Proyecto Final: Sistema de Observación Meteorológica

Jorge Domínguez González (alu0101330600@ull.edu.es)
Adnan Hawari Capa (Alu0100417012@ull.edu.es)
Paula Regalado De León (Alu0101330174@ull.edu.es)

January 12, 2024

Contents

List of Figures

1 Introducción

1.1 Práctica anterior

En la anterior práctica se presenta una implementación de un sistema de observación meteorológica en Java. Este sistema utiliza el patrón de diseño Observador para recibir actualizaciones de las condiciones meteorológicas de diversas ciudades a través de la API de WeatherStack. WeatherStack es un servicio de API de pronóstico del tiempo que proporciona información actualizada sobre condiciones meteorológicas actuales, incluyendo temperatura, velocidad del viento, humedad, presión atmosférica y más.

1.2 Objetivo de la Práctica

El objetivo de la práctica es implementar un historial de los distintos datos recibidos, es decir, desarrollar una aplicación de visualización de datos meteorológicos utilizando Java y el framework JFreeChart. La aplicación permite seleccionar una ciudad y visualizar datos meteorológicos como temperatura, probabilidad de precipitación, velocidad del viento, presión atmosférica y cobertura de nubes. Además, el usuario puede elegir entre dos tipos de gráficos: líneas y barras.

2 Estructura del Directorio

En la figura 1 podemos observar la estructura de nuestro proyecto.

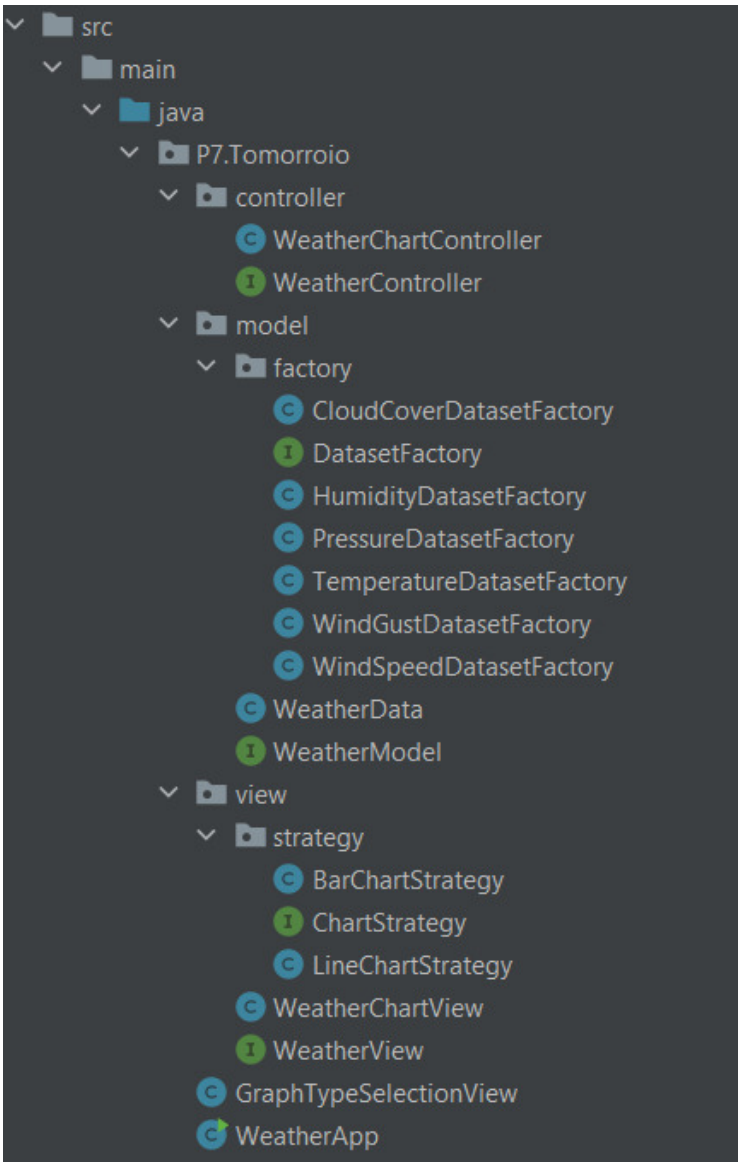


Figure 1: Estructura del directorio del proyecto.

3 Estructura del Código

3.1 Clase WeatherApp

Clase principal que contiene el método main. Crea una instancia de WeatherChartController y muestra la vista.

