



Tecnológico  
de Monterrey



UNIVERSIDAD  
DE BURGOS

# DESARROLLO DE UN GESTOR DE TAREAS CON METODOLOGÍA SCRUM

Rodrigo Gutierrez Garcia  
Liliana Odette Ortega Quezada  
Victoria Rodríguez Domínguez  
Carlos Julian Lopez Chavez  
Vicente Jesus Ramos Chavez



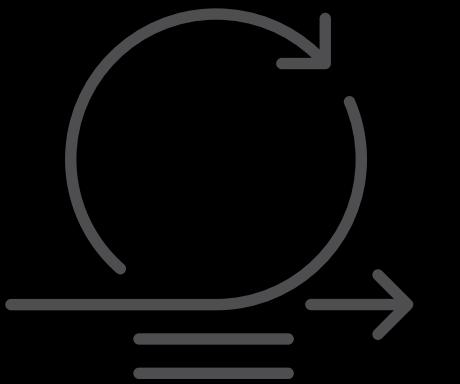
# OBJETIVO DE LA PRÁCTICA



Gestionar tareas.



Aplicar reglas de  
negocio realistas.



Trabajar siguiendo los  
principios y eventos de  
Scrum.



# INICIO

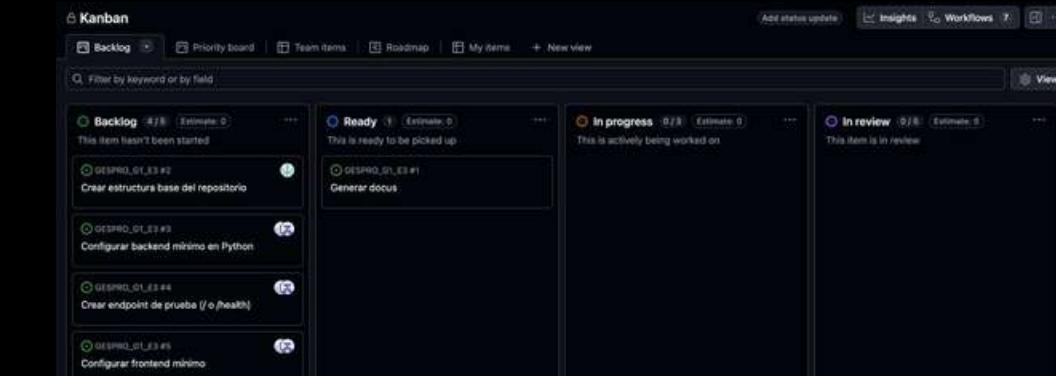
- Configuración del entorno



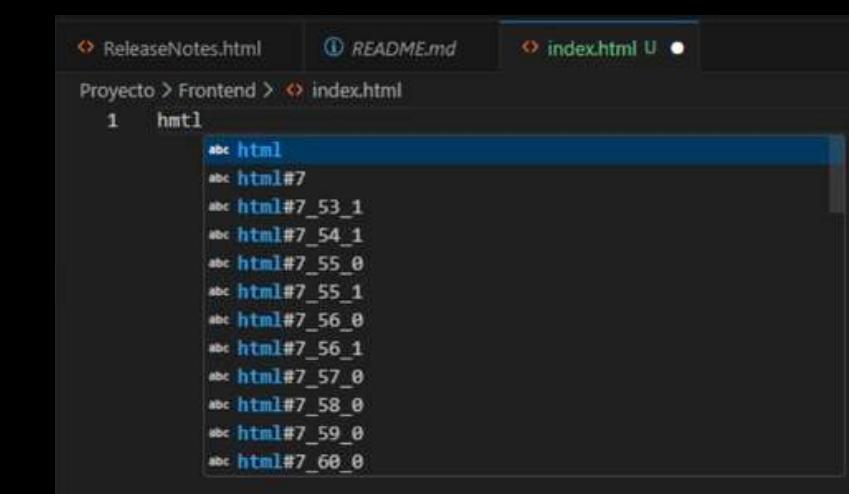
- Crear repository en GitHub y agregar colaboradores



