TFG Desarrollo web

Bottempo

Antonio José Peleteiro Crespo

- Grado de Ingeniería Informática
- Itinerario de Ingeniería del software
- Área de Desarrollo web

Consultor: Gregorio Robles Martínez

Profesor responsable: Santi Caballe Llobet

08 de Enero de 2021

Universitat Oberta de Catalunya

A Olaia, por facilitarme el regalo del tiempo, sin el cual nunca habría llegado al final de este camino.

A Antón, por ser el faro que mantenía encendida la luz de la motivación en mi horizonte.

A Víctor, por su infinita paciencia cuando le pedía que probase el bot.

A Óscar Otero, por diseñar un logo increíble para la aplicación.

A Isa, mi tutora, por estar todos estos años ahí siempre que la he necesitado.

Al *post-rock*, por inspirarme en tantos duros momentos durante este viaje.

Moitas grazas e... Se chove que chova!

FICHA DEL TRABAJO FINAL	
Título del trabajo:	Bottempo
Nombre del autor:	Antonio José Peleteiro Crespo
Nombre del consultor:	Gregorio Robles Martínez
Nombre del PRA:	Santi Caballe Llobet
Fecha de entrega:	08/01/2021
Titulación:	Grado de Ingeniería Informática
Área del Trabajo Final:	Desarrollo web
Idioma del trabajo:	Castellano
Palabras clave:	Bot, Telegram, Meteorología

Resumen del Trabajo

La realización de este trabajo final pretende, por una parte, llevar a la práctica la mayor parte de los conocimientos adquiridos durante el grado y, por otra, cubrir una necesidad presente en una parte de la sociedad en lo que a obtención de información meteorológica se refiere.

Bottempo es un bot de Telegram creado con el objetivo de poder planificar y gestionar distintas alertas meteorológicas en el territorio de la Comunidad Autónoma de Galicia, obteniendo los datos a través de una API pública que provee la *Consellería de Medio Ambiente* de la *Xunta de Galicia*, disponible para tal fin.

El desarrollo se ha apoyado en 4 pilares principales: el primero ha sido la API anteriormente mencionada; el segundo la utilización del *framework* Laravel para la implementación del esquema básico de la aplicación; el tercero el uso de BotMan, *framework*, que permite la creación de bots para distintos plataformas; y el último pilar la propia plataforma de bots de Telegram.

Además, también se ha aprovechado la implementación para realizar una incursión en el procesamiento del lenguaje natural (*NLP* en inglés), lo que ha permitido asimismo llevar a cabo la internacionalización del bot, al estar disponible en español, gallego e inglés.

Abstract

The completion of this final work pursues, on the one hand, to put into practice most of the knowledge acquired during the degree and, on the other, to cover a present need in a part of society in terms of obtaining meteorological information.

Bottempo is a Telegram bot created with the aim of being able to plan and manage different meteorological alerts in the territory of the Autonomous Community of Galicia, obtaining the data through a public API provided by the Department of Environment of the Regional Government of Galicia, available for that goal.

The development has been based on 4 main pillars: the first has been the aforementioned API; the second the use of Laravel framework for the implementation of the basic scheme of the application; the third the use of BotMan, framework, which allows the creation of bots for different platforms; and the last pillar is the Telegram bot platform itself.

In addition, the implementation has also been used to make a foray into natural language processing (NLP), which has also made it possible to carry out the internationalization of the bot, as it is available in Spanish, Galician and English.

Citas

"*É feliz o que soñando, morre. Desgraciado o que morra sen soñar*"

Rosalía de Castro

"Cualquier noche puede salir el Sol..."

Lorenzo Morales «El Noi»

"*Do or do not, there is no try*"

***Grand Master* Yoda**

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del trabajo

La idea del trabajo surgió de una situación personal cotidiana, la necesidad de saber las condiciones meteorológicas diariamente y en diversas franjas horarias. En concreto a primera hora de la mañana de cara a elegir la ropa para mi hijo, así como al mediodía para saber si es necesario el paraguas a la hora de recogerlo del colegio.

Esta situación, unido a mi afición por Telegram, llevaron a plantearme el desarrollo de un bot en el que pudiese configurar alertas meteorológicas que hiciesen que me llegasen mensajes con la previsión a ciertas horas establecidas previamente o que incluso me notifique de la aparición de algún fenómeno concreto.

Durante la elaboración de este trabajo, se publicó una noticia en el medio digital **9to5Google**, la cual se hace eco de que en el *widget* del lanzador de aplicaciones de los teléfonos Pixel de Google se mostrarán alertas meteorológicas, lo cual no hace más que añadir valor a la propuesta elegida para plasmar en este TFG.

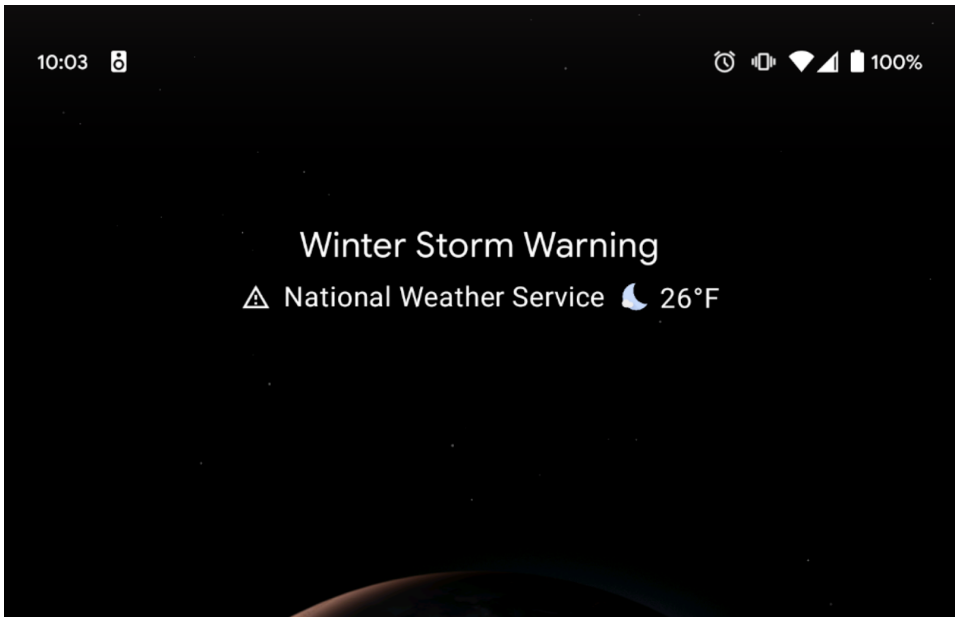


Ilustración 1 - Alerta en el lanzador de Aplicaciones de Android. Fuente: 9to5Google

1.2. Objetivos del Trabajo

Objetivo principal

El objetivo principal que se persigue es la creación de un bot de Telegram que se conecte a una API meteorológica y permita, al usuario, configurar alertas de distintos tipos.

Objetivos secundarios

Los subobjetivos u objetivos secundarios que han permitido adquirir nuevas competencias serían los siguientes:

- Realizar el diseño y la implementación completa de una aplicación funcional, eficiente y escalable utilizando para ello el lenguaje de programación PHP.
- Llevar a cabo el estudio de una API pública para poder consumir datos de la misma e integrarlos en otro sistema.
- Profundizar en la utilización de un *framework* PHP como Laravel que posibilite el desarrollo eficiente y seguro tanto de aplicaciones web como de APIs completas.
- Descubrir el campo de los *chatbots* mediante la implementación de un bot a través del *framework* más popular para ello en PHP, BotMan.
- Investigar la plataforma de bots de Telegram junto con todas las funcionalidades que ofrece para la construcción de un *chatbot*.
- Realizar la configuración del servidor que permitirá el despliegue de la aplicación en producción para poder hacer un uso real del bot.
- Aplicar las técnicas de Gestión de Proyectos adquiridos durante el Grado así como ampliar los conocimientos en este área.
- Profundizar en el sistema de gestión de bases de datos relacional MySQL apoyándose en los conocimientos adquiridos en las asignaturas Uso de bases de datos y Diseño de bases de datos.

1.3. Enfoque y método seguido

Para el análisis y el diseño de la aplicación se han utilizado diversas estrategias de la Ingeniería de software, como son la realización de diversos diagramas UML (de clases, de casos de uso, de secuencia).

En cuanto al diseño de la base de datos, se ha llevado a cabo el ciclo habitual basado en 3 etapas: diseño conceptual, diseño lógico y diseño físico.

Una vez dispuesto los diseños, se ha optado por emplear el sistema *kanban* para la implementación, el cual nos permite utilizar tarjetas para organizar las tareas pendientes, las cuales he dividido en 4 posibles estados: pendientes, en desarrollo, en testing y publicadas.

Para facilitar el seguimiento de las tareas, he utilizado la herramienta Trello, con lo que pude llevar el registro de actividades mediantes tarjetas virtuales, tal y como se aprecia en la siguiente ilustración:

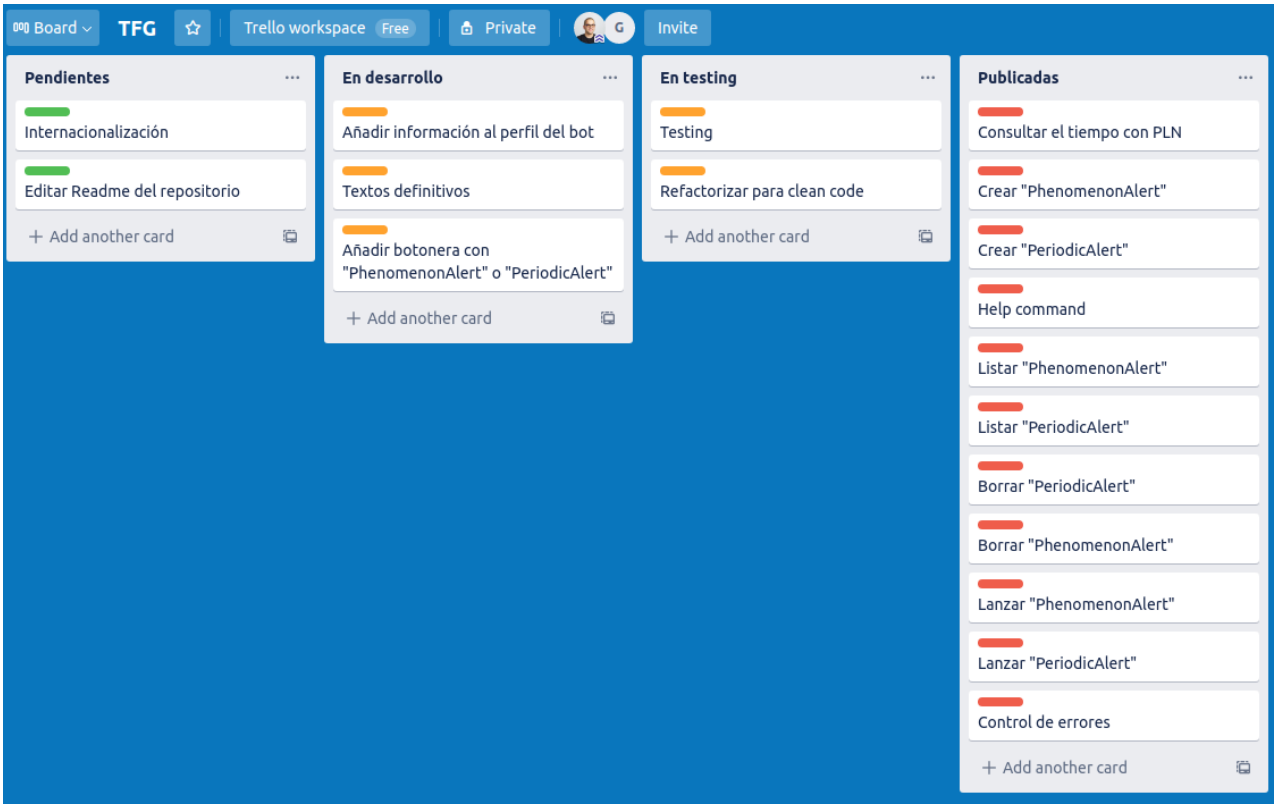
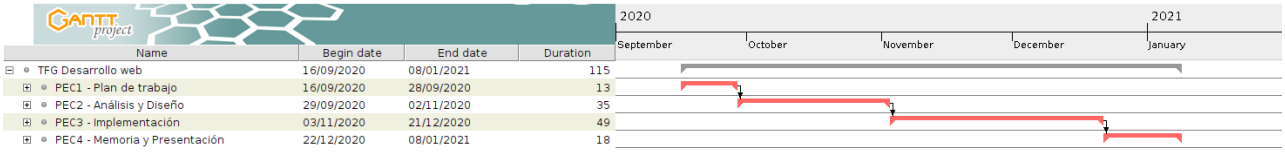


