# Comparativa de Tipos de Arquitectura de Software

Proyecto: E Urbana (Sistema de Alumbrado Inteligente)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Tipo de Arquitectura | Descripción | Ventajas | Desventajas | Aplicabilidad en E Urbana | Viabilidad |
| Centrada en Datos | Enfocada en un almacenamiento central al que acceden otros componentes. | Facilita la centralización de datos, adecuada para sistemas con alta dependencia de información. | Puede generar cuellos de botella, dependencia del almacenamiento central. | Útil si el sistema depende de un repositorio central de datos (como sensores que almacenan en un servidor). | Media |
| Flujo de Datos | Transformaciones secuenciales de datos mediante componentes conectados. | Modularidad, procesamiento continuo y eficiente de datos. | Difícil manejar errores y control de flujo complejo. | Ideal si los datos de sensores fluyen constantemente y deben procesarse en etapas. | Alta |
| Llamada y Retorno | Módulos interactúan mediante llamadas y retornos (Cliente - Servidor). | Simplicidad, fácil de entender e implementar. | Dificultad para escalar y mantener en sistemas grandes. | Puede usarse para comunicaciones simples entre interfaz y servidor central. | Media |
| Orientada a Objetos | Componentes como objetos con datos y comportamientos comunicándose por mensajes. | Reusabilidad, mantenibilidad y claridad en la modelación. | Puede complejizar el diseño si no se gestiona bien. | Útil si se diseña un sistema modular y extensible con clases de dispositivos y controladores. | Alta |
| En Capas | Organización en niveles donde cada capa comunica con las adyacentes. | Separación de responsabilidades, facilidad de mantenimiento y pruebas. | Puede aumentar la latencia, dependencia entre capas. | Recomendable para dividir interfaz, lógica de negocio, control y datos. | Muy Alta |

Conclusión:

Para el proyecto E Urbana, la arquitectura en capas es la más viable debido a su alta organización, separación clara de responsabilidades y escalabilidad, lo cual es esencial en sistemas inteligentes distribuidos como el de alumbrado urbano.

Caso de Estudio "Ecoluz Urbana"

# Plataforma WEB para la Gestión Inteligente del Alumbrado Público

## Integrantes del proyecto:

* Luis Iván Marquez Azuara
* Brayn Kalid Reyes Silva
* Aldo Tolentino Domingo
* Angel David Reyes Telléz

# Plan del Proceso de Desarrollo WEB (Metodología Ágil: Shape Up)

Se adoptará la metodología Shape Up, que se estructura en ciclos de 6 semanas, enfocados en el desarrollo de funcionalidades completas, seguidos de una etapa de buffer (2 semanas) para correcciones, pruebas y refinamientos.

## Ciclo 1 (Semanas 1-6)+

* Desarrollo de la API REST (Node.js + Express)
* Implementación de la base de datos (MongoDB Atlas)
* Autenticación mediante JWT
* Pruebas unitarias y de integración

## Ciclo 2 (Semanas 7-12)

* Desarrollo del Dashboard administrativo (React.js)
* Integración con Leaflet.js para mapas
* Visualización de gráficas con Chart.js
* Exportación de reportes (CSV/PDF)
* Pruebas de usabilidad

## Buffer (Semanas 13-14)

* Optimizaciones de UI/UX
* Ajustes funcionales
* Documentación técnica

# Justificación de la Arquitectura

Se utiliza una arquitectura en capas, que permite separar responsabilidades y facilitar la escalabilidad, el mantenimiento y la seguridad del sistema. Esta elección se basa en la necesidad de:

* **Modularidad:** Cada capa tiene una responsabilidad específica, lo que facilita el desarrollo y el mantenimiento independiente de cada componente.
* **Escalabilidad:** Permite escalar componentes individuales (por ejemplo, el backend) sin afectar otras partes del sistema.
* **Mantenibilidad:** Los cambios en una capa tienen un impacto mínimo en las demás, simplificando la resolución de problemas y la implementación de nuevas funcionalidades.
* **Seguridad:** La separación de capas, especialmente entre la presentación y los datos, ayuda a implementar controles de seguridad más robustos.
* **Flexibilidad:** Facilita la adaptación a futuros cambios tecnológicos o requerimientos, como la integración de nuevas tecnologías IoT o interfaces de usuario.

## Capas Principales:

* **Capa de Presentación (Frontend):** React.js para la interfaz de administradores y Kotlin Multiplatform para las aplicaciones móvil y de smartwatch. Es responsable de la interacción con el usuario y la visualización de la información.
* **Capa de Lógica de Negocio (Backend):** Node.js con Express para el manejo de reglas de negocio, validaciones, procesamiento de datos y definición de rutas de la API. Actúa como intermediario entre la capa de presentación y la capa de persistencia/IoT.
* **Capa de Persistencia (Base de Datos):** MongoDB Atlas para el almacenamiento de datos estructurados y no estructurados de luminarias, sensores, usuarios, etc.
* **Capa IoT:** Involucra los Sensores Eco Meter y la Raspberry Pi que actúan como pasarela para la comunicación vía MQTT, recolectando y enviando datos al backend.

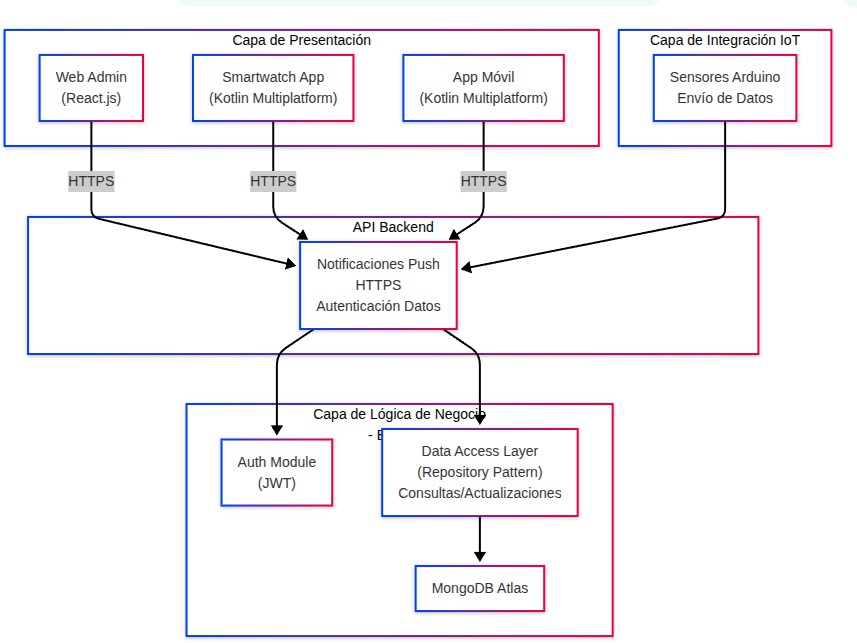
Esta arquitectura facilita la recolección, procesamiento, y distribución eficiente de información entre los dispositivos IoT, el backend y las interfaces web/móvil/smartwatch, lo que es fundamental para una gestión inteligente del alumbrado público.

