



**UNIVERSIDAD  
DE GRANADA**

**TRABAJO FIN DE GRADO**

**GRADO EN INGENIERÍA INFORMÁTICA**

# **Bot de Telegram para la gestión de bandas**

---

**Autor**

Daniel Haro Contreras

**Directora**

Rosana Montes Soldado



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE  
TELECOMUNICACIÓN**

—  
Granada, noviembre de 2022









# Bot de Telegram para la gestión de bandas

---

**Autor**

Daniel Haro Contreras

**Directora**

Rosana Montes Soldado



## **Bot de Telegram para la gestión de bandas: Subtítulo del proyecto**

Daniel Haro Contreras

**Palabras clave:** Telegram, bot, gestión, música, banda

### **Resumen**

Poner aquí el resumen.



**Project Title: Project Subtitle**

First name, Family name (student)

**Keywords:** Keyword1, Keyword2, Keyword3, ....

**Abstract**

Write here the abstract in English.



---

**Yo, Daniel Haro Contreras**, alumno de la titulación TITULACIÓN de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada, con DNI 76656133P, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Daniel Haro Contreras

Granada a X de mes de 201 .



---

D.<sup>a</sup> **Rosana Montes Soldado**, profesora del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

**Informa:**

Que el presente trabajo, titulado *Bot de Telegram para la gestión de bandas*, ha sido realizado bajo su supervisión por **Daniel Haro Contreras**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a X de noviembre de 2022.

**La directora:**

**Rosana Montes Soldado**



# **Agradecimientos**

Poner aquí agradecimientos...



# Índice general

<b>1. Introducción</b>	<b>19</b>
1.1. Motivación y justificación del proyecto . . . . .	19
1.2. Objetivos . . . . .	21
<b>2. Estado del arte</b>	<b>23</b>
2.1. Dominio del problema a resolver . . . . .	23
2.2. Aplicaciones similares . . . . .	25
2.2.1. Glissandoo . . . . .	25
2.2.2. Agregación de servicios no especializados . . . . .	27
<b>3. Mordente</b>	<b>29</b>
3.1. Desarrollo de Mordente . . . . .	29
3.1.1. Diseño Centrado en el Usuario . . . . .	29
3.1.2. Design Thinking . . . . .	30
3.2. Personas ficticias . . . . .	30
3.3. Malla receptora de información . . . . .	31
3.3.1. Aspectos interesantes o relevantes de la idea . . . . .	32
3.3.2. Críticas constructivas o posibilidades de mejora . . . . .	32
3.3.3. Preguntas nuevas a partir de la experiencia del usuario	33
3.3.4. Ideas que surgen . . . . .	34
3.4. Descripción de la propuesta . . . . .	34
3.4.1. Propuesta del producto . . . . .	34
3.4.2. Elección de metodología de desarrollo . . . . .	36
3.4.3. Planificación . . . . .	36
3.4.4. Presupuesto . . . . .	38
<b>4. Diseño de la interfaz de usuario</b>	<b>41</b>
4.0.1. Bocetos de interfaz . . . . .	42
<b>5. Diseño técnico</b>	<b>45</b>
5.1. Listado de historias de usuario (Backlog) . . . . .	45
5.2. Historias de usuario . . . . .	45
5.3. Metodologías y tecnologías de base que podrían usarse . . . . .	45
5.3.1. Lenguaje de programación . . . . .	45

5.3.2. Framework para el desarrollo del bot . . . . .	46
5.3.3. Base de datos . . . . .	47
5.3.4. Almacenamiento de archivos . . . . .	48
5.3.5. IDE . . . . .	48
5.3.6. Documentación . . . . .	48
5.4. Modelo de la base de datos . . . . .	49
5.5. Arquitectura de Mordente . . . . .	49
5.5.1. Arquitectura entre puntos . . . . .	49
5.5.2. Arquitectura entre servicios . . . . .	50
5.6. Seguridad . . . . .	52
<b>6. Implementación</b>	<b>53</b>
6.1. Metodología de trabajo . . . . .	53
6.1.1. Planificación temporal . . . . .	53
6.2. Presupuesto . . . . .	54
6.3. Herramientas y tecnologías escogidas . . . . .	54
6.3.1. Lenguaje de programación . . . . .	54
6.3.2. Framework . . . . .	54
6.3.3. Base de datos . . . . .	54
6.3.4. ORM . . . . .	54
6.3.5. IDE . . . . .	54
6.4. Primer sprint . . . . .	54
6.5. Segundo sprint . . . . .	54
<b>7. Pruebas</b>	<b>55</b>
7.1. Diseño de las pruebas . . . . .	55
7.1.1. Participantes . . . . .	55
7.1.2. Tareas a realizar . . . . .	55
7.1.3. Preguntas a realizar . . . . .	56
7.2. Realización de las pruebas . . . . .	56
7.3. Informe final de las pruebas . . . . .	56
7.4. Demostración en vídeo . . . . .	56
<b>8. Conclusiones y trabajos futuros</b>	<b>59</b>
8.1. Conclusiones . . . . .	59
8.2. Trabajos futuros . . . . .	59
8.2.1. Webapp . . . . .	59
8.2.2. Webapp integrada en el bot . . . . .	59
<b>Bibliografía</b>	<b>63</b>

# Índice de figuras

2.1. Capturas de la aplicación <i>Glissando</i> . . . . .	26
3.1. Persona: Eugenio Soto . . . . .	31
3.2. Persona: Clara Villena . . . . .	32
3.3. Persona: Eugenio Soto . . . . .	33
3.4. Persona: Inma Medina . . . . .	34
3.5. Diagrama de Gantt del proyecto . . . . .	40
4.1. Diseño de la interfaz de lista . . . . .	43
4.2. Diseño de la interfaz del detalle de un elemento . . . . .	44
4.3. Diseño de la interfaz de creación de un elemento . . . . .	44
5.1. Arquitectura entre puntos . . . . .	49
5.2. Arquitectura entre servicios . . . . .	50



# Índice de tablas

3.1. Planificación temporal . . . . .	38
3.2. Presupuesto . . . . .	39
5.1. Comparación de lenguajes de programación . . . . .	46
5.2. Comparación de frameworks para creación de bots de Telegram	47
7.1. Formulario para la prueba de usabilidad . . . . .	57



# Capítulo 1

## Introducción

Este proyecto intenta dar respuesta a las agrupaciones musicales que demandan una plataforma gratuita y libre para la mejora de su gestión.

### 1.1. Motivación y justificación del proyecto

En 2018 se contabilizaron unas 6197 agrupaciones musicales en España según el INE [1], número que se mantiene estable a lo largo de los años. Cada agrupación musical, definida como “dos o más personas que, a través de la voz o de instrumentos musicales, interpretan obras musicales pertenecientes a diferentes géneros y estilos” según la RAE, se compone de distintos miembros, formando una sociedad. Como en todas las sociedades, se necesita cierto orden para poder alcanzar los objetivos, por lo que algunos de los miembros se tienen que encargar de su gestión: el director musical, el director artístico, el presidente de la asociación, el secretario... Siendo la gestión distinta en cada agrupación.

Los miembros de las agrupaciones musicales realizan ensayos periódicamente con el doble objetivo de mejorar su técnica de interpretación y preparar la celebración de eventos, tales como conciertos, certámenes, concursos, pasacalles... En los ensayos y eventos se interpreta una o varias obras musicales.

El director musical de una agrupación musical se encarga de coordinar la interpretación de las obras musicales, así como planificar y preparar los ensayos y eventos. En ocasiones se sustituye o se complementa con el rol de director artístico, visibilizando el hecho de que los eventos a menudo no son puramente musicales, sino que se añaden teatros o espectáculos visuales como bailes o musicales.

Los administradores de la agrupación (habitualmente, junta directiva) se encargan de las labores de administración no musicales, ayudando también al director. Habitualmente entre estos encontramos un presidente, un secretario y un tesorero.

Para interpretar una obra musical, los miembros de la agrupación leen un papel. El papel puede ser de dos tipos: el que lee el director se denomina *partitura*, y es el papel que recoge la música de todos los instrumentos y voces. Por otra parte, cada músico lee su *particella* (también llamada “parte”), que solo recoge lo que debe interpretar él individualmente.

En los últimos tiempos la gestión de estas agrupaciones se ha realizado de forma manual, consistente en que los administradores, el director y los miembros se comunican mediante un grupo de mensajería sin ningún tipo de automatización.

De esta forma, algunas de las tareas más frecuentes en la administración de la agrupación podrían ser:

- Para la asistencia a eventos y ensayos, cada miembro escribe a través de un grupo, o individualmente al director o un administrador, su intención de asistir o no. Esta comunicación previa le es útil para gestionar qué repertorio es mejor ensayar, en qué orden, o si por el contrario es conveniente aplazar o suspender el evento o ensayo.
- Para el repertorio, los distintos documentos (partitura y partes) que necesitan los músicos se envían a través de alguna plataforma de almacenamiento en la nube (como Google Drive, OneDrive o Dropbox) o se imprimen en papel físico por los administradores, repartiéndose en un ensayo a los miembros presentes.
- Respecto a la comunicación de eventos y ensayos, se realiza por el canal no especializado de comunicación y la responsabilidad de recordarlos debidamente a los miembros antes de su celebración recae en los administradores.

Esta gestión manual y dependiente de aplicaciones de comunicación no especializadas deriva en varios problemas:

- La probabilidad de error humano aumenta en el momento en el que los administradores son responsables de recordar eventos y gestionar asistencia.
- La carga de trabajo de los administradores es mayor dada la responsabilidad de gestión manual que tienen a la hora de organizar un evento.
- Se crea una duplicidad de comunicación, ya que los miembros pueden escribir sobre su asistencia al director o a los administradores, pudiendo darse el caso de que finalmente no todos dispongan de los datos necesarios para organizar un ensayo o evento, o que dispongan de datos contradictorios entre sí.

