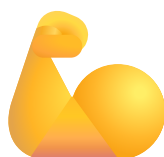




# Proyecto biUNestar

---

Estrategia, Planificación y Avance del Desarrollo



Sistema de seguimiento de hábitos de bienestar desarrollado con arquitectura Django  
y PostgreSQL



# Organización del Equipo

## Pablo Rosales

### Líder Técnico / Integración

Supervisión, Arquitectura y Deployment

🔧 GitHub, Trello

## Omar Quiñones

### Frontend / UI

Diseño visual, plantillas Django (UX/UI) y usabilidad

🎨 Trello, Discord

## Jaime Quiñones

### Backend / Lógica

Modelado de datos (ORM), lógica de negocio y Vistas

⚙️ GitHub

## Metodología Ágil (Agile)

📋 **Gestión:** Trello utilizado como tablero Kanban para priorización y seguimiento de tareas

🔄 **Flujo de Trabajo:** Backlog (Pendiente) → To Do (Por Hacer) → In Progress (En Progreso) → Done (Terminado)

🕒 **Ritmo:** Reuniones semanales de planificación de 60 minutos para revisar el tablero Kanban, estimar tareas, sincronizar al equipo, identificar bloqueos y validar el plan de acción

💬 **Comunicación:** Discord para colaboración inmediata y notificaciones de commits



# Arquitectura y Decisiones Técnicas

## Decisiones Principales

Decisión Técnica	Razón	Beneficio
Framework: Django	Robustez, seguridad integrada y patrón MVC maduro	Arquitectura MVC centralizada + API REST interna
Base de Datos: PostgreSQL	Alta concurrencia, integridad de datos	Escalabilidad garantizada
Seguridad	Protección de datos sensibles	Autenticación nativa Django + Cifrado SHA-256

## Flujo de Datos (Arquitectura MVC)



El usuario interactúa con la vista, la lógica es manejada por el controlador, los datos son gestionados por el modelo y almacenados en PostgreSQL

## Modelos Implementados

**User (Perfil):** Extensión del usuario Django para almacenar datos específicos del perfil de bienestar

**Habit:** Modelo principal para registrar tipos de hábitos (ej: Sueño, Agua, Ejercicio)

**HabitRecord:** Modelo para registrar la constancia diaria del usuario (la clave del sistema)

## Integración Futura

La API interna permitirá que el módulo de reportes (Semana 3) se comunique directamente con la base de datos sin sobrecargar las vistas principales.



# Cronograma de Implementación

**Meta:** Entrega final en Semana 4 de Noviembre

## Semana 1 (Actual) - Funcionalidad Básica de Frontend

45%

- Diseño responsivo (mobile-first) de la dashboard
- Validación client-side de ingreso de hábitos

## Semana 2 - Integración de Datos y Backend

65%

- CRUD completo de hábitos
- Implementación del modelo de Reportes
- Configuración final de variables de entorno (ENV)

## Semana 3 - Características Clave (MVP Completo)

85%

- Módulo de Reportes Gráficos (Chart.js/Matplotlib)
- Sistema de Recordatorios/Notificaciones
- Background Tasks implementados

## Semana 4 - QA, Pruebas y Deployment

100%

- Pruebas unitarias e integración
- Documentación técnica final
- Preparación del entorno de deployment (Docker/Gunicorn)



# Problemas Identificados y Soluciones

## Error de Conexión a la Base de Datos (UnicodeDecodeError)

### Descripción del Problema:

Error que detuvo el proceso de migraciones e inicialización del proyecto. El sistema no podía establecer conexión con PostgreSQL.

### Causa Raíz Identificada:

Las contraseñas del servidor contenían caracteres especiales o acentos que no eran compatibles con la codificación ASCII esperada por el sistema operativo.

### ✓ Solución Implementada:

Reconfiguración de credenciales del servidor para usar contraseñas sin caracteres especiales o acentos (ASCII puras). El error era de infraestructura (el sistema operativo no entendía la contraseña), no de código Python.

## Impacto en el Proyecto

- **Positivo:** Reforzó la capacidad del equipo para distinguir entre bugs de código y errores de configuración de infraestructura
- **Aprendizaje:** Mejoró el proceso de deployment y gestión de infraestructura
- **Prevención:** Implementación de mejores prácticas para manejo de credenciales y variables de entorno
- **Documentación:** Se creó una guía interna para evitar problemas similares en el futuro

## Lección Aprendida

### 📊 Diagnóstico Correcto = Ahorro de Tiempo

Es fundamental distinguir entre errores de código (bugs) y errores de configuración de infraestructura (deployment o BD). Un diagnóstico incorrecto puede llevar a horas de debugging innecesario en el código cuando el problema real está en la configuración del entorno.



# Reflexión y Aprendizajes

## Principales Aprendizajes Reforzados



### Diagnóstico de Fallas

Hemos reforzado la capacidad de distinguir entre errores de código (bug) y errores de configuración de infraestructura (deployment o BD)



### POO y Modularidad

La separación estricta de responsabilidades en Django (Modelos, Vistas, Templates) nos ha permitido trabajar en paralelo de forma efectiva



### Control de Versiones

Uso avanzado de Git (branching) y resolución efectiva de conflictos entre ramas



### Gestión Ágil

El uso de Kanban con Trello facilitó la priorización transparente y el seguimiento continuo de tareas

## ¿Qué Haríamos Diferente?



### Lección Clave para Futuros Proyectos:

Empezaríamos con **PostgreSQL desde el inicio** y no con SQLite. Esto habría evitado la migración posterior y el tiempo perdido solucionando el problema de conexión y configuración.

*"Elegir la infraestructura correcta desde el principio ahorra tiempo y recursos valiosos en el largo plazo."*

## Competencias Desarrolladas

- **Trabajo en Equipo:** Colaboración efectiva con roles claramente definidos
- **Resolución de Problemas:** Capacidad de análisis y diagnóstico de errores complejos
- **Planificación:** Gestión de tiempo y recursos con metodología Ágil
- **Adaptabilidad:** Capacidad de ajustar el plan ante obstáculos imprevistos
- **Documentación:** Registro detallado del proceso y decisiones técnicas



**Estado Actual del Proyecto: 45% completado**



**Meta Final: 100% (Semana 4 de Noviembre)**