Histolog

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autor:
David Martínez Moya
Tutor/es:
Francisco García Sánchez

Facultad Informática Universidad Murcia

17 de Junio de 2021

Histolog

Aplicación para la gestión de información turística.

Autor
David Martínez Moya

Tutor/es
Francisco García Sánchez
Departamento de Informática y Sistemas

Facultad de Informática

Grado en Ingeniería Informática

DIS Departamento de Informática y Sistemas

UNIVERSIDAD DE MURCIA

Murcia, 17 de Junio de 2021

Resumen

Viajar ha sido siempre una de las experiencias más deseadas. El hecho de salir de nuestra comodidad para descubrir todo lo que el destino turístico puede llegar a ofrecer. La posibilidad de aprender la cultura del lugar y toda la historia que la envuelve.

Es por este motivo por el que es necesario buscar nuevas y mejores maneras de transmitir esta información. En especial, haciendo uso de la tecnología, la cual ya forma parte de nuestro día a día. Se hace posible la creación de una aplicación que permita ofrecer una manera sencilla y rápida de consultar información turística.

Se trata de una aplicación *web*, la cual permite la gestión de información turística. Contará con dos vistas, una dedicada a los turistas, en la que podrán visualizar la información. La otra está destinada a los encargados de administrar los datos turísticos. La *web* cuenta además con características que permiten una fácil gestión y visualización de la información en diferentes idiomas.

En este trabajo se detallan todos los procesos que han intervenido en la creación de esta aplicación *web*. Se detallan las razones de la necesidad de este proyecto, la metodología seguida, el análisis de los requisitos, el diseño del sistema y, finalmente, su implementación.

Extended Abstract

Traveling has always been and continues to be one of the most desired experiences. In recent years, thanks to all the advances that have been taking place, traveling is becoming a faster and cheaper activity. Today a person is able to be anywhere in the world in a few days without any problem, this fact has led many people every year to pack their bags and go to visit other countries. Naturally, one of the many things someone does on a tourist trip is learn about the history and culture that surrounds them. They are constantly looking for simple and fast ways to obtain as much information as possible, since time on every trip is usually scarce and must be used to the maximum.

To this end, governments around the world are constantly dedicating resources to innovation in tourism services and infrastructures. With them, they look for a way to satisfy the tourist, something that can prolong their stay or that can attract more tourists due to the quality of the services offered. It is an investment that should be taken into account in countries whose economy is based on tourism, such as Spain. In this context, the possibility arises of using the significant advance that new technologies have experienced in recent years after the creation of the Internet, the appearance of mobile devices and the connectivity between them. The development of a tool capable of providing information of tourist interest in a few seconds is made possible.

The application documented in this project is called *Histolog*, a tool for managing tourist information. Its operation consists of scanning QR codes using tourist devices. These codes will be placed in the places designated as tourist in a city, and will redirect the user to a web page in which relevant data will be displayed. The page presents the information in an accessible way, and offers the user the possibility to choose the language in which they wish to view the information. The advantages that are obtained by having decided to create a web application and not a mobile application are the facilities that the user has when using it. In this way, any device will be able to access the information without the need to make any download that entails a loss of time, unnecessary use of mobile data or the use of storage on the device.

However, this tool not only represents advantages for the tourist, but also for the entities dedicated to the maintenance of tourist infrastructures. These will make use of another web page that will have a multitude of characteristics that will allow the administration of the information provided. This tool offers a system of roles divided into administrators and writers. The former are in charge of managing all the information entered in the application. They are provided with many functionalities with which the different registered users can manage the posters created and the jobs started for each

poster. These works are established in one of the languages available in the system, and one of the editors is assigned to carry them out based on the languages he knows. The editors carry out the tasks received from the administrator, which consist of the modification or creation of information of each tourist location through the works. Once the editor sends the work, it can be accepted or rejected. In the case of being accepted, the content of the work will be applied to the content of its corresponding poster, and if it is rejected, the editor continues with the work to solve the errors found by the administrator. There is also an option for the administrator to cancel the job at any time. In this case, the job will end, the writer will not be able to continue working and no changes will be made to the poster.

If the state of the solutions that exist today for the type of tool that is being presented is studied, solutions appear that are not very well implemented. In the case of Murcia, its tourism website has a pleasant design and provides the necessary information for a tourist, however, it has some disadvantages. One of them is how difficult it can be for a user to search for information about a certain monument from their mobile device. Normally, tourists are on the street touring the city, so the only device they will have access to at that time will be their mobile phone. Another disadvantage is the quality of the information on the page when the language is changed. In general there is incomplete information, but especially in the pages of tourist places. In these pages the information is presented in Spanish or no details are given. There are other languages in which the problem goes further and the design changes completely, showing less information than is provided in other languages and making use of an outdated design. These disadvantages can be a source of frustration for users who are visiting the city.

However, a solution has already been found to the problems described above, and they are the information signs located near tourist places. In them you can read an extract of the history of the place, accompanied by a series of translations into other languages. However, the posters continue to have some problems such as: the ease of being damaged by acts of vandalism or by exposure to the elements, the little versatility to add translations into other languages, etc. With this, the idea of the web application that has been presented previously arises, which makes use of the versatility and maintainability that a website has to offer, in order to solve the problems that have been found in the posters. Therefore, this page simulates the operation of the posters but in a web format.

