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 web diseñado para automatizar la administración de tareas y procesos comunes dentro de AndesK9, una empresa de adiestramiento canino.

El sistema propuesto busca mejorar la eficiencia en la gestión de visitas de entrenamiento, la venta de servicios adicionales y la administración de la base de datos de clientes, facilitando el flujo de trabajo del personal y la experiencia de los usuarios.

El sistema desarrollado será implementado en una infraestructura web accesible, optimizada para dispositivos de escritorio y móviles, lo cual permitirá su uso tanto en oficinas administrativas como en el campo. Además, proporcionará una interfaz intuitiva y accesible para todos los usuarios, asegurando que las gestiones se realicen de manera eficiente y organizada, cumpliendo con los estándares modernos de desarrollo web y seguridad de dato

1. **Propósito**

El propósito de este documento es servir como guía para el diseño e implementación de la arquitectura del sistema *AndesK9*. 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 eficazmente las visitas y servicios para una empresa de adiestramiento canino en un esquema de administración unificado.

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

1. **Alcance**

El sistema gestionará las siguientes áreas:

* Base de datos
* Gestion de tienda y servicios
* Gestión de visitas
* Gestión de pagos
* Gestión de contacto
* Gestión de noticias y actividades

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 | Andesk9 |
| Product Owner | Nelson |
| Desarrolladores Backend | Giovanni piña alvarado |
| Desarrolladores Frontend | Giovanni piña alvarado |
| DBA-Especialista en Base de Datos | Giovanni piña alvarado |
| Tester/QA | Giovanni piña alvarado |
| Analista de Requisitos | Giovanni piña alvarado |
| Administrador Plataformas | Giovanni piña alvarado |

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 xxxx 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 |
| --- | --- | --- |
| Nelson | Administrador | Administrar distintas funciones de la pagina tales como:citas, servicios, cliente y actividades. |
| Cliente | Cliente | comprar servicios. |
| Nelson | adiestrador | revisar horas agendadas. |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

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:  Agendar cita  Crear hora | 2 |
| Vista Física | Diagrama Topológico de Servicios | 1 |
| Vista de Despliegue | Diagrama de componentes  Diagrama de Capas (MVC) | 1  1 |

* 1. Vistas de la arquitectura

| UML-VISTA DE ESCENARIO |
| --- |
| Diagrama de Caso de uso extendido /Subproyecto [Inserte nombre del Subproyecto asignado] |
|  |

### **Actor: Cliente**

* **Menú inicio**: Punto de entrada para el cliente donde puede acceder a otras funcionalidades.
  + **Iniciar sesión**: Caso de uso para que el cliente inicie sesión en el sistema.
    - **Reserva servicio**: Permite al cliente reservar un servicio.
      * **Agendar citas**: Una vez reservado el servicio, el cliente puede agendar una cita.
        + **Realizar pago**: Tras agendar la cita, el cliente puede realizar el pago.
  + **Solicitar contacto**: Permite al cliente solicitar un contacto para obtener ayuda o más información.
  + **Ver actividades**: Extiende el menú de inicio y permite al cliente ver las actividades relacionadas con sus servicios o citas.

### **Actor: Admin**

* **Menú administración**: Punto de entrada para el administrador, que incluye las siguientes funcionalidades de gestión.
  + **Crear horas**: Caso de uso para que el administrador configure y cree horarios de disponibilidad.
  + **Administrar usuarios**: Permite al administrador gestionar los usuarios en el sistema.
    - **Crear usuario**: Opción para agregar un nuevo usuario al sistema.
    - **Eliminar usuario**: Extiende la funcionalidad de administración de usuarios y permite la eliminación de usuarios existentes.

### **Actor: Entrenador**

* **Menú entrenador**: Punto de entrada para el entrenador.
  + **Ver visitas**: Permite al entrenador ver las visitas programadas con los clientes.

