# PROYECTO LABORATORIO INGENIERIA DE SOFTWARE: HOMYGO

REALIZADO POR:

MIGUEL AHIJÓN HORMIGOS

ANAÍS CHIQUITO PATIÑO

DAVID MARCOS VALHONDO

CLAUDIA MARTÍN MONTERO

# ÍNDICE

# Introducción

El presente proyecto tiene como objetivo el desarrollo de una aplicación de gestión de inmuebles y reservas, denominada **Homygo**. La finalidad principal es crear un sistema funcional que permita registrar y autenticar usuarios, gestionar inmuebles, realizar reservas, llevar un control de pagos y disponibilidades, y ofrecer una interfaz de usuario accesible a través de la web (sistema parecido a Airbnb).

El proyecto se concibe como un **sistema multicapa**, integrando la capa de presentación, la lógica de negocio y la persistencia de datos, para garantizar que cada módulo del sistema cumpla con sus responsabilidades específicas y que la información se gestione de manera coherente y segura. Además, se busca que la aplicación sea **escalable y mantenible**, de manera que se puedan incorporar nuevas funcionalidades en el futuro sin comprometer la estabilidad del sistema.

La elección de tecnologías modernas y robustas, como **Java, Spring Boot, Thymeleaf y Derby embebido**, ha permitido desarrollar un prototipo funcional que integra la gestión de usuarios, inmuebles y reservas con una experiencia de usuario clara y accesible. Este proyecto no solo permite aplicar los conceptos de diseño y arquitectura de software aprendidos en el entorno académico, sino que también proporciona una visión práctica del desarrollo de aplicaciones web completas, desde la planificación inicial hasta la implementación y la validación de funcionalidades.

# Planificación y Gestión

## Metodología de Trabajo

Para el desarrollo de esta práctica se ha seguido una **metodología ágil iterativa e incremental**, definiendo fases de trabajo pequeñas y controlables que permiten avanzar progresivamente y garantizar la funcionalidad del sistema en cada entrega. Cada fase se ha concebido como un **sprint**, con objetivos concretos y entregables funcionales que permiten validar los avances y detectar posibles errores o mejoras a tiempo.

Cada sprint tuvo como objetivo un incremento funcional del sistema, asegurando que al final de cada iteración la aplicación fuera ejecutable y funcional. Además, durante el desarrollo se realizaron ajustes continuos en la configuración y en la planificación, demostrando la capacidad del equipo para adaptarse a cambios y resolver problemas técnicos de manera ágil.

### *Fase 1: Configuración del Entorno*

Se preparó el entorno utilizando **Eclipse** como IDE, por ser la herramienta con la que el equipo tiene mayor experiencia. Se integró **Maven** para la gestión de dependencias y automatización de compilaciones, y **Apache Derby** embebido como base de datos, gestionada y verificada mediante **DBeaver**.

### *Fase 2: Estructura de paquetes y clase principal*

Se diseñó la **estructura de carpetas** del proyecto siguiendo el UML proporcionado, organizando los paquetes en presentación, negocio y persistencia. Esto facilitó la definición clara de responsabilidades y la posterior implementación del sistema.

*Fase 3: Implementación inicial de entidades, controladores y repositorios*

Una vez creada la estructura, decidimos continuar **implementando** algunos **códigos esenciales** que consideramos más importantes que el resto para esta entrega intermedia. Son **Usuario, GestorUsuarios y VentanaRegistro.** Con esta implementación conseguimos los siguientes resultados:

*Fase 4: Integración de vistas HTML con Thymeleaf y verificación de persistencia*

Se implementaron las interfaces de usuario mediante plantillas HTML, integradas en el proyecto dentro de la carpeta **templates**. Esto permitió la validación de la persistencia de datos y la interacción real con el sistema mediante formularios de registro y login.

## Adaptación a Cambios

Durante el desarrollo del proyecto, el equipo tuvo que enfrentarse a diversos cambios y ajustes en la configuración del sistema. Entre estos se incluyen la **actualización del archivo pom.xml,** necesaria para gestionar correctamente las dependencias y asegurar la compatibilidad con las librerías utilizadas, así como el **cambio del motor de base de datos de H2 a Derby embebido,** con el objetivo de cumplir con los requisitos establecidos y facilitar la gestión de datos de manera persistente.

Asimismo, se llevaron a cabo **correcciones en las rutas de los paquetes,** reorganizándolos según el UML proporcionado por el profesor, para mantener la coherencia del proyecto y garantizar que todas las clases estuvieran correctamente estructuradas dentro de la arquitectura multicapa.

Estos ajustes reflejan la capacidad del equipo para adaptarse a cambios durante el proceso de desarrollo, así como la habilidad para resolver problemas técnicos de forma ágil, asegurando que el proyecto avanzara de manera controlada y sin comprometer la funcionalidad del sistema.