Having made clear the type of software needed for this job, it remains to choose the technologies that will be used for its creation. This choice has been made based on certain factors: the popularity of the technology, the amount of documentation available and its ease of use. The stack that has been used is going to be detailed, although it is worth highlighting the way in which a web application is usually distributed. Internally, a web application is composed of two fundamental parts: the front end or client side and the back end or server side. The front end is the interface the user interacts with to modify the system. It includes components that will obtain the information entered by the user and that will carry out the server requests. In addition, the interface must have

an accessible and usable design so that it can be used by any user, and be responsive, so it can be accessed without problems from any device. On the other hand, the back end is in charge of applying the logic and making the information persistent in the database. It receives HTTP requests through the defined routes and acts accordingly.

The front end has been developed mainly with *React*, a framework created by Facebook and intended for the creation of user interfaces. It is based on the creation of components, which can be easily reused throughout the application. This makes development easy and offers a similar layout and function across the entire page. Along with React, many other technologies have been used that have facilitated development work, among them the use of *TypeScript*, a language derived from *JavaScript* that implements the use of types. This language has been used both on the back end and on the front end, and thanks to it, it has been possible to create a more readable code and has helped to discover programming errors. On the back end, the most prominent technology has been *Express*, a *Node.js* framework that facilitates the start-up of a REST server. With this service, the requests that reach the server are processed and, if necessary, communications will be made with the database. Another technology that has facilitated the connection to the database has been *TypeORM*. With *TypeORM* the access and administration of the database has been quite simple. To complete the stack used, it is necessary to mention that *PostgreSQL* has been used.

Before starting to work on the project, the objectives and the methodology to be followed have been established. The main tasks that had to be carried out had to be differentiated, these are the ones that have been defined: a preliminary analysis of the state of the art and of the technologies to be used in development, an analysis of the requirements extracted from the application specification and a design of the system that will be developed, the development of the different parts of the application, the verification and validation of the software created and the documentation of the system and preparation of the report. All these tasks have been divided into subtasks and a planning has been made for their execution that covers the months from January to June.

Regarding the analysis and design of the system, it should be noted that many of the engineering techniques learned during the career have intervened. It starts with a complete specification of what the application should look like and the functionality that it should implement has begun. The extraction of functional requirements from the created specification has continued. These requirements have been established as use cases, with which it has been possible to distinguish some characteristics that must be taken into account, such as the actors that participate, the preconditions for the use case to occur, the main execution flow and the alternative ones. With well-defined functionality, the domain model has continued to be designed, which represents the entities and relationships between them that will exist in the system. From the domain model, the database schema is obtained, which reflects the way in which the information will be stored in the database. Next, the back end has been designed, indicating the access routes that will be implemented to access the API. Finally, the front end of the

application has been designed, so the pages that the web application will be composed of and their hierarchy have been differentiated from. Some mockups of the pages have also been created, so that you can quickly see the distribution that the functionality will have once the project has been completed. All the diagrams and models made up until this moment have greatly facilitated subsequent development work.

Regarding the programming section, it should be noted that the project has been developed as a monorepo. Therefore, all the code base, the client side and the server side, have been created in the same project. The tools used to create the monorepo are part of the *Nx* framework, and with them all the project's configuration files have been created, something that has allowed us to start programming on a solid basis and in a very short time. First, the back end was developed, during which numerous tests, both manual and automatic, have been carried out to verify its correct operation. Once finished, it has gone to the front end, where the interface views have been designed and it has been connected to the back end. After all this development process, it has been verified that the final product meets the requirements specified in the previous sections.

After all the work done it can be considered that the creation of a product that meets the specified requirements has been achieved. It is in a fairly mature state and, although it has not been tested in a production environment, it can be used as the basis for building a complete system. Although some modifications are necessary to make it a usable product, some of the improvements are: adding a notification system, improving the editors' text editor, improving the administrator's communication with editors and adding search bars. Once the complete system has been achieved, it can be expanded with new functionalities such as: adding audio guides or assigning a property that determines if a poster is visible or not to the public.

Índice general

Índice de figuras

Índice de tablas

Índice de Códigos

1. Introducción

Muchas personas estarán de acuerdo en que viajar es uno de los mayores placeres que existen en este mundo. La fascinación que se siente al explorar lugares completamente diferentes a los habituados mueve a mucha gente todos los años a reencontrarse con esta sensación, a poder disfrutar una vez más de todo lo que puede ofrecer una experiencia así.

Y ya no solo de disfrutar entornos diferentes, sino de aprender la cultura del lugar, sus costumbres, la historia que lo envuelve. Las personas desean exprimir todo lo posible la esencia del destino, pues la estancia es breve en comparación a toda la información valiosa que alberga el destino.

La gran evolución experimentada en la tecnología pone a nuestra disposición suficiente información como para responder cualquier duda que nos surja y de aprender todo aquello que deseemos; en la mayoría de casos de una forma rápida, eficiente y eficaz.

En este contexto, surge la posibilidad de hacer uso de las nuevas tecnologías para facilitar y mejorar la experiencia de los turistas a la hora de informarse de la historia del lugar que visita. *Histolog* es en una aplicación *web* destinada a la creación y difusión de información turística. Proveerá a los servicios municipales de una plataforma *web* donde podrán mantener registrada y actualizada toda información sobre los lugares de mayor interés turístico, y al usuario, una forma sencilla de visualizar toda esa información.

La motivación principal para la creación de este trabajo es el impacto que supone cada año el turismo en la economía española[1]. Es razón suficiente por la que se debería tomar en consideración mejorar las infraestructuras destinadas a ese uso, proveyendo a los potenciales visitantes de los mejores servicios.