# Diagrama de la Arquitectura

Para ilustrar la arquitectura de "Ecoluz Urbana", presentaremos tres tipos de diagramas que resaltan diferentes aspectos del sistema:

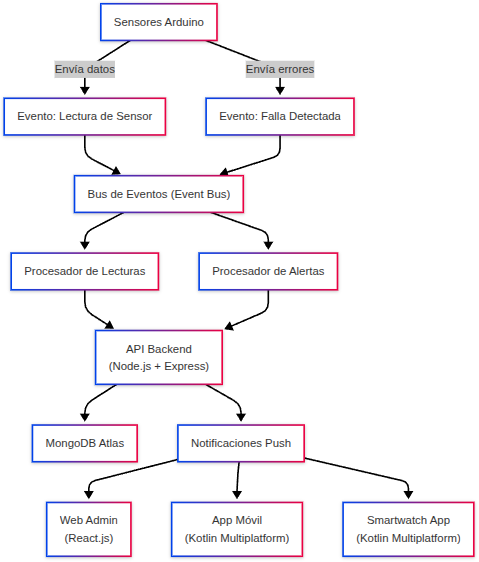
## Diagrama de Arquitectura de Capas (Layered Architecture Diagram)

Este diagrama es fundamental para "Ecoluz Urbana" ya que la justificación de la arquitectura se basa en un modelo en capas. Muestra la separación lógica del sistema en niveles horizontales, donde cada capa tiene responsabilidades específicas y se comunica principalmente con las capas adyacentes.



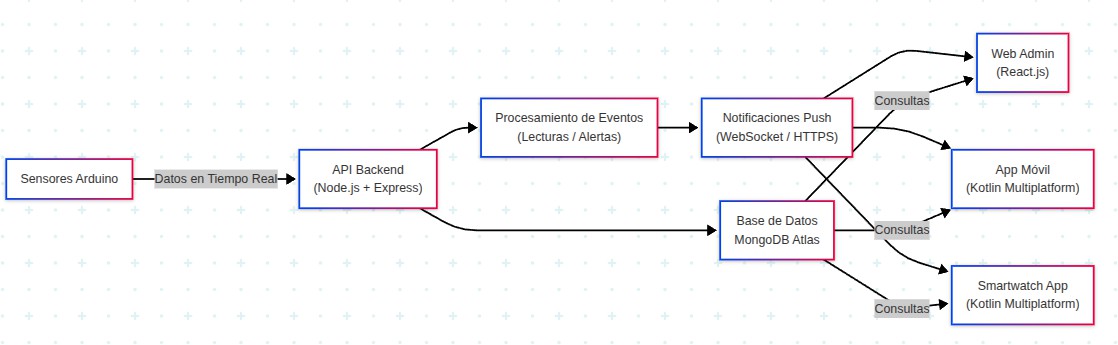
## Diagrama de Arquitectura Orientada a Eventos (Event-Driven Architecture Diagram)

Dado que el proyecto maneja datos en tiempo real de sensores IoT y genera notificaciones, una arquitectura orientada a eventos es una excelente forma de visualizar cómo los "eventos" (lecturas de sensores, fallos) disparan acciones dentro del sistema.



## Diagrama de Flujo de Información (Information Flow Diagram)

Este diagrama se centra en el camino que sigue la información a través del sistema, desde su origen en los sensores hasta su visualización y control en las interfaces de usuario. Es más conceptual y fácil de entender rápidamente.



# Propuesta de Patrones de Diseño

* **MVC (Model-View-Controller):** Aplicado en el frontend (React.js) y backend (Node.js + Express) para la separación de la lógica de negocio, la presentación de datos y la interacción del usuario, promoviendo mayor claridad y modularidad.
* **Repository Pattern:** Utilizado en el backend (Node.js) para encapsular la lógica de acceso a datos de MongoDB Atlas, proporcionando una abstracción limpia entre la capa de negocio y la de persistencia. Esto facilita el cambio de la base de datos subyacente si fuera necesario y mejora la capacidad de prueba.
* **Observer Pattern:** Implementado en el backend para el envío de alertas y notificaciones push a los dispositivos (smartwatch y aplicación móvil) ante fallos o anomalías detectadas en las luminarias. Permite que múltiples observadores reaccionen a eventos específicos.
* **Singleton Pattern:** Para la gestión de configuraciones globales, como la conexión a MongoDB Atlas o las claves secretas para JWT. Asegura que solo exista una instancia de estos recursos compartidos en toda la aplicación.
* **Shared Module (Kotlin Multiplatform):** En las aplicaciones móvil y de smartwatch, se utilizará un módulo compartido de Kotlin Multiplatform para reutilizar la lógica de negocio común, modelos de datos y posiblemente parte de la lógica de red, reduciendo la duplicación de código y facilitando el mantenimiento en ambas plataformas.