3.2 Clase WeatherView

Interfaz que define el contrato para las vistas de la aplicación. Proporciona métodos para mostrar la vista y actualizar los gráficos con nuevos conjuntos de datos.

3.2.1 Clase WeatherChartView

Implementa la interfaz WeatherView. Crea y muestra una ventana que contiene un gráfico. Proporciona métodos para actualizar el gráfico con nuevos conjuntos de datos.

3.3 Clase GraphTypeSelectionView

Crea y muestra una ventana que permite al usuario seleccionar el tipo de gráfico y la ciudad. Proporciona métodos para obtener la ciudad y el tipo de gráfico seleccionados por el usuario.

3.4 Clase ChartStrategy

La interfaz 'ChartStrategy' define el contrato para las estrategias de gráficos. A través de esta interfaz, especificamos los métodos necesarios que las estrategias concretas deben implementar para la creación de gráficos.

3.4.1 Clase LineChartStrategy

La clase 'LineChartStrategy' implementa la interfaz 'ChartStrategy'. Es una estrategia abstracta que proporciona métodos específicos para crear un gráfico de líneas utilizando JFreeChart.

3.4.2 Clase BarChartStrategy

La clase 'BarChartStrategy' es una implementación concreta de 'ChartStrategy' especializada en la creación de gráficos de barras. Hereda de 'LineChartStrategy' y proporciona la implementación específica para crear gráficos de barras utilizando JFreeChart.

3.5 Clase WeatherData

La clase 'WeatherData' implementa la interfaz 'Observable' y proporciona métodos para obtener datos meteorológicos actualizados de la API de Tomorrow.io. También proporciona métodos para crear conjuntos de datos utilizando 'DatasetFactory'.

3.6 Clase DatasetFactory

La interfaz ‘DatasetFactory‘ define el contrato para las factorías de conjuntos de datos. A través de esta interfaz, especificamos los métodos necesarios que las factorías concretas deben implementar para la creación de conjuntos de datos.

3.6.1 Clase TemperatureDatasetFactory

La clase ‘TemperatureDatasetFactory‘ implementa ‘DatasetFactory‘ y se especializa en la creación de conjuntos de datos para la temperatura. Proporciona la implementación necesaria para este tipo de dato meteorológico.

3.6.2 Clase HumedityDatasetFactory

La clase ‘HumedityDatasetFactory‘ implementa ‘DatasetFactory‘ y se especializa en la creación de conjuntos de datos para la humedad. Proporciona la implementación necesaria para este tipo de dato meteorológico.

3.6.3 Clase WindSpeedDatasetFactory

La clase ‘WindSpeedDatasetFactory‘ es una implementación concreta de ‘DatasetFactory‘ especializada en la creación de conjuntos de datos para la velocidad del viento.Proporciona la implementación específica para este tipo de dato meteorológico.

3.6.4 Clase PressureDatasetFactory

La clase ‘PressureDatasetFactory‘ implementa ‘DatasetFactory‘ y se especializa en la creación de conjuntos de datos para la presión atmosférica. Proporciona la implementación necesaria para este tipo de dato meteorológico.

3.6.5 Clase WindGustDatasetFactory

La clase ‘WindGustDatasetFactory‘ es una implementación concreta de ‘DatasetFactory‘ especializada en la creación de conjuntos de datos para la ráfaga de viento.Proporciona la implementación específica para este tipo de dato meteorológico.

3.6.6 Clase CloudCoverDatasetFactory

La clase ‘CloudCoverDatasetFactory‘ implementa ‘DatasetFactory‘ y se especializa en la creación de conjuntos de datos para la cobertura de nubes. Proporciona la implementación necesaria para este tipo de dato meteorológico.

Estas clases concretas de factorías implementan métodos específicos para la creación de conjuntos de datos relacionados con sus respectivos tipos de datos meteorológicos. Al emplear el patrón Abstract Factory, logramos una estructura modular que facilita la extensión del sistema al agregar nuevos tipos de datos meteorológicos en el futuro.

3.7 Clase WeatherChartController

La clase ‘WeatherChartController‘ es el controlador principal de la aplicación. Implementa la interfaz ‘Observer‘ y proporciona métodos para actualizar la vista con nuevos conjuntos de datos. También proporciona métodos para obtener datos meteorológicos actualizados de ‘WeatherData‘ y crear conjuntos de datos utilizando ‘DatasetFactory‘.

