Documento arquitectónico final

Vanessa Tobón Pérez
Maria Andrea Nieto Valencia

Institución Universitaria Pascual Bravo
Medellín
2025

Tabla de contenido

1.0. Análisis del dominio

1.1. Introducción al negocio

Hoy en día, muchas instituciones y comunidades todavía gestionan las reservas de canchas de forma manual (libretas, llamadas o acuerdos verbales), lo que genera problemas como:

- Confusión por doble reserva o traslape de horarios.
- Dificultad para llevar control de pagos, historial y disponibilidad.
- Falta de transparencia para los clientes.

Con este sistema web de Reserva de Canchas Deportivas buscamos resolver estos problemas ofreciendo una plataforma digital fácil de usar y accesible desde cualquier navegador.

1.1.2. Objetivos estratégicos

- Optimizar el uso de las canchas disponibles.
- Facilitar la experiencia de reserva a los usuarios.
- Dar más confianza y transparencia tanto a los usuarios como a los
- administradores.
- Automatizar procesos administrativos (reservas, pagos, horarios).
- Escalar el servicio para soportar múltiples escenarios (clubes, colegios,
- barrios).

1.2. Análisis de actores y procesos clave

1.2.1. Clasificación de actores.

Primarios:

- Cliente: realiza el registro, consulta disponibilidad y reserva canchas.
- Administrador: gestiona canchas, horarios, usuarios y reservas.

Secundarios:

- Visitantes no registrados: personas que entran al sistema para ver información básica, pero que necesitan registrarse si quieren reservar.

Soporte:

- Equipo técnico/desarrolladores: se encargan de mantener el sistema funcionando y ayudar en caso de fallas o mejoras.

1.2.2. Procesos críticos que el sistema debe soportar

- Registro e inicio de sesión.
- Consulta de canchas disponibles.
- Reserva de cancha.
- Gestión de canchas y horarios.
- Gestión de reservas (confirmar y cancelar).

1.2.3. Representación en una matriz: diagrama de actores y procesos

ID	Tipo	Nombre del actor	Procesos en los que se ve involucrado
A1	Primario	Cliente (Usuario registrado)	Registro e inicio de sesión, Consulta de canchas disponibles, Reserva de cancha, Gestión de sus reservas (confirmar/cancelar).
A2	Primario	Administrador	Gestión de canchas y horarios, Gestión de usuarios, Gestión de reservas (confirmar y cancelar), Monitoreo del sistema.
A3	Secundario	Visitante no registrado	Navegación en la página, Consulta de información básica (sin posibilidad de reservar).
A4	Soporte	Equipo técnico / Desarrolladores	Mantenimiento del sistema, Soporte técnico, Implementación de mejoras.

1.3. Modelo de dominio inicial

1.3.1. Entidades principales y atributos básicos

- **Usuarios**: Id usuario, nombre, email, contraseña, teléfono, dirección, fecha registro, rol.
- **Cancha**: Id cancha, nombre cancha, tipo cancha, estado, hora apertura, hora cierre, id municipio, id estado, id país, dirección, capacidad, precio.

- **Estado**: Id estado, nombre estado, id país.
- **Municipio**: Id municipio, nombre municipio, id estado, id país.
- **País**: id país, nombre país.
- **Reservas**: Estado, fecha reserva, hora fin, hora inicio, id cancha, id reserva, id usuario, motivo cancelación

1.3.2. Relaciones y multiplicidad

- Un Usuario puede realizar muchas Reservas.
- Una Cancha puede tener muchas Reservas.
- Un País puede tener muchos Estados.
- Un Estado puede tener muchos Municipios.
- Un Municipio puede tener muchas Canchas.

El dominio debe tener la relación entra (país-canchas) (estado-canchas) para poder obtener la dirección, sin embargo, el único que tiene multiplicidad directa con la entidad “canchas" es la entidad “municipios” en términos de ubicación.

1.3.3. Reglas de negocio.

Registro de usuarios

- Todo usuario debe registrarse con un correo electrónico único y una contraseña válida.
- Un usuario debe tener asignado un rol (cliente o administrador).

Gestión de canchas

- Una cancha debe pertenecer a un municipio, el cual a su vez pertenece a un estado y este a un país.
- Una cancha puede estar en estado disponible o no disponible.
- Cada cancha debe tener definido al menos un horario para que pueda reservarse.

Reservas

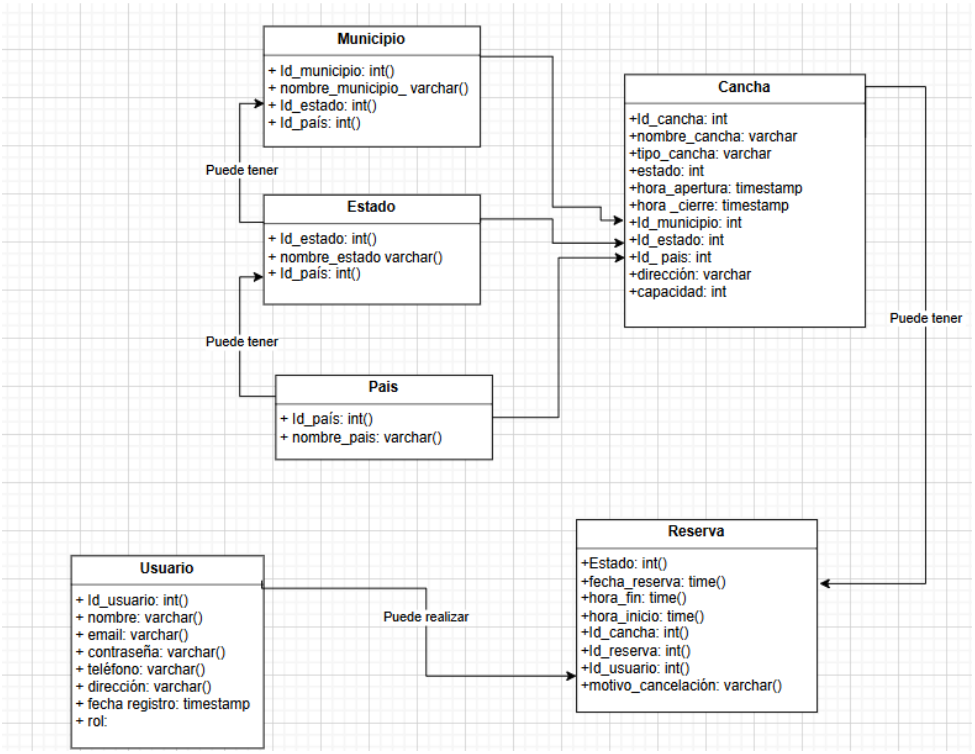
- Un usuario puede realizar múltiples reservas, pero una reserva solo pertenece a un usuario.

- Una reserva debe estar asociada obligatoriamente a una cancha y a un horario disponible.
- El sistema no debe permitir que dos personas reserven la misma cancha en el mismo horario.
- Toda reserva debe tener un estado: confirmada, cancelada o completada.
- Una reserva debe registrar fecha y horas de inicio y fin.

Localización

- Cada municipio pertenece a un estado y cada estado a un país.
- Una cancha debe estar ubicada dentro de un municipio válido.