1. **Justificación de Frameworks de Desarrollo WEB a Utilizar**

|  |  |  |
| --- | --- | --- |
| Componente | Framework / Lenguaje | Justificación |
| Backend | Node.js + Express | Alto rendimiento y escalabilidad: Node.js es ideal para aplicaciones de red escalables y de alta concurrencia gracias a su modelo de E/S no bloqueante y asíncrono, lo que es crucial para manejar el flujo constante de datos de sensores IoT. Amplio ecosistema: npm (Node Package Manager) ofrece una vasta colección de librerías y herramientas, acelerando el desarrollo. |
| Frontend | React.js | Desarrollo rápido y basado en componentes: Permite construir interfaces de usuario complejas a partir de componentes reutilizables y declarativos, lo que agiliza el desarrollo y mejora la mantenibilidad.  Gran comunidad y ecosistema: Amplio soporte, documentación y herramientas disponibles, lo que facilita la resolución de problemas y el aprendizaje. |
| Base de Datos | MongoDB Atlas | Flexibilidad con datos no estructurados: Al ser una base de datos NoSQL basada en documentos, es altamente flexible para almacenar datos de sensores IoT que pueden variar en estructura. Escalabilidad automática: MongoDB Atlas ofrece escalabilidad horizontal y alta disponibilidad como servicio gestionado en la nube, simplificando la administración de la infraestructura de datos. |
| Mapas | Leaflet.js | Biblioteca ligera y eficiente: Diseñada para ser ligera y de alto rendimiento, ideal para integración en aplicaciones web. Open Source y extensible: Ofrece una gran flexibilidad y puede ser extendida con numerosos plugins para funcionalidades avanzadas de geolocalización y visualización de marcadores interactivos. Es compatible con OpenStreetMap. |
| Gráficas | Chart.js | Integración sencilla con React: Permite crear visualizaciones de datos atractivas e intuitivas con facilidad. Personalizable y responsivo: Ofrece una amplia gama de tipos de gráficos y opciones de personalización, además de ser responsivo para adaptarse a diferentes tamaños de pantalla. |
| Móvil/Smartwat ch | Kotlin Multiplatform | Desarrollo multiplataforma nativo: Permite compartir código entre Android, iOS (y Wear OS), reduciendo el tiempo y costo de desarrollo sin sacrificar la experiencia de usuario nativa. Rendimiento: Compila a código nativo, asegurando un rendimiento óptimo en cada plataforma. |
|  |  |  |

1. **Esquema de Pruebas**

|  |  |  |
| --- | --- | --- |
| Tipo de prueba | Herramientas | Objetivo |
| Unitarias | Jest (React), JUnit (Kotlin) | Validar el correcto funcionamiento de componentes individuales, funciones y métodos de forma aislada. Asegurar la lógica interna. |
| Integración | Postman, Insomnia | Verificar la comunicación y el flujo de datos entre diferentes módulos o servicios (e.g., API Backend y Base de Datos, Frontend y API). |
| Rendimiento | JMeter | Simular cargas de usuarios concurrentes y el flujo de datos de múltiples sensores para evaluar la capacidad de respuesta y estabilidad del sistema bajo estrés. |
| Usabilidad | Firebase Test Lab, Pruebas manuales | Evaluar la facilidad de uso, la intuición de la interfaz de usuario y la experiencia general del usuario en la plataforma web, móvil y smartwatch. |
| Seguridad | OWASP ZAP | Identificar y validar vulnerabilidades comunes de seguridad en la aplicación web y API, siguiendo las directrices del OWASP Top 10. |
| End-to-End (E2E) | Cypress (o similar) | Simular el flujo de usuario completo a través de todo el sistema, desde la interacción en el frontend hasta la persistencia de datos. |

1. **Requerimientos Funcionales (RF)**

* **RF-01: Autenticación de Administradores:** El sistema debe permitir el inicio de sesión de administradores utilizando tokens JWT para la gestión segura.
* **RF-02: Gestión de Luminarias y Sensores:** El sistema debe permitir el registro, modificación y eliminación de información de luminarias y sus sensores asociados.
* **RF-03: Visualización de Datos Energéticos:** La plataforma debe mostrar datos históricos y en tiempo real del consumo energético por luminaria y área.
* **RF-04: Dashboard con Mapas Interactivos:** El dashboard administrativo debe integrar mapas interactivos (Leaflet.js) para la visualización de la ubicación y estado de las luminarias mediante marcadores.
* **RF-05: Recepción de Datos en Tiempo Real (IoT):** El backend debe ser capaz de recibir y procesar datos en tiempo real provenientes de los sensores Eco Meter vía MQTT.
* **RF-06: Filtrado de Datos:** La plataforma debe permitir filtrar datos energéticos e históricos por fecha, rango de tiempo, zona geográfica o luminaria específica.
* **RF-07: Historial de Mantenimiento:** El sistema debe mantener un registro detallado del historial de mantenimiento realizado en cada luminaria.
* **RF-8: Visualización de Consumo en Smartwatch:** El smartwatch debe mostrar el consumo energético actual y el estado operativo de las luminarias cercanas.
* **RF-9: Panel de Alertas y Anomalías:** El dashboard debe presentar un panel centralizado para visualizar y gestionar alertas y anomalías detectadas en el sistema.
* **RF-10: Gestión de Usuarios y Roles:** El sistema debe permitir la creación, modificación y gestión de usuarios con diferentes niveles de acceso y roles (administrador, operador, etc.).
* **RF-11: Consulta de Estado Operativo:** Los usuarios deben poder consultar el estado operativo (encendido/apagado, funcionando/fallo) de las luminarias desde la plataforma web y la aplicación móvil.
* **RF-12: Enlace IoT-Base de Datos:** El sistema debe asegurar una conexión fiable y el correcto almacenamiento de los datos transmitidos desde los sensores IoT a la base de datos.