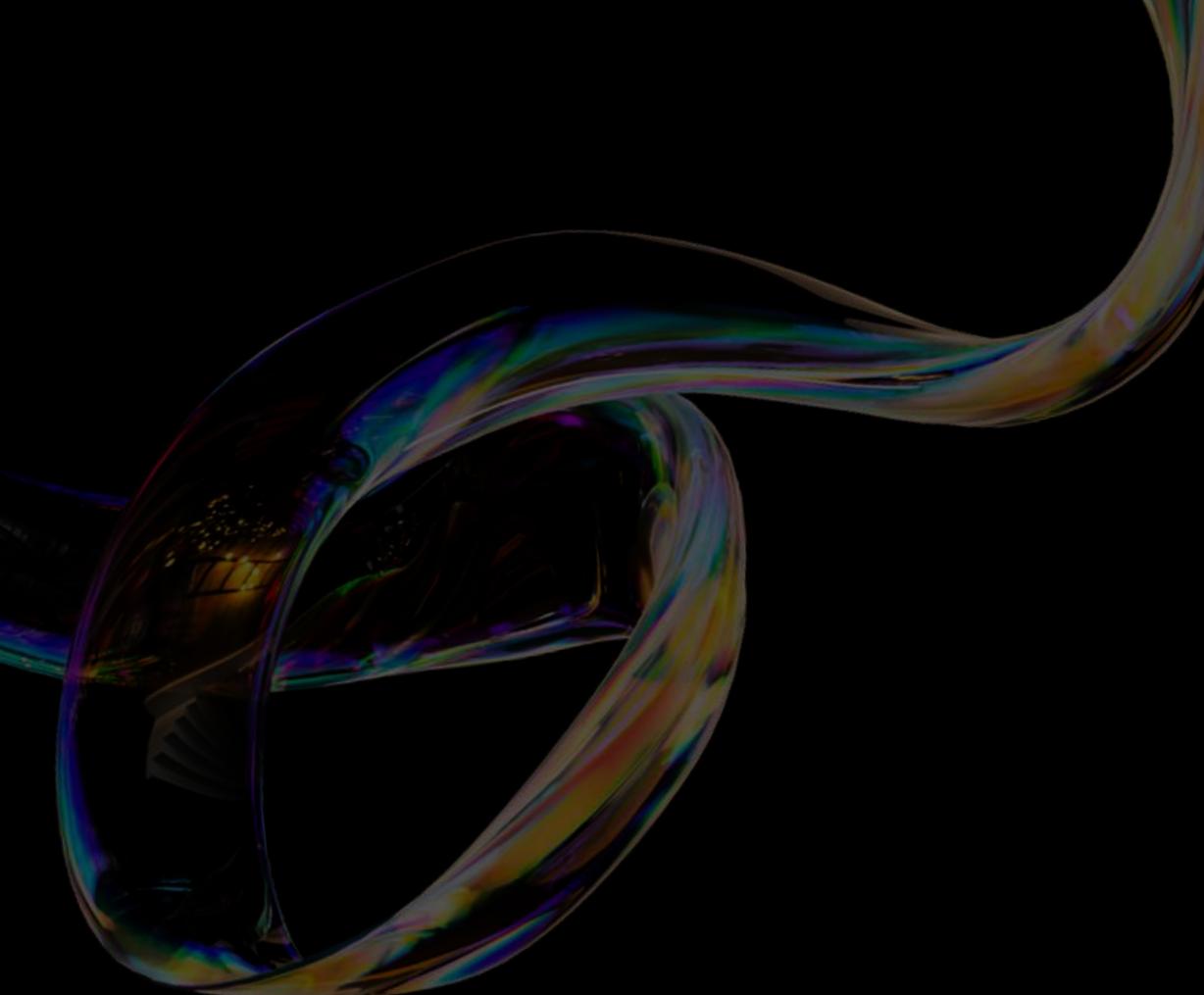
- Crear Kanban en GitHub Projects

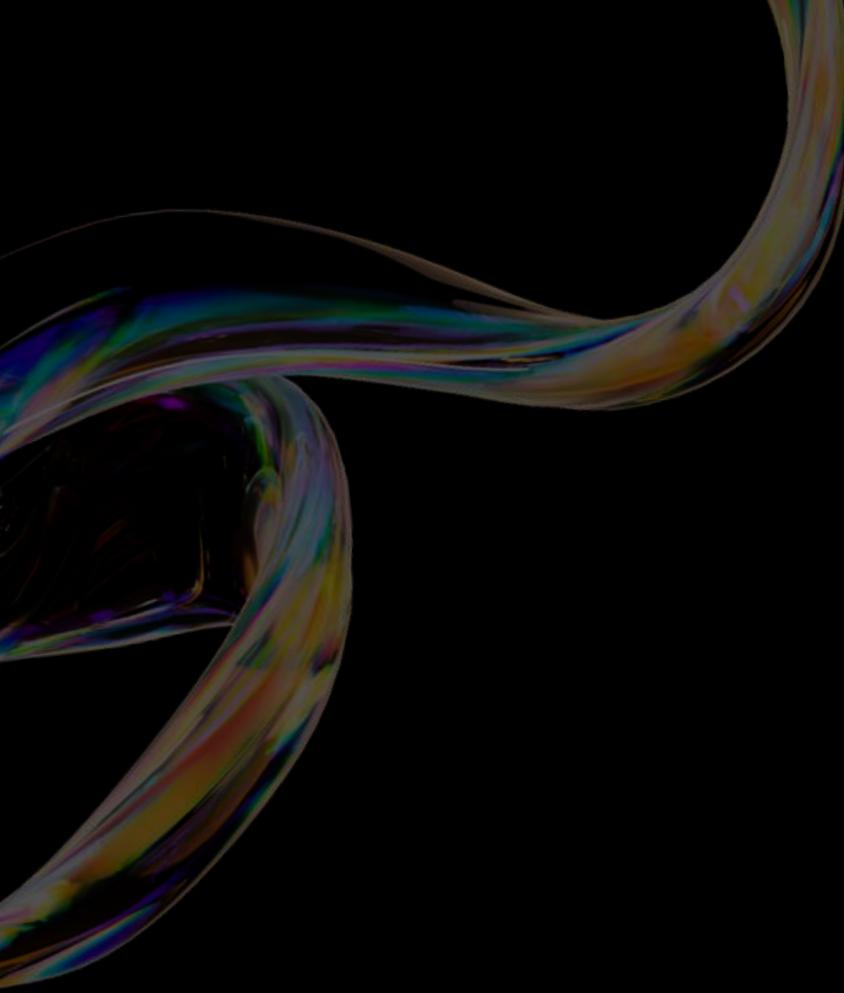


- Clonar el repo
- Crear entorno y abrir VS Code

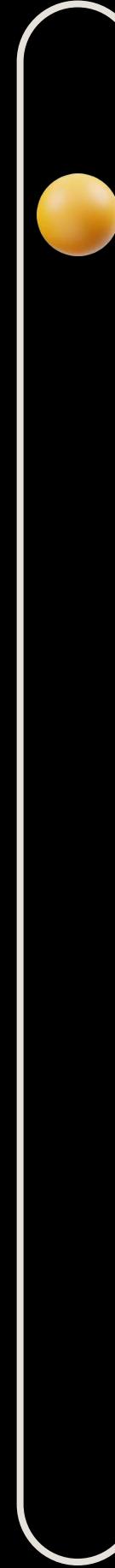
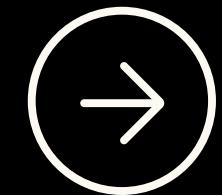


# QUÉ SE HA CONSTRUIDO (DEMO)





# QUÉ SE HA CONSTRUIDO (DEMO)



## SPRINT 1 – CONFIGURACIÓN Y BASE DEL PROYECTO

### Crear estructura base del repositorio

Se crea la estructura inicial del proyecto para mantener el código organizado:

- Carpeta `backend/` para el servidor
- Carpeta `frontend/` para la parte visual
- Carpeta `docs/` para documentación
- Archivo `.gitignore` para excluir archivos innecesarios (entorno virtual, caché, etc.)

