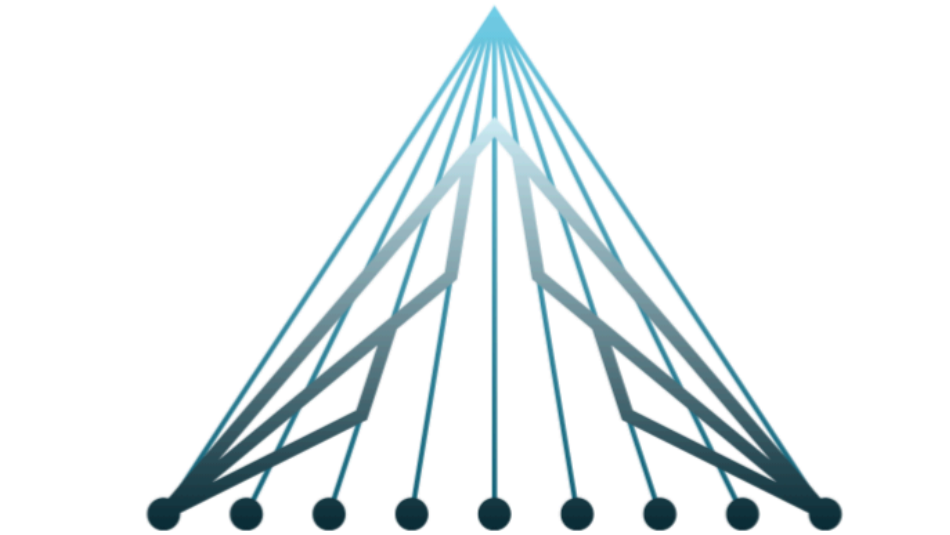


Unidad de Trabajo 5
Diseño
Modelado

Autor: Ricardo Ruiz Anaya

ÍNDICE

Autor: Ricardo Ruiz Anaya

1. Introducción al Modelado de Software

1.1 Definición y Objetivo del Modelado

El **modelado de software** es un proceso clave en el desarrollo de sistemas que implica la creación de representaciones abstractas y visuales del sistema a desarrollar. Su objetivo principal es simplificar la complejidad de un sistema al descomponerlo en componentes comprensibles, facilitando así la comprensión, el análisis y la comunicación entre todos los involucrados en el proyecto.
Es la creación de modelos visuales que representan de manera estructurada y sistemática los distintos aspectos de un sistema de software, desde sus componentes estructurales hasta sus procesos y comportamientos.

Objetivos del modelado:

- **Visualizar el sistema** antes de su construcción.
- **Comprender y analizar** los requisitos del sistema.
- **Diseñar soluciones eficientes** alineadas con los objetivos del proyecto.
- **Documentar** los procesos y componentes para facilitar la comunicación entre equipos.
- **Reducir errores** anticipando posibles problemas antes de la implementación.

Ejemplo práctico:
En el desarrollo de una aplicación bancaria, el modelado permite representar cómo interactúan los usuarios con el sistema (diagrama de casos de uso), cómo están organizadas las bases de datos (diagrama de clases) y cómo fluyen los procesos de transacción (diagrama de actividades).

1.2 Importancia del Modelado en el Ciclo de Vida del Software

El modelado desempeña un papel fundamental en cada fase del **Ciclo de Vida del Desarrollo de Software (SDLC)**, ya que proporciona una base sólida para el diseño, implementación y mantenimiento del sistema. Su correcta aplicación tiene impactos positivos en la calidad del software, la eficiencia del desarrollo y la satisfacción del cliente.

Importancia en cada fase del SDLC:

- **Fase de análisis:** Permite interpretar los requisitos del cliente a través de modelos que describen el comportamiento esperado del sistema.
- **Fase de diseño:** Facilita la estructuración del sistema mediante diagramas que detallan cómo se organizarán los componentes.
- **Fase de implementación:** Los modelos guían a los desarrolladores en la codificación al representar visualmente la arquitectura y los flujos de trabajo.
- **Fase de pruebas:** Ayuda a identificar escenarios críticos para validar el funcionamiento del sistema.
- **Fase de mantenimiento:** Sirve como documentación para futuras actualizaciones y mejoras.

