Logotipo

Descripción generada automáticamente





Contenido

[**1.**](#_heading=h.gjdgxs) **Introducción** 3

[**2.**](#_heading=h.30j0zll) **Propósito** 3

[**3.**](#_heading=h.1fob9te) **Alcance** 3

[**4.**](#_heading=h.3znysh7) **Equipo de Trabajo-Actores del Desarrollo** 4

[**5.**](#_heading=h.2et92p0) **Recomendaciones de conformidad con esta práctica.** 4

[Para asegurar que la arquitectura del sistema cumpla con los requisitos y estándares establecidos, se recomiendan las siguientes prácticas: 4](#_heading=h.tyjcwt)

[1. Implementar revisiones de arquitectura cada xxxx para asegurar que se ajusten a los cambios de requisitos. 4](#_heading=h.3dy6vkm)

[2. Utilizar patrones de diseño desacoplados como MVC para modularidad y escalabilidad o por capas. 4](#_heading=h.1t3h5sf)

[3. Adoptar metodologías ágiles como Scrum para la entrega continua de mejoras y funcionalidades. 4](#_heading=h.4d34og8)

[**6.**](#_heading=h.2s8eyo1) **Referencias y estándares aplicables a este documento:** 5

[**7.**](#_heading=h.17dp8vu) **Definiciones, acrónimos y abreviaciones.** 5

[**8.**](#_heading=h.3rdcrjn) **DESCRIPCIÓN DE ARQUITECTURA:** La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas. La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas. 6

[8.1 VISTAS: 6](#_heading=h.26in1rg)

[8.2 TIPOS DE VISTAS: 6](#_heading=h.lnxbz9)

[8.3 Framework Conceptual 7](#_heading=h.35nkun2)

[8.3.1 Descripción de la arquitectura en contexto 7](#_heading=h.1ksv4uv)

[8.3.1 Actores Usuarios y sus roles: 8](#_heading=h.44sinio)

[8.3.2 Actividades de arquitectura en el ciclo de vida 9](#_heading=h.2jxsxqh)

[8.3.3 Descripciones prácticas de arquitectura. 9](#_heading=h.z337ya)

[8.4 Documentación de la arquitectura 10](#_heading=h.3j2qqm3)

[8.5 Selección de puntos de vista de la arquitectura 10](#_heading=h.1y810tw)

[8.6 Vistas de la arquitectura 11](#_heading=h.4i7ojhp)

[UML-VISTA DE ESCENARIO 11](#_heading=h.2xcytpi)

[UML-VISTA LÓGICA-DIAGRAMA DE CLASES 12](#_heading=h.1ci93xb)

[UML-VISTA LÓGICA-DIAGRAMA DE BASE DE DATOS 14](#_heading=h.qsh70q)

[UML-VISTA DE PROCESO-DIAGRAMAS DE ACTIVIDAD 16](#_heading=h.1pxezwc)

[UML-VISTA DE DESPLIEGUE 18](#_heading=h.49x2ik5)

[UML-VISTA FÍSICA – DIAGRAMA INFRAESTRUCTURA 19](#_heading=h.2p2csry)

[Usabilidad y Calidad de software- Front-end -Vistas Principales 21](#_heading=h.147n2zr)

[Usabilidad y Calidad de software – Vistas específicas de usuario 22](#_heading=h.3o7alnk)

[8.7 Consistencia en la cantidad de vistas de la arquitectura. 23](#_heading=h.23ckvvd)

[8.7.1 DESCRIPCION DE MODULOS 23](#_heading=h.ihv636)

[8.7.2 DESCRIPCIÓN DE COMPONENTES 23](#_heading=h.32hioqz)

[8.7.3 DESCRIPCIÓN DE CONECTORES 23](#_heading=h.1hmsyys)

[9.1 Acceso a fuentes de información del proyecto 25](#_heading=h.41mghml)

1. **Introducción**

El presente informe tiene como objetivo describir en detalle la arquitectura de un sistema de hotelería canina diseñado para automatizar la administración de tareas y procesos comunes dentro de la empresa Perriot.

El sistema propuesto busca mejorar la optimización y eficiencia de las reservas y pagos del hotel.

