DESCRIPCIÓN DEL PROYECTO POSVENTA360

Posventa360 es una plataforma web corporativa diseñada para transformar por completo la gestión de solicitudes, incidencias y requerimientos en el segmento PyME y empresarial. En el entorno moderno de las telecomunicaciones, las empresas dependen de conexiones estables, soporte constante y atención inmediata para garantizar la continuidad de sus operaciones. Sin embargo, los métodos tradicionales utilizados en la mayor parte de áreas de posventa —como correos electrónicos, llamadas, mensajería instantánea y hojas de Excel— dificultan el seguimiento estructurado, la trazabilidad y la correcta priorización de los casos.

La propuesta Posventa360 centraliza todos estos procesos en una sola plataforma robusta, accesible y completamente digitalizada. El proyecto surge como respuesta a la necesidad real de mejorar la gestión interna, reducir los tiempos de respuesta y ofrecer una experiencia más eficiente tanto para los colaboradores de posventa como para los clientes corporativos. La plataforma permitirá registrar, gestionar, monitorear y cerrar tickets con información detallada, indicadores clave, evidencia visual y un sistema de roles para asegurar el orden y la seguridad de la información.

Uno de los principales problemas actuales del área de posventa radica en la falta de un sistema centralizado que permita controlar el flujo completo de cada caso. Esto genera retrasos, pérdidas de información, dificultades al momento de realizar auditorías internas y una incapacidad para medir desempeños o identificar patrones de fallas. Posventa360 elimina estas limitaciones al implementar un mecanismo estructurado en donde cada ticket sigue un ciclo definido, con estados visibles, responsables asignados, tiempos registrados y notificaciones oportunas.

La plataforma también integrará un expediente digital por cliente, permitiendo almacenar el historial completo de todas sus incidencias, solicitudes técnicas, visitas técnicas, migraciones, reportes de velocidad y evidencias. Esto facilitará la gestión del cliente

corporativo, permitirá tomar decisiones basadas en datos y fortalecerá la relación entre la empresa y Claro, mejorando sustancialmente la percepción de servicio y calidad.

Además, Posventa360 incorporará herramientas de análisis, como dashboards de KPIs y reportes automáticos que mostrarán métricas como tiempos de respuesta, número de casos resueltos, casuísticas más frecuentes, desempeño por asesor y carga de trabajo por área. Estas métricas son fundamentales para una operación moderna y permitirán a los supervisores y gerencia tomar decisiones estratégicas de forma informada.

El diseño de la plataforma está basado en una arquitectura tecnológica moderna que garantiza escalabilidad, seguridad y rendimiento. Se empleará React en el frontend para asegurar una interfaz rápida y responsiva; Node.js con Express como backend para la gestión de lógica empresarial; y PostgreSQL como base de datos relacional para garantizar integridad y disponibilidad de información. Esta arquitectura permite que la solución crezca en el futuro, integrándose con herramientas como Power BI, correo corporativo e incluso sistemas internos utilizados por la compañía.

Posventa360 no solo busca optimizar procesos, sino también fortalecer la cultura de trazabilidad, profesionalismo y excelencia en el servicio dentro del área de posventa. A través de su implementación, los colaboradores contarán con una herramienta clara y eficiente que facilitará su trabajo diario, proporcionará orden y dará visibilidad del estado real de la operación, evitando confusiones y duplicidades que hoy representan un problema frecuente.

Finalmente, este proyecto se concibe como un sistema innovador capaz de adaptarse a diferentes niveles operativos y con potencial para convertirse en un pilar tecnológico dentro de la atención corporativa. Con una experiencia intuitiva, una estructura sólida y un enfoque centrado en la productividad, Posventa360 se posiciona como una solución esencial para modernizar el servicio posventa en el sector PyME y corporativo.

La arquitectura tecnológica propuesta para **Posventa360** busca garantizar **rendimiento, disponibilidad, seguridad y escalabilidad**, considerando que la plataforma será utilizada por colaboradores corporativos y manejará información sensible de clientes PyME y empresariales. a continuación se justifica el stack seleccionado, comparándolo con otras alternativas disponibles.