1.3.4. Diagrama de clases UML del dominio



1.4. Validación del modelo

1.4.1. Contraste con requisitos funcionales

ID	Nombre	Descripción (qué debe hacer el sistema)	Actores
RF-01	Registro de usuario	Permitir que una persona cree una cuenta con correo único y contraseña válida.	Cliente, Administrador

RF-02	Inicio de sesión	Autenticar usuarios registrados para acceder a las funciones del sistema.	Cliente, Administrador
RF-03	Consulta de disponibilidad	Permitir a los usuarios buscar canchas disponibles por fecha, hora y ubicación.	Cliente, Visitante
RF-04	Crear reserva	Registrar reservas asociando usuario, cancha y horario.	Cliente
RF-05	Validación de traslape de horarios	Evitar que dos reservas se crucen en la misma cancha y franja horaria.	Cliente, Administrador
RF-06	Gestión de estado de reservas	Toda reserva debe tener un estado (confirmada, cancelada, completada) y poder cambiar de uno a otro.	Cliente, Administrador
RF-07	Cancelar o reprogramar	El usuario puede cancelar o mover una reserva dentro de las políticas definidas.	Cliente
RF-08	Gestión de canchas	Crear, modificar y administrar información de cada cancha (nombre, tipo, estado, capacidad, ubicación, precio).	Administrador
RF-09	Definir apertura y cierre	Configurar las horas de apertura y cierre de cada cancha.	Administrador
RF-10	Gestión de localización	Registrar y relacionar País → Estado → Municipio y asociarlos a las canchas.	Administrador

1.4.2. Relación con requisitos no funcionales (seguridad, rendimiento, escalabilidad, disponibilidad)

ID	**Nombre**	**Descripción (cómo debe comportarse el sistema)**	**Relación con RF**	**Actores**
RNF-01	Seguridad	Manejo seguro de contraseñas, sesiones protegidas y permisos según el rol elegido.	RF-01, RF-02, RF-11	Cliente, Administrador

RNF-02	Disponibilidad	El sistema debe estar disponible casi siempre, con un tiempo de actividad de al menos 99.9% al mes.	Todos los RF	Cliente, Administrador
RNF-03	Rendimiento	Al consultar qué canchas están libres, el sistema debe responder en máximo 2 segundos, y todo el proceso de reserva no debería tomar más de 5 segundos.”	RF-03, RF-04, RF-05, RF-07	Cliente
RNF-04	Escalabilidad	El sistema debe crecer sin perder rendimiento, soportar al menos 50 usuarios conectados al mismo tiempo y 100 reservas mensuales.	RF-03, RF-04, RF-08, RF-09	Cliente, Administrador
RNF-05	Usabilidad	El flujo de reserva debe ser **interactivo, sencillo y eficaz**, pensado para que cualquier persona pueda usarlo sin dificultad.	RF-03, RF-04, RF-07	Cliente, Visitante
RNF-06	Integridad de la información	Al crear o cancelar reservas, los cambios deben aplicarse de manera confiable y consistente, sin dejar datos incompletos o duplicados.	RF-04, RF-05, RF-06, RF-07	Cliente, Administrador
RNF-07	Privacidad de datos	Los datos personales deben estar protegidos, tanto en almacenamiento como en transmisión, cumpliendo con buenas prácticas de privacidad.	RF-01, RF-02, RF-11	Cliente

1.5. Documento de atributos priorizados

1.5.1. Selección de atributos de calidad relevantes

- Disponibilidad.
- Escalabilidad.
- Seguridad.
- Rendimiento.

- Usabilidad.

1.5.2. Priorización con justificación desde el negocio. ¿por qué este atributo es crítico?

Disponibilidad (Alta prioridad):

- Crítica porque el negocio depende de que los clientes puedan reservar 24/7.
- Una caída impacta directamente en ingresos y experiencia del usuario.

Seguridad (Alta prioridad):

- Los datos de los usuarios (contraseñas, teléfonos, correos) son sensibles y cualquier filtración afectaría la confianza en la plataforma.

Rendimiento (Media-Alta prioridad):

- Una reserva lenta (más de 2–3 segundos) genera abandono del proceso y pérdida de clientes.
- Relacionado con la experiencia del usuario.

Escalabilidad (Media-Alta prioridad):

- El sistema puede iniciar con pocos usuarios, pero debe estar preparado para crecer hacia más canchas y más usuarios en el futuro.

Usabilidad (Media prioridad)

- La facilidad de uso mejora la adopción del sistema, especialmente entre usuarios con baja alfabetización digital. Aunque no es tan crítico como seguridad o disponibilidad, sí impacta en la experiencia de cliente y en la reducción de soporte técnico.

1.5.3. Definición de métricas asociadas.

Disponibilidad:

- El sistema debe estar en línea y funcionando casi siempre, con una disponibilidad de al menos 99.9% al mes (solo se permitirían caídas muy cortas).

Seguridad:

- Las contraseñas de los usuarios deben guardarse de forma protegida, usando métodos seguros que las hagan ilegibles en caso de robo.
- Cada persona debe tener permisos según su rol (ejemplo: el cliente no puede hacer lo mismo que un administrador).

Rendimiento:

- Las consultas más importantes, como ver qué canchas están libres, deben responder en menos de 2 segundos.
- Todo el proceso de hacer una reserva debe tardar como máximo 5 segundos en completarse.

Escalabilidad:

- El sistema debe poder manejar, en la primera etapa, hasta 50 personas conectadas al mismo tiempo sin fallar.
- También debe ser capaz de procesar al menos 100 reservas al mes sin que se ponga lento.

Usabilidad:

- La interfaz debe ser clara, legible y accesible, permitiendo que cualquier usuario pueda completar una reserva sin necesidad de ayuda.
- Curva de aprendizaje máxima de 10 minutos para un usuario nuevo (medible mediante pruebas de usabilidad), tasa de error en tareas básicas < 5%.

1.5.4. Relación entre drivers de negocio y métricas de calidad

Driver de negocio	**Atributo de calidad**	**Métrica asociada**
Necesidad de disponibilidad 24/7	Disponibilidad	Tiempo de actividad ≥ 99.9%
Confianza del usuario y protección de datos	Seguridad	Cifrado de contraseñas, roles diferenciados
Experiencia ágil y sin frustraciones	Rendimiento	Respuesta < 2s en consultas, reserva < 5s
Crecimiento proyectado del sistema a futuro	Escalabilidad	50 usuarios concurrentes, 100 reservas mensuales

Facilidad de uso para todo tipo de cliente	Usabilidad	Reserva en máximo 3-4 clics, 80% de usuarios de prueba completan una reserva sin ayuda

2.0. Incorporación de patrones de diseño

2.1. Selección de patrones.

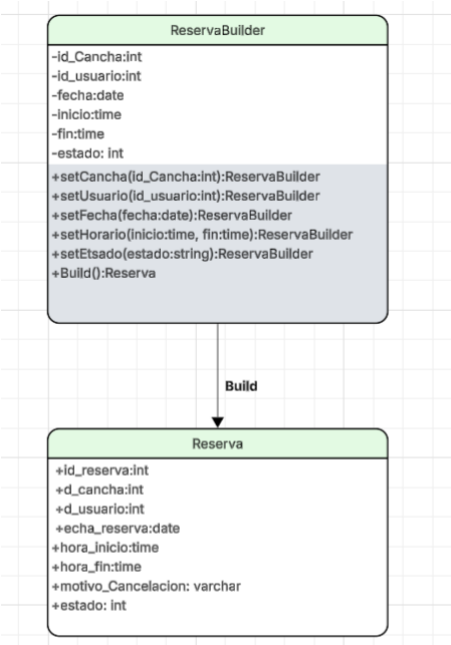
2.1.1. Builder (Creacional) — Módulo: Reservas

Problema que resuelve: Crear una reserva puede variar según la información ingresada, por ejemplo, la cancha, la fecha y el horario. Si el diseño no es bueno, el código corre el riesgo de volverse rígido, difícil de entender para otros desarrolladores, complicado de mejorar en el futuro y poco reutilizable.

Justificación: El Builder ayuda a construir reservas fáciles, guiando el proceso por pasos, esto separa la creación de la representación final, lo que brinda una gran flexibilidad al momento de modificar el código sin afectar otras partes del sistema. También fomenta un diseño claro y entendible, facilitando el trabajo entre desarrolladores y la evolución en el futuro.

Alternativa descartada (Constructores): Usar constructores largos con múltiples parámetros puede complicar la comprensión y el mantenimiento del código. Cuando el número de argumentos crece se comienza a hacer más difícil recordar el orden correcto y entender qué representa cada uno, lo que aumenta el riesgo de errores.