El sistema desarrollado será implementado en un servicio web. El sistema también proporcionará una interfaz intuitiva y accesible para todos los usuarios, asegurando que las gestiones se realicen de manera eficiente y organizada basada en los estándares de la empresa Perriot.

1. **Propósito**

El propósito de este documento es servir como guía para el diseño e implementación de la arquitectura del sistema web diseñado. En él se detallan las decisiones técnicas, los componentes principales y los puntos de integración que permitirán implementar una solución robusta y escalable, capaz de gestionar múltiples comunidades bajo un esquema de administración unificado.

Este informe es esencial para los desarrolladores y stakeholders del proyecto, ya que les proporcionará una visión detallada de la estructura y comportamiento del sistema, así como las relaciones entre las diferentes funcionalidades y componentes. Asimismo, establece las bases para futuras expansiones y mejoras del sistema, considerando factores de disponibilidad, seguridad y usabilidad.

1. **Alcance**

El sistema gestionará las siguientes áreas:

* A
* B
* C
* D
* E

1. **Equipo de Trabajo-Actores del Desarrollo**

El desarrollo del sistema requiere la participación de un equipo multidisciplinario con roles claramente definidos para asegurar que el proyecto se lleve a cabo de manera eficiente. A continuación, se presenta una tabla con los principales roles y sus responsabilidades:

| Rol | Responsabilidades |
| --- | --- |
| Cliente-Stakeholder | Perriot |
| Product Owner | Ignacio Coloma |
| Desarrolladores Backend | Ignacio Coloma |
| Desarrolladores Frontend | Gabriel Avendaño |
| DBA-Especialista en Base de Datos | Gabriel Avendaño - Ignacio Coloma |
| Tester/QA | Gabriel Avendaño |
| Analista de Requisitos | Gabriel Avendaño - Ignacio Coloma |
| Administrador Plataformas | Gabriel Avendaño - Ignacio Coloma |

1. **Recomendaciones de conformidad con esta práctica.**

Para asegurar que la arquitectura del sistema cumpla con los requisitos y estándares establecidos, se recomiendan las siguientes prácticas:

1. Implementar revisiones de arquitectura cada semana para asegurar que se ajusten a los cambios de requisitos.
2. Utilizar patrones de diseño desacoplados como MVC para modularidad y escalabilidad o por capas.
3. Adoptar metodologías ágiles como Scrum para la entrega continua de mejoras y funcionalidades.
4. **Referencias y estándares aplicables a este documento:**

El documento sigue los siguientes estándares de calidad para asegurar la robustez y escalabilidad del sistema:

* **IEEE 830-1998 ST**: Estandarización de requisitos de software.
* **ISO 9126-2001**: Métricas de calidad del software.
* **Modelo 4+1 de Kruchten**: Modelo estructural basado en múltiples vistas de escenarios.
* **ATAM (Architecture Tradeoff Analysis Method)**: Metodología para evaluar y documentar decisiones arquitectónicas

1. **Definiciones, acrónimos y abreviaciones.**

| Término | Definición |
| --- | --- |
| DAS | Documento de Arquitectura de Software |
| MVC | Modelo-Vista-Controlador, patrón de diseño utilizado para separar la lógica de negocio de la interfaz de usuario |
| UML | Unified Modeling Language, lenguaje gráfico para representar la arquitectura del sistema |
| CRUD | Operaciones básicas de Create, Read, Update, Delete para la gestión de datos. |

1. **DESCRIPCIÓN DE ARQUITECTURA:**La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas. La arquitectura del sistema se presenta mediante el uso de vistas basadas en el Modelo 4+1 de Kruchten, con el fin de proporcionar una visión completa y estructurada del sistema desde diferentes perspectivas.
   1. VISTAS:

Cada vista se representará en un diagrama UML para describir la estructura y comportamiento del sistema.

