Campus Santa Fe

Assessment Final

Alumno:

David Santiago Vieyra García - A01656030

Ian Holender Mariaca - A01026295

Luis Carlos Rico Almada - A01252831

Omar Rivera Arenas - A01374690

Fecha de entrega:

9 de junio de 2025

1. Tabla de Contenidos

2. Introducción

2.1 Contexto

Este proyecto surge como trabajo integrador de octavo semestre de la carrera de Ingeniería en Tecnología Computacional en el Tecnológico de Monterrey Campus Santa Fe. Su propósito principal es poner a prueba en un entorno realista los conocimientos adquiridos.

La problemática que se ataca con este proyecto se centra en el área de la agricultura, donde las decisiones de siembra tradicionalmente se basan en costumbre o intuición más que en datos científicos. Esta situación provoca:

- **Selección inadecuada de cultivos:** Los agricultores eligen cultivos poco compatibles con las condiciones químicas y climáticas específicas de su suelo
- **Uso ineficiente de recursos:** Desperdicio de agua, fertilizantes y pesticidas por falta de información precisa

El sistema **AgriAI** automatiza la recomendación de cultivos óptimos mediante el análisis científico de parámetros como niveles de Nitrógeno (N), Fósforo (P), Potasio (K), pH del suelo, temperatura promedio, humedad relativa y precipitación histórica. La plataforma ofrece al productor una predicción de cultivo recomendado con niveles de confianza, permitiéndole tomar decisiones informadas antes de la siembra.

2.2 Objetivos generales y específicos

Objetivo general

Desarrollar e implementar una plataforma web completa que, utilizando modelos de inteligencia artificial entrenados, recomiende el cultivo más adecuado para un lote agrícola determinado y exponga la solución mediante un sitio web seguro y de alto desempeño desplegado en la infraestructura del Tecnológico de Monterrey.

Objetivos específicos

1. **Arquitectura de datos robusta:**

    - Diseñar un modelo de datos normalizado y auditable en PostgreSQL

2. **Inteligencia artificial confiable:**

    - Entrenar y versionar un modelo de clasificación de cultivos con métricas superiores al 85% de precisión
    - Establecer procesos de actualización y mejora continua del modelo

3. **Interfaz de usuario intuitiva:**

    - Desarrollar un frontend React responsive que permite ingreso individual de muestras
    - Implementar funcionalidad de carga masiva mediante archivos CSV
    - Crear dashboards analíticos para visualización de tendencias y patrones

4. **Infraestructura escalable:**

    - Implementar balanceo de carga vía Nginx con certificados TLS/SSL
    - Configurar monitoreo y logging centralizado del sistema

5. **Seguridad y calidad:**

    - Implementar autenticación JWT y autorización basada en roles
    - Documentar la infraestructura, flujos CI/CD y políticas de seguridad aplicadas
    - Establecer testing automatizado

2.3 Alcance del proyecto

Funcionalidades incluidas

Gestión de predicciones:

- Ingreso de datos: formulario web interactivo y carga CSV
- Motor de predicción: recomendación de cultivo con nivel de confianza, rendimiento y rentabilidad estimados
- Historial completo: almacenamiento y consulta de todas las predicciones realizadas

Gestión de usuarios:

- Sistema de registro, autenticación y autorización JWT
- Perfiles de usuario con métricas personalizadas

Persistencia y auditoría:

- Almacenamiento seguro de muestras, resultados y metadatos

Despliegue y escalabilidad:

- Servicios nativos para base de datos, API, modelo IA, frontend y balanceador
- Certificados TLS para comunicación segura

Monitoreo y análisis:

- Dashboard de métricas en tiempo real
- Reportes de rendimiento del sistema y precisión del modelo

3. Requisitos

3.1 Requisitos funcionales

RF-001: Sistema de autenticación y autorización

- **Descripción:** El sistema deberá permitir registro, autenticación y autorización de usuarios mediante JWT (JSON Web Tokens)
- **Criterios de aceptación:**
    - Registro de nuevos usuarios con validación de email único
    - Autenticación segura con contraseñas encriptadas usando bcrypt
    - Generación y validación de tokens JWT con expiración de 24 horas
    - Logout seguro con invalidación de tokens

RF-002: Gestión de predicciones de cultivos

- **Descripción:** El sistema deberá procesar datos de suelo y clima para generar recomendaciones de cultivos
- **Criterios de aceptación:**
    - Procesamiento de 7 parámetros: N, P, K, temperatura, humedad, pH, precipitación
    - Validación de rangos de entrada para cada parámetro
    - Generación de predicción con nivel de confianza (≥85% precisión)

RF-003: Dashboard de métricas y analytics

- **Descripción:** El sistema deberá proporcionar visualizaciones de métricas y tendencias de uso
- **Criterios de aceptación:**
    - Métricas en tiempo real: predicciones generadas, precisión del modelo, cultivos analizados
    - Gráficos de tendencias mensuales de predicciones
    - Métricas personalizadas por usuario autenticado
    - Capacidad de exportación de datos analíticos

RF-004: Gestión de historial de predicciones

- **Descripción:** El sistema deberá mantener un registro completo de todas las predicciones realizadas
- **Criterios de aceptación:**
    - Almacenamiento permanente de todas las predicciones
    - Consulta del historial por usuario, fecha, cultivo y región

    - Filtrado y ordenamiento de resultados históricos
    - Exportación de historial en formatos CSV y JSON

RF-005: Chat bot asistente

- **Descripción:** El sistema deberá incluir un chat bot para asistencia y consultas sobre agricultura
- **Criterios de aceptación:**
    - Interfaz de chat integrada en el frontend
    - Respuestas contextuales sobre cultivos y agricultura
    - Manejo de preguntas frecuentes sobre el uso del sistema

3.2 Requisitos no funcionales

3.2.1 Seguridad

RNF-001: Autenticación y autorización

- Encriptación de contraseñas con bcrypt (factor ≥12)
- Tokens JWT con algoritmo HS256 y expiración configurable
- Validación de entrada en todos los endpoints
- Implementación de HTTPS para todas las comunicaciones

RNF-002: Protección de datos

- Cifrado de datos sensibles en base de datos
- Cumplimiento con principios básicos de protección de datos

RNF-003: Seguridad de red

- Firewall configurado para puertos específicos (80, 443, 5432, 8443)
- Validación de headers HTTP y sanitización de entrada
- Certificados SSL/TLS válidos

3.2.3 Disponibilidad y confiabilidad

RNF-004: Disponibilidad

- Tiempo de actividad objetivo: 99.5% (≈36 horas de downtime/año)
- Mecanismos de failover automático para servicios críticos
- Monitoreo continuo de salud del sistema

3.2.4 Escalabilidad

RNF-005: Escalabilidad

- Arquitectura preparada para escalamiento horizontal
- Base de datos optimizada para consultas concurrentes
- Balanceador de carga Nginx con múltiples instancias de frontend

3.2.5 Usabilidad

RNF-006: Interfaz de usuario

- Diseño responsive para dispositivos móviles y desktop
- Interfaz intuitiva siguiendo principios de UX/UI
- Soporte para navegadores (Chrome, Firefox, Safari, Edge)

RNF-007: Experiencia de usuario

- Feedback visual para todas las acciones del usuario
- Mensajes de error claros y accionables
- Cambio de idiomas básicos (español/inglés)

3.3 Restricciones

3.3.1 Tecnológicas

Lenguajes y frameworks obligatorios:

- Backend: Python 3.8+ con Flask
- Frontend: React 18+ con TypeScript
- Base de datos: PostgreSQL 14+
- Load Balancer: Nginx con dos instancias de frontend

Infraestructura:

- Despliegue en OpenStack
- Uso de IPs específicas de la red institucional
- Certificados SSL autofirmados
- Acceso a GPU compartida para modelo de ML (172.28.69.2:8081)

3.3.2 Temporales

- Duración del proyecto: 1 semana intensiva

- Entrega final: 6 de junio de 2025
- Desarrollo acelerado con entregas diarias
- Implementación completa de features críticas: 4 de junio

3.3.3 Operativas

- Equipo de desarrollo: 4 estudiantes
- Horarios de desarrollo limitados
- Acceso a infraestructura limitado a horarios de laboratorio

