****

Campus Santa Fe

Implementación de redes de área amplia y servicios distribuidos

Desarrollo de aplicaciones avanzadas de ciencias computacionales

TC3003B.501 y TC3002B.502

Documentación de BD e IA

**Assessment**

Benjamín Alejandro Cruz Cervantes - A01747811

Alberto Iván Tamez González - A01026999

Bernardo Alejandro Limón Montes de Oca - A01736575

Fecha de entrega:

Lunes 09 de Junio de 2025

**ÍNDICE:**

[**1. Introducción 2**](#)

[**2. Diseño y estructura de la base de datos 2**](#)

[***2.1 Modelo relacional* 2**](#)

[2.1.1 Tablas principales **2**](#)

[***2.2 Consultas frecuentes* 3**](#)

[***2.3 Justificación del diseño* 3**](#)

[**3. Inteligencia Artificial 3**](#)

[***3.1 Entrenamiento del modelo* 3**](#)

[***3.2 Guardado y uso del modelo* 5**](#)

[***3.3 Microservicio IA* 5**](#)

[***3.4 Ejecución y pruebas* 5**](#)

[***3.5 Integración con backend* 6**](#)

[**4. Conclusión 6**](#)

**Documento 2: Base de Datos e Inteligencia Artificial**

# 1. Introducción

Este documento describe el diseño, implementación y uso de la base de datos relacional y del modelo de inteligencia artificial utilizados en la plataforma de recomendación de cultivos. Ambos elementos forman el núcleo lógico del sistema: la base de datos permite el almacenamiento y consulta eficiente de usuarios, predicciones y acciones; mientras que el modelo IA proporciona la capacidad de análisis automatizado de datos agrícolas, entregando una recomendación por cada fila del archivo cargado por el usuario.

# 2. Diseño y estructura de la base de datos

## 2.1 Modelo relacional

La base de datos fue diseñada en tercera forma normal (*3FN*) para garantizar integridad y evitar redundancia. Se utilizaron claves foráneas para mantener la relación entre usuarios y sus predicciones.

### 2.1.1 Tablas principales:

CREATE TABLE users (  
 id SERIAL PRIMARY KEY,   
 workos\_user\_id VARCHAR(255) UNIQUE,   
 email VARCHAR(255) UNIQUE NOT NULL,  
 full\_name VARCHAR(255),  
 is\_admin BOOLEAN DEFAULT FALSE,  
 is\_active BOOLEAN DEFAULT TRUE,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
CREATE TABLE predictions (  
 id SERIAL PRIMARY KEY,  
 user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,  
 input\_source VARCHAR(10) NOT NULL,  
 file\_name VARCHAR(255),  
 crop\_result VARCHAR(100) NOT NULL,  
 created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
CREATE TABLE activity\_logs (  
 id SERIAL PRIMARY KEY,  
 user\_id INTEGER REFERENCES users(id) ON DELETE CASCADE,  
 action VARCHAR(50) NOT NULL,  
 timestamp TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
);  
  
CREATE TABLE app\_sessions (  
 id SERIAL PRIMARY KEY,  
 user\_id INTEGER NOT NULL,  
 workos\_user\_id VARCHAR(255) NOT NULL,  
 workos\_session\_id VARCHAR(255) UNIQUE NOT NULL,  
 encrypted\_refresh\_token TEXT NOT NULL,  
 ip\_address VARCHAR(45),  
 user\_agent TEXT,  
 issued\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 refresh\_token\_expires\_at TIMESTAMP WITH TIME ZONE NOT NULL,  
 created\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 updated\_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT\_TIMESTAMP,  
 CONSTRAINT fk\_user FOREIGN KEY(user\_id) REFERENCES users(id) ON DELETE CASCADE  
);



## 2.2 Consultas frecuentes

Estas consultas se utilizan para alimentar las visualizaciones del administrador:

-- Cultivos más recomendados  
SELECT result, COUNT(\*) FROM predictions GROUP BY result;  
  
-- Acciones más frecuentes  
SELECT action, COUNT(\*) FROM activity\_logs GROUP BY action;  
  
-- Actividad por hora  
SELECT date\_part('hour', timestamp) AS hora, COUNT(\*) FROM activity\_logs GROUP BY hora ORDER BY hora;



## 2.3 Justificación del diseño

* **Escalabilidad**: cada usuario puede generar múltiples predicciones sin duplicar datos.
* **Auditoría**: los logs permiten rastrear accesos, predicciones y operaciones administrativas.
* **Seguridad**: no se almacenan contraseñas (autenticación delegada a *WorkOS*).
* **Modularidad**: es posible migrar la *BD* a otro host sin romper la lógica del backend.

# 3. Inteligencia Artificial

## 3.1 Entrenamiento del modelo

Para entrenar el modelo de inteligencia artificial, se optó por un perceptrón multicapa (*MLP*), debido a su versatilidad para problemas de clasificación con datos estructurados. Se evaluaron tres arquitecturas diferentes para determinar la mejor configuración en términos de precisión, velocidad de entrenamiento y generalización.

El conjunto de datos utilizado consistía en registros sintéticos generados a partir de condiciones agrícolas comunes: humedad, pH, tipo de suelo, temperatura, etc. Estos fueron transformados a *CSV* y procesados antes del entrenamiento.

Las arquitecturas probadas fueron:

* **Arquitectura 1**: Entrada → [32] → Salida
* **Arquitectura 2**: Entrada → [64, 32] → Salida (la más efectiva)
* **Arquitectura 3**: Entrada → [64, 32, 16] → Salida

Las configuraciones detalladas en *PyTorch* para la arquitectura seleccionada fueron:

model = nn.Sequential(  
 nn.Linear(input\_size, 64),  
 nn.ReLU(),  
 nn.Linear(64, 32),  
 nn.ReLU(),  
 nn.Linear(32, output\_size)  
)



Durante el preprocesamiento se normalizaron los valores numéricos, y se transformaron etiquetas categóricas en variables dummy (one-hot encoding). El conjunto de datos se dividió en 80% para entrenamiento y 20% para validación.

* **Función de pérdida**: nn.CrossEntropyLoss()
* **Optimizador**: torch.optim.Adam(lr=0.001)
* **Número de épocas**: 50 (*early stopping* activado si la pérdida no mejoraba)

Se hizo uso de métricas como precisión, recall y matriz de confusión en validación. Se observó un aumento del 6% en precisión al pasar de la arquitectura 1 a la 2. Finalmente, el modelo se exportó en formato .pth para ser usado en producción dentro del microservicio IA. usando un perceptrón multicapa (*MLP*), con variaciones en el número de capas ocultas y neuronas:

* Arquitectura 1: Entrada → [32] → Salida
* Arquitectura 2: Entrada → [64, 32] → Salida
* Arquitectura 3: Entrada → [64, 32, 16] → Salida

La mejor precisión se obtuvo con la arquitectura 2. Las configuraciones fueron:

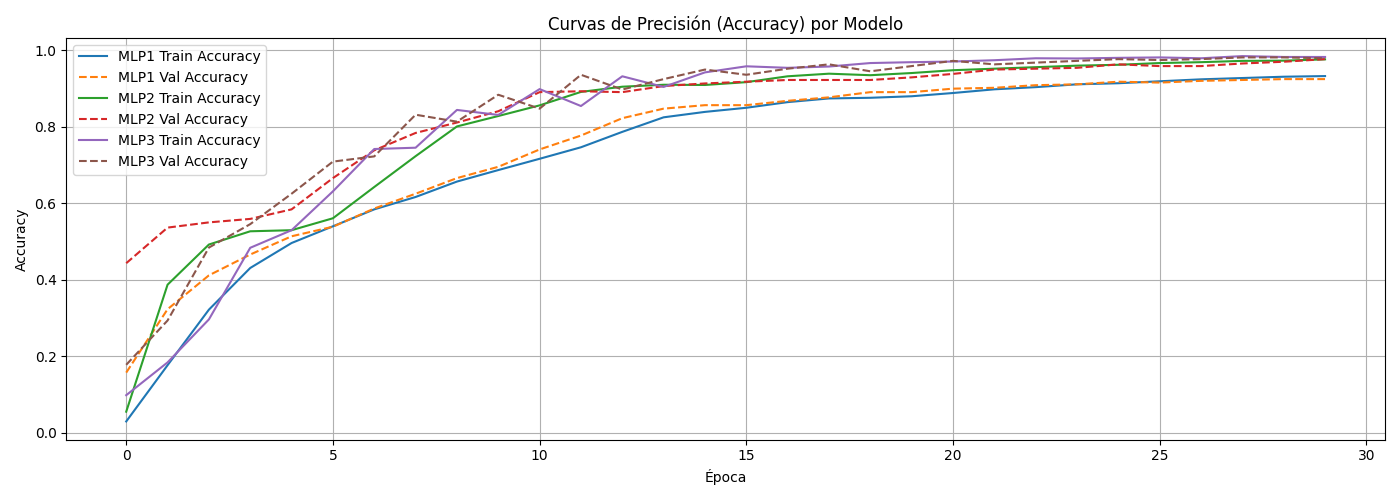
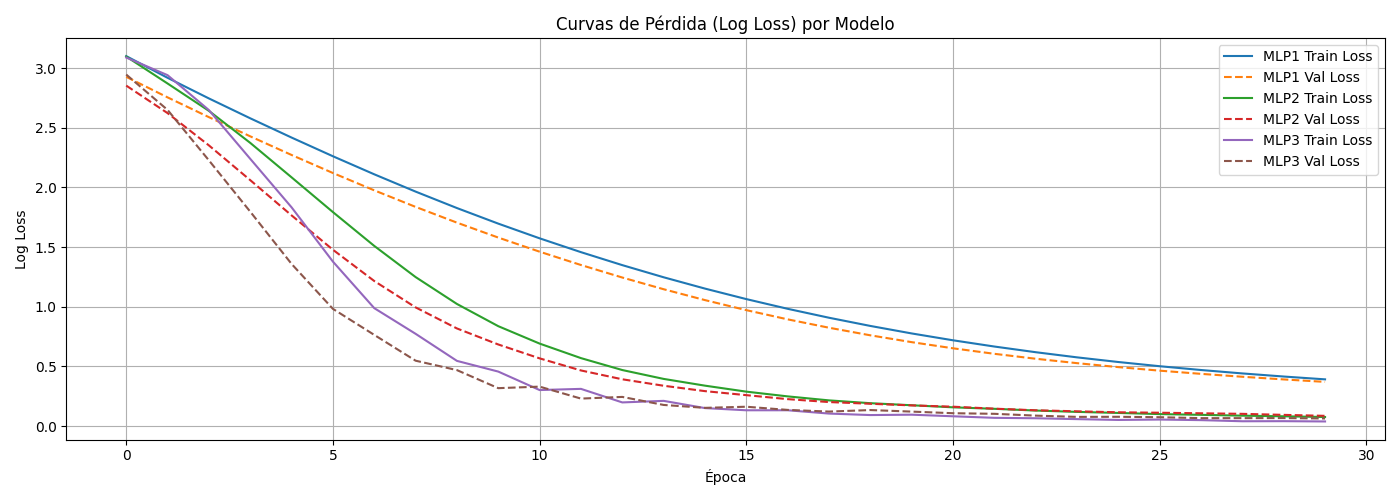
model = nn.Sequential(  
 nn.Linear(input\_size, 64),  
 nn.ReLU(),  
 nn.Linear(64, 32),  
 nn.ReLU(),  
 nn.Linear(32, output\_size)  
)



* Dataset: datos sintéticos formateados como CSV.
* División: 80% entrenamiento, 20% validación.
* Función de pérdida: CrossEntropyLoss()
* Optimizador: Adam con tasa de aprendizaje 0.001

## 3.2 Estadísticas del entrenamiento (modelos de ejemplo)

Al entrenar modelos mediante inteligencia artificial, es fundamental monitorear su desempeño y evaluar cuál de ellos ofrece los mejores resultados. Para ello, una práctica común es visualizar métricas como Accuracy y Log Loss, que permiten analizar el progreso de cada modelo entrenado y facilitar la identificación del más óptimo. A continuación, se muestra un ejemplo de cómo se despliegan estas métricas en forma gráfica:

* Accuracy: indica el porcentaje de predicciones correctas que hace el modelo sobre el total de casos evaluados.
* Log Loss: mide qué tan lejos están las predicciones del modelo respecto a los valores reales donde se intenta minimizar durante el entrenamiento.

## 3.3 Guardado y uso del modelo

El modelo entrenado se guardó con:

torch.save(model.state\_dict(), "model.pth")



Y se carga al iniciar el microservicio:

model.load\_state\_dict(torch.load("model.pth"))  
model.eval()



El archivo principal de inferencia es *crop\_recommendation\_2.6.py*, el cual toma como entrada un archivo *CSV*, lo convierte en *DataFrame*, normaliza sus valores y retorna un resultado por fila.

## 3.4 Microservicio IA

El modelo fue desplegado como un microservicio en *FastAPI*, separado del backend principal. Recibe archivos como multipart/form-data:

@app.post("/predict")  
def predict(file: UploadFile = File(...)):  
 # Leer y procesar archivo  
 df = pd.read\_csv(file.file)  
 # Realizar predicción  
 return model\_output

Para protegerlo del resto de la red, se validan solicitudes mediante *API Key*:

@app.middleware("http")  
async def check\_api\_key(request: Request, call\_next):  
 if request.headers.get("x-api-key") != os.getenv("INTERNAL\_API\_KEY"):  
 raise HTTPException(status\_code=403)  
 return await call\_next(request)



## 3.5 Ejecución y pruebas

Ejecución local:

uvicorn crop\_recommendation\_2.6:app --host 0.0.0.0 --port 1337

Prueba con curl:

curl -X POST "http://localhost:1337/predict" \  
 -H "accept: application/json" \  
 -H "X-API-Key: clave-secreta" \  
 -H "Content-Type: multipart/form-data" \  
 -F "file=@dataset.csv"



## 3.6 Integración con backend

Desde el backend FastAPI se hace la conexión al microservicio:

headers = {"x-api-key": os.getenv("INTERNAL\_API\_KEY")}  
response = requests.post("http://172.28.69.157:1337/predict", files=files, headers=headers)

El resultado se guarda en la base de datos y se vincula al usuario autenticado.

# 4. Conclusión

La base de datos y el sistema de IA constituyen el corazón lógico de la plataforma. El diseño modular permite escalar o mejorar el modelo sin afectar la experiencia del usuario. La base de datos mantiene trazabilidad y control, mientras que el microservicio de predicción aporta valor inmediato con una respuesta rápida y confiable basada en datos estructurados. La separación de responsabilidades y el uso de estándares modernos garantizan la mantenibilidad y seguridad del sistema.