Cómo se implementará en el código: La clase ReservaBuilder va a permitir crear objetos de la tabla tbl_reservas de forma ordenada y flexible, sin la necesidad del uso de constructores. Con métodos específicos como setCancha, setUsuario, setFecha, setHorario y setEstado, se configuran los atributos paso a paso. Por último, el método build() genera el objeto completo y listo para guardar en la base de datos.



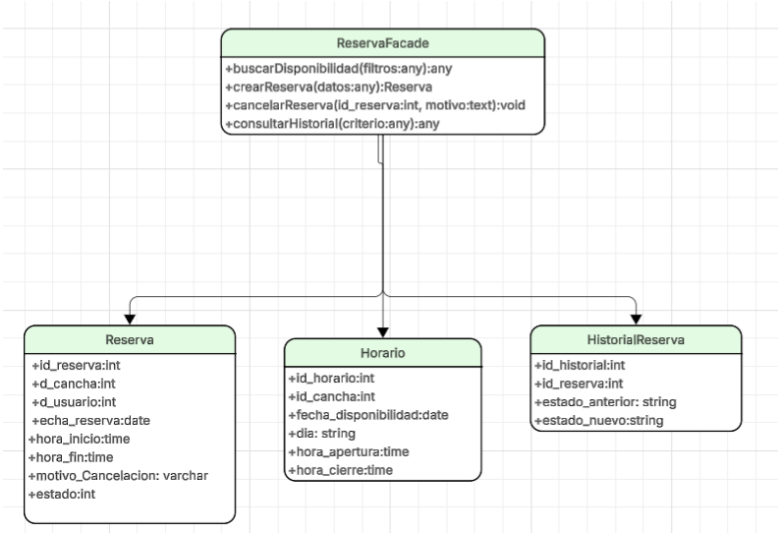
2.1.2. Facade (Estructural) — Módulo: Capa de aplicación/API

Problema que resuelve: El sistema de reservas función a con varias tablas que se encuentran relacionadas entre sí, y para ejecutar operaciones simples, como verificar la disponibilidad de una cancha o consultar el historial de un usuario, es necesario recorrer varias de estas tablas y combinar información, lo cual es complejo y poco eficiente. Y el usuario final, que solo busca hacer una reserva rápida, ve esto como un proceso muy largo y poco fluido.

Justificación: El patrón Facade une y simplifica la interacción con la base de datos proporcionando una interfaz única, lo cual permite que los usuarios y controladores tengan una única interfaz para consultar disponibilidad, crear reservas y consultar el historial, logrando que la capa de aplicación se vuelva más clara, fácil de mantener y en concordancia con las necesidades del sistema.

Alternativa descartada (Controladores): La alternativa inicial era acceder directamente a cada tabla desde el controlador, lo que genera duplicación en el código y mayor complejidad lo que da pie a posibles errores por inconsistencias en las consultas.

Cómo se implementará en el código: Se definirá una clase ReservaFacade que actuará como intermediaria entre los controladores y las tablas más relevantes. Esta clase ofrecerá métodos como buscarDisponibilidad, crearReserva, cancelarReserva y consultarHistorial. Internamente, ReservaFacade gestionará las operaciones necesarias sobre tbl_reservas, tbl_horarios y tbl_historial_reservas, asegurando que la lógica de negocio sea coherente en todo el sistema



2.1.3. Proxy (Estructural) — Módulo: Permisos

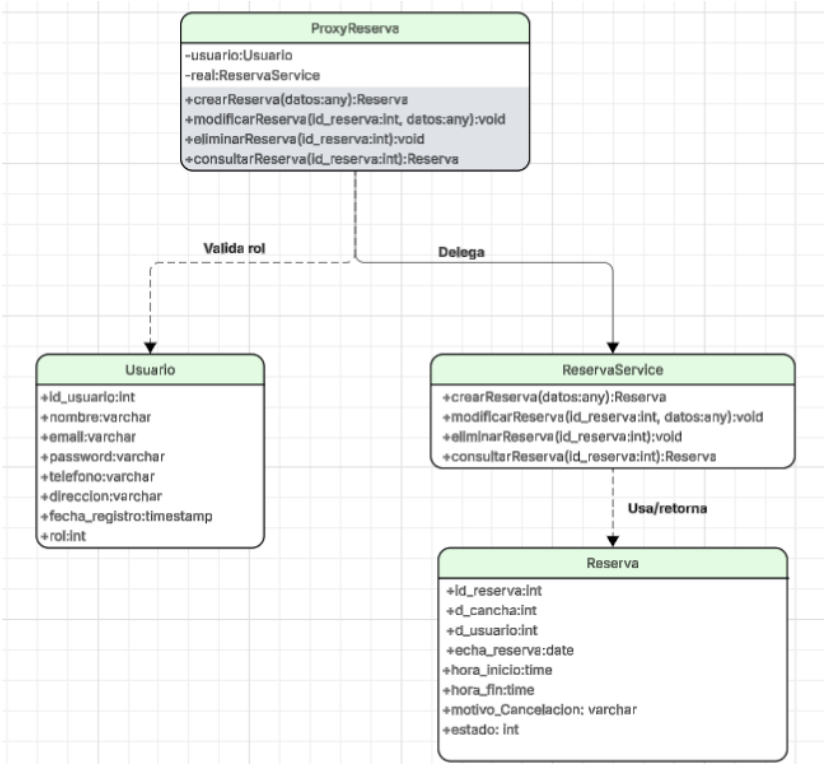
Problema que resuelve: No todos los usuarios tienen los mismos permisos dentro del sistema. Por ejemplo, un cliente no puede modificar canchas ni acceder a toda la información disponible de ellas. Por el contrario, los administradores tienen acceso completo para gestionar las reservas y las canchas. Esta diferenciación de roles ayuda a mantener la seguridad, evitar errores y garantizar que cada usuario interactúe solo con lo que le corresponde.

Justificación: El Proxy permite controlar el acceso a las funcionalidades del sistema dependiendo del rol del usuario (admin o cliente). La capa intermedia que se crea con este patrón actúa como un filtro para el usuario permitiendo identificar de que tipo es. Esta división mejora la seguridad, mantiene el sistema organizado y es más fácil de mantener a

medida que crece.

Alternativa descartada: Una alternativa es validar los permisos directamente dentro de cada método del controlador, verificando en cada operación si el usuario tiene autorización para ejecutarla, pero esto genera duplicidad dentro del código y adicionalmente complica una refactorización futura.

Cómo se implementará en el código: Se construirá el ProxyReserva, encargado de validar el rol del usuario antes de ejecutar acciones de ingresar. Solo los administradores pueden modificar o eliminar reservas, mientras que los clientes tienen el acceso limitado. Esta capa evita violaciones al sistema, mejora la seguridad y organiza mejor el código. También el Proxy facilita la extensibilidad: nuevos roles pueden integrarse fácilmente, lo que refuerza la mantenibilidad y escalabilidad del software.



2.1.4. Decorator (Estructural) — Módulo: Gestión de canchas

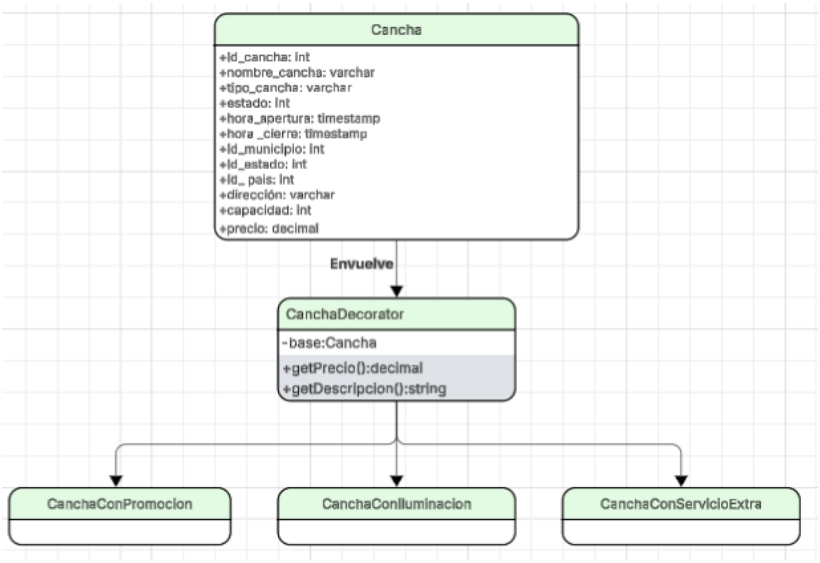
Problema que resuelve: Es necesario que al sistema se le puedan añadir nuevas

características a las canchas registradas en tbl_canchas, como promociones temporales, iluminación nocturna, alquiler de implementos deportivos u otros servicios adicionales. Implementarlas directamente en la clase base de cancha, generaría que el código crezca de forma rígida y sería difícil de mantener, ya que cada vez que se añada o quite una característica habría que modificar la clase original.