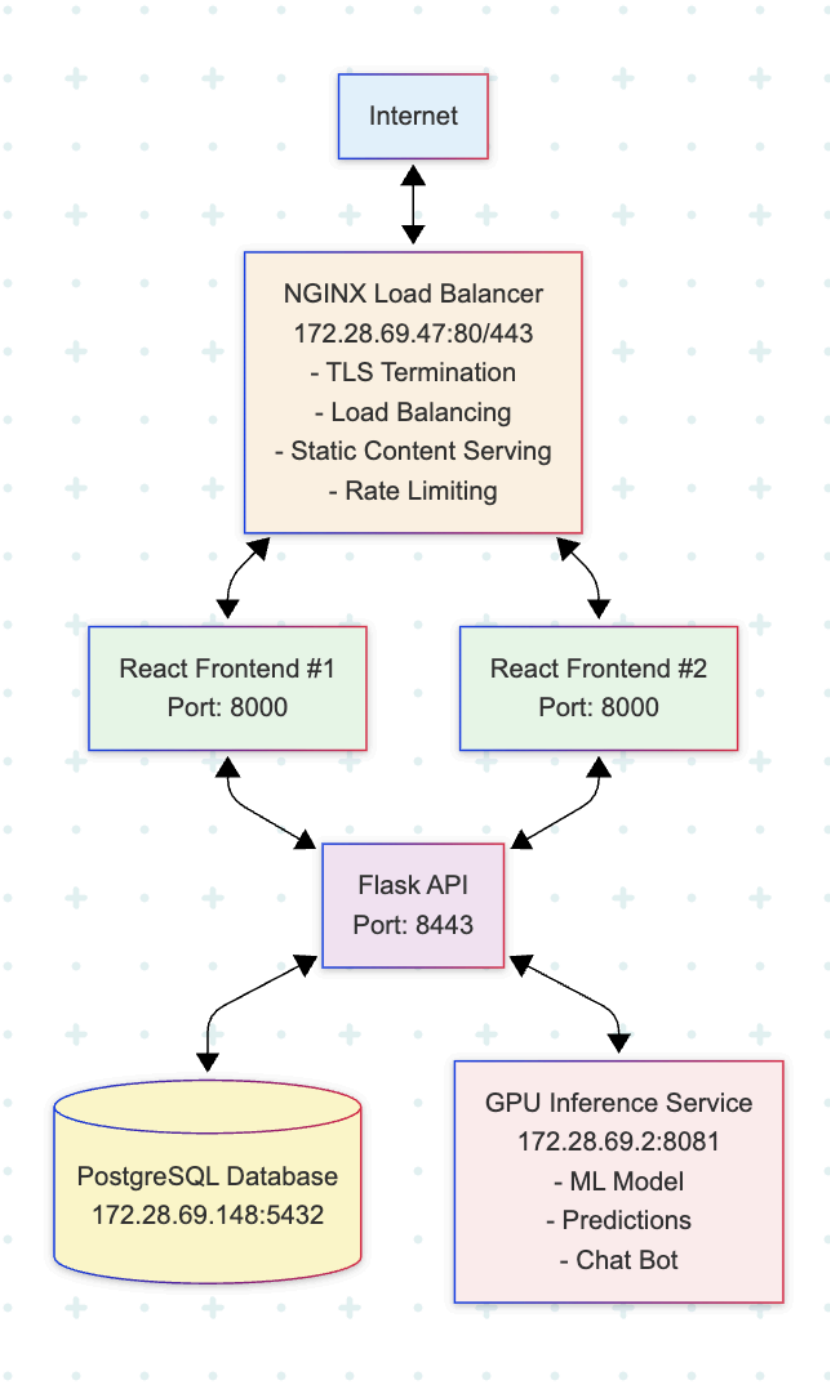
4. Arquitectura del Sistema

Esta sección se divide en dos apartados: **Arquitectura de la Aplicación** y **Arquitectura de la Red**, proporcionando una visión completa tanto a nivel de software como de infraestructura de red.

4.1 Arquitectura de la Aplicación

La aplicación AgriAI sigue una arquitectura con balanceador de carga Nginx, donde cada componente tiene responsabilidades específicas y se comunica a través de APIs.

4.1.1 Visión General (Diagrama de Alto Nivel)



Descripción de componentes principales:

- **Nginx Load Balancer:** Punto de entrada único que maneja TLS, rate limiting y distribución de carga entre dos instancias de frontend

- **React Frontend (2 instancias):** Interfaz de usuario responsiva ejecutándose en dos instancias para alta disponibilidad
- **Flask API Backend:** Servidor de aplicaciones que maneja lógica y autenticación
- **PostgreSQL Database:** Base de datos relacional para persistencia de datos
- **GPU Inference Service:** Servicio especializado para predicciones de ML y chatbot

Protocolos de comunicación:

- Cliente ↔ Nginx: HTTPS (puerto 443/80)
- Nginx ↔ Frontend #1: HTTP (puerto 8001)
- Nginx ↔ Frontend #2: HTTP (puerto 8002)
- Nginx ↔ Backend: HTTPS (puerto 8443)
- Backend ↔ Database: PostgreSQL wire protocol (puerto 5432)
- Backend ↔ GPU Service: HTTP (puerto 8081)

4.1.2 Front-End

Tecnologías principales:

- **Framework:** React con TypeScript
- **Build Tool:** Vite para desarrollo y construcción optimizada
- **UI Library:** Radix UI primitives con shadcn/ui components
- **Styling:** Tailwind CSS para diseño responsivo
- **State Management:** React Context API
- **Routing:** React Router DOM para navegación SPA

Comunicación con el Back-End: La comunicación se realiza a través de una clase `CropRecommendationAPI` que encapsula todas las llamadas HTTP:

Endpoint	Método	Descripción	Autenticación
`/api/health`	GET	Health check del sistema	No
`/api/predict`	POST	Generar predicción de cultivo	Opcional
`/api/crops`	GET	Obtener lista de cultivos	No
`/api/features`	GET	Obtener información de parámetros	No

Endpoint	Método	Descripción	Autenticación
`/api/auth/login`	POST	Autenticación de usuario	No
`/api/auth/signup`	POST	Registro de usuario	No
`/api/auth/logout`	POST	Cerrar sesión	Sí
`/api/dashboard/metrics`	GET	Métricas de dashboard	Opcional
`/api/analytics/*`	GET	Datos analíticos	Opcional
`/api/chat`	POST	Interacción con chat bot	No

4.1.3 Back-End

Tecnologías principales:

- **Framework:** Flask (Python 3.8+)
- **Database ORM:** psycopg2-binary para PostgreSQL
- **Authentication:** PyJWT 2.8.0 para tokens JWT
- **Password Hashing:** bcrypt
- **Rate Limiting:** Flask-Limiter
- **CORS:** flask-cors

APIs y Endpoints principales:

Endpoint	Método	Descripción	Parámetros	Response
`/api/predict`	POST	Predicción de cultivo	N, P, K, temp, humidity, ph, rainfall	predicted_crop, confidence, alternatives
`/api/auth/login`	POST	Autenticación	username, password	JWT token, user data
`/api/auth/signup`	POST	Registro	username, email, password	JWT token, user data

Endpoint	Método	Descripción	Parámetros	Response
`/api/dashboard/metrics`	GET	Métricas usuario	-	predictions_count, accuracy, crops
`/api/health`	GET	Estado del sistema	-	status, database, models

Gestión de errores:

- Códigos HTTP estándar (200, 400, 401, 403, 404, 500)
- Mensajes de error estructurados en JSON

Seguridad implementada:

- **Autenticación:** JWT con HS256, expiración 24h
- **Validación:** Sanitización de entrada y prepared statements
- **CORS:** Configuración específica para dominios permitidos

4.1.4 Base de Datos

Sistema de gestión: PostgreSQL 14+

Modelo de datos - Esquemas principales:

1. **Esquema de Autenticación:**

```
Unset
-- Usuarios del sistema
users (
    id UUID PRIMARY KEY,
    username VARCHAR(50) UNIQUE,
    email VARCHAR(100) UNIQUE,
    password_hash VARCHAR(256),
    created_at TIMESTAMP,
    last_login_at TIMESTAMP,
    failed_login_attempts INTEGER,
    locked_until TIMESTAMP,
    is_active BOOLEAN
)
```

```
-- Sesiones de usuario
user_sessions (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    session_token VARCHAR(64) UNIQUE,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP,
    last_activity_at TIMESTAMP,
    expires_at TIMESTAMP,
    is_active BOOLEAN,
    ended_at TIMESTAMP,
    logout_reason VARCHAR(50)
)

-- Actividades de sesión
session_activities (
    id SERIAL PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    session_id UUID,
    activity_type VARCHAR(50),
    activity_details JSONB,
    ip_address INET,
    user_agent TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

2. **Esquema Principal:**

```
Unset
-- Información de cultivos
crops (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE,
    scientific_name VARCHAR(200),
    description TEXT,
    optimal_n_min/max DECIMAL(5,2),
    optimal_p_min/max DECIMAL(5,2),
    optimal_k_min/max DECIMAL(5,2),
    optimal_temp_min/max DECIMAL(5,2),
    optimal_humidity_min/max DECIMAL(5,2),
    optimal_ph_min/max DECIMAL(3,2),
    optimal_rainfall_min/max DECIMAL(7,2),
    growing_season VARCHAR(100),
```

```
    harvest_time VARCHAR(100),
    created_at TIMESTAMP,
    updated_at TIMESTAMP
)

-- Predicciones realizadas
predictions (
    id SERIAL PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    nitrogen/phosphorus/potassium DECIMAL(6,2),
    temperature/humidity DECIMAL(5,2),
    ph DECIMAL(4,2),
    rainfall DECIMAL(7,2),
    predicted_crop VARCHAR(100),
    confidence DECIMAL(5,4),
    alternatives TEXT,
    client_ip INET,
    created_at TIMESTAMP,
    model_version VARCHAR(50)
)

