**GRADO EN INGENIERÍA INFORMÁTICA**

DESARROLLO DE SOFTWARE


UNIVERSIDAD
DE GRANADA

**Ejercicio Individual**

# Conversión de Distancias entre Sistemas de Navegación

## Implementación del Patrón Adaptador

Blanca Girón Ricoy (*blancagiron@correo.ugr.es*)

**Repositorio:** https://github.com/blancagiron/DS-Individual

Marzo 2025

# Índice

# 1. Introducción y objetivos

## 1.1. Introducción

En los sistemas de información geográfica y las aplicaciones de navegación, el uso de distintas unidades de medida hace necesario contar con mecanismos de conversión para asegurar una integración fluida entre sistemas. En este ejercicio, se aborda el problema de conversión de distancias entre dos sistemas que utilizan unidades diferentes: un servicio de información geográfica en España, que trabaja con kilómetros, y una aplicación de navegación de Los Ángeles, que proporciona las distancias en millas.

A lo largo de esta tarea se desarrolla una solución que permita integrar ambos sistemas sin necesidad de modificar su estructura original ni las clases existentes. Para ello, se implementa el **Patrón de Diseño Adaptador**, que facilita la conversión automática de las distancias de millas a kilómetros, asegurando la compatibilidad entre ambos servicios.

Además, se ha desarrollado una interfaz gráfica en React, utilizando Vite y TailwindCSS, que simula un entorno real de uso. Los usuarios pueden ingresar distancias en millas y visualizar el resultado en kilómetros. Este enfoque garantiza una solución modular, reutilizable y escalable.

## 1.2. Objetivos

Se plantean los siguientes objetivos específicos:

- **Analizar el problema** entre el servicio de información geográfica en España y la aplicación de navegación en Los Ángeles.
- **Estudiar y decidir que patrón es el adecuado para resolver el problema**,que permita transformar las distancias de millas a kilómetros de manera eficiente y sin alterar la estructura del código original.
- Desarrollar una **interfaz de usuario** que simule un entorno real de uso, permitiendo a los usuarios ingresar distancias en millas y obtener su equivalencia en kilómetros.

# 2. Patrón de Diseño Utilizado

## 2.1. Patrón Adaptador

En este ejercicio se busca convertir las distancias de millas a kilómetros de forma transparente, permitiendo que ambos sistemas trabajen de forma conjunta, a pesar de la incompatibilidad de unidades de medida entre ellos. La cuestión principal es encontrar una manera en la que integrar los dos sistemas.

Para resolver este problema, se ha decidido implementar el **Patrón Adaptador**.

La elección se basa en varios factores que lo hacen el patrón ideal para este caso:

- **Compatibilidad entre sistemas con interfaces incompatibles**: Este patrón permite que el sistema español, que trabaja con kilómetros, pueda utilizar los datos de la aplicación de Los Ángeles, que utiliza millas, sin tener que modificar ninguno de los dos sistemas. El adaptador actúa como un puente, realizando la conversión.
- **Minimizar el impacto en el código existente**: En el enunciado se especifica que el servicio español debe utilizar la información de la aplicación de Los Ángeles sin modificar las clases existentes. El patrón Adaptador permite integrar sistemas sin tener que alterar el código fuente existente.
- **Escalabilidad y reutilización:** Este patrón favorece la reusabilidad del código al mantenerlo modular y organizado. Esto resulta útil si en el futuro se presentan nuevos escenarios que requieran la conversión entre otras unidades de medida. Se pueden agregar fácilmente nuevos adaptadores sin necesidad de modificar el código original de los sistemas involucrados.
- **Lógica del cliente:** Usando este patrón, el servicio español (el cliente) no tiene que preocuparse por las conversiones de unidades, ya que el adaptador realiza esta tarea automáticamente. De esta forma el servicio español recibe las distancias en el formato esperado, en kilómetros.

El **Patrón Adaptador** cuenta con estas componentes principales:

- **Target**: Es la interfaz que espera el cliente.
- **Adaptee**: Es lo que se quiere adaptar. Es la fuente de datos original con una interfaz incompatible.
- **Adapter**: Actúa como intermediario, convirtiendo los datos del *Adaptee* al formato requerido por el *Target*.
- **Client**.

## 2.2. Aplicación en el Proyecto

En este proyecto, el **Patrón Adaptador** permite integrar sistemas con diferentes unidades de medida sin modificar el código original. En este caso:

- `ServicioLA.js` como **Adaptee**: Actúa como la clase a adaptar. Representa el sistema que proporciona distancias en millas.
- `Adaptador.js` como **Adapter** y **Target**: En el archivo `Adaptador.js`, la clase `Adaptador` cumple el rol de **Adapter**, pero también queda definido de forma implícita el **Target**.
    - **Adapter:** La clase `Adaptador` adapta la salida del `ServicioLA` (en millas) a la unidad esperada por el servicio español (kilómetros). Utiliza el método `getDistanciaKm()` para realizar esta conversión.

  - **Target:** Aunque JavaScript no usa interfaces como en otros lenguajes, el método `getDistanciaKm()` actúa como el **Target** implícito, proporcionando la interfaz esperada por el cliente para obtener la distancia en kilómetros.

- `App.jsx` como **Cliente**: Utiliza el Adaptador para interactuar con el ServicioLA, sin necesidad de modificar el código original. Además, contiene la interfaz de usuario implementada con React.

# 3. Desarrollo e Implementación

## 3.1. Decisiones de Diseño

Tras la identificación del patrón adecuado para resolver el problema planteado, el siguiente paso fue la elección del lenguaje y el entorno de desarrollo. Para la implementación del sistema, se consideraron diferentes lenguajes de programación, con el objetivo de seleccionar la opción más adecuada para aplicar el **patrón Adaptador** y garantizar una integración eficiente entre servicios.