Ilustración 2 - Panel *kanban* de Trello

1.4. Planificación del Trabajo

La planificación del trabajo se ha llevado a cabo en base al tiempo disponible para la realización del proyecto, 115 días desde la fecha de inicio del semestre (16/09/2020) hasta la fecha de la última entrega (08/01/2021).

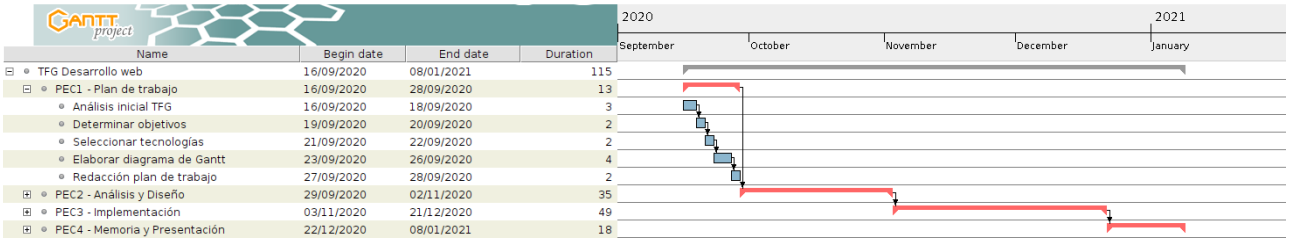


Name	Begin date	End date	Duration
TFG Desarrollo web	16/09/2020	08/01/2021	115
PEC1 - Plan de trabajo	16/09/2020	28/09/2020	13
PEC2 - Análisis y Diseño	29/09/2020	02/11/2020	35
PEC3 - Implementación	03/11/2020	21/12/2020	49
PEC4 - Memoria y Presentación	22/12/2020	08/01/2021	18

Ilustración 3 - Diagrama de Gantt (Completo)

Durante dicho periodo el desarrollo se ha dividido en 4 etapas fundamentales coincidiendo con las distintas entregas intermedias.

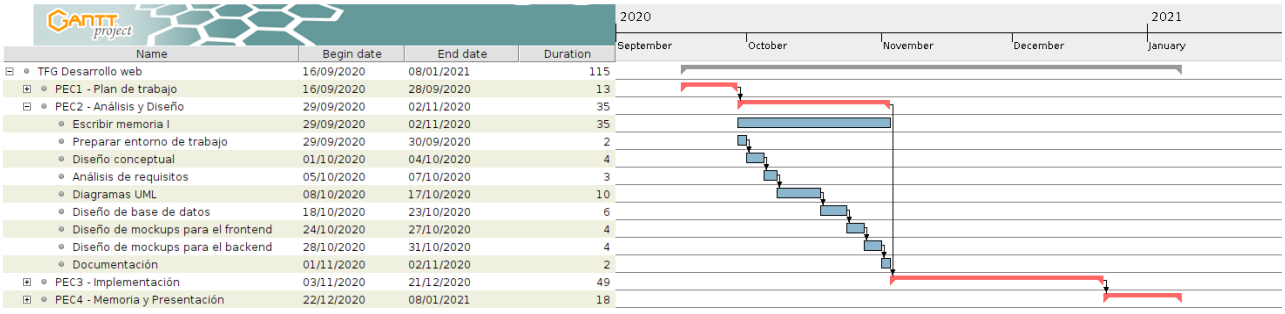
Así, la primera etapa coincide con la realización del Plan de trabajo del TFG:



Name	Begin date	End date	Duration
TFG Desarrollo web	16/09/2020	08/01/2021	115
PEC1 - Plan de trabajo	16/09/2020	28/09/2020	13
Análisis inicial TFG	16/09/2020	18/09/2020	3
Determinar objetivos	19/09/2020	20/09/2020	2
Seleccionar tecnologías	21/09/2020	22/09/2020	2
Elaborar diagrama de Gantt	23/09/2020	26/09/2020	4
Redacción plan de trabajo	27/09/2020	28/09/2020	2
PEC2 - Análisis y Diseño	29/09/2020	02/11/2020	35
PEC3 - Implementación	03/11/2020	21/12/2020	49
PEC4 - Memoria y Presentación	22/12/2020	08/01/2021	18

Ilustración 4 - Diagrama de Gantt (Plan de trabajo)

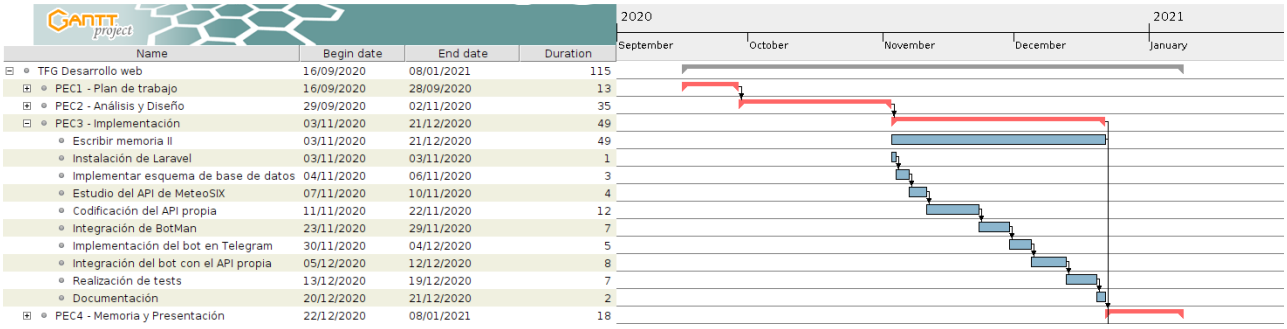
La segunda etapa incluiría el Análisis y el Diseño del proyecto:



Name	Begin date	End date	Duration
TFG Desarrollo web	16/09/2020	08/01/2021	115
PEC1 - Plan de trabajo	16/09/2020	28/09/2020	13
PEC2 - Análisis y Diseño	29/09/2020	02/11/2020	35
Escribir memoria I	29/09/2020	02/11/2020	35
Preparar entorno de trabajo	29/09/2020	30/09/2020	2
Diseño conceptual	01/10/2020	04/10/2020	4
Análisis de requisitos	05/10/2020	07/10/2020	3
Diagramas UML	08/10/2020	17/10/2020	10
Diseño de base de datos	18/10/2020	23/10/2020	6
Diseño de mockups para el frontend	24/10/2020	27/10/2020	4
Diseño de mockups para el backend	28/10/2020	31/10/2020	4
Documentación	01/11/2020	02/11/2020	2
PEC3 - Implementación	03/11/2020	21/12/2020	49
PEC4 - Memoria y Presentación	22/12/2020	08/01/2021	18

Ilustración 5 - Diagrama de Gantt (Análisis y diseño)

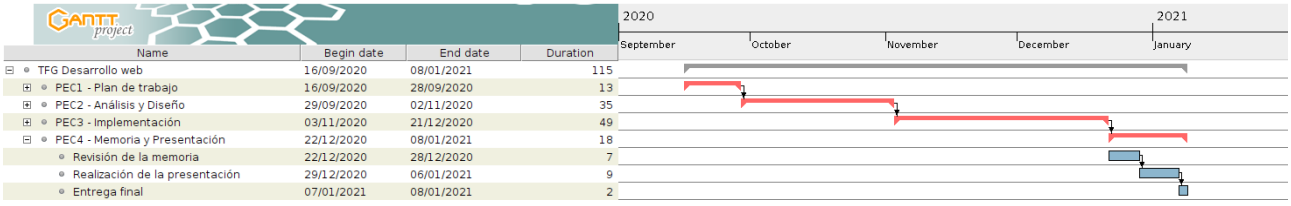
En la tercera etapa nos hemos centrado en la Implementación propiamente dicha:



Name	Begin date	End date	Duration
TFG Desarrollo web	16/09/2020	08/01/2021	115
PEC1 - Plan de trabajo	16/09/2020	28/09/2020	13
PEC2 - Análisis y Diseño	29/09/2020	02/11/2020	35
PEC3 - Implementación	03/11/2020	21/12/2020	49
Escribir memoria II	03/11/2020	21/12/2020	49
Instalación de Laravel	03/11/2020	03/11/2020	1
Implementar esquema de base de datos	04/11/2020	06/11/2020	3
Estudio del API de MeteoSIX	07/11/2020	10/11/2020	4
Codificación del API propia	11/11/2020	22/11/2020	12
Integración de BotMan	23/11/2020	29/11/2020	7
Implementación del bot en Telegram	30/11/2020	04/12/2020	5
Integración del bot con el API propia	05/12/2020	12/12/2020	8
Realización de tests	13/12/2020	19/12/2020	7
Documentación	20/12/2020	21/12/2020	2
PEC4 - Memoria y Presentación	22/12/2020	08/01/2021	18

Ilustración 6 - Diagrama de Gantt (Implementación)

La última de las etapas sería en la que se completa la Memoria y se lleva a cabo la Presentación:



Name	Begin date	End date	Duration
TFG Desarrollo web	16/09/2020	08/01/2021	115
PEC1 - Plan de trabajo	16/09/2020	28/09/2020	13
PEC2 - Análisis y Diseño	29/09/2020	02/11/2020	35
PEC3 - Implementación	03/11/2020	21/12/2020	49
PEC4 - Memoria y Presentación	22/12/2020	08/01/2021	18
Revisión de la memoria	22/12/2020	28/12/2020	7
Realización de la presentación	29/12/2020	06/01/2021	9
Entrega final	07/01/2021	08/01/2021	2

Ilustración 7 - Diagrama de Gantt (Memoria y Presentación)

1.5. Breve sumario de productos obtenidos

Se ha implementado un bot que obtiene información meteorológica de una API y que a su vez está integrado en la plataforma de bots de Telegram, donde los usuarios pueden interaccionar con él.