1. Frontend: React + Vite + TailwindCSS

Stack Elegido

React.js como librería principal.

Vite como bundler para un entorno de desarrollo rápido.

TailwindCSS para estilos modulares y consistentes.

Justificación

Experiencia de usuario fluida: React permite construir interfaces altamente interactivas, esenciales para flujos de tickets, cargas de archivos, dashboards y formularios dinámicos.

Reutilización de componentes: Perfecto para vistas repetitivas como tarjetas de tickets, paneles, historiales y formularios de registro.

Ecosistema robusto: Compatible con librerías como React Query para consumo de API, Zustand para estado global y librerías para notificaciones y modales.

Vite: Compilación rapidísima y hot reload eficiente.

Tailwind: Permite crear una identidad visual corporativa clara sin escribir CSS repetitivo.

Comparación con Alternativas

Alternativa Ventajas Desventajas

Angular Arquitectura completa, ideal para proyectos grandes del gobierno/empresas
Curva de aprendizaje alta, más rígido, desarrollo más lento

Vue.js Ligero, fácil de aprender Menor adopción corporativa en Latinoamérica, menos librerías maduras

Svelte Excelente rendimiento Comunidad más pequeña, difícil escalabilidad a largo plazo

Conclusión: React proporciona velocidad, modularidad y escalabilidad, ideal para una plataforma corporativa que crecerá con el tiempo.

2. Backend: Node.js con NestJS

Stack Elegido

Node.js como entorno de ejecución.

NestJS como framework estructurado.

Justificación

Arquitectura modular clara: NestJS aporta capas bien definidas (controladores, servicios, módulos), ideal para separar roles, tickets, reportes, auditorías, notificaciones, etc.

Alto rendimiento en tiempo real: Node.js soporta flujos asincrónicos, perfecto para actualizaciones de tickets, estados, logs o sockets.

Integraciones sencillas: APIs REST y WebSockets integrados nativamente.

Seguridad corporativa: NestJS facilita la implementación de JWT, RBAC y middlewares de seguridad.

Comparación con Alternativas

Alternativa Ventajas Desventajas

Django (Python) Excelente para desarrollo rápido, ORM poderoso Menor rendimiento en tiempo real, menos eficiente para WebSockets

Laravel (PHP) Muy usado en Latinoamérica, curva de aprendizaje baja Menor rendimiento bajo cargas altas comparado con Node.js

Spring Boot (Java) Ideal para sistemas bancarios/enterprise críticos Muy pesado para equipos pequeños, tiempos de desarrollo más largos

Conclusión: Node.js + NestJS ofrece un backend rápido, moderno, escalable y adecuado para requerimientos corporativos de Claro.

3. Base de Datos: PostgreSQL

Stack Elegido

PostgreSQL como base de datos relacional.

ORM: Prisma o TypeORM.

Justificación Modelo relacional robusto: Perfecto para estructurar tickets, usuarios, roles, clientes, logs, historial de cambios, archivos adjuntos, etc.

Integridad transaccional: Asegura que la información no se pierda ni corrompa, indispensable en sistemas corporativos.

JSONB híbrido: Permite almacenar metadatos o información flexible sin perder la estructura clásica.

Escalabilidad vertical y horizontal: Bases grandes, consultas complejas, reportes, dashboards e índices avanzados.

Comparación con Alternativas

Alternativa Ventajas Desventajas

MySQL/MariaDB Muy común, rápido en consultas simples Menos sólido en integridad y relaciones complejas MongoDB (NoSQL) Flexible, veloz, esquemas libres No ideal para reportes, KPIs, procesos transaccionales y auditoría

SQL Server Solución corporativa completa Costoso, licenciamiento, infraestructura más compleja

Conclusión: PostgreSQL es la elección ideal por su robustez, potencia, seguridad, costo cero y excelente integración con APIs corporativas.

4. Infraestructura Recomendada:

Servidor en la nube (AWS / Azure): Contenedores Docker + CI/CD.

Autenticación: JWT + Refresh Tokens + Roles corporativos.

Almacenamiento de evidencia: AWS S3 o Azure Blob Storage.

Monitorización: Grafana + Prometheus.