Por todos estos problemas derivados de la gestión manual se necesita software dedicado específicamente a este propósito, automatizando la gestión, que sea personalizable a las necesidades de cada agrupación, no implique enormes gastos de uso y sea fácilmente usable por todos los usuarios.

De este modo, el proyecto está justificado por la necesidad de alternativas para mejorar la gestión del gran número de agrupaciones musicales que existen en nuestro país y que no han podido automatizar aún su gestión.

## 1.2. Objetivos

Los objetivos de este trabajo se pueden resumir en tres fundamentales:

- Por un lado, dar respuesta a los requerimientos de numerosas agrupaciones musicales con un software que les permita automatizar la gestión de eventos, miembros, ensayos, repertorio, etc., ayudando a una organización más ágil y efectiva.
- Por otro lado, investigar hasta qué punto la interfaz de usuario proporcionada por un bot de Telegram es suficiente, competitiva y cómoda para el usuario. Esta interfaz está integrada en un chat, por lo que se pretende identificar las limitaciones que puedan darse, y explorar soluciones que sorteen dichas carencias.
- Aportar al conocimiento colectivo aportando guías sobre el desarrollo de un bot de Telegram usando tecnologías actuales.

Para el primer objetivo se cumpla de manera satisfactoria, se establecen varios requisitos básicos:

- Se debe disponer de una página web donde quede claro qué problemas pretende resolver la herramienta, de modo que las agrupaciones puedan ver las ventajas de su uso. Esta información también debe quedar clara en la propia herramienta.
- El uso de la herramienta debe ser lo más sencillo y obvio posible, dada la heterogeneidad de los usuarios, por su edad, profesión, nivel de estudios o grado de implicación en la agrupación.
- En el desarrollo de la herramienta se tienen que tener en cuenta los roles de miembro no administrador y quien sí es administrador, de forma que cada rol deberá tener distintas funcionalidades disponibles. Por ejemplo, solo un administrador podrá eliminar a otros miembros.
- Se debe buscar el mejor diseño que se pueda ofrecer dentro de las limitaciones de la tecnología escogida, de modo que un mal diseño no haga que los usuarios prefieran volver a la gestión manual.



## **Capítulo 2**

# **Estado del arte**

### **2.1. Dominio del problema a resolver**

En la actualidad, la digitalización de empresas, administraciones y organismos públicos avanza a ritmo imparable, y la situación excepcional generada por la pandemia de COVID-19 no ha hecho más que afianzar este avance.

Los datos del Índice de Digitalización de la Economía y la Sociedad (DESI) en 2022[2], índice que monitoriza el rendimiento digital de Europa y el progreso de los países de la Unión Europea en su competitividad digital, muestran que todos los países han progresado en su digitalización. Destaca el hecho de que los países que han empezado en un nivel más bajo de desarrollo digital crecen a un ritmo más rápido, y que numerosos países crecen a un ritmo más elevado del que se les espera, como es el caso de España[3] (con un 0.7 % más de progreso del esperado). En concreto, nuestro país es el séptimo con mayor nivel de digitalización según este índice, y el número de personas con competencias digitales básicas es del 64 %, por encima de la media de la UE, del 54 %. Está por encima de la UE también en implementación de la banda ancha fija de al menos 100 Mbps, usuarios de la administración electrónica o pymes con nivel básico de intensidad digital.

A nivel empresarial otros estudios concluyen que la digitalización está más estancada[4], debido a una baja inversión en TIC.

Por otro lado, a nivel de la administración se están impulsando estrategias para avanzar en la digitalización, como España Digital 2026[5], fomentando programas para acelerar la digitalización de pymes (pequeñas y medianas empresas), mejorar la conectividad o

Pese a todo esto, algunos sectores de la sociedad aún manifiestan reticencias para proceder a su digitalización, por la falta de herramientas, recursos o ayuda. Una de ellas es la que nos ocupa en este trabajo: las asociaciones o agrupaciones musicales. La complejidad de su gestión es suficiente para que los beneficios de la digitalización se puedan manifestar, dada la cantidad de

personas que se pueden encontrar en cada una de ellas y la naturaleza de los procesos que siguen, como la programación de ensayos y eventos, la gestión de repertorio o la gestión de miembros.

Por ejemplo, para la correcta planificación de ensayos y eventos, tanto el director como los administradores necesitan conocer con antelación qué músicos tienen pensado asistir y cuáles estarán ausentes. Esto les permitirá saber si se necesita llamar a un músico externo a la agrupación para que participe en un evento, o si es mejor cancelar o aplazar el evento. Para esta tarea, habitualmente el director avisa del evento por un grupo de mensajería a los miembros, y pide que quien no tenga pensado asistir se lo notifique. De este modo se generan varios problemas:

- La información sobre qué miembros han notificado su *no asistencia* solo la tiene el director, por lo que si otros administradores quieren conocerla se la tienen que pedir individualmente.
- Si algún miembro ha comunicado su “no asistencia” a otro administrador, tiene que poner la información en común con la del director, pudiendo dar lugar a discrepancias.
- Si el director quiere tener la información de asistencia centralizada, tiene que repasar sus conversaciones con los miembros para elaborar una tabla de asistencia, que tiene que ir actualizando manualmente, con la carga de trabajo que ello supone.

Similares problemas se crean para los demás procesos que se llevan a cabo en una agrupación musical.

Es evidente, por tanto, que se necesita una herramienta específica que permita automatizar sus procesos y avanzar en su digitalización para equipararla a la del resto de la sociedad.

Aunque ya existen herramientas como Glissandoo, que se analiza en la sección 2.2.1, son de pago, poco personalizables o están poco integradas en el flujo de trabajo actual de las agrupaciones. Es por ello que este trabajo pretende aportar una alternativa libre que solucione estos inconvenientes, de modo que:

- Cada agrupación pueda montar su propio servidor con la aplicación, personalizándola y adaptándola a sus necesidades.
- Los gastos por uso de la herramienta solo dependan del servidor que use la agrupación, si usa un servidor dedicado.
- Sea usable en todas las plataformas en las que se pueda acceder a un navegador web.

- Esté integrada en una herramienta de mensajería que ya esté en uso como es **Telegram**<sup>1</sup>, pero añadiendo funcionalidad específica y automática necesaria para la gestión de la agrupación, eximiendo a los administradores y el director de tareas que se pueden automatizar.

## 2.2. Aplicaciones similares

Aunque aún la mayoría de agrupaciones usan Whatsapp como herramienta de comunicación sin usar ningún software más que automatice las tareas, en los últimos tiempos ha surgido otra herramienta que da respuesta a sus necesidades:

### 2.2.1. Glissandoo

Glissando<sup>2</sup> es un software creado por la empresa Plausible Technologies, con sede en Valencia. En su página web se define como “un software que nace con el objetivo de profesionalizar, modernizar y digitalizar las instituciones musicales”. Incorpora numerosas funcionalidades para digitalizar instituciones musicales, tal y como pretende este trabajo. Entre ellas se encuentran:

- Organización de ensayos y conciertos. Los miembros pueden ver los ensayos y conciertos en la página de inicio.
- Previsión de asistencia y pasar lista: para cada ensayo o concierto, el miembro puede seleccionar si tiene prevista su asistencia o no.
- Distribución de partituras: en la sección de inicio se puede acceder a un listado de todo el repertorio de los ensayos y conciertos programados, y en cada uno de los ensayos se puede ver el repertorio asociado.
- Comunicaciones: los administradores pueden añadir avisos que reciben los miembros, teniendo la capacidad de responder, reaccionar, y adjuntar archivos a los comunicados.

Se pueden ver varias capturas de la aplicación en la figura 2.1.

Algunas ventajas de esta alternativa son:

- Presenta un diseño intuitivo y fácil de usar para todos los usuarios.
- Está disponible para las plataformas móviles iOS y Android, ampliamente usadas por los usuarios.
- Es gratuita si la agrupación tiene menos de 20 músicos.

---

<sup>1</sup><https://telegram.org/>

<sup>2</sup><https://glissandoo.com/>

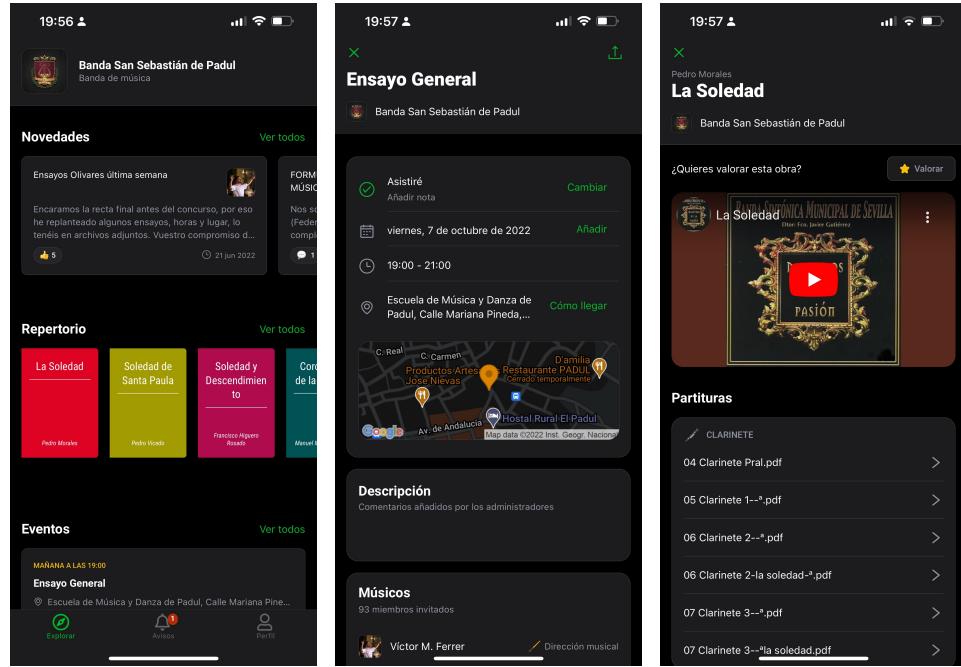


Figura 2.1: Capturas de la aplicación *Glissandoo*

- Es elaborada en por una empresa valenciana, lo cual da seguridad de que se adaptará a las necesidades de las agrupaciones musicales españolas.
- Incorpora un módulo de comunicaciones para poder centralizar todos los procesos de la agrupación en la aplicación, sin dejar los comunicados en otra aplicación de mensajería como WhatsApp.