Autor: Ricardo Ruiz Anaya

Beneficios del modelado:

- **Comunicación clara:** Mejora la comunicación entre desarrolladores, diseñadores, testers y clientes.
- **Reducción de riesgos:** Permite identificar errores o inconsistencias antes de escribir código.
- **Optimización de recursos:** Ahorra tiempo y dinero al evitar retrabajos por malentendidos.
- **Flexibilidad y adaptabilidad:** Facilita la incorporación de cambios en el diseño.

Ejemplo práctico:
En un sistema de gestión de inventarios, un **diagrama de casos de uso** identifica las acciones del usuario (añadir productos, generar reportes), mientras que un **diagrama de clases** define la relación entre los objetos del sistema (producto, proveedor, categoría).

1.3 Relación entre Modelado y Diseño de Soluciones

El **modelado** es una extensión directa del proceso de diseño. Mientras que el diseño implica tomar decisiones sobre cómo resolver un problema, el modelado proporciona las herramientas y técnicas para plasmar esas decisiones de forma visual y comprensible.

Relación directa:

- **El modelado es el medio para diseñar soluciones.** Permite estructurar ideas, probar alternativas y validar conceptos antes de implementarlos.
- **Transforma los requisitos en soluciones técnicas.** Los modelos convierten los requisitos recopilados en estructuras organizadas listas para ser desarrolladas.
- **Facilita la toma de decisiones.** Al visualizar los componentes del sistema, se pueden detectar áreas de mejora o ajustes necesarios en el diseño.

Proceso de integración:

1. **Identificación de requisitos:** Los analistas definen qué necesita el cliente.
2. **Creación de modelos:** Se representan visualmente los requisitos mediante diagramas.
3. **Evaluación del diseño:** Los modelos permiten validar si el diseño cumple con los objetivos.
4. **Implementación:** Los desarrolladores utilizan los modelos como guía para escribir el código.
5. **Validación:** Los modelos ayudan a diseñar las pruebas para garantizar que el sistema cumple su propósito.

Ventajas de integrar modelado y diseño:

- **Claridad en los objetivos del proyecto.**
- **Detección temprana de problemas.**
- **Adaptación rápida a cambios.**
- **Mejora continua del sistema.**

2. Principios del Modelado de Sistemas

El modelado de sistemas es un proceso que requiere de principios claros para garantizar la creación de modelos eficientes, coherentes y alineados con los objetivos del proyecto. Estos principios guían la forma en que se representan los sistemas, asegurando que los modelos sean útiles para la toma de decisiones, la comunicación y el desarrollo de soluciones efectivas.

2.1 Abstracción y Simplificación de Sistemas Complejos

Uno de los principios fundamentales del modelado de sistemas es la **abstracción**, que consiste en representar sólo los aspectos esenciales de un sistema, omitiendo detalles innecesarios. Esto permite simplificar sistemas complejos, facilitando su análisis, diseño e implementación.

Características de la abstracción:

- **Enfoque en lo esencial:** Se representan solo los elementos clave del sistema.
- **Eliminación de detalles irrelevantes:** Se omiten aspectos que no aportan valor al análisis o al diseño.
- **Facilita la comprensión:** Reduce la complejidad, permitiendo que todas las partes interesadas comprendan el sistema.

Tipos de abstracción:

- **Abstracción funcional:** Se centra en las funciones y procesos del sistema.
- **Abstracción de datos:** Describe cómo se organizan y relacionan los datos.
- **Abstracción de control:** Representa el flujo de control y la lógica del sistema.

Ejemplo práctico:
En un sistema de gestión de bibliotecas, la abstracción funcional incluiría procesos como **"Préstamo de libros"** y **"Devolución de libros"**, omitiendo detalles técnicos como el formato de almacenamiento de los datos.

2.2 Coherencia y Consistencia en el Diseño

La **coherencia** y la **consistencia** son principios clave para asegurar que los modelos sean comprensibles, integrados y libres de contradicciones. Ambos principios garantizan que los modelos reflejen fielmente los requisitos del sistema y que las distintas representaciones del sistema estén alineadas entre sí.

Coherencia:

- Se refiere a que los modelos mantengan una relación lógica entre sus componentes.
- Asegura que los distintos diagramas (estructurales, de comportamiento, de interacción) estén alineados.
- Evita contradicciones en la representación del sistema.

Autor: Ricardo Ruiz Anaya

Consistencia:

- Garantiza que los elementos del modelo estén correctamente definidos y aplicados.
- Se refiere a la uniformidad en el uso de nombres, símbolos y relaciones.
- Mantiene la integridad de la información en todos los modelos.

