UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

GRADO EN INGENIERÍA INFORMÁTICA

PROYECTO DE INGENIERÍA INFORMÁTICA

ARTEMUS

ISRAEL GÓMEZ

PABLO PIQUERAS

ALDO ZAMORA

XIOAJIE HU

CURSO 2025-2026

Índice

Capítulo 1. RESUMEN DEL PROYECTO

Artemus es un sistema IoT simulado para la monitorización inteligente de parques urbanos. La propuesta modela el Parque Central (3,6 hm2) con sensores virtuales de temperatura, humedad, viento, humo/calidad del aire, accesos (puertas) e iluminación. El sistema se implementa con Python 3 y Flet, usa persistencia local en JSON y sigue un enfoque Model-View-Controller (MVC) con servicios y repositorios.

El resultado es un prototipo funcional con dashboard en tiempo real, mapa interactivo, histórico filtrable, monitor de salud de sensores, gestión de usuarios y solicitudes, y un protocolo de catástrofe activable por el administrador. La arquitectura es modular y mantiene una base sólida para evolucionar hacia hardware real y una base de datos remota.

Capítulo 2. INTRODUCCIÓN

La gestión sostenible de parques urbanos requiere información actualizada sobre condiciones ambientales, aforo y estado de infraestructuras. Artemus plantea una solución ligera y modular que combina simulación de sensores, almacenamiento local y una interfaz gráfica accesible con control por roles. En un contexto académico, la simulación permite validar la arquitectura, la experiencia de usuario y el flujo de datos sin necesidad de hardware físico.

2.1 Competencias trabajadas

- **CT4**: Capacidad de análisis y síntesis.
- **CT5**: Aplicación práctica de los conocimientos.
- **CT6**: Comunicación oral y escrita.
- **CT7**: Conciencia de valores éticos.
- **CT11**: Planificación y gestión del tiempo.
- **CT14**: Innovación y creatividad.
- **CT15**: Responsabilidad.
- **CT17**: Trabajo en equipo.
- **CE8**: Planificación y dirección de proyectos, servicios y sistemas informáticos.
- **CE13**: Selección y uso de estructuras de datos adecuadas.
- **CE14**: Diseño, construcción y mantenimiento de aplicaciones robustas.
- **CE19**: Herramientas para almacenamiento, procesamiento y acceso a sistemas de información.
- **CE20**: Programación paralela, concurrente y distribuida.
- **CE22**: Metodologías y ciclos de vida del software.
- **CE23**: Diseño de interfaces persona-computador con usabilidad y accesibilidad.

2.2 Perfil del usuario principal

Usuario principal propuesto: gestor municipal o responsable operativo del parque. **Necesidades clave:**

- Visualizar en tiempo real el estado del parque (ambiental, aforo, energía).
- Detectar alertas y priorizar intervenciones.
- Coordinar tareas de mantenimiento y gestionar roles de usuario.
- Consultar históricos y trazabilidad de eventos.

Intereses y particularidades:

- Paneles claros y accionables.
- Acceso rápido a incidencias y solicitudes.
- Información fiable y consistente, incluso sin conectividad.

2.3 Estado del arte

Los sistemas Smart Park suelen integrar sensores ambientales, control de aforo y paneles de visualización con analítica básica. En entornos IoT reales, las arquitecturas más habituales son cliente-servidor con almacenamiento en la nube y dashboards con alertas. Artemus adapta esas buenas prácticas al entorno académico mediante sensores simulados y persistencia en JSON, manteniendo la separación por capas que se esperaría en un despliegue real.

Capítulo 3. OBJETIVOS

3.1 Objetivos generales

Diseñar e implementar una aplicación IoT simulada para monitorizar un parque urbano, con interfaz gráfica en tiempo real, persistencia en JSON y gestión de usuarios por roles.

3.2 Objetivos específicos

- Implementar una arquitectura MVC en Python 3 con Flet.
- Simular sensores ambientales y de infraestructura con actualización periódica.
- Persistir datos históricos en ficheros JSON diarios y explotarlos desde la interfaz.
- Diferenciar vistas y permisos según rol (admin, maintenance, user).
- Permitir registro de usuarios y administración completa de cuentas.
- Integrar solicitudes de mantenimiento y control de emergencias.
- Garantizar datos de usuario realistas (DNI, teléfono, dirección) mediante validaciones.