Sin embargo, esta herramienta tiene varias desventajas:

- El código es privado, y una agrupación no puede montar su propio servidor con la aplicación en cuestión.
- Es de pago para uso en agrupaciones con más de 20 miembros.
- Solo se puede usar en móvil, ya que no dispone de versión web ni de escritorio. (Esto ha cambiado durante la realización de este trabajo, ya que han implementado una versión web).
- Sigue existiendo duplicidad, ya que aunque los miembros usen esta aplicación, la comunicación bidireccional ocasional entre administradores y miembros de la banda se sigue realizando a través de una herramienta distinta de comunicación no especializada.

### 2.2.2. Agregación de servicios no especializados

Mediante *agregación de servicios no especializados* nos referimos al uso de varias herramientas que no están especializadas para este caso de uso, pero que de cierta forma pueden suplir las necesidades:

- Como almacenamiento de partituras, existen servicios como **Google Drive**<sup>3</sup>, **Dropbox**<sup>4</sup> o **Microsoft OneDrive**<sup>5</sup>.
- Para la planificación de eventos y conciertos se puede usar un calendario web como **Google Calendar**<sup>6</sup> o **Microsoft Outlook**<sup>7</sup>.
- Para estimar la asistencia se pueden usar herramientas que permiten encuestas de este tipo como **Doodle**<sup>8</sup>.
- Para controlar la asistencia real se puede usar una hoja de cálculo, por ejemplo en **Google Sheets**<sup>9</sup>.
- Como medio de comunicación se utiliza otro servicio como **WhatsApp**<sup>10</sup> o **Telegram**<sup>11</sup>.

La ventaja de esta aproximación (la mayormente utilizada en la actualidad) es el mayor grado de personalización, ya que los servicios se pueden intercambiar si la funcionalidad o confiabilidad no es la esperada.

No obstante, las desventajas son numerosas, como ya se ha comentado anteriormente:

- Gran carga de trabajo para los administradores.
- Redundancia de comunicaciones entre los miembros y los administradores.
- Muchas de estas herramientas requieren de suscripción *premium* para usar toda la funcionalidad.

---

<sup>3</sup><https://drive.google.com/>

<sup>4</sup><https://www.dropbox.com/>

<sup>5</sup><https://onedrive.live.com/>

<sup>6</sup><https://calendar.google.com/>

<sup>7</sup><https://outlook.live.com/>

<sup>8</sup><https://doodle.com/>

<sup>9</sup><https://docs.google.com/spreadsheets/>

<sup>10</sup><https://www.whatsapp.com/>

<sup>11</sup><https://telegram.org/>



# Capítulo 3

## Mordente

Tras estudiar la motivación de este trabajo y habiendo justificado la necesidad de una alternativa software para la gestión de agrupaciones musicales, pasamos a desarrollar el producto. Para esta fase se va a hacer uso de la llamada metodología de Diseño Centrado en el Usuario (UCD por sus siglas en inglés), descrita en el siguiente apartado.

### 3.1. Desarrollo de Mordente

En este punto se van a establecer las metodologías de diseño a seguir durante el desarrollo de la herramienta, de modo que tengamos disponibles unas guías a seguir durante el proceso y unas herramientas que nos permitan definir la funcionalidad esperada y concretar las necesidades de los futuros usuarios.

#### 3.1.1. Diseño Centrado en el Usuario

El Diseño Centrado en el Usuario está estrechamente relacionado con el Diseño Centrado en el Humano[6]. Este último “es una aproximación al desarrollo de sistemas interactivos que se centra específicamente en hacer sistemas usables. Es una actividad multi-disciplinar” [7].

Los principios[8] en los que se fundamenta se pueden resumir en:

1. **Aproximación temprana a usuarios y tareas:** la recogida de información es estructurada y sistemática.
2. **Medida empírica y testeo del uso del producto:** se centra en la facilidad de aprendizaje y de uso, y la prueba de prototipos con usuarios reales.
3. **Diseño iterativo:** el producto se diseña, modifica y prueba de forma repetida, permitiendo repensar completamente la idea gracias a la prueba de modelos conceptuales tempranos.

### 3.1.2. Design Thinking

El Design Thinking es un proceso de diseño descrito en cinco fases[9]: definir el problema, buscar las necesidades, idear, construir y probar. Este proceso nos ayuda a aplicar el Diseño Centrado en el Usuario mediante una serie de procesos.

#### Procesos

Otros autores hablan de que el Design Thinking no se compone de fases sino de varios espacios que no son secuenciales: se solapan y se forma un bucle entre ellos, repitiéndolo más de una vez y explorando distintas aproximaciones [10].

1. **Inspiración:** se observa cómo funcionan las personas para encontrar problemas y oportunidades.
2. **Empatía:** se trata de entender los deseos y necesidades de los potenciales clientes, no solo de forma técnica sino sabiendo por qué hacen las cosas de la forma actual y qué les resulta más importante.
3. **Ideación:** se trata de una combinación de pensamiento divergente y convergente. El pensamiento divergente genera nuevas ideas de forma creativa, mientras que el convergente trata de sintetizar las ideas y elegir las mejores. Teniendo diversas personas en el equipo, la técnica más utilizada es el *brainstorming*, en el que todos los miembros dan ideas espontáneas que se apuntan de forma que cada nueva idea puede fomentar la creatividad de los demás.
4. **Implementación y prototipado:** en este proceso las ideas se convierten en productos concretos. Durante este proceso se van generando diversos prototipos que se prueban con usuarios reales para seguir mejorando la idea.

## 3.2. Personas ficticias

Las personas ficticias se crean dentro de una técnica perteneciente al *Design Thinking*. Se tratan de representaciones ficticias de los usuarios, con metas, motivaciones, características y comportamientos de un grupo real de usuarios [11].

Esta técnica nos permite diseñar para personas reales y no para "usuarios" abstractos.

En el caso del proyecto que nos ocupa, se han creado cuatro personas ficticias: dos que cumplirían el rol de administrador en el sistema y otras dos que lo usarían únicamente como miembros.

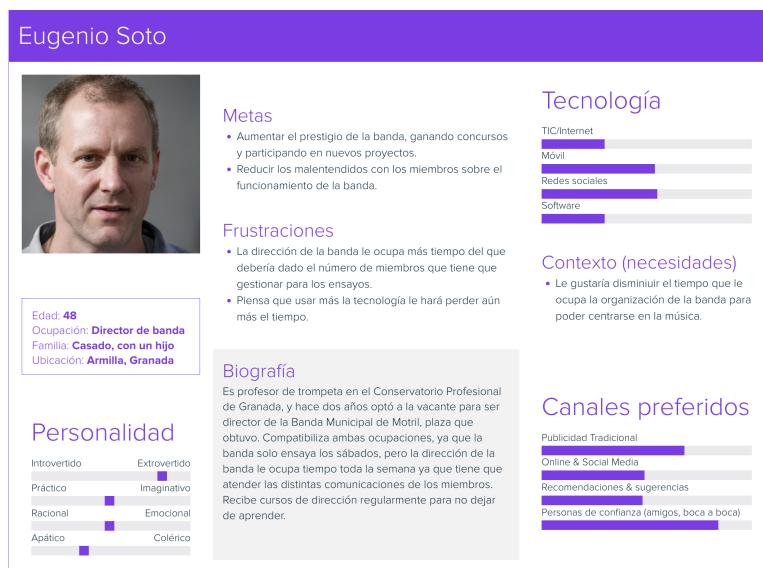


Figura 3.1: Persona: Eugenio Soto

Las figuras 3.1 y 3.2 muestran dos personas que se dedican a la organización de la banda, Eugenio más desde el punto de vista musical y Clara desde el logístico.

Mientras Clara está claramente más familiarizada con las nuevas tecnologías, Eugenio muestra algunas reticencias que deberemos tratar durante el desarrollo, de forma que el producto final sea fácilmente usable y accesible. Igualmente, ya que Clara sí usa las nuevas tecnologías habitualmente, el producto tiene que ser atractivo para ella de forma que no suponga un retroceso con respecto a las herramientas actuales.

Los perfiles que vemos en las figuras 3.3 y 3.4 se refieren a personas que acuden como miembros a la banda de su respectivo pueblo. A Inma se aplican las mismas restricciones que a Eugenio, además de que debemos asegurarnos de que realmente facilitamos que los miembros recuerden los eventos diarios para hacerles la vida más fácil. Con respecto a David, cabe reseñar que, ya que la música no es su mayor afición, preferirá usar la app durante el menor tiempo posible, por lo que se debe intentar que las notificaciones que le lleguen le permitan responder de forma rápida.

### 3.3. Malla receptora de información

La malla receptora de información (*Feedback Capture Grid*) es un método para organizar feedback de manera estructurada en el cual se divide una hoja de papel en cuatro cuadrantes, cada uno representado con un símbolo diferente. El superior izquierdo se representa con un símbolo "+", y en él



Figura 3.2: Persona: Clara Villena

anota el feedback positivo. En el superior derecho, con símbolo de triángulo, se recogen las críticas. En el inferior izquierdo, con una interrogación, se añaden las preguntas que han surgido y en el inferior derecho, con una bombilla, las nuevas ideas que surgen [12].

A continuación se resumen los puntos recogidos para cada uno de los apartados.

### 3.3.1. Aspectos interesantes o relevantes de la idea

- Está integrado en una app de mensajería.
- Funciona en todos los dispositivos móviles y de escritorio.
- El software es de código abierto.
- Lo puede utilizar cualquier agrupación.
- Se reciben notificaciones a través de la misma app de comunicación.
- Se pueden crear instancias del bot configurables a las necesidades de distintas agrupaciones.