También se ha creado una base de datos MySQL, la cual es utilizada por el bot para el almacenamiento de las distintas alertas que configuran los usuarios.

Por último, se ha obtenido un documento formal de especificaciones sobre el diseño y la implementación de todo el proyecto. Se trataría de esta Memoria.

1.6. Breve descripción de los otros capítulos de la memoria

A continuación se realizará una breve descripción de los contenidos abordados en cada uno de los capítulos de los que se compone la Memoria, con el objetivo de facilitar búsquedas concretas de información.

- **Tecnologías utilizadas**

  En este capítulo se hará un recorrido por todas las tecnologías utilizadas tanto en las fases de diseño como en las fases de implementación del proyecto.

- **Requisitos del proyecto**

  Este capítulo repasará los diferentes requisitos del proyecto.

- **Diseño del proyecto**

  Podremos ver el avance en el diseño tanto de la aplicación como de la base de datos que acompaña a la misma.

- **Desarrollo e implementación del proyecto**

  Capítulo donde se describirán los procesos de implementación seguidos para llevar a buen puerto el desarrollo del bot.

- **Flujo de ejecución de la aplicación**

  Se realiza un recorrido por una ejecución real de la aplicación y sus distintas funcionalidades.

- **Conclusiones**

  Finalmente, en este capítulo recogeremos todas la conclusiones obtenidas en la elaboración del trabajo así como las posibilidades futuras del mismo.

2. Tecnologías utilizadas

A continuación se va a hacer un pequeño recorrido por todas las tecnologías utilizadas para el diseño e implementación del proyecto. No se profundizará en exceso en ellas debido a que esto se llevará a cabo en los siguientes apartados.

En las fases iniciales de diseño, se ha utilizado el software generador de diagramas **Draw.io**, con la cual hemos diseñado todos los diagramas UML así como el diagrama entidad-relación del diseño conceptual de la base de datos.

Una vez disponibles los diseños, hemos desplegado un *stack* LAMP (Linux, Apache, **MySQL**, **PHP**) en un servidor virtual del proveedor **DigitalOcean**, en el cual hemos instalado un certificado SSL de **Let's Encrypt** para cifrar el tráfico entrante y saliente del servidor.

Como herramienta de gestión de bases de datos hemos utilizado **DBeaver** y como IDE se ha elegido **PhpStorm** al estar la aplicación principalmente codificada en PHP, tal y como se puede apreciar en la siguiente captura del repositorio del proyecto en GitLab:

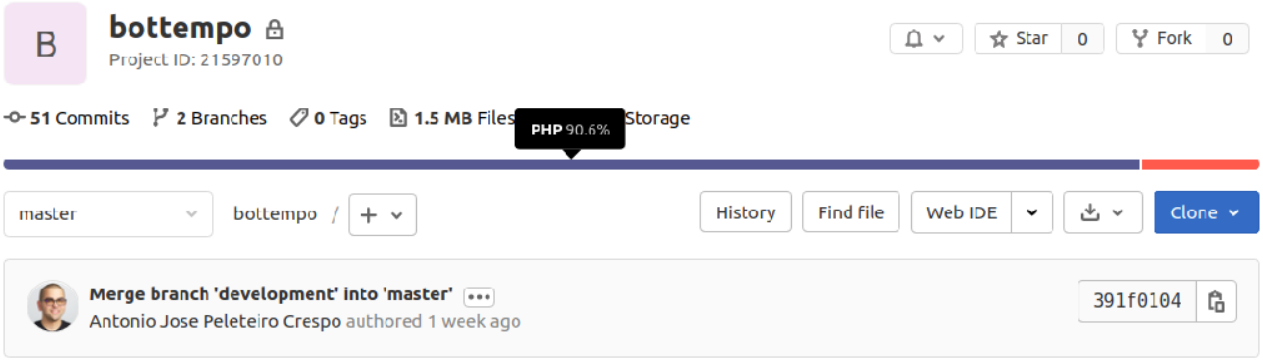


Ilustración 8 - Lenguajes utilizados en el repositorio de GitLab

Como se deduce de la imagen anterior, el sistema de control de versiones utilizado para el desarrollo ha sido **Git**, y en concreto se ha optado por **GitLab** como la plataforma sobre la que alojar el repositorio del proyecto.

Los *frameworks* PHP sobre los que se ha realizado la implementación son **Laravel** para el esqueleto de la misma y poder hacer uso de su ORM y **BotMan** para el desarrollo concreto de las funcionalidades del bot.

Por otro lado, se ha realizado una incursión en el campo del procesamiento del lenguaje natural (NLP en inglés), para lo que nos hemos apoyado en la herramienta de Google **Dialogflow**.

Por último, la integración final de todos los elementos anteriores se ha realizado en la plataforma de bots de **Telegram**, lo que ha permitido que el bot sea plenamente funcional.

Cabe señalar que también se ha registrado un dominio, **bottempo.gal**, para hacer visible en la web el bot y, además, ser el encargado de recibir las peticiones de Telegram ya que bajo dicho dominio es donde se encuentra desplegada la aplicación.

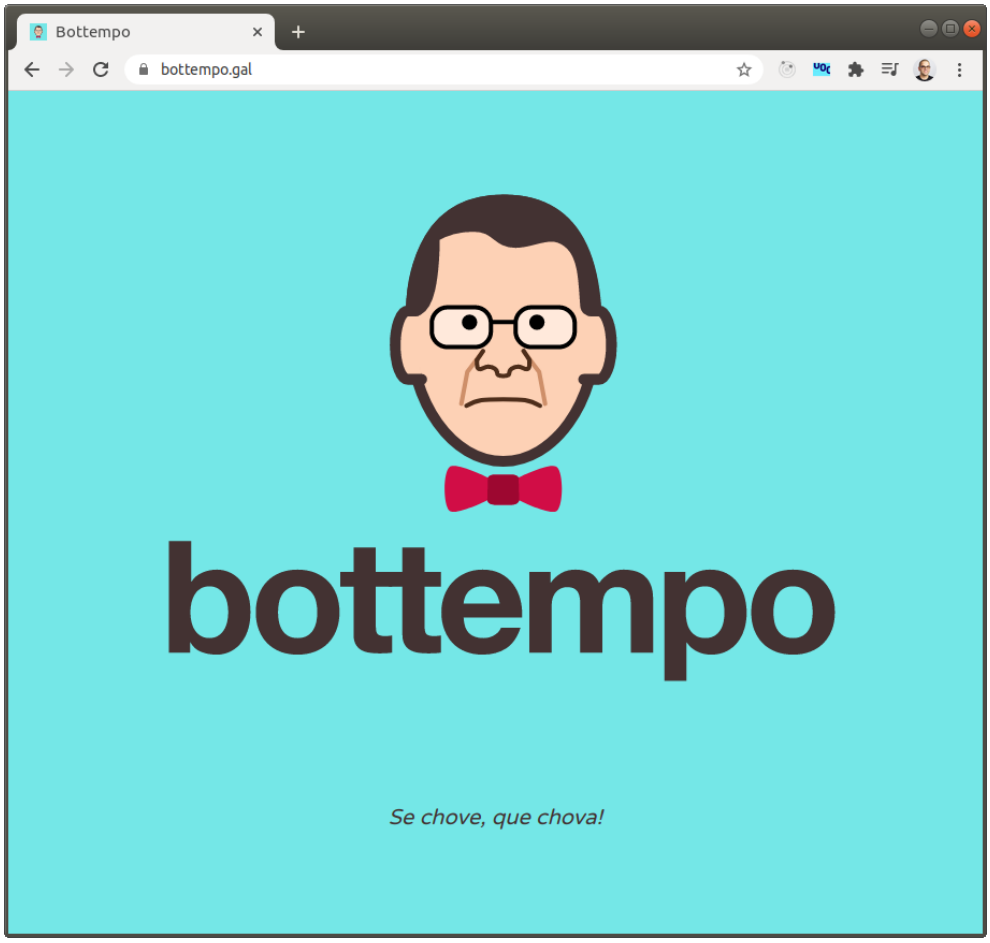


Ilustración 9 - Página web de Bottempo

3. Requisitos del proyecto

Para la realización de la aplicación se opta por utilizar el *framework* de PHP Laravel en su última versión, por lo que la plataforma en la que se va a realizar el despliegue debe contar con los siguientes **requisitos técnicos**:

- PHP >= 7.1.3
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- Ctype PHP Extension
- JSON PHP Extension
- BCMath PHP Extension

También teniendo en cuenta que la gestión de las alertas se realizará mediante el *framework* para desarrollo de bots BotMan, el cual se integra en Laravel, los requisitos técnicos anteriores son aplicables también a este.

Además, otros requisitos funcionales que debe cumplir nuestra aplicación son:

- El bot debe ser **adaptativo**, por lo que debe funcionar con cualquier versión de Telegram que se utilice, bien sea en dispositivo móvil, escritorio o versión web.

- Deben de tenerse en consideración los requisitos de **seguridad**, por lo que optaremos por cifrar las comunicaciones con un certificado SSL así como mantener el servidor en el que se encuentra desplegado lo más *endurecido* posible.

- Los requisitos de **usabilidad** también son elevados, para lo que realizaremos la localización de la aplicación en 3 idiomas: español, gallego e inglés.

4. Diseño del proyecto

4.1. Diseño UML de la aplicación

Los primeros pasos en el diseño del proyecto han sido la elaboración de diferentes **diagramas UML**, con el objetivo de contextualizar y encaminar la implementación lo máximo posible.

