🚀 Planning Completo - Pland-IA (Arquitectura Microservicios)

> **Proyecto:** Sistema multiplataforma de productividad con Despensa Inteligente
> **Duración:** 12 semanas (~240-300 horas)
> **Desarrollador:** Mario Alguacil Juárez
> **Fecha inicio:** Noviembre 2025
> **Objetivo:** Portfolio Full-Stack + Aprendizaje profundo

📋 Índice

🎯 Concepto del Proyecto

⚠️ IMPORTANTE: Pland-IA NO es solo un planificador de comidas.

Pland-IA es un NOTION SIMPLE - Un planificador personal completo donde organizas TODA tu vida:

Funcionalidad Principal (80%):

- 🗂️ **Workspaces:** Espacios de trabajo (Trabajo, Personal, Estudios, etc.)
- 📊 **Projects:** Proyectos dentro de cada workspace
- 📄 **Pages:** Páginas/documentos/notas (como en Notion)
- ☑️ **Tasks:** Tareas con estados, prioridades, fechas límite
- 🔍 **Búsqueda:** Encuentra cualquier cosa rápidamente
- 🎨 **Organización visual:** Kanban, listas, vistas personalizadas

Feature Diferenciadora (20%):

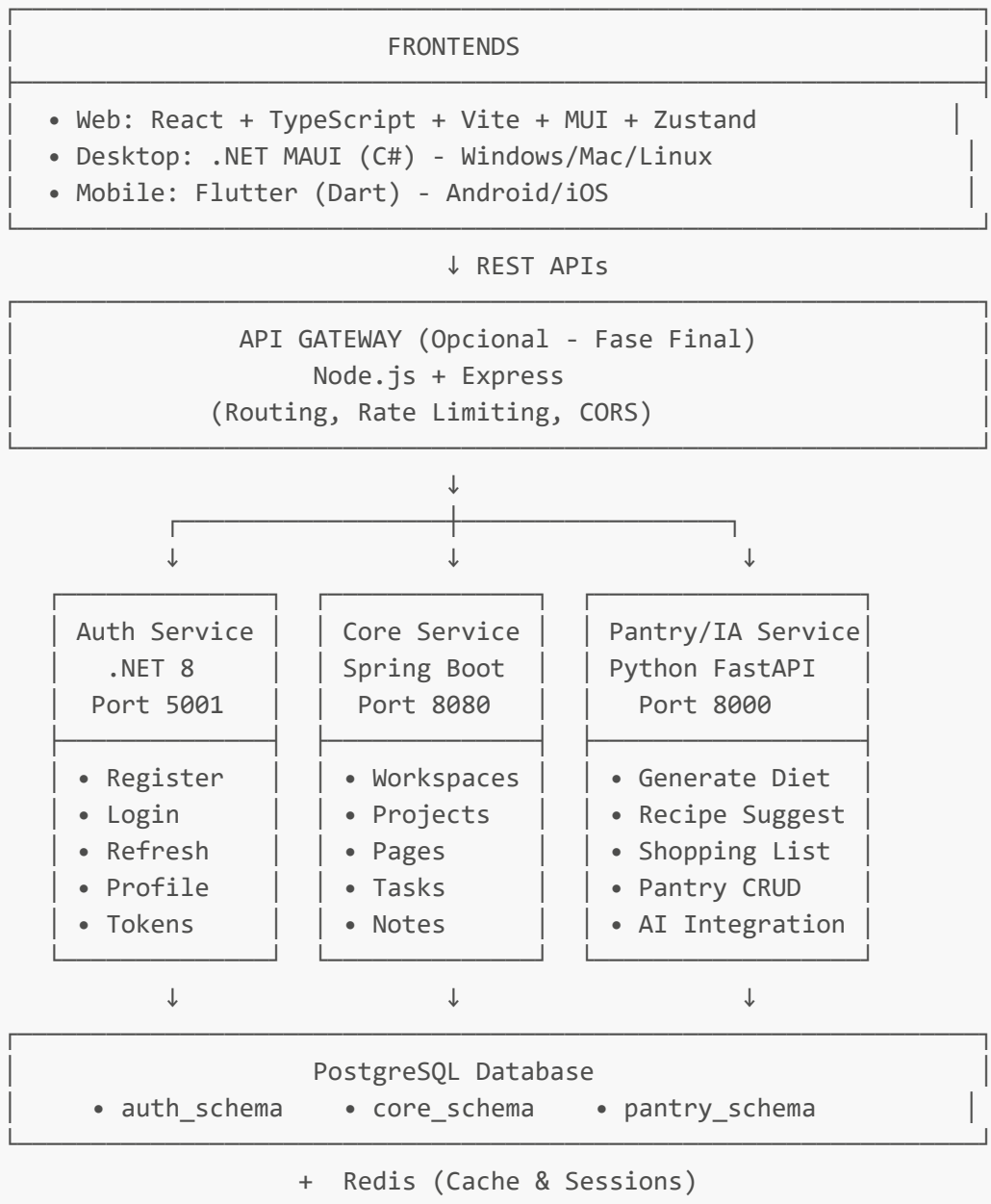
- 🍽️ **Despensa Inteligente con IA:**
  - Gestionar ingredientes de tu despensa
  - Generar dietas personalizadas con OpenAI
  - Recibir sugerencias de recetas según lo que tienes
  - Crear listas de compra automáticas