### 3.3.2. Críticas constructivas o posibilidades de mejora

- Depende del funcionamiento de Telegram.
- La interfaz es más limitada que una aplicación web.

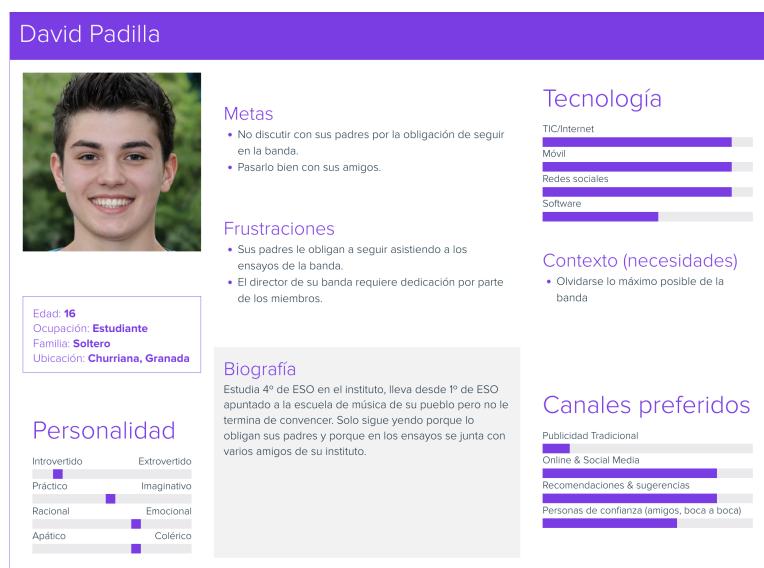


Figura 3.3: Persona: Eugenio Soto

- Se requiere tener cuenta de Telegram.
- Se requiere un servidor propio en el que esté alojada la aplicación para funcionar, o pagar el uso de uno en la nube.
- La interfaz en ordenador requiere de demasiados clics para llevar a cabo las tareas.

### 3.3.3. Preguntas nuevas a partir de la experiencia del usuario

- ¿Cuál es el coste de mantener la aplicación?
- ¿Se podrá pedir algo al bot desde un grupo?
- ¿Se podrán elegir qué notificaciones y recordatorios queremos recibir?
- ¿Los administradores podrán ver informes de la asistencia a ensayos?
- ¿La aplicación fomentará la asistencia de los miembros a los ensayos?
- ¿Los usuarios más reacios a la tecnología sabrán usar el bot con soltura?
- ¿Se va a poder usar el bot en varios idiomas?



Figura 3.4: Persona: Inma Medina

### 3.3.4. Ideas que surgen

- Hacer también una aplicación web para no depender de Telegram.
- Implementar un sistema de premios para los usuarios que cumplan objetivos de asistencia.
- Tener una instancia del bot funcionando para las agrupaciones que no lo quieran tener en un servidor propio.
- Crear una página web con guías para el uso del bot.
- Implementar internacionalización con varios idiomas.

## 3.4. Descripción de la propuesta

Teniendo en cuenta el análisis que se ha hecho hasta ahora, se puede dar una propuesta de solución que responda a las preguntas y necesidades expuestas, tanto en las personas ficticias como en el *Feedback Capture Grid*.

### 3.4.1. Propuesta del producto

Primeramente se determinan unas características y funcionalidades mínimas para tener un Producto Mínimo Viable, o MVP por sus siglas en inglés. El Producto Mínimo Viable es una técnica que se basa en disponer

de suficientes características para satisfacer a los primeros usuarios, de manera que las siguientes funcionalidades se pueden desarrollar teniendo en cuenta el *feedback* de estos primeros usuarios [13]. En nuestro caso, las funcionalidades incorporadas en esta primera versión serán:

1. Agrupaciones:
  - a) [Administrador] Crear nuevas agrupaciones
  - b) [Administrador] Eliminar agrupaciones
  - c) [Administrador] Invitar miembros a la agrupación
  - d) [Administrador] Eliminar miembros de la agrupación
  - e) [Usuario] Unirse a una agrupación existente
  - f) [Usuario] Salirse de una agrupación
  - g) [Usuario] Ver todas las agrupaciones en las que estoy inscrito
  - h) [Usuario] Ver el detalle de una agrupación
2. Eventos:
  - a) [Administrador] Crear nuevos eventos en mi agrupación
  - b) [Administrador] Eliminar eventos
  - c) [Administrador] Invitar miembros a un evento
  - d) [Administrador] Eliminar miembros de un evento
  - e) [Usuario] Ver los eventos próximos
  - f) [Usuario] Ver el detalle de un evento
3. Asistencia:
  - a) [Administrador] Ver qué usuarios han respondido su asistencia a un evento
  - b) [Usuario] Responder la asistencia a un evento
  - c) [Usuario] Registrar la razón por la que no puedo asistir a un evento
4. Recordatorios:
  - a) [Usuario] Recibir recordatorios diarios con los eventos de ese día
5. Repertorio:
  - a) [Administrador] Añadir una obra a la agrupación
  - b) [Administrador] Eliminar una obra la agrupación
  - c) [Administrador] Subir partituras de las obras

- d) [Administrador] Eliminar una obra de la agrupación
- e) [Usuario] Ver las obras de mi agrupación
- f) [Usuario] Descargar las partituras de mi agrupación

#### 6. Administración:

- a) [Administrador] Añadir administradores
- b) [Administrador] Quitar administradores

#### 7. Notificaciones:

- a) [Administrador] Recibir notificación cuando un miembro responde sobre su asistencia
- b) [Usuario] Recibir notificación cuando se asigna un evento

### 3.4.2. Elección de metodología de desarrollo

Dadas las características del proyecto, se va a apostar por una metodología de desarrollo ágil como es SCRUM, aunque con ciertas adaptaciones ya que el equipo de desarrollo va a estar compuesto por una sola persona.

Esta metodología ágil nos permitirá ir mejorando el producto a lo largo del desarrollo, pudiendo probarlo desde etapas tempranas. SCRUM es un método de desarrollo e innovación que enfatiza un conjunto de valores y prácticas para la gestión de proyectos, más que tratar requerimientos e implementación. Trata de hacer el proceso de la gestión más **empírico** que **definido** [14].

La naturaleza flexible del marco de trabajo SCRUM se adapta a proyectos como el que nos ocupa, dado que no se conocen todos los requisitos concretamente con antelación y nos ayuda a que el usuario sea parte del desarrollo, algo útil si queremos hacer un desarrollo centrado en el usuario.

### 3.4.3. Planificación

Para poder tener un producto que cumpla con los objetivos reseñados anteriormente, es necesario establecer en qué orden se van a realizar las distintas tareas.

#### Planificación en sprints

Dado que la metodología de desarrollo a usar será SCRUM, para la planificación del proyecto se va a usar uno de sus elementos más importantes como es la planificación de los sprints.

Los sprints son eventos de una longitud fija, generalmente menor a un mes. Durante un sprint se intenta realizar todo el trabajo necesario para alcanzar el objetivo del producto.

Los sprints corresponden a objetivos concretos del producto, por lo que el resto de tareas (como la toma de decisiones o el aprendizaje) se van a incluir en la planificación temporal pero no como sprints. Los sprints planificados para este proyecto son:

1. **Primer sprint:** Se va a implementar un bot de prueba con funcionalidad de actualización de una base de datos de palabras. Este bot servirá como exploración de las distintas funcionalidades que se usarán posteriormente.
2. **Segundo sprint:** Se va a implementar la lógica para responder a los comandos de los usuarios, así como la funcionalidad para crear agrupaciones y unirse a una agrupación.
3. **Tercer sprint:** Se van a crear diversos plugins, uno para permitir guardar la sesión del usuario en la base de datos y otro para usar un menú de calendario que sirva para fijar las fechas de eventos. Estos plugins serán publicados.
4. **Cuarto sprint:** Se añadirá la funcionalidad básica relacionada con los eventos, como la creación y eliminación. También permitir a un miembro salirse de una agrupación, así como habilitar enlaces de invitación.
5. **Quinto sprint:** Se podrán añadir y eliminar obras a las agrupaciones. Función de edición de agrupaciones, eventos y obras.
6. **Sexto sprint:** Notificaciones de asignación de eventos y adición de miembros a las agrupaciones además de recordatorios de los eventos diarios.

### Planificación temporal

La planificación del desarrollo del bot en el tiempo se sucederá de la siguiente forma:

- Primeramente se tomarán las decisiones oportunas sobre las herramientas a utilizar, requisitos concretos del software y análisis de la competencia.
- Después habrá un periodo de formación en las herramientas necesarias para el desarrollo de la aplicación. Esta formación también se extenderá a lo largo de todo el desarrollo.
- Los sprints de desarrollo vendrán tras el periodo inicial de formación.
- Finalmente tendremos un periodo de pruebas y de finalización de la memoria.

Toma de decisiones	10 horas
Formación	100 horas
Implementación	150 horas
Pruebas	10 horas
Memoria	150 horas
Total	420 horas

Tabla 3.1: Planificación temporal

En la tabla 3.1 se muestra el tiempo estimado que requerirá cada tarea del desarrollo. Dado que el tiempo requerido para cada crédito ECTS es de entre 25 y 30 horas y esta asignatura es de 12 créditos, hablaríamos de 300-360 horas, sin embargo el tiempo para el desarrollo se estima mayor por la formación previa requerida en las nuevas tecnologías utilizadas.

Este exceso de tiempo no se estima perjudicial dada la naturaleza formativa de este trabajo, ya que los conocimientos adquiridos podrán seguir siendo utilizados posteriormente a la finalización.

#### Diagrama de Gantt

El diagrama de Gantt que se puede ver en la figura 3.5 se ha construido teniendo en cuenta que cada día se van a trabajar 2 horas, por lo tanto se van a trabajar 210 días, o lo que es lo mismo, 42 semanas.