Ejemplo práctico:
Si en un **diagrama de clases** existe una clase llamada **"Usuario"**, esta misma entidad debe aparecer con el mismo nombre y relación en el **diagrama de casos de uso** y en el **diagrama de secuencia**, manteniendo coherencia y consistencia en todo el diseño.

2.3 Trazabilidad desde los Requisitos al Diseño

La **trazabilidad** es el principio que permite seguir el rastro de cada requisito desde su origen hasta su implementación y validación. Asegura que todas las decisiones de diseño estén justificadas y que cada funcionalidad implementada esté alineada con los objetivos del proyecto.

Características de la trazabilidad:

- **Bidireccionalidad:** Permite rastrear un requisito desde su origen hasta el código fuente y viceversa.
- **Documentación clara:** Cada requisito debe estar vinculado a los modelos, pruebas y resultados correspondientes.
- **Control de cambios:** Facilita la gestión de cambios, evaluando el impacto de modificar o eliminar requisitos.

Ventajas de la trazabilidad:

- **Evita requisitos no implementados o erróneos.**
- **Permite verificar el cumplimiento de los objetivos.**
- **Facilita la validación y mantenimiento del sistema.**

Ejemplo práctico:
En un proyecto de comercio electrónico, el requisito **"Permitir pagos con tarjeta de crédito"** debe estar reflejado en el **diagrama de casos de uso** (acción del usuario), en el **diagrama de clases** (clase **Pago**) y en los casos de prueba que validan la funcionalidad.

3. Tipos de Modelos en el Desarrollo de Software

El desarrollo de software implica la creación de diferentes modelos que representan de forma visual y estructurada de diversos aspectos del sistema. Estos modelos permiten comprender, analizar y diseñar soluciones efectivas, facilitando la comunicación entre todos los involucrados en el proyecto.

Los modelos se pueden clasificar en tres grandes categorías:

1. **Modelos Estructurales**
2. **Modelos de Comportamiento**
3. **Modelos de Implementación**

3.1 Modelos Estructurales

Los **modelos estructurales** representan la organización estática del sistema. Se enfocan en los elementos del sistema y cómo estos se relacionan entre sí, describiendo la arquitectura y los componentes.

Diagramas de Clases

- Representan la estructura del sistema a través de clases, atributos, métodos y relaciones.
- Permiten identificar los objetos que interactúan en el sistema y cómo se relacionan.
- Se utilizan para modelar la lógica de negocio y la base de datos.

Elementos clave:

- **Clases:** Definen los objetos del sistema.
- **Atributos:** Características de las clases.
- **Métodos:** Funcionalidades de las clases.
- **Relaciones:** Asociación, herencia, composición y agregación.

Ejemplo:
En una tienda online:

- **Clase:** Producto
- **Atributos:** nombre, precio, stock
- **Relaciones:** Un **Producto** puede pertenecer a una **Categoría** y estar en múltiples **Pedidos**.

Diagramas de Componentes

- Representan la organización física de los componentes de software.
- Describen cómo los módulos de software están interconectados y cómo se distribuyen.
- Se utilizan para modelar la arquitectura de sistemas complejos.

Autor: Ricardo Ruiz Anaya

Elementos clave:

- **Componentes:** Módulos o subsistemas.
- **Interfaces:** Puntos de interacción entre componentes.
- **Dependencias:** Conexiones o relaciones entre componentes.

Ejemplo:
En una aplicación web:

- **Componente de Autenticación** que se conecta con un **Componente de Gestión de Usuarios**.
- **Interfaz API REST** que comunica la aplicación web con el servidor.

3.2 Modelos de Comportamiento

Los **modelos de comportamiento** representan cómo interactúan los elementos del sistema y cómo se desarrollan los procesos dentro de este. Son esenciales para comprender la lógica de funcionamiento y el flujo de acciones.

Diagramas de Actividades

- Representan flujos de trabajo o procesos de negocio.
- Se utilizan para modelar algoritmos, procesos o tareas.
- Permiten visualizar decisiones y acciones secuenciales o paralelas.

Elementos clave:

- **Actividades:** Tareas o acciones.
- **Decisiones:** Puntos donde el flujo puede bifurcarse.
- **Flujos:** Líneas que conectan las actividades.