Stack Final Seleccionado para Posventa360

Capa Tecnología Justificación

Frontend React + Vite + Tailwind Rápido, moderno, escalable, interfaz dinámica

Backend Node.js + NestJS Arquitectura modular, alta concurrencia, APIs seguras

Base de Datos PostgreSQL Integridad, relaciones claras, rendimiento para reportes

Infraestructura Docker + AWS/Azure Despliegue escalable y professional.

Posventa360 requiere almacenar información de tickets, usuarios, clientes corporativos, roles, adjuntos y trazabilidad de cambios.

A continuación, se presenta el **ERD principal**:

DISEÑO TÉCNICO – POSVENTA360

1. Esquema de Base de Datos (ERD)

Entidades principales:

- USERS(id, nombre, email, password_hash, rol_id, creado_en)

- ROLES(id, nombre, descripcion)

- CLIENTES(id, nombre_empresa, nit, telefono, direccion, creado_en)

- TICKETS(id, titulo, descripcion, estado, prioridad, usuario_id, cliente_id, creado_en)

- HISTORIAL(id, ticket_id, usuario_id, accion, comentario, fecha)

- ADJUNTOS(id, ticket_id, url, tipo_archivo, subido_por, fecha_subida)

Relaciones:

- Un usuario crea muchos tickets.

- Un cliente tiene muchos tickets.

- Un ticket posee muchos registros de historial.

- Un ticket puede tener múltiples adjuntos.

2. Diseño de API (REST)

Endpoint 1: Crear Ticket

POST /tickets

Request:

{

"titulo": "Falla en enlace dedicado",

"descripcion": "Cliente reporta intermitencia",

"prioridad": "alta",

"cliente_id": 12,

"usuario_id": 5

}

Response:

{

"message": "Ticket creado exitosamente",

"ticket_id": 1043,

"estado": "abierto"

}

Endpoint 2: Actualizar estado de Ticket

PATCH /tickets/{id}/estado

Request:

{

"nuevo_estado": "en_proceso",

"usuario_id": 5,

"comentario": "Escalado a soporte nivel 2"

}

Response:

{

"message": "Estado actualizado",

"estado_actual": "en_proceso"

}

Endpoint 3: Obtener Tickets por Cliente

GET /clientes/{id}/tickets

Response:

{

"cliente_id": 12,

"empresa": "Grupo Industrial Morales",

"tickets": [

{"id":1043,"titulo":"Falla","estado":"en_proceso","prioridad":"alta"}

]

}

Endpoint 4: Subir archivo adjunto

POST /tickets/{id}/adjuntos

Endpoint 5: Historial de un Ticket

GET /tickets/{id}/historial

Endpoint 6: Autenticación

POST /auth/login

GERENCIA DEL PROYECTO – POSVENTA360

1. Estimación de Horas de Desarrollo

A) Análisis y Requerimientos: 30 h

B) Diseño UI/UX: 36 h

C) Desarrollo Frontend: 106 h

D) Desarrollo Backend: 114 h

E) QA Testing y Correcciones: 40 h

F) Despliegue y Documentación: 24 h

Total de horas estimadas: 350 horas

2. Costos por Rol

Tarifas:

- Desarrollador Full Stack: Q160/h

- Diseñador UI/UX: Q140/h

- QA Tester: Q100/h

- Project Manager: Q200/h

Distribución:

- Full Stack (220h): Q35,200

- UI/UX (36h): Q5,040

- QA (30h): Q3,000

- PM (64h): Q12,800

Costo total del proyecto: Q56,040

3. Porcentajes por Área

Desarrollador Full Stack: 63%

Project Manager: 23%

UI/UX: 9%

QA Testing: 5%

4. Justificación del Presupuesto

El presupuesto contempla desarrollo completo del frontend y backend, seguridad JWT,

gestión avanzada de tickets y adjuntos, dashboards, pruebas QA, documentación técnica

y despliegue en nube. Se considera un costo competitivo para una plataforma corporativa.