Así mismo, no se concretan las fechas reales ya que pueden cambiar por factores externos durante el desarrollo del proyecto.

#### 3.4.4. Presupuesto

El coste de infraestructura es nulo en esta primera versión, puesto que el bot puede estar alojado en un servidor propio o usando el plan gratuito de cualquier servicio de alojamiento de servidores virtuales en la nube. Por tanto, el coste a considerar es el del tiempo empleado por el desarrollador.

Según el Anexo I del XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública<sup>1</sup> el salario mínimo a partir del 31 de diciembre de 2019 para un Programador Junior es de 15860,56 euros al año. La tabla 3.2 muestra el presupuesto estimado teniendo en cuenta el siguiente cálculo y la suposición de 12 meses el año:

$$12 \text{ meses} \times 4 \frac{\text{semanas}}{\text{mes}} \times 40 \frac{\text{horas}}{\text{semana}} = 1920 \text{ horas}$$

$$\text{coste por hora: } \frac{15860,56}{1920} = 8,26 \text{ €}$$

---

<sup>1</sup>[https://www.boe.es/diario\\_boe/txt.php?id=BOE-A-2018-3156](https://www.boe.es/diario_boe/txt.php?id=BOE-A-2018-3156)

Concepto	Unidades	€/unidad	Total
Hora de Programador Junior	420	8,26	3469,2
Total			3469,2

Tabla 3.2: Presupuesto

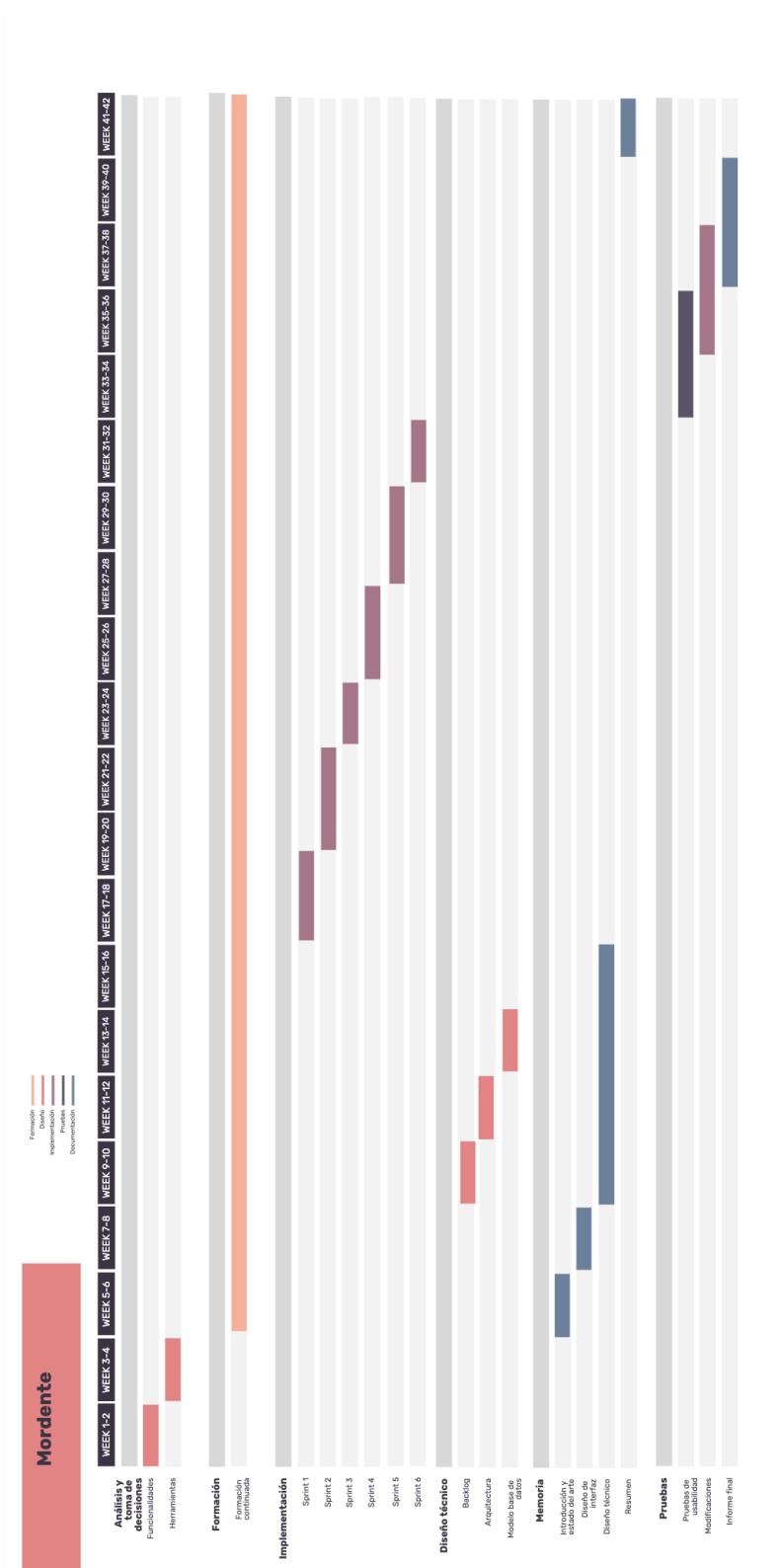


Figura 3.5: Diagrama de Gantt del proyecto

## Capítulo 4

# Diseño de la interfaz de usuario

A nivel de diseño de la interfaz, lo principal es reseñar que nos encontramos con una gran limitación: no podemos desarrollar una interfaz<sup>1</sup> totalmente a nuestro gusto, sino que debemos usar la API que nos proporciona Telegram.

Es decir, nuestro bot va a operar dentro de un chat, y como todo chat, la interfaz estará compuesta sola y únicamente por mensajes, que podrán ser de distinto tipo (texto, imágenes, documentos, stickers...). La única adición que aporta Telegram a esta interfaz es que los mensajes que envía el bot pueden adjuntar un menú de opciones que el usuario puede pulsar para pedir al bot una determinada acción, además de tener un menú de comandos disponible continuamente junto al cuadro de introducción de texto.

Esta limitación de interfaz (autoimpuesta) debe ser vista también como una serie de ventajas:

- Las decisiones de diseño son mucho más simples, puesto que no tenemos todo un elenco de estilos, formatos e interacciones que elegir. Por ejemplo, si el usuario quiere acceder a la información general sobre la aplicación, la mejor solución dentro del bot es que el usuario use un comando /about al cual el bot responde con un texto explicativo. En el caso de una aplicación web, la interacción por parte del usuario podría ser desde pulsar un botón hasta usar un atajo de teclado, y la visualización podría ser desde sobreponer un cuadro con la información hasta enlazar a una página distinta.
- La personalización ya está implementada para nosotros: esta característica se delega a la propia aplicación de Telegram, que permite se-

---

<sup>1</sup>El 16 de abril de 2022, la API para bots Telegram fue actualizada[15] con una nueva función para abrir una aplicación web sin salir del bot como respuesta a varios eventos. Sin embargo, esa opción se ha descartado para este trabajo ya que la implementación del bot se encontraba en estado avanzado y se ha analizado que las complicaciones técnicas de implementar esta nueva interfaz extenderían el tiempo de desarrollo considerablemente. Se propone como mejora futura.

lecciónar el tema claro u oscuro o cambiar el tamaño del texto.

- La accesibilidad está garantizada, ya que la interfaz de mensajes ya está totalmente adaptada a personas con discapacidad auditiva o visual.
- La familiarización del usuario con la interfaz es mayor de antemano, ya que con toda probabilidad habrá usado un chat antes y la interfaz del bot no es más que una extensión de un chat corriente. Esto le supondrá una adaptación más sencilla y más centrada en saber cuál es la funcionalidad que en cómo está dispuesta.
- El usuario interacciona de forma privada en un chat con el bot, de modo que el tono de respuesta del bot y el intercambio de mensajes hacen que el usuario perciba la interfaz como más amigable y cercana.
- Los entornos que más necesitan la comunicación se benefician de esta integración. Por ejemplo, en el caso que nos ocupa siempre se suele usar un grupo de mensajería, y un bot puede ser integrado en un grupo de Telegram.

Decimos que la limitación es autoimpuesta porque es tarea de este trabajo también analizar si las ventajas expuestas superan o igualan a las desventajas de operar dentro de un chat, es decir, si un bot de Telegram de este tipo puede operar sin la ayuda de una aplicación web o móvil que la acompañe.

#### 4.0.1. Bocetos de interfaz

Con todo lo anterior en mente, pasamos a crear varios bocetos de lo que debería ser la interfaz entre el usuario y el bot:

##### **Lista de elementos**

Aunque se ha expuesto que la interfaz está limitada a la API disponible, en el caso de las listas tenemos 3 opciones si queremos que el usuario pueda acceder al detalle de cada elemento:

1. Enviar un mensaje muy corto acompañado de un menú, cada botón del menú muestra el título de un elemento y al pulsar en él se recibe el detalle.
2. Igual que la opción 1, pero añadiendo en el texto una lista de los elementos con un resumen de cada uno.
3. Nueva opción explorada en este trabajo como variación de la opción 2: omitir el menú para no repetir los nombres, y en su lugar acompañar

cada elemento de la lista con un comando del estilo /event\_N de manera que al pulsar y enviar el comando el bot responda con el detalle del elemento.

En todo caso la lista debe ir acompañada de un botón para añadir elementos.

El diseño se ve en la figura 4.1.