Ejemplo:
Proceso de compra online:

1. Seleccionar productos.
2. Confirmar el carrito.
3. Realizar pago.
4. Confirmar pedido.

Diagramas de Secuencia

- Modelan la interacción entre objetos a lo largo del tiempo.
- Describen cómo los objetos se comunican mediante mensajes.
- Se utilizan para visualizar el flujo de ejecución de un proceso.

Elementos clave:

- **Objetos:** Participantes en la interacción.
- **Mensajes:** Comunicación entre objetos.
- **Líneas de vida:** Representan el tiempo de existencia de los objetos.

Autor: Ricardo Ruiz Anaya

Ejemplo:
Proceso de autenticación:

1. El usuario ingresa sus credenciales.
2. El sistema valida los datos.
3. Se concede acceso o se muestra un error.

3.3 Modelos de Implementación

Los **modelos de implementación** representan cómo se desplegará físicamente el sistema, cómo se distribuyen sus componentes y cómo interactúan los diferentes entornos.

Diagramas de Despliegue

- Representan la distribución física del software en el hardware.
- Muestran la relación entre nodos (servidores, dispositivos) y los componentes de software.

Elementos clave:

- **Nodos:** Dispositivos físicos o entornos virtuales.
- **Componentes:** Elementos de software asignados a los nodos.
- **Conexiones:** Comunicación entre nodos.

Ejemplo:
Despliegue de una aplicación web:

- **Servidor web** (Apache) que ejecuta la aplicación.
- **Servidor de base de datos** (MySQL) para almacenar la información.
- **Cliente** (navegador) que accede al sistema.

4. Herramientas de Modelado

El modelado de software requiere el uso de herramientas especializadas que permitan representar de manera visual, estructurada y detallada los distintos aspectos de un sistema. Estas herramientas facilitan la creación de diagramas, la documentación de procesos y la validación de diseños antes de su implementación. Elegir la herramienta adecuada depende de las necesidades del proyecto, la complejidad del sistema y el enfoque metodológico utilizado.

4.1 Herramientas UML (Unified Modeling Language)

El **Lenguaje Unificado de Modelado (UML)** es un estándar ampliamente utilizado en el desarrollo de software para visualizar, especificar, construir y documentar sistemas. UML proporciona un conjunto de diagramas que representan diferentes aspectos de un sistema, tanto estructurales como de comportamiento.

Principales Herramientas UML

1. **Enterprise Architect**
    - Completa y robusta para modelado UML y diseño de software.
    - Permite crear diagramas de clases, de casos de uso, de secuencia, de actividades, entre otros.
    - Soporta ingeniería inversa y generación de código.
2. **Visual Paradigm**
    - Ofrece modelado UML y herramientas de diseño de bases de datos.
    - Integra metodologías ágiles y DevOps.
    - Permite crear diagramas UML, BPMN y de arquitectura.
3. **StarUML**
    - Herramienta ligera y flexible para modelado UML.
    - Compatible con múltiples lenguajes de programación.
    - Ideal para diagramas UML y ER (Entidad-Relación).
4. **Lucidchart**
    - Basada en la nube, permite crear diagramas UML de forma colaborativa.
    - Integra diagramas de flujo, organigramas y mapas mentales.
    - Fácil de usar, ideal para equipos distribuidos.

Ventajas del uso de UML:

- **Estandarización:** UML es un lenguaje universalmente aceptado.
- **Versatilidad:** Cubre aspectos estructurales y de comportamiento.
- **Documentación clara:** Mejora la comunicación entre equipos técnicos y de negocio.

4.2 Herramientas de Prototipado Rápido

Las herramientas de **prototipado rápido** son fundamentales para diseñar interfaces de usuario (UI) y experiencias de usuario (UX) antes de la implementación. Permiten visualizar

cómo interactúan los usuarios con el sistema, identificar mejoras y validar ideas con los stakeholders.

Principales Herramientas de Prototipado

1. **Figma**
    - Plataforma basada en la nube para diseño colaborativo.
    - Permite crear interfaces interactivas y prototipos navegables.
    - Ideal para equipos que trabajan de forma remota.
2. **Adobe XD**
    - Herramienta especializada en diseño de interfaces y experiencias.
    - Integra prototipos interactivos y flujos de navegación.
    - Compatible con herramientas de Adobe Creative Cloud.
3. **Sketch**
    - Herramienta de diseño vectorial para macOS.
    - Ampliamente utilizada en diseño de interfaces web y móviles.
    - Integración con plugins para mejorar la productividad.
4. **InVision**
    - Plataforma para prototipado y validación de diseños.
    - Permite crear maquetas interactivas y flujos de navegación.
    - Ideal para pruebas de experiencia de usuario.