### Backend en Visual Studio

Configuración del entorno Python

Creación de aplicación básica con Flask/FastAPI

Verificación de que el servidor arranca correctamente

### Crear 1 endpoint de prueba

Endpoint `/health` o `/`

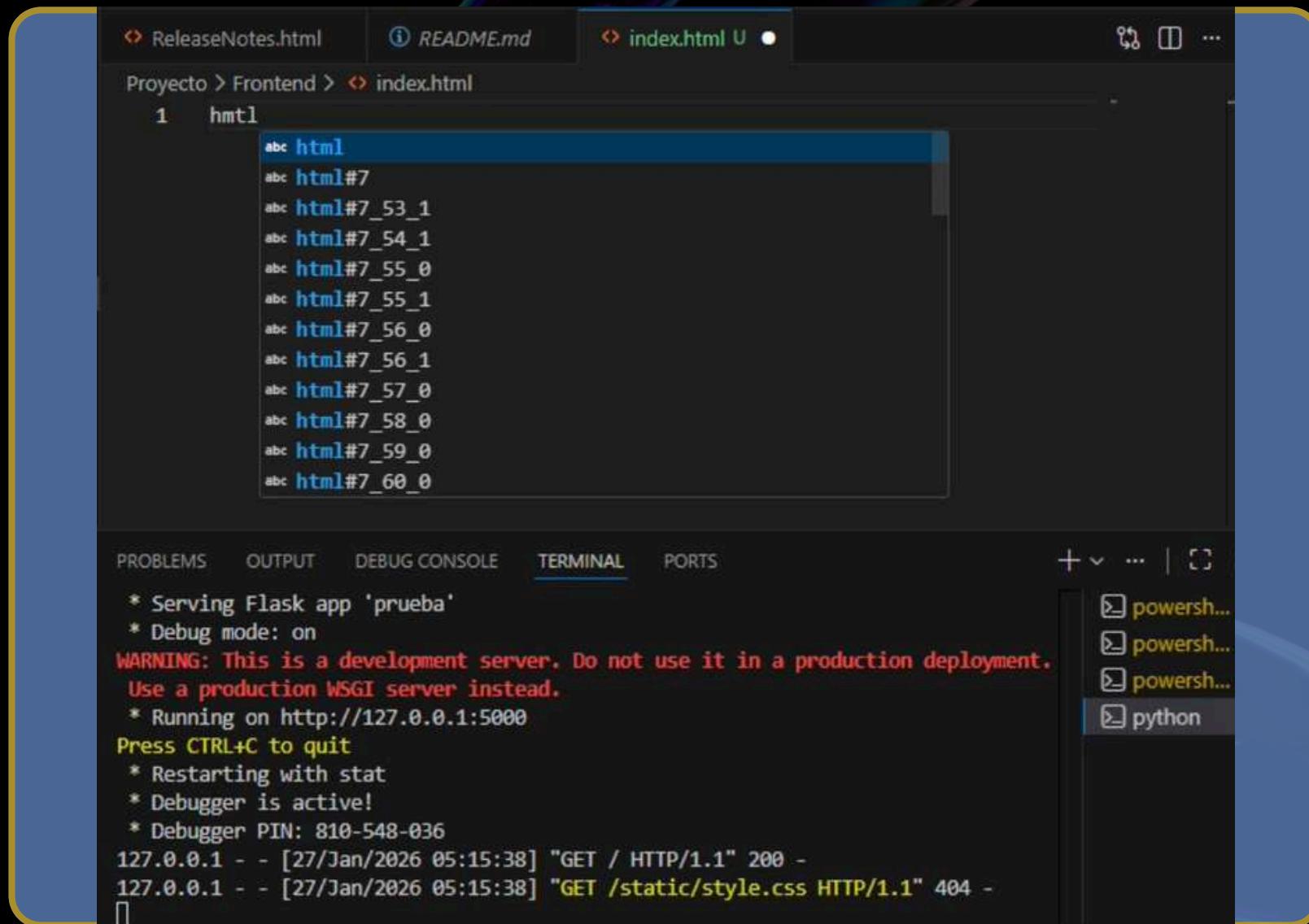
Devuelve mensaje o JSON de prueba

Accesible desde navegador

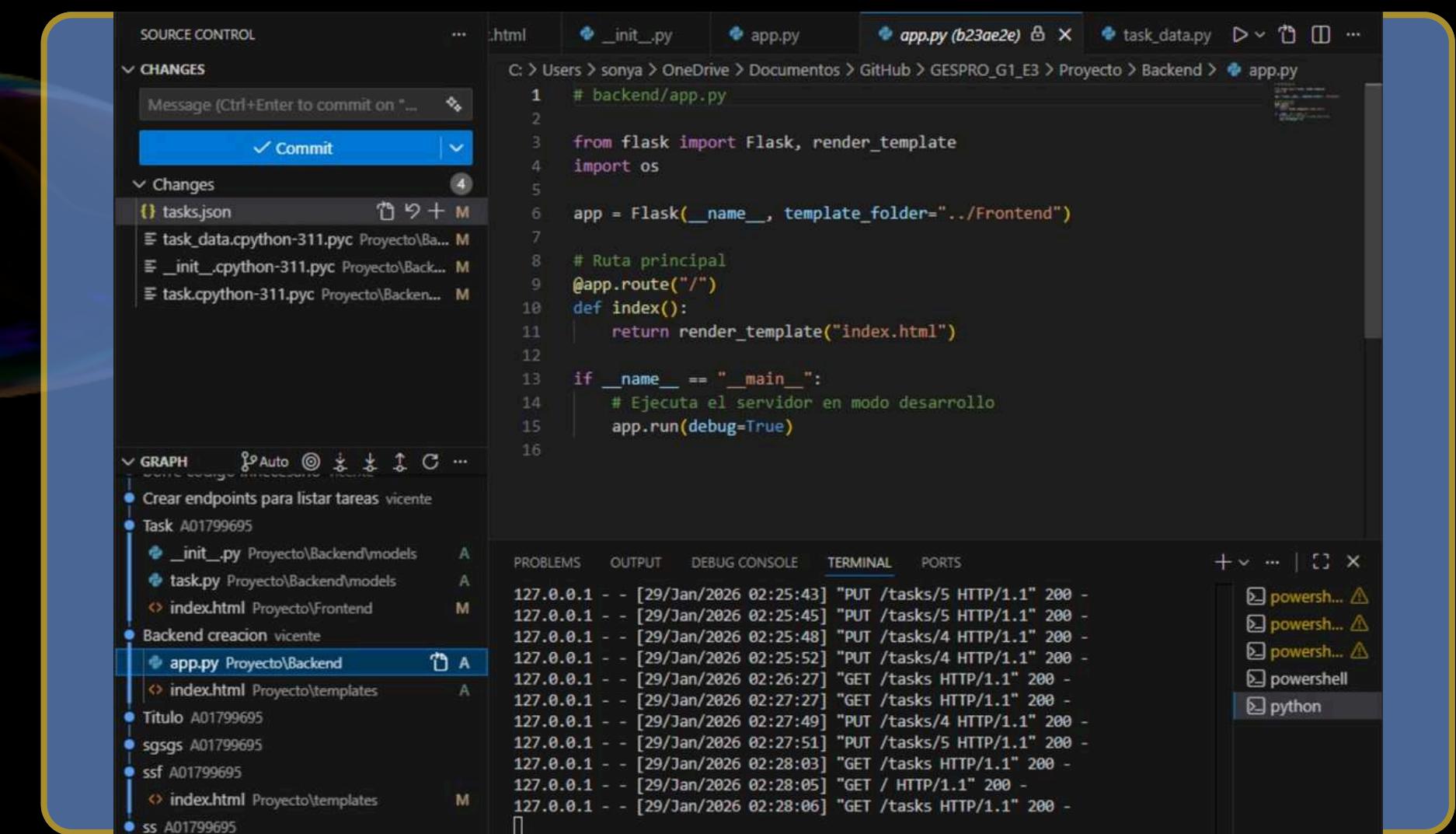
### Frontend

Llamada básica al backend

Mostrar en pantalla el texto devuelto por el endpoint

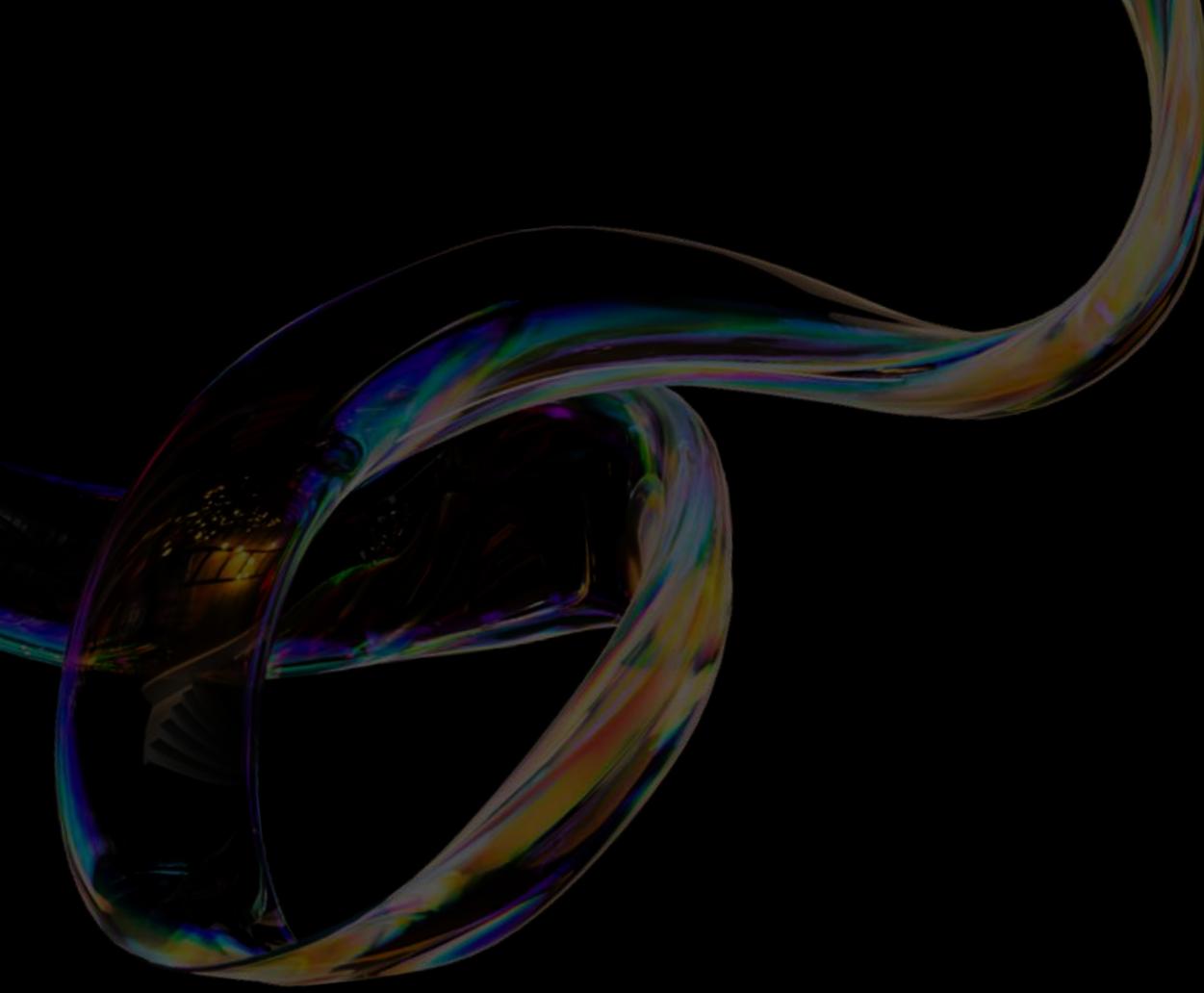


# ESTRUCTURA DEL PROYECTO EN VS CODE

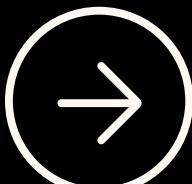


## CREACION DE APP . PY (APLICACION DE USO)

**CREACION DE INDEX (HTML) "PAGINA WEB"**



# QUÉ SE HA CONSTRUIDO (DEMO)



## SPRINT 2 –

### Conexión Frontend - Backend

Implementación básica entre ambas capas del sistema:

- Uso de **fetch** para realizar peticiones al backend.
- Uso del endpoint de prueba creado en el Sprint anterior.
- Verificación de respuesta correcta desde el navegador

### Definición del modelo de datos Task

Se define la estructura básica de una tarea:

- id
- título
- estado

### Crear endpoint GET /tasks

- Devuelve lista de tareas
- Respuesta en formato JSON
- Datos simulados inicialmente

### Renderizado en el frontend

- Captura de la respuesta del servidor
- Inserción dinámica del contenido en el HTML
- Confirmación del flujo completo:

**Servidor > Cliente**

### Resultado del Sprint

- Comunicación funcional entre frontend y backend
- Validación de que la arquitectura del proyecto funciona

SOURCE CONTROL

CHANGES

Message (Ctrl+Enter to commit on "...")

✓ Commit

Changes

- tasks.json M
- task\_data.cpython-311.pyc Proyecto\Backen... M
- \_init\_.cpython-311.pyc Proyecto\Back... M
- task.cpython-311.pyc Proyecto\Backen... M

GRAPH

- Paso 9 y 10 A01799695
- task.cpython-311.pyc Proyecto\Backen... A
- task\_data.py Proyecto\Backend A
- Mostrar listado de tareas en el frontend... D
- borré código innecesario vicente
- Crear endpoints para listar tareas vicente
- Task A01799695
- \_init\_.py Proyecto\Backend\models A
- task.py Proyecto\Backend\models A
- index.html Proyecto\Frontend M

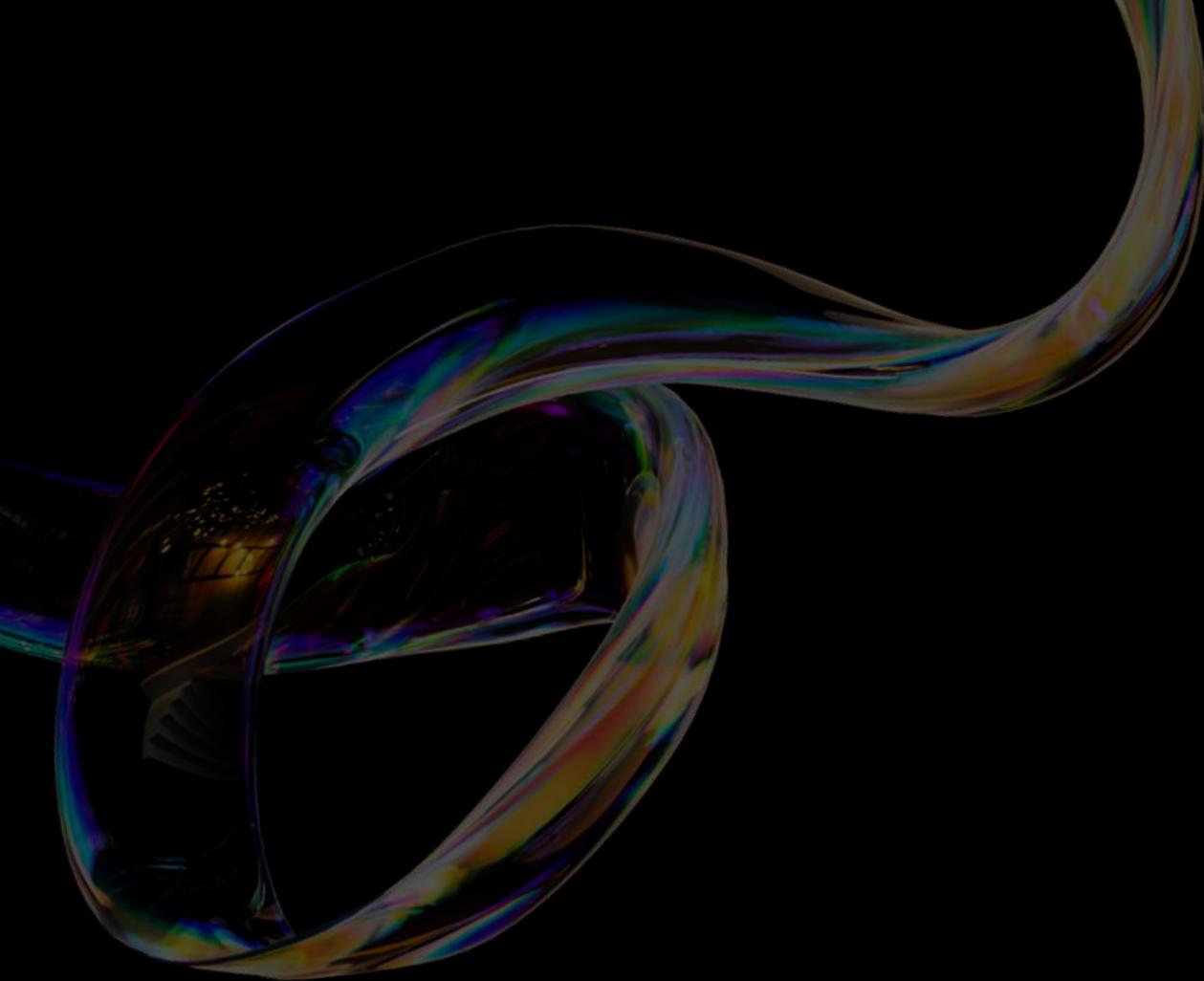
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
127.0.0.1 - - [29/Jan/2026 02:25:43] "PUT /tasks/5 HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:25:45] "PUT /tasks/5 HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:25:48] "PUT /tasks/4 HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:25:52] "PUT /tasks/4 HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:26:27] "GET /tasks HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:27:27] "GET /tasks HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:27:49] "PUT /tasks/4 HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:27:51] "PUT /tasks/5 HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:28:03] "GET /tasks HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:28:05] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Jan/2026 02:28:06] "GET /tasks HTTP/1.1" 200 -
```

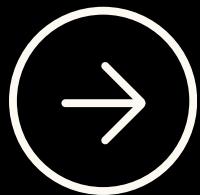
SPRINT 2



# CODIGOS DE RESULTADOS DEL SPRINT 2



# QUÉ SE HA CONSTRUIDO (DEMO)



## SPRINT 3 – CONFIGURACIÓN Y BASE DEL PROYECTO

### Crear endpoint POST/ tasks

**Se implementa la creación de nuevas tareas desde el backend:**

- Recepción de datos en formato JSON.
- Generación automática de id.
- Agregado a lista en memoria.

### Formulario en el frontend

- Input para título de tarea
- Botón de envío
- Envío mediante fetch (POST)

### Actualización automática de la lista

- Refresh de las tareas tras su creación
- Mostrar nueva tarea sin recargar página

```
20      ]
21      # Convertimos cada objeto a diccionario
22      return jsonify([task.to_dict() for task in tasks])
23
24+ def tasks_get():
25+     return jsonify([task.to_dict() for task in get_tasks()])
26
27+ # POST /tasks → crea una nueva tarea
28+ @app.route("/tasks", methods=["POST"])
29+ def tasks_post():
30     data = request.get_json()
31
32     if not data or "titulo" not in data:
33         return jsonify({"error": "El campo 'titulo' es obligatorio"}), 400
34
35     nueva_tarea = add_task(data["titulo"])
36     return jsonify(nueva_tarea.to_dict()), 201
37
38
39 if __name__ == "__main__":
40     # Ejecuta el servidor en modo desarrollo
```

## CREACION DE METODO “POST” EN LA RUTA “/”