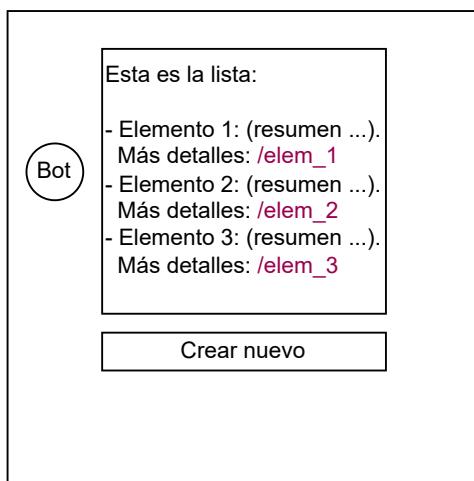


Figura 4.1: Diseño de la interfaz de lista

### Detalle de elemento

En este caso, toda la información sobre el elemento se muestra en el mensaje de respuesta, que va acompañado de un menú con las acciones que se pueden realizar con el elemento. Ver figura 4.2.

### Creación de elemento

La interfaz más sencilla y equivalente a un formulario para crear un elemento es una conversación: el bot va preguntando cada campo requerido al usuario, que puede saltar los campos opcionales mediante un botón de "Saltar". Idealmente, para la introducción de fechas se debería mostrar un menú con forma de calendario. Ver figura 4.3.

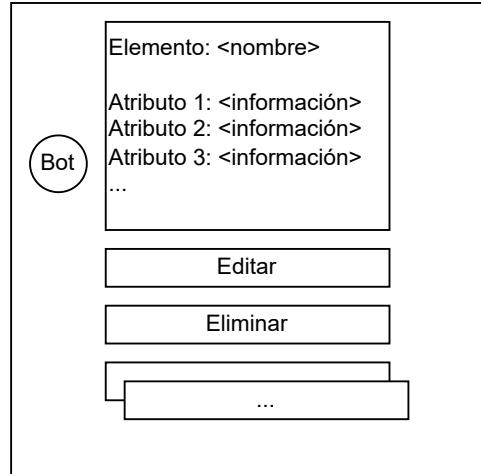


Figura 4.2: Diseño de la interfaz del detalle de un elemento

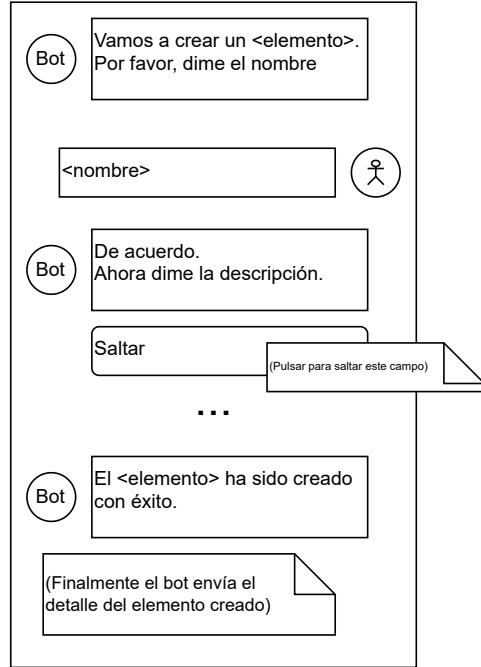


Figura 4.3: Diseño de la interfaz de creación de un elemento

# Capítulo 5

## Diseño técnico

### 5.1. Listado de historias de usuario (Backlog)

### 5.2. Historias de usuario

### 5.3. Metodologías y tecnologías de base que podrían usarse

#### 5.3.1. Lenguaje de programación

Se plantean los siguientes lenguajes de alto nivel, que disponen de frameworks implementados para crear un bot de Telegram:

- **JavaScript**: es un lenguaje compilado en tiempo de ejecución, con tipado dinámico y multi-paradigma, soportando programación orientada a objetos, funcional, imperativa y dirigida por eventos[16]. Se conforma al estándar “ECMAScript”, y es una de las tecnologías centrales de la World Wide Web: el 98 % de los sitios web lo usan en el lado del cliente[17], y desde el surgimiento de Node.js es una tecnología en auge para servidores.
- **TypeScript** (<https://www.typescriptlang.org/>): es un superconjunto de JavaScript que añade sintaxis para tipos estáticos, y a través de un compilador genera código JavaScript[18]. El añadido de tipos estáticos permite detectar errores de forma más temprana y agilizar la escritura de código gracias al autocompletado. Por otro lado, la adición de tipos es un tiempo extra empleado por el programador, por lo que debe analizarse si es conveniente su uso o no.
- **Python** (<https://www.python.org/>): es un lenguaje interpretado, interactivo y principalmente orientado a objetos, aunque soporta otros

	JavaScript	TypeScript	Python	PHP
Tipo	Compilado en tiempo de ejecución	Compilado a JavaScript	Interpretado	Interpretado
Tipado estático	No	Sí	Poco estricto, y poco uso	Poco estricto, y poco uso
Coste <sup>a</sup>	Nulo	Bajo	Medio	Medio
Popularidad <sup>b</sup>	65.36 %	34.83 %	48.07 %	20.87 %
Comunidad <sup>c</sup>	2432281	196707	2035248	1447104

<sup>a</sup>Aprendizaje necesario por parte del autor para poder implementar el proyecto

<sup>b</sup><https://survey.stackoverflow.co/2022/>

<sup>c</sup>Preguntas totales en stackoverflow: <https://stackoverflow.com/tags>

Tabla 5.1: Comparación de lenguajes de programación

paradigmas como el funcional o el procedimental[19]. Es muy usado en el campo de la computación científica y la inteligencia artificial. Su uso en el desarrollo web se ha extendido para el lado del servidor con la aparición de frameworks como Django (<https://www.djangoproject.com/>) o Flask (<https://flask.palletsprojects.com/>).

- **PHP** (<https://www.php.net/>): es un lenguaje interpretado usado principalmente para desarrollo web en el lado del servidor, cuya principal característica es que puede ser embebido en HTML, de forma que cada “trozo de PHP” se ejecuta para intercambiarse por HTML. Sin embargo su uso ha cambiado en los últimos años, dando lugar a frameworks como Symphony (<https://symfony.com/>) o Laravel (<https://laravel.com/>) basados en la arquitectura Modelo-vista-controlador.

Se proporciona una tabla comparativa entre los lenguajes, la tabla 5.1.

### 5.3.2. Framework para el desarrollo del bot

Existen diversas bibliotecas que ayudan a crear un bot, de forma que no haya que escribir todo el código desde cero para interaccionar con la API de Telegram, siguiendo el principio “Don’t Reinvent the Wheel” (“No reinventes la rueda”) para también ahorrar presupuesto.

Las opciones que se analizan son:

- **Telegraf** (<https://telegraf.js.org/>): biblioteca desarrollada inicialmente para JavaScript, migrada en la versión 4 dando soporte a TypeScript. No dispone de documentación guiada, y la migración a la ver-

	telegraf	grammY	node-telegram-bot-api	python-telegram-bot
Lenguaje	JavaScript	TypeScript	JavaScript	Python
Documentación	Autogenerada, solo API	Completa, numerosos tutoriales	Básica	Autogenerada, solo API
Versión API Telegram	6.2	6.2	6.2	6.2
Popularidad <sup>a</sup>	6.1k	663	6.5k	19.8k

<sup>a</sup>Estrellas en GitHub a 7 de octubre de 2022

Tabla 5.2: Comparación de frameworks para creación de bots de Telegram

sión 4 trajo consigo más complejidad de uso. Está inspirada el sistema de middleware de Express.<sup>1</sup>

- **grammY** (<https://grammy.dev/>): biblioteca desarrollada desde un inicio para TypeScript (aunque se puede usar con JavaScript). Está inspirada en telegraf y fue creada desde cero por uno de los antiguos mantenedores de Telegraf como única forma de resolver sus mayores inconvenientes<sup>2</sup>. Dispone de una completa documentación con guías para cada uno de los conceptos importantes.
- **node-telegram-bot-api** (<https://github.com/yagop/node-telegram-bot-api>): biblioteca ligera para interactuar con la API de Telegram, diseñada para Node.js, y no implementa un sistema de middleware.
- **python-telegram-bot** (<https://python-telegram-bot.org/>): biblioteca para Python que provee clases de alto nivel como capa de abstracción sobre la API de Telegram.

La tabla 5.2 compara los distintos frameworks.

### 5.3.3. Base de datos

La base de datos es “una colección compartida de datos lógicamente relacionados, junto con una descripción de estos datos, que están diseñados para satisfacer las necesidades de información de una organización”.[20]

Por otra parte, el sistema gestor de base de datos (SGBD) es “un sistema software que permite a los usuarios definir, mantener, crear y controlar el acceso a la base de datos.[20]

<sup>1</sup><https://expressjs.com/es/guide/using-middleware.html>

<sup>2</sup><https://github.com/telegraf/telegraf/discussions/1526>

Necesitaremos estas herramientas ya que buscamos un sistema que guarde datos de manera persistente y poder recuperarlos y actualizarlos en cualquier momento.

Se analizan alternativas para bases de datos relacionales, basadas en el concepto matemático de relación, representado por tablas[20]; y también para bases de datos no relacionales documentales.

- Para bases de datos relacionales:
  - PostgreSQL (<https://www.postgresql.org/>): es un SGBD de código abierto y centrado en la escabilidad.
  - MySQL (<https://www.mysql.com/>): es un SGDB de código abierto desarrollado por Oracle.
  - MariaDB (<https://mariadb.org/>): es un SGDB de código abierto creado por desarrolladores de MySQL como respuesta a la adquisición por parte de Oracle, y con características muy parecidas.
- Para bases de datos no relacionales:
  - MongoDB (<https://www.mongodb.com/>): SGDB de código disponible (no necesariamente de código abierto, por usar una licencia SSPL.[21]
  - Firestore (<https://firebase.google.com/docs/firestore>): incluido en la suite de servicios “Firebase” de Google destinada a la gestión rápida y centralizada de aplicaciones.