El primero de los diagramas llevados a cabo es el **diagrama de clases UML**, el cual describe la estructura general del sistema:

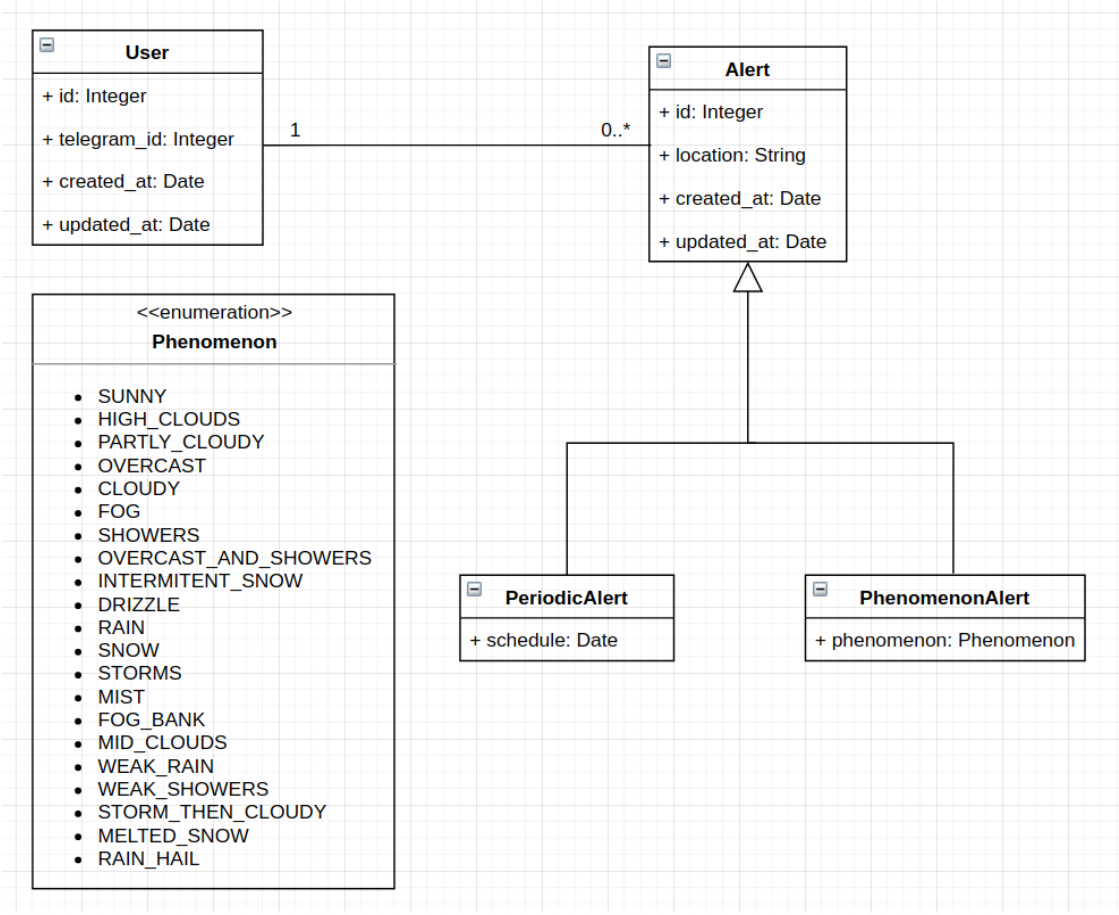


Ilustración 10 - Diagrama de clases UML

En el anterior diagrama, podemos observar como el sistema debería presentar, en primera instancia, al menos 4 clases, correspondientes con los usuario, las alertas, las alertas periódicas y las alertas por fenómenos. Además, apreciamos también las relaciones que se establecen entre ellas así como la multiplicidad, la cual nos indica que

un usuario puede tener múltiples alertas, pero que una alerta sólo puede pertenecer a un usuario.

Con respecto al tipo enumerado *Phenomenon*, correspondiente a los fenómenos meteorológicos disponibles, sus valores han sido recogidos de los posibles valores que el API MeteoSIX facilita.

Se incluye a continuación un **diagrama de casos de uso** que nos permite apreciar de un vistazo las distintas acciones realizables por los usuarios del sistema, donde podemos ver que el usuario puede consultar por el tiempo, así como programar, editar y eliminar alertas meteorológica tanto periódicas como debidas a diversos fenómenos:

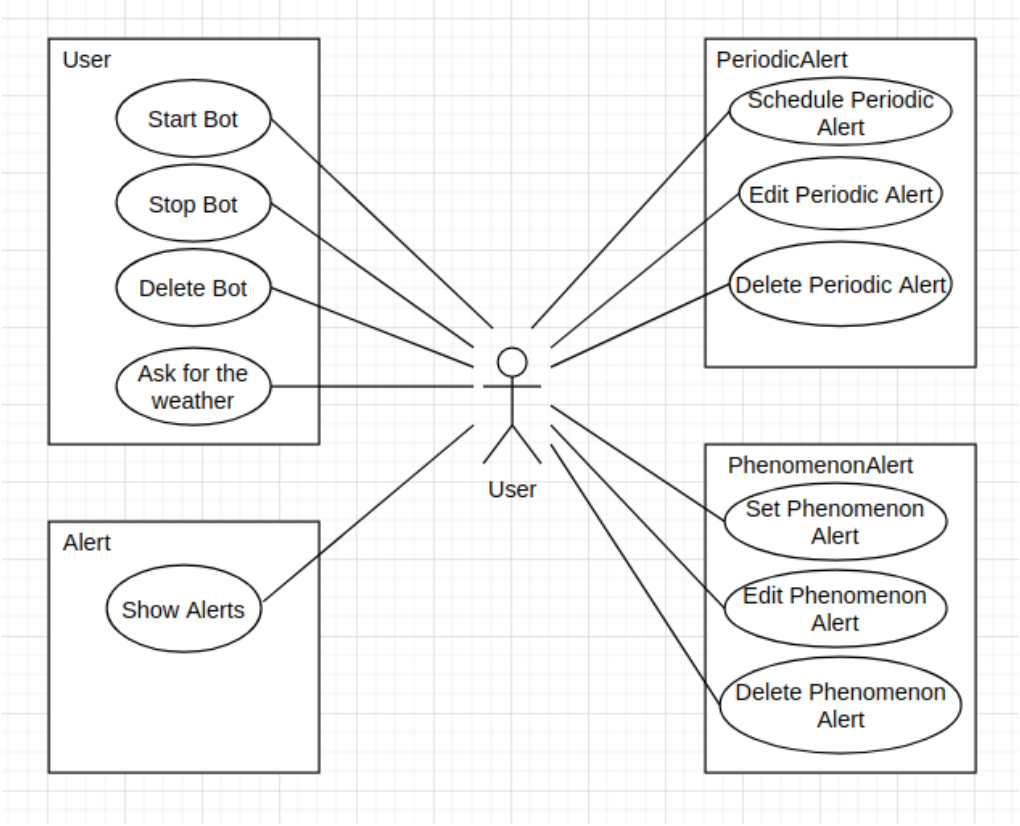


Ilustración 11 - Diagrama de casos de uso

Del mismo modo, pasamos a profundizar más en el uso del sistema mediante la elaboración de dos **diagramas de secuencia**, diagramas que nos permiten modelar la interacción entre los distintos objetos del sistema.

El primero de ellos es para la acción concreta de programar una alerta debida a un fenómeno atmosférico:

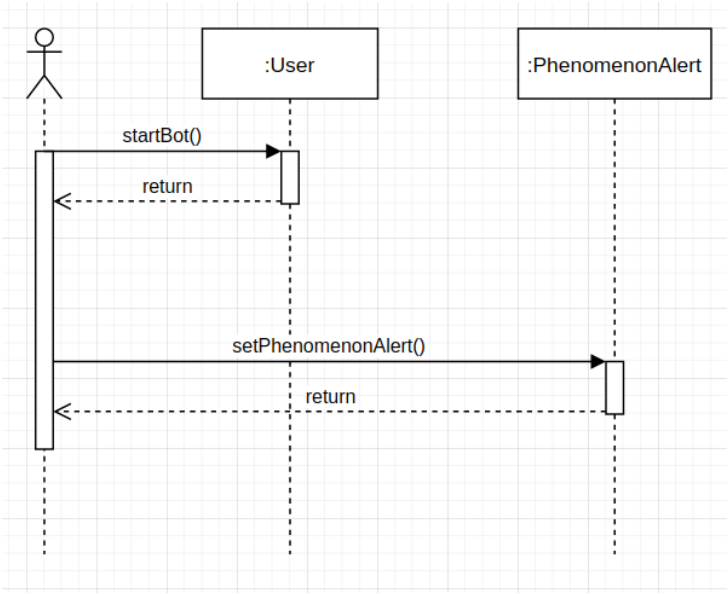


Ilustración 12 - Diagrama de secuencia (Programar alerta por fenómeno)

El segundo diagrama de secuencia se corresponde a la acción de eliminación de una alerta periódica previamente programada:

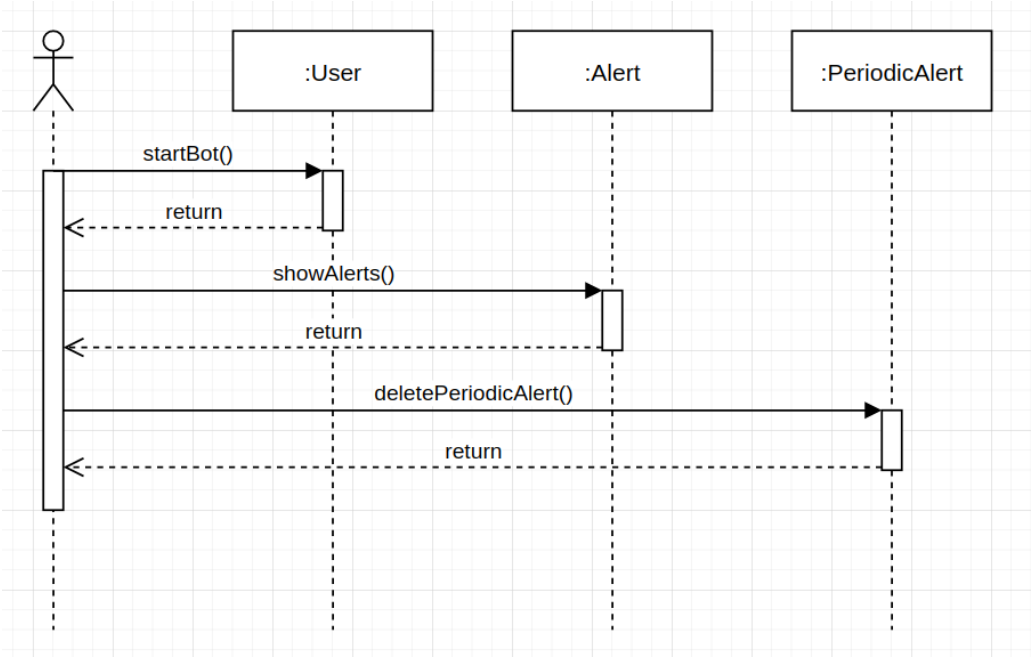


Ilustración 13 - Diagrama de secuencia (Eliminar alerta periódica)

4.2. Diseño de base de datos

En cuanto al **diseño de la base de datos**, comenzaremos como punto de partida con el **diseño conceptual**. Esta sería una etapa de análisis formal en la que vamos a intentar representar la realidad con un modelo semántico o descriptivo aproximado, para lo que vamos a utilizar herramientas conceptuales como el **diagrama entidad-relación**:

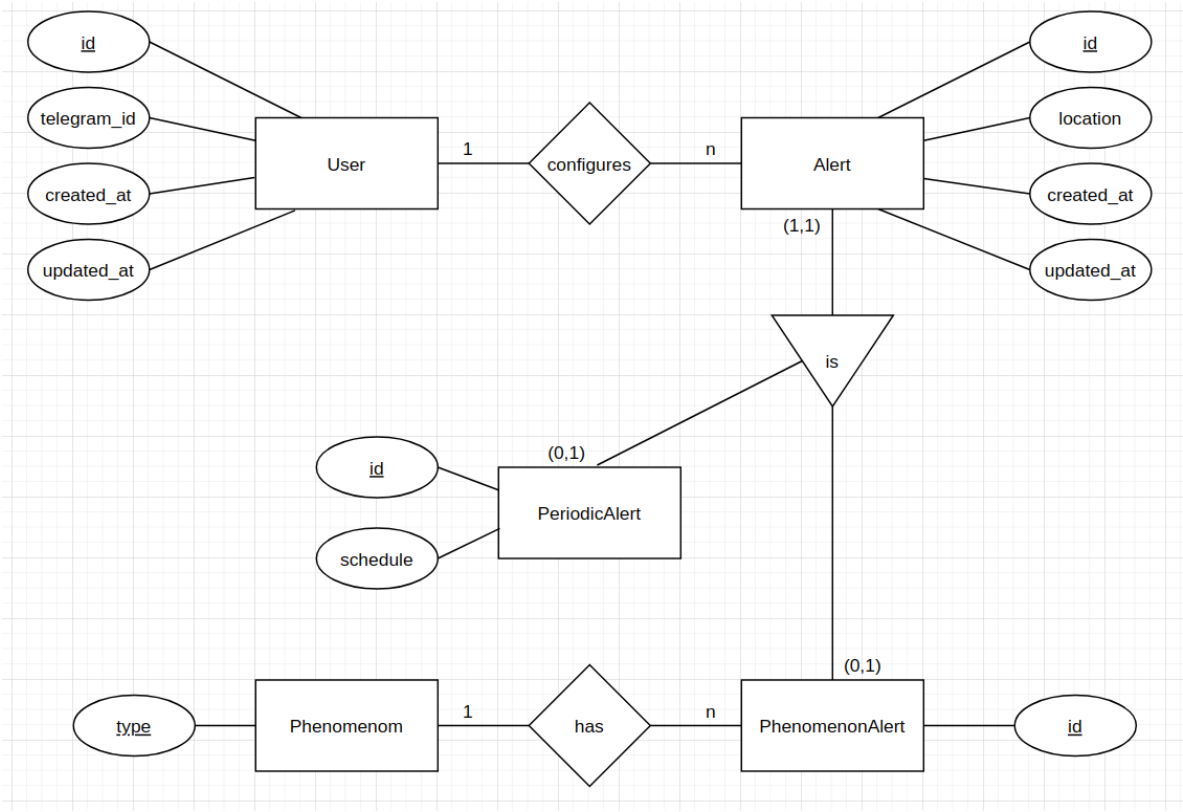


Ilustración 14 - Diagrama entidad-relación

Mediante este diseño conceptual, estamos expresando de manera normalizada unas necesidades de información, aunque la expresividad del diagrama entidad-relación es limitada, debido a lo que propondremos una solución al problema mediante el **diseño lógico** y la elaboración del correspondiente **esquema lógico** mediante el modelo relacional:

USER (id, telegram_id, created_at, updated_at)
Primary Key: id

ALERT (id, location, created_at, updated_at, user_id, type)
Primary Key: id
Foreign Key: user_id

PERIODIC_ALERT (id, schedule, alert_id)
Primary Key: id
Foreign Key: alert_id

PHENOMENON_ALERT (id, phenomenon, alert_id)
Primary Key: id
Foreign Key: phenomenon
Foreign Key: alert_id

PHENOMENON (type)
Primary Key: type

Ilustración 15 - Esquema lógico

Gracias al anterior diseño lógico, obtenemos una descripción usando el modelo relacional que nos permite obtener las relaciones que concuerdan con la semántica del problema a resolver.

Por último, aplicando los distintos procesos de **normalización**, así como teniendo en cuenta la operativa interna del ORM Eloquent, presente en Laravel, obtenemos el siguiente **esquema físico** para la base de datos:

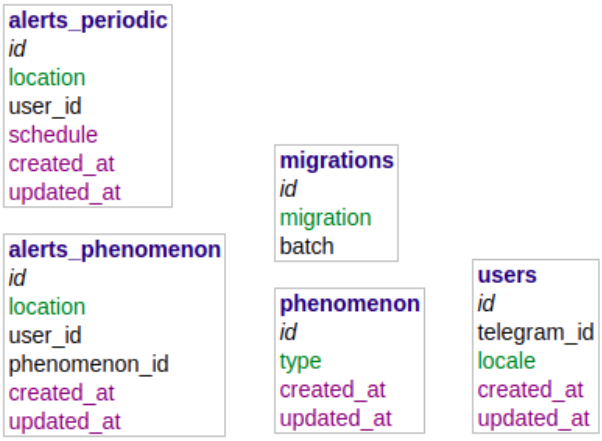


Ilustración 16 - Esquema físico

5. Desarrollo e implementación del proyecto

Para el desarrollo de la aplicación se ha optado como base por el *framework* de código abierto **Laravel**, el cual permite el uso de una sintaxis elegante y expresiva y que además permite multitud de funcionalidades adicionales fácilmente incorporables, por ejemplo a través del gestor de paquetes **Composer**.

5.1. Patrón Modelo Vista Controlador

Laravel [1] permite una aplicación casi inmediata del patrón **MVC (Modelo Vista Controlador)**, el cual permite separar los datos y la lógica del negocio (Modelo) de su representación (Vista) y de la gestión de los eventos y las comunicaciones (Controlador).

MVC se basa en las ideas de reutilización de código y en la separación de conceptos, principios que buscan facilitar la tarea del desarrollo, escalabilidad y mantenimiento de aplicaciones [2].

En este patrón, la Vista equivaldría a la interfaz gráfica a través de la cual el usuario visualiza el contenido y con la que interactúa, por ejemplo enviando una petición, la cual es recibida por el Controlador, que se encarga de transformar los datos del Modelo y devolverlos a la Vista para que el usuario obtenga la respuesta esperada a su petición.

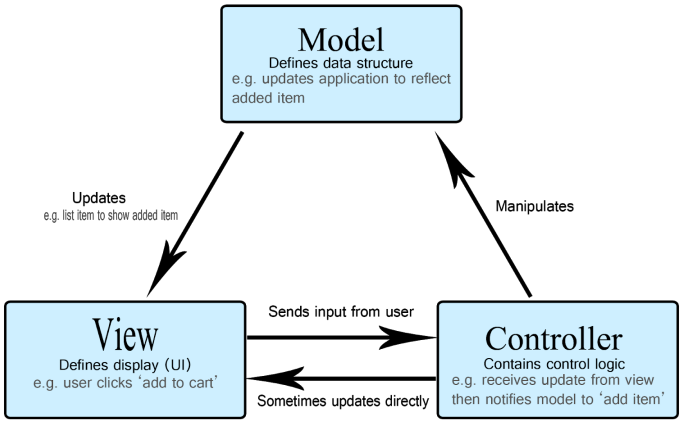


Ilustración 17 - Patrón Modelo Vista Controlador. Fuente: MDN Web Docs

En las aplicaciones web, las peticiones son recibidas a través de una URL, lo que comúnmente es denominado **ruta**. Esta ruta generalmente ejecuta una acción concreta de un controlador, lo que permitiría seguir el flujo del gráfico anterior y esta acción es conocida como **enrutado**.

Es conveniente señalar que la implementación que hace Laravel del patrón MVC es un tanto peculiar, ya que usa un sistema de rutas denominado "*Routes with Closures*", cuyo objetivo es hacer el código lo más legible posible y que consiste en responder a una petición a una ruta mediante la ejecución de una función conocida como *closure*:

```
use Illuminate\Support\Facades\Route;

Route::get('/greeting', function () {
    return 'Hello World';
});
```

Ilustración 18 - Enrutado en Laravel I. Fuente: Laravel

La *closure* puede estar contenida en un controlador para la correcta implementación del patrón MVC, tal y como puede apreciarse en el siguiente ejemplo:

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

Ilustración 19 - Enrutado en Laravel II. Fuente: Laravel

Sobre Laravel, se ha añadido el componente **BotMan** [3], el cual es un nuevo *framework* de código abierto que se apoya en Laravel para su funcionamiento y que está enfocado al desarrollo de aplicaciones de chat multiplataforma, con soporte para las plataformas más populares (Telegram, Slack, Facebook Messenger...) y que implementa el mismo sistema de "*Routes with Closures*" de Laravel aunque a su propia manera.

Así, en lugar de aceptar una petición HTTP al uso, lo que hace BotMan es "escuchar" una serie de comandos, mensajes o acciones desde la aplicación de chat en la que se esté interactuando con el bot.

Estos comandos esperados serían las rutas de una aplicación web convencional, y, si alguno de ellos es lanzado, se procede a ejecutar la función (*closure*) correspondiente:

```
$botman->hears(StartConversation::COMMAND, function ($bot) {
    Log::debug( message: 'Debug::Command: ' . StartConversation::COMMAND);
    $bot->startConversation(new StartConversation($bot));
})->stopsConversation();

$botman->hears(AlertsConversation::COMMAND, function ($bot) {
    Log::debug( message: 'Debug::Command: ' . AlertsConversation::COMMAND);
    $bot->startConversation(new AlertsConversation($bot));
})->stopsConversation();

$botman->hears(HelpConversation::COMMAND, function ($bot) {
    Log::debug( message: 'Debug::Command: ' . HelpConversation::COMMAND);
    $bot->startConversation(new HelpConversation($bot));
})->stopsConversation();

$botman->hears(NewAlertConversation::COMMAND, function ($bot) {
    Log::debug( message: 'Debug::Command: ' . NewAlertConversation::COMMAND);
    $bot->startConversation(new NewAlertConversation($bot));
})->stopsConversation();
```

Ilustración 20 - Comandos escuchados por BotMan

Por otro lado, BotMan reconvierte el Controlador convencional del patrón MVC en un sistema de conversaciones, donde una Conversación [4] pasaría a ser su equivalente. Las conversaciones disponen de un método *run()* que sería el que iniciaría la conversación ejecutada cuando el bot escuche alguno de los comandos esperados:

```
/**
 * Start the conversation.
 *
 * @return mixed|void
 */
public function run()
{
    $translation = __( key: 'messages.alertsType', [], $this->locale);
    $question = Question::create($translation)
        ->addButtons([
            Button::create(trans_choice( key: 'messages.periodicAlert',  number: 2, [], $this->locale))
                ->value( value: PeriodicAlert::VALUE),
            Button::create(trans_choice( key: 'messages.phenomenonAlert',  number: 2, [], $this->locale))
                ->value( value: PhenomenonAlert::VALUE)

        ]);

    $this->ask($question, function ($answer) {
        if ($answer->isInteractiveMessageReply()) {
            $alerts = $this->getAlertsService->getAlerts($answer->getValue(), $this->bot->getMessage()->getSender());
            $this->listAlerts($answer, $alerts);
        } else {
            $this->repeat();
        }
    });
}
```