# Codificación del Proyecto

La fase de codificación del proyecto Homygo constituye un paso fundamental en el desarrollo, ya que consiste en transformar los diseños y especificaciones previas en un producto de software funcional y operativo. Durante esta etapa, se **implementaron** todas las **funcionalidades esenciales del sistema**, garantizando que la lógica de negocio definida en el análisis y diseño se ejecutara correctamente en un entorno real.

Asimismo, esta fase permitió asegurar que la información se gestionara de manera coherente y segura dentro de la base de datos, integrando las capas de presentación, negocio y persistencia del proyecto. La codificación no solo sirvió para poner en marcha el sistema, sino también para **validar la estructura y los conceptos de diseño aplicados,** asegurando que la aplicación Homygo fuera funcional, mantenible y preparada para futuros desarrollos o ampliaciones.

## Enfoque general de desarrollo

El enfoque general de desarrollo del proyecto Homygo se ha centrado en garantizar un código **limpio, modular y fácil de mantener.** Se ha utilizado **Java** como lenguaje principal, aprovechando su robustez, portabilidad y compatibilidad con frameworks modernos de desarrollo web.

La aplicación se ha estructurado siguiendo la arquitectura **Modelo-Vista-Controlador (MVC),** lo que permite separar de manera clara la lógica de negocio, la presentación de la información y la gestión de datos. Esta separación facilita la escalabilidad del sistema y simplifica el mantenimiento a largo plazo.

Durante todo el desarrollo, se ha priorizado la **modularidad**, la **legibilidad** y **la reutilización del código,** asegurando que cada componente cumpla una función concreta y pueda integrarse de manera coherente con el resto del sistema. Esta aproximación también facilita la incorporación de nuevas funcionalidades sin comprometer la estabilidad del proyecto.

## Frameworks y tecnologías utilizadas

| **Tecnología / Framework** | **Propósito principal** |
|---|---|
| Spring Boot | Framework principal que simplifica la configuración del entorno y la ejecución de la aplicación. |
| Spring Web (Spring MVC) | Gestión de peticiones HTTP, enrutamiento y controladores REST. |
| Spring Data JPA | Implementación de la capa de persistencia de datos, facilitando la interacción con la base de datos mediante entidades y repositorios. |
| Maven | Herramienta de gestión de dependencias y automatización de la construcción del proyecto. |
| Apache Derby | Base de datos embebida para almacenamiento de datos. |
| Thymeleaf | Plantillas HTML para interfaz de usuario. |

## Buenas prácticas implementadas

En el desarrollo del proyecto Homygo se han seguido buenas prácticas de programación para asegurar la calidad, mantenibilidad y fiabilidad del código. Se ha aplicado de manera consistente los principios **SOLID** en las clases del sistema, como Usuario, GestorUsuarios y VentanaRegistro, garantizando que cada clase tenga una **responsabilidad** clara, que las **extensiones** del sistema puedan realizarse **sin modificar código** existente, y que las **dependencias** estén orientadas hacia **abstracciones** y no hacia implementaciones concretas.

Asimismo, se ha implementado un **control de excepciones** adecuado y validación de datos en la clase GestorUsuarios, protegiendo la aplicación de entradas inválidas y evitando errores que podrían comprometer la integridad de los datos de los usuarios.

El código se ha documentado con comentarios descriptivos, siguiendo la convención de nombres estándar de Java, lo que facilita su comprensión y mantenimiento tanto para los miembros del equipo como para futuros desarrolladores.

Finalmente, se han realizado **pruebas unitarias básicas** en los métodos más críticos, como el registro y login de usuarios, asegurando que las funcionalidades esenciales del proyecto se comporten correctamente antes de su integración con otras partes del sistema. Esto ha permitido validar de forma temprana el correcto funcionamiento de las operaciones fundamentales de Homygo.



## Conclusión de la fase de codificación

La fase de codificación del proyecto Homygo permitió obtener un **prototipo funcional completo**, en el que se integran correctamente las tres **capas fundamentales**: presentación, lógica de negocio y persistencia de datos. Gracias al uso de **Spring Web (MVC) y Spring Data JPA**, se consiguió reducir la complejidad técnica del desarrollo, automatizando gran parte de la gestión de datos y simplificando la comunicación entre la base de datos y la aplicación. Esto garantizó la coherencia y consistencia de la información, permitiendo que las operaciones se ejecuten de forma confiable y segura dentro del sistema. El prototipo resultante sirvió como base sólida para validar el diseño y facilitar futuras ampliaciones del proyecto.

# Gestión de Configuración

En este apartado trataremos cómo se ha organizado y controlado el proyecto Homygo durante su desarrollo, incluyendo la **gestión de dependencias**, el **control de versiones** y la **distribución del trabajo**. Esta sección muestra las herramientas y prácticas implementadas para asegurar que el proyecto sea reproducible, mantenible y fácilmente escalable, facilitando la colaboración del equipo y la trazabilidad de los cambios realizados.