# Requerimientos No Funcionales (RNF)

* **RNF-01: Rendimiento de la API:** El tiempo de respuesta de la API para consultas estándar (ej. consulta de datos de una luminaria) debe ser menor a 500ms.
* **RNF-02: Seguridad de Datos en Tránsito:** Toda la comunicación entre los clientes (web, móvil, smartwatch) y el backend debe ser cifrada mediante TLS 1.3 para proteger la confidencialidad e integridad de los datos.
* **RNF-03: Disponibilidad del Sistema:** El sistema debe garantizar una disponibilidad mensual igual o superior al 99%.
* **RNF-04: Escalabilidad del Backend:** La arquitectura del backend debe permitir la escalabilidad horizontal para soportar un aumento en la carga de usuarios y datos de sensores.
* **RNF-05: Compatibilidad de Navegadores:** La interfaz web debe ser totalmente compatible y funcionar correctamente en las últimas versiones de Chrome, Firefox y Edge.
* **RNF-06: Control de Acceso:** El acceso a las funcionalidades del sistema debe estar restringido y gestionado mediante un sistema de roles y permisos.
* **RNF-07: Compatibilidad de App Móvil (Android):** La aplicación móvil debe ser compatible con dispositivos Android 10 (API 29) y versiones posteriores.
* **RNF-08: Compatibilidad de Smartwatch:** La aplicación para smartwatch debe ser compatible con dispositivos Wear OS 3+ y posteriores.
* **RNF-09: Optimización de Consultas (MongoDB):** Se deben implementar índices apropiados en MongoDB para optimizar las consultas frecuentes y mejorar los tiempos de respuesta.
* **RNF-10: Concurrencia de Dispositivos:** El sistema debe ser capaz de soportar la conexión y el procesamiento de datos de un mínimo de 10,000 dispositivos IoT concurrentemente.
* **RNF-11: Respaldo de Base de Datos:** Se debe implementar una estrategia de respaldo automático semanal de la base de datos para prevenir la pérdida de información.
* **RNF-12: Logs de Actividad:** El sistema debe generar logs detallados de las actividades de los usuarios y eventos del sistema para auditoría y depuración.
* **RNF-13: Tolerancia a Fallos:** El sistema debe estar diseñado para ser tolerante a fallos en componentes individuales, asegurando que un fallo en una parte no detenga todo el sistema.
* **RNF-14: Interfaz Web Responsiva:** La interfaz del dashboard web debe ser responsiva, adaptándose y mostrando correctamente la información en diferentes tamaños de pantalla y dispositivos.
* **RNF-15: Integración Continua:** Se implementará un pipeline de Integración Continua (CI) utilizando GitHub Actions para automatizar las pruebas y la construcción del proyecto.
* **RNF-16: Documentación Actualizada:** La documentación técnica del proyecto, incluyendo especificaciones y manuales, se mantendrá actualizada en Notion.
* **RNF-17: Tiempo de Carga del Dashboard:** El dashboard administrativo debe cargarse completamente en menos de 2 segundos en condiciones de red estándar.
* **RNF-18: Estándares de Codificación Segura:** Todo el código de la aplicación debe adherirse a los estándares de seguridad recomendados por OWASP para prevenir vulnerabilidades comunes.
* **RNF-19: Accesibilidad Web:** La interfaz web debe cumplir con los principios de Accesibilidad para Contenido Web (WCAG) en el nivel AA, garantizando que sea utilizable por personas con diversas capacidades.
* **RNF-20: Cumplimiento Normativo:** El sistema debe estar diseñado considerando el cumplimiento con la NOM-001-SEDE-2012 (Instalaciones Eléctricas) y la norma ISO 50001 (Sistemas de Gestión de la Energía), donde aplique.

Estimación de Esfuerzo del Proyecto EcoLuz Urbana usando Puntos de Casos de Uso (UCP)

**Nombre del proyecto:** EcoLuz Urbana

**Equipo:** Luis Iván Márquez Azuara, Aldo Tolentino Domingo, Brayn Kalid Reyes Silva, Ángel David Reyes Téllez

**Universidad:** Universidad Tecnológica de Xicotepec de Juárez

**Fecha:** Junio 2025

# Paso 1: Cálculo de Puntos de Casos de Uso sin Ajustar (UUCP)

Casos de Uso Identificados:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de uso** | **Tipo** | **Pes o** | **Cantidad** | **Total** |
| Monitoreo de luminarias | Medio | 10 | 1 | 10 |
| Control automático de intensidad | Medio | 10 | 1 | 10 |
| Mantenimiento predictivo | Complejo | 15 | 1 | 15 |
| Generación de informes | Medio | 10 | 1 | 10 |
| Visualización y gestión vía plataforma | Complejo | 15 | 1 | 15 |
| Reporte ciudadano de fallas | Medio | 10 | 1 | 10 |
| **Total UUCW:** 70 |  |  |  |  |

Actores Identificados:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Actor** | **Tipo** | **Pes o** | **Cantidad** | **Total** |
| Administrador Municipal | Complejo (GUI) | 3 | 1 | 3 |
| Sistema de sensores (Eco Meter) | Simple (API) | 1 | 1 | 1 |
| Plataforma web | Medio (protocolo web) | 2 | 1 | 2 |
| **Total UAW:** 6 |  |  |  |  |

UUCP = UUCW + UAW = 70 + 6 = 76

# Paso 2: Cálculo del TCF (Technical Complexity Factor)

Fórmula: TCF = 0.6 + (0.01 × Σ factores técnicos)

|  |  |  |  |
| --- | --- | --- | --- |
| **Factor** | **Pes o** | **Impacto percibido** | **Resultad o** |
| T1 Sistema distribuido | 2 | 4 | 8 |
| T2 Rendimiento | 1 | 5 | 5 |
| T3 Eficiencia del usuario final | 1 | 4 | 4 |
| T4 Procesamiento interno complejo | 1 | 5 | 5 |
| T6 Instalación | 0.5 | 3 | 1.5 |
| T7 Facilidad de uso | 0.5 | 5 | 2.5 |
| T8 Portabilidad | 2 | 4 | 8 |
| T11 Seguridad | 1 | 3 | 3 |
| **Suma TCF:** 37.0 |  |  |  |
| **TCF = 0.6 + (0.01 × 37) = 0.97** |  |  |  |

# Paso 3: Cálculo del ECF (Environmental Complexity Factor)

Fórmula: ECF = 1.4 + (-0.03 × Σ factores ambientales)

|  |  |  |  |
| --- | --- | --- | --- |
| **Factor** | **Pes o** | **Valor** | **Resultad o** |
| E1 Familiaridad con UML | 1.5 | 4 | 6.0 |
| E3 Capacidad del líder | 0.5 | 5 | 2.5 |
| E5 Experiencia OO | 1 | 5 | 5.0 |
| E6 Motivación | 1 | 5 | 5.0 |
| E8 Estabilidad de requerimientos | 2 | 3 | 6.0 |
| **Suma ECF:** 24.5 |  |  |  |
| **ECF = 1.4 + (-0.03 × 24.5) = 0.665** |  |  |  |