Justificación: Decorator permite añadir atributos a las canchas de manera flexible sin modificar su estructura fundamental, lo cual hace posible mejorar la oferta de canchas agregándole elementos extras según las necesidades del negocio. Esto da más versatilidad, conserva la clase original ordenada y simplifica la inclusión de nuevas funcionalidades más adelante.

Alternativa descartada: La alternativa descartada era incluir todos los atributos que pudieran surgir adicionalmente en la tbl_canchas, lo cual generaría una tabla sobrecargada y código difícil de gestionar ya que no serían atributos obligatorios para todas las canchas.

Cómo se implementará en el código: Se definirá una clase base Cancha con sus atributos principales, como nombre, tipo, capacidad y precio. A partir de ella, se podrán implementar decoradores como CanchaConPromocion, CanchaConIluminacion o CanchaConServicioExtra, que añadirán nuevas funcionalidades al objeto sin modificar su estructura original. De este modo, el sistema podrá combinar diferentes decoradores según las necesidades de cada cancha.



2.1.5. Composite (Estructural) — Módulo: Gestión de ubicaciones

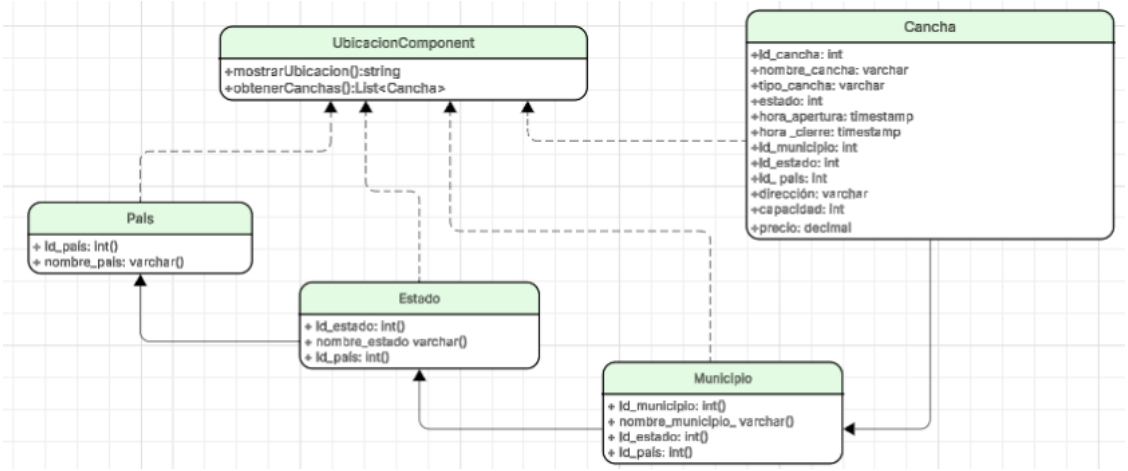
Problema que resuelve: La información de la ubicación de las canchas está dividida en varias tablas relacionadas (país, estado, municipio, cancha). Consultar o recorrer esta jerarquía puede ser tedioso cuando se necesitan realizar búsquedas por ubicación, ya que implica navegar múltiples niveles y unir datos dispersos.

Justificación: Este patrón permite organizar la jerarquía como un árbol, (País → Estado → Municipio → Cancha), facilitando su exploración de manera estable, esto logra que puedan ser manejados individualmente como "nodo" en la misma organización, simplificando búsquedas, uniones y generación de informes.

Alternativa descartada (Múltiples consultas): La alternativa inicial era realizar múltiples consultas separadas a las tablas y unir los resultados de manera manual. Este enfoque se conoce como manejo jerárquico plano, que aumenta la complejidad y duplica lógica en los controladores.

Cómo se implementará en el código: Se definirá una interfaz bicacionComponent que será implementada por Pais, Estado, Municipio y Cancha. Cada uno podrá ser tratado como un nodo del árbol, lo que permite recorrerlos con operaciones uniformes como mostrarUbicacion() o obtenerCanchas(), sin necesidad de que el controlador conozca los

detalles de cada tabla.



2.2. Conclusiones generales sobre los beneficios de aplicar patrones

Incorporar modelos de diseño a nuestra plataforma ofrece beneficios cruciales para la arquitectura, la calidad del software y su adaptabilidad. Inicialmente, fomenta un sistema flexible y claro, minimizando la interdependencia entre componentes. Esto simplifica la comprensión y modificación del sistema, reduciendo errores. Además, una estructura clara con interfaces definidas y métodos estandarizados para la creación de objetos y la coordinación de procesos asegura un código consistente y organizado. Esto facilita la integración de nuevas lógicas de negocio, algoritmos o servicios sin alterar el núcleo del sistema, permitiendo una rápida respuesta a los cambios. Adicionalmente, al distribuir la lógica de manera reutilizable, el mantenimiento y la actualización del sistema se simplifican y economizan a largo plazo. Finalmente, los patrones de diseño mejoran aspectos críticos como la seguridad, el rendimiento, la disponibilidad y la integridad, garantizando un funcionamiento eficaz y fiable en operaciones básicas y avanzadas. En resumen, estas ventajas fortalecen y flexibilizan la estructura del sistema, preparándolo para las operaciones actuales y la expansión futura.

Nota: Al momento de asignar los patrones de diseño evidenciamos que era importante agregar dos tablas más al diagrama de clases inicial, que corresponden a: Horarios e historial de reservas.

3.0. Modelado arquitectónico con UML y modelo C4

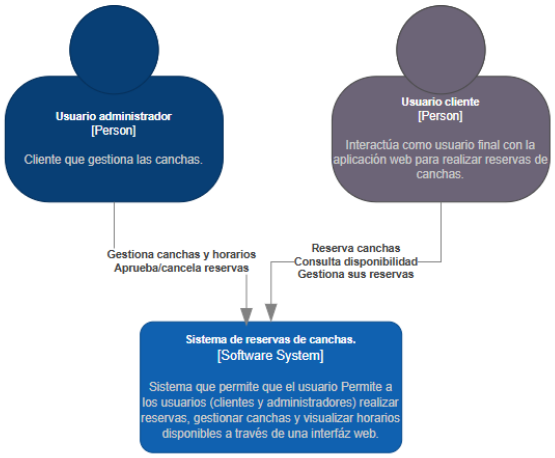
3.1. Modelo C4.

3.1.1. Nivel 1 (contexto)

En la capa #1 representamos la visión general del sistema, es decir, cómo interactúan los usuarios externos con el sistema de reservas.

- Usuario administrador: puede gestionar canchas, horarios y aprobar o cancelar reservas.
- Usuario cliente: puede consultar la disponibilidad, realizar reservas y gestionar sus propias reservas.
- Sistema de reservas de canchas: actúa como el software central, que recibe solicitudes de ambos tipos de usuarios mediante una interfaz web.

Este nivel responde a la pregunta: "¿Quién usa el sistema y qué obtiene de él?"