1.1. Estructura de la obra

Este documento se organiza como sigue:

1. **Introducción.** Se establece una presentación con los motivos de la realización del trabajo y la estructura del documento.

2. **Estado del arte.** Se muestran las deficiencias de las tecnologías empleadas actualmente y se realiza una propuesta de aplicación que las solvente.

[1] En 2019, el turismo generó unos ingresos totales de 176.000 millones de euros y más de 2 millones y medio de empleos [7].

3. **Análisis de objetivos y metodologías.** Se delimitan los objetivos a conseguir y la forma en la que van a llevarse a cabo.

4. **Diseño y resolución del trabajo realizado.** Se explican todos los procedimientos que han intervenido en la creación de la solución.

5. **Conclusiones y vías futuras.** Se presentan las conclusiones obtenidas y los caminos que se podrían tomar a partir del trabajo realizado.

1.2. Convenciones

Estas son algunas de las normativas de marcado que se van a emplear:

- Las citas textuales externas a la obra serán "entrecomilladas" y escritas en letra *cursiva*.
- Las palabras extranjeras se remarcarán en *cursiva*.
- Cuando sea posible expresar un término con siglas, se escribirá el nombre completo junto a sus siglas en parentesis la primera vez que aparezca. A partir de ese momento, cada vez que se mencione se hará uso de las siglas.
- La **negrita** será usada para facilitar la lectura de textos en los que se incluyan conceptos nuevos.
- Se subrayarán palabras que representen un enlace a un recurso externo a la memoria, por ejemplo, a una página web.

2. Estado del arte

Esta sección incluye un estudio de las tecnologías empleadas actualmente para resolver el problema planteado. En este análisis, se harán visibles los puntos fuertes y débiles de dichas tecnologías. Con ellos, se justificará la necesidad del proyecto y el alcance que deberá tener.

Antes de comenzar, será conveniente aclarar una serie de términos. En primer lugar, ¿cuál es la naturaleza de las tecnologías que se van a revisar? Estas tecnologías forman parte de lo que se conoce como un "sistema de información turística". Estos sistemas han sido descritos por Bigné como "*un proceso de recopilación, tratamiento, ordenación y distribución de la información precisa para los objetivos de planificación, de acción y de evaluación turística para los distintos agentes turísticos públicos y empresariales de un destino*" [4].

Como se deduce de la definición, en un sistema de este tipo se pueden agrupar desde oficinas de turismo, hasta páginas *web* destinadas a los visitantes, en las que se va a centrar la atención. Gayete en su Trabajo Fin de Grado (TFG), en el que trata los sistemas de información turística, realiza un análisis de una de estas páginas *web* de turismo [13]. En él establece una serie de características que debería poseer el sitio para ser catalogado como excelente:

- Sitio *web* turístico propio.
- Tiene un logotipo o marca del destino.
- Posee una breve descripción con los contenidos alojados en el sitio.
- Información sobre la historia, cultura, gastronomía y atractivos turísticos.
- Enlaces de interés.
- Variedad de idiomas.
- Fotografías de alta calidad.
- Accesibilidad a información de interés turístico.
- Fácil comunicación con los usuarios.

Si se tiene como objetivo la construcción de una infraestructura *web* adaptada al turismo, concluye que los puntos anteriores deben ser alcanzados. De esta forma, teniendo en cuenta las propiedades enumeradas, se va a proceder con la crítica a la *web* oficial de turismo de la Región de Murcia [17].

2.1. Crítica al estado del arte

La página se encuentra principalmente enfocada a aquellas personas que deseen viajar a la Región de Murcia. Cuenta con un logotipo propio y una guía en su página principal sobre los contenidos alojados en el sitio. Al mismo tiempo, se muestran fotografías de alta calidad de los lugares más destacados que posee la comunidad. Por otro lado, métodos de contacto son encontrados tanto en una sección independiente como en el pie de la página. En ellos se encuentran enlaces a redes sociales, direcciones de correo y números de teléfono para la resolución de dudas.

En general, la *web* cumple con la mayoría de los requisitos expuestos en la sección 2. Sin embargo, salen a la luz varios problemas al consultar la información de los lugares de interés, especialmente en otro idioma.

Cuando se cambia el idioma de la página, la experiencia de usuario se ve alterada. En inglés se comporta de forma semejante a como lo hace en castellano, pero en idiomas como el alemán o el francés no se muestra traducida la información sobre los emplazamientos. Esta cuestión se agrava en el resto de idiomas, ya que redirigen al usuario a otra página distinta en funcionalidad e información a la principal. Estas páginas no poseen apartados que provean de información sobre los lugares destacados, únicamente información general y de contacto. Es más, en casos como el ruso y el checo, el diseño no está acutualizado al de una página *web* actual.

El proceso de búsqueda de información de lugares de interés turístico puede ser complicado. La accesibilidad es muy importante a la hora de facilitar datos sobre el lugar en el que se encuentra el turista en ese momento, más aún teniendo en cuenta que en ese momento solamente dispondrá de su dispositivo móvil. Para este inconveniente se aplicaron soluciones alejadas del ámbito de las nuevas tecnologías.