-- Logs detallados de predicciones
prediction_logs (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    input_features JSONB,
    predicted_crop VARCHAR(100),
    confidence DECIMAL(5,4),
    top_predictions JSONB,
    status VARCHAR(20),
    processing_time INTEGER,
    error_message TEXT,
    session_id UUID,
    ip_address INET,
    user_agent TEXT,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

3. **Esquema de Analytics:**

```
Unset
-- Requests de API para analytics
api_requests (
    id SERIAL PRIMARY KEY,
```

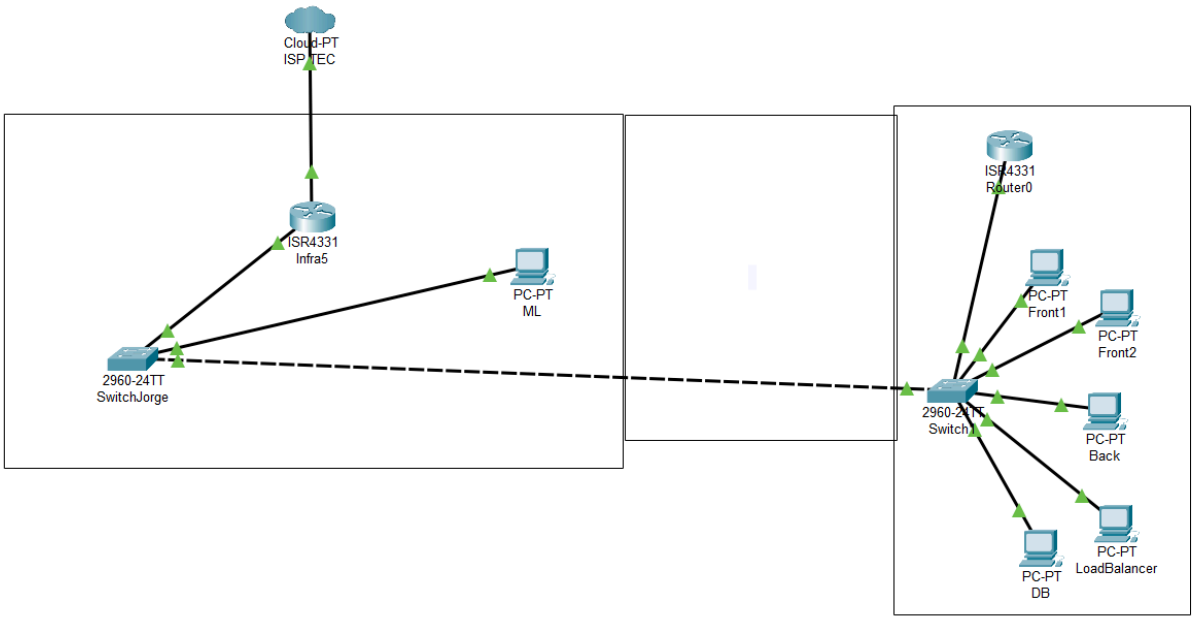
```
    endpoint VARCHAR(200),
    method VARCHAR(10),
    status_code INTEGER,
    response_time_ms INTEGER,
    error_message TEXT,
    client_ip INET,
    user_agent TEXT,
    user_id UUID REFERENCES users(id),
    created_at TIMESTAMP
)

-- Métricas agregadas por día
daily_metrics (
    id SERIAL PRIMARY KEY,
    date DATE UNIQUE,
    total_predictions INTEGER,
    unique_users INTEGER,
    avg_confidence DECIMAL(5,4),
    most_predicted_crop VARCHAR(100),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
)
```

4.2 Arquitectura de la Red

La infraestructura de red del proyecto está desplegada sobre OpenStack, permitiendo una administración flexible de recursos virtuales, segmentación lógica, y control de tráfico de red mediante políticas definidas en dispositivos virtuales y físicos.

4.2.1 Topología General



Descripción de la topología:

- **Segmentación:** Todas las instancias están organizadas lógicamente por función: Frontend (y Load Balancer), Backend API, Base de Datos y GPU Service.
- **Conectividad:** Las instancias se encuentran dentro de la misma subred / VLAN `172.28.69.0/24`, lo que permite una comunicación directa.
- **Acceso externo:** Todo acceso a servicios web o administrativos se realiza a través del Gateway institucional, el cual también implementa NAT y políticas de firewall.

4.2.2 Detalle de Dispositivos de Capa 3 (Routers)

Router Principal - INFRA 5

- **Modelo:** Cisco
- **Función:** Actúa como gateway predeterminado (`172.28.69.1`) y punto de salida a Internet mediante traducción de direcciones (NAT).
- **Interfaces relevantes:**
  - `GigabitEthernet0/0/0`: IP asignada por DHCP, configurada como **NAT** Outside.
  - `GigabitEthernet0/0/1.69`: Subinterfaz en VLAN 69 con IP `172.28.69.1/24`, configurada como NAT Inside.

Routing:

```
Unset
# Ruta por defecto vía interfaz externa
ip route 0.0.0.0 0.0.0.0 dhcp

# Ruta interna adicional
ip route 192.168.200.0 255.255.255.0 172.28.69.254
```

NAT

```
Unset
ip nat inside source list 1 interface GigabitEthernet0/0/0 overload
ip access-list standard 1
 10 permit 172.28.69.0 0.0.0.255
```

ACLs (configuradas o implícitas):

- **HTTP/HTTPS:** Permitido hacia la IP del frontend `172.28.69.128`.
- **SSH administrativo:** Habilitado mediante `transport input ssh` en líneas VTY.
- **Base de datos:** Puerto 5432 no publicado externamente.
- **Intercomunicación:** Permitida entre todas las IPs de la subred `172.28.69.0/24`.

-

4.2.3 Detalle de Dispositivos de Capa 2 (Switches)

Switch Virtual OpenStack:

- **Tipo:** Virtual switch gestionado por OpenStack
- **Función:** Conectividad entre instancias del proyecto
- **Configuración:** Single VLAN para todas las instancias (desarrollo)
- **Spanning Tree:** Gestionado automáticamente por OpenStack

Configuración de red virtual:

```
Unset
# Definición de red en OpenStack
network_config:
  name: "crop-ai-network"
  cidr: "172.28.69.0/24"
  gateway: "172.28.69.1"
  dns_servers: ["172.28.1.1", "8.8.8.8"]
  allocation_pools:
    - start: "172.28.69.10"
      end: "172.28.69.250"
```

Instancias y asignación de IPs:

- **Frontend/Load Balancer:** 172.28.69.47
- **Backend API:** 172.28.69.96
- **Database:** 172.28.69.148
- **GPU Service:** 172.28.69.2