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

El **Diagrama de Clases** para el sistema de gestión de tienda de servicios 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:** userId, name, email, password, role, imagePath
    - **Métodos:** register(), login(), logout(), viewProfile(), updateProfile()
  + **Admin (Heredado de Usuario):**
    - **Atributos:** userId, name, email, password, role, imagePath
    - **Métodos:** register(), login(), logout(), viewProfile(), updateProfile(), deleteUser()
  + **Cliente (Heredado de Usuario):**
    - **Atributos:** userId, name, email, password, role, imagePath
    - **Métodos:** register(), login(), logout(), viewProfile(), updateProfile()
  + **Mascota:**
    - **Atributos:** petId, name, breed
    - **Métodos:** create(), view(), delete(), update()
  + **Servicio:**
    - **Atributos:** userId, name, description, password, price, duration, imagePath
    - **Métodos:** login(), view(), update(), delete()
  + **Actividad:**
    - **Atributos:** activityId, name, description, date, time, location
    - **Métodos:** create(), view(), update(), delete()
  + **Cita (Appointment):**
    - **Atributos:** appointmentId, user (Usuario), service (Servicio), date, time, status
    - **Métodos:** schedule(), cancel(), confirm()
  + **Pago (Payment):**
    - **Atributos:** paymentId, user (Usuario), appointment (Cita), amount, date, method
    - **Métodos:** processPayment(), refund()
  + **Venta (Sale):**
    - **Atributos:** paymentId, user (Usuario), amount
    - **Métodos:** processPayment(), refund()

**3. Relaciones:**

**Usuario - Mascota:**

* Relación de **asociación**: Un usuario puede tener múltiples mascotas (1 a n), pero una mascota pertenece solo a un usuario.

**Usuario - Actividad:**

* Relación de **composición**: Un usuario puede realizar múltiples actividades.

**Usuario - Cita:**

* Relación de **asociación**: Un usuario puede agendar varias citas (1 a n) para diferentes servicios.

**Usuario - Pago:**

* Relación de **asociación**: Un usuario puede realizar múltiples pagos para sus citas.

**Cita - Servicio:**

* Relación de **asociación**: Una cita está relacionada con un solo servicio, y cada servicio puede tener múltiples citas.

**Venta - Usuario:**

* Relación de **asociación**: Una venta está asociada a un usuario que realizó el pago.

**Pago - Venta:**

* Relación de **composición**: Una venta está vinculada a un pago, y cada pago puede usarse en una venta específica.

| 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**: ID\_Usuario (PK), Nombre (name\_user), Contraseña (pass\_user), Email (email\_user), Teléfono (phone\_user), Imagen (image\_user), ID\_Cupon (id\_coupon, FK).
  + **Mascota**: ID\_Mascota (PK), Nombre (name\_pet), Descripción (descrt\_pet), Imagen (image\_pet), ID\_Usuario (id\_user, FK).
  + **Cita**: ID\_Cita (PK), ID\_Usuario (id\_user, FK), Dirección (address\_appoin), Fecha (date\_appoin), ID\_Mascota (id\_pet, FK).
  + **Cupón de Descuento**: ID\_Cupon (PK), Porcentaje de Descuento (discount\_percent).
  + **Venta**: ID\_Venta (PK), ID\_Cupon (id\_coupon, FK), Valor de Venta (value\_sale), Descripción (descrt\_sale), ID\_Usuario (id\_user, FK).
  + **Producto**: ID\_Producto (PK), Nombre del Servicio (name\_service), Descripción (descr\_service), Valor del Servicio (value\_service), Imagen (image\_service), ID\_Venta (ide\_sale, FK).
  + **Reserva**: ID\_Reserva (PK), Tiempo de Reserva (reservation\_time), Duración en Semanas (duration\_weeks), Estado (status), Ubicación (location), ID\_Usuario (id\_user, FK), ID\_Servicio (id\_service, FK).
  + **Aviso**: ID\_Aviso (PK), Tipo de Aviso (tipe\_notice), Nombre del Aviso (name\_notice), Descripción (desct\_notice), Imagen (imag\_notice), ID\_Usuario (id\_user, FK).
* **Relaciones**:
  + Un **Usuario** puede tener múltiples **Mascotas**.
  + Un **Usuario** puede tener múltiples **Citas**.
  + Un **Usuario** puede usar múltiples **Cupones de Descuento**.
  + Un **Usuario** puede realizar múltiples **Ventas**.
  + Una **Mascota** pertenece a un solo **Usuario**.
  + Una **Cita** está asociada a un solo **Usuario** y a una sola **Mascota**.
  + Un **Cupón** de Descuento puede ser utilizado en múltiples **Ventas**.
  + Una **Venta** puede incluir múltiples **Productos**.
  + Un **Producto** puede estar asociado a una única **Venta**.
  + Una **Reserva** está vinculada a un **Usuario** y a un **Producto**.
  + Un **Aviso** puede ser creado por un **Usuario**.