#### **5.3.4. Almacenamiento de archivos**

- Amazon S3
- DigitalOcean Spaces
- Google Cloud Storage / Firestore Storage

#### **5.3.5. IDE**

- VSCode
- JetBrains

#### **5.3.6. Documentación**

- Docusaurus
- GitBook
- VuePress

## 5.4. Modelo de la base de datos

## 5.5. Arquitectura de Mordente

Expliquemos la arquitectura utilizada en dos pasos:

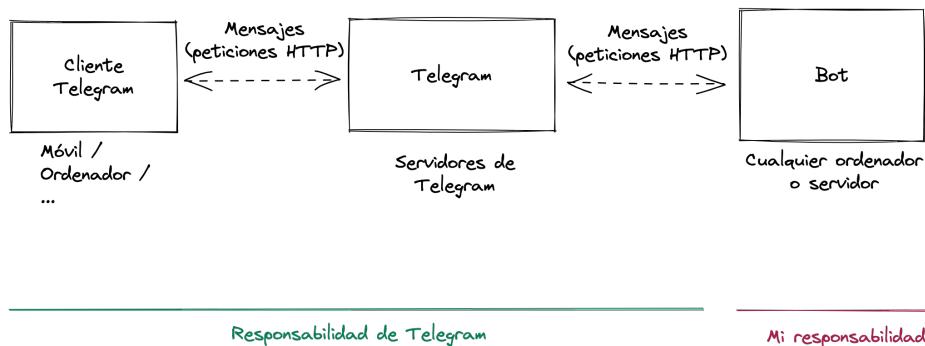
### 5.5.1. Arquitectura entre puntos

Con la arquitectura entre puntos nos referimos a la visualización a alto nivel de las distintas partes físicas necesarias para que nuestra herramienta funcione.

En nuestro caso, existen tres partes claramente diferenciadas:

- **El dispositivo del usuario**, o también llamado cliente. Tiene instalada la aplicación de Telegram, y se comunica bidireccionalmente con los servidores de Telegram para enviar y recibir mensajes.
- **Los servidores de Telegram**. Actúan como intermediarios entre distintos usuarios, así como entre los usuarios y nuestro bot.
- **El bot**. Es un programa que se comunica con los servidores de Telegram para enviar y recibir mensajes, tal y como los clientes, aunque con ciertas diferencias de funcionamiento con los clientes. Este programa puede estar alojado en cualquier ordenador, y en nuestro caso estará alojado en un servidor privado virtual.

Esta arquitectura está representada en la figura 5.1.



Responsabilidad de Telegram

Mi responsabilidad

Figura 5.1: Arquitectura entre puntos

Es importante entender que la única responsabilidad que nos corresponde a nosotros es la de desarrollar la última parte: el bot. El cliente y los servidores de Telegram son partes externas fuera de nuestra responsabilidad, lo cual es una de las ventajas de desarrollar un servicio de este tipo.

### 5.5.2. Arquitectura entre servicios

Para la funcionalidad que necesitamos, el bot debe estar compuesto por varios microservicios a su vez:

- **El propio bot:** es el microservicio donde implementaremos la comunicación con los servidores de Telegram para recibir mensajes, procesarlos y responderlos adecuadamente.
- El bot realiza peticiones a una **base de datos** donde almacenaremos de forma persistente los datos de los usuarios. Esto garantizará que no perdamos información cuando el bot se reinicie.
- Opcionalmente, podremos añadir un microservicio encargado de realizar una **copia de seguridad** periódica de la base de datos.
- Por otro lado tendremos un servicio de **almacenamiento de archivos** para guardar las copias de seguridad y otros archivos (en nuestro caso, las partituras).

En la figura 5.2 se puede visualizar la arquitectura entre los distintos servicios.

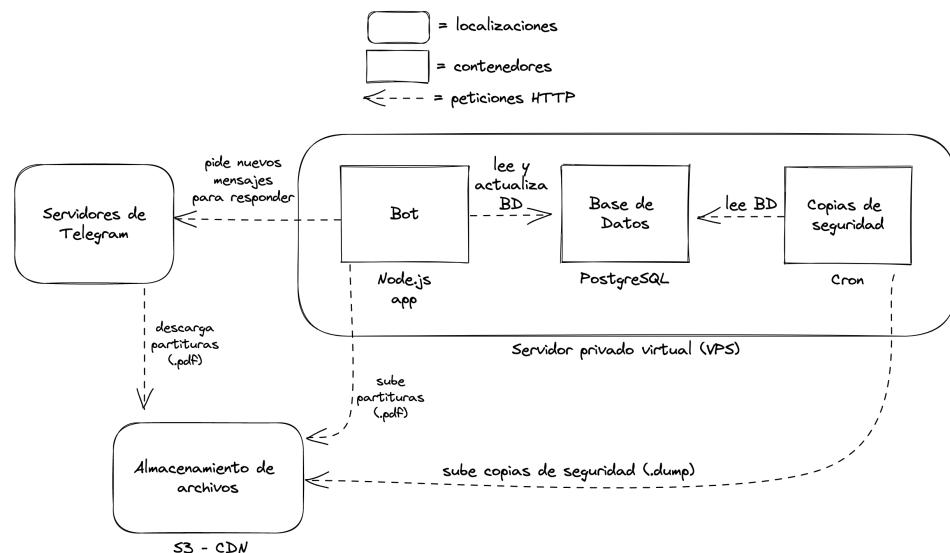


Figura 5.2: Arquitectura entre servicios

#### Localización de los servicios

Con respecto a dónde disponer los distintos servicios, existen al menos dos opciones: una es alojarlos en el propio servidor que gestionaremos nosotros y cuyos microservicios podremos levantar en un solo paso usando

*Docker Compose*<sup>3</sup>, y otra es utilizar servicios auto-gestionados por plataformas externas. Hagamos una reflexión para cada uno de los servicios:

- El **bot** puede estar en un servidor propio o en un servicio gestionado, ambas opciones son válidas. La ventaja de tenerlo en el servidor propio será la mayor cercanía posible con la base de datos, resultando en una mayor velocidad de respuesta. La ventaja de utilizar un servicio gestionado es la mayor facilidad de escalado (mantener la capacidad de respuesta independientemente del número de usuarios), sin embargo esto no es algo prioritario en la primera fase de desarrollo.
- Casi todos los proveedores de servicios en la nube ofrecen **bases de datos** gestionadas<sup>4</sup>, aunque siempre es conveniente que el bot y la base de datos se encuentren físicamente cercanos para reducir la latencia de las frecuentes consultas. Es por ello y porque no se prevé una gran cantidad de información almacenada durante el periodo inicial que optaremos por alojar la base de datos como un contenedor dentro de nuestro servidor, al igual que el bot.
- El **almacenamiento de archivos** siempre se suele delegar a un servicio externo de almacenamiento de objetos gestionado<sup>5</sup>. La mayor ventaja de esta aproximación es la disponibilidad de una CDN (*Content Delivery Network*, o Red de Entrega de Contenidos)[22]. Gracias a esta red tendremos un conjunto de servidores distribuidos geográficamente que pueden entregar rápidamente el contenido. Además, usar un servicio fuera de nuestro servidor garantiza que podamos realizar copias de seguridad de la base de datos que no se pierdan si el servidor se rompe por cualquier causa.

### Entorno de producción y de desarrollo

Es importante tener en cuenta también que esta configuración puede variar dependiendo de si nos encontramos en el entorno de desarrollo o de producción. En nuestro caso, el servicio de copias de seguridad no existirá en el entorno de desarrollo.

Además, tendremos dos réplicas (una de producción y otra de desarrollo) del bot, la base de datos y el almacenamiento de archivos para que las pruebas no afecten en ningún caso a los datos de los usuarios reales.

---

<sup>3</sup><https://docs.docker.com/compose/>

<sup>4</sup>Ejemplos: Digital Ocean Managed Databases (<https://www.digitalocean.com/products/managed-databases>), Heroku Postgres (<https://www.heroku.com/postgres>), Google Cloud Firestore (<https://cloud.google.com/firestore>)

<sup>5</sup>Ejemplos: Amazon S3 (<https://aws.amazon.com/es/s3/>), Digital Ocean Spaces (<https://www.digitalocean.com/products/spaces>) o Google Cloud Storage (<https://cloud.google.com/storage>)

## 5.6. Seguridad

Se intentará maximizar la seguridad de la aplicación gracias a los siguientes puntos:

- Los contenedores de Docker utilizados en el servidor están completamente aislados de la red excepto en los puertos que configuremos manualmente. En nuestro caso, el bot podrá conectarse a la base de datos pero ninguno de los contenedores tendrá puertos expuestos al exterior que pudieran ser usados por atacantes.
- Gracias al punto anterior, los ataques de denegación de servicio (DDoS) no son un riesgo. Además el servidor tendrá un Firewall configurado.
- El uso de herramientas *open-source* como Prisma para hacer las consultas a la base de datos en lugar de realizarlas manualmente con código SQL disminuye al máximo la posibilidad de Inyecciones SQL.
- La inyección de código al servidor no es posible ya que no se ejecuta ningún código enviado por el cliente, y por otro lado el cliente es responsabilidad de Telegram.
- Se añadirán métodos que permitan detectar automáticamente las vulnerabilidades introducidas por dependencias externas. Como el repositorio estará alojado en GitHub, se propone el uso de *Dependabot*<sup>6</sup>.

---

<sup>6</sup><https://docs.github.com/es/code-security/dependabot>

# Capítulo 6

# Implementación

## 6.1. Metodología de trabajo

Se va a usar una metodología ágil usando el *framework* Scrum adaptado a un equipo unipersonal.

### 6.1.1. Planificación temporal

El periodo desde la asignación del trabajo (29 de octubre de 2021) hasta el inicio del primer sprint se dedicará a revisar documentación sobre posibles tecnologías a usar, probándolas siempre que sea posible. Antes del inicio del primer sprint se tendrá:

- Una base del código del proyecto montada y lista para desplegar en un servidor.
- Una decisión de las tecnologías a usar.
- Una decisión sobre las herramientas a usar durante el desarrollo (gestión de tareas, gestión de versiones para el código, gestión de versiones para la documentación).