```
{
    "id": 7,
    "titulo": "Creacion de frontend",
    "estado": "TODO",
    "puntos": 1,
    "asignado_a": "Julian"
},
{
    "id": 8,
    "titulo": "Creacion de Backend",
    "estado": "DONE",
    "puntos": 1,
    "asignado_a": "Diego"
},
{
    "id": 9,
    "titulo": "Creacion de Base de Datos",
    "estado": "TODO",
    "puntos": 1,
    "asignado_a": "Marco"
},
{
    "id": 10,
    "titulo": "Documentacion y pruebas",
    "estado": "TODO",
    "puntos": 1,
    "asignado_a": "Daniela"
}
```

## ALMACENAMIENTO DE DATOS EN JSON

### Gestor de Tareas

Nueva Tarea

Título de la tarea  Crear

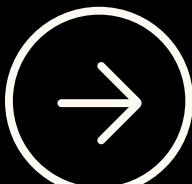
Lista de Tareas

Aprender Flask	PENDIENTE
ID: 1	
Crear API REST	PENDIENTE
ID: 2	
Conectar frontend	COMPLETADA
ID: 3	
hacer olimpiadas	PENDIENTE
ID: 4	
hacer tarea	PENDIENTE
ID: 5	

## RESULTADO DEL SPRINT



# QUÉ SE HA CONSTRUIDO (DEMO)



## SPRINT 4 – MEJORAR LA FUNCIONALIDAD Y USABILIDAD DEL GESTOR DE TAREAS

### Validaciones al crear tareas

- Evitar títulos vacíos
- Mostrar mensajes de error

### Estado de las tareas

- Implementación de estados (pendiente, en progreso, completada)

### Mejora del frontend

- Interfaz más clara
- Organización visual por columnas

### Mover tareas entre columnas

- Cambio de estado reflejado dinámicamente

### Eliminar tareas

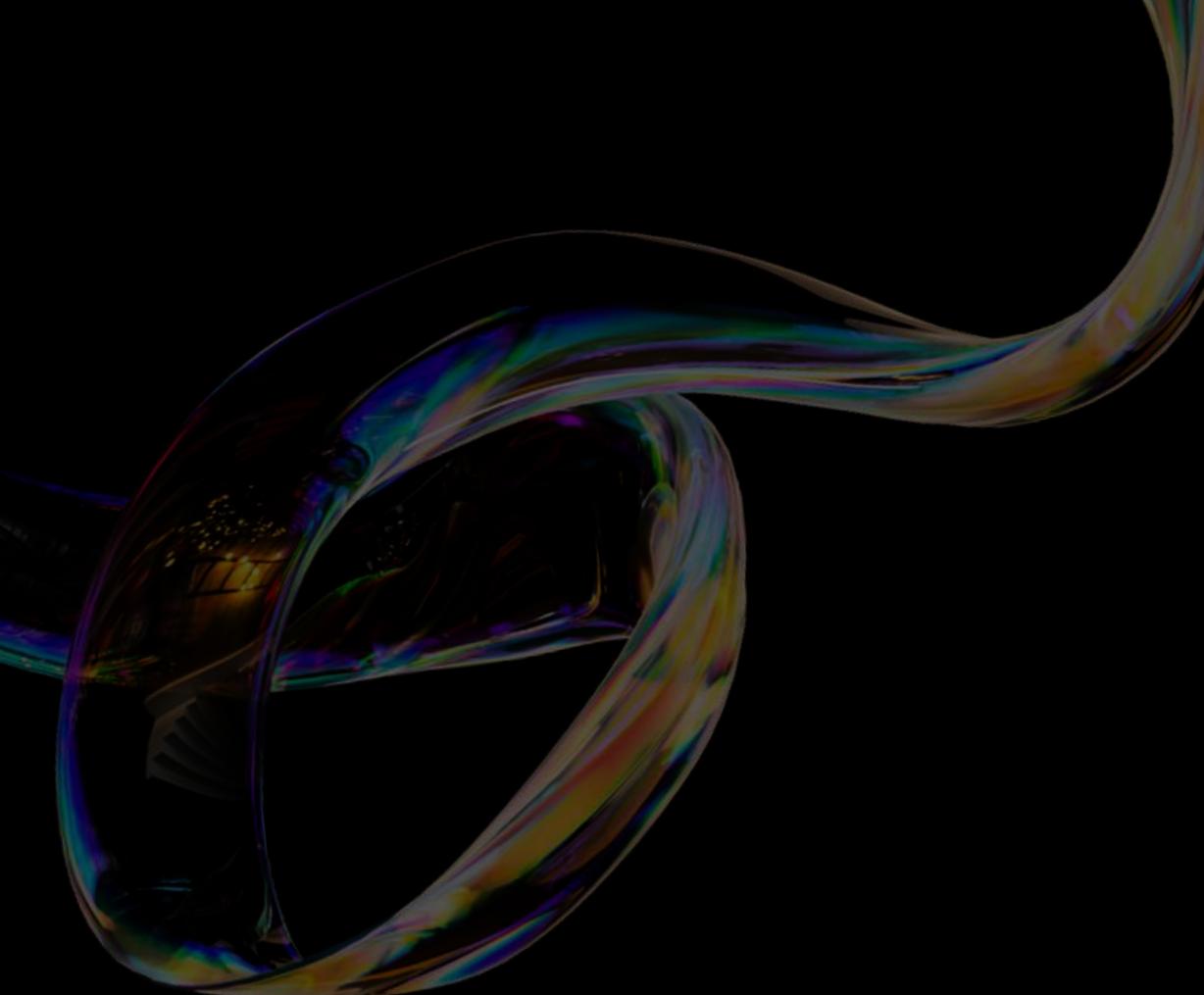
- Botón para eliminar tareas
- Actualización automática en pantalla

## ESTRUCTURA DEL PROYECTO EN VS CODE

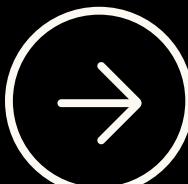
The screenshot displays a task management application interface. On the left, a "Gestor de Tareas" section shows a backlog with items like "Nueva Tarea" and a progress board with columns: TO DO, IN PROGRESS, and DONE. The progress board lists tasks such as "Creacion de frontend", "Creacion de Base de Datos", and "Documentacion y pruebas". On the right, a "VS Code" interface is shown, displaying the project structure and code for "tasks.json". The code defines tasks with fields like id, titulo, estado, puntos, and asignado\_a. The "Source Control" panel shows recent changes and a commit message. The "Terminal" tab shows logs of API requests.

## ESTRUCTURA DEL PROYECTO EN VS CODE

The screenshot shows a Kanban board with four columns: Ready, In progress, In review, and Done. The Ready column has one item: "This is ready to be picked up". The In progress column has three items: "Validaciones al crear tareas", "Estado de las tareas", and "Mejorar frontend". The In review column has five items: "Crear estructura base del repositorio", "Generar docus", "Configurar frontend mínimo", "Configurar backend mínimo en Python", and "Crear endpoint de prueba (/o/health)". The Done column has eleven items: "Este item es en revisión", "Validaciones al crear tareas", "Estado de las tareas", "Mejorar frontend", "Mover tareas entre columnas", "Eliminar tareas", "Crear estructura base del repositorio", "Generar docus", "Configurar frontend mínimo", "Configurar backend mínimo en Python", and "Definir modelo de datos Task".



# QUÉ SE HA CONSTRUIDO (DEMO)



## SPRINT 5

### Mejora de estructura y organización

- Limpieza de código
- Separación de lógica
- Revisión de commits

### Documentación inicial

- Creación de README.md
- Instrucciones para ejecutar el proyecto en local
- Descripción de funcionalidades implementadas

**Se implementan reglas de negocio simples:**

- Evitar títulos vacíos
- Control de errores en backend
- Mensajes de error al usuario

**APP.PY**

**HTML**

The image shows two side-by-side instances of Microsoft Visual Studio Code. The left instance, labeled 'APP.PY', displays the Python code for the Backend, specifically the file 'app.py'. The right instance, labeled 'HTML', displays the HTML code for the Frontend, specifically the file 'index.html'. Both instances show a dark theme with syntax highlighting for their respective languages.