Capítulo 4. DESARROLLO DEL PROYECTO

4.1 Metodología y herramientas empleadas

El trabajo se organizó con metodología ágil tipo Scrum, con planificación por sprints y seguimiento de tareas en un backlog de producto. Las historias de usuario se priorizaron por valor y dependencias técnicas, y se revisaron en cada iteración. Como criterio de finalización se exigió la integración completa del flujo de datos (simulación -> persistencia -> visualización).

Herramientas y tecnologías principales:

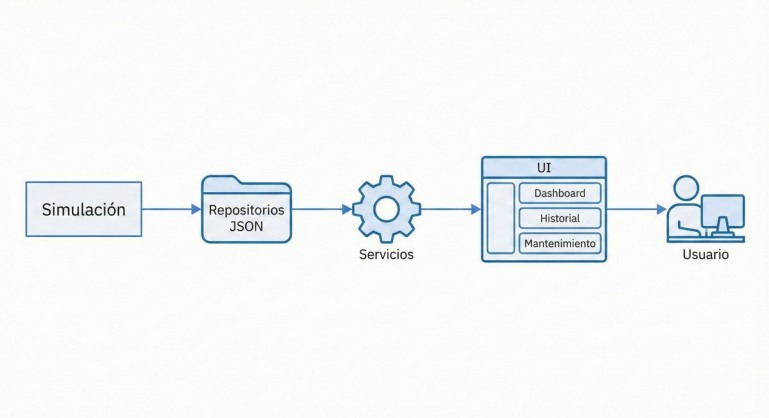
- **Lenguaje y runtime:** Python 3 (requirements.txt).
- **Interfaz gráfica:** Flet (ArtemusPark/main.py y ArtemusPark/view/).
- **Persistencia:** JSON local por tipo y día (ArtemusPark/json/).
- **Concurrencia:** asyncio y hilos para simulación de sensores.
- **Arquitectura:** MVC con servicios y repositorios.
- **Gestión del trabajo:** backlog en Artemus Product Backlog.xlsx y tablero Scrum.

4.2 Arquitectura general

La aplicación sigue un esquema MVC adaptado a simulación IoT:

- **Modelos:** dataclasses para sensores (ArtemusPark/model/).
- **Controladores:** simuladores de sensores (ArtemusPark/controller/).
- **Repositorios:** lectura/escritura de JSON diarios (ArtemusPark/repository/).
- **Servicios:** agregación, histórico y riesgos (ArtemusPark/service/).
- **Vistas:** páginas Flet y componentes (ArtemusPark/view/).
- **Configuración:** sensores, colores y horarios (ArtemusPark/config/).

Flujo de datos:



4.3 Simulación y actualización

Existen dos mecanismos complementarios de simulación:

- **Modo GUI:** sensor_simulation_loop en ArtemusPark/main.py genera snapshots cada 3 segundos con generate_sensor_snapshot, guardando datos de todos los sensores definidos en ArtemusPark/config/Sensor_Config.py. Tras cada ciclo se envía refresh_dashboard por PubSub para actualizar UI y gráficos.
- **Modo simulación completa (hilos):** ArtemusPark/controller/Sensor_Controller.py lanza hilos por sensor con frecuencia de 1s. El script ArtemusPark/view/init.py permite ejecutar esta simulación de forma independiente. Se aplican umbrales de riesgo de viento (ArtemusPark/config/Wind_Config.py) y alarma de humo (ALARM > 95). Cuando se detecta emergencia, se fuerza la apertura del parque.

Otros aspectos relevantes:

- El parque se considera abierto entre 09:00 y 18:00 (ArtemusPark/config/Park_Config.py).
- La iluminación se activa automáticamente entre 19:00 y 07:00 (ArtemusPark/controller/Light_Controller.py).
- Los sensores de puerta generan eventos cada 5s si el parque está abierto (ArtemusPark/controller/Door_Controller.py).
- Los logs de simulación se almacenan en ficheros como door_controller.log y wind_controller.log.

4.4 Roles y acceso

El sistema implementa control de acceso basado en roles (RBAC). Un único login gestiona el acceso y, según el rol, se habilitan vistas y acciones distintas (ArtemusPark/main.py).

4.4.1 Roles definidos en la aplicación

- **Admin:** administración de usuarios y roles, gestión de solicitudes, monitor de energía y protocolo de catástrofe (ArtemusPark/view/pages/Admin_Page.py).
- **Maintenance:** monitor de sensores, lista de sensores asignados y envío de solicitudes (ArtemusPark/view/pages/Maintenance_Page.py).
- **User:** consulta de dashboard y eventos (ArtemusPark/view/pages/Dashboard_Page.py). Teóricamente llevan un dispositivo NFC la cual les identifica al entrar o salir del parque

4.4.2 Registro y gestión de usuarios

- Login único con registro desde ArtemusPark/view/pages/Login_Page.py.
- Validaciones de DNI y teléfono en login y administración (_is_valid_dni).
- El administrador puede crear, editar o eliminar usuarios, asignar sensores y gestionar supervisores/subordinados.