¿Por qué esto es valioso?

- Notion es COMPLEJO y abrumador para muchos usuarios
- Pland-IA es SIMPLE: el 80% de la potencia con el 20% de la complejidad
- La Despensa Inteligente lo hace ÚNICO - nadie más tiene esto

---

🏗️ Arquitectura del Sistema



🛠️ Stack Tecnológico Completo

Backend (Microservicios)

1️⃣ **Auth Service (.NET Core 8)**

```
Lenguaje: C#
Framework: ASP.NET Core Web API
ORM: Entity Framework Core
Database: PostgreSQL (schema: auth_schema)
Autenticación: JWT (Access + Refresh Tokens)
Password Hash: BCrypt.Net
Testing: xUnit + Moq
```

```
Documentación: Swagger/OpenAPI
Containerización: Docker
```

Dependencias principales:

- Microsoft.AspNetCore.Authentication.JwtBearer
- Microsoft.EntityFrameworkCore.PostgreSQL
- BCrypt.Net-Next
- Swashbuckle.AspNetCore
- Serilog.AspNetCore
- FluentValidation

Responsabilidades:

- ☑ Registro de usuarios
- ☑ Login con JWT
- ☑ Refresh tokens
- ☑ Gestión de perfil
- ☑ Validación de tokens para otros servicios

---

2 Core Service (Spring Boot 3.x)

```
Lenguaje: Java 17+ (o Kotlin)
Framework: Spring Boot
ORM: Spring Data JPA (Hibernate)
Database: PostgreSQL (schema: core_schema)
Seguridad: Spring Security + JWT validation
Testing: JUnit 5 + Mockito + TestContainers
Documentación: SpringDoc OpenAPI
Containerización: Docker
```

Dependencias principales:

- spring-boot-starter-web
- spring-boot-starter-data-jpa
- spring-boot-starter-security
- spring-boot-starter-validation
- postgresql
- lombok
- mapstruct (DTOs)
- springdoc-openapi-starter-webmvc-ui

Responsabilidades:

- ☑ CRUD de Workspaces
- ☑ CRUD de Projects
- ☑ CRUD de Pages (contenido tipo Notion)

- ☑ CRUD de Tasks (con estados, prioridades)
- ☑ Búsqueda y filtros
- ☑ Paginación y ordenamiento
- ☑ Validación de permisos por usuario

Entidades principales:

```
Workspace (id, name, userId, createdAt, updatedAt)
Project (id, workspaceId, name, description, color, icon)
Page (id, projectId, title, content, parentPageId)
Task (id, pageId, title, description, status, priority, dueDate)
```

---

3 Pantry/IA Service (Python 3.11+ FastAPI)

```
Lenguaje: Python
Framework: FastAPI
ORM: SQLAlchemy 2.0
Database: PostgreSQL (schema: pantry_schema)
IA: OpenAI API (GPT-4)
Testing: pytest + httpx
Documentación: FastAPI auto-docs
Containerización: Docker
```