* 1. TIPOS DE VISTAS:

1. **Vista de Escenarios (Diagrama de Caso de Uso)**: Describe las funcionalidades del sistema y cómo interactúan los diferentes actores con él.
2. **Vista Lógica (Diagrama de Clases)**: Detalla la estructura de datos y las relaciones entre las principales entidades del sistema.
3. **Vista de Procesos (Diagrama de Actividad)**: Muestra el flujo de actividades dentro del sistema y cómo se coordinan los diferentes procesos.
4. **Vista de Despliegue (Diagrama de Componentes)**: Representa la distribución de componentes en la infraestructura física.
5. **Vista Física (Diagrama Topológico)**: Representa la infraestructura de red y la comunicación entre los diferentes servicios del sistema
   1. Framework Conceptual
      1. Descripción de la arquitectura en contexto

El sistema será diseñado y desarrollado utilizando un enfoque ágil con metodologías como Scrum, lo que permitirá la entrega incremental de funcionalidades, la retroalimentación constante y la capacidad de adaptarse a cambios en los requisitos a lo largo del ciclo de vida del proyecto. Esto asegurará que se puedan abordar necesidades emergentes y maximizar el valor del producto para los usuarios.

El diseño del sistema se basará en la separación de vistas, que ofrecerán diferentes perspectivas de la arquitectura para una comprensión integral. A continuación, se describen las principales vistas:

* **Vista de Escenarios**: Representada mediante diagramas de Casos de Uso, que describen las interacciones del usuario con el sistema y los flujos principales.
* **Vista Lógica**: Incluye el Modelo de Clases, que muestra las entidades y relaciones del sistema, así como el Modelo de Datos que detalla la estructura de la base de datos.
* **Vista de Procesos**: Utiliza Diagramas de Actividad para ilustrar los flujos internos del sistema y las interacciones entre los distintos módulos.
* **Vista de Despliegue**: Se implementará a través de Diagramas de Componentes o Modelos de Capas para reflejar la distribución de la lógica de negocio, la interfaz de usuario y la capa de acceso a datos.
* **Vista Física**: Utiliza Diagramas Topológicos o de Servicios para mostrar la distribución del sistema en el entorno físico, como servidores, redes y otros componentes de infraestructura.

Cada una de estas vistas está interconectada para garantizar la coherencia y la alineación con los requisitos de negocio. No hay ninguna vista separada de una implementación específica, sino que todas contribuyen a una visión holística de la arquitectura del sistema descrita en este documento.

Estas vistas se han creado utilizando el Lenguaje de Modelado Unificado (UML) en su versión 2.0 y han sido desarrolladas empleando herramientas de modelado estándar como **StarUML** y Lucidchart y en front-end **Canva, Miro, Figma u otro** para facilitar la creación de diagramas visuales que respalden la documentación arquitectónica.

El estilo arquitectónico se centra en una combinación de patrones de diseño orientados a servicios xxxx (ejemplo: SOA) y arquitectura por capas, en donde cada módulo del sistema se corresponde con un conjunto específico de casos de uso que resuelven los procesos definidos para la operación.

Estos procesos corresponden a:

Ejemplo:

* **Gestión de Reservas y Espacios Comunes**: Administración de las reservas de áreas comunes y la validación del uso adecuado de los recursos por parte de los residentes.
* **Gestión de Gastos Comunes**: Cálculo y distribución de gastos, así como la creación de reportes financieros.
* **Monitoreo y Seguridad del Sistema**: Gestión de accesos, control de uso de áreas restringidas y monitoreo de seguridad
  + 1. Actores Usuarios y sus roles:

Este documento representa la identificación de Actores/Usuarios Stakeholders y sus roles a partir de la interpretación de los casos de uso del Negocio asociados.

| ACTOR | ROL / CARGO | FUNCION |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

8.3.2 Actividades de arquitectura en el ciclo de vida

El desarrollo de la arquitectura del sistema xxxx se abordó utilizando un ciclo de vida **iterativo incremental** basado en **Scrum**. Esta metodología fue seleccionada porque permite realizar entregas parciales y frecuentes de software funcional, facilitando la validación temprana de los requisitos por parte de los stakeholders y el productowner con la adaptación a cambios en el entorno de negocio.

El ciclo de vida se organizó en **sprints** de 2 a 3 semanas, con cada iteración centrada en la construcción y mejora de un módulo específico del sistema (por ejemplo, el módulo de gestión de gastos comunes o el módulo de reservas). Al final de cada sprint, se realiza una revisión detallada de los resultados para asegurar la alineación con las expectativas del cliente y se priorizan las tareas para el siguiente sprint.