Ventajas del uso de herramientas de prototipado:

- **Validación temprana:** Permiten validar conceptos antes de invertir en desarrollo.
- **Interactividad:** Simulan la experiencia real del usuario.
- **Iteración rápida:** Facilitan ajustes y mejoras continuas.

Autor: Ricardo Ruiz Anaya

6. Integración del Modelado en el Proceso de Desarrollo

El **modelado de software** no es un proceso aislado, sino una parte esencial y continua dentro del ciclo de vida del desarrollo de software. Integrar correctamente los modelos en todas las fases del proyecto permite alinear el diseño técnico con los requisitos del negocio, mejorar la comunicación entre los equipos y reducir errores durante la implementación. Este proceso garantiza que cada decisión de diseño tenga un propósito claro y esté respaldada por una documentación visual coherente.

6.1 Trazabilidad entre Modelos y Código Fuente

Definición de Trazabilidad

La **trazabilidad** en el desarrollo de software implica la capacidad de seguir el rastro de cada requisito desde su definición hasta su implementación, validación y despliegue. Esto incluye conectar los modelos de diseño con el código fuente y los casos de prueba.

Importancia de la Trazabilidad

- **Alineación de objetivos:** Asegura que el software desarrollado cumple con los requisitos del cliente.
- **Gestión de cambios:** Permite evaluar cómo los cambios en los requisitos afectan al diseño y la implementación.
- **Mejora de la calidad:** Facilita la identificación de errores y omisiones al mantener una relación directa entre los modelos y el código.
- **Facilidad de mantenimiento:** Simplifica las actualizaciones y modificaciones al tener documentada la relación entre componentes.

Estrategias para Mantener la Trazabilidad

- **Análisis de impacto:** Antes de realizar cambios, analizar cómo afectan los requisitos al diseño y al código.
- **Documentación detallada:** Registrar todas las relaciones entre los requisitos, los modelos y el código.
- **Automatización de trazabilidad:** Utilizar herramientas como **Enterprise Architect** o **Jira** para vincular requisitos con modelos y tareas de desarrollo.

Ejemplo práctico:
En un sistema de reservas de vuelos, el requisito **“Permitir la selección de asientos”** se representa en el **diagrama de casos de uso**, se detalla en el **diagrama de clases** con la clase **Asiento** y se implementa en el código fuente con una clase concreta.

6.2 Impacto del Modelado en la Calidad del Software

El modelado de software tiene un impacto directo en la calidad del producto final, ya que permite planificar y prever posibles errores antes de que ocurran durante la implementación. Los modelos proporcionan una base sólida para garantizar la robustez, escalabilidad y eficiencia del sistema.

Aspectos Clave en la Mejora de la Calidad

- **Detección temprana de errores:** Los modelos permiten identificar problemas de diseño antes de que se traduzcan en errores de código.
- **Optimización de procesos:** Los diagramas de flujo y actividades ayudan a mejorar procesos internos.
- **Estandarización:** El uso de UML y BPMN asegura que el diseño siga estándares reconocidos, lo que facilita la colaboración.
- **Validación continua:** Los modelos se pueden revisar con los stakeholders para verificar que el diseño cumpla con los requisitos.

Relación entre Modelado y Calidad

- **Modelos precisos → Mejor diseño → Menos errores de implementación.**
- **Modelos claros → Mejor comunicación → Equipos alineados.**
- **Documentación visual → Facilita validación y mantenimiento.**

Ejemplo práctico:
Un **diagrama de secuencia** bien elaborado puede ayudar a identificar problemas de sincronización en un sistema distribuido, evitando fallos costosos en producción.

6.3 Adaptación del Modelado a Metodologías Ágiles y Tradicionales

El modelado se adapta a distintas metodologías de desarrollo, ya sean ágiles (como **Scrum** o **Kanban**) o tradicionales (como el **modelo en cascada**). Su integración depende del enfoque y la dinámica de trabajo del equipo.

Modelado en Metodologías Tradicionales

- **Documentación completa y detallada:** Se crean modelos exhaustivos antes de la implementación.
- **Secuencia de fases:** El modelado se realiza en fases específicas (análisis y diseño).
- **Mayor formalidad:** Uso de modelos más complejos y documentación extensa.