Dependencias principales:

- fastapi
- uvicorn[standard]
- sqlalchemy
- asyncpg
- pydantic
- openai
- python-jose[cryptography] (JWT)
- python-multipart
- pytest
- httpx

Responsabilidades:

- ☑ CRUD de Despensa (ingredientes, cantidades, fechas caducidad)
- ☑ CRUD de Recetas
- ☑ Generación de Dietas con IA (OpenAI)
- ☑ Sugerencias de recetas según despensa
- ☑ Generación automática de listas de compra
- ☑ Análisis nutricional

Entidades principales:

```
Diet (id, user_id, name, start_date, end_date, calories, preferences)
Meal (id, diet_id, day_of_week, meal_type, recipe_id)
Recipe (id, name, ingredients_json, instructions, calories, prep_time)
Ingredient (id, name, category, unit)
PantryItem (id, user_id, ingredient_id, quantity, expiry_date)
ShoppingList (id, user_id, status, items_json, created_at)
```

Frontend

4️⃣ Web Application (React 18 + TypeScript)

```
Framework: React 18
Lenguaje: TypeScript
Build Tool: Vite
UI Library: Material-UI (MUI)
State Management: Zustand
Routing: React Router v6
HTTP Client: Axios
Forms: React Hook Form + Zod
Testing: Vitest + React Testing Library
```

Características:

- ☑ Diseño responsive
- ☑ Dark mode
- ☑ PWA (Progressive Web App)
- ☑ Code splitting
- ☑ Lazy loading
- ☑ Internacionalización (i18n)

5️⃣ Desktop Application (.NET MAUI)

```
Framework: .NET MAUI
Lenguaje: C#
Pattern: MVVM
UI: XAML
HTTP Client: HttpClient
Plataformas: Windows, macOS, Linux
```

6️⃣ Mobile Application (Flutter)

```
Framework: Flutter 3.x
Lenguaje: Dart
State Management: Riverpod
Routing: go_router
HTTP Client: dio
Local Storage: Hive
Plataformas: Android, iOS
```

📅 Planning Detallado por Fases

🟢 FASE 1: Auth Service con .NET Core (Semanas 1-2)

⏰ Dedicación: 30-40 horas

🎯 Objetivo: Servicio de autenticación completo con JWT

📝 Semana 1 (15-20 horas)

Lunes-Martes (4-5h): Setup y Fundamentos

Tareas paso a paso:

1. **Instalar .NET 8 SDK**

```
# Windows (winget)
winget install Microsoft.DotNet.SDK.8

# Verificar instalación
dotnet --version  # Debe mostrar 8.x.x
```

2. **Crear proyecto Web API**

```
cd c:\Users\mario\Desktop\Pland-IA\apps
mkdir auth-service
cd auth-service

dotnet new webapi -n AuthService
cd AuthService

# Abrir en VS Code
code .
```

3. **Instalar dependencias**

```
# Entity Framework Core
dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.Design
dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL

# JWT
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

# Password Hashing
dotnet add package BCrypt.Net-Next

# Validación
dotnet add package FluentValidation.AspNetCore

# Logging
dotnet add package Serilog.AspNetCore

# Swagger
dotnet add package Swashbuckle.AspNetCore
```

4. **Estructura de carpetas**

```
AuthService/
├── Controllers/
├── Services/
├── Models/
│   ├── Entities/
│   └── DTOs/
├── Data/
├── Middleware/
└── Program.cs
```

5. **Primer endpoint de prueba**

```
// Controllers/HealthController.cs
[ApiController]
[Route("api/[controller]")]
public class HealthController : ControllerBase
{
    [HttpGet]
    public IActionResult Get()
    {
        return Ok(new { status = "healthy", service = "auth-service" });
    }
}
```

6. **Ejecutar y probar**

```
dotnet watch run
# Abrir: https://localhost:5001/api/health
```

🎓 **Conceptos aprendidos:**

- Estructura de un proyecto Web API en .NET
- Program.cs y configuración de servicios
- Inyección de dependencias básica
- Hot reload con `dotnet watch`
- Routing y Controllers