El Problema

En el entorno corporativo de telecomunicaciones, especialmente en el segmento PyME y empresarial, la gestión de incidencias y solicitudes postventa se realiza de forma dispersa y poco estructurada.
Actualmente, muchas áreas dependen de:

- Correos electrónicos
- Conversaciones en WhatsApp
- Llamadas telefónicas

- Hojas de Excel independientes
- Registro manual en documentos no centralizados

Esto genera múltiples problemas:

- **Falta de trazabilidad real** del ciclo de vida de un caso
- **Información duplicada o perdida**
- **Retrasos** en la atención por ausencia de un sistema de priorización
- **Dificultad para medir KPIs**, tiempos de respuesta y desempeño por asesor
- **Incapacidad para auditar**, monitorear o escalar casos de forma controlada
- **Historial del cliente fragmentado** o inexistente

La falta de una plataforma centralizada afecta directamente la productividad interna y la experiencia del cliente empresarial.

La Solución

Posventa360 es una plataforma web corporativa diseñada para **centralizar, digitalizar y automatizar** la gestión de tickets postventa del segmento PyME y empresarial.

Su funcionalidad principal incluye:

- **Registro estructurado de tickets** (fallas, solicitudes, migraciones, cambios técnicos)
- **Gestión de estados** (abierto, en proceso, cerrado)
- **Asignación de responsables**
- **Adjuntos y evidencia fotográfica/documental**

- **Historial completo y trazabilidad por caso**
- **Expediente del cliente corporativo**
- **Dashboard de KPIs** (tiempos de respuesta, carga por asesor, casos frecuentes)
- **Control de roles y acceso seguro**
- **Notificaciones internas**

En lugar de múltiples herramientas aisladas, Posventa360 integra todo el flujo operativo en **una sola plataforma uniforme y accesible desde cualquier dispositivo**.

Justificación

La solución es innovadora y necesaria por varias razones:

✓ Centraliza procesos actualmente fragmentados

Unifica correo, chat, llamadas, hojas de Excel y documentos en una sola interfaz profesional.

✓ Aumenta la productividad

Provee estructura y orden, reduce tiempos muertos y agiliza la asignación y resolución de casos.

✓ Mejora la experiencia del cliente empresarial

Con atención más rápida, oportuna y con evidencias documentadas.

✓ Genera trazabilidad completa

Permite auditorías internas, seguimiento histórico y control sobre cada acción realizada.

✓ Ofrece datos accionables

Los dashboards permiten identificar patrones de fallas, medir desempeño y tomar decisiones gerenciales basadas en datos.

✓ Se adapta al crecimiento

Su arquitectura escalable permite integrar nuevos módulos: reportes avanzados, inteligencia artificial, integración con WhatsApp Business, Power BI, CRM, etc.

✓ Soluciona un problema real en áreas corporativas

El manejo de casos postventa es una necesidad crítica en telecomunicaciones y muchas empresas aún carecen de un sistema oficial para gestionarlo.

README.md

```
# Posventa360 – Proyecto Full Stack (Demo académico)

Este repositorio contiene una implementación DEMO del proyecto Posventa360:

- `/backend`: API REST con NestJS (in-memory, sin BD).

- `/frontend`: Interfaz React + Vite para login y gestión básica de tickets.

## Cómo ejecutar

### 1) Backend

```bash

cd backend

npm install

npm run start:dev

```
```

API: `http://localhost:3000/api/v1`

2) Frontend

```bash
cd frontend
npm install
npm run dev
```

App: `http://localhost:5173`

backend/package.json

```
{
"name": "posventa360-backend",
"version": "1.0.0",
"description": "API REST Posventa360 (NestJS demo)",
"main": "dist/main.js",
"scripts": {
"start": "nest start",
"start:dev": "nest start --watch",
"build": "nest build"
},
"dependencies": {
"@nestjs/common": "^10.0.0",
```

```
"@nestjs/core": "^10.0.0",
"@nestjs/platform-express": "^10.0.0",
"reflect-metadata": "^0.1.13",
"rxjs": "^7.0.0"
},
"devDependencies": {
"@nestjs/cli": "^10.0.0",
"@nestjs/schematics": "^10.0.0",
"@nestjs/testing": "^10.0.0",
"typescript": "^5.0.0",
"@types/node": "^20.0.0"
}
}
```

backend/tsconfig.json

```
{
"compilerOptions": {
"module": "commonjs",
"declaration": false,
"removeComments": true,
"emitDecoratorMetadata": true,
"experimentalDecorators": true,
```

"allowSyntheticDefaultImports": true,

"target": "ES2017",

"sourceMap": true,

"outDir": "./dist",

"baseUrl": "./",

"incremental": true

}

}

backend/README.md

Posventa360 – Backend (NestJS demo)

API REST de ejemplo para el proyecto Posventa360.