3.8 WeatherController

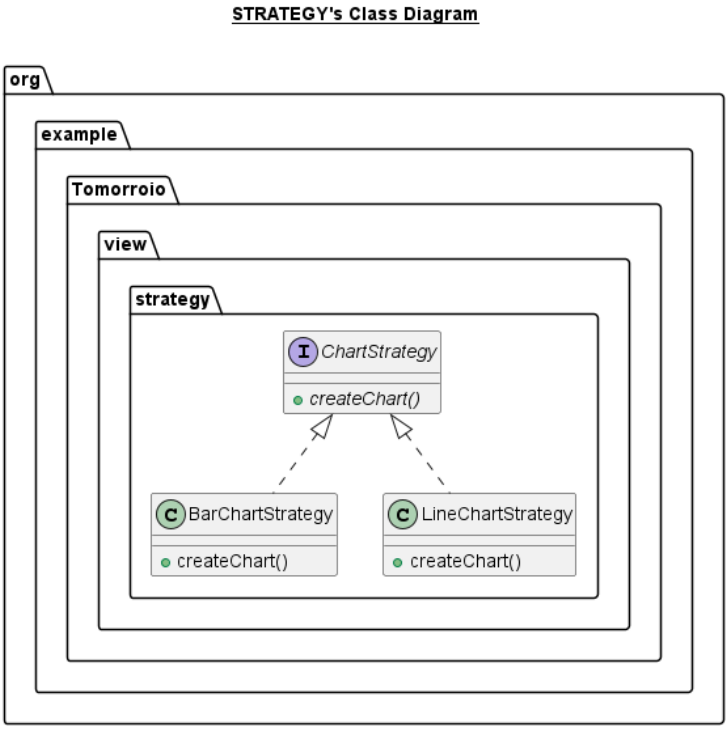
La interfaz ‘WeatherController‘ define el contrato para los controladores de la aplicación. A través de esta interfaz, especificamos los métodos necesarios que los controladores concretos deben implementar para la gestión de la aplicación.

4 Patrones

En esta práctica usamos varios patrones que hemos visto a lo largo de la asignatura, como puede ser el patrón estrategia, el abstract factory y el modelo de vista controlador que iremos señalando en el siguiente apartado:

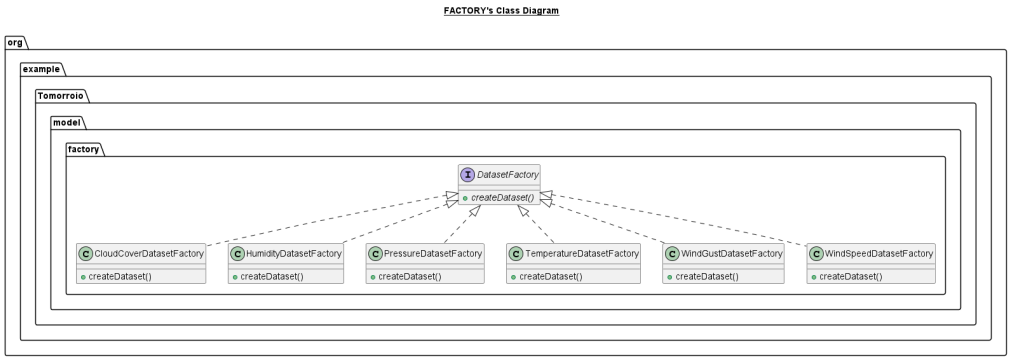
4.1 Patrón Estrategia

El patrón de diseño Estrategia se ha utilizado para implementar diferentes tipos de gráficos. Como se puede observar en la figura 4.1, la clase WeatherChartView utiliza una estrategia para crear un gráfico de líneas o de barras. Esto permite que la aplicación sea flexible y extensible, ya que podemos agregar nuevos tipos de gráficos en el futuro sin modificar el código existente.



4.2 Patrón Abstract Factory

El patrón de diseño Abstract Factory se ha utilizado como se puede observar en la figura 4.2 para crear de forma dinámica los diferentes tipos de datos de forma ordenada y flexible.



4.3 Patrón Modelo Vista Controlador

El patrón de diseño Modelo Vista Controlador se ha utilizado para separar la lógica de la aplicación de la interfaz gráfica. Como se puede observar en la figura 4, la clase WeatherChartController actúa como el controlador principal de la aplicación. Implementa la interfaz WeatherController y proporciona métodos para actualizar la vista con nuevos conjuntos de datos.