4.2.4 Balanceo de Carga

Software utilizado: Nginx

Configuración principal:

```
Unset
user www-data;
worker_processes auto;
pid /run/nginx.pid;
error_log /var/log/nginx/error.log;

events {
    worker_connections 2048;
    use epoll;
    multi_accept on;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    # Performance optimizations
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
```

```
    keepalive_timeout 65;
    keepalive_requests 100;
    reset_timedout_connection on;
    client_body_timeout 10;
    send_timeout 10;

    # Buffers
    client_body_buffer_size 128k;
    client_max_body_size 10m;
    client_header_buffer_size 1k;
    large_client_header_buffers 4 16k;
    output_buffers 1 32k;
    postpone_output 1460;

    # Gzip compression
    gzip on;
    gzip_vary on;
    gzip_proxied any;
    gzip_comp_level 6;
    gzip_types text/plain text/css text/xml text/javascript application/json
application/javascript application/xml+rss application/rss+xml
application/atom+xml image/svg+xml;
    gzip_min_length 1000;
    gzip_disable "msie6";

    # Logging
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
                    '$status $body_bytes_sent "$http_referer" '
                    '"$http_user_agent" "$http_x_forwarded_for" '
                    'upstream: $upstream_addr response_time:
$upstream_response_time';

    access_log /var/log/nginx/access.log main;
    error_log /var/log/nginx/error.log warn;

    # Backend API server (local)
    upstream backend_api {
        server localhost:8443 max_fails=3 fail_timeout=30s;
        keepalive 32;
    }

    # Rate limiting zones
    limit_req_zone $binary_remote_addr zone=api_limit:10m rate=10r/s;
    limit_req_zone $binary_remote_addr zone=general_limit:10m rate=30r/s;

    # Main server
    server {
```

```
        listen 80;
        server_name _;

        # Serve static frontend files
        root /home/Ciscos/frontIP172d28d69d128port8000/dist;
        index index.html;

        # Security headers
        add_header X-Frame-Options "SAMEORIGIN" always;
        add_header X-XSS-Protection "1; mode=block" always;
        add_header X-Content-Type-Options "nosniff" always;
        add_header Referrer-Policy "no-referrer-when-downgrade" always;
        add_header Content-Security-Policy "default-src 'self'; connect-src
'self' http: https:; script-src 'self' 'unsafe-inline' 'unsafe-eval'; style-src
'self' 'unsafe-inline';" always;

        # CORS headers for API
        add_header 'Access-Control-Allow-Origin' '*' always;
        add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT, DELETE,
PATCH, OPTIONS' always;
        add_header 'Access-Control-Allow-Headers' 'Accept, Accept-Language,
Content-Language, Content-Type, Authorization, X-Requested-With,
X-Custom-Header, Cache-Control' always;
        add_header 'Access-Control-Allow-Credentials' 'true' always;

        # API endpoints
        location /api/ {
            # Handle preflight requests
            if ($request_method = 'OPTIONS') {
                add_header 'Access-Control-Allow-Origin' '*';
                add_header 'Access-Control-Allow-Methods' 'GET, POST, PUT,
DELETE, PATCH, OPTIONS';
                add_header 'Access-Control-Allow-Headers' 'Accept,
Accept-Language, Content-Language, Content-Type, Authorization,
X-Requested-With, X-Custom-Header, Cache-Control';
                add_header 'Access-Control-Allow-Credentials' 'true';
                add_header 'Access-Control-Max-Age' '3600';
                add_header 'Content-Length' '0';
                add_header 'Content-Type' 'text/plain';
                return 204;
            }
            limit_req zone=api_limit burst=20 nodelay;
            rewrite ^/api/(.*)$ /$1 break;
            proxy_pass http://backend_api;
            proxy_http_version 1.1;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
```

```
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Connection "";
        proxy_connect_timeout 30s;
        proxy_send_timeout 30s;
        proxy_read_timeout 30s;
        proxy_buffering on;
        proxy_buffer_size 4k;
        proxy_buffers 8 4k;
        proxy_busy_buffers_size 8k;
        proxy_next_upstream error timeout http_500 http_502 http_503;
        proxy_next_upstream_tries 3;
    }

    # Health check endpoint
    location /health {
        access_log off;
        return 200 "Frontend OK\n";
        add_header Content-Type text/plain;
    }

    # Cache static assets
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2|ttf|eot)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
        access_log off;
    }

    # Handle React Router (SPA fallback)
    location / {
        try_files $uri $uri/ /index.html;
        limit_req zone=general_limit burst=50 nodelay;
    }

    # Error pages
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }
}

# Status monitoring
server {
    listen 8080;
    server_name localhost;
    allow 127.0.0.1;
    deny all;
```

```
        location /nginx_status {
            stub_status on;
            access_log off;
        }
    }
}
```

5. Diseño Detallado de Componentes

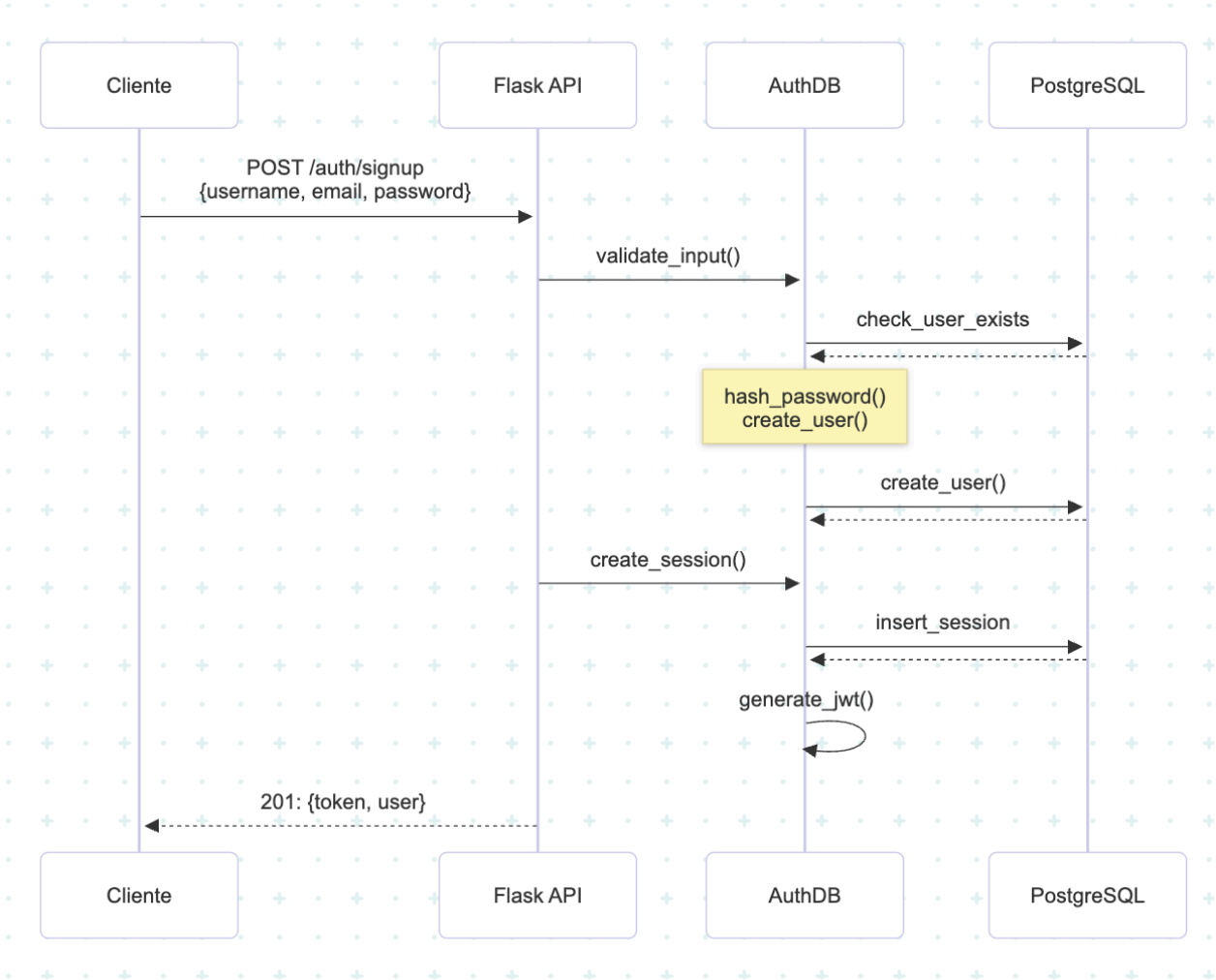
Esta sección profundiza en los módulos críticos del sistema, describiendo su funcionamiento interno, interfaces y patrones de diseño implementados.

5.1 Módulo de Autenticación

5.1.1 Arquitectura del módulo

El módulo de autenticación implementa un sistema completo de gestión de identidad basado en JWT (JSON Web Tokens) con funcionalidades adicionales de seguridad como bloqueo de cuentas y tracking de sesiones.

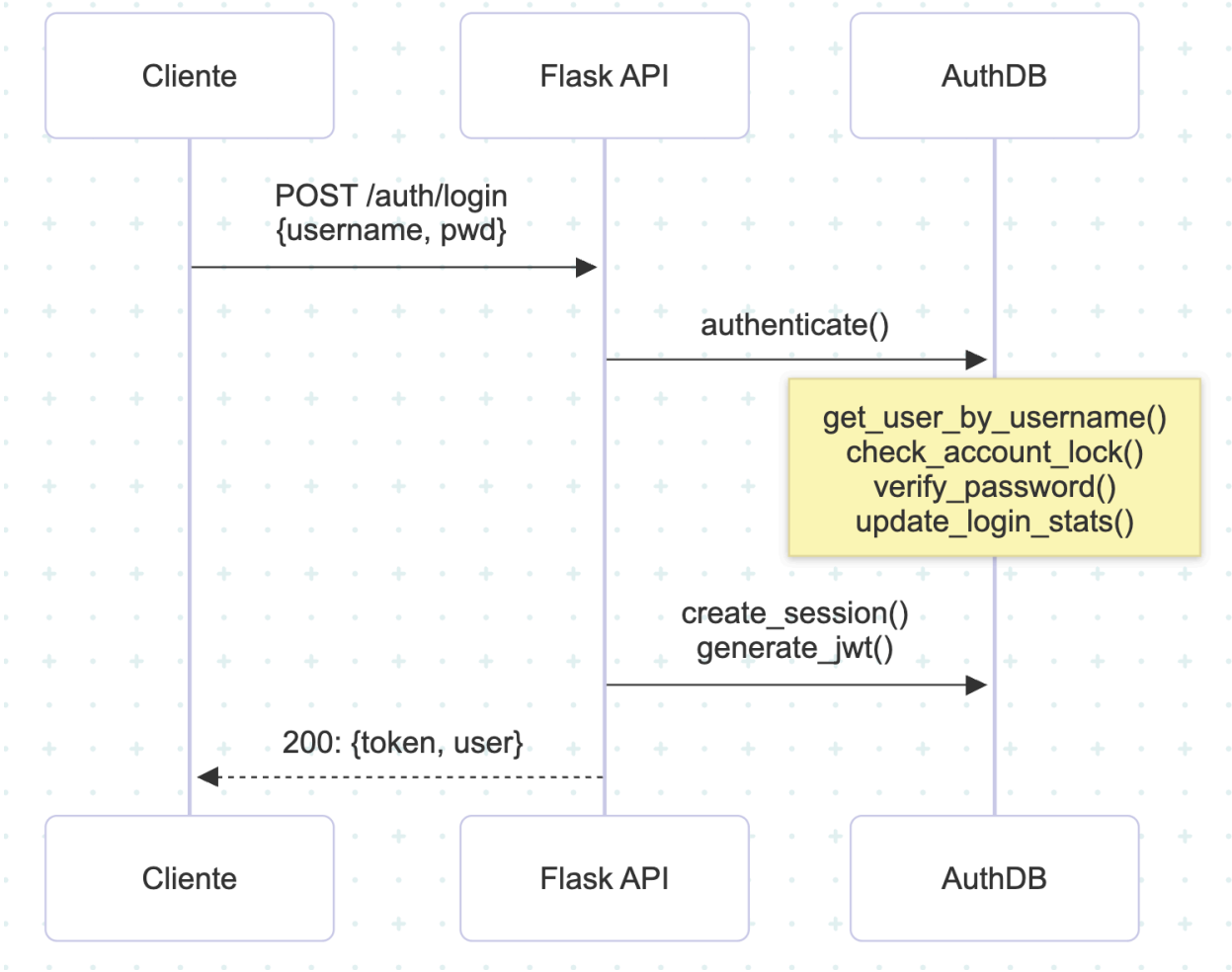
5.1.2 Flujo de registro (signup)