Requisitos

- Node.js 18+

- Nest CLI (`npm i -g @nestjs/cli`)

Instalación

```bash

cd backend

npm install

```

Ejecución

```bash
```

npm run start:dev

```

La API se expone en: `http://localhost:3000/api/v1`.

Endpoints principales:

- `POST /api/v1/auth/login`
- `POST /api/v1/tickets`
- `PATCH /api/v1/tickets/:id/estado`
- `GET /api/v1/tickets/cliente/:clienteId`
- `GET /api/v1/tickets/:id/historial`

backend/src/main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
async function bootstrap() {
const app = await NestFactory.create(AppModule);
app.setGlobalPrefix('api/v1');
app.enableCors();
await app.listen(3000);
console.log('Posventa360 API corriendo en http://localhost:3000/api/v1');
}
bootstrap();
```

backend/src/app.module.ts
```

```
import { Module } from '@nestjs/common';
import { TicketsModule } from './tickets/tickets.module';
import { AuthModule } from './auth/auth.module';
@Module({
imports: [TicketsModule, AuthModule],
})
export class AppModule {}
```

backend/src/tickets/tickets.module.ts

```
import { Module } from '@nestjs/common';
import { TicketsService } from './tickets.service';
import { TicketsController } from './tickets.controller';
@Module({
controllers: [TicketsController],
providers: [TicketsService],
})
export class TicketsModule {}
```

backend/src/tickets/tickets.service.ts

```
import { Injectable, NotFoundException } from '@nestjs/common';
import { Ticket } from './entities/ticket.entity';
import { CreateTicketDto } from './dto/create-ticket.dto';
import { UpdateTicketStatusDto } from './dto/update-ticket-status.dto';
```

```
@Injectable()

export class TicketsService {

private tickets: Ticket[] = [];

private currentId = 1;

crearTicket(dto: CreateTicketDto): Ticket {

const ticket = new Ticket({

id: this.currentId++,

titulo: dto.titulo,

descripcion: dto.descripcion,

prioridad: dto.prioridad,

estado: 'abierto',

clienteId: dto.clienteId,

usuarioId: dto.usuarioId,

});

this.tickets.push(ticket);

return ticket;

}

actualizarEstado(id: number, dto: UpdateTicketStatusDto): Ticket {

const ticket = this.tickets.find(t => t.id === id);

if (!ticket) {

throw new NotFoundException('Ticket no encontrado');
```

```
}

ticket.estado = dto.nuevoEstado;

return ticket;

}

obtenerTicketsPorCliente(clienteId: number): Ticket[] {

return this.tickets.filter(t => t.clienteId === clienteId);

}

obtenerHistorialMock(id: number) {

const ticket = this.tickets.find(t => t.id === id);

if (!ticket) {

throw new NotFoundException('Ticket no encontrado');

}

return {

ticketId: id,

historial: [

{

fecha: ticket.creadoEn,

usuario: 'Sistema',

accion: 'Creación',

comentario: 'Ticket registrado',

},
```

```
],
};
}
}
```

backend/src/tickets/tickets.controller.ts

```
import { Controller, Post, Patch, Get, Param, Body, ParseIntPipe } from
'@nestjs/common';
import { TicketsService } from './tickets.service';
import { CreateTicketDto } from './dto/create-ticket.dto';
import { UpdateTicketStatusDto } from './dto/update-ticket-status.dto';
@Controller('tickets')
export class TicketsController {
constructor(private readonly ticketsService: TicketsService) {}
// Endpoint 1: Crear Ticket
@Post()
crearTicket(@Body() dto: CreateTicketDto) {
const ticket = this.ticketsService.crearTicket(dto);
return {
message: 'Ticket creado exitosamente',
ticketId: ticket.id,
estado: ticket.estado,
```