3.1.2. Nivel 2 (contenedores)

En la capa #2 se muestra la arquitectura lógica y tecnológica del sistema, detallando los contenedores que lo componen y cómo se comunican:

Aplicación Web (Front-End)

- Desarrollada con HTML, CSS, JavaScript y PHP.
- Proporciona la interfaz visual para clientes y administradores.
- Envía peticiones HTTP al backend y recibe vistas o respuestas.

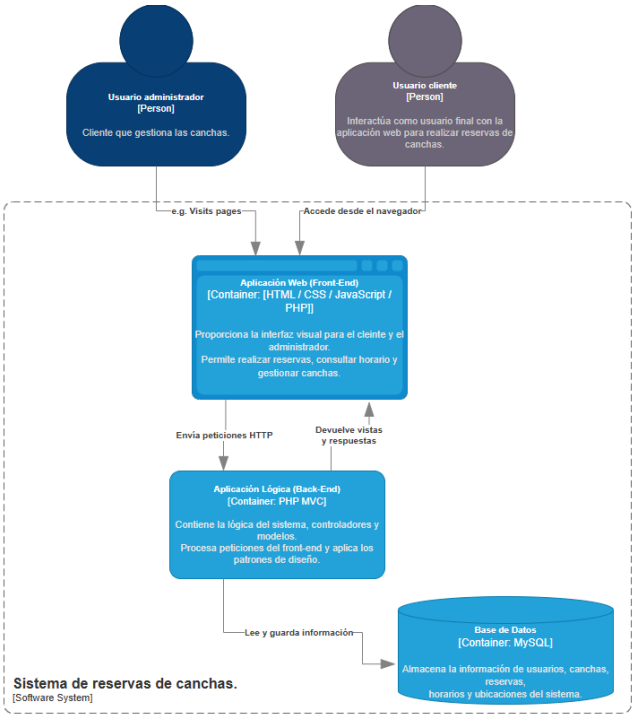
Aplicación Lógica (Back-End PHP MVC)

- Implementa la lógica del negocio, controladores y modelos.
- Aplica los patrones de diseño para mantener modularidad, reutilización y bajo
- acoplamiento.
- Procesa las solicitudes, consulta la base de datos y devuelve resultados.

Base de Datos (MySQL)

- Almacena usuarios, canchas, reservas, horarios y ubicaciones (país, estado,
- municipio).
- Se comunica con el backend a través de consultas SQL.

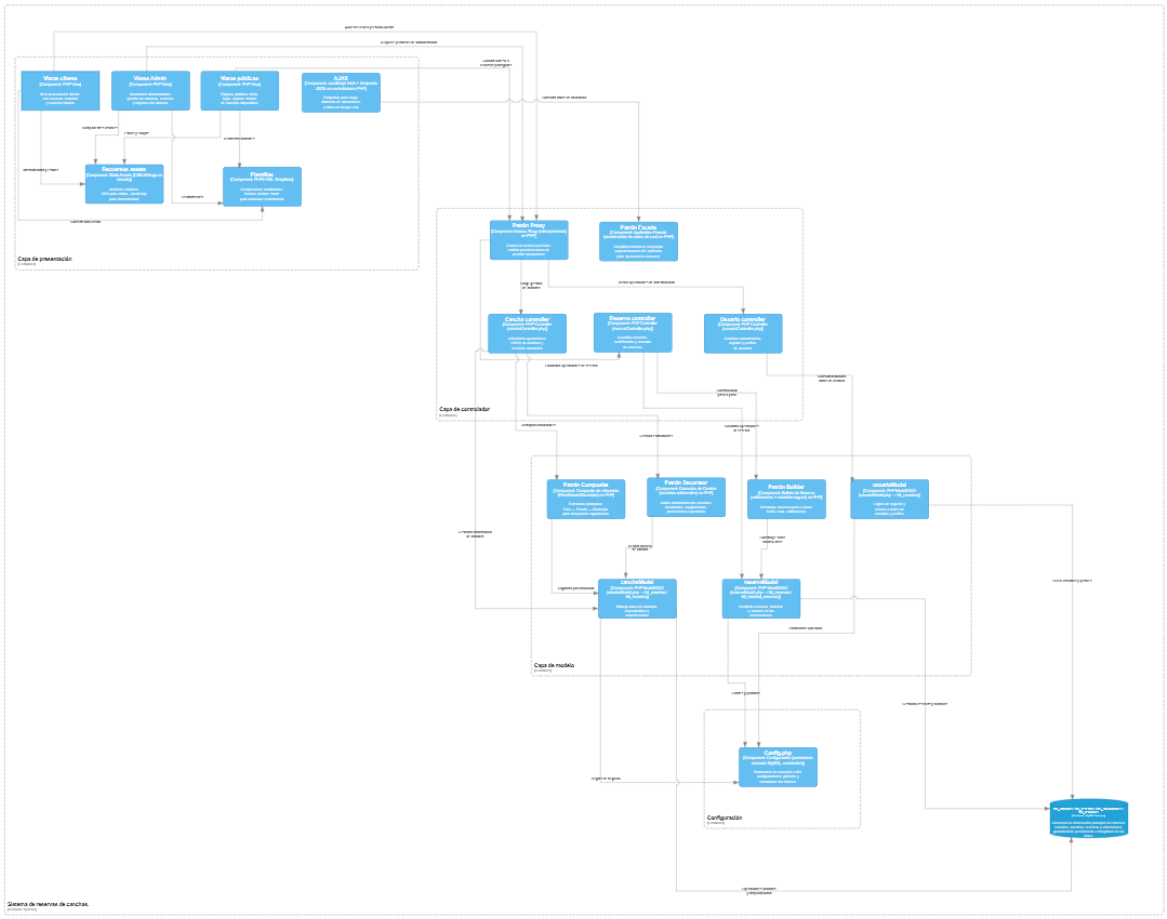
Este nivel responde a la pregunta: "¿Cómo está estructurado el sistema y qué tecnologías usa?"



3.1.3. Nivel 3 (componentes)

En la capa #3 se muestran las partes internas del sistema y como cada una aporta al funcionamiento general. Está separado por capas y a continuación explicaremos de manera resumida a que corresponde cada una.

- **Capa de presentación:** en esta capa incluimos las vistas del cliente, del administrador y las páginas públicas. Las cuales se encargan de mostrar la información al usuario y de enviar solicitudes al servidor a través de AJAX, sin necesidad de recargar la página.
- **Capa de controlador:** Esta capa contiene los controladores que gestionan la comunicación entre las vistas y la lógica del negocio. Aquí se aplican los patrones Proxy, para validar roles y accesos y Facade, con la finalidad de de simplicar operaciones.
- **Capa modelo:** en esta capa se define cómo se manejan los datos y la lógica del negocio. Los modelos trabajan con la base de datos MySQL e implementan patrones como Builder, Decorator y Composite, que ayudan a crear reservas, añadir servicios y organizar ubicaciones de manera flexible.
- **Base de datos MySQL:** Guarda toda la información del sistema: usuarios, canchas, reservas y ubicaciones, garantizando la persistencia y consistencia de los datos.