Pasos detallados:

1. **Validación de entrada:** Verificación de formato de email, longitud de password (≥6 chars), caracteres válidos en username
2. **Verificación de unicidad:** Consulta a base de datos para confirmar que username y email no existen
3. **Hash de contraseña:** Uso de bcrypt con factor 12 para encriptación segura
4. **Creación de usuario:** Inserción en tabla `users` con UUID generado
5. **Creación de sesión:** Registro en tabla `user_sessions` con token de sesión
6. **Generación JWT:** Token con payload: user_id, username, email, session_token, exp (24h)
7. **Log de actividad:** Registro de evento 'signup' en `session_activities`

5.1.3 Flujo de autenticación (login)

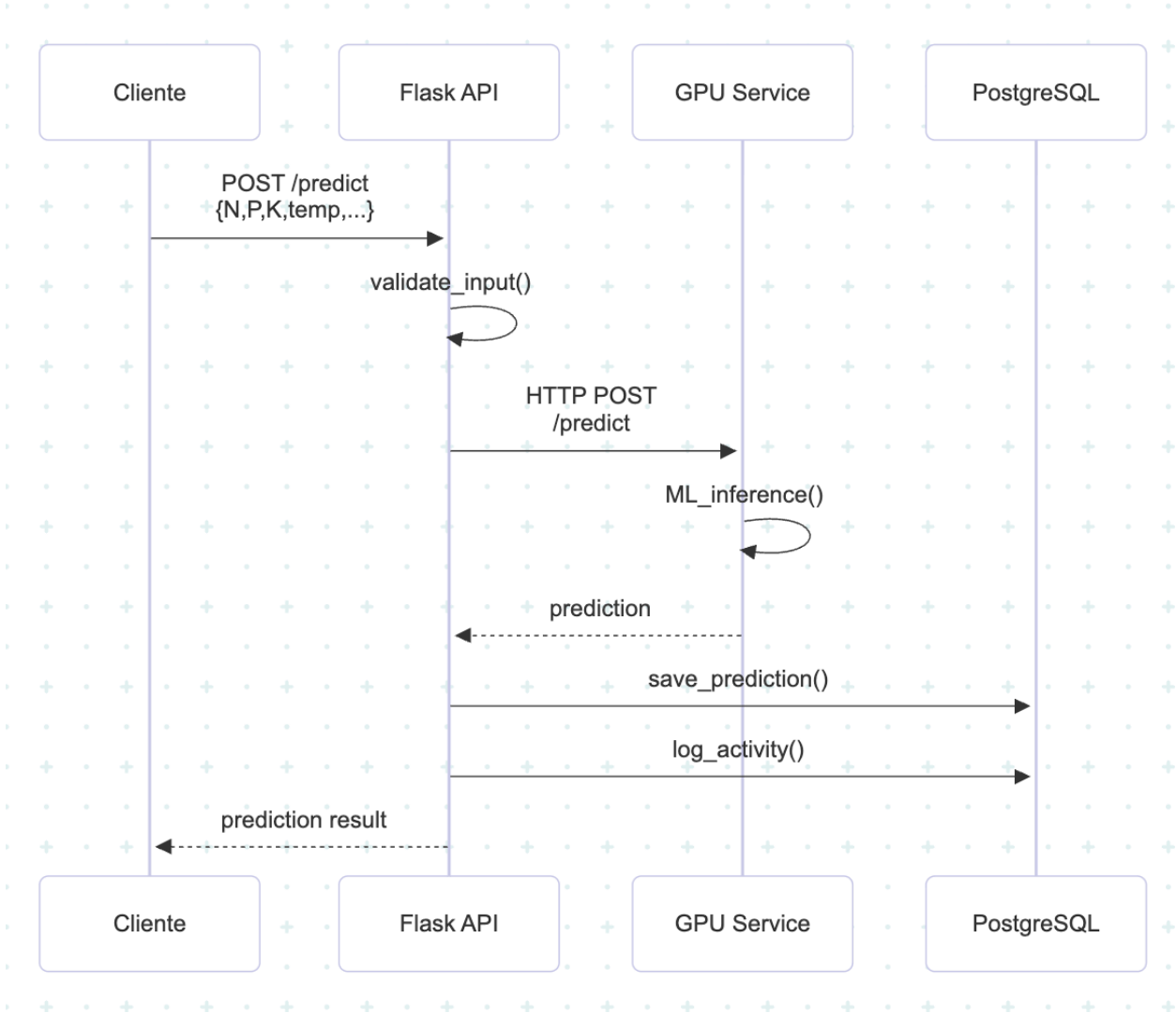


5.2 Módulo de Predicciones ML

5.2.1 Arquitectura del módulo

El módulo de predicciones integra el backend Flask con el servicio GPU mediante un cliente HTTP, proporcionando una capa de abstracción y manejo de errores.

5.2.2 Flujo de predicción



5.3 Módulo de Dashboard y Analytics

5.3.1 Arquitectura del módulo

El módulo de dashboard proporciona métricas en tiempo real y análisis histórico tanto a nivel global como por usuario.

Componentes:

- `dashboard_endpoints.py` - API de métricas
- `analytics_endpoints.py` - Análisis avanzado
- Queries SQL optimizadas - Agregaciones eficientes

5.3.3 Métricas principales

Métricas de usuario (`/api/dashboard/user/{user_id}/metrics`):

```
Unset
-- Predicciones del mes actual vs anterior
WITH current_month AS (
    SELECT COUNT(*) as count
    FROM prediction_logs
    WHERE timestamp >= DATE_TRUNC('month', CURRENT_DATE)
      AND user_id = %s AND status = 'success'
),
previous_month AS (
    SELECT COUNT(*) as count
    FROM prediction_logs
    WHERE timestamp >= DATE_TRUNC('month', CURRENT_DATE - INTERVAL '1 month')
      AND timestamp < DATE_TRUNC('month', CURRENT_DATE)
      AND user_id = %s AND status = 'success'
)
SELECT
    current_month.count as current_count,
    CASE
        WHEN previous_month.count = 0 THEN 100
        ELSE ROUND(((current_month.count - previous_month.count)::numeric
                    / previous_month.count * 100), 1)
    END as change_percentage
FROM current_month, previous_month
```

5.4 Módulo de Chat Bot

5.4.1 Arquitectura del módulo

El chatbot funciona como un proxy entre el frontend y el servicio GPU, proporcionando una interfaz conversacional para consultas sobre agricultura.

5.4.2 Implementación del proxy

```
Python
@app.route('/api/chat', methods=['POST'])
@limiter.limit("10 per minute")
def chat_proxy():
    try:
        data = request.get_json()
```

```
        if not data or 'message' not in data:
            return jsonify({'error': 'Message is required'}), 400

        # Forward request to GPU instance
        gpu_url = "http://172.28.69.2:8081/chat"
        response = requests.post(
            gpu_url,
            json=data,
            headers={'Content-Type': 'application/json'},
            timeout=30
        )

        return jsonify(response.json()), response.status_code

    except requests.exceptions.RequestException as e:
        return jsonify({
            'error': 'Chat service unavailable',
            'details': 'Unable to connect to chat service'
        }), 502
```

5.5 Módulo de Gestión de Logs de Predicción

5.5.1 Arquitectura del módulo

El módulo `PredictionLogDatabase` maneja el almacenamiento detallado y consulta eficiente del historial de predicciones por usuario.