```
};
}
// Endpoint 2: Actualizar estado del Ticket
@Patch(':id/estado')
actualizarEstado(
@Param('id', ParseIntPipe) id: number,
@Body() dto: UpdateTicketStatusDto,
) {
const ticket = this.ticketsService.actualizarEstado(id, dto);
return {
message: 'Estado actualizado',
ticketId: ticket.id,
estadoActual: ticket.estado,
};
}
// Endpoint 3: Obtener Tickets por Cliente
@Get('/cliente/:clienteId')
obtenerPorCliente(@Param('clienteId', ParseIntPipe) clienteId: number) {
const tickets = this.ticketsService.obtenerTicketsPorCliente(clienteId);
return {
clienteId,
```

```
total: tickets.length,

tickets,

};

}

// Endpoint 5: Historial de un Ticket (versión mock)

@Get(':id/historial')

historial(@Param('id', ParseIntPipe) id: number) {

return this.ticketsService.obtenerHistorialMock(id);

}

}
```

backend/src/tickets/dto/create-ticket.dto.ts

```
import { TicketPrioridad } from '../entities/ticket.entity';

export class CreateTicketDto {

titulo: string;

descripcion: string;

prioridad: TicketPrioridad;

clienteId: number;

usuarioId: number;

}
```

backend/src/tickets/dto/update-ticket-status.dto.ts

```
import { TicketEstado } from '../entities/ticket.entity';
```

```
export class UpdateTicketStatusDto {

nuevoEstado: TicketEstado;

usuarioId: number;

comentario?: string;

}
```

backend/src/tickets/entities/ticket.entity.ts

```
export type TicketEstado = 'abierto' | 'en_proceso' | 'cerrado';

export type TicketPrioridad = 'baja' | 'media' | 'alta' | 'critica';

export class Ticket {

id: number;

titulo: string;

descripcion: string;

estado: TicketEstado;

prioridad: TicketPrioridad;

clienteId: number;

usuarioId: number;

creadoEn: Date;

constructor(partial: Partial<Ticket>) {

Object.assign(this, partial);

this.creadoEn = this.creadoEn ?? new Date();

}
```

```
}
```

backend/src/auth/auth.module.ts

```
import { Module } from '@nestjs/common';
import { AuthService } from './auth.service';
import { AuthController } from './auth.controller';
@Module({
controllers: [AuthController],
providers: [AuthService],
})
export class AuthModule {}
```

backend/src/auth/auth.service.ts

```
import { Injectable, UnauthorizedException } from '@nestjs/common';
@Injectable()
export class AuthService {
login(email: string, password: string) {
// DEMO: Validación simple para efectos académicos
if (email !== 'demo@claro.com' || password !== '123456') {
throw new UnauthorizedException('Credenciales inválidas');
}
return {
token: 'token-demo-posventa360',
```

usuario: {

id: 1,

nombre: 'Usuario Demo',

rol: 'Asesor PyME',

},

};

}

}

backend/src/auth/auth.controller.ts

import { Controller, Post, Body } from '@nestjs/common';

import { AuthService } from './auth.service';

@Controller('auth')

export class AuthController {

constructor(private readonly authService: AuthService) {}

// Endpoint 6: Autenticación

@Post('login')

login(@Body() body: { email: string; password: string }) {

return this.authService.login(body.email, body.password);

}

}

frontend/package.json

```
{
"name": "posventa360-frontend",
"version": "1.0.0",
"private": true,
"scripts": {
"dev": "vite",
"build": "vite build",
"preview": "vite preview"
},
"dependencies": {
"react": "^18.0.0",
"react-dom": "^18.0.0"
},
"devDependencies": {
"@types/react": "^18.0.0",
"@types/react-dom": "^18.0.0",
"typescript": "^5.0.0",
"vite": "^5.0.0"
}
}
```

frontend/tsconfig.json