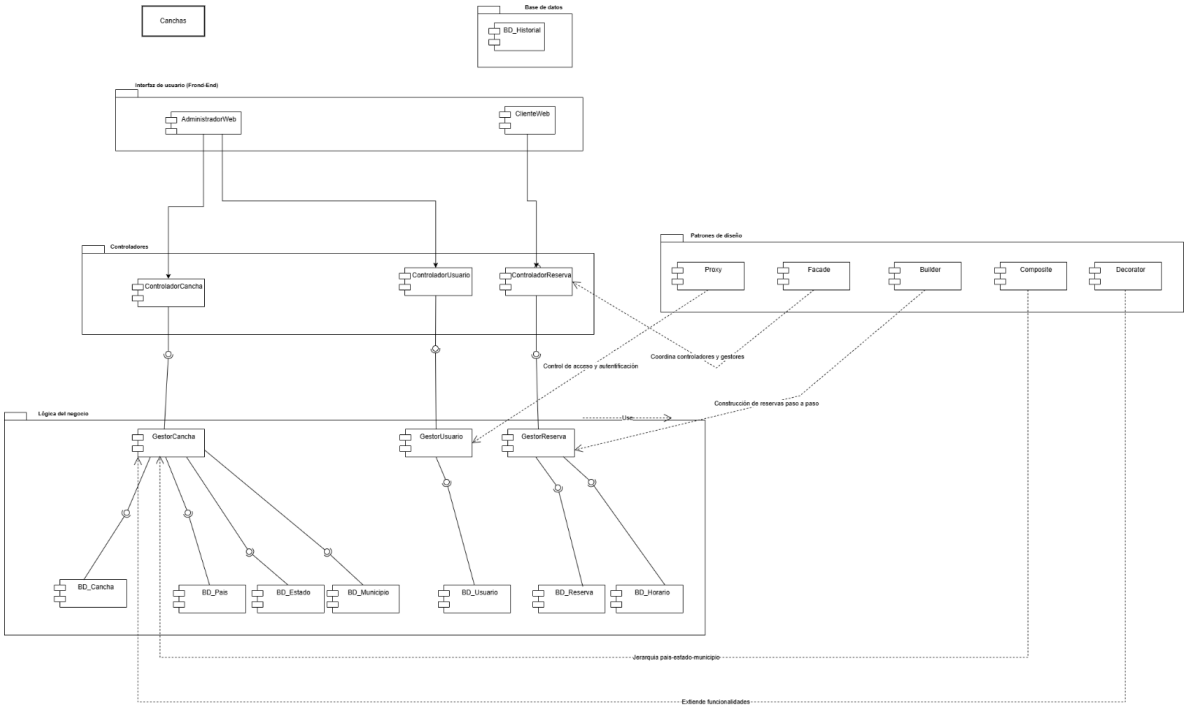


3.2. Diagramas UML

3.2.1. Diagrama de componentes

El diagrama de componentes con patrones de diseño representa una arquitectura robusta, modular y extensible, alineada con los principios SOLID y las buenas prácticas de ingeniería de software.

- Los controladores gestionan la interacción del usuario con el sistema.
- Los gestores concentran la lógica de negocio y aplican los patrones según su
- responsabilidad.
- Los modelos se encargan de la persistencia de datos.
- Los patrones de diseño fortalecen la estructura interna, promoviendo la reutilización, el bajo acoplamiento y la alta cohesión entre módulos.



3.2.2. Diagrama de clases con patrones de diseño aplicados

Durante el desarrollo del sistema, se procuró que las relaciones entre clases reflejaran las dependencias entre las entidades principales: Usuario, Reserva, Cancha y Ubicación. Esto garantiza que el modelo sea coherente con el funcionamiento del sistema en la vida real. Además, los patrones de diseño que no representan entidades concretas se integraron como dependencias o notas, respetando las buenas prácticas de modelado UML.

Patrón Composite – Ubicaciones

Para representar la jerarquía geográfica entre País, Estado, Municipio y Cancha, se aplicó el patrón Composite. Este enfoque permite tratar todas las ubicaciones de forma uniforme, lo que simplifica tanto las consultas como el mantenimiento de la estructura territorial del sistema.

Patrón Decorator – Canchas

Las canchas pueden tener características adicionales como iluminación nocturna o promociones especiales. En lugar de modificar la clase base, se utilizó el patrón Decorator, lo que permite extender funcionalidades de forma flexible y sin alterar la estructura original. Esto facilita la evolución del sistema ante nuevos requerimientos.

Patrón Facade – Reservas

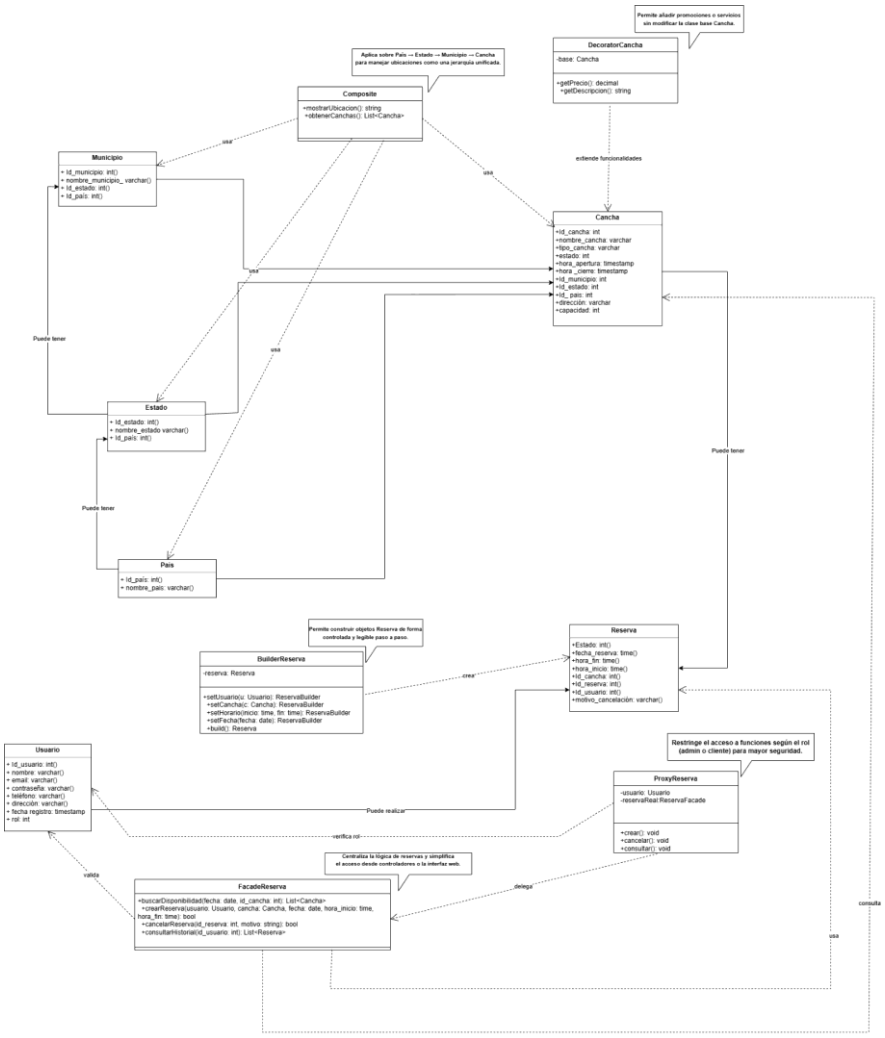
Para manejar la complejidad de las operaciones de reserva, se implementó este patrón que actúa como intermediario entre los controladores y los modelos. Esto centraliza la lógica, reduce el acoplamiento y mejora la organización del código, haciendo que el sistema sea más mantenible.

Patrón Builder – Creación de reservas

La creación de una reserva implica múltiples pasos y validaciones. El patrón Builder permite construir objetos Reserva de manera progresiva, asegurando que los datos estén completos y correctamente inicializados. Esto mejora la claridad del proceso y reduce errores.

Patrón Proxy – Control de permisos

Para proteger las funciones sensibles del sistema, se aplicó el patrón Proxy. Este patrón permite controlar el acceso a ciertas operaciones según el tipo de usuario (por ejemplo, administrador o cliente), reforzando la seguridad y evitando acciones no autorizadas



3.2.3. Diagrama de despliegue

Este diagrama muestra como los diferentes componentes de la aplicación web se despliegan dentro del entorno local de ejecución. La aplicación opera en un servidos XAMPP, que integra apache, PHP y MySQL, simulando el funcionamiento completo del sistema.

En la parte superior se representa el navegador web, desde el cual los usuarios acceden al sistema a través de localhost. El navegador envía solicitudes HTTP al servidor Apache, donde se aloja la aplicación web desarrollada en PHP.