La figura 2.1 muestra uno de los muchos carteles que se encuentran situados cerca de los lugares emblemáticos de la ciudad. En ellos se indica el nombre y, de forma resumida, la historia del sitio. Se trata de una técnica que permite situar a los turistas en la historia de la ciudad de una manera clara y concisa, aunque tienen una serie de desventajas:

- Al estar situados a pie de calle, no sólo están expuestos a los elementos, sino a actos vandálicos como las pintadas del cartel localizadas en la figura 2.1.
- La existencia de erratas en el texto o de información desactualizada.
- La escasez de idiomas que se pueden ofrecer. Algo que se agrava al comprobar la procedencia de turistas en los últimos años[1].

[1]Según los datos recogidos por el Instituto Nacional de Estadística (INE) en 2019 [16], los países predominantes en cuanto a número de visitantes extranjeros en la Región de Murcia fueron: Reino Unido, Francia, Italia, Alemania, Portugal, Países Bajos, Belgica y Rusia

Figura 2.1: Cartel turístico del campus universitario de la Merced

- El código *QR* que se encuentra justo debajo del nombre del cartel no referencia a una página *web* con más información del lugar, sino a la página principal de turismo.

De estas desventajas se derivan otros problemas como: el alto coste de mantenimiento de los carteles, la insatisfacción del turista, el empeoramiento de la imagen de la ciudad, etc.

2.2. Propuesta

De cara a mejorar la experiencia del turista es importante que exista una infraestructura con un nivel de calidad acorde a las necesidades de sus usuarios. También hay que tener en mente al ayuntamiento encargado de suministrar la información, para que cuenten con herramientas que les faciliten ese proceso de incorporación y actualización.

La propuesta que se presenta en este documento es la de una aplicación *web* con dos facetas. Una de ellas es la encargada de llevar la información de interés turístico al público. Deberá adaptarse a cualquier tipo de dispositivo y cumplir con todas las guías de accesibilidad, de manera que pueda ser usada por cualquiera. En un principio, al sitio se accederá mediante carteles con códigos *QR* situados en los lugares de interés,

algo que contrarresta las desventajas del cartel de la figura 2.1, ya que:

- Pueden situarse a una mayor altura, algo que los prevendrán de ser dañados.
- Al mostrarse la información en la *web*, esta puede ser actualizada y corregida sin ningún coste adicional.
- La página *web* ofrece los diferentes idiomas disponibles en los que se puede visualizar la información.

La otra faceta está dirigida a las personas encargadas de modificar la información. La aplicación web estará compuesta por las herramientas necesarias para llevar a cabo las modificaciones, además de incorporar una distribución basada en roles, la cual se detalla en la sección 4.

La forma en la que se ha planteado la propuesta da con una herramienta capaz de solventar muchas de las deficiencias encontradas en la sección 2.1. Pero esta solución no solo es aplicable al caso de la Región de Murcia, otras muchas comunidades, e incluso países, pueden beneficiarse de este proyecto y sacar a la luz su riqueza cultural e histórica.

2.3. Tecnologías y herramientas

Como último apartado en esta sección, se van a describir las tecnologías que van a ser empleadas para el desarrollo de la apliación descrita en la sección 2.2. Además, se decidirán las herramientas adecuadas para el uso de las tecnologías elegidas.

2.3.1. Cliente

El lado del cliente, o *front-end*, en el contexto de un producto *software* se refiere a las interfaces de las que va a hacer uso el cliente a la hora de emplear el programa. En el caso de las apliaciones web, son empleados principalmente tres lenguajes para su funcionamiento:

- *HyperText Markup Language* (HTML) es un lenguaje de marcado que existe desde la creación de la web por Tim Berners-Lee [39]. Es empleado para indicar los elementos de los que se compone un sitio.
- *Cascading Style Sheets* (CSS) es un leguaje de estilos destinado a descrbir la presentación de documentos HTML [22].
- *JavaScript* (JS) es un lenguaje de programación interpretado que aplica lógica a los sitios web. Desde su creación en 1995 por Brendam Eich, ha ganado popularidad hasta posicionarse como uno de los tres pilares en el desarrollo web [1].

Hoy en día se han creado numerosas infraestructuras que emplean estos tres lenguajes y facilitan las labores de desarrollo. En el proyecto va a emplearse *React*, una biblioteca de JS creada por *Facebook* para la construcción de interfaces de usuario. Ofrece características útiles como: la unión de HTML y JS mediante JSX o la actualización inmediata de componentes con el DOM Virtual [10]. Además, es uno de los *frameworks* más usados por los desarrolladores web[2]. Cabe mencionar que en lugar de JS se ha empleado *TypeScript* (TS)[3] para facilitar la creación de código libre de errores.

Se va a usar *Bootstrap* [5], una biblioteca de componentes ampliamente utilizada. Está provista de una batería de componentes que permite dar un aspecto agradable a la interfaz de una forma sencilla y sin el uso directo de CSS. Otra razón de su elección es el diseño que se ha aplicado en los componentes para que sigan los estándares de accesibilidad. De manera adicional, se ha empleado la librería de iconos *Bootstrap Icons* [6]

A la hora de mantener el estado de la aplicación, se ha seleccionado *Redux* [8]. Contiene funciones que facilitan la interacción de los componentes de la interfaz con los datos. De manera adicional, ha sido necesario el uso de *Redux-Saga* [31], para la correcta ejecución de las llamadas al servidor y de otros efectos secundarios que *Redux* no es capaz de manejar.

Otras tantas tecnologías, no tan grandes como las anteriores, han facilitado el desarrollo. Estas son algunas de ellas: *Axios*, *Formik*, *Yup*, *React Helmet Async*, *React Markdown Editor* y *React Select*.