8.3.3 Descripciones prácticas de arquitectura.

Durante el proceso de desarrollo, la arquitectura del sistema fue escalada desde un **escenario principal de caso de uso** hasta un diseño modular a través de los siguientes pasos:

1. **Análisis de Requerimientos**: Se analizaron los requisitos iniciales del sistema para identificar los casos de uso principales, tales como "Administrar Gastos Comunes", "Reservar Espacios Comunes" y "Generar Reportes de Cobros".
2. **Definición de la Vista Lógica**: A partir del escenario principal, se identificaron las entidades y relaciones claves en un diagrama de clases UML, representando el núcleo del sistema.
3. **Modelo de Datos**: Se desarrolló un diagrama de base de datos para definir la estructura de la información, manteniendo la coherencia con la vista lógica.
4. **Vista de Procesos**: Se modelaron los flujos de las actividades y transacciones principales del sistema mediante diagramas de actividad.
5. **Despliegue**: Se definió la infraestructura física y lógica para alojar el sistema en la nube, detallando los servicios necesarios, como balanceadores de carga, servidores de aplicaciones y bases de datos distribuidas.
6. **Definición de Planes de pruebas y técnicas de pruebas testing**.
   1. Documentación de la arquitectura

* **ISO/IEC 25010**: Se utilizó para evaluar la calidad del sistema en términos de funcionalidad, seguridad y eficiencia.
* **IEEE 1471-2000**: Proporcionó la guía para la representación de vistas arquitectónicas y la identificación de stakeholders.
* **Modelo 4+1 de Kruchten**: Estructura la arquitectura en vistas lógicas, de desarrollo, de procesos y física, facilitando la visualización desde múltiples perspectivas.
* **Planes de prueba**: (Declarar los tipos de pruebas a realizar)
  1. Selección de puntos de vista de la arquitectura

| VISTAS | UML | Cantidad |
| --- | --- | --- |
| Escenario Principal | Diagrama de Caso de uso | 1 (2-3 los necesarios) |
| Vista Lógica | Diagrama de Clases  MER | 1  1 |
| Vista de Proceso | Diagrama de Actividad:  Nombre 1.  Nombre 2  Nombre 3. | 1 (2-3, los necesarios) |
| Vista Física | Diagrama Topológico de Servicios | 1 |
| Vista de Despliegue | Diagrama de componentes  Diagrama de Capas (MVC) | 1  1 |

* 1. Vistas de la arquitectura

Ejemplos:

| UML-VISTA DE ESCENARIO |
| --- |
| Diagrama de Caso de uso extendido /Subproyecto Hotelería Perriot |
|  |

**Gestión de Condominios**: Permite al Súper Administrador crear, modificar y eliminar condominios. Incluye la gestión de propiedades bajo el régimen de copropiedad y la asignación de administradores.

**Gestión de Usuarios**: Facilita la creación y administración de los diferentes roles de usuarios dentro del sistema.

**Notificación de Gastos Comunes**: El sistema envía automáticamente notificaciones por correo electrónico a los residentes sobre sus gastos comunes mensuales, asegurando que estén al tanto de sus obligaciones.

**Visualización de Gastos Comunes**: Los residentes pueden acceder a su estado de cuenta y revisar los detalles de los gastos comunes acumulados.

**Cálculo de Gastos Comunes**: El sistema calcula automáticamente los gastos comunes basados en la estructura del condominio y los servicios utilizados.

**Pago de Gastos Comunes**: Los residentes pueden realizar pagos a través del sistema utilizando diversas formas de pago.

**Aplicación de Multas Automáticas**: Si los residentes no realizan sus pagos a tiempo, el sistema aplica automáticamente multas y actualiza el estado de la deuda.

**Reserva de Estacionamiento y Áreas Comunes**: Los residentes pueden consultar la disponibilidad de estacionamiento y reservar áreas como quinchos, multicanchas y salas de eventos.

**Generación de Informes y Reportes**: El sistema permite al administrador generar informes sobre la gestión financiera y de reservas.