Dentro de la aplicación, se distinguen varias capas:

- La capa de presentación contiene las vistas públicas, del cliente y del administrador, junto con los recursos estáticos (HTML, CSS y JavaScript) que conforman la interfaz visual.
- La capa de controlador coordina las operaciones entre las vistas y los modelos, aplicando los patrones Proxy y Facade para controlar permisos y simplificar procesos.
- La capa de modelo gestiona la lógica del negocio y el acceso a los datos, empleando los patrones Builder, Decorator y Composite para construir reservas, añadir servicios y manejar la jerarquía de ubicaciones.

• La configuración del sistema define parámetros de conexión y variables globales utilizadas por todos los módulos.

Finalmente, el servidor MySQL almacena la información principal del sistema — usuarios, canchas, reservas y ubicaciones— garantizando la persistencia de los datos y la comunicación con los modelos a través de consultas SQL.

Justificación técnica de las decisiones arquitectónicas y aplicación de patrones.

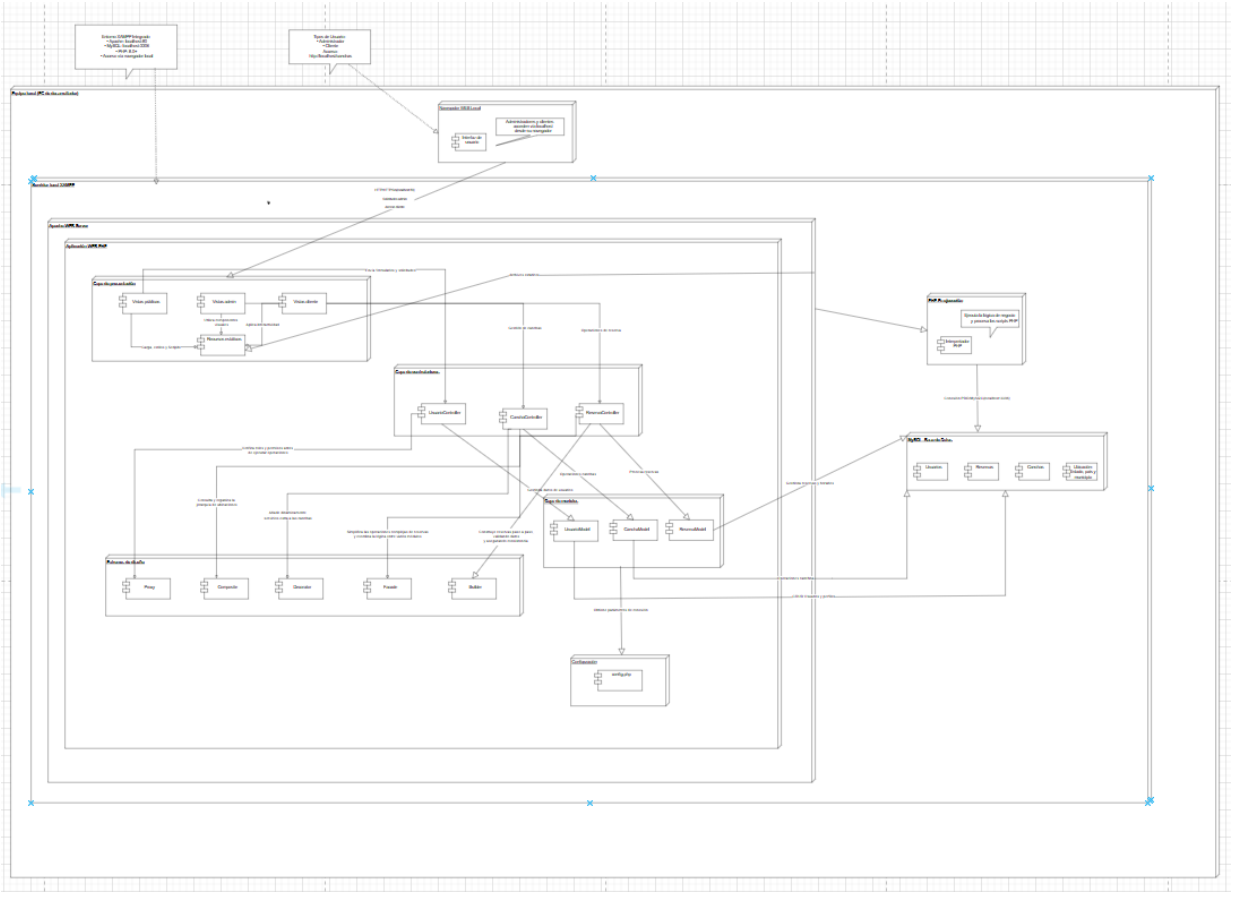
El sistema de reservas de canchas se pensó siguiendo el enfoque MVC (Modelo-vista-Controlador), utilizando XAMPP, que integra Apache, PHP y MySQL. Esta arquitectura permite dividir el sistema en capas bien definidas: una para la lógica de negocio, otra para la interfaz de usuario y otra para el acceso a datos. Gracias a esta separación, el sistema se vuelve más fácil de mantener, escalar y adaptar, respetando además los principios SOLID, fundamentales para un diseño limpio y robusto.

Para reforzar el modularidad y facilitar la reutilización del código, se incorporaron varios patrones de diseño que aportan soluciones elegantes y probadas:

- **Facade**: se implementó para simplificar la comunicación entre los controladores y los gestores del sistema, agrupando las operaciones complejas en una única interfaz clara.
- **Proxy**: se encargó de validar los roles de usuario y controlar el acceso a ciertas
- funcionalidades, como las operaciones administrativas, fortaleciendo la seguridad.

- **Builder**: se utilizó para construir reservas de manera progresiva, lo que permite mayor flexibilidad y asegura que los datos estén completos y bien estructurados.
- **Composite**: se aplicó para organizar las ubicaciones de forma jerárquica (país, estado, municipio, cancha), facilitando la navegación y el mantenimiento de esta estructura.
- **Decorator**: se empleó para añadir características adicionales a las canchas, sin modificar su clase base, lo que permite extender el sistema sin modificar su diseño original.

Estas decisiones técnicas no solo mejoran la organización interna del sistema, sino que también lo hacen más seguro y adaptable para crecer en el futuro sin perder coherencia ni calidad.



4.0.Selección del estilo arquitectónico principal para el proyecto integrador

Luego de evaluar cuidadosamente los requerimientos del sistema y el entorno tecnológico, se optó por una arquitectura por capas, implementada a través del patrón Modelo–Vista–Controlador (MVC).

4.1. Justificación

4.1.1. Compatibilidad con las necesidades del sistema.

La arquitectura por capas (Modelo–Vista–Controlador) es completamente compatible con las necesidades de la aplicación web de reservas de canchas, ya que se adapta al entorno de desarrollo definido (PHP, Apache y MySQL en XAMPP) y responde de manera estructurada a los requerimientos funcionales y no funcionales establecidos.

Este estilo organiza el sistema en tres capas principales:

- **Presentación (Vistas y recursos estáticos):** es la interfaz web que permite a los usuarios, ya sean clientes o administradores, interactuar con el sistema a través de formularios y peticiones AJAX.
- **Lógica de negocio (Controladores y patrones de diseño):** se encarga de manejar las operaciones de reserva, validación, autenticación y gestión de canchas, manteniendo separadas las reglas de negocio del diseño visual.
- **Persistencia (Modelos y Base de Datos MySQL): se** ocupa de la comunicación con la base de datos, garantizando la integridad y consistencia de los datos mediante transacciones y consultas parametrizadas.

Esta separación permitirá que el sistema sea más ordenado, mantenible y seguro. Además, el patrón MVC se adapta de forma nativa al entorno PHP, por lo que resulta totalmente compatible con las tecnologías seleccionada para el desarrollo.

4.1.2. Ventajas y limitaciones del diseño seleccionado.

Ventajas:

- Separa la presentación, la lógica de negocio y el acceso a datos, manteniendo el sistema ordenado.

- Facilita la mantenibilidad y permite agregar nuevas funciones sin afectar el resto del código.
- Es compatible de forma nativa con PHP y el entorno XAMPP.
- Permite reutilizar componentes y simplifica las pruebas y depuración.
- Favorece el trabajo en equipo al permitir desarrollar cada capa de manera independiente.