2.3.2. Servidor

El lado del servidor, o *back-end*, tiene que ver con los procesos que aplican la lógica al *front-end*. Estos procesos consiguen enviar la información introducida por el usuario para ser tratada, o bien, mostrar los datos solicitados por el usuario en la interfaz.

Muchos lenguajes a lo largo de los años han sido empleados para realizar esta tarea: *Java*, *PHP* o *Ruby*; sin embargo, al igual que ocurre con el lado cliente, el *back-end* va a ser escrito en TS. Para llevarlo a cabo se va a usar *Node.js*, un entorno de ejecución de JS [4] destinado a eventos asíncronos [26]. Además, *Node.js* se utiliza mediante un *framework* que simplificará y acelerará el desarrollo, este es *Express* [25].

Por otro lado, la información deberá guardarse en una base de datos. Se ha elegido *PostgreSQL* [35], un tipo de base de datos relacional que, después de *MySQL*, es una de las más populares[5].

[2]Según la encuesta de 2020 de *Stack Overflow*, *React* se encuentra en segunda posición como infraestructura web más utilizada entre los profesionales [33].

[3]Es una evolución de JS desarrollada por *Microsoft* y cuya mayor diferencia es la implementación de tipos [20].

[4]También compatible con TS.

[5]Según la encuesta de 2020 de *Stack Overflow*, *PostgreSQL* se encuentra en segunda posición como base de datos más utilizada entre los profesionales [32].

Las consultas a la base de datos y ejecución de operaciones CRUD se han realizado usando *TypeORM* [37]. Provee funciones y una configuración con la que la comunicación con la base de datos se efectua de forma simple.

La definición del *Application Programming Interfaces* (API) que ofrece el servidor al cliente se ha conseguido empleando *tsoa* [36]. Esta tecnología permite, esencialmente, la creación y protección de rutas a un API. Tanto su uso, como la documentación de las rutas con *Swagger*, son muy simples.

En la seguridad del *back-end* han sido necesarias varias tecnologías. Por un lado, *JWT* [3], un estándar basado en la generación de *tokens*, usados en el caso de este proyecto para la autenticación de las sesiones de los usuarios. Por otro lado, *bcrypt* [18], una librería de *Node.js* que ofrece funcionalidades para la encriptación de datos. Se ha empleado con el fin de almacenar datos sensibles[6] en la base de datos.

2.3.3. Herramientas

La pieza de software más importante en el desarrollo es el editor de código, que en este caso es *VSCode*. Su principal punto fuerte es el uso de extensiones para aumentar su funcionalidad [21]. Varias de las tecnologías mencionadas en las secciones 2.3.1 y 2.3.2 poseen extensiones para el editor.

De cara a automatizar la producción de código sin errores y bien estructurado se han usado dos herramientas. La primera es *ESLint*, un analizador de código que avisa al programador de la presencia de errores en el código [24]. La segunda herramienta es *Prettier*, un formateador de código altamente customizable [29].

Para la creación de pruebas se usa *Jest*, una infraestructura enfocada a la producción de pruebas unitarias. Cuenta con soporte para ser usado tanto en *React*, como en *Node.js* [11]. También ha sido necesario *SuperTest* [38], que simplifica realizar pruebas con llamadas a servidores.

Las pruebas iniciales al *back-end* se han llevado a cabo mediante el uso de *Postman*. Es un programa que permite realizar llamadas a servidores para comprobar su funcionamiento [28].

Por otro lado, ha sido necesario la utilización de *Docker*. Con *Docker* se hace posible la creación de contenedores, que abstraen la puesta en marcha de la aplicación web de la máquina en la que se ejecuta [9].

En cuanto a la planificación de tareas, al recuento de tiempo y a la realización de la memoria, se han usado distintas tecnologías. Como paquete ofimático se ha optado por *Google Workspace*, un conjunto de programas que ha facilitado: la escritura de borradores del TFG, el recuento del tiempo empleado y la creación de la presentación [14]. También destaca *Trello*, una herramienta que incorpora tableros *Kanban* para la organización de tareas [2]. Por último, el uso de LaTeX, un sistema de composición de texto empleado para presentar esta memoria [34]. En concreto, se ha empleado la plantilla adaptada por Pablo Rocamora [27].

[6] Como es el caso de las contraseñas de los usuarios.

El diseño de la interfaz de usuario se ha realizado con el programa *Figma*, una herramienta destinada a la creación de *mockups* de interfaces [12].

diagrams.net es una herramienta empleada para la creación de diagramas. Con ella se han realizado todos los diagramas incluidos en este documento.

Los archivos base del proyecto y su configuración inicial han sido generados con el conjunto de herramientas *Nx* [23]. Se especializa en la creación de *monorepos*, es decir, de proyectos que albergan su *front-end* y su *back-end* en una misma carpeta.

Por último, cabe mencionar al sistema de control de versiones *Git*, y más concretamente a *GitHub*, donde se han almacenado todos los recursos empleados para la creación de este TFG.

3. Análisis de objetivos y metodología

Este apartado aclara los objetivos alcanzables en este trabajo. A partir de ellos, se desarrollan las tareas pertinentes para resolverlos y se planifican con el tiempo disponible para la realización del proyecto.