# Paso 4: Cálculo de UCP (Use Case Points)

UCP = UUCP × TCF × ECF = 76 × 0.97 × 0.665 = 49.14

# Paso 5: Estimación de horas-hombre (con PF = 20)

Total estimado: 49.14 × 20 = 982.8 horas-hombre

**Resumen Final**

|  |  |
| --- | --- |
| **Concepto** | **Valor** |
| UUCP (sin ajustar) | 76 |
| TCF | 0.97 |
| ECF | 0.665 |
| UCP | 49.14 |
| Productividad (PF) | 20 horas/UCP |
| Total de horas estimadas | **~983**  **horas-hombre** |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Característica / Herramienta*** | **Confluence** | **Google Workspace** | **Microsoft 365 (SharePoint)** | **Notion** | **GitBook** |
| ***Enfoque principal*** | Documentación colaborativa empresarial | Colaboración en documentos en tiempo real | Gestión de documentos y colaboración | Notas, wikis, gestión de tareas y bases de datos | Documentación técnica y wikis |
| ***Colaboración en tiempo real*** | Sí | Sí | Sí | Sí | Sí |
| ***Control de versiones*** | Sí (versionado de páginas) | Sí (con Historial de versiones en Docs, etc.) | Sí (control de versiones en documentos) | Limitado (historial de cambios) | Sí (con historial claro y controlado) |
| ***Integraciones*** | Atlassian Suite, Jira, Slack | Gmail, Drive, Meet, otras de Google | Teams, Outlook, Office apps | Slack, GitHub, Zapier, otros | GitHub, Slack, herramientas de desarrollo |
| ***Permisos y***  ***control de acceso*** | Avanzado por espacios/página s | Básico a intermedio (según Google Drive) | Muy avanzado (por grupos, sitios, etc.) | Intermedio (por página o espacio) | Avanzado (por espacios o grupos) |
| ***Facilidad de uso*** | Interfaz profesional, curva media | Muy intuitiva y conocida | Curva media, más corporativa | Muy amigable y flexible | Enfocada en desarrolladores, interfaz clara |
| ***¿Ofrece plantillas?*** | Sí | Sí | Sí | Sí | Sí |
| ***Búsqueda avanzada*** | Sí | Sí | Sí | Sí | Sí |
| ***Ideal para*** | Equipos técnicos, empresas grandes | Cualquier tipo de equipo | Empresas grandes con estructura IT | Equipos creativos, startups, freelancers | Equipos técnicos, proyectos de documentación |
| ***Precio (plan base)*** | Pago (plan gratuito limitado) | Gratuito con cuenta Google, planes pagos | Pago (Office incluido en licencias) | Gratuito (limitado), planes pagos | Gratuito (limitado), planes pagos |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Característica / Herramienta*** | **Power BI** | **Tableau** | **Google Data Studio** | **Jira Dashboards** | **Excel Avanzado** |
| ***Enfoque principal*** | Inteligencia empresarial y análisis | Visualización interactiva y BI | Visualización de datos en la nube | Seguimiento de proyectos y KPIs en Jira | Análisis de datos y dashboards |
| ***Interactividad***  ***de dashboards*** | Alta (filtrado, clics, segmentaciones  ) | Alta (muy dinámica) | Alta (con filtros, controles interactivos) | Media (widgets y filtros predeterminados  ) | Media-Alta (con macros, tablas dinámicas) |
| ***Facilidad de***  ***uso*** | Interfaz amigable, curva media | Profesional, curva de aprendizaje mayor | Muy intuitiva, parecida a Google Sheets | Intuitiva si conoces Jira | Familiar para la mayoría, curva media |
| ***Integraciones*** | Excel, SQL, APIs, Google, Azure, etc. | Excel, bases de datos, servicios cloud | Google Sheets, BigQuery, otros productos Google | Jira apps, Confluence, plugins externos | Amplia (SQL, Power Query, Add-ins) |
| ***Automatizació n de datos*** | Sí (Power Query, flujos de datos) | Sí (actualizacione s programadas) | Sí (actualización en tiempo real con Google sources) | Limitada (basado en actividades del proyecto) | Sí (Power Query, VBA, macros) |
| ***Colaboración*** | Compartir en la nube o en Teams | Tableau Server o Public | Compartir fácilmente por enlace | Dentro del entorno Jira | Por correo, red o SharePoint |
| ***Curva de aprendizaje*** | Media | Alta | Baja | Baja si ya se usa Jira | Baja-media |
| ***Costo*** | Gratuito (Desktop) / Pago (Pro, Premium) | Pago (Tableau Public es gratuito limitado) | Gratuito | Incluido en Jira (planes pagos) | Incluido en Microsoft 365 o licencia única |
| ***Ideal para*** | Empresas que buscan BI robusto | Analistas de datos, grandes empresas | Usuarios Google, proyectos colaborativos | Equipos ágiles y de desarrollo | Usuarios frecuentes de Excel, analistas |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Característica / Herramienta*** | **Microsoft Teams** | **Zoom** | **Slack (Canales de voz)** | **Google Meet** | **Webex** |
| ***Tipo de comunicación*** | Chat, llamadas, videollamadas, canales | Videollamadas y llamadas | Chats, canales de voz y videollamadas (con apps) | Videollamadas, chat | Videollamadas, llamadas y chat |
| ***Enfoque principal*** | Trabajo en equipo y colaboración | Reuniones virtuales y webinars | Comunicación ágil en equipos | Reuniones rápidas y educación | Reuniones corporativas |
| ***Videollamada s grupales*** | Sí (alta calidad, grabación incluida) | Sí (muy estable y popular) | Sí (limitado, más orientado a canales de voz) | Sí (integrado con Google Calendar) | Sí (opciones avanzadas para empresas) |
| ***Integraciones destacadas*** | Office 365, SharePoint, Planner | Outlook, Google Calendar, apps externas | Google Drive, Jira, GitHub, Zoom, Teams | Google Workspace | Microsoft 365, Salesforce, herramientas Cisco |
| ***Colaboración***  ***en documentos*** | Sí (edición en tiempo real con Office) | Limitada (pantalla compartida, plugins) | Limitada (puede integrar apps externas) | Sí (Google Docs, Sheets, etc.) | Sí (pantalla compartida, algunas integraciones) |
| ***Grabación de reuniones*** | Sí | Sí | No nativa (apps externas o pagos) | Sí | Sí |
| ***Calidad de audio/video*** | Alta (depende de conexión y carga) | Muy alta (optimizada para estabilidad) | Media (voz por canal, no siempre HD) | Alta (especialmente en Google Chrome) | Alta (orientada a entornos corporativos) |
| ***Facilidad de***  ***uso*** | Media (más completo pero algo complejo) | Muy intuitivo y directo | Muy fácil de usar | Muy fácil, especialmente en Gmail | Media (más orientado a entornos empresariales) |
| ***Precio*** | Gratuito (básico) / Planes pagos | Gratuito (limitado) / Planes pagos | Gratuito (con limitaciones) / Planes pagos | Gratuito / Planes con más capacidad | Gratuito / Planes pagos |
| ***Ideal para*** | Equipos corporativos, | Reuniones, webinars, | Startups, equipos ágiles, desarrolladores | Educación, equipos que usan Google | Empresas medianas y grandes |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| oficinas híbridas | educación remota |  |  |  |
|  |  |  |  |  |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Característica***  ***/ Herramienta*** | **Email corporativo** | **Slack (mensajes)** | **Microsoft Teams (chat)** | **Trello (comentarios**  **)** | **Confluence (comentarios y páginas)** |
| ***Tipo de comunicació***  ***n*** | Formal, estructurada | Mensajería rápida, informal | Chat interno y colaborativo | Comentarios en tareas y tarjetas | Comentarios y discusiones sobre contenido |
| ***Notificacione***  ***s*** | Sí (correo o app) | Sí (notificaciones por canal o directas) | Sí (banners, sonidos, correo) | Sí (en tarjetas asignadas o seguidas) | Sí (en páginas seguidas o con @menciones) |
| ***Organización***  ***de la información*** | Por carpetas, etiquetas, hilos | Por canales y hilos | Por chats y canales | Por tarjetas, listas y tableros | Por espacios, páginas e historial |
| ***Edición de mensajes*** | No (una vez enviado) | Sí (editar y eliminar) | Sí (editar y eliminar) | No (comentarios no editables directamente) | Sí (comentarios y páginas editables) |
| ***Seguimiento de tareas*** | Limitado (según plataforma usada) | Integrado con apps externas | Integrado con Planner, To Do, etc. | Sí (orientado a gestión de tareas) | Sí (con macros o integración con Jira) |
| ***Facilidad de búsqueda*** | Alta (por asunto, remitente, fecha) | Alta (por palabra, canal, usuario) | Alta (por canal, usuario, palabra clave) | Media (por tarjeta o palabra clave) | Alta (por título, contenido, autor) |
| ***Nivel de formalidad*** | Alto | Bajo a medio | Medio | Bajo | Medio a alto |
| ***Control de***  ***acceso*** | Sí (según servidor y dominio) | Sí (por canales y permisos) | Sí (por equipos y canales) | Sí (por tableros y miembros) | Sí (por espacios, páginas y permisos) |
| ***Ideal para*** | Comunicación oficial y externa | Comunicación ágil entre equipos | Comunicación interna estructurada | Seguimiento de tareas y coordinación | Documentación con colaboración diferida |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Característic a / Herramienta*** | **Jira** | **Asana** | **Trello** | **Microsoft Project** | **Monday.com** |
| ***Enfoque principal*** | Gestión de proyectos ágiles (Scrum/Kanban  ) | Gestión de tareas y proyectos | Gestión visual de tareas (Kanban) | Planificación avanzada de proyectos | Gestión de trabajo y proyectos colaborativos |
| ***Tipos de comunicació n integrada*** | Comentarios en tareas, integración con Confluence, notificaciones | Comentarios, @menciones  , inbox | Comentarios en tarjetas, @menciones | Notas, dependencias  , comentarios limitados | Comentarios en elementos, actualizaciones |
| ***Menciones y etiquetas*** | Sí (@usuario, etiquetas personalizadas) | Sí (@usuario, secciones, campos) | Sí (@usuario, etiquetas en tarjetas) | Limitado | Sí (@usuario, etiquetas por color o texto) |
| ***Historial de***  ***actividad*** | Completo por ticket y proyecto | Completo por tarea y proyecto | Por tarjeta (actividad reciente) | Limitado (dependiendo del plan) | Sí, línea de tiempo de actualizaciones |
| ***Integración con otras***  ***apps*** | Alta (Slack, Teams, Confluence, GitHub…) | Alta (Slack, Zoom, Gmail, etc.) | Alta (Slack, Google Drive, etc.) | Media (Outlook, Excel, SharePoint) | Alta (Slack, Teams, Zoom, Gmail, etc.) |
| ***Alertas y notificacione***  ***s*** | Sí (correo, push, en plataforma) | Sí (bandeja, correo, push) | Sí (correo, notificaciones en app) | Sí (limitado y menos interactivo) | Sí (correo, notificaciones en app y push) |
| ***Visualización del proyecto*** | Tablero Kanban, Scrum, cronograma | Lista, tablero Kanban, cronograma | Tablero Kanban | Diagrama de Gantt, cronograma | Tablero, cronograma, tabla, calendario |
| ***Facilidad de***  ***uso*** | Media (más técnico) | Alta (intuitivo y visual) | Muy alta (simple y flexible) | Media-baja (más formal y técnico) | Alta (muy visual y moderna) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Ideal para*** | Equipos de desarrollo, TI | Equipos generales y creativos | Equipos pequeños, startups | Empresas grandes con proyectos estructurados | Equipos ágiles de cualquier tamaño |