Ilustración 21 - Conversación en BotMan

Para la realización de muchas de las acciones, se ha optado por la creación de ***Servicios***, con el objetivo de hacer la aplicación lo más desacoplada posible, permitiendo así que si es necesario realizar un cambio en alguno de ellos, solo haya que modificar dicho punto:

```php
<?php

namespace App\Services;

use App\Models\PeriodicAlert;
use App\Models\PhenomenonAlert;

class DeleteAlertService
{
    public function delete($answer)
    {
        list($alertType, $alertId) = explode( separator: '|', $answer);

        switch ($alertType) {
            case PeriodicAlert::VALUE:
                $alert = PeriodicAlert::find($alertId);
                $alert->delete();
                break;
            case PhenomenonAlert::VALUE:
                $alert = PhenomenonAlert::find($alertId);
                $alert->delete();
                break;
        }
    }
}
```

Ilustración 22 - Servicio en la aplicación

Para explicar el beneficio en cuanto a acoplamiento que la utilización de Servicios presenta, podemos volver nuevamente al patrón MVC. Si cuando el Controlador interactúa con el Modelo, lo hace a través de un Servicio, estamos encapsulando la lógica en dicho elemento, por lo que de cara a futuros cambios en la aplicación, solo habría que modificar dicho componente.

Así, si por ejemplo queremos pasar de una base de datos MySQL a una MongoDB por cualquier motivo, y el acceso a datos lo realizamos a través de un Servicio, solamente sería necesario actualizar el código de dicho servicio, el resto de la aplicación seguiría siendo válida.

5.2. *Workflow*, implementación y despliegue

Conviene señalar que el proyecto se ha desarrollado utilizando **Git** como sistema de control de versiones y **GitLab** como plataforma para alojar el repositorio con el código, el cual se encuentra públicamente disponible para su consulta en el siguiente enlace:

- **https://gitlab.com/apeleteiro/bottempo**

El uso de un sistema de control de versiones [5] presenta numerosas ventajas en los procesos de desarrollo, ya que permite implementar varias versiones con distintas funcionalidades en paralelo y añadir solo aquellas que finalmente consideremos, además de mantener un histórico de versiones por si es necesario volver atrás debido a cualquier motivo.

En concreto, Git se basa en un sistema de **ramas** o ***branches*** [6] en los que existe una rama principal que contiene las funcionalidades finales de la aplicación, y de la que partes otras ramas en las que se van implementando las nuevas necesidades que serán finalmente integradas en la rama principal en caso de que se aprueben.

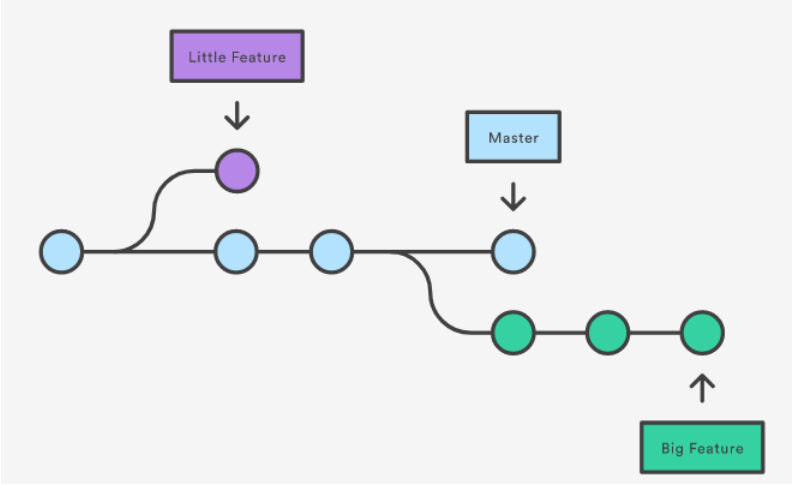


Ilustración 23 - Ramas en Git. Fuente: Atlassian

El flujo de trabajo o *workflow* que se ha seguido básicamente consistía en el mantenimiento de 2 ramas principales en Git, *development* y *master*. En la primera es donde se iban implementando las nuevas funcionalidades para, una vez listas, pasar a integrarse en la segunda, que es la que se encontraba desplegada en producción.

Por su parte, para la creación de tablas en la base de datos, y para el acceso y la interacción del código con con esos mismos datos, se ha optado por hacer uso del **ORM Eloquent**, el cual está incluido en Laravel.

Un ORM [7] es una técnica que permite establecer una relación entre los objetos de la aplicación y los datos de la base de datos, lo que simplifica las tareas de acceso a los datos facilitando el mapeo de datos y la creación de modelos, además de disponer de una sintaxis realmente expresiva que mejora enormemente la lectura del código.

Una vez completado el desarrollo, la aplicación se ha desplegado en servidor virtual privado del proveedor **DigitalOcean**, donde se ha instalado un *stack* LAMP (Linux, Apache, MySQL y PHP) junto con un certificado SSL de **Let's Encrypt** para asegurar las comunicaciones entre el servidor y la aplicación.

La aplicación es accesible a través del dominio **bottempo.gal**, el cual se ha registrado para hacer visible la propia aplicación, y desde dicha web, pinchando en la imagen, automáticamente lleva a la URL propia del bot en Telegram: **https://t.me/BottempoBot**

Dicha URL es configurada una vez se lleva a cabo el registro del bot en el sistema de **Telegram**, lo cual proporciona un clave única o *token* necesario para la conexión de la plataforma desplegada en la instancia de DigitalOcean con Telegram y poder así hacer funcional el bot.

Para la API meteorológica se ha optado por **MeteoSIX**, mantenida por la *Consellería de Medio Ambiente* de la *Xunta de Galicia* y que permite obtener predicciones meteorológicas de cualquier punto de la geografía gallega.

5.3. Procesamiento de lenguaje natural

Finalmente, se ha realizado una pequeña aproximación al **procesado de lenguaje natural** (PLN o NLP por sus siglas en inglés) [8], para lo cual se ha optado por la herramienta **Dialogflow**, propiedad de Google, la cual permite que el bot pueda responder en base a patrones lingüísticos sin tener que hacer una coincidencia o *match* exacto con el mensaje esperado.

El procesado del lenguaje natural es un campo de la Inteligencia Artificial encargado de investigar la manera de que las máquinas puedan comunicarse con las personas

mediante el uso de lenguas naturales, como son el español, el inglés o el gallego, idiomas empleados en la implementación de la aplicación.

En concreto, se ha optado por Dialogflow debido a la facilidad para integrar dicha plataforma en distintas aplicaciones como dispositivos móviles, aplicaciones web, bots, etc. y el análisis del lenguaje que realiza de manera automática [9].

Para ello, simplemente hay que crear lo que Dialogflow denomina “Intenciones” o ***Intents***. Una *Intent* sería un patrón lingüístico que un usuario podría escribir, en el cual se buscan varios patrones y se definen variables, que posteriormente son tratadas por la aplicación y reaccionan en base al comportamiento esperado por las mismas [10].

A continuación se puede apreciar la creación del *Intent Weather* en Dialogflow, donde se observa la búsqueda activa del lugar para el que se realizará la consulta meteorológica en varios idiomas:

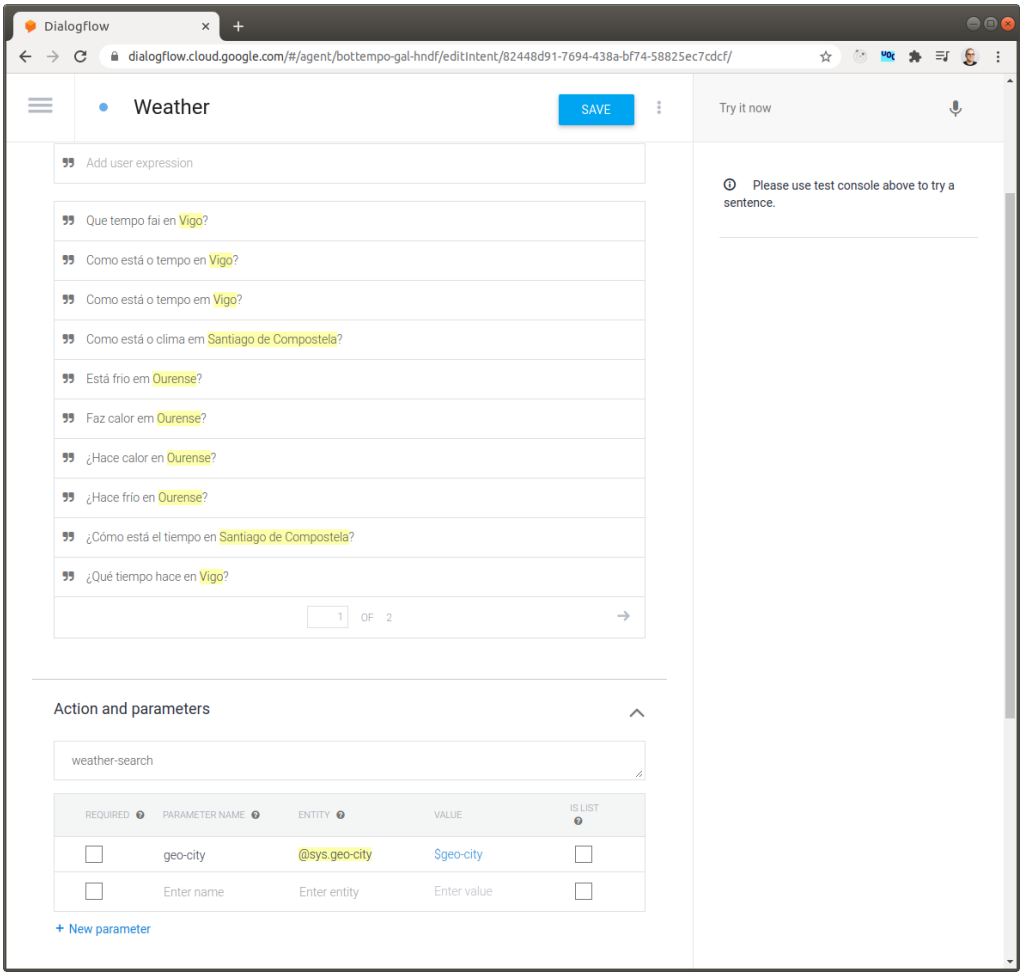


Ilustración 24 - Uso de Dialogflow

6. Pruebas y flujo de ejecución de la aplicación

En el siguiente apartado procedo a realizar una batería completa de pruebas sobre la aplicación, mediante una demostración práctica de ejecución del bot y de las distintas funcionalidades que se encuentran desplegadas en producción en el proyecto y que pueden ser probadas públicamente.