3.1. Objetivos

El objetivo de este TFG es el desarrollo de una herramienta *web*, que permita la gestión y acceso a una base de datos con información sobre los lugares históricos o turísticos de una ciudad. Este objetivo engloba una serie de subobjetivos específicos:

1. Estudio del dominio de los sistemas de información turística.
2. Estudio de las tecnologías y herramientas para el desarrollo de aplicaciones *web*.
3. Análisis del sistema y planteamiento de requisitos.
4. Diseño del modelo de datos.
5. Diseño de la aplicación: *front-end* y *back-end*.
6. Desarrollo de la aplicación: *front-end* y *back-end*.
7. Validación de la aplicación en un entorno de pruebas.
8. Documentación del desarrollo de la aplicación.

Cabe mencionar que este trabajo persigue la creación y documentación de un prototipo, no se pretende crear una versión final. Sin embargo, se detallarán posibles características o mejoras que podrían incluirse en el proyecto en la sección 5.

3.2. Metodología

Una vez establecido el objetivo general, es momento de establecer un plan para alcanzarlo, por lo que se deben estipular las tareas que permitan conseguir cada uno de los subobjetivos. En este trabajo se ha perseguido la consecución de las siguientes tareas:

1. **Análisis preliminar**
   Objetivo(s): 1, 2
   1.1. Estudiar el dominio de los sistemas de información turística.
   1.2. Estudiar el diseño de otras aplicaciones similares a esta.
   1.3. Estudiar las tecnologías existentes para el desarrollo del *front-end*.
   1.4. Estudiar las tecnologías existentes para el desarrollo del *back-end*.
   1.5. Estudiar las herramientas y librerías que se emplearán en el desarrollo.

2. **Análisis y diseño del sistema**
   Objetivo(s): 3, 4, 5
   2.1. Definir alcance del sistema.
   2.2. Establecer los requisitos funcionales y no funcionales del sistema.
   2.3. Diseñar el modelo del dominio.
   2.4. Diseñar el componente *back-end* de la aplicación.
   2.5. Diseñar el componente *front-end* de la aplicación.

3. **Desarrollo de la aplicación**
   Objetivo(s): 6
   3.1. Crear del esquema de la base de datos.
   3.2. Desarrollar los servicios en el *back-end*.
   3.3. Desarrollar el *front-end*.
   3.4. Conectar el *front-end* y el *back-end*.

4. **Verificación y validación de la aplicación**
   Objetivo(s): 7
   4.1. Verificar la aplicación frente a los requisitos especificados.
   4.2. Definir los casos de prueba.
   4.3. Poblar la base de datos con conjunto de datos necesarios para las pruebas.
   4.4. Validar el sistema a partir de los casos de prueba definidos.

5. **Documentación del sistema**
   Objetivo(s): 8
   5.1. Reuniones periódicas con el tutor para revisar el estado del TFG.
   5.2. Documentar internamente cada uno de los componentes software desarrollados.
   5.3. Elaborar la memoria del TFG.

5.4. Elaborar la presentación del TFG.

Estas tareas han sido planificadas de acuerdo con el diagrama de Gantt que se muestra en la figura 3.1. Se da comienzo con el análisis preliminar, en el que se estudian las soluciones aplicadas en la actualidad para resolver el problema. A continuación, se pasa con el análisis y diseño del sistema, mientras se estudian las tecnologías adecuadas en la creación del proyecto. Se sigue con el desarrollo y prueba de la aplicación. Por último, la documentación del sistema será una tarea llevada a cabo en paralelo a las anteriores.

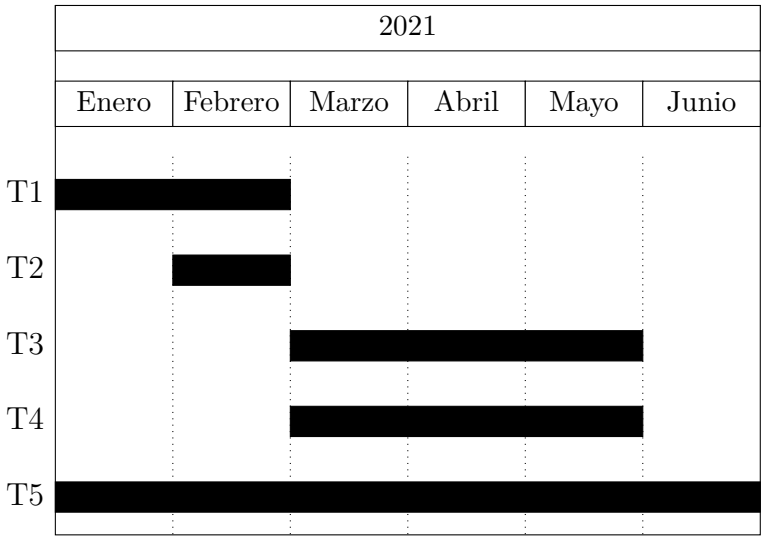

Figura 3.1: Diagrama de Gantt con la planificación de las tareas

3.2.1. Tiempos

A lo largo del proyecto, se han ido recopilando datos sobre el tiempo dedicado a las diferentes tareas mencionadas en la sección 3.2. Este tiempo se ha controlado y organizado teniendo en cuenta la tarea y el día. En este apartado se presentan los tiempos y con ellos se comprueba si la planificación realizada se ha seguido.