## Uso de Maven

Para la gestión de dependencias y construcción del proyecto, se ha utilizado **Maven** como herramienta principal. El archivo pom.xml define las dependencias esenciales para el funcionamiento de Homygo, entre las que se incluyen: **spring-boot-starter-web** para la gestión de peticiones HTTP y controladores web, **spring-boot-starter-data-jpa** para la capa de persistencia mediante JPA**, spring-boot-starter-thymeleaf** para la integración de plantillas HTML, así como **derby y derbytools** para el manejo de la base de datos embebida. Maven facilita la compilación automática, la actualización de librerías y garantiza la portabilidad del proyecto a distintos entornos de desarrollo.

## Control de versiones

Para el control de versiones, se utilizó **Git** de manera puntual y estratégica. Aunque no se realizaron **commits** frecuentes, los que se hicieron fueron muy eficaces, ya que permitieron subir al repositorio avances consolidados y estables del proyecto Homygo. Esta estrategia facilitó mantener un **historial de cambios significativo**, asegurando que las versiones subidas fueran funcionales y coherentes, y permitiendo la trazabilidad de los progresos más importantes del desarrollo.

## Distribución del trabajo

En cuanto a la distribución del trabajo, el equipo **documentó** de manera sistemática los avances y las pruebas realizadas a lo largo del desarrollo del proyecto Homygo. Se utilizó Eclipse como entorno principal de programación, aprovechando su integración con Maven y su facilidad para manejar proyectos Java, mientras que DBeaver se empleó como herramienta de verificación y gestión de la base de datos Derby, permitiendo comprobar y modificar los datos de manera segura y eficiente. Esta combinación aseguró que el trabajo estuviera organizado y que cada miembro del equipo pudiera verificar la correcta funcionalidad de las distintas partes del sistema.

# Arquitectura del proyecto

El proyecto Homygo se ha estructurado siguiendo una **arquitectura Modelo-Vista-Controlador (MVC)** que permite separar de manera clara la lógica de negocio, la presentación y la gestión de datos. Esta organización asegura modularidad, mantenibilidad y facilidad de escalado del sistema:

- **Modelo:** Representado por las entidades del sistema, como Usuario, Inmueble, Reserva, Propietario, Inquilino, entre otras. Estas clases definen la estructura de los datos y sus relaciones, sirviendo como base para la persistencia y la lógica de negocio.
- **Controlador:** Encargado de gestionar la lógica de negocio mediante clases como GestorUsuarios, GestorReservas o GreetingController. Estas clases procesan las solicitudes de los usuarios, coordinan operaciones sobre los datos y controlan el flujo de la aplicación.
- **Vista:** Implementada con plantillas Thymeleaf, que permiten generar interfaces dinámicas y seguras donde los usuarios interactúan con el sistema.
- **Persistencia:** Gestionada mediante DAOs (AbstractEntityDAO, ReservaDAO) y Spring Data JPA (GreetingDAO) para algunas entidades. Esto permite abstraer la comunicación con la base de datos y facilitar la realización de operaciones CRUD.

# Base de datos y persistencia

Se ha **utilizado Apache Derby** en modo embebido como sistema de gestión de bases de datos, asegurando portabilidad y simplicidad en el despliegue. Los archivos de la base de datos se encuentran organizados en el directorio **/database**, incluyendo ficheros .dat, .ctrl y .lck, con estricta adherencia a las reglas de seguridad indicadas en los archivos *README_DO_NOT_TOUCH_FILES* para evitar la corrupción de datos.

La persistencia se ha gestionado mediante:

- DAOs propios para operaciones generales sobre las entidades.
- Spring Data JPA para facilitar la persistencia de ciertas entidades, como Greeting, con soporte automático de operaciones CRUD.

Esto garantiza que todas las operaciones sobre datos sean consistentes y seguras.

# Flujo de funcionamiento y pantallas

El flujo principal de la aplicación se organiza de la siguiente manera:

1. **Registro y login de usuarios:** Accesibles mediante las rutas /usuarios/registro y /usuarios/login, gestionando la creación de usuarios y la autenticación de manera segura.
2. **Gestión de inmuebles:** Alta, búsqueda y reservas de inmuebles, coordinadas mediante controladores específicos.
3. **Visualización de datos:** Interfaces HTML renderizadas con Thymeleaf, que permiten interacción directa con los usuarios y visualización de información de forma clara.
4. **Persistencia y validación:** Todas las operaciones sobre datos se almacenan en la base de datos Derby y se validan mediante los controladores y DAOs, asegurando integridad y coherencia.

# Conclusiones

El proyecto Homygo ha logrado:

- Una arquitectura modular y escalable, con separación clara de responsabilidades.
- La correcta implementación del patrón MVC y del patrón DAO, asegurando mantenimiento y evolución futura.
- Un prototipo funcional listo para extender funcionalidades relacionadas con reservas, gestión de inmuebles y pagos.
- Una coordinación de equipo efectiva y capacidad de adaptación ágil ante cambios y problemas técnicos durante el desarrollo.