En un primer momento, es necesario entablar una conversación con el bot, para lo cual es necesario buscar al usuario @BottempoBot en Telegram, o bien acceder a través de la URL propia del bot **https://t.me/BottempoBot** o incluso desde la URL pública **https://bottempo.gal/**

Una vez se inicia la conversación, el bot nos pregunta por el idioma en el que queremos interactuar, ya que se ha llevado a cabo una labor de localización del bot y se ha realizado la implementación en 3 idiomas, español, gallego e inglés:

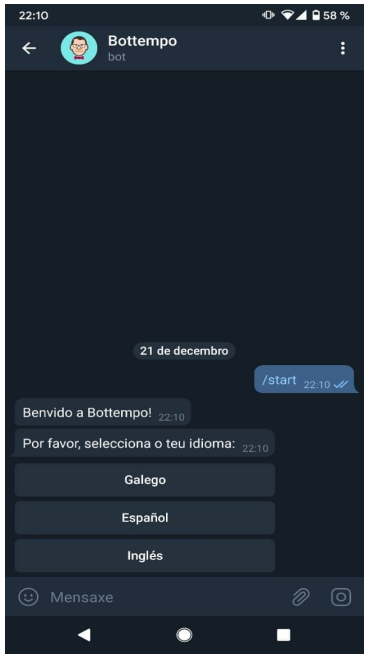


Ilustración 25 - Inicio del bot

Acto seguido, el bot nos confirma la selección y pasa a ejecutarse en dicho idioma, mostrando a continuación información sobre diversas funcionalidades que podemos realizar, en concreto preguntar por el tiempo que hace en cualquier lugar de Galicia así como la posibilidad de crear alertas periódicas o por fenómeno:

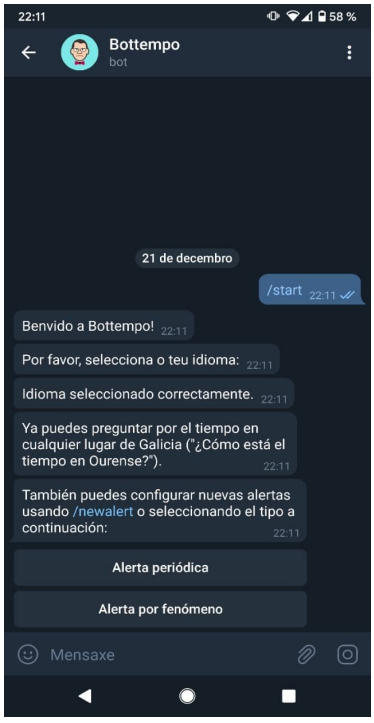


Ilustración 26 - Idioma seleccionado

Si pulsamos en el botón de creación de **Alerta periódica**, e introducimos los datos para los que queramos crear la alerta (Santiago y 22:15 en el ejemplo), la alerta se crea correctamente:

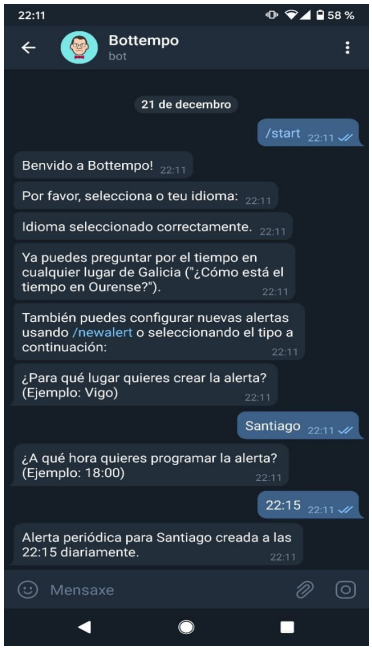


Ilustración 27 - Creación de alerta periódica

Así mismo, el bot presenta un comando de ayuda mediante la orden "***/help***", la cual nos informa nuevamente de las operaciones disponibles y que podemos llevar a cabo: gestionar nuestras alertas mediante el comando "***/alerts***" o bien crear nuevas alertas mediante "***/newalert***":

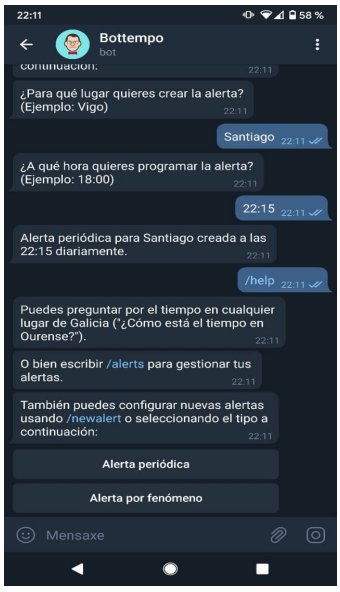


Ilustración 28 - Ayuda del bot

Si ahora lanzamos el comando "***/alerts***", tal y como nos informaba el bot en el comando de ayuda, podremos gestionar nuestras alertas, permitiéndonos en un primer momento seleccionar qué alertas queremos listar para gestionar, si las alertas periódicas o las alertas por fenómeno:

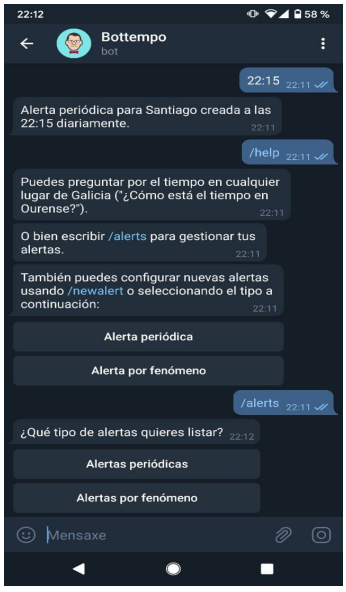


Ilustración 29 - Listado de alertas

Una vez seleccionado el tipo de alerta, estas pasarán a listarse y podremos gestionar las alertas elegidas, disponiendo de la opción de Eliminar aquellas que consideremos oportunas:

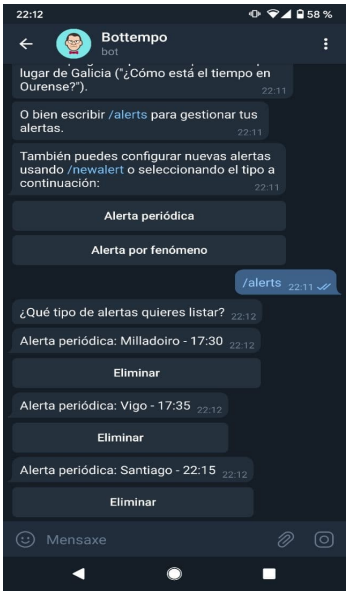


Ilustración 30 - Listado de alertas para eliminar

En caso de que no tengamos ninguna alerta de algún tipo configurada, y tratemos de listar dicho tipo, se nos mostrará un mensaje informándonos al respecto:

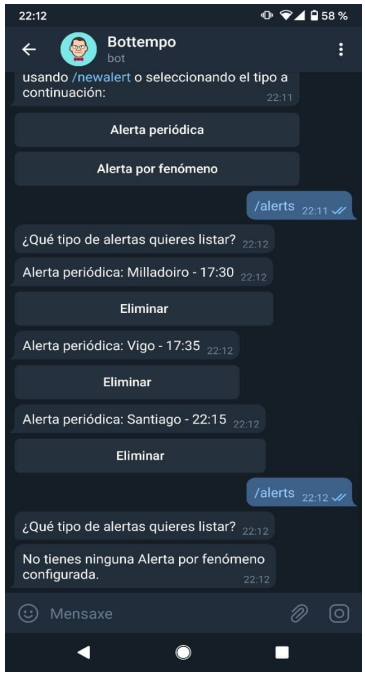


Ilustración 31 - Listado sin alertas

El otro comando del que nos informaba la ayuda, es el de creación de alertas "***/newalert***" el cual, como se intuye por su nombre, nos permite crear nuevas alertas de alguno de los dos tipos: periódicas o por fenómenos atmosféricos:

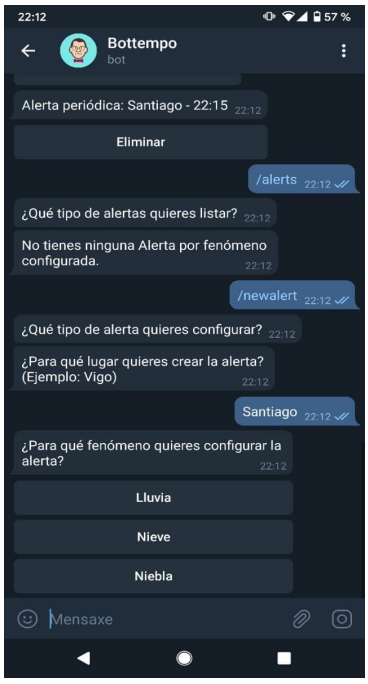


Ilustración 32 - Creación de alerta por comando

En cualquier momento durante la ejecución del bot podemos consultar el estado del tiempo en cualquier punto de Galicia en cualquiera de los idiomas disponibles, ya que como se ha indicado anteriormente, se ha optado por la inclusión del sistema de **procesamiento de lenguaje natural** que proporciona **Dialogflow**.