**Alertas Automáticas**: Notificaciones sobre eventos importantes, como pagos atrasados o cambios en las normativas del condominio.

**Gestión de Multas y Deudas**: Los administradores pueden modificar deudas y aplicar descuentos según corresponda.

**Registro de Usuarios Morosos**: El sistema mantiene un registro de residentes que no han realizado sus pagos, generando reportes para su gestión.

| UML-VISTA LÓGICA-DIAGRAMA DE CLASES |
| --- |
| [Diagrama de Clases] |
|  |

El **Diagrama de Clases** para el sistema de gestión de condominios muestra las clases involucradas, sus atributos, métodos y las relaciones entre ellas. Este diagrama ayuda a modelar la estructura y el comportamiento del sistema de una manera orientada a objetos.

**2. Elementos Principales**

* **Clases**:
  + **Usuario**:
    - **Atributos**: ID\_Usuario, Nombre, Apellido, Rol, Email, Contraseña, Estado.
    - **Métodos**: iniciarSesion(), realizarPago().
  + **Condominio**:
    - **Atributos**: ID\_Condominio, Nombre, Dirección, Estado, Tipo.
    - **Métodos**: agregarPropiedad(), gestionarGastosComunes().
  + **Propiedad**:
    - **Atributos**: ID\_Propiedad, Número, ID\_Condominio, Tipo.
    - **Métodos**: obtenerUsuarios(), agregarGastoComún().
  + **Gasto Común**:
    - **Atributos**: ID\_Gasto\_Común, Monto Total, Descripción, Mes/Año, Estado de Pago, ID\_Propiedad, ID\_Administrador.
    - **Métodos**: generarFactura(), marcarComoPagado().
  + **Multa**:
    - **Atributos**: ID\_Multa, Monto, Motivo, Fecha de Generación, Fecha de Vencimiento, Estado.
    - **Métodos**: aplicarMulta(), verificarEstado().
  + **Pago**:
    - **Atributos**: ID\_Pago, Monto Pagado, Fecha de Pago, ID\_Gasto\_Común, ID\_Usuario, Método de Pago.
    - **Métodos**: procesarPago(), enviarComprobante().
  + **Espacio Común**:
    - **Atributos**: ID\_Espacio\_Común, Nombre, Capacidad, Costo de Reserva, Estado, ID\_Condominio.
    - **Métodos**: reservarEspacio(), cancelarReserva().
  + **Reserva de Espacio**:
    - **Atributos**: ID\_Reserva, Fecha y Hora de Inicio, Fecha y Hora de Fin, ID\_Espacio\_Común, ID\_Usuario, Estado.
    - **Métodos**: confirmarReserva(), modificarReserva().
  + **Historial de Actividades**:
    - **Atributos**: ID\_Actividad, Tipo, Descripción, ID\_Usuario, Fecha y Hora.
    - **Métodos**: registrarActividad(), obtenerHistorialPorUsuario().

**3. Relaciones:**

* **Usuario** tiene una relación de **asociación** con **Condominio** (múltiples condominios).
* **Condominio** tiene una relación de **composición** con **Propiedad** y **Espacio Común** (un condominio contiene propiedades y espacios).
* **Propiedad** tiene una relación de **asociación** con **Gasto Común** (múltiples gastos asociados).
* **Gasto Común** tiene una relación de **asociación** con **Pago** y **Multa**.
* **Espacio Común** tiene una relación de **asociación** con **Reserva de Espacio** (múltiples reservas pueden existir para un espacio común).
* **Historial de Actividades** tiene una relación de **asociación** con **Usuario** (registrar acciones del usuario)

| UML-VISTA LÓGICA-DIAGRAMA DE BASE DE DATOS |
| --- |
| [Diagrama de BD - MER] |
|  |

El **Modelo Entidad-Relación (MER)** para el sistema de gestión de condominios representa las entidades involucradas, sus atributos y las relaciones que existen entre ellas. Este modelo permite visualizar cómo se organiza la información relacionada con la administración del condominio.