```
APP.PY (Backend code):
```

```
from flask import Flask, render_template, jsonify, request
from models.task import Task
from data.task_data import get_tasks, add_task, update_task, delete_task

app = Flask(__name__, static_folder='../Frontend', template_folder='../Frontend')

# Ruta principal - sirve el HTML
@app.route("/")
def index():
    return render_template("index.html")

# GET /tasks → devuelve todas las tareas ordenadas por tiempo
@app.route("/tasks", methods=["GET"])
def tasks_get():
    try:
        tasks = get_tasks()
        # Convertir a formato JSON
        tasks_data = [task.to_dict() for task in tasks]
        return jsonify(tasks_data)
    except Exception as e:
        return str(e)
```

```
HTML (Frontend code):
```

```
<html lang="es">
<body>
<script>
document.addEventListener('keydown', (e) => {
    if (e.key === 'Escape') {
        cerrarModal();
    }
});

// Cerrar modal haciendo clic fuera
document.getElementById('modal-edicion').addEventListener('click', (e) => {
    if (e.target.id === 'modal-edicion') {
        cerrarModal();
    }
});
</script>
</body>
</html>
```

**EDICIÓN Y GESTIÓN DE TAREA**

**Gestor de Tareas**

Organiza, prioriza y asigna tareas con tiempo estimado

**Nueva Tarea**

Título de la tarea *	Puntos (1-10)	Tiempo estimado (minutos)	Asignado a	Estado inicial
Ej: Revisar documentación	1	60	Ej: Juan	TODO

**Crear Tarea**

Total: 4 tareas | TODO: 1 | IN PROGRESS: 1 | DONE: 2

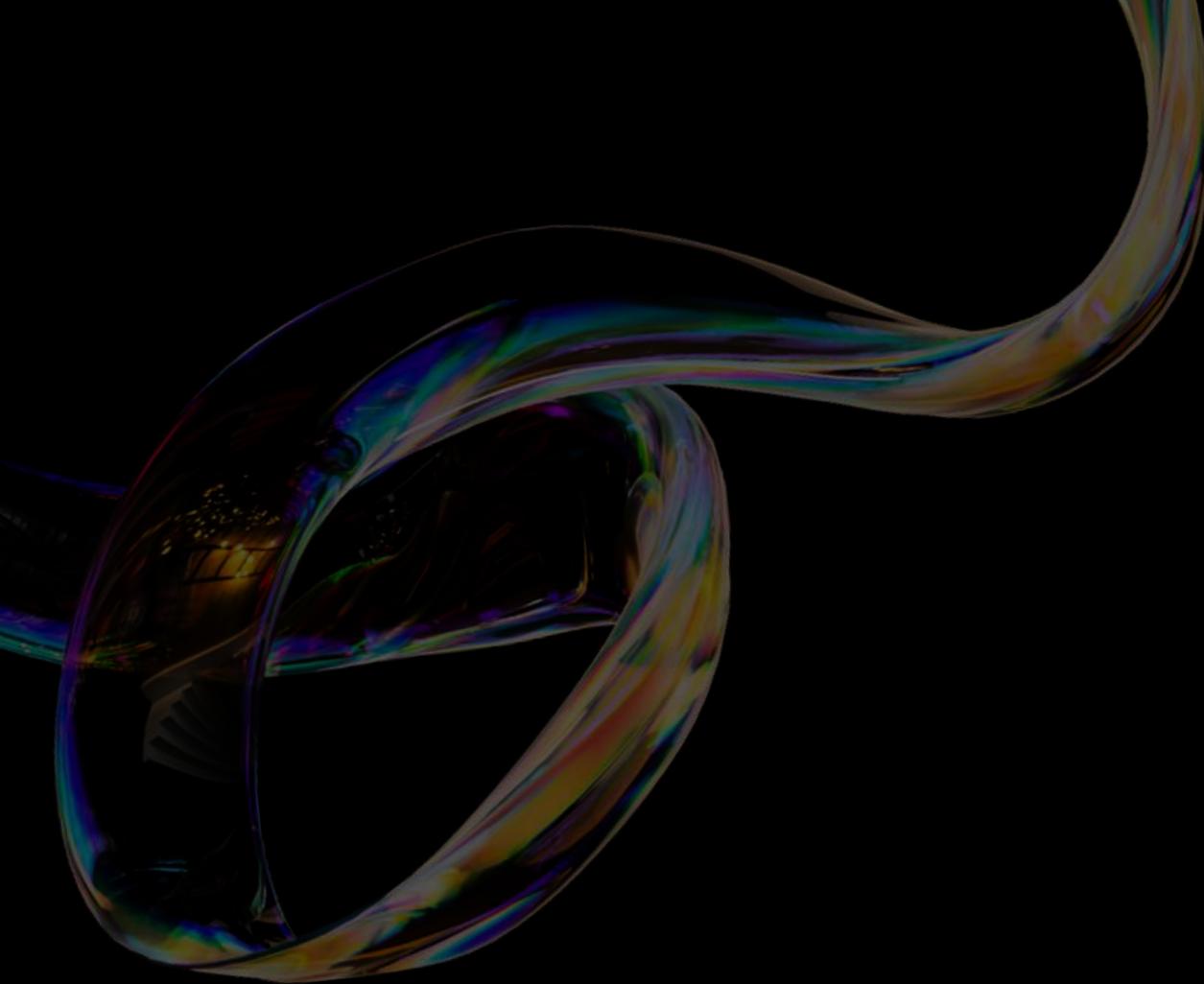
Tiempo total estimado: 285 minutos (4h 45m)

**TABLERO INICIAL**

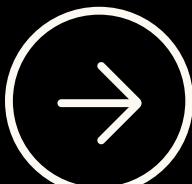
The image shows the application's main interface. At the top, there are two tabs: 'EDICIÓN Y GESTIÓN DE TAREA' and 'TABLERO INICIAL'. The 'TABLERO INICIAL' tab is active, displaying a dashboard with three columns: 'TODO', 'IN PROGRESS', and 'DONE'. Each column lists tasks with their details (title, points, estimated time, assignee, and status). Below the dashboard, there are two sections: 'Creación de Backend' and 'Creación de Base de Datos', each with their own progress bars and status indicators.

**EDICIÓN Y GESTIÓN DE TAREA**

**TABLERO INICIAL**



# QUÉ SE HA CONSTRUIDO (DEMO)



## SPRINT 6

### Persistencia de tareas en fichero JSON

- bbdd/tareas.json
- gitignore
- Límite para WIP
- Flexibilidad en el límite WIP