Miércoles-Jueves (5-6h): Base de Datos

Tareas paso a paso:

1. **Crear modelo User**

```
// Models/Entities/User.cs
public class User
{
    public Guid Id { get; set; }
    public string Email { get; set; } = string.Empty;
    public string PasswordHash { get; set; } = string.Empty;
    public string Name { get; set; } = string.Empty;
    public string? Avatar { get; set; }
    public DateTime CreatedAt { get; set; }
    public DateTime UpdatedAt { get; set; }
}
```

2. **Crear DbContext**

```
// Data/AppDbContext.cs
public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options)
        : base(options) { }

    public DbSet<User> Users { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.HasDefaultSchema("auth_schema");

        modelBuilder.Entity<User>(entity =>
        {
            entity.HasKey(e => e.Id);
            entity.HasIndex(e => e.Email).IsUnique();
```

```
            entity.Property(e => e.Email).IsRequired().HasMaxLength(255);
            entity.Property(e => e.PasswordHash).IsRequired();
            entity.Property(e => e.Name).IsRequired().HasMaxLength(255);
        });
    }
}
```

3. **Configurar conexión en appsettings.json**

```
{
  "ConnectionStrings": {
    "DefaultConnection":
"Host=localhost;Port=5432;Database=plandiadb;Username=postgres;Password=myse
cretpassword;Search Path=auth_schema"
  }
}
```

4. **Registrar DbContext en Program.cs**

```
builder.Services.AddDbContext<AppDbContext>(options =>

options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnecti
on")));
```

5. **Crear migración inicial**

```
# Instalar herramientas de EF
dotnet tool install --global dotnet-ef

# Crear migración
dotnet ef migrations add InitialCreate

# Aplicar a la BD
dotnet ef database update
```

🎓 **Conceptos aprendidos:**

- Entity Framework Core Code-First
- DbContext y DbSet
- Configuración de entidades con Fluent API
- Migraciones de base de datos
- Connection strings

Viernes-Domingo (8-9h): Autenticación JWT

Tareas paso a paso:

1. **Crear DTOs**

```
// Models/DTOs/RegisterDto.cs
public record RegisterDto(string Email, string Password, string Name);

// Models/DTOs/LoginDto.cs
public record LoginDto(string Email, string Password);

// Models/DTOs/AuthResponseDto.cs
public record AuthResponseDto(string Token, string RefreshToken, UserDto
User);

// Models/DTOs/UserDto.cs
public record UserDto(Guid Id, string Email, string Name, string? Avatar);
```

2. **Crear servicio de autenticación**

```
// Services/IAuthService.cs
public interface IAuthService
{
    Task<AuthResponseDto> RegisterAsync(RegisterDto dto);
    Task<AuthResponseDto> LoginAsync(LoginDto dto);
    Task<AuthResponseDto> RefreshTokenAsync(string refreshToken);
}
```

3. **Implementar lógica de hash y JWT** (te guiaré en la implementación)

4. **Crear AuthController**

5. **Configurar JWT en Program.cs**

6. **Probar endpoints**

🎓 **Conceptos aprendidos:**

- BCrypt para hash de passwords
- Generación de JWT tokens
- Claims y políticas
- Middleware de autenticación
- Atributo `[Authorize]`

🟧 Semana 2 (15-20 horas)

Lunes-Martes (5-6h): Refresh Tokens

- Modelo RefreshToken en BD

- Lógica de generación y validación
- Endpoint `/api/auth/refresh`
- Revocación de tokens

Miércoles-Jueves (5-6h): Validación y Errores

- FluentValidation para DTOs
- Middleware de excepciones global
- Responses estandarizados
- Logging con Serilog

Viernes-Domingo (5-8h): Testing y Docker

- Tests con xUnit
- Swagger configurado
- Dockerfile
- README del servicio
- CORS configurado

☑ Entregable Fase 1

- ☑ Auth Service funcional
- ☑ Endpoints: `/register`, `/login`, `/refresh`, `/profile`
- ☑ JWT implementado
- ☑ Tests básicos
- ☑ Dockerizado
- ☑ Documentación Swagger en `/swagger`