**2. Elementos Principales**

* **Entidades**:
  + **Usuario**:
    - **Atributos**: ID\_Usuario (PK), Nombre, Apellido, Rol, Email, Contraseña, Estado.
  + **Condominio**:
    - **Atributos**: ID\_Condominio (PK), Nombre del Condominio, Dirección, Estado, Tipo.
  + **Propiedad**:
    - **Atributos**: ID\_Propiedad (PK), Número de Propiedad, ID\_Condominio (FK), Tipo de Propiedad.
  + **Gasto Común**:
    - **Atributos**: ID\_Gasto\_Común (PK), Monto Total, Descripción, Mes/Año de Facturación, Estado de Pago, ID\_Propiedad (FK), ID\_Administrador (FK).
  + **Multa**:
    - **Atributos**: ID\_Multa (PK), Monto, Motivo, Fecha de Generación, Fecha de Vencimiento, Estado, ID\_Gasto\_Común (FK), ID\_Propiedad (FK).
  + **Pago**:
    - **Atributos**: ID\_Pago (PK), Monto Pagado, Fecha de Pago, ID\_Gasto\_Común (FK), ID\_Usuario (FK), Método de Pago.
  + **Espacio Común**:
    - **Atributos**: ID\_Espacio\_Común (PK), Nombre del Espacio, Capacidad, Costo de Reserva, Estado, ID\_Condominio (FK).
  + **Reserva de Espacio**:
    - **Atributos**: ID\_Reserva (PK), Fecha y Hora de Inicio, Fecha y Hora de Fin, ID\_Espacio\_Común (FK), ID\_Usuario (FK), Estado.
  + **Historial de Actividades**:
    - **Atributos**: ID\_Actividad (PK), Tipo de Actividad, Descripción, ID\_Usuario (FK), Fecha y Hora de la Actividad.
* **Relaciones**:
  + **Usuario** puede estar asociado a **múltiples condominios**.
  + **Condominio** tiene **múltiples propiedades** y **múltiples espacios comunes**.
  + **Propiedad** tiene **múltiples usuarios residentes** y puede tener un **propietario**.
  + **Gasto Común** está asociado a una **propiedad** y es administrado por un **administrador**.
  + **Multa** se asocia a un **gasto común** específico.
  + **Pago** está vinculado a un **gasto común** y realizado por un **usuario**.
  + **Espacio Común** pertenece a un **condominio** y puede tener **múltiples reservas**.
  + **Reserva de Espacio** está asociada a un **espacio común** y a un **usuario**.
  + **Historial de Actividades** está vinculado a un **usuario** y a un **componente del sistema**.

| UML-VISTA DE PROCESO-DIAGRAMAS DE ACTIVIDAD |
| --- |
| [Diagramas de Actividad] |
|  |

El **diagrama de actividad** representa el flujo de actividades que se llevan a cabo en un proceso específico, en este caso, el proceso de pago de gastos comunes. Este diagrama es útil para visualizar los pasos secuenciales y las decisiones que un residente debe tomar al realizar un pago.

**Componentes**:

1. **Inicio**: El residente inicia sesión en el sistema de gestión de condominios.
2. **Consultar Estado de Cuenta**: El residente accede a su sección de "Gastos Comunes".
3. **Seleccionar Método de Pago**: El residente elige su método preferido (tarjeta de crédito, transferencia bancaria).
4. **Realizar Pago**:
   * Si elige tarjeta de crédito:
     + El sistema envía los datos a la pasarela de pagos para procesar la transacción.
   * Si elige transferencia bancaria:
     + Se le proporcionan los detalles de la cuenta bancaria del condominio.
5. **Verificación del Pago**:
   * Si el pago es exitoso:
     + El sistema genera un comprobante y actualiza el estado de cuenta del residente.
   * Si el pago falla:
     + El residente recibe una notificación de error y se le solicita intentar de nuevo.
6. **Aplicación de Multa (si corresponde)**:
   * Si el pago se realiza después de la fecha de vencimiento, el sistema aplica automáticamente una multa.
7. **Notificación de Confirmación**: El sistema envía un correo electrónico confirmando el pago o informando sobre la multa.
8. **Fin**: El proceso finaliza con la actualización del estado de cuenta y la generación del comprobante de pago.

| UML-VISTA DE DESPLIEGUE |
| --- |
| Sistema de Gestión de Condominios |
|  |

**Servidor de Aplicaciones**:

* **Aplicación Web de Gestión de Condominios**: Proporciona una interfaz amigable para todos los actores del sistema (residentes, administradores, conserjes). Permite la consulta de gastos comunes, reservas de espacios, y la gestión de pagos.