```
{
"compilerOptions": {
"target": "ESNext",
"useDefineForClassFields": true,
"lib": [
"DOM",
"DOM.Iterable",
"ESNext"
],
"allowJs": false,
"skipLibCheck": true,
"esModuleInterop": true,
"allowSyntheticDefaultImports": true,
"strict": true,
"forceConsistentCasingInFileNames": true,
"module": "ESNext",
"moduleResolution": "bundler",
"resolveJsonModule": true,
"isolatedModules": true,
"noEmit": true,
"jsx": "react-jsx"
```

```
},
"include": [
"src"
]
}
```

frontend/vite.config.ts

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';
export default defineConfig({
plugins: [react()],
server: {
port: 5173,
},
});
```

frontend/index.html

```
<!doctype html>
<html lang="es">
<head>
<meta charset="UTF-8" />
<title>Posventa360</title>
</head>
```

<body>

<script type="module" src="/src/main.tsx"></script>

</body>

</html>

frontend/README.md

Posventa360 – Frontend (React + Vite demo)

Interfaz mínima para el proyecto Posventa360.

Requisitos

- Node.js 18+

Instalación

```bash

cd frontend

npm install

```

Ejecución

```bash

npm run dev

```

La app se abre en: `http://localhost:5173`.

Credenciales demo:

- Correo: `demo@claro.com`

- Contraseña: `123456`

frontend/src/main.tsx

```
import React from 'react';

import ReactDOM from 'react-dom/client';

import { App } from './App';

ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(

<React.StrictMode>

<App />

</React.StrictMode>

);
```

frontend/src/App.tsx

```
import React, { useState } from 'react';

import { Login } from './pages/Login';

import { Dashboard } from './pages/Dashboard';

export const App: React.FC = () => {

const [token, setToken] = useState<string | null>(localStorage.getItem('token'));

const [userName, setUserName] = useState<string |
null>(localStorage.getItem('userName'));

const handleLogin = (token: string, nombre: string) => {

setToken(token);
```

```
setUserName(nombre);

localStorage.setItem('token', token);

localStorage.setItem('userName', nombre);

};

const handleLogout = () => {

setToken(null);

setUserName(null);

localStorage.removeItem('token');

localStorage.removeItem('userName');

};

if (!token) {

return <Login onLogin={handleLogin} />;

}

return <Dashboard token={token} userName={userName} onLogout={handleLogout} />;

};
```

frontend/src/pages/Login.tsx

```
import React, { useState } from 'react';

import { login } from '../services/api';

interface Props {

onLogin: (token: string, nombre: string) => void;

}
```

```
export const Login: React.FC<Props> = ({ onLogin }) => {

const [email, setEmail] = useState('demo@claro.com');

const [password, setPassword] = useState('123456');

const [error, setError] = useState<string | null>(null);

const [loading, setLoading] = useState(false);

const handleSubmit = async (e: React.FormEvent) => {

e.preventDefault();

setLoading(true);

setError(null);

try {

const res = await login(email, password);

onLogin(res.token, res.usuario.nombre);

} catch (err: any) {

setError('Credenciales inválidas');

} finally {

setLoading(false);

}

};

return (

<div style={{ maxWidth: 400, margin: '80px auto', padding: 24, border: '1px solid #ddd',
borderRadius: <h2>Posventa360 - Login</h2>
```

```
<form onSubmit={handleSubmit}>

<div style={{ marginBottom: 12 }}>

<label>Correo</label>

<input

type="email"

value={email}

onChange={e => setEmail(e.target.value)}

style={{ width: '100%' }}

/>

</div>

<div style={{ marginBottom: 12 }}>

<label>Contraseña</label>

<input

type="password"

value={password}

onChange={e => setPassword(e.target.value)}

style={{ width: '100%' }}

/>

</div>

{error && <p style={{ color: 'red' }}>{error}</p>}

<button type="submit" disabled={loading}>
```

```
{loading ? 'Ingresando...' : 'Ingresar'}

</button>

</form>

<p style={{ fontSize: 12, marginTop: 12 }}>Demo: demo@claro.com / 123456</p>

</div>

);

};
```