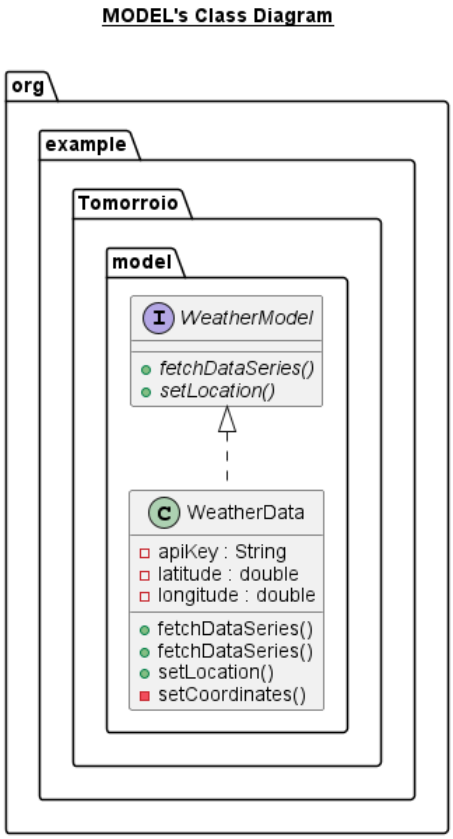


Figure 2: Implementación Modelo

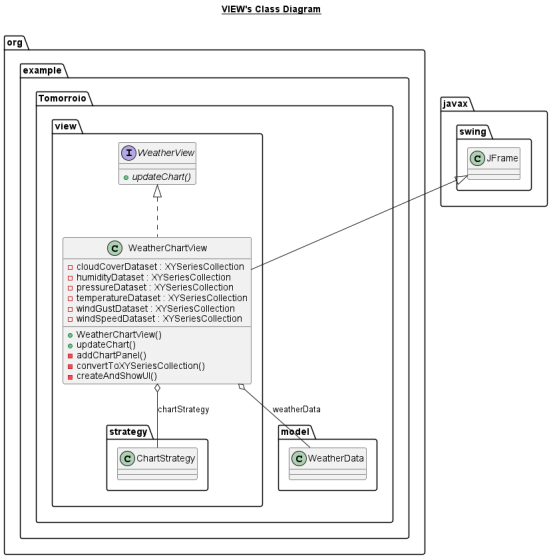


Figure 3: Implementación Vista

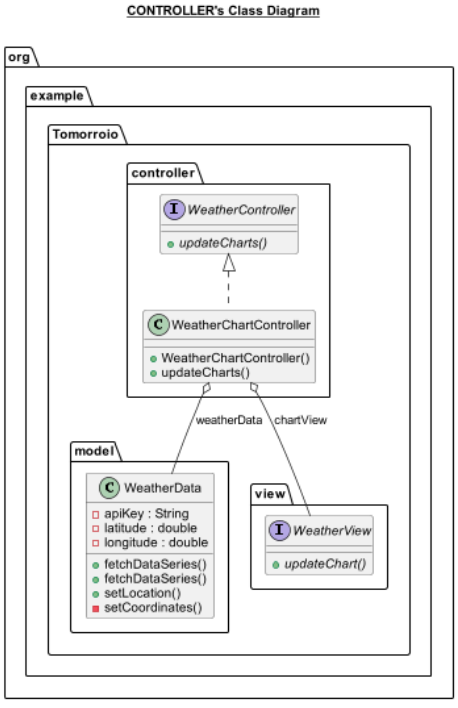


Figure 4: Implementación Controlador

5 Pasos del Flujo del Programa

El programa sigue estos pasos:

1. El usuario selecciona una ciudad y un tipo de gráfico.
2. El controlador principal crea una instancia de la vista y la muestra.
3. La vista crea una instancia de la estrategia de gráfico seleccionada por el usuario.
4. La vista crea una instancia de la factoría de conjuntos de datos correspondiente al tipo de gráfico seleccionado por el usuario.
5. La vista crea un gráfico utilizando la estrategia y el conjunto de datos.
6. El controlador principal crea una instancia de WeatherData.
7. WeatherData obtiene datos meteorológicos de la API de Tomorrow.io.
8. El controlador principal recibe los datos meteorológicos actualizados y crea un conjunto de datos utilizando la factoría de conjuntos de datos.
9. El controlador principal actualiza la vista con el nuevo conjunto de datos.
10. El usuario puede seleccionar una nueva ciudad y un nuevo tipo de gráfico.

6 Interfaz Gráfica para el Usuario

La interfaz gráfica para el usuario se muestra en las figuras 5, 6, 7 y 8.