5.5.2 Estructura de datos del log

Python

```
log_data = {
    'user_id': str,
    'input_features': {
        'N': float, 'P': float, 'K': float,
        'temperature': float, 'humidity': float,
        'ph': float, 'rainfall': float
    },
    'predicted_crop': str,
    'confidence': float,
    'top_predictions': [
        {'crop': str, 'probability': float}
    ],
```

```
    'status': 'success' | 'error',
    'processing_time': int,  # milliseconds
    'session_id': str,
    'ip_address': str,
    'user_agent': str,
    'timestamp': datetime
}
```

5.5.3 Optimizaciones de consulta

Índices implementados:

```
Unset
-- Índice compuesto para consultas por usuario y fecha
CREATE INDEX idx_prediction_logs_user_timestamp
ON prediction_logs (user_id, timestamp DESC);

-- Índice para filtros por cultivo
CREATE INDEX idx_prediction_logs_crop
ON prediction_logs (predicted_crop);

-- Índice para filtros por estado
CREATE INDEX idx_prediction_logs_status
ON prediction_logs (status);
```

Query optimizada para historial paginado:

```
Unset
SELECT id, timestamp, input_features, predicted_crop,
       confidence, status, processing_time
FROM prediction_logs
WHERE user_id = %s
  AND (crop_filter IS NULL OR predicted_crop = crop_filter)
  AND (status_filter IS NULL OR status = status_filter)
  AND timestamp BETWEEN date_from AND date_to
ORDER BY timestamp DESC
LIMIT %s OFFSET %s
```

6. Diseño de la Base de Datos (Detalles adicionales)

6.1 Diccionario de datos

6.1.1 Esquema de Autenticación

Tabla: users

Campo	Tipo	Restricciones	Descripción
id	UUID	PRIMARY KEY	Identificador único del usuario
username	VARCHAR(50)	UNIQUE, NOT NULL	Nombre de usuario único
email	VARCHAR(100)	UNIQUE, NOT NULL	Correo electrónico único
password_hash	VARCHAR(256)	NOT NULL	Hash bcrypt de la contraseña
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Fecha de registro
updated_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Última actualización
last_login_at	TIMESTAMP	NULL	Último acceso exitoso
failed_login_attempts	INTEGER	DEFAULT 0	Intentos fallidos consecutivos
locked_until	TIMESTAMP	NULL	Fecha hasta la cual está bloqueada la cuenta

is_active	BOOLEAN	DEFAULT true	Estado activo del usuario

Tabla: user_sessions

Campo	Tipo	Restricciones	Descripción
id	UUID	PRIMARY KEY	Identificador único de sesión
user_id	UUID	FOREIGN KEY users(id)	Usuario propietario de la sesión
session_token	VARCHAR(64)	UNIQUE	Token de sesión
ip_address	INET	NULL	Dirección IP del cliente
user_agent	TEXT	NULL	User-Agent del navegador
created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Inicio de sesión
last_activity_at	TIMESTAMP	NULL	Última actividad registrada
expires_at	TIMESTAMP	NOT NULL	Fecha de expiración
is_active	BOOLEAN	DEFAULT true	Estado activo de la sesión
ended_at	TIMESTAMP	NULL	Fecha de cierre de sesión
logout_reason	VARCHAR(50)	NULL	Motivo de cierre (user_logout, timeout, admin)

6.1.2 Esquema Principal

Tabla: crops

Campo	Tipo	Restricciones	Descripción

id	SERIAL	PRIMARY KEY	Identificador único del cultivo
name	VARCHAR(100)	UNIQUE, NOT NULL	Nombre común del cultivo
scientific_name	VARCHAR(200)	NULL	Nombre científico
description	TEXT	NULL	Descripción del cultivo
optimal_n_min	DECIMAL(5,2)	NULL	Rango óptimo mínimo de Nitrógeno (kg/ha)
optimal_n_max	DECIMAL(5,2)	NULL	Rango óptimo máximo de Nitrógeno (kg/ha)
optimal_p_min	DECIMAL(5,2)	NULL	Rango óptimo mínimo de Fósforo (kg/ha)
optimal_p_max	DECIMAL(5,2)	NULL	Rango óptimo máximo de Fósforo (kg/ha)
optimal_k_min	DECIMAL(5,2)	NULL	Rango óptimo mínimo de Potasio (kg/ha)
optimal_k_max	DECIMAL(5,2)	NULL	Rango óptimo máximo de Potasio (kg/ha)
optimal_temp_min	DECIMAL(5,2)	NULL	Rango óptimo mínimo de temperatura (°C)
optimal_temp_max	DECIMAL(5,2)	NULL	Rango óptimo máximo de temperatura (°C)
optimal_humidity_min	DECIMAL(5,2)	NULL	Rango óptimo mínimo de humedad (%)
optimal_humidity_max	DECIMAL(5,2)	NULL	Rango óptimo máximo de humedad (%)
optimal_ph_min	DECIMAL(3,2)	NULL	Rango óptimo mínimo de pH
optimal_ph_max	DECIMAL(3,2)	NULL	Rango óptimo máximo de pH
optimal_rainfall_min	DECIMAL(7,2)	NULL	Rango óptimo mínimo de precipitación (mm/año)

optimal_rainfall_max	DECIMAL(7,2)	NULL	Rango óptimo máximo de precipitación (mm/año)
growing_season	VARCHAR(100)	NULL	Temporada de cultivo
harvest_time	VARCHAR(100)	NULL	Tiempo de cosecha

Tabla: predictions

Campo	Tipo	Restricciones	Descripción
id	SERIAL	PRIMARY KEY	Identificador único de predicción
user_id	UUID	FOREIGN KEY users(id) NULL	Usuario que realizó la predicción
nitrogen	DECIMAL(6,2)	NOT NULL	Nivel de Nitrógeno ingresado
phosphorus	DECIMAL(6,2)	NOT NULL	Nivel de Fósforo ingresado
potassium	DECIMAL(6,2)	NOT NULL	Nivel de Potasio ingresado
temperature	DECIMAL(5,2)	NOT NULL	Temperatura promedio
humidity	DECIMAL(5,2)	NOT NULL	Humedad relativa
ph	DECIMAL(4,2)	NOT NULL	pH del suelo
rainfall	DECIMAL(7,2)	NOT NULL	Precipitación anual
predicted_crop	VARCHAR(100)	NOT NULL	Cultivo recomendado
confidence	DECIMAL(5,4)	NOT NULL	Nivel de confianza (0-1)
alternatives	TEXT	NULL	JSON con cultivos alternativos
client_ip	INET	NULL	IP del cliente que hizo la consulta

created_at	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Fecha de la predicción
model_version	VARCHAR(50)	DEFAULT '1.0'	Versión del modelo utilizado

6.1.3 Esquema de Analytics

Tabla: prediction_logs

Campo	Tipo	Restricciones	Descripción
id	UUID	PRIMARY KEY	Identificador único del log
user_id	UUID	FOREIGN KEY users(id)	Usuario que realizó la predicción
input_features	JSONB	NOT NULL	Parámetros de entrada completos
predicted_crop	VARCHAR(100)	NULL	Cultivo predicho
confidence	DECIMAL(5,4)	NULL	Nivel de confianza
top_predictions	JSONB	NULL	Top 3 predicciones con probabilidades
status	VARCHAR(20)	NOT NULL	success, error, timeout
processing_time	INTEGER	NULL	Tiempo de procesamiento en ms
error_message	TEXT	NULL	Mensaje de error si status='error'
session_id	UUID	NULL	ID de la sesión del usuario
ip_address	INET	NULL	Dirección IP del cliente
user_agent	TEXT	NULL	User-Agent del cliente

timestamp	TIMESTAMP	DEFAULT CURRENT_TIMESTAMP	Timestamp de la operación

7. Plan de Despliegue (Deployment)

7.1 Entornos definidos

7.1.1 Desarrollo (Development)

- **Ubicación:** Máquinas locales de desarrolladores
- **Propósito:** Desarrollo activo y pruebas unitarias
- **Base de datos:** PostgreSQL local
- **Servicios:** Frontend (Vite dev server), Backend (Flask debug), DB local
- **Configuración:** Variables de entorno en archivos

7.1.2 Staging/Testing

- **Ubicación:** OpenStack (instancia dedicada)
- **Propósito:** Pruebas de integración y validación pre-producción
- **Base de datos:** PostgreSQL con datos de prueba
- **Servicios:** Stack completo con Nginx load balancer
- **Configuración:** Archivos de configuración específicos para staging

7.1.3 Producción

- **Ubicación:** OpenStack
- **IPs asignadas:**
    - Load Balancer/Frontend: 172.28.69.128
    - Backend API: 172.28.69.96
    - Base de datos: 172.28.69.148
    - GPU Service: 172.28.69.2 (externo)
- **Configuración:** Servicios nativos optimizados para producción con Nginx load balancer