frontend/src/pages/Dashboard.tsx

```
import React from 'react';

import { TicketList } from '../components/TicketList';

interface Props {

token: string;

userName: string | null;

onLogout: () => void;

}

export const Dashboard: React.FC<Props> = ({ token, userName, onLogout }) => {

return (

<div style={{ padding: 24 }}>

<header style={{ display: 'flex', justifyContent: 'space-between', marginBottom: 24 }}>

<h2>Posventa360 - Dashboard</h2>

<div>
```

```
<span style={{ marginRight: 12 }}>Hola, {userName ?? 'Usuario'}</span>

<button onClick={onLogout}>Cerrar sesión</button>

</div>

</header>

<section>

<h3>Tickets del Cliente #1</h3>

<TicketList token={token} clienteId={1} />

</section>

</div>

);

};
```

frontend/src/components/TicketList.tsx

```
import React, { useEffect, useState } from 'react';

import { createTicket, getTicketsByCliente } from '../services/api';

interface Ticket {

id: number;

titulo: string;

descripcion: string;

estado: string;

prioridad: string;

}
```

```
interface Props {

token: string;

clienteId: number;

}

export const TicketList: React.FC<Props> = ({ token, clienteId }) => {

const [tickets, setTickets] = useState<Ticket[]>([]);

const [titulo, setTitulo] = useState('Falla en enlace dedicado');

const [descripcion, setDescripcion] = useState('Cliente reporta intermitencia');

const [prioridad, setPrioridad] = useState('alta');

const cargarTickets = async () => {

const res = await getTicketsByCliente(clienteId);

setTickets(res.tickets ?? []);

};

useEffect(() => {

cargarTickets();

}, []);

const handleCrear = async (e: React.FormEvent) => {

e.preventDefault();

await createTicket({

titulo,

descripcion,
```

```
prioridad,
clienteId,
usuarioId: 1,
});
await cargarTickets();
};
return (
<div>
<form onSubmit={handleCrear} style={{ marginBottom: 16 }}>
<input
placeholder="Título"
value={titulo}
onChange={e => setTitulo(e.target.value)}
style={{ marginRight: 8 }}
/>
<input
placeholder="Descripción"
value={descripcion}
onChange={e => setDescripcion(e.target.value)}
style={{ marginRight: 8 }}
/>
```

```
<select value={prioridad} onChange={e => setPrioridad(e.target.value)} style={{
marginRight: 8 }}>

<option value="baja">Baja</option>

<option value="media">Media</option>

<option value="alta">Alta</option>

<option value="critica">Crítica</option>

</select>

<button type="submit">Crear Ticket</button>

</form>

{tickets.length === 0 ? (

<p>No hay tickets aún.</p>

) : (

<table border={1} cellPadding={4}>

<thead>

<tr>

<th>ID</th>

<th>Título</th>

<th>Estado</th>

<th>Prioridad</th>

</tr>

</thead>
```

```
<tbody>
{tickets.map(t => (
<tr key={t.id}>
<td>{t.id}</td>
<td>{t.titulo}</td>
<td>{t.estado}</td>
<td>{t.prioridad}</td>
</tr>
))}
</tbody>
</table>
)}
</div>
);
};
```

frontend/src/services/api.ts

```
const API_URL = 'http://localhost:3000/api/v1';
export async function login(email: string, password: string) {
const res = await fetch(`${API_URL}/auth/login`, {
method: 'POST',
headers: { 'Content-Type': 'application/json' },
```

```
body: JSON.stringify({ email, password }),
});
if (!res.ok) {
throw new Error('Error de login');
}
return res.json();
}
export async function getTicketsByCliente(clienteId: number) {
const res = await fetch(`${API_URL}/tickets/cliente/${clienteId}`);
if (!res.ok) {
throw new Error('Error al obtener tickets');
}
return res.json();
}
interface CreateTicketPayload {
titulo: string;
descripcion: string;
prioridad: string;
clienteId: number;
usuarioId: number;
}
```

```
export async function createTicket(payload: CreateTicketPayload) {

const res = await fetch(`${API_URL}/tickets`, {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify(payload),

});

if (!res.ok) {

throw new Error('Error al crear ticket');

}

return res.json();

}
```