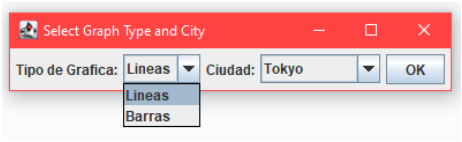


Figure 5: Interfaz gráfica para el usuario.

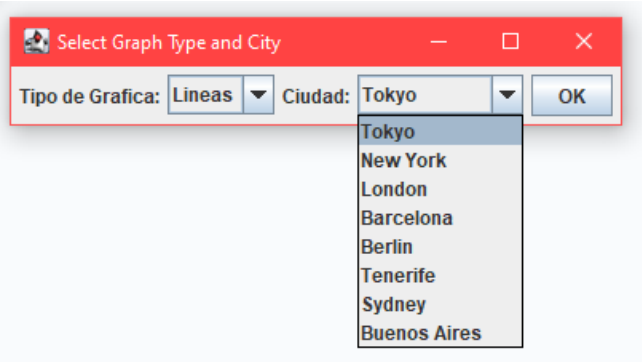


Figure 6: Interfaz gráfica para el usuario.

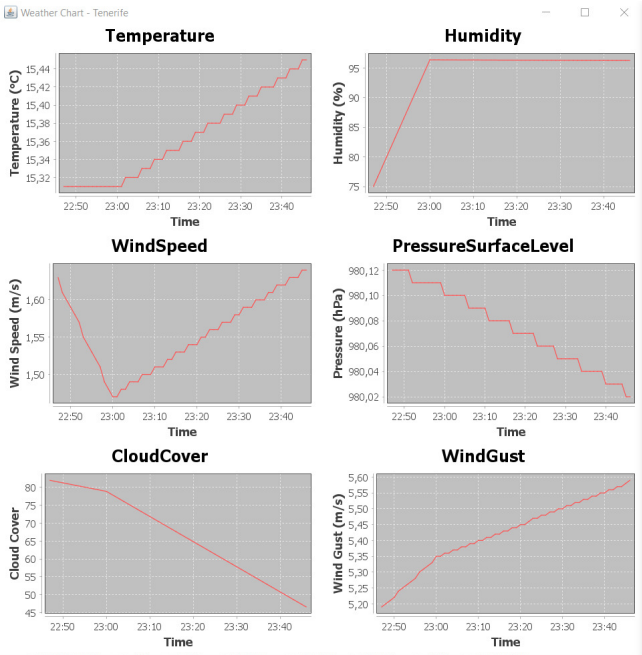


Figure 7: Interfaz gráfica para el usuario.

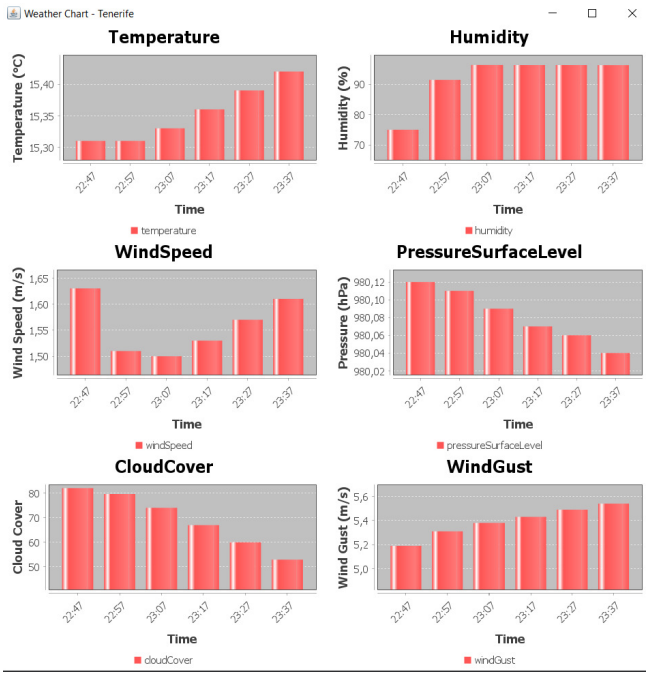


Figure 8: Interfaz gráfica para el usuario.

7 Diagrama UML

El diagrama UML del proyecto se muestra en la Figura 9.