Inicialmente, se consideró utilizar **Java**, siguiendo así fielmente la estructura del patrón Adaptador presentada en los apuntes de la asignatura. Java posee una estructura clara de clases e interfaces, lo que facilita la comprensión de los patrones de diseño orientados a objetos. Sin embargo, la creación de interfaces gráficas atractivas con Swing requiere más código y ofrece menos flexibilidad para el diseño.

Otra alternativa analizada fue **Python**, debido a su sintaxis sencilla, familiaridad con su uso y su amplio sistema de bibliotecas, lo que habría permitido implementar el adaptador de forma ágil con menor cantidad de código en comparación con Java. Por otra parte, la interfaz gráfica con tkinter o PyQt no ofrecía la experiencia de desarrollo ni el resultado estético buscado.

Finalmente, tras ver los requisitos del ejercicio y priorizar factores de diseño como la experiencia de usuario, la facilidad de mantenimiento y una representación realista del escenario planteado, además de que ya se ha utilizado para implementar soluciones similares en otras asignaturas, la opción seleccionada ha sido **JavaScript** con **React**:

- **Entorno de desarrollo más realista**: Las aplicaciones de navegación con frecuencia usan tecnologías web, por lo que una implementación con JavaScript y React representa un escenario más cercano a cómo se podría implementar esta solución en un entorno real.
- **Aprendizaje**: Implementando el patrón con React y JavaScript se aplican los conceptos teóricos del patrón Adaptador en un entorno distinto al presentado en clase, demostrando así la universalidad del patrón y permitiendo comprenderlo mejor. Además, se profundizan los conocimientos sobre estas tecnologías adquiridos en experiencias previas.
- **Personalización de la interfaz de usuario**: React facilita la creación de interfaces de usuario modernas, interactivas y responsivas, lo que mejora la experiencia del usuario.

## 3.2. Tecnologías Utilizadas

Para implementar la solución, se seleccionaron las siguientes tecnologías:

- **JavaScript**: Como lenguaje de programación para implementar el patrón, debido a su flexibilidad y capacidad para ejecutarse tanto en el frontend como en la lógica de conversión.
- **React**: Para el frontend. Facilita la construcción de aplicaciones dinámicas y reutilizables, dividiendo la interfaz en componentes, actualizando solo los elementos necesarios y facilitando la gestión del estado de la aplicación mediante React Hooks.

- **TailwindCSS**: Para los estilos, es un framework de CSS que permite diseñar interfaces de manera rápida y eficiente.
- **Vite**: Como herramienta de construcción y servidor de desarrollo.

## 3.3. Diseño e Interfaz Gráfica

La interfaz gráfica ha sido diseñada con un enfoque en la usabilidad, siguiendo distintos principios de diseño centrados en el usuario:

- **Estructura responsiva**: El diseño se adapta a distintos tamaños de pantalla. En dispositivos de escritorio se muestra un diseño en dos columnas, mientras que en dispositivos móviles se reorganiza en un diseño vertical.
- **Componentes**:
  - Sección Izquierda: Esta sección proporciona contexto sobre la información, incluye un encabezado de sección, un título principal, un párrafo explicativo sobre el propósito de la aplicación y un elemento visual *localización.svg* para reforzar el concepto de servicio geográfico.
  - Sección Derecha: Contiene los elementos interactivos para realizar la conversión.
- **Usabilidad y estética**: la interfaz emplea un diseño moderno e intuitivo con un esquema de colores consistente y tipografía clara con jerarquía visual.

Antes de comenzar la implementación en React, se diseñó un prototipo utilizando **Canva** como herramienta de diseño visual. Este paso permitió una definición visual previa, facilitando la visualización de la estructura, la disposición de elementos y el esquema de colores antes de escribir cualquier línea de código. El prototipo sirvió como guía de referencia durante el desarrollo, asegurando la coherencia entre el diseño planificado y la implementación final.


Figura 1: Prototipo de la interfaz diseñado en Canva

# 4. Resultados Obtenidos

A continuación, se muestran capturas de la aplicación en funcionamiento:

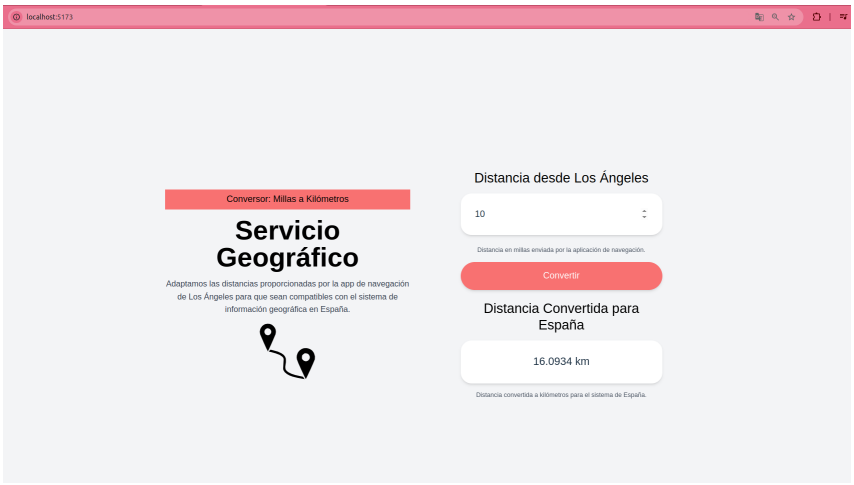


Figura 2: Versión de escritorio


Figura 3: Versión móvil