```
<!-- Contadores y WIP -->
<div class="contadores-container">
  <div class="contador-tareas" id="contador-tareas">
    Total: 0 tareas | TODO: 0 | IN PROGRESS: 0 | DONE: 0
  </div>
  <div class="tiempo-total" id="tiempo-total">
    Tiempo total estimado: 0 minutos
  </div>
  <div class="wip-container" id="wip-container">
    <div class="wip-header">
      Límite WIP Flexible
    </div>
    <div class="wip-limite">
      Límite base:<input type="number" id="wip-limit" class="wip-input" min="1" max="20" value="5">
      <button class="btn-wip" onclick="actualizarWIP()>Actualizar</button>
    </div>
    <div class="wip-info">
      Límite flexible: <span id="wip-flexible" class="wip-flexible">5 (+10% = 5)</span>
    </div>
    <div class="wip-actual">
      Tareas en progreso:<span id="wip-actual" class="wip-value">0</span>
    </div>
    <div class="wip-actual">
      Porcentaje (base):<span id="wip-porcentaje" class="wip-value">0%</span>
    </div>
    <div class="wip-progress-bar">
      <div id="wip-progress-fill" class="wip-progress-fill" style="width: 0%></div>
    </div>
  </div>
</div>
```

CÓDIGO

**SALIDA DEL SPRINT**

## Gestor de Tareas Kanban

Organiza, prioriza y controla el trabajo en progreso con límite flexible

**Nueva Tarea**

Título de la tarea *	Puntos (1-10)	Tiempo estimado (minutos)	Asignado a	Estado inicial
Ej: Revisar documentación	1	60 5-480 min (8h)	Ej: Juan	TODO

**Contadores y WIP**

Total: 9 tareas | TODO: 4 | IN PROGRESS: 1 | DONE: 4

Tiempo total estimado: 540 minutos (9h 0m)

Límite WIP Flexible

Límite base: 4 Actualizar

Límite flexible:  $4 + 10\% = 5$

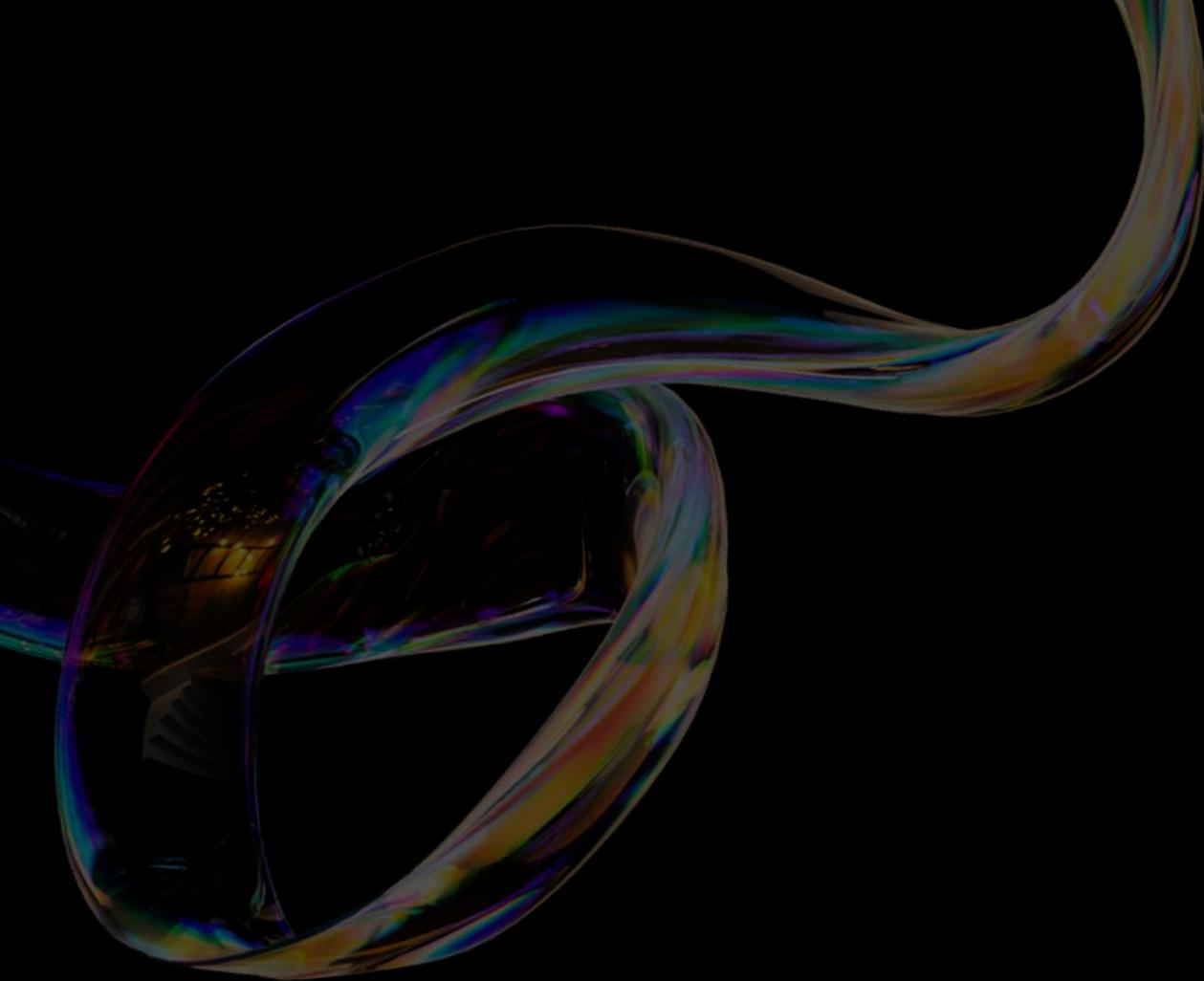
Tareas en progreso: 1

Porcentaje (base): 25%

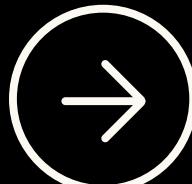
Dentro del límite base

**Kanban Board**

Estado	Nombre	Puntos	Tiempo estimado	Asignado a	Opciones
TODO	b	1 punto	1h 0m	Sin asignar	PROGRESS, DONE, Editar, Eliminar
IN PROGRESS	b	1 punto	1h 0m	Sin asignar	TODO, DONE, Editar, Eliminar
DONE	a	3 puntos	1h 0m	Sin asignar	TODO, PROGRESS, Editar, Eliminar



# QUÉ SE HA CONSTRUIDO (DEMO)



## SPRINT 7

**Añadir fichero con user/pass/nombre**

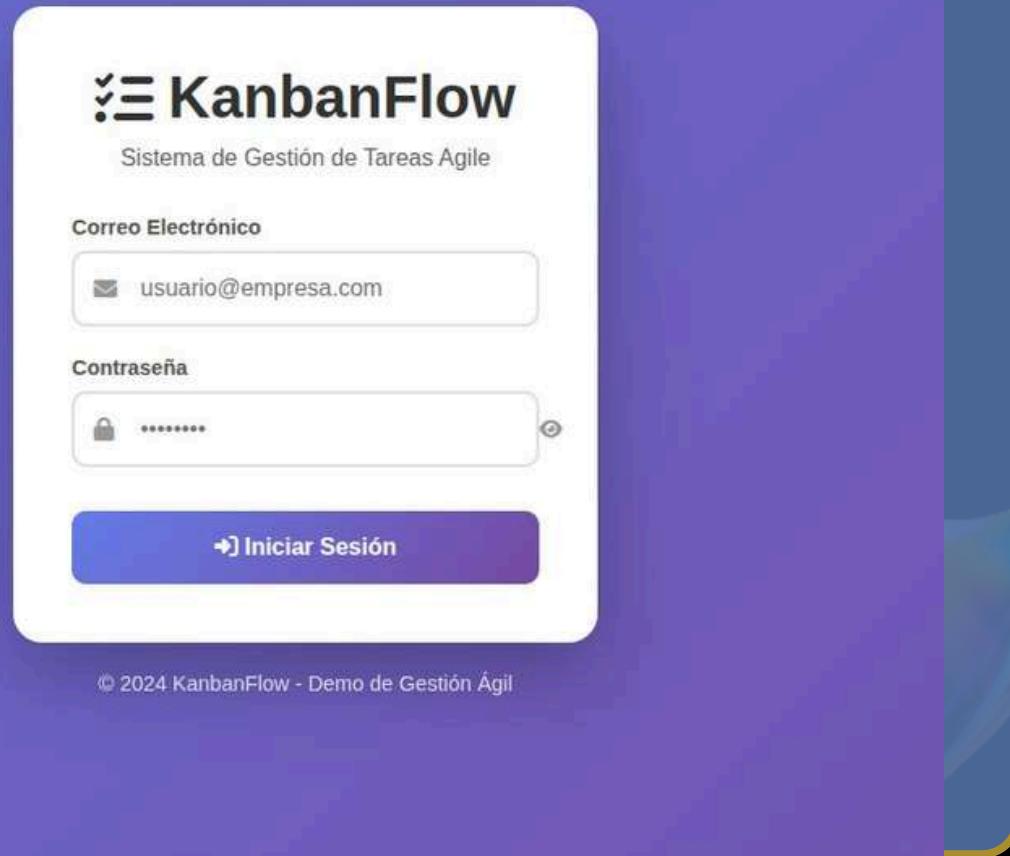
- bbdd/usuarios.txt

**Modificar frontend con desplegable usuarios/equipo**

**Añadir roles al equipo**

- product owner