A partir del 21 de marzo de 2022, el trabajo simultáneo de desarrollo y escritura de la memoria se dividirá en cuatro sprints:

- Primer sprint: 21 de marzo a 1 de abril
- Segundo sprint: 18 de julio a 29 de julio
- Tercer sprint: 3 de octubre a 12 de octubre
- Cuarto sprint: 17 de octubre a 28 de octubre

El periodo entre el 28 de octubre y la entrega del trabajo se dedicará a la revisión de la memoria y el análisis de uso de la aplicación en un caso de uso real.

## 6.2. Presupuesto

### 6.3. Herramientas y tecnologías escogidas

#### 6.3.1. Lenguaje de programación

TypeScript

#### 6.3.2. Framework

GrammY

#### 6.3.3. Base de datos

PostgreSQL

#### 6.3.4. ORM

Prisma

#### 6.3.5. IDE

JetBrains

## 6.4. Primer sprint

Analizador de comandos Unirse a agrupación, Lógica de negocio para los modelos Middleware para plantillas

## 6.5. Segundo sprint

Middleware para información de usuario/cuenta Crear evento, eliminar evento Crear agrupación, eliminar agrupación Crear membresía, eliminar membresía Habilitar/deshabilitar enlace de invitación Crear evento, eliminar evento

Contribuciones a GrammY: Prisma Adapter, conversations issue, calendario

# **Capítulo 7**

## **Pruebas**

Tras el capítulo anterior, ya tenemos una implementación de nuestra herramienta que los usuarios pueden probar para darnos retroalimentación de cara a mejorar la usabilidad de la plataforma.

Las pruebas que se van a realizar en este trabajo, dado que la metodología usada ha sido la del *Design Thinking* y el Diseño Centrado en el Usuario, serán pruebas de usabilidad con usuarios reales.

### **7.1. Diseño de las pruebas**

En esta sección se va a determinar cuál será el diseño de las pruebas de usabilidad. Para ello debemos concretar quién participará en las pruebas, qué tendrán que hacer y qué se les va a preguntar.

#### **7.1.1. Participantes**

Para esta primera versión se harán pruebas con 10 personas.

Dado que hay dos roles claramente distinguibles en la aplicación, se seleccionarán 3 administradores y 7 miembros para completar la prueba de usabilidad.

#### **7.1.2. Tareas a realizar**

Para comprobar la usabilidad los flujos de trabajo distintos de administradores y miembros, se van a proponer tareas distintas para cada uno de los dos roles.

En el caso de los administradores:

1. Crea una agrupación.
2. Crea un evento.
3. Sube una obra a la agrupación.

4. Invita a un miembro de tu banda.
5. Haz que no pueda unirse nadie más a la agrupación.
6. Haz administrador al miembro que se ha unido.
7. Quítale los permisos de administrador al miembro que se ha unido.
8. Elimina el evento que creaste.

En el caso de los miembros:

1. Únete a una agrupación.
2. Mira los eventos de la agrupación.
3. Descarga alguna obra de la agrupación.
4. Sal de la agrupación.

### **7.1.3. Preguntas a realizar**

Las preguntas que se van a realizar a los usuarios son las descritas en la tabla 7.1.

Las relacionadas con tareas concretas nos ayudarán a realizar una mejora de la usabilidad antes de la entrega, solucionando los problemas que manifiesten los usuarios y que se puedan solucionar rápidamente, y planificando como trabajos futuros las soluciones más costosas.

Por otro lado, la primera pregunta genérica nos ayudará a calcular el *NET Promoter Score* o NPS. Según esta herramienta, los encuestados que respondan entre 0 y 6 son detractores, entre 7 y 8 son pasivos y entre 9 y 10 son promotores. De esta forma, se obtendrá un índice realizando el siguiente cálculo:

$$\text{NET} = \frac{\text{Promotores} - \text{Detractores}}{\text{Encuestados}} \times 100$$

Este índice puede resultar entre -100 y 100. Diremos que tiene un resultado positivo si es mayor a 0, y excelente si es mayor a 50.

## **7.2. Realización de las pruebas**

### **7.3. Informe final de las pruebas**

### **7.4. Demostración en vídeo**

Se puede visualizar el funcionamiento del bot en el vídeo disponible en este enlace:

Preguntas por tarea	
Pregunta	Tipo de respuesta
1. ¿Has podido realizar la tarea?	Sí/No
2. ¿Ha sido fácil realizar la tarea?	Entre 1 y 5.
Preguntas genéricas	
Pregunta	Tipo de respuesta
1. ¿Recomendarías la aplicación a un amigo?	Entre 0 y 10
2. ¿Tienes algún comentario?	Respuesta libre.

Tabla 7.1: Formulario para la prueba de usabilidad



## **Capítulo 8**

# **Conclusiones y trabajos futuros**

### **8.1. Conclusiones**

### **8.2. Trabajos futuros**

#### **8.2.1. Webapp**

Webapp utilizando el código existente.

#### **8.2.2. Webapp integrada en el bot**



# Bibliografía

- [1] INE, "Anuario Estadístico de España 2020." [https://www.ine.es/prodyser/pubweb/anuario20/anu20\\_04cultu.pdf](https://www.ine.es/prodyser/pubweb/anuario20/anu20_04cultu.pdf). [Online; Recuperado: 2021-09-27].
- [2] European Comission, "Digital Economy and Society Index (DESI) 2022." <https://ec.europa.eu/newsroom/dae/redirection/document/88764>. [Online; Recuperado: 2021-09-27].
- [3] European Comission, "Índice de la Economía y la Sociedad Digitales (DESI) 2022 - España." <https://ec.europa.eu/newsroom/dae/redirection/document/88760>. [Online; Recuperado: 2021-09-27].
- [4] UGT - Servicio de estudios, "Digitalización de la empresa española." <https://servicioestudiosugt.com/digitalizacion-de-la-empresa-espanola-3-edicion/>. [Online; Recuperado: 2021-09-27].
- [5] Ministerio de Asuntos Económicos y Transformación Digital, "España Digital 2026." [https://espanadigital.gob.es/sites/espanadigital/files/2022-07/Espa%C3%B1a\\_Digital\\_2026.pdf](https://espanadigital.gob.es/sites/espanadigital/files/2022-07/Espa%C3%B1a_Digital_2026.pdf). [Online; Recuperado: 2021-09-27].
- [6] W3C - Web Accessibility Initiative, "Notes on user centered design process (ucd)." <https://web.archive.org/web/20220901034811/https://www.w3.org/WAI/redesign/ucd>, 2004. [Online; recuperado: 2022-09-03].
- [7] ISO 13407, "Human centred design processes for interactive systems," 1999.
- [8] J. Rubin, "Handbook of usability testing: How to plan, design, and conduct effective tests," 1984.
- [9] L. J. L. Hasso Plattner, Christoph Meinel, "Design thinking: understand, improve, apply," 2011.
- [10] J. Brown, T. Wyatt, "Design thinking for social innovation. stanford social innovation review," 2010.

- [11] N. Turner, "Getting the most out of personas." <https://web.archive.org/web/20220722145021/https://www.uxforthemasses.com/personas/>, 2010. [Online; recuperado: 2022-07-22].
- [12] R. F. Dam and T. Y. Siang, "Test your prototypes: How to gather feedback and maximise learning." <https://web.archive.org/web/20220124203828/https://www.interaction-design.org/literature/article/test-your-prototypes-how-to-gather-feedback-and-maximise-learning>, 2021. [Online; recuperado: 2022-01-24].
- [13] Techopedia, "What is a minimum viable product (mvp)?" <https://web.archive.org/web/20220914143342/https://www.techopedia.com/definition/27809/minimum-viable-product-mvp>, 2020. [Online; recuperado: 2022-09-14].
- [14] C. Larman, *Agile and iterative development : a manager's guide / Craig Larman.* [electronic resource]. Agile software development series, Boston: Addison-Wesley, 1st edition ed., 2004.
- [15] Telegram, "Notification Sounds, Bot Revolution and More." <https://web.archive.org/web/20220905222430/https://telegram.org/blog/notifications-bots>, 2022. [Online; recuperado: 2022-09-05].
- [16] Wikipedia, "JavaScript — Wikipedia, the free encyclopedia." <http://en.wikipedia.org/w/index.php?title=JavaScript&oldid=1114045733>, 2022. [Online; recuperado: 2022-10-07].
- [17] w3techs.com, "Usage statistics of javascript as client-side programming language on websites." <https://wayback.archive-it.org/20220213043439/https://w3techs.com/technologies/details/cp-javascript>, 2022. [Online; recuperado: 2022-02-13].
- [18] "JavaScript with syntax for types.." <https://web.archive.org/web/20221004155521/https://www.typescriptlang.org/>, 2022. [Online; recuperado: 2022-10-04].
- [19] Python Software Foundation, "General Python FAQ — Python 3.10.7 documentation." <https://web.archive.org/web/20220722225638/https://docs.python.org/3.10/faq/general.html>. [Online; recuperado: 2022-07-22].
- [20] T. M. Connolly, *Sistemas de bases de datos [Recurso electrónico] : un enfoque práctico para diseño, implementación y gestión / Thomas M. Connolly, Carolyn E. Begg.* Madrid: Pearson Educación, 4<sup>a</sup> ed. ed., 2006.

- [21] OSI Board of Directors, "The sspl is not an open source license." <https://web.archive.org/web/20220930105202/https://opensource.org/node/1099>, 2021. [Online; recuperado: 2022-10-06].
- [22] Cloudflare, "¿Qué es una CDN?." <https://web.archive.org/web/20221011192027/https://www.cloudflare.com/es-es/learning/cdn/what-is-a-cdn/>, 2022. [Online; recuperado: 2022-10-11].
- [23] Wikipedia, "Python — Wikipedia, the free encyclopedia." <http://en.wikipedia.org/w/index.php?title=Python&oldid=1111676421>, 2022. [Online; recuperado: 2022-10-07].