**Servidor de Base de Datos**:

* **Base de Datos del Condominio**: Almacena toda la información relevante, incluyendo detalles de los residentes, gastos comunes, reservas de áreas comunes y registros de pago. Este servidor es fundamental para que la aplicación funcione correctamente, ya que procesa las consultas y actualizaciones de datos.

**Servidor de Correo**:

* **Sistema de Notificación**: Se encarga de enviar correos electrónicos automáticos a los residentes sobre sus gastos comunes, confirmaciones de reservas y alertas de pagos atrasados. Esto ayuda a mantener informados a todos los usuarios del sistema.

| UML-VISTA FÍSICA – DIAGRAMA INFRAESTRUCTURA |
| --- |
| Sistema Gestión De Condominios |
|  |

**Centro de Datos Principal**: Es el corazón del sistema donde se aloja la infraestructura central. Incluye servidores dedicados para la aplicación web, base de datos, notificaciones y backups. Todos estos elementos están protegidos por un firewall para garantizar la seguridad de los datos.

**Red del Condominio**: Esta red incluye los dispositivos de acceso dentro del condominio, como las estaciones de trabajo del administrador y del conserje, así como el control de acceso y las cámaras de seguridad. La red local se conecta al Centro de Datos a través del Router Principal.

**Conexiones Externas**: El acceso a Internet está proporcionado por un Proveedor de Servicios de Internet (ISP) y se distribuye entre el centro de datos y la red del condominio.

**Dispositivos de los Residentes**: Aunque no se muestran explícitamente en este diagrama, los residentes pueden conectarse desde sus dispositivos personales a la aplicación web ubicada en el servidor de aplicaciones

| Usabilidad y Calidad de software- Front-end -Vistas Principales |
| --- |
| Vistas principales del sistema – Vista Home Principal de entrada |
|  |

La **Vista Front-End** del sistema de gestión de condominios ha sido diseñada para ofrecer una experiencia de usuario intuitiva, moderna y fácil de navegar. La interfaz está dividida en diferentes secciones con menús y paneles dinámicos que se adaptan a cada tipo de usuario, brindando acceso a funcionalidades específicas según su rol (Administrador, Residente, Conserje, Directiva). Se presenta un diseño limpio y ordenado que resalta las opciones más importantes, permitiendo a los usuarios interactuar con el sistema de manera eficiente

**Estilo Moderno y Profesional**: Colores sobrios y elementos gráficos que se destacan sin distraer, utilizando una paleta de colores asociada a la identidad visual del condominio.

**Interactividad y Usabilidad**: Uso de gráficos animados, transiciones suaves y elementos interactivos que facilitan la navegación.

**Iconografía Clara**: Uso de iconos intuitivos para representar cada módulo y función, ayudando a los usuarios a ubicar rápidamente la opción deseada

| Usabilidad y Calidad de software – Vistas específicas de usuario |
| --- |
| Vistas principales del sistema – Vista Principal de Usuario a escritorio de trabajo principal de su componente de sistema |
|  |

Cada tipo de usuario tiene un escritorio personalizado con acceso a las funcionalidades correspondientes a su rol:

* **Administrador**: Control total de finanzas y reservas.
* **Residente**: Visualización y pago de gastos comunes, consulta de reservas y gestión de su perfil.
* **Conserje**: Validación de reservas y reportes de uso de áreas comunes.
  1. Consistencia en la cantidad de vistas de la arquitectura.