Ventaja: Proporciona una planificación detallada y controlada.
Desventaja: Menor flexibilidad para adaptarse a cambios.

Ejemplo:
En el **modelo en cascada**, se elaboran **diagramas de clases**, **diagramas de componentes** y **diagramas de despliegue** completos antes de iniciar la codificación.

Modelado en Metodologías Ágiles

- **Modelos ligeros y flexibles:** Se utilizan modelos simples que evolucionan junto con el desarrollo.
- **Iteraciones rápidas:** Los modelos se actualizan conforme cambian los requisitos.
- **Colaboración continua:** Los modelos se revisan con el equipo y los clientes durante cada sprint.

Ventaja: Mayor adaptabilidad a cambios de requisitos.
Desventaja: La documentación puede ser menos detallada.

Ejemplo:
En **Scrum**, se puede crear un **diagrama de flujo** para representar un proceso específico y actualizarlo en función del feedback del cliente en cada sprint.

Herramientas para Integrar Modelado en Diferentes Metodologías

- **Jira + Draw.io:** Para metodologías ágiles, integrando tareas con modelos visuales.
- **Enterprise Architect:** Para metodologías tradicionales, con documentación formal y detallada.
- **Lucidchart:** Para colaboración visual en tiempo real, adaptable a cualquier metodología.

Ejercicio Práctico: Diseño de un Diagrama de Flujo de Pantallas para un Sistema de Gestión de Reservas en un Gimnasio

🎯 Objetivo

Diseñar un **Diagrama de Flujo de Pantallas** que represente de manera clara e intuitiva la navegación del usuario a través del sistema de gestión de reservas de un gimnasio. Este diagrama debe reflejar todas las pantallas necesarias y las transiciones entre ellas, cubriendo tanto la experiencia del usuario como la gestión administrativa.

📖 Situación del Problema

Un gimnasio busca modernizar su proceso de gestión de clases mediante una aplicación web. Los clientes deben poder:

- Registrarse e iniciar sesión.
- Consultar las clases disponibles.
- Reservar o cancelar clases.
- Recibir notificaciones de confirmación y recordatorio.

Por otro lado, los administradores necesitan:

- Gestionar los horarios de clases.
- Asignar instructores.
- Controlar la capacidad de las clases.

📌 Requisitos Funcionales

1. **Registro e inicio de sesión de usuarios.**
2. **Consulta de clases y disponibilidad.**
3. **Reserva y cancelación de clases.**
4. **Gestión de horarios e instructores (para administradores).**
5. **Envío de notificaciones de confirmación.**

🛠️ Instrucciones para el Desarrollo del Diagrama

1. Identificación de Pantallas

Define las pantallas que formarán parte del flujo de navegación. Se recomienda incluir:

Autor: Ricardo Ruiz Anaya

- **Pantalla de Inicio**
- **Pantalla de Registro/Iniciar Sesión**
- **Menú Principal**
- **Pantalla de Consulta de Clases**
- **Pantalla de Detalle de Clase**
- **Pantalla de Confirmación de Reserva**
- **Pantalla de Notificaciones**
- **Pantalla de Gestión de Clases** (acceso exclusivo para administradores)
- **Pantalla de Perfil de Usuario**
- **Pantalla de Error de Autenticación**

2. Diseño del Flujo de Navegación

Establece cómo se conectan estas pantallas entre sí. Considera todas las posibles rutas que puede seguir el usuario o el administrador.

Flujo sugerido:

1. **Pantalla de Inicio** → Botón **"Iniciar Sesión"** o **"Registrarse"**.
2. **Pantalla de Registro/Iniciar Sesión** →
    - Si los datos son correctos → **Menú Principal**.
    - Si los datos son incorrectos → **Pantalla de Error de Autenticación**.
3. **Menú Principal** → Opciones:
    - **Consultar Clases** → **Pantalla de Consulta de Clases**.
    - **Perfil de Usuario** → **Pantalla de Perfil**.
    - **Gestión de Clases** → Solo visible para **Administradores**.
4. **Consulta de Clases** → Seleccionar clase → **Detalle de Clase**.
5. **Detalle de Clase** → Botón **"Reservar"** → **Confirmación de Reserva**.
6. **Confirmación de Reserva** → Enviar notificación → **Pantalla de Notificaciones**.