1. Herramienta para creación compartida: Notion

¿Por qué?

* + Es una plataforma muy flexible y visual, ideal para documentación de proyectos, listas de tareas, wikis, bases de datos ligeras y colaboración en tiempo real.
  + Permite organizar todo el contenido del proyecto (desde ideas hasta entregables finales) de forma modular y accesible.
  + Puedes integrarlo con GitHub.
  + Perfecto para E. Urbana que necesitan agilidad y orden sin complicarse con herramientas demasiado pesadas.

1. Herramienta para visualización: Power BI

¿Por qué?

* + Ideal para crear dashboards interactivos con los datos de los sensores de E. Urbana.
  + Permite conectar fuentes externas como bases de datos, archivos CSV, APIs, etc., y transformarlos en gráficos dinámicos.
  + Es profesional, ampliamente adoptado en el ámbito empresarial y tiene una versión gratuita con muchas capacidades.
  + Excelente para presentar resultados tanto a equipos técnicos como a instituciones interesadas.

1. Herramienta para comunicación en tiempo real : Microsoft Teams

¿Por qué?

* + Integra chat, videollamadas y colaboración en documentos en una sola plataforma.
  + Muy útil para coordinar entre áreas (por ejemplo, software y energías renovables en E. Urbana).
  + Puedes trabajar en tiempo real sobre documentos de Word, Excel o PowerPoint sin salir de Teams.
  + Se adapta bien a un equipo en crecimiento y puede integrarse con Notion, Power Bi.

1. Herramienta para comunicación asincrónica: Microsoft Teams (chat y mensajes)

¿Por qué?

* + Historial claro de conversaciones: Puedes consultar todo lo hablado en un canal o chat privado incluso semanas después.
  + Permite dejar mensajes fuera del horario: útil para equipos que no siempre están activos al mismo tiempo.
  + @Menciones y notificaciones: aseguran que la persona correcta vea el mensaje.
  + Un solo canal de comunicación: evita tener que usar otra plataforma (como Slack) y simplifica el flujo del equipo.
  + Organización por temas: los canales pueden actuar como foros para diferentes aspectos del proyecto.

1. Plataforma que integra comunicación y tareas: Trello

¿Por qué?