Tarea	Descripción	Horas estimadas	Horas reales
T1	**Análisis preliminar**	**16**	**12**
1.1	Estudiar sistemas de información turística	3	2,5
1.2	Estudiar estado del arte	3	2
1.3	Estudiar tecnologías front-end	4	2,5
1.4	Estudiar tecnologías back-end	4	4
1.5	Estudiar herramientas y librerías	2	1
T2	**Análisis/diseño del sistema**	**40**	**37**
2.1	Definir alcance del sistema	5	3
2.2	Establecer requisitos	10	11,25
2.3	Diseñar la BBDD	5	6,5
2.4	Diseñar back-end	10	5,25
2.5	Diseñar front-end	10	11
T3	**Desarrollo de la aplicación**	**155**	**145,75**
3.1	Crear esquema de la BBDD	20	2
3.2	Desarrollar el back-end	40	36
3.3	Desarrollar el front-end	80	98,75
3.4	Conectar back-end y front-end	15	9
T4	**Verificación/Validación**	**30**	**13,5**
4.1	Verificar los requisitos	4	1,5
4.2	Definir los casos de prueba	12	10,5
4.3	Poblar la base de datos	4	0,5
4.4	Validar el sistema	10	1
T5	**Documentación**	**59**	**78**
5.1	Reuniones con el tutor	10	6,25
5.2	Documentar el desarrollo	8	0,75
5.3	Elaborar la memoria	36	50,75
5.4	Elaborar la presentación	5	20,25
	Total	**300**	**286,25**

Figura 3.2: Tabla de horas estimadas y dedicadas

En la tabla 3.2 se encuentran listadas todas las tareas del proyecto. Junto a ellas se presentan dos columnas, el tiempo estimado para cada tarea y el tiempo real que se le ha dedicado.

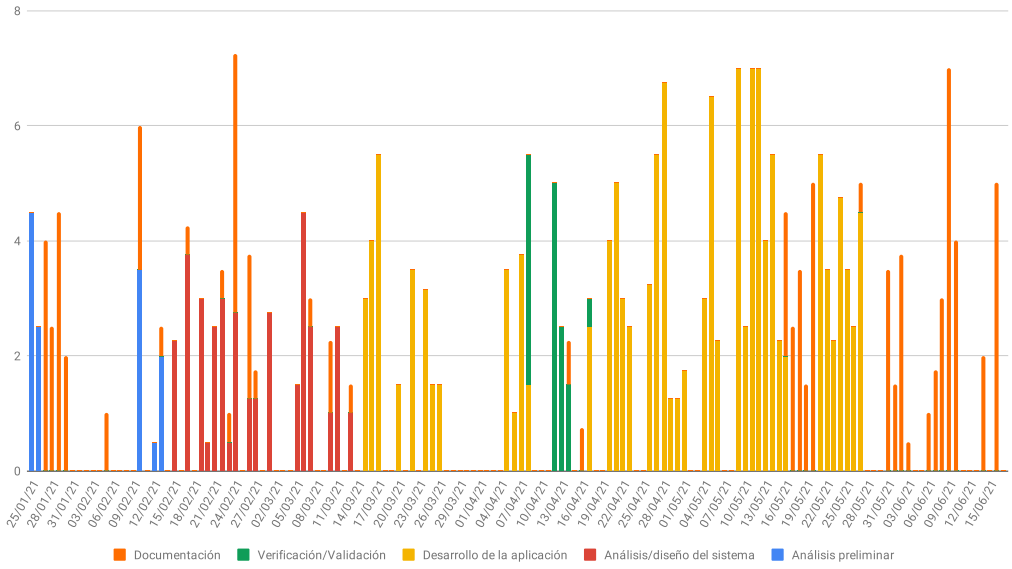

Figura 3.3: Gráfico de horas dedicadas

En el gráfico de la figura 3.3 se representan las horas dedicadas a cada tarea en la duración del proyecto. Para cada día, se diferencian las tareas a las que se les ha dedicado tiempo haciendo uso de los colores indicados en la leyenda del gráfico. De esta manera, comparando con el diagrama de Gantt de esta misma sección, se puede verificar que el plan ha sido seguido fielmente.

4. Diseño y resolución del trabajo realizado

Este es el apartado principal del TFG, donde se va a explicar todo el trabajo llevado a cabo. La sección contemplará el análisis de la funcionalidad del sistema, el diseño de sus estructuras internas, el desarrollo, las pruebas aplicadas y la documentación.

4.1. Análisis

En la sección 2.2 se dió una explicación inicial del sistema. En este apartado, se va a dar una versión detallada de la explicación y se va a definir la funcionalidad pricipal que implementará.

4.1.1. Alcance del sistema

La aplicación web contará con páginas públicas, accedidas por los turistas mediante códigos QR y que mostrarán información relevante sobre los diferentes lugares. El idioma de la información podrá ser seleccionado por el usuario.

Por otro lado, la página también contará con un apartado destinado a los encargados de mantener la información de los sitios. Estos encargados poseen uno de los siguientes roles: **administrador** o **redactor**.

El **administrador** se encarga de las tareas de gestión y organización. Administra las cuentas de los usuarios, determinando para cada uno el rol que va a desempeñar en el sistema. Administra los **carteles**, que en este contexto serían los objetos que albergan la información de cada lugar mostrada a los turistas. También se encarga de gestionar la solicitud, cancelación y confirmación de **trabajos**. Los **trabajos** son peticiones que hace el administrador a un **redactor** para la modificación de información de un **cartel**. Estas solicitudes, una vez completadas, deben ser aprobadas por el **administrador** para que se hagan visibles en el **cartel**.

El **redactor** es el usuario encargado de la inclusión de datos históricos en el idioma que se le haya asignado. Como usuario, realizará los **trabajos** solicitados por el **administrador** con herramientas de edición de texto incluidas en la página web.