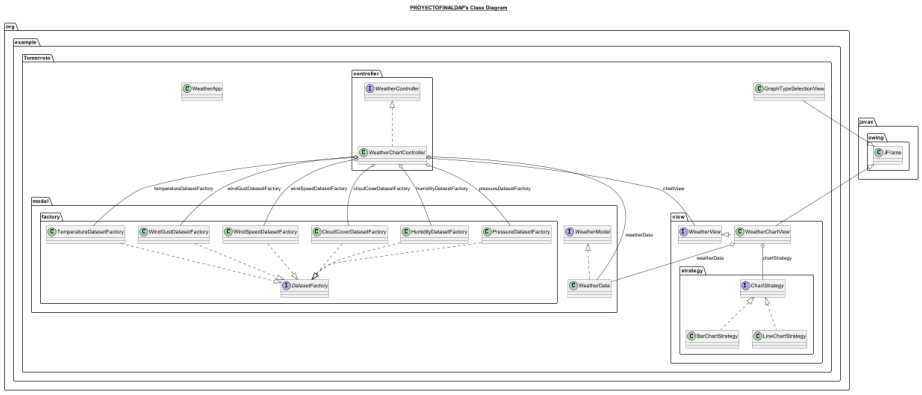


Figure 9: Diagrama UML del proyecto.

8 Errores y Desafíos Encontrados

Durante el desarrollo del proyecto, nos enfrentamos a varios desafíos que nos permitieron mejorar nuestras habilidades de resolución de problemas y fortalecer la calidad general del sistema.

8.1 Conexión con la API de Tomorrow.io

Uno de los desafíos iniciales fue establecer una conexión efectiva con la API de Tomorrow.io para obtener datos meteorológicos en tiempo real. En algunos casos, experimentamos problemas de tiempo de espera y errores de conexión. Para superar esto, implementamos un manejo robusto de errores y optimizamos nuestras solicitudes para mejorar la eficiencia.

8.2 Sincronización de Datos en Tiempo Real

La sincronización de datos en tiempo real para proporcionar actualizaciones constantes en la interfaz gráfica fue un desafío importante. Implementamos un mecanismo de actualización basado en eventos para garantizar que los gráficos reflejaran con precisión los datos meteorológicos más recientes. Esto implicó coordinar eficientemente la obtención de datos, su procesamiento y la actualización de la interfaz de usuario.

8.3 Manejo de Excepciones

En el proceso de desarrollo, identificamos la necesidad de un manejo exhaustivo de excepciones para garantizar la estabilidad y la usabilidad de la aplicación. Implementamos una estrategia sólida de manejo de excepciones para notificar al usuario sobre posibles problemas y evitar fallos inesperados.

Estos desafíos contribuyeron significativamente a nuestro aprendizaje y resiliencia durante el desarrollo del proyecto. La capacidad para superar estos obstáculos mejoró nuestras habilidades de resolución de problemas y fortaleció la calidad general del sistema.

9 Estimación de Plazo

Table 1: Estimación de Plazo

Tarea	**Tiempo Estimado**	**Tiempo Real**
Elegir propuesta	30 minutos	15 minutos
Implementar JFreeChart	3 hora	5 hora y media
Obtener datos historial	30 minutos	50 minutos
Implementar patrones	1 hora	2 horas
Implementar interfaces	2 horas	2 horas y media
Pruebas y depuración	2 horas	3 horas
Documentación	1 hora	1 hora y media
Implementación Github	1 hora y media	1 hora y media

10 Conclusiones

En conclusión, el uso de patrones de diseño ha simplificado el diseño y la implementación de la aplicación. Ha proporcionado una estructura modular y extensible, facilitando la incorporación de nuevas funcionalidades en algún futuro. Además de utilizar diferentes herramientas, como el JFreeChart, para realizar un proyecto bastante práctico y elaborado.

11 Bibliografía

References

[1] omorrow.io API: https://www.tomorrow.io/

[2] FreeChart: https://www.jfree.org/jfreechart/

[3] FreeChart Developer Guide: https://www.jfree.org/jfreechart/api/javadoc/org/jfree/chart/JFreeChart.html

[4] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/index.htm

[5] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_quick_guide.htm

[6] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_overview.htm

[7] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_xy_chart.htm

[8] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_bar_chart.htm

[9] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_line_chart.htm

[10] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_pie_chart.htm

[11] eatherStack API: https://weatherstack.com/

[12] trones de diseño: https://www.tutorialspoint.com/design_pattern/index.htm

[13] atrones de diseño: https://www.geeksforgeeks.org/software-design-patterns/

[14] atrones de diseño: https://www.journaldev.com/1827/java-design-patterns-example-tutorial

[15] atrones de diseño: https://www.javatpoint.com/design-patterns-in-java