* + Visual, simple y efectivo: organiza tareas en tableros estilo Kanban (Por hacer, En progreso, Hecho).
  + Comentarios por tarea: cada tarjeta permite discusiones específicas, evitando perder contexto.
  + Fechas, asignaciones y listas de verificación: útiles para gestionar responsables y entregables.
  + Integraciones: se conecta fácilmente con Notion, Microsoft Teams, Power BI y otros.
  + Ideal para equipos ágiles y pequeños: el flujo es natural y rápido de aprender.

**Justificación y Beneficios del Patrón MVC en E. Urbana**

La adopción del patrón **Model-View-Controller (MVC)** en *E. Urbana* responde a la necesidad de estructurar de manera clara y eficiente un sistema que gestiona:

* + **Recolección de datos en tiempo real** de múltiples sensores (turbidez, nivel de agua, calidad).
  + **Procesamiento y validación** de lecturas antes de almacenar o mostrar.
  + **Presentación interactiva** en un dashboard web.

# ¿Por qué MVC en E. Urbana?

## Modularidad y separación de responsabilidades

MVC divide el proyecto en tres capas independientes:

* + - **Model:** Representa los datos y reglas de negocio (ej. validación de lecturas, gestión de formatos).
    - **View:** Define cómo se muestran los datos al usuario (tableros, gráficos).
    - **Controller:** Actúa como intermediario, recibiendo solicitudes de la interfaz y coordinando Model y View.

Esta separación evita que la lógica de negocio se mezcle con la presentación o la gestión de datos.

## Escalabilidad para nuevos sensores y funciones

Al incorporar un nuevo tipo de sensor, solo se debe:

* + - Extender el **Model** con la estructura y validaciones específicas.
    - Agregar o ajustar la **View** para mostrar sus datos.
    - Actualizar el **Controller** que coordina la obtención y envío de lecturas.

De esta forma, el resto del sistema permanece intacto.

## Facilidad de mantenimiento y colaboración

* + - Los equipos de frontend y backend pueden trabajar en paralelo:
      * **Backend:** Reglas de negocio y acceso a datos (*Model*, *Controller*).
      * **Frontend:** Presentación y experiencia de usuario (*View*, *Controller*).
    - Cambios en la base de datos, lógica de negocio o interfaz de usuario se aíslan en sus respectivas capas.

## Mejor testabilidad

* + - Permite pruebas unitarias específicas para cada capa:
      * Validar el **Model** sin necesidad de UI o base de datos real.
      * Verificar respuestas del **Controller** simulando distintos escenarios.
      * Revisar renderizado de la **View** con datos controlados.

# Ventajas concretas en E. Urbana

* **Desarrollo incremental:** Implementación funcionalidad por funcionalidad, sensor por sensor y vista por vista.
* **Consistencia:** Estandarización del flujo de datos y respuestas, tanto en la API como en la interfaz web.
* **Reutilización:** Componentes de **View** y lógica de **Controller** reutilizables para diferentes tipos de sensores.
* **Rápida incorporación de mejoras:** Cambios en la visualización o lógica de validación afectan solo la capa correspondiente.

# Patrones de Diseño

1. **Estructurales: Adapter**

## ¿Qué hace?

Permite que clases con interfaces incompatibles trabajen juntas, actuando como un “adaptador” entre ellas.

## Ejemplo de proyecto:

**Integrar sensores antiguos a un sistema moderno.**

Supón que tienes un sistema de monitoreo de agua que recibe datos de sensores modernos, pero ahora quieres agregar sensores antiguos que envían información en un formato diferente. Usas un *Adapter* para convertir los datos de los sensores antiguos al formato que entiende tu sistema, sin cambiar el código original del sistema.

# Conductuales: Observer

## ¿Qué hace?

Permite que un objeto (sujeto) notifique automáticamente a otros objetos (observadores) cuando cambia su estado.

## Ejemplo de proyecto:

**Dashboard de monitoreo en tiempo real.**

Imagina un tablero web que muestra los niveles de agua en diferentes tanques. Cada vez que un sensor detecta un cambio en el nivel de agua, el sistema (sujeto) notifica automáticamente al dashboard (observador), actualizando los datos en tiempo real para los usuarios.

# Creacionales: Singleton

## ¿Qué hace?

Asegura que una clase tenga solo una instancia y provee un punto global de acceso a ella.

## Ejemplo de proyecto:

**Gestor de configuración de una aplicación.**

Supón que tu sistema de monitoreo necesita acceder a la configuración general (como rutas de sensores, límites de alerta, etc.) desde varios lugares. Usas un *Singleton* para que solo haya una instancia del gestor de configuración y todos los módulos accedan a la misma configuración, evitando inconsistencias.

# Emergentes: MVC (Model-View-Controller)

## ¿Qué hace?

Separa la aplicación en tres capas: Model (datos y lógica), View (interfaz), y Controller (gestión de flujo).

## Ejemplo de proyecto:

**Sistema web para monitoreo urbano (E. Urbana).**

El sistema recibe datos de sensores (Model), los procesa y muestra en gráficos en una página web (View), y los usuarios pueden pedir reportes o filtrar información mediante botones o menús (Controller), todo separado para facilitar mantenimiento, pruebas y escalabilidad.

M O D E L O D E N E G O C I O S C A N V A S

ECOLUZ URBANA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Asociaciones Clave   * Proveedores * Franquiciados | Actividades Clave  -Desarrollo del sistema de monitoreo y gestión de energía.  -Monitoreo continuo del sistema y atención de fallas | Propuestas de Valor   * Valor añadido * Productos en cada segmento   -Qué necesidades satisfacemos | | Relación con los clientes   * Asistencia personal * Autoservicio * Colaboración | Segmentos de clientes   * Nichos * Segmentos * Mercados de masas * Diversificación |
| * Evaluación periódica   RdeelciumrspoasctColyavaejustes para optimizar el  -fuRnecicounrasmoisenhtuomanos   * Económicos * Intelectuales * Físicos | Canales   * Modo de entrega * Canales mas eficientes * Post venta |
| Estructura de costes   * Salarios de empleado * Costes de instalación * Expansión * Seguros y aspectos legales | | | Fuentes de ingresos   * Precio del producto * Métodos de pago * Cuánto quieren pagar * Porcentajes de ingres | | |

## Modelo de Negocio (Business Model Canvas) – E-Urbana Asociaciones Clave

1. Proveedores: fabricantes de sensores IoT (corriente, luminosidad), gateways de red (LoRaWAN/NB-IoT) y luminarias inteligentes; garantizan calidad de hardware y soporte técnico.
2. Franquiciados: empresas integradoras locales autorizadas que instalan y mantienen el sistema bajo estándares EcoLuz; permiten expansión rápida y un servicio uniforme.