4.1.2. Casos de uso

A partir del alcance que se ha establecido del sistema se va a extraer la funcionalidad requerida. Mediante los casos de uso, no solo se dan explicaciones de la funcionalidad, sino que se establecen: las condiciones necesarias para que se den, las consecuencias, los actores involucrados, etc. Toda esta información será de ayuda en las labores de desarrollo, sin embargo, en este apartado solo se incluye la descripción y diagrama de cada caso de uso. Las plantillas completas están disponibles en el anexo A. Por último, cabe mencionar que los casos de uso han sido agrupados teniendo en cuenta los actores que lo realizan y el tipo de las operaciones.

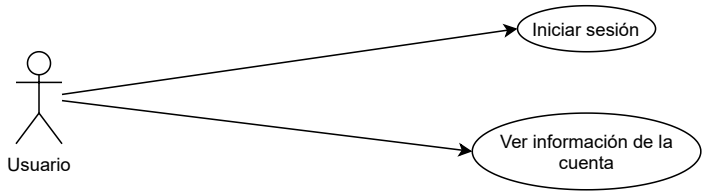

Figura 4.1: Casos de uso globales de cualquier usuario

En la figura 4.1 se muestra la funcionalidad que es capaz de realizar cualquier usuario registrado en el sistema. Estas son las descripciones de cada uno:

- **Iniciar sesión.** Un usuario con cualquier rol deberá identificarse con su apodo de usuario y contraseña para acceder al resto de las funcionalidades de la página.
- **Ver información de la cuenta.** Un usuario puede comprobar información relevante de su cuenta. Se mostrará su nombre real, su apodo de usuario, el rol que desempeña y los idiomas que domina.

Figura 4.2: Casos de uso del administrador sobre los usuarios

En la figura 4.2 se muestra la funcionalidad CRUD[1] del administrador sobre las cuentas de usuario. Esta es su descripción:

- **CRUD usuario.** El administrador controla todas las operaciones de creación, edición y eliminación de los usuarios registrados en el sistema.

[1]CRUD (*Create*, *Read*, *Update* y *Delete*) compone el conjunto de funciones básicas empleadas para el manejo de datos: crear, leer, actualizar y borrar.

Figura 4.3: Casos de uso del administrador sobre los carteles

En la figura 4.3 se muestra la funcionalidad CRUD del administrador sobre los carteles. Esta es su descripción:

- **CRUD cartel.** El administrador controla todas las operaciones de creación, edición y eliminación de los carteles creados en el sistema.

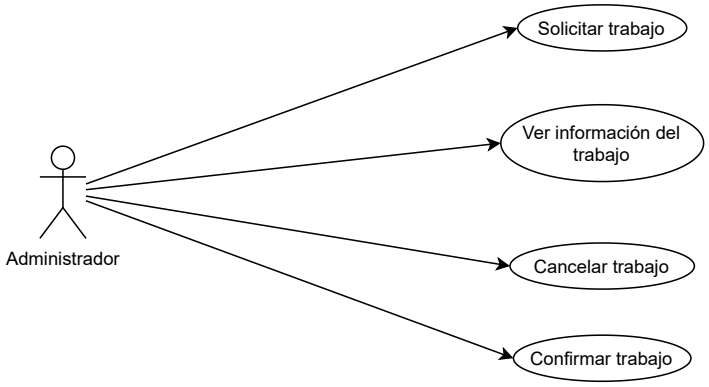

Figura 4.4: Casos de uso del administrador sobre los trabajos

En la figura 4.4 se muestra la funcionalidad del administrador sobre los trabajos. Estas son las descripciones de cada uno:

- **Solicitar trabajo.** El administrador puede solicitar la ejecución de un trabajo sobre un cartel. Deberá especificar su idioma y, en función del elegido, tendrá que seleccionar el redactor que lo llevará a cabo. Opcionalmente, puede escribir indicaciones sobre cómo se debe realizar el trabajo.
- **Ver información del trabajo.** Para cada trabajo, el administrador tiene la opción de ver información relevante. Puede visualizar el nombre, el estado actual, el redactor asignado y el idioma.
- **Cancelar trabajo.** Para cada trabajo en proceso, el administrador puede cancelarlo en caso de que no sea necesario que se lleve a cabo.
- **Confirmar trabajo.** Para cada propuesta de trabajo completado por el redactor, el administrador podrá aceptarla o denegarla. En caso de ser denegada, deberá incluirse una explicación o los cambios que se deben realizar.

Figura 4.5: Casos de uso del redactor y traductor sobre los trabajos

En la figura 4.5 se muestra la funcionalidad del redactor sobre los trabajos. Esta es su descripción:

- **Completar trabajo.** Para cada solicitud de trabajo, el redactor podrá hacer uso de herramientas de edición de texto para realizarla y enviarla al administrador.

4.2. Diseño

Una vez delimitado el alcance del sistema, se va a pasar a explicar las decisiones tomadas para el diseño de la aplicación. El diseño comprende: el modelado de las entidades que participan y de sus relaciones, la estructura de la base de datos y la arquitectura del sistema. Para facilitar la comprensión de la información se van a hacer uso de diagramas y esquemas.

4.2.1. Modelado del dominio

Un primer paso a tomar en el diseño del sistema es determinar las entidades que lo compondrán. A partir de los resultados obtenidos de la sección 4.1, se concluyen las siguientes entidades: “Usuario”, “Redactor”, “Administrador”, “Trabajo”, “Cartel”, “Cartel Idioma” e “Idioma”.