🟢 FASE 2: Core Service con Spring Boot (Semanas 3-5)

⏰ Dedicación: 45-60 horas

🎯 Objetivo: API REST completa para gestión de productividad

📅 Semana 3 (15-20 horas)

Lunes-Martes (4-5h): Setup Spring Boot

Tareas:

1. **Generar proyecto en Spring Initializr**
   - Ir a https://start.spring.io/
   - Project: Maven
   - Language: Java
   - Spring Boot: 3.2.x
   - Group: com.plandaia
   - Artifact: core-service
   - Dependencies:

- Spring Web
- Spring Data JPA
- PostgreSQL Driver
- Spring Security
- Lombok
- Validation
- Spring Boot DevTools

2. **Descargar y descomprimir en** `apps/core-service`

3. **Estructura de carpetas**

```
core-service/
└── src/main/java/com/plandaia/core/
    ├── controller/
    ├── service/
    ├── repository/
    ├── model/
    │   ├── entity/
    │   └── dto/
    ├── config/
    ├── exception/
    └── CoreServiceApplication.java
```

4. **Configurar application.yml**

```
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/plandiadb?
currentSchema=core_schema
    username: postgres
    password: mysecretpassword
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        default_schema: core_schema
server:
  port: 8080
```

5. **Primer controller de prueba**

🎓 **Conceptos aprendidos:**

- Spring Boot Initializr
- application.yml vs properties

- Anotaciones: @RestController, @Service, @Repository
- Lombok (@Data, @Builder, @NoArgsConstructor)

Miércoles-Viernes (8-10h): Modelos y Relaciones JPA

Entidades a crear:

1. **Workspace**

```
@Entity
@Table(name = "workspaces")
public class Workspace {
    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private UUID id;

    @Column(nullable = false)
    private UUID userId;  // Del Auth Service

    @Column(nullable = false)
    private String name;

    private String description;

    @OneToMany(mappedBy = "workspace", cascade = CascadeType.ALL)
    private List<Project> projects = new ArrayList<>();

    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
}
```

2. **Project**

3. **Page**

4. **Task**

DTOs con MapStruct o manual

🎓 **Conceptos aprendidos:**

- JPA Entities y anotaciones
- @OneToMany, @ManyToOne
- CascadeType y FetchType
- DTOs para separar capas

Sábado-Domingo (5-6h): Repositorios y Servicios

Tareas:

1. **Crear JpaRepository para cada entidad**

```
@Repository
public interface WorkspaceRepository extends JpaRepository<Workspace, UUID>
{
    List<Workspace> findByUserId(UUID userId);

    @Query("SELECT w FROM Workspace w WHERE w.userId = :userId AND w.name
LIKE %:name%")
    List<Workspace> searchByUserIdAndName(@Param("userId") UUID userId,
@Param("name") String name);
}
```

2. **Implementar servicios con lógica de negocio**

3. **Validaciones con Bean Validation**

🎓 **Conceptos aprendidos:**

- Spring Data JPA
- Derived queries
- @Query con JPQL
- @Transactional

📓 Semana 4 (15-20 horas)

Lunes-Miércoles (8-10h): Seguridad e Integración con Auth Service

Tareas:

1. **Configurar Spring Security**
2. **JWT Filter para validar tokens**
3. **RestTemplate o WebClient para llamar al Auth Service (.NET)**
4. **SecurityContext con userId**
5. **@PreAuthorize en endpoints**

🎓 **Conceptos aprendidos:**

- Spring Security filters
- JWT validation sin generar tokens
- Comunicación entre microservicios
- SecurityContextHolder

Jueves-Domingo (10-12h): CRUD Completo

Endpoints a implementar:

Workspaces:

- GET `/api/workspaces` - Listar del usuario autenticado
- POST `/api/workspaces` - Crear
- GET `/api/workspaces/{id}` - Obtener uno
- PUT `/api/workspaces/{id}` - Actualizar
- DELETE `/api/workspaces/{id}` - Eliminar

Projects:

- GET `/api/workspaces/{workspaceId}/projects`
- POST `/api/workspaces/{workspaceId}/projects`
- PUT `/api/projects/{id}`
- DELETE `/api/projects/{id}`