Limitaciones:

- Tiene una escalabilidad limitada al ser una aplicación monolítica.
- Depende de un único servidor, por lo que un fallo afecta a todo el sistema.
- Requiere una estructura más compleja y organizada desde el inicio.
- Puede presentarse mezcla de responsabilidades si no se sigue correctamente la separación entre capas.

4.1.3. Impacto en los atributos de calidad del sistema.

La elección de una arquitectura por capas basada en el patrón Modelo–Vista–Controlador (MVC) no solo responde a criterios técnicos, sino que también tiene un impacto directo y positivo en varios atributos clave de calidad del sistema:

- **Escalabilidad**: El sistema estructurado con capas claras permite que agregar nuevas funciones resulte más facil, sin necesidad de modificar nada interno. Un ejemplo podría ser: si más adelante se decide agregar módulos para torneos, desarrollar informes personalizados o integrar métodos de pago, estos se incorporarían como funcionalidades independientes. Gracias a esto, el sistema se expande fluidamente y de manera prolongada.
- **Mantenibilidad**: Separar la lógica de negocio (modelos), la gestión de eventos (controladores) y lo que se presenta (vistas) simplifica el código, facilitando su mantenimiento. Así, los programadores tienen mayor facilidad para identificar errores, realizar modificaciones o añadir mejoras sin afectar todo el sistema.
- **Seguridad**: Implementar un Proxy para regular el acceso de los usuarios es principal para la seguridad. De esta manera, se garantiza que cada acción dentro del sistema esté controlada, permitiendo a los usuarios acceder solamente a la información pertinente.

- **Usabilidad**: El diseño MVC, con AJAX y vistas dinámicas, proporciona una experiencia de usuario más fluida. Las interacciones se dan sin tener que recargar por completo la página, mejora la velocidad y agilidad. Resulta útil en reservas, donde la velocidad y la claridad en la interfaz importan mucho.
- **Rendimiento**: Con una arquitectura local optimizada, el sistema responde rápido, a veces hasta menos de 2 segundos en consultas normales. Esto da una experiencia más eficiente al usuario final, sobre todo con alta demanda o grandes datos.

4.1.4. Integración con la implementación, despliegue y pruebas.

Inclinarse por una arquitectura por capas siguiendo el enfoque Modelo–Vista–Controlador (MVC) no es solo una elección técnica: es una decisión que influye directamente en cada etapa del desarrollo del sistema de reservas.

Implementación

Durante la fase de implementación, este estilo permite organizar el código de forma lógica y ordenada. Cada capa tiene una responsabilidad bien definida:

- **Presentación**: la interfaz que ve y usa el usuario.
- **Negocio**: los controladores que gestionan la lógica funcional.
- **Datos**: los modelos que se conectan con la base de datos.

Esta separación no solo mejora la claridad del proyecto, sino que también facilita el trabajo en equipo. Varios desarrolladores pueden trabajar en paralelo sobre distintas capas sin interferir entre sí, lo que promueve la reutilización del código y permite escalar el sistema con facilidad.

Despliegue

La arquitectura se adapta perfectamente al entorno local XAMPP, que integra Apache, PHP y MySQL en un solo servidor. Esto permite alojar tanto la aplicación web como la base de datos de forma eficiente. Además, el uso de capas facilita el control de versiones y la portabilidad del sistema hacia otros entornos, como servidores en la nube o entornos de producción. Al tener módulos independientes, las actualizaciones se pueden

aplicar de forma aislada, reduciendo riesgos y evitando que una modificación afecte el funcionamiento general.

Pruebas

La escalabilidad del diseño también tiene ventajas claras en la etapa de pruebas. Se pueden evaluar componentes individuales como modelos o controladores y también probar el sistema de forma integrada. Esto permite detectar errores de forma temprana y asegurar la estabilidad antes del despliegue final. Además, facilita la ejecución de pruebas no funcionales, como rendimiento, seguridad y trazabilidad, ya que cada capa está aislada y bien definida.

Conclusión

El desarrollo del sistema de reservas permitió consolidar una arquitectura clara, mantenible y respaldada por buenas prácticas de diseño. La separación en capas a través del patrón MVC facilitó la organización del código y la distribución adecuada de responsabilidades, favoreciendo la extensibilidad y el control del flujo interno de la aplicación.

La incorporación de patrones de diseño como Builder, Facade, Proxy, Decorator y Composite aportó soluciones específicas a problemas recurrentes dentro del dominio. Builder ordenó la construcción de reservas, Facade simplificó la interacción con procesos complejos, Proxy permitió controlar permisos y accesos, mientras que Decorator y Composite extendieron la funcionalidad de canchas y ubicaciones sin alterar la estructura base del sistema. La integración de estos patrones fortaleció el modularidad del proyecto y permitió mantener un código flexible, preparado para cambios futuros.

En conjunto, las decisiones arquitectónicas adoptadas resultaron en un sistema robusto y escalable, con capacidad de evolucionar sin comprometer la estabilidad ni aumentar innecesariamente la complejidad. El proyecto demuestra cómo el uso adecuado de patrones y principios de diseño contribuye a construir soluciones de software más ordenadas, sostenibles y coherentes con las necesidades del negocio.

Links de los diagramas UML elaborados:

Modelo C4, Nivel 1, 2 y 3

Diagrama de despliegue

Diagrama de clases con patrones de diseño

Diagrama de componentes

Bibliografía.

Bass, L., Clements, P., & Kazman, R. (2012). *Software architecture in practice* (3rd ed.). Addison-Wesley.

Evans, E. (2003). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley.

Fowler, M. (2002). *Patterns of enterprise application architecture*. Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

García, J. (2019). *Arquitectura de software y diseño de sistemas de información*. Universidad Nacional de Colombia.

IEEE Computer Society. (2022). *IEEE Std 42010-2022: Systems and software engineering—Architecture description*. IEEE.

International Organization for Standardization. (2011). *ISO/IEC 25010: Systems and software engineering—Systems and software quality requirements and evaluation (SQuaRE)—System and software quality models*. ISO.

Pressman, R. S., & Maxim, B. R. (2020). *Ingeniería del software: Un enfoque práctico* (9.ª ed.). McGraw-Hill Education.

Sommerville, I. (2016). *Ingeniería del software* (10.ª ed.). Pearson Educación.

Universidad Nacional de Colombia. (2020). *Patrones de diseño y arquitectura de software*. Facultad de Ingeniería.

Universidad de Antioquia. (2022). *Modelado y diseño de software con UML y patrones*. Escuela de Ingeniería de Sistemas.

Ministerio de Tecnologías de la Información y las Comunicaciones. (2021). *Guía de arquitectura empresarial para entidades públicas*. MinTIC.

Ministerio de Tecnologías de la Información y las Comunicaciones. (2022). *Guía para el diseño de arquitecturas de software en sistemas públicos digitales*. MinTIC.

Fuentes digitales

Álvarez, M. Á. (2023, septiembre 20). *¿Qué es MVC?* DesarrolloWeb.com. https://desarrolloweb.com/articulos/que-es-mvc.html

Microsoft. (s. f.). *Design patterns*. Microsoft Docs. https://learn.microsoft.com/

Refactoring Guru. (s. f.). *Catálogo de patrones de diseño*. https://refactoring.guru/es/design-patterns

RJ Code Advance. (s. f.). *Patrones de software y arquitectura en capas: Análisis completo y ejemplo DDD (parte 5)*. https://rjcodeadvance.com/patrones-de-software-arquitectura-en-capas-analisis-completo-ejemplo-ddd-parte-5/

Uso de herramientas de IA

OpenAI. (2025). *Asistencia en el desarrollo y documentación del sistema de reservas de canchas deportivas mediante ChatGPT (versión GPT-5)*. Herramienta utilizada para redacción técnica, análisis arquitectónico y generación de diagramas UML.