Se garantiza que cada vista esté alineada con los requisitos funcionales y no funcionales del sistema, evitando redundancias y manteniendo la coherencia en toda la arquitectura

* + 1. DESCRIPCION DE MODULOS

Ejemplos:

| Nombre del módulo | Descripción | Componentes incluidos |
| --- | --- | --- |
| Gestión de Gasto Comunes | Permite calcular y registrar gastos comunes. | Módulo de cálculo, Módulo de reportes, Módulo de facturación |
| Reserva de Espacios Comunes | Controla la reserva y uso de espacios comunes | Módulo de reservas, Módulo de notificaciones, Módulo de reportes |
| Gestión de Usuarios | Administración de cuentas y roles | Módulo de registro, Módulo de autenticación, Módulo de perfiles |

* + 1. DESCRIPCIÓN DE COMPONENTES

| Nombre del Componente | Descripción | Componentes Relaciones |
| --- | --- | --- |
| Calculadora de Gastos | Realizar cálculos de gastos basados en tarifas y consumos | Módulo de Gastos, módulo de Reportes |
| Motor de Notificaciones | Envía notificaciones por correo y mensajes emergentes | Módulo de Reservas, Módulo de Gastos |
| Gestor de Usuarios | Maneja la creación y eliminación de Usuarios | Módulo de Gestión de Usuarios, Módulo de Seguridad |

* + 1. DESCRIPCIÓN DE CONECTORES

| Nombre del Conector | Tipo | Propósito |
| --- | --- | --- |
| Rest API | Comunicación | Transferencia de datos entre módulos |
| Socket.io | Comunicación | Actualizaciones en tiempo real |

* + - 1. **Arquitectura lógica**
      2. Performances

El sistema está diseñado para manejar una alta concurrencia, soportando más de 1000 usuarios simultáneamente sin comprometer el rendimiento. Para lograr tiempos de respuesta inferiores a 2 segundos por transacción, se implementarán las siguientes estrategias:

* **Escalabilidad Horizontal**: Se utilizarán contenedores Docker para desplegar microservicios en múltiples instancias en la infraestructura de AWS, permitiendo la adición de recursos según la demanda.
* **Balanceo de Carga**: Se implementará un balanceador de carga que distribuye las solicitudes de los usuarios entre las diferentes instancias de los microservicios, asegurando una distribución equitativa del tráfico y evitando sobrecargas en un único servicio.
* **Optimización de Consultas**: Se aplicarán técnicas de optimización en las consultas a la base de datos PostgreSQL, incluyendo índices y consultas asíncronas para reducir los tiempos de respuesta.
  + - 1. Criterios de Calidad

El sistema debe cumplir con los siguientes criterios de calidad:

* **Disponibilidad 24/7**: La arquitectura se diseñará para asegurar un tiempo de actividad continuó, implementando redundancia y recuperación ante desastres en la infraestructura de AWS.
* **Respuesta Rápida**: Todas las transacciones deberán completarse en menos de 2 segundos, lo que se logrará mediante el uso de técnicas de caché y un diseño eficiente de la API.
* **Alta Seguridad**: Se implementará cifrado AES-256 para proteger datos sensibles, así como autenticación y autorización robustas para garantizar la integridad de la información y la privacidad del usuario.
  + - 1. Detalles de la implementación

La implementación del sistema se basará en una arquitectura de microservicios, utilizando las siguientes tecnologías:

* **Backend**: Node.js con Express para la creación de APIs RESTful.
* **Frontend**: React junto con Material-UI para la creación de una interfaz de usuario moderna y responsiva.
* **Base de Datos**: PostgreSQL para la gestión eficiente de datos estructurados.
* **Infraestructura**: AWS utilizando contenedores Docker para una gestión flexible y escalable de los microservicios.
  + - 1. Lenguajes y plataformas

Las tecnologías y versiones específicas que se utilizarán en el desarrollo del sistema incluyen:

* **Node.js**: v10.x
* **Express**: v4.x
* **React**: v16.x
* **Material-UI**: v4.x
* **PostgreSQL**: v9.6
* **Docker**: v17.x
* **AWS**: Servicios de EC2, RDS y ECS.
* **Python**: v3.7
* **Visual Studio Code**: v1.50
  1. Acceso a fuentes de información del proyecto

Para acceder a la carpeta de evidencias acumuladas y otros documentos relevantes del proyecto, puede utilizar el siguiente enlace:  
  
Insertar link: Carpeta de Evidencia del Proyecto  
  
Esta carpeta incluye documentación técnica, diagramas, pruebas de rendimiento y cualquier otra evidencia relacionada con el desarrollo y la implementación del sistema