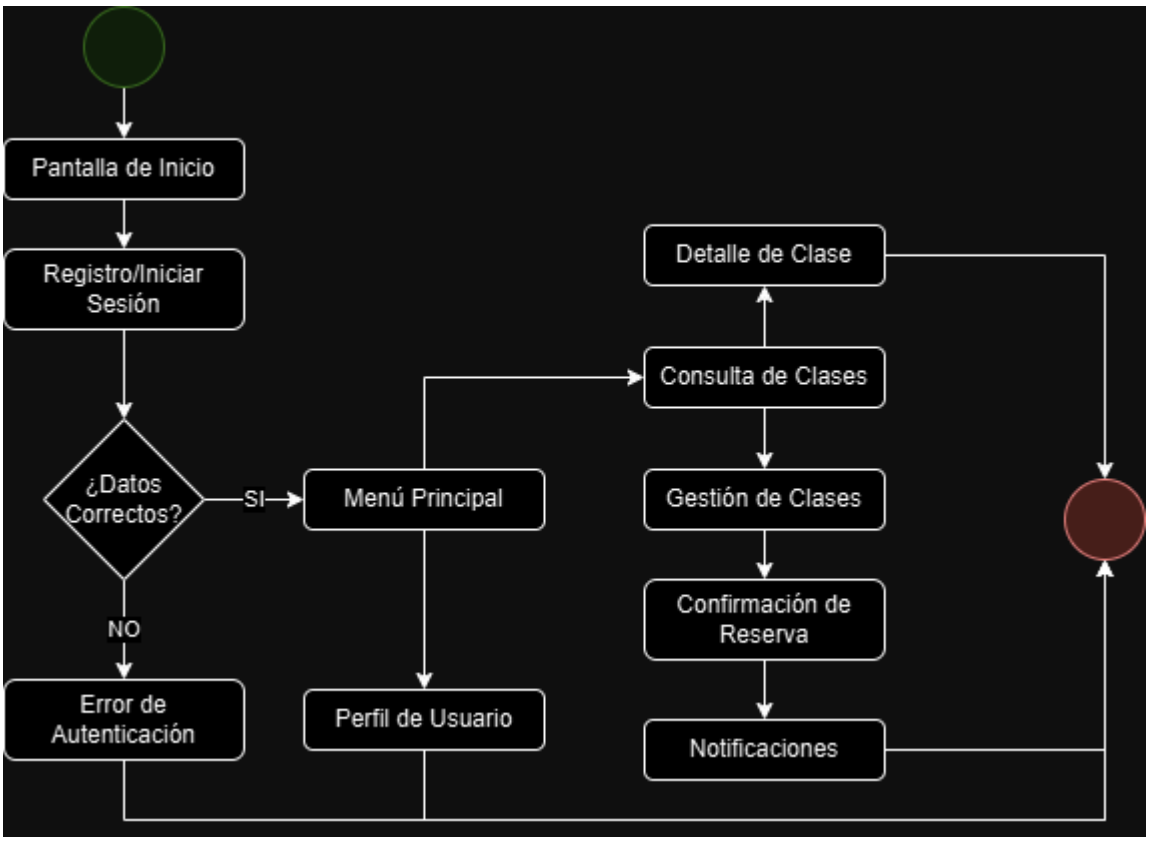
3. Elaboración del Diagrama de Flujo de Pantallas

Indicaciones:

- Representa cada **pantalla** con un **rectángulo**.
- Utiliza **flechas** para conectar las pantallas y mostrar el flujo.
- Usa **rombos** para decisiones (ejemplo: validación de usuario).
- Asegúrate de que el flujo sea **claro** y **lógico**.

Ejemplo Visual Simplificado:

Autor: Ricardo Ruiz Anaya



4. Herramientas Recomendadas

Para crear el diagrama, utiliza la siguiente herramienta **Draw.io** (herramienta gratuita y fácil de usar).

✅ Resultado Esperado

El **Diagrama de Flujo de Pantallas** debe incluir:

- **Todas las pantallas clave** del sistema.
- **Transiciones claras** y **correctamente etiquetadas**.
- Decisiones lógicas representadas con **rombos**.
- Un flujo de navegación **intuitivo** y **coherente**.

El diseño debe reflejar claramente cómo interactúan los usuarios y administradores con el sistema de gestión de reservas, cubriendo todo el proceso desde el acceso hasta la confirmación de reservas.

Autor: Ricardo Ruiz Anaya