| 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**:agendar cita

1. **Login**: El Cliente inicia sesión en el sistema.
2. **validar datos**: El sistema valida las credenciales".
3. **Seleccionar Servicio**: El cliente escoge un servicio a elección
4. **Seleccionar comuna**: El cliente escoge una comuna e ingresa la dirección en la que se realizarán las citas.
5. **Seleccionar hora**: el cliente escoge la hora disponible para realizar la cita.
6. **Seleccionar medio de pago**: El cliente ingresa los datos bancarios para el pago del servicio.
7. **Ingresar datos bancarios**:El cliente ingresara los datos de su tarjeta bancaria por el portal de pago:
8. **Validar datos**:
   1. Si los datos son correctos:
      1. Se realizará el cobro al cliente.
   2. Si los datos son incorrectos:
      1. El cliente recibe una notificación de error y se le solicita intentar de nuevo.
9. **Mostrar mensaje de éxito:**Se mostrará un mensaje de éxito.
10. **Crear hora de cita**: El sistema agendará la hora para las citas del adiestrador.
11. **Fin**: El proceso finaliza y se enviará el comprobante al correo del cliente.

**Componentes**:Crear hora.

1. **Login**: El administrador inicia sesión en el sistema.
2. **validar datos**: El sistema valida las credenciales".
3. **Menú de administración de horas**: El administrador escoge un servicio a elección
4. **Ver horas disponibles**: El cliente escoge una comuna e ingresa la dirección en la que se realizarán las citas
   1. **seleccionar hora disponible**:el administrador selecciona una hora disponible en la cual puede ver los datos de esta hora.
   2. **eliminar hora:**El administrador elimina una hora disponible
5. **Seleccionar comuna:** El administrador selecciona una comuna:
6. **Ingresar hora:** El administrador ingresa una hora disponible:
7. **validar datos:** El sistema valida junto a la base de datos que esa hora esté disponible en esa comuna:
   1. **Si está disponible**:Ingresa la hora disponible;
   2. **Si no está disponible**: El sistema mostrará un mensaje solicitando el ingreso de otra hora.
8. **Mostrar mensaje de éxito**: El sistema mostrará un mensaje de éxito al crear la hora.
9. **Crear hora de cita**: Se crea y almacena la nueva hora disponible.
10. **Fin**: Finaliza el proceso y envía un correo de confirmación con la nueva hora disponible.

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

**Servidor de Aplicaciones**:

* **Aplicación Web de Gestión de servicios**: Proporciona una interfaz amigable para todos los actores del sistema (clientes, administradores, adiestradores). Permite la consulta de citas, servicios y la gestión de pagos.

**Servidor de Base de Datos**:

* **Base de Datos del Gestión de servicios**: Almacena toda la información relevante, incluyendo detalles de los clientes, pagos, citas 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 clientes y administradores sobre sus confirmaciones de reservas y solicitud de contacto. 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.

**Lógica del negocio**: Esta red incluye los dispositivos de acceso dentro de la aplicación, como los servicios y citas.

**Conexiones Externas**: El acceso a Internet está proporcionado por un Proveedor de portal de pagos y chatbox.

**Base de datos**: Servicio donde almacena los datos correspondientes al sistema y alojamiento de imágenes.

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

La **Vista Front-End** de AndesK9 está diseñada para brindar una experiencia de usuario moderna, amigable y adaptada a las necesidades tanto de los clientes como del equipo administrativo. La interfaz incluye menús y paneles específicos para cada rol de usuario (Administrador, Cliente y Entrenador), lo que permite acceder a las funciones específicas de manera eficiente.

**Estilo Moderno y Profesional:** La interfaz de AndesK9 utiliza una paleta de colores cálidos y naturales, inspirados en la naturaleza de los Andes y el entorno de los animales, lo que crea un ambiente acogedor y profesional. Los elementos gráficos se integran de forma discreta para mantener el enfoque en las funciones principales.

**Interactividad y Usabilidad:** La interfaz incluye gráficos dinámicos y transiciones suaves que mejoran la experiencia del usuario. Elementos interactivos y retroalimentación visual (como botones de estado y notificaciones) ayudan a los usuarios a navegar fácilmente por las diferentes secciones de la plataforma.

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

**Diseño Adaptativo:** La plataforma está diseñada para ser totalmente responsiva, adaptándose tanto a dispositivos de escritorio como móviles, lo que permite a los usuarios acceder a la información desde cualquier lugar y en cualquier momento.

**Experiencia Personalizada según el Rol:** Cada usuario tiene acceso a una vista específica con funciones ajustadas a su rol. Los administradores tienen acceso a menús de gestión (administración de usuarios, creación de horarios, etc.), los clientes pueden reservar servicios y ver actividades, mientras que los entrenadores acceden a su propio panel para gestionar sus horarios y ver las citas programadas.

| 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**:MySql 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**: v18.17.1
* **React**: v18.1
* **Visual Studio Code**: v1.94
  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