4.4.3 Registro automático de datos

ArtemusPark/repository/Auth_Repository.py crea un conjunto de usuarios por defecto si users.json no existe o está vacío. Esto asegura el cumplimiento de requisitos mínimos de roles y relaciones.

4.5 Interfaz gráfica

La interfaz se organiza en vistas claras y consistentes, accesibles desde una barra lateral (ArtemusPark/view/components/Sidebar.py) que muestra nombre y rol del usuario.

- **Login y registro:** interfaz unificada con validaciones de DNI, teléfono y dirección (ArtemusPark/view/pages/Login_Page.py).
- **Dashboard:** indicadores ambientales, aforo, panel de eventos, gráfico de temperatura y mapa interactivo con marcadores (ArtemusPark/view/pages/Dashboard_Page.py y ArtemusPark/view/components/Map_Card.py).
- **Historial:** tabla de eventos con filtros por rango temporal y orden asc/desc (ArtemusPark/view/pages/History_Page.py).
- **Mantenimiento:** estado global de sensores, sensores asignados resaltados y diálogo de solicitud (ArtemusPark/view/pages/Maintenance_Page.py).
- **Solicitudes:** listado con estados y acciones de aprobación/rechazo para admin (ArtemusPark/view/pages/Requests_Page.py).
- **Administración:** gestión de usuarios, monitor de consumo eléctrico en tiempo real y botón de catástrofe (ArtemusPark/view/pages/Admin_Page.py).

4.6 Datos y sensores

El sistema simula sensores ambientales e infraestructura del parque:

Sensores configurados (ver ArtemusPark/config/Sensor_Config.py):

- **Temperatura (3):** Zona Norte, Zona Sur, Central.
- **Humedad (2):** Jardines, Invernadero.
- **Viento (1):** Torre Principal.
- **Humo (2):** Cafetería, Almacén.
- **Puertas (2):** Torniquete Principal, Acceso Proveedores.
- **Iluminación (2):** Paseo Central, Parking.

Lógica y umbrales destacados:

- **Viento:** WARNING > 20 km/h, RISK > 40 km/h (ArtemusPark/config/Wind_Config.py).
- **Humo:** WARNING > 40, ALARM > 95 (ArtemusPark/controller/Smoke_Controller.py).
- **Aforo:** cálculo por entradas/salidas de puertas (ArtemusPark/service/Dashboard_Service.py), con capacidad máxima 2000 personas.
- **Salud de sensores:** un sensor se considera en línea si ha reportado datos en los últimos 15 segundos (ArtemusPark/service/Dashboard_Service.py).

4.7 Estructura de datos (JSON)

Los ficheros JSON se organizan por tipo de sensor y día:

ArtemusPark/json/{temperature,humidity,wind,smoke,door,light}/<prefijo>_YYYY-MM-DD.json

Convenciones:

- *timestamp* en epoch (segundos).

- Cada archivo almacena un único tipo de dato para evitar bloqueos de escritura.

- Los nombres de sensores se resuelven desde *SENSOR_CONFIG*.

Ejemplos de estructura real:


```
1. {
2.   "sensor_id":"temp_01",
3.   "timestamp":1764831600.0,
4.   "value":24,
5.   "status":"MILD"
6. }
7. {
8.   "sensor_id":"wind_01",
9.   "timestamp":1764833400.0,
10.   "speed":24,
11.   "state":"WARNING"
12. }
13. {
14.   "sensor_id":"smoke_01",
15.   "timestamp":1764832500.0,
```

```
16.   "value":55,
17.   "status":"WARNING"
18. }
19. {
20.   "sensor_id":"door_01",
21.   "timestamp":1764832520.0,
22.   "is_open":true,
23.   "direction":"IN",
24.   "username":"user_demo"
25. }
26. {
27.   "sensor_id":"light_01",
28.   "timestamp":1764832600.0,
29.   "is_on":true,
30.   "status":"ON",
31.   "value":150.0
32. }
```