Pages y Tasks: Similar

Implementar:

- Paginación con Pageable
- Ordenamiento
- Búsqueda y filtros
- Validación de permisos (solo el dueño puede editar)

📝 Semana 5 (12-15 horas)

Testing y Optimización

Tareas:

1. **Tests unitarios con JUnit 5 y Mockito**

```
@ExtendWith(MockitoExtension.class)
class WorkspaceServiceTest {
    @Mock
    private WorkspaceRepository repository;

    @InjectMocks
    private WorkspaceService service;

    @Test
    void shouldCreateWorkspace() {
        // Arrange, Act, Assert
    }
}
```

2. **Tests de integración con TestContainers**
3. **Exception handling con @ControllerAdvice**
4. **Caching con @Cacheable**

5. **SpringDoc OpenAPI** (documentación automática)
6. **Dockerfile**
7. **Docker Compose** (Auth + Core + PostgreSQL + Redis)

☑ Entregable Fase 2

- ☑ Core Service funcional
- ☑ CRUD completo de Workspaces, Projects, Pages, Tasks
- ☑ Integrado con Auth Service
- ☑ Tests unitarios e integración
- ☑ Documentación OpenAPI en `/swagger-ui`
- ☑ Docker Compose funcional

🟢 FASE 3: Pantry/IA Service con Python FastAPI (Semanas 6-7)

⏰ Dedicación: 30-40 horas

🎯 Objetivo: API de Despensa Inteligente con IA

📝 Semana 6 (15-20 horas)

Lunes-Martes (4-5h): Setup FastAPI

Tareas:

1. **Crear entorno virtual**

```
cd c:\Users\mario\Desktop\Pland-IA\apps
mkdir pantry-service
cd pantry-service

python -m venv venv
venv\Scripts\activate  # Windows
```

2. **Instalar dependencias**

```
pip install fastapi uvicorn[standard] sqlalchemy asyncpg pydantic python-
jose[cryptography] openai python-dotenv

# Guardar
pip freeze > requirements.txt
```

3. **Estructura de carpetas**

```
pantry-service/
├── app/
│   ├── __init__.py
│   ├── main.py
│   ├── config.py
│   ├── database.py
│   ├── models/
│   ├── schemas/
│   ├── routers/
│   ├── services/
│   └── middleware/
├── tests/
├── requirements.txt
└── .env
```

4. **Primer endpoint asíncrono**

```python
# app/main.py
from fastapi import FastAPI

app = FastAPI(title="Pantry/IA Service", version="1.0.0")

@app.get("/health")
async def health():
    return {"status": "healthy", "service": "pantry-service"}
```

5. **Ejecutar**

```
uvicorn app.main:app --reload --port 8000
# Docs automáticas: http://localhost:8000/docs
```

🎓 **Conceptos aprendidos:**

- FastAPI framework
- Async/await en Python
- Type hints con Pydantic
- Auto-documentación con Swagger

Miércoles-Viernes (8-10h): Modelos SQLAlchemy

Modelos a crear:

```python
# app/models/diet.py
from sqlalchemy import Column, String, Integer, Date, JSON
from sqlalchemy.dialects.postgresql import UUID
```

```
import uuid

class Diet(Base):
    __tablename__ = "diets"
    __table_args__ = {'schema': 'pantry_schema'}

    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid.uuid4)
    user_id = Column(UUID(as_uuid=True), nullable=False)
    name = Column(String(255), nullable=False)
    start_date = Column(Date, nullable=False)
    end_date = Column(Date, nullable=False)
    calories = Column(Integer)
    preferences = Column(JSON)  # ["vegetarian", "no_gluten"]
    created_at = Column(DateTime, default=datetime.utcnow)
```

Otros modelos:

- Meal
- Recipe
- Ingredient
- PantryItem
- ShoppingList

Sábado-Domingo (5-6h): CRUD Básico

Endpoints:

- POST `/api/pantry/items` - Añadir ingrediente
- GET `/api/pantry/items` - Listar despensa del usuario
- PUT `/api/pantry/items/{id}` - Actualizar cantidad
- DELETE `/api/pantry/items/{id}` - Eliminar
- GET `/api/recipes` - Listar recetas
- POST `/api/recipes` - Crear receta

