docs.github.com/en/actions