Usuarios (ArtemusPark/json/users.json):


```
1. {
1.   "admin_super":{
2.     "password":"root2025",
3.     "role":"admin",
4.     "full_name":"Sonia Ortega",
5.     "dni":"22222222B",
6.     "phone":"600222222",
7.     "address":"Avenida Siempre Viva 45",
8.     "subordinates":[
9.       "client_ana",
10.       "visit_tom",
11.       "user_demo"
12.     ]
13.   },
14.   "maint_joe":{
15.     "password":"fixitnow",
16.     "role":"maintenance",
17.     "full_name":"Jose Pardo",
18.     "dni":"55555555E",
19.     "phone":"600555555",
20.     "address":"Paseo de la Castellana 50",
21.     "supervisors":[
22.       "admin_super"
23.     ],
24.     "assigned_sensors":[
25.       "temp_01",
26.       "hum_01",
27.       "door_01"
28.     ]
29.   }
30. }
```

Solicitudes (ArtemusPark/json/requests.json):


```
[
  {
    "id":1764832600123,
    "user":"maint_joe",
    "type":"sensor_change",
    "message":"Solicito añadir el sensor de puerta norte.",
    "status":"PENDING",
    "timestamp":1764832600.0
  }
]
```

4.8 Requisitos técnicos

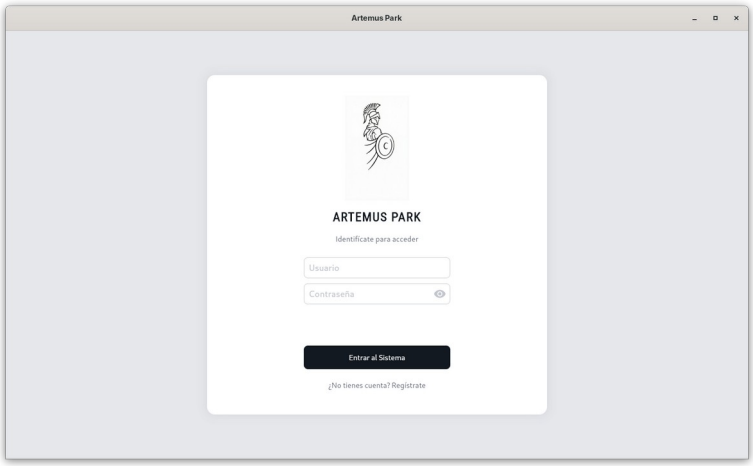
- Python 3.x.
- Flet 0.28.3 y flet-desktop 0.28.3 (requirements.txt).
- Sistema operativo: Windows, Linux o macOS.
- Rutas relativas para assets y JSON (assets/, ArtemusPark/json/).
- Resolución recomendada: ancho mínimo 1420 y alto mínimo 800 (ArtemusPark/main.py).

4.9 Pruebas realizadas

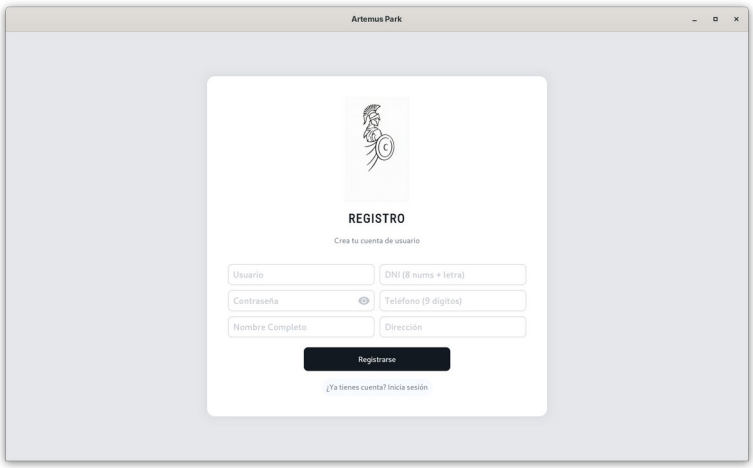
- Registro y login con validaciones de DNI y teléfono (Login_Page y Admin_Page).
- Acceso por rol y navegación por vistas (main.py y Sidebar).
- Actualización en tiempo real mediante PubSub (main.py y Dashboard_Page).
- Historial con filtros 1 mes / 1 semana / 1 día (History_Page).
- Monitor de sensores y solicitudes de cambio (Maintenance_Page).
- Gestión de solicitudes y aprobaciones (Requests_Page).
- Gestión de usuarios y protocolo de catástrofe (Admin_Page).
- Persistencia JSON y lecturas de histórico (ArtemusPark/repository/).