## Actividades Clave

1. Desarrollo de la plataforma IoT: diseño, programación y pruebas del backend, frontend y microservicios para procesamiento de datos en tiempo real.
2. Monitoreo continuo y atención de fallas: supervisión 24/7 de cada luminaria; disparo automático de alertas y coordinación de reparaciones.
3. Evaluación periódica y optimización: análisis de consumos históricos, ajustes de algoritmos adaptativos de encendido y generación de informes de mejora.

## Propuestas de Valor

1. Monitorización y control en tiempo real: visibilidad instantánea del estado y consumo energético de cada luminaria.
2. Mantenimiento predictivo: alertas automáticas ante anomalías para reducir tiempos de inactividad y costes de reparación.
3. Encendido adaptativo inteligente: ajustes de brillo basados en sensores de movimiento y luminosidad ambiental.
4. Informes y mapas interactivos: herramientas visuales para planificar inversiones y demostrar ahorros.
5. Asistencia personal: account manager dedicado durante implementación y resolución de incidencias críticas.
6. Autoservicio: portal web con tutoriales, FAQs y sistema de ticketing para consultas rápidas.
7. Colaboración continua: workshops, feedback loops y actualizaciones conjuntas para evolucionar la plataforma según necesidades.

## Recursos Clave

1. Humanos: equipo de desarrolladores, ingenieros de campo, analistas de datos y personal de soporte.
2. Económicos: inversión en infraestructura cloud (AWS/Azure/GCP), licencias de software y marketing.
3. Intelectuales: propiedad intelectual de algoritmos predictivos y know-how de IoT.
4. Físicos: servidores, centros de datos y stock de hardware (sensores, gateways).

## Canales

1. Modo de entrega: plataforma SaaS accesible vía web y móvil; hardware enviado e instalado por franquiciados.
2. Canales de venta: licitaciones públicas para gobiernos, demos en ferias de Smart Cities y alianzas con integradores de IoT.
3. Post-venta: soporte 24/7, actualizaciones automáticas y programas de capacitación periódica.
4. Nichos: municipios medianos que buscan modernizar alumbrado sin grandes inversiones iniciales.
5. Segmentos: grandes ciudades con flotas extensas de luminarias; empresas de servicios públicos.
6. Mercados de masas: zonas residenciales y comerciales con necesidad de gestión energética.
7. Diversificación: parques industriales, campus universitarios y desarrollos privados que requieran iluminación inteligente.

## Estructura de Costes

1. Salarios de equipo: desarrolladores, ingenieros de campo y personal de soporte.
2. Costes de instalación: transporte y montaje de hardware por franquiciados.
3. Expansión: formación continua y certificación de nuevos franquiciados.
4. Seguros y aspectos legales: cobertura de responsabilidad civil, licencias de operación y certificaciones de equipo.

## Fuentes de Ingresos

1. Suscripción SaaS: tarifa mensual o anual por luminaria conectada.
2. Servicios de implementación: honorarios únicos por instalación de hardware y configuración inicial.
3. Módulos premium: analítica avanzada, informes a medida y consultorías especializadas.
4. Soporte y mantenimiento: planes SLA con distintos niveles de respuesta y cobertura.

# Actividades Clave

## Desarrollo y mejora continua de la plataforma

Implementación de nuevas métricas, visualizaciones y algoritmos predictivos.

## Integración y pruebas de campo

Validaciones en calle, calibración de sensores y aseguramiento de calidad.

## Gestión de proyectos e implementación

Planificación de despliegues, coordinación con equipos de instalación y formación al cliente.

## Marketing y generación de demanda

Creación de casos de éxito, contenido técnico y participación en eventos sectoriales.

# Socios Clave

## Proveedores de hardware IoT

Fabricantes de sensores de corriente, luminosidad y motores de comunicación (LoRaWAN, NB-IoT).

## Integradores de infraestructura eléctrica

Empresas instaladoras y mantenedoras de redes de alumbrado público.

## Consultoras de energía y sostenibilidad

Alianzas para ofrecer paquetes “llave en mano” a municipios interesados.

## Plataformas de nube y CDN

AWS, Azure o GCP para alojar la plataforma con garantía de uptime y escalabilidad.

# Estructura de Costos

## Costes de desarrollo y operación de la plataforma

Salarios de equipo DevOps, licencias de software, servidores en la nube.

## Gastos de I+D y certificaciones

Pruebas de campo, homologaciones de dispositivos y patentes.

## Marketing y ventas

Ferias, publicidad online, generación de contenidos y comisiones de ventas.

## Soporte y mantenimiento

Infraestructura de atención al cliente, herramientas de ticketing y formación continua.

|  |  |  |  |
| --- | --- | --- | --- |
| **Flujo de Trabajo** | **Características Principales** | **Ventajas** | **Desventajas** |
| **Git Flow** | - Ramas: main, develop, feature/\*, release/\*, hotfix/\*- Ideal paraproyectos con ciclos de versiones claros | - Organización clara- Ideal para equipos grandes | - Complejidad elevada- Muchas ramas que mantener |
| **GitHub Flow** | - Rama main siempre desplegable- Se trabaja con ramas pequeñas y PR tempranos | - Sencillo y ágil- Enfocado en colaboración y revisión continua | - No ideal para versiones grandes- Menor control en ambientes |
| **GitLab Flow** | - Asociado a entornos (production, staging, etc.)- Merge Requests entre entornos | - Buena integración con pipelines- Gestión de entornos más clara | - Configuración más compleja |
| **Trunk-Based Development** | - Todo el trabajo en main o trunk- Feature flags para funcionalidades- Commits  frecuentes | - Ideal para CI/CD avanzado- Historial limpio y continuo | - Requiere cultura fuerte de testing e integración |

Elijo **GitHub Flow** porque:

* **Despliegue continuo**: la rama **main** siempre está en estado desplegable, lo que facilita lanzar actualizaciones tan pronto como se apruebe un Pull Request.
* **Simplicidad y agilidad**: solo necesitas crear ramas cortas para cada feature o corrección, hacer un PR, revisarlo y fusionarlo; sin gestionar múltiples ramas “release” o “hotfix”.
* **Revisión temprana**: fomenta la colaboración y el feedback constante, ya que cualquier cambio se integra mediante Pull Requests desde el inicio.
* **Menor sobrecarga**: al no tener que coordinar ramas de preparación de versiones, reducimos el mantenimiento y aceleramos el ritmo de desarrollo.