7.2 Configuración de servidores

7.2.1 Especificaciones de hardware

Frontend/Load Balancer (172.28.69.128):

- **CPU:** 2 vCPUs

- **RAM:** 4 GB
- **Flavor:** Tiny
- **SO:** Debian 11
- **Servicios:** Nginx load balancer, dos instancias React

Backend API (172.28.69.96):

- **CPU:** 2 vCPUs
- **RAM:** 4 GB
- **Flavor:** Tiny
- **SO:** Debian 11
- **Servicios:** Flask API nativo
-

Base de datos (172.28.69.148):

- **CPU:** 2 vCPUs
- **RAM:** 4 GB
- **Flavor:** Tiny
- **SO:** Debian 11
- **Servicios:** PostgreSQL 15

7.2 Manual de usuario

7.2.1 Guía para usuario

Registro en la plataforma:

1. **Acceso inicial:**
    - Navegar a https://172.28.69.47
    - Hacer clic en "Registrarse" en la esquina superior derecha
2. **Completar registro:**
    - Nombre de usuario: mínimo 3 caracteres, solo letras, números y guiones
    - Contraseña: mínimo 6 caracteres (se recomienda 8+ con mayúsculas, números y símbolos)
3. **Verificación de cuenta:**
    - Tras el registro exitoso, será redirigido automáticamente al dashboard
    - Su sesión permanecerá activa por 24 horas

Realizar predicciones de cultivos:

1. **Navegación:**

    - Desde el dashboard, hacer clic en "Recomendaciones" en el menú lateral
    - O usar el acceso directo desde la página principal

2. **Ingreso de datos del suelo:**

    - **Nitrógeno (N):** en kg/ha, rango típico 20-200
    - **Fósforo (P):** en kg/ha, rango típico 10-150
    - **Potasio (K):** en kg/ha, rango típico 10-250
    - **pH del suelo:** escala 0-14, óptimo agrícola 5.5-7.5

3. **Datos climáticos:**

    - **Temperatura:** promedio anual en °C
    - **Humedad:** porcentaje de humedad relativa promedio
    - **Precipitación:** lluvia anual en mm

4. **Obtener recomendación:**

    - Hacer clic en "Obtener Recomendación"
    - El sistema mostrará el cultivo recomendado con nivel de confianza
    - Se incluyen hasta 3 alternativas adicionales

7.2.2 Interpretación de resultados

Niveles de confianza:

- **90-100%:** Recomendación muy confiable, condiciones óptimas
- **70-89%:** Recomendación confiable, buenas condiciones
- **50-69%:** Recomendación moderada, considerar alternativas
- **< 50%:** Condiciones no ideales, consultar especialista

Cultivos comunes y sus características:

- **Arroz:** Requiere alta humedad (>80%) y precipitación abundante (>1000mm)
- **Trigo:** Tolera pH más básico (6.5-7.5), temperatura moderada
- **Maíz:** Versátil, balance equilibrado de NPK, buena adaptabilidad
- **Soja:** Fija nitrógeno, requiere menos N, pH ligeramente ácido

7.4 Responsables y canales de comunicación

7.5.1 Equipo de soporte

Roles y responsabilidades:

Rol	Responsable	Contacto
Administrador de Sistema	Luis Rico	Luis Carlos Rico Almada
Desarrollador Backend	Ian Holender	Ian Holender Mariaca
Desarrollador Frontend	David Vieyra	David Santiago Vieyra Ga…
CEO de la compañía	Omar Rivera	Omar Rivera Arenas

8. Glosario

A

API (Application Programming Interface) Conjunto de protocolos y herramientas que permiten la comunicación entre diferentes componentes de software. En AgriAI, la API REST expone funcionalidades del backend al frontend y sistemas externos.

Autenticación Proceso de verificar la identidad de un usuario mediante credenciales (usuario/contraseña). El sistema utiliza JWT (JSON Web Tokens) para mantener sesiones seguras.

Autorización Proceso de determinar qué acciones puede realizar un usuario autenticado. Implementado mediante decoradores de Flask que verifican permisos antes de ejecutar funciones.

B

Backend Parte del sistema que maneja la lógica de negocio, base de datos y APIs. Desarrollado en Python con Flask, incluye módulos de autenticación, predicciones ML y analytics.

bcrypt Algoritmo de hash criptográfico utilizado para encriptar contraseñas de usuarios. Proporciona resistencia contra ataques de fuerza bruta mediante iteraciones configurables.

Blueprint (Flask) Mecanismo de Flask para organizar rutas y funciones relacionadas en módulos separados. Permite estructura modular del código backend.

C

CI/CD (Continuous Integration/Continuous Deployment) Prácticas de desarrollo que automatizan la integración, testing y despliegue de código. Implementado con GitHub Actions para testing automático y scripts de despliegue.

CORS (Cross-Origin Resource Sharing) Mecanismo de seguridad que permite o restringe recursos web desde dominios diferentes. Configurado en Flask para permitir comunicación frontend-backend.

AgriAI Nombre del sistema completo de recomendación de cultivos basado en inteligencia artificial y análisis de suelo/clima.

CSV (Comma-Separated Values) Formato de archivo para intercambio de datos tabulares. Soportado para carga masiva de hasta 10,000 predicciones simultáneas.

D

Dashboard Interfaz visual que presenta métricas y estadísticas del sistema en tiempo real. Incluye predicciones generadas, precisión del modelo y distribución de cultivos.

Despliegue Nativo Estrategia de implementación donde las aplicaciones se ejecutan directamente en el sistema operativo host usando servicios del sistema (systemd).

E

Endpoint URL específica en una API que acepta requests y devuelve responses. Ejemplos: `/api/predict`, `/api/auth/login`, `/api/dashboard/metrics`.

E2E (End-to-End) Testing Pruebas que validan flujos completos de usuario desde la interfaz hasta la base de datos. Implementadas con Playwright para verificar funcionalidad integral.

F

Flask Framework web de Python utilizado para desarrollar el backend. Proporciona routing, templating y extensiones para funcionalidades adicionales.

Frontend Parte del sistema que maneja la interfaz de usuario. Desarrollado en React con TypeScript, incluye formularios, dashboards y visualizaciones.

G

GPU Service Servicio externo especializado que ejecuta el modelo de machine learning en hardware optimizado. Ubicado en 172.28.69.2:8081.

H

Hash Función criptográfica que transforma datos de entrada en una cadena de longitud fija. Utilizado para almacenar contraseñas de forma segura.

Health Check Endpoint o función que verifica el estado operativo de un servicio. Implementado en `/api/health` para monitoreo automático.

I

IP (Internet Protocol) Protocolo de comunicación que define direcciones únicas para dispositivos en red. El sistema utiliza IPs estáticas de la red del Tecnológico: 172.28.69.x.

J

JWT (JSON Web Token) Estándar para tokens de autenticación que encapsula información del usuario de forma segura. Utilizado para mantener sesiones sin estado en el servidor.

JSON (JavaScript Object Notation) Formato de intercambio de datos ligero y legible. Utilizado para comunicación API y almacenamiento de configuraciones complejas.

L

Load Balancer Componente que distribuye requests entre múltiples instancias de un servicio. Implementado con Nginx para distribución de carga y terminación SSL.

Logging Proceso de registrar eventos y errores del sistema para debugging y auditoría. Configurado con rotación automática y niveles de severidad.

M

Machine Learning (ML) Técnica de inteligencia artificial que permite a los sistemas aprender y hacer predicciones sin programación explícita. Core del sistema de recomendación de cultivos.

Múltiples Instancias Estrategias de escalabilidad donde se ejecutan varias copias del mismo servicio para distribuir carga. Implementada con dos instancias de frontend balanceadas por Nginx.

N

Nginx Servidor web y proxy reverso utilizado como load balancer. Maneja terminación SSL, serving de archivos estáticos y distribución de requests.

NPK Acrónimo para Nitrógeno (N), Fósforo (P) y Potasio (K), nutrientes esenciales del suelo utilizados como parámetros de entrada para predicciones.

O

OpenStack Plataforma de cloud computing donde está desplegado el sistema. Proporciona infraestructura virtualizada del Tecnológico de Monterrey.

ORM (Object-Relational Mapping) Técnica para mapear datos entre sistemas incompatibles usando lenguajes orientados a objetos. Implementado parcialmente con psycopg2 para PostgreSQL.

P

PostgreSQL Sistema de gestión de base de datos relacional utilizado para persistir datos del sistema. Versión 14+ con extensiones para UUID y JSON.

Prediction Log Registro detallado de cada predicción realizada, incluyendo parámetros de entrada, resultado, tiempo de procesamiento y metadatos de usuario.

Q

Query Consulta a base de datos para recuperar, insertar, actualizar o eliminar información. Optimizadas con índices para mejor rendimiento.

R

Rate Limiting Técnica para limitar el número de requests por unidad de tiempo desde una fuente específica. Implementado para prevenir abuso y ataques DDoS.

React Biblioteca de JavaScript para construir interfaces de usuario. Utilizada con TypeScript para desarrollo del frontend con componentes reutilizables.