4.10 Capturas de pantalla

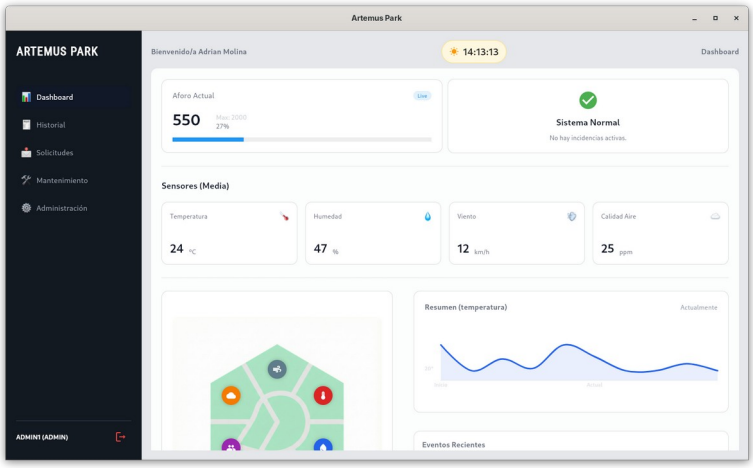
Capturas de pantallas realizadas con un usuario de rol *admin:*



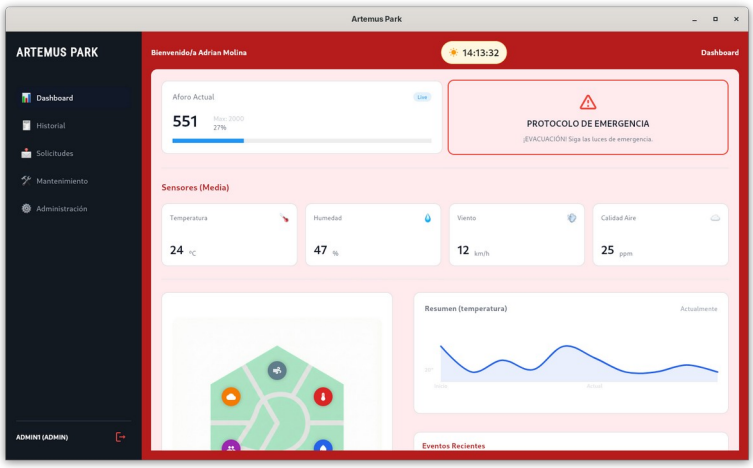
Pantalla de login



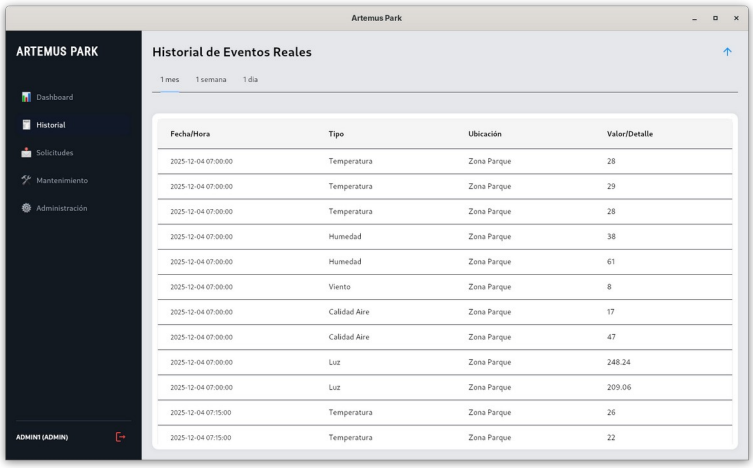
Pantalla de registro



Dashboard inicial (Única vista para el rol usuario)