A continuación, podemos apreciar que independientemente del idioma elegido para realizar la pregunta (español, gallego o inglés), así como incluso si incorpora diversas variaciones en la pregunta, el bot es capaz de devolvernos el resultado de la consulta meteorológica para el lugar sobre el que le estamos preguntando:

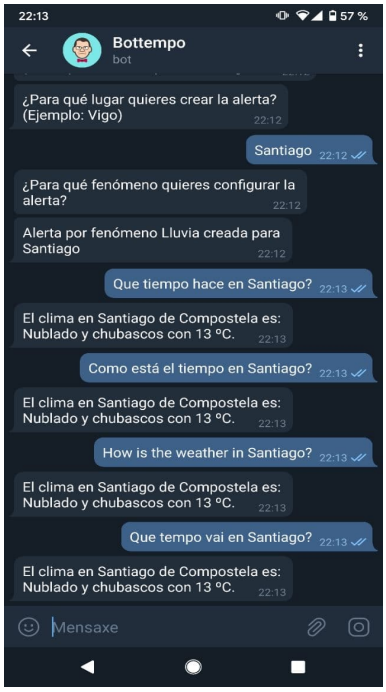


Ilustración 33 - Procesamiento de lenguaje natural

El **control de errores** también ha sido tenido en cuenta. Así, por ejemplo, si introducimos una orden no existente, como por ejemplo "*Comando no existente*", el bot nos informa de que la instrucción no es correcta y nos insta a usar la ayuda:

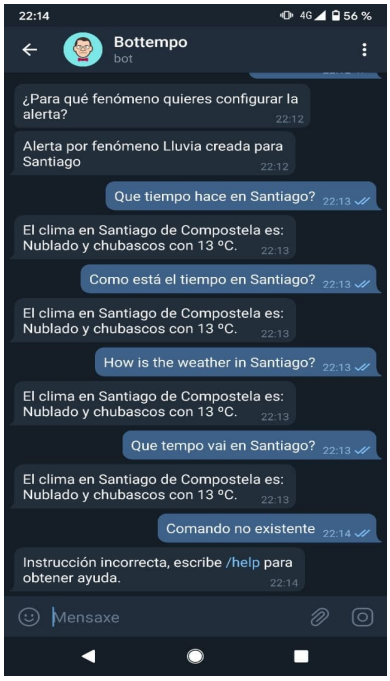


Ilustración 34 - Comando no existente

Por su parte, si intentamos crear una alerta para un lugar no válido, el bot también nos informa de esta situación y nos vuelve a preguntar por el lugar para el que queremos crear la alerta:

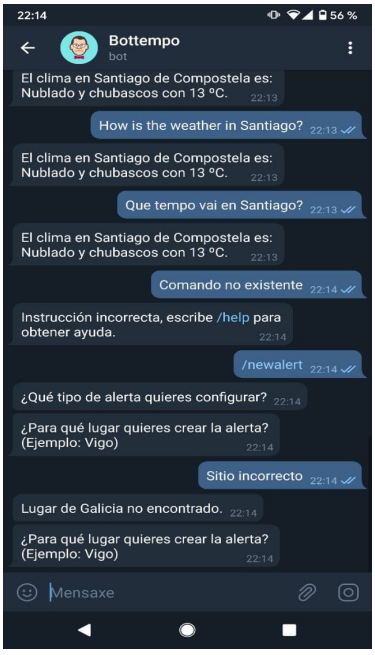


Ilustración 35 - Lugar no encontrado

Continuando con el control de errores, si tratamos de lanzar un comando no válido, por ejemplo al eliminar una alerta, el bot nos instará a repetir la acción hasta que la ejecución sea correcta:

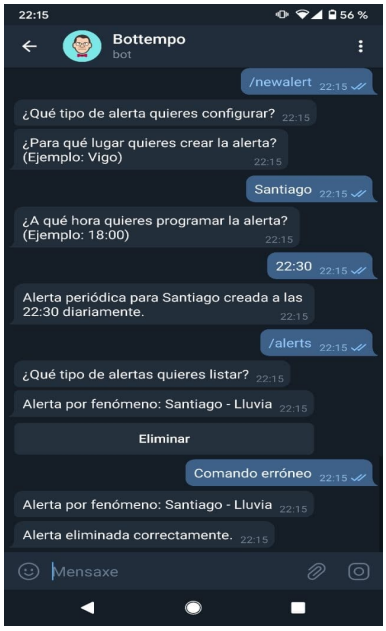


Ilustración 36 - Orden no válida

Por último, una vez que tengamos configuradas diversas alertas, el bot nos notificará a este respecto. Por ejemplo, si se trata de una alerta periódica, nos enviará un mensaje a la hora que hayamos indicado en la alerta:

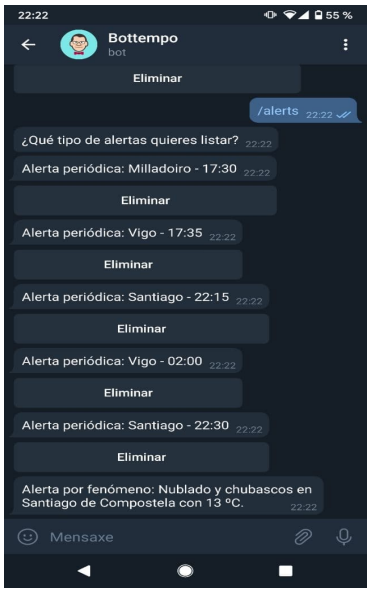


Ilustración 37 - Notificación de alerta periódica

En cambio, si hemos configurado una alerta por un fenómeno atmosférico, por ejemplo lluvia, nos enviará una notificación en cuanto detecte dicho fenómeno en el lugar que hayamos configurado:

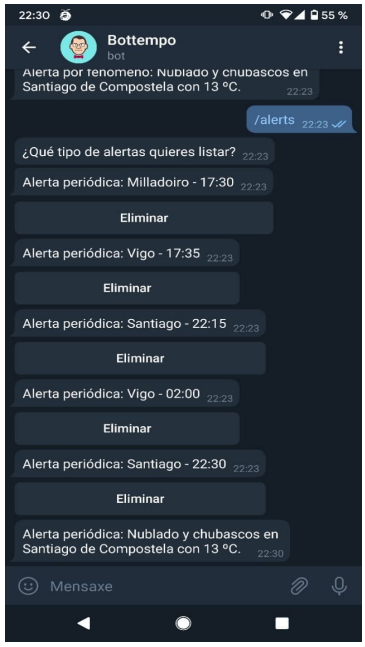


Ilustración 38 - Notificación de alerta por fenómeno

7. Conclusiones

A continuación se procede a detallar las principales conclusiones que se han obtenido durante todo este largo proceso en el que ha consistido la asignatura de Trabajo Final de Grado, donde se han aplicado gran parte de los conocimientos adquiridos durante el plan de estudios.

Así, dividiremos las conclusiones en tres grandes bloques: consideraciones previas; diseño y desarrollo; y posibilidades futuras.

7.1. Consideraciones previas y objetivos

Cabe señalar que las expectativas que tenía depositadas en esta asignatura eran realmente altas, ya que el proceso completo incluía todos los pasos necesarios para el desarrollo de un proyecto completo, desde los pasos iniciales en los que se define la idea, pasando por las fases de diseño e implementación, hasta finalizar con la puesta en producción y la elaboración de los documentos técnicos finales.

Ante esto, he de reconocer que el objetivo principal que me había marcado ha sido cumplido con creces y todo aquello que quería ver reflejado en la aplicación, finalmente se encuentra implementado, pues salvo algunos recortes en el alcance del proyecto debido a la falta de tiempo (presencia de un sistema de *testing*, valorar despliegue automático) el resto se encuentra publicado en producción.

7.2. Diseño y desarrollo

Las fases de diseño e implementación es donde realmente se ha podido apreciar la aplicación de los conocimientos adquiridos en diferentes asignaturas del Grado.

Así, por ejemplo en el diseño de la base de datos se aplican la mayoría de conceptos de Uso de bases de datos junto con Diseño de bases de datos. Además, en el diseño conceptual también se han aplicado las nociones adquiridas en Ingeniería del software o en Diseño y programación orientada a objetos.

Por otro lado, la asignatura de Gestión de proyectos se aplica transversalmente durante todo el proyecto, lo que da una idea de la importancia de la misma en el plan de estudios.

Con todo esto mi conclusión es que este trabajo ha sido la guinda a un pastel que ha durado varios años y que finalmente da sentido y aplicación práctica a todo lo aprendido durante el proceso de aprendizaje.

7.3. Futuro

Las consideraciones futuras a tener en cuenta para la evolución de la aplicación se centran en varios aspectos: calidad del código, usabilidad y nuevas funcionalidades.

En lo que respecta a la calidad del código, va a ser necesario un proceso de inclusión de pruebas tanto funcionales (probar funcionalidades completas) como unitarias (probar acciones atómicas), ya que actualmente la aplicación carece de un sistema de *testing* automatizado.

La usabilidad podría ser potenciada mediante la incorporación de elementos gráficos que hagan visualmente más atractivo al bot, como por ejemplo acompañar las previsiones meteorológicas con *smileys* representativos de los propios fenómenos.

Por último, como principal funcionalidad a incorporar al bot, se podría valorar la posibilidad de que el bot pudiese ser añadido a otras conversaciones grupales y que mencionando al mismo, pudiese ser preguntado por la información meteorológica. También sería posible configurar el bot para capturar las comunicaciones del grupo, establecer un patrón, y si este es conveniente, informar de la previsión.

Este último paso podría llevarse a la práctica profundizando en el sistema de procesamiento de lenguaje natural incluido actualmente en la aplicación, incluyendo más funcionalidades en el mismo y un mayor nivel de aprendizaje automático.

8. Glosario

- **PHP:** Lenguaje de programación de uso general utilizado principalmente en desarrollo web.
- **Bot:** Programa informático que reacciona de manera autónoma a ciertos comandos que espera recibir.
- **Framework:** Estructura conceptual y tecnológica que presenta diversos artefactos o módulos concretos de software y que puede servir de base para el desarrollo de software.
- **API:** Conjunto de subrutinas, funciones y procedimientos que un software ofrece para ser utilizado por otro software.
- **MySQL:** Sistema de gestión de bases de datos relacional propiedad de Oracle Corporation.
- **UML:** Lenguaje unificado de modelado (UML, por sus siglas en inglés). Se trata de un lenguaje de modelado utilizado para visualizar, especificar, construir y documentar sistemas software.
- **Apache:** Servidor web que se instala en un servidor (físico o virtual) y permite servir aplicaciones web.
- **Certificado SSL:** Estándar de seguridad que permite la transferencia de datos cifrados entre un navegador y un servidor web como Apache.
- **IDE:** Entorno de desarrollo integrado (IDE, por sus siglas en inglés). Aplicación informática que proporciona todos los servicios necesarios para el desarrollo de software.
- **ORM:** Técnica que permite establecer una relación entre los objetos de una aplicación y los datos de una base de datos.
- **Endurecimiento:** Proceso de asegurar un sistema informático reduciendo sus vulnerabilidades o agujeros de seguridad.
- **Composer:** Sistema de gestión de paquetes para programar en PHP.
- **Acoplamiento:** Interdependencia entre componentes software.
- **Encapsulamiento:** En programación, ocultación intencionada del estado de un objeto, con el objetivo de que solo pueda ser modificado mediante los métodos disponibles para tal fin.

9. Bibliografía

- [1] Documentación oficial de Laravel: https://laravel.com/docs/8.x
- [2] Model View Controller: https://developer.mozilla.org/en-US/docs/Glossary/MVC
- [3] Documentación oficial de BotMan: https://botman.io/2.0/welcome
- [4] Marcel Pociot. Build A Chatbot: https://course.buildachatbot.io/
- [5] What is version control?: https://www.atlassian.com/git/tutorials/what-is-version-control
- [6] Using branches: https://www.atlassian.com/git/tutorials/using-branches
- [7] Object-relational Mappers (ORMs): https://www.fullstackpython.com/object-relational-mappers-orms.html
- [8] Procesamiento del lenguaje natural ¿qué es?: https://www.iic.uam.es/inteligencia/que-es-procesamiento-del-lenguaje-natural/
- [9] Ewerton da Costa Vaz. (2020). Chatbot with Botman Studio and Dialogflow: Step by step to build your virtual assistant (English Edition): https://amzn.to/34R8RwN
- [10] Documentación oficial de Dialogflow : https://cloud.google.com/dialogflow/docs