Middleware de autenticación (validar JWT del Auth Service)

Semana 7 (15-20 horas)

Lunes-Miércoles (8-10h): Integración con OpenAI

Tareas:

1. **Instalar OpenAI SDK**

```
pip install openai
```

2. **Configurar API key en .env**

```
OPENAI_API_KEY=sk-...
```

3. **Crear servicio de IA**

```
# app/services/ai_service.py
from openai import OpenAI

class AIService:
    def __init__(self):
        self.client = OpenAI()

    async def generate_diet(self, user_prefs: dict) -> dict:
        prompt = self._build_diet_prompt(user_prefs)

        response = self.client.chat.completions.create(
            model="gpt-4",
            messages=[
                {"role": "system", "content": "Eres un nutricionista
experto..."},
                {"role": "user", "content": prompt}
            ],
            response_format={"type": "json_object"}
        )

        return response.choices[0].message.content
```

4. **Endpoint de generación**

```
@router.post("/ai/generate-diet")
async def generate_diet(request: GenerateDietRequest):
    # 1. Llamar a OpenAI
    # 2. Parsear respuesta JSON
    # 3. Guardar Diet y Meals en BD
    # 4. Retornar plan semanal
```

🎓 **Conceptos aprendidos:**

- OpenAI Chat Completions API
- Prompt engineering
- JSON mode
- Manejo de rate limits

Jueves-Viernes (5-6h): Sugerencias de Recetas

Endpoint:

```
@router.post("/ai/suggest-recipes")
async def suggest_recipes(user_id: UUID):
    # 1. Obtener ingredientes de la despensa del usuario
    # 2. Construir prompt con ingredientes disponibles
    # 3. Llamar a OpenAI
    # 4. Parsear recetas sugeridas
    # 5. Retornar lista
```

Sábado-Domingo (5-6h): Lista de Compra Inteligente

Endpoint:

```
@router.post("/ai/generate-shopping-list")
async def generate_shopping_list(diet_id: UUID, user_id: UUID):
    # 1. Obtener comidas de la dieta
    # 2. Extraer ingredientes necesarios
    # 3. Comparar con despensa actual
    # 4. Calcular ingredientes faltantes
    # 5. Crear ShoppingList
    # 6. Retornar lista optimizada por categorías
```

☑ Entregable Fase 3

- ☑ Pantry Service funcional
- ☑ CRUD de despensa, recetas, dietas
- ☑ Integración con OpenAI
- ☑ Generación de dietas personalizadas
- ☑ Sugerencias de recetas
- ☑ Lista de compra automática
- ☑ Dockerizado

FASE 4: Frontend Web con React (Semanas 8-9)

Dedicación: 30-40 horas

Objetivo: Web app completa y responsive

Semana 8 (15-20 horas)

Lunes-Martes (4-5h): Setup React

Tareas:

```
cd c:\Users\mario\Desktop\Pland-IA\apps
npm create vite@latest web-desktop -- --template react-ts
cd web-desktop
npm install

# Dependencias
npm install @mui/material @emotion/react @emotion/styled
npm install zustand
npm install react-router-dom
npm install axios
npm install react-hook-form zod @hookform/resolvers
npm install @tanstack/react-query
```

Estructura:

```
web-desktop/
└── src/
    ├── components/
    ├── pages/
    ├── services/
    ├── stores/
    ├── hooks/
    ├── types/
    └── utils/
```

Miércoles-Viernes (8-10h): Autenticación

Páginas:

- LoginPage
- RegisterPage

Store de auth con Zustand:

```
interface AuthState {
  user: User | null;
  token: string | null;
  login: (email: string, password: string) => Promise<void>;
  logout: () => void;
}
```

Axios interceptor para JWT

ProtectedRoute component

Sábado-Domingo (5-6h): Layout Base

- AppBar con menú
- Sidebar con navegación
- Theme MUI personalizado
- Dark mode

Semana 9 (15-20 horas)

Lunes-Miércoles (8-10h): Workspaces y Tareas

Páginas:

- WorkspacesPage
- ProjectsPage
- TasksBoardPage (Kanban)

Jueves-Domingo (10-12h): Despensa Inteligente

Páginas:

- PantryPage (CRUD ingredientes)
- DietGeneratorPage (formulario con IA)
- WeeklyPlanPage (plan semanal)
- RecipesPage (recetas sugeridas)
- ShoppingListPage (checklist)

Entregable Fase 4

- [x] Web app funcional
- [x] Login/Register
- [x] Gestión de productividad
- [x] Despensa inteligente
- [x] UI profesional con MUI

FASE 5: Desktop con .NET MAUI (Semanas 10-11)

Dedicación: 25-30 horas

Resumen:

- Proyecto .NET MAUI
- MVVM pattern
- Consumir APIs REST
- Compilar para Windows/Mac/Linux

🟢 FASE 6: Mobile con Flutter (Semanas 11-12)

⏰ Dedicación: 25-30 horas

Resumen:

- Flutter project
- Riverpod
- UI móvil optimizada
- Build para Android

☑ Checklist de Progreso General

Backend

- ☐ Auth Service (.NET) - Semanas 1-2
  - ☐ Setup y proyecto base
  - ☐ Entity Framework + PostgreSQL
  - ☐ JWT Authentication
  - ☐ Refresh Tokens
  - ☐ Tests y Docker
- ☐ Core Service (Spring Boot) - Semanas 3-5
  - ☐ Setup y entidades JPA
  - ☐ Repositorios y servicios
  - ☐ Spring Security + JWT validation
  - ☐ CRUD completo
  - ☐ Tests y Docker Compose
- ☐ Pantry/IA Service (Python) - Semanas 6-7
  - ☐ Setup FastAPI + SQLAlchemy
  - ☐ Modelos de despensa
  - ☐ OpenAI integration
  - ☐ Generación de dietas
  - ☐ Docker

Frontend

- ☐ Web (React) - Semanas 8-9
  - ☐ Setup Vite + React + TypeScript
  - ☐ Autenticación
  - ☐ Layout y navegación
  - ☐ Productividad (workspaces, tasks)
  - ☐ Despensa inteligente
- ☐ Desktop (.NET MAUI) - Semanas 10-11
  - ☐ Proyecto MAUI
  - ☐ MVVM
  - ☐ HTTP Client
  - ☐ Build multiplataforma

- [ ] Mobile (Flutter) - Semanas 11-12
  - [ ] Flutter setup
  - [ ] Pantallas principales
  - [ ] HTTP con dio
  - [ ] Build Android

DevOps

- [ ] Docker para cada servicio
- [ ] Docker Compose completo
- [ ] GitHub Actions CI/CD
- [ ] README profesional con capturas
- [ ] Video demo (opcional)

Recursos de Aprendizaje

.NET / C#

- Documentación oficial ASP.NET Core
- Entity Framework Core
- JWT en .NET

Spring Boot / Java

- Spring Boot Reference
- Spring Data JPA
- Spring Security

Python / FastAPI

- FastAPI Documentation
- SQLAlchemy 2.0
- OpenAI API Reference

React / TypeScript

- React Docs
- TypeScript Handbook
- Material-UI
- Zustand

Flutter / Dart

- Flutter Documentation
- Riverpod

Valor de Portfolio

Este proyecto demuestra:

- ☑ **Arquitectura de Microservicios**
- ☑ **Backend multipllenguaje** (.NET, Spring Boot, Python)
- ☑ **Frontend multiplataforma** (Web, Desktop, Mobile)
- ☑ **Integración con IA** (OpenAI)
- ☑ **Bases de datos relacionales** (PostgreSQL)
- ☑ **Testing** (xUnit, JUnit, pytest)
- ☑ **Docker y containerización**
- ☑ **JWT Authentication**
- ☑ **Clean Architecture**
- ☑ **Git workflow profesional**

🎯 Próximo Paso

Cuando estés listo para empezar:

```
"Listo para empezar con .NET Auth Service.
Guíame para instalar el SDK y crear el proyecto."
```

Creado por: Mario Alguacil Juárez
Fecha: Noviembre 2025
Versión: 1.0