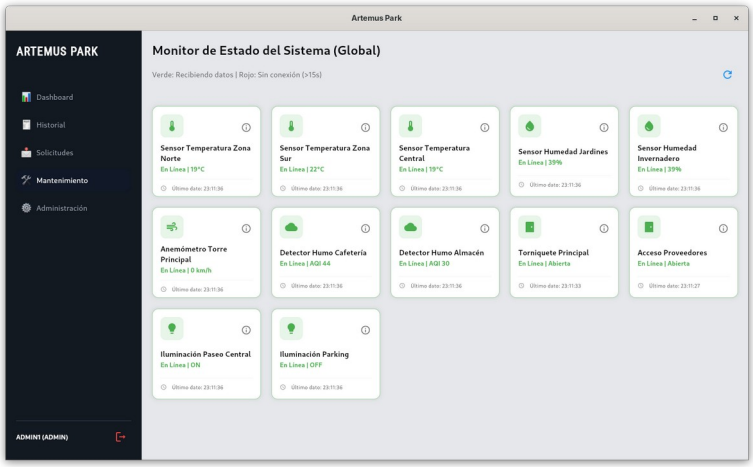


Dashboard inicial en protocolo de emergencia

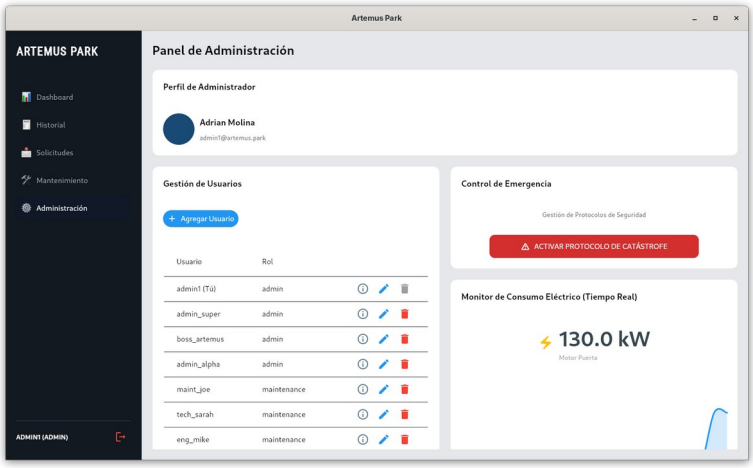


Histórico de eventos

Pantalla de gestión de solicitudes / Solicitudes



Pantalla de monitor de Estado del Sistema



Pantalla de panel de administración (Solo administradores)

Capítulo 5. Planificación del proyecto

El proyecto se planificó con sprints de 2 semanas y seguimiento en Scrum. La planificación típica quedó estructurada en fases, cuyo alcance funcional está determinado por las Historias de Usuario detalladas en el *Artemus Product Backlog.xlsx*:

- **Sprint 0 (idea y análisis):** estado del arte, perfil de usuario, anteproyecto y mockups.
- **Sprint 1 (arquitectura):** estructura MVC, configuración base y UI inicial.
- **Sprint 2 (sensores y JSON):** simulación de datos, repositorios y histórico.
- **Sprint 3 (roles y UI):** login/registro, vistas por rol, solicitudes y mantenimiento.
- **Sprint 4 (integración):** ajustes de interfaz, monitor de energía, pruebas y documentación.

Roadmap PII2 (futuro): integración con BBDD remota, sensores físicos (Arduino/Raspberry), mensajería interna y análisis estadístico avanzado.

Para la gestión operativa y la trazabilidad de dichas historias, se emplea un sistema de tablero **KANBAN** (implementado a través de la herramienta **Trello**). Esto permite al equipo monitorizar eficazmente el flujo de trabajo, el estado de las tareas y el progreso general del desarrollo en tiempo real.

A continuación, se presenta el **cronograma** temporal mediante el diagrama de *Gantt*

Capítulo 6. PRESUPUESTO

Estimación académica basada en esfuerzo y recursos utilizados.

Tipo de coste	**Valor estimado**	**Comentarios**
Horas de trabajo (equipo)	240 h ~ 3.600 EUR	4 personas x 60 h, 15 EUR/h.
Equipo técnico	3.000 EUR	4 portátiles (~750 EUR) como valor de mercado.
Software	0 EUR	Python, Flet, PyCharm Community.
Materiales	0 EUR	Simulación sin hardware físico.
Total estimado	**6.600 EUR**	Estimación académica sin inversión real.

Capítulo 7. Equipo de trabajo

- **Pablo Piqueras (Scrum Master):** coordinación de sprints, integración general y simulación de sensores.
- I**srael Gómez (Product Owner):** definición de requisitos, UX/UI, validaciones y lógica de negocio.
- **Aldo Zamora (Hardware/IoT):** investigación de sensores y proyección a hardware real.
- **Xiaojie Hu (QA/Documentación):** pruebas funcionales, validación y documentación técnica.

REFERENCIAS Y BIBLIOGRAFIA

- Guia de Anteproyecto.docx (estructura y estilo).
- Lista de requisitos Proyecto.pdf (requisitos docentes).
- PropuestaProyectoII.pdf (competencias y objetivos).
- README.md (contexto del proyecto).
- Documentación oficial de Python 3: https://docs.python.org/3/
- Documentación oficial de Flet: https://flet.dev/