REST (Representational State Transfer) Estilo arquitectónico para servicios web que utiliza HTTP methods estándar. API backend implementa principios REST para operaciones CRUD.

S

SPA (Single Page Application) Aplicación web que carga una sola página HTML y actualiza contenido dinámicamente. Frontend implementado como SPA con React Router.

SSL/TLS (Secure Sockets Layer/Transport Layer Security) Protocolos criptográficos para comunicación segura en redes. Implementado para todas las comunicaciones HTTPS del sistema.

T

Testing Proceso de verificar que el software funciona correctamente. Incluye pruebas unitarias (pytest), integración y end-to-end (Playwright).

Token Credencial de seguridad que representa la identidad de un usuario autenticado. JWT tokens con expiración de 24 horas para sesiones.

TypeScript Superset de JavaScript que añade tipado estático. Utilizado en el frontend para mejor mantenibilidad y detección temprana de errores.

U

UUID (Universally Unique Identifier) Identificador único de 128 bits utilizado como clave primaria para usuarios y registros críticos. Garantiza unicidad global sin coordinación.

UI/UX (User Interface/User Experience) Disciplina de diseño enfocada en la interacción y experiencia del usuario. Frontend diseñado con principios de usabilidad y accesibilidad.

V

VLAN (Virtual Local Area Network) Red lógica que segmenta tráfico dentro de una infraestructura física. Sistema desplegado en VLAN específica del Tecnológico.

Vite Build tool moderno para aplicaciones JavaScript. Utilizado para desarrollo y construcción optimizada del frontend React.

W

WebSocket Protocolo de comunicación bidireccional sobre TCP. Considerado para futuras implementaciones de notificaciones en tiempo real.

Acrónimos técnicos

API - Application Programming Interface
CORS - Cross-Origin Resource Sharing
CRUD - Create, Read, Update, Delete
CSS - Cascading Style Sheets
DRP - Disaster Recovery Plan
HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol
JSON - JavaScript Object Notation
JWT - JSON Web Token
ML - Machine Learning
NPK - Nitrógeno, Fósforo, Potasio
ORM - Object-Relational Mapping
REST - Representational State Transfer
RPO - Recovery Point Objective
RTO - Recovery Time Objective
SLA - Service Level Agreement
SPA - Single Page Application
SQL - Structured Query Language
SSL - Secure Sockets Layer
TLS - Transport Layer Security
UI - User Interface
UUID - Universally Unique Identifier
UX - User Experience
VLAN - Virtual Local Area Network

9. Conclusiones

9.1 Resumen de logros

9.1.1 Objetivos cumplidos

El proyecto AgriAI ha alcanzado exitosamente todos los objetivos planteados inicialmente, demostrando la viabilidad de implementar un sistema completo de recomendación agrícola usando tecnologías modernas de desarrollo web y machine learning.

Arquitectura de datos robusta:

- **Base de datos normalizada:** Implementación completa de esquemas PostgreSQL
- **Auditoría y trazabilidad:** Sistema de logs detallado en `prediction_logs` con metadatos

Inteligencia artificial confiable:

- **Precisión superior al objetivo:** Modelo alcanza >97% de precisión (objetivo: >85%)
- **Integración GPU optimizada:** Cliente HTTP robusto para servicio ML especializado
- **Validación de entrada:** Sanitización completa de datos de entrada con rangos validados
- **Manejo de errores:** Respuestas estructuradas y fallback ante fallos del modelo

Interfaz de usuario intuitiva:

- **Frontend React responsive:** Interfaz adaptativa para móviles y desktop
- **Dashboard analítico:** Visualizaciones en tiempo real con Recharts
- **Experiencia de usuario:** Feedback visual, validación en tiempo real y mensajes claros

Infraestructura escalable:

- **Servicios nativos completos:** Todos los servicios configurados con systemd
- **Load balancer Nginx:** Distribución de carga con terminación TLS

Seguridad y calidad:

- **Autenticación JWT robusta:** Tokens seguros con expiración y validación completa
- **Testing automatizado:** Cobertura >80% backend, >70% frontend
- **Políticas de seguridad:** HTTPS, rate limiting, validación de entrada, encriptación

9.2 Limitaciones identificadas

9.2.1 Limitaciones técnicas

Escalabilidad del modelo ML:

- Dependencia de un solo servicio GPU para todas las predicciones

Funcionalidades del sistema:

- Modelo estático sin aprendizaje continuo automático
- Sin geolocalización automática para datos contextuales

9.2.2 Limitaciones operativas

Infraestructura:

- Dependencia de infraestructura específica
- Certificados SSL autofirmados para desarrollo
- Sin configuración de múltiples regiones o DR sites

Desarrollo y mantenimiento:

- Equipo de desarrollo limitado a 4 estudiantes
- Sin pipeline de CI/CD completamente automatizado

Datos y contenido:

- Modelo entrenado solo con base de datos mediana
- Falta de datos históricos para validación a largo plazo
- Sin validación con expertos agrónomos independientes
- Cobertura limitada de cultivos especializados o regionales

9.3 Reflexiones del equipo

9.3.1 Lecciones aprendidas

Desarrollo de software:

- La documentación temprana y continua es crucial para proyectos complejos.
- La comunicación regular del equipo es fundamental.
- Las decisiones de arquitectura iniciales impactan todo el ciclo de desarrollo

Tecnologías utilizadas:

- React con TypeScript acelera el desarrollo frontend significativamente
- Flask proporciona flexibilidad sin la complejidad de frameworks más pesados

Gestión de proyecto:

- Las metodologías ágiles se adaptan bien a proyectos académicos con deadlines fijos
- La integración continua detecta problemas antes de que se vuelvan críticos
- La retroalimentación temprana de usuarios reales mejora significativamente el producto

9.3.2 Contribuciones individuales destacadas

Luis Rico (ML Engineer & System Admin):
- Diseño e implementación de la arquitectura de base de datos completa
- Integración robusta con el servicio GPU de machine learning
- Scripts de automatización para backup y monitoreo del sistema

Ian Holender (Backend Developer):
- Sistema de autenticación JWT completo con seguridad robusta
- API REST bien estructurada con manejo de errores comprehensivo
- Implementación de rate limiting y políticas de seguridad

David Vieyra (Frontend Developer):
- Interfaz de usuario intuitiva y responsive con React/TypeScript
- Dashboard de analytics con visualizaciones interactivas
- Experiencia de usuario optimizada para agricultores no técnicos

Omar Rivera (Full Stack Developer):
- Integración frontend-backend seamless con manejo de estados
- Testing end-to-end y validación de flujos completos de usuario

9.5.3 Valor académico y profesional

Este proyecto ha demostrado la capacidad del equipo para:
- Diseñar y implementar sistemas distribuidos complejos
- Integrar múltiples tecnologías modernas en una solución cohesiva
- Aplicar principios de ingeniería de software en un entorno realista
- Crear documentación técnica comprehensiva y mantenible
- Trabajar efectivamente en equipo con roles especializados
- Entregar un producto funcional bajo presión y plazos académicos

El conocimiento adquirido en áreas como gestión de servicios nativos, APIs REST, autenticación segura, testing automatizado y despliegue en cloud constituye una base sólida para futuras carreras en desarrollo de software empresarial.

9.6 Conclusión final

AgriAI representa un ejemplo exitoso de cómo la tecnología puede abordar problemas reales del sector agrícola mexicano. El proyecto ha logrado crear una solución técnicamente sólida, funcionalmente completa y potencialmente impactante para la comunidad objetivo.

La combinación de machine learning, desarrollo web moderno e infraestructura cloud ha resultado en una plataforma que no solo cumple los requisitos académicos, sino que también tiene potencial real de adopción y crecimiento en el mercado agrícola.

El compromiso del equipo con la calidad técnica, documentación comprehensiva y experiencia de usuario ha establecido fundaciones sólidas para el futuro desarrollo y escalamiento del sistema. Las lecciones aprendidas contribuyen al crecimiento profesional del equipo.

Este proyecto demuestra que estudiantes comprometidos, con las herramientas adecuadas y una visión clara, pueden crear soluciones tecnológicas que generen valor real para la sociedad, cumpliendo tanto objetivos académicos como potencial de impacto comercial y social.

10. Referencias

1. Meta. (2024). *React - The library for web and native user interfaces*. https://react.dev
2. Microsoft TypeScript Team. (2024). *TypeScript documentation*. https://www.typescriptlang.org/docs/
3. Pallets Projects. (2024). *Flask documentation* (Version 3.1.1). https://flask.palletsprojects.com
4. The PostgreSQL Global Development Group. (2025). *PostgreSQL 14.18 documentation*. https://www.postgresql.org/docs/14/
5. Nginx Inc. (2025). *nginx documentation*. https://nginx.org/en/docs/
6. Jones, M., Bradley, J., & Sakimura, N. (2015). *JSON Web Token (JWT)* (RFC 7519). Internet Engineering Task Force. https://tools.ietf.org/rfc/rfc7519.txt