- normal
- invitado

**Inicio de sesión, usuario**



**KanbanFlow**  
Sistema de Gestión de Tareas Ágil

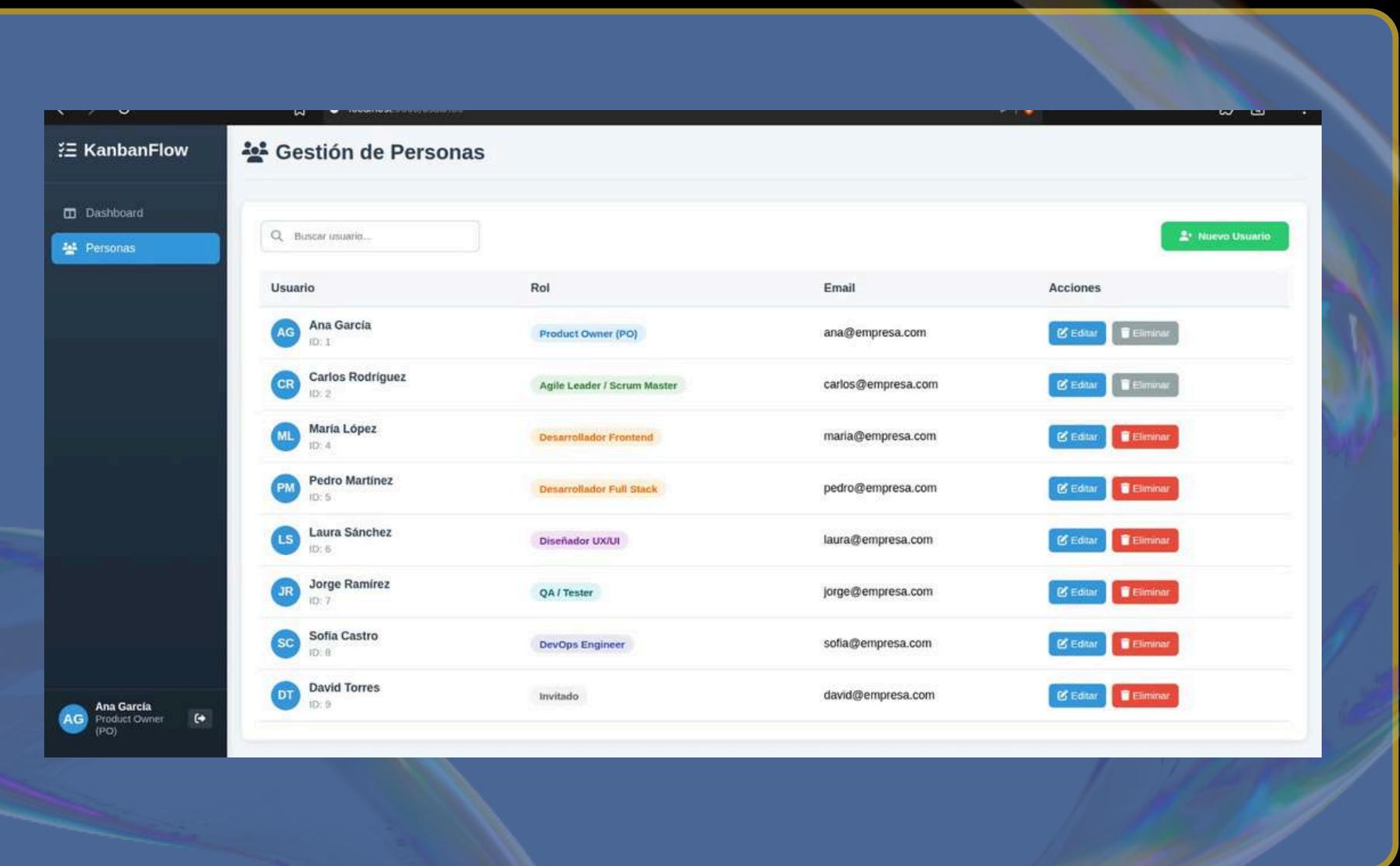
Correo Electrónico

Contraseña

**Iniciar Sesión**

© 2024 KanbanFlow - Demo de Gestión Ágil

## VISUALIZACIÓN INICIO DE SESIÓN



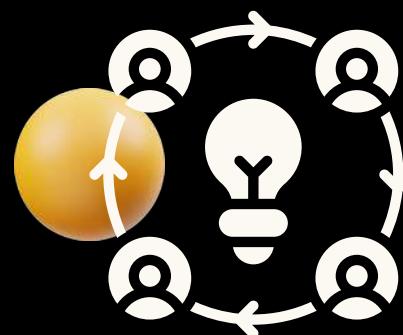
**Gestión de Personas**

Usuario	Rol	Email	Acciones
AG Ana García ID: 1	Product Owner (PO)	ana@empresa.com	<a href="#">Editar</a> <a href="#">Eliminar</a>
CR Carlos Rodriguez ID: 2	Agile Leader / Scrum Master	carlos@empresa.com	<a href="#">Editar</a> <a href="#">Eliminar</a>
ML María López ID: 4	Desarrollador Frontend	maria@empresa.com	<a href="#">Editar</a> <a href="#">Eliminar</a>
PM Pedro Martinez ID: 5	Desarrollador Full Stack	pedro@empresa.com	<a href="#">Editar</a> <a href="#">Eliminar</a>
LS Laura Sánchez ID: 6	Diseñador UX/UI	laura@empresa.com	<a href="#">Editar</a> <a href="#">Eliminar</a>
JR Jorge Ramírez ID: 7	QA / Tester	jorge@empresa.com	<a href="#">Editar</a> <a href="#">Eliminar</a>
SC Sofia Castro ID: 8	DevOps Engineer	sofia@empresa.com	<a href="#">Editar</a> <a href="#">Eliminar</a>
DT David Torres ID: 9	Invitado	david@empresa.com	<a href="#">Editar</a> <a href="#">Eliminar</a>

## TABLERO INICIAL

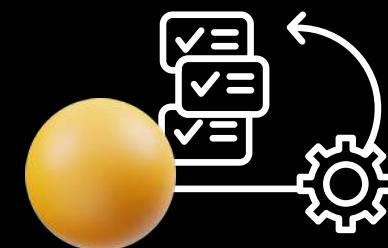
# METODOLOGÍA DE TRABAJO

¿Cómo se ha trabajado con Scrum?



## Gestión en GitHub

- Uso de GitHub Projects para organizar tareas
- Control de versiones mediante commits frecuentes
- Colaboración en repositorio compartido
- Documentación en README y Wiki



## Trabajo con Scrum

- Trabajo en 5 sprints de 1h 10m
- Sprint Planning al inicio de cada iteración
- Daily Scrums breves durante el desarrollo
- Sprint Review con demostración del incremento
- Sprint Retrospective para identificar mejoras

# QUÉ DIFICULTADES HAN ENCONTRADO.

La parte de la programación resultó especialmente complicada, ya que solo una persona comprendía con claridad qué acciones debían realizarse y cómo estructurar el desarrollo. Esto generó una alta dependencia y dificultades para avanzar en equipo, además de que el código presentaba fallas frecuentes, lo que provocó retrabajo constante y retrasos en el progreso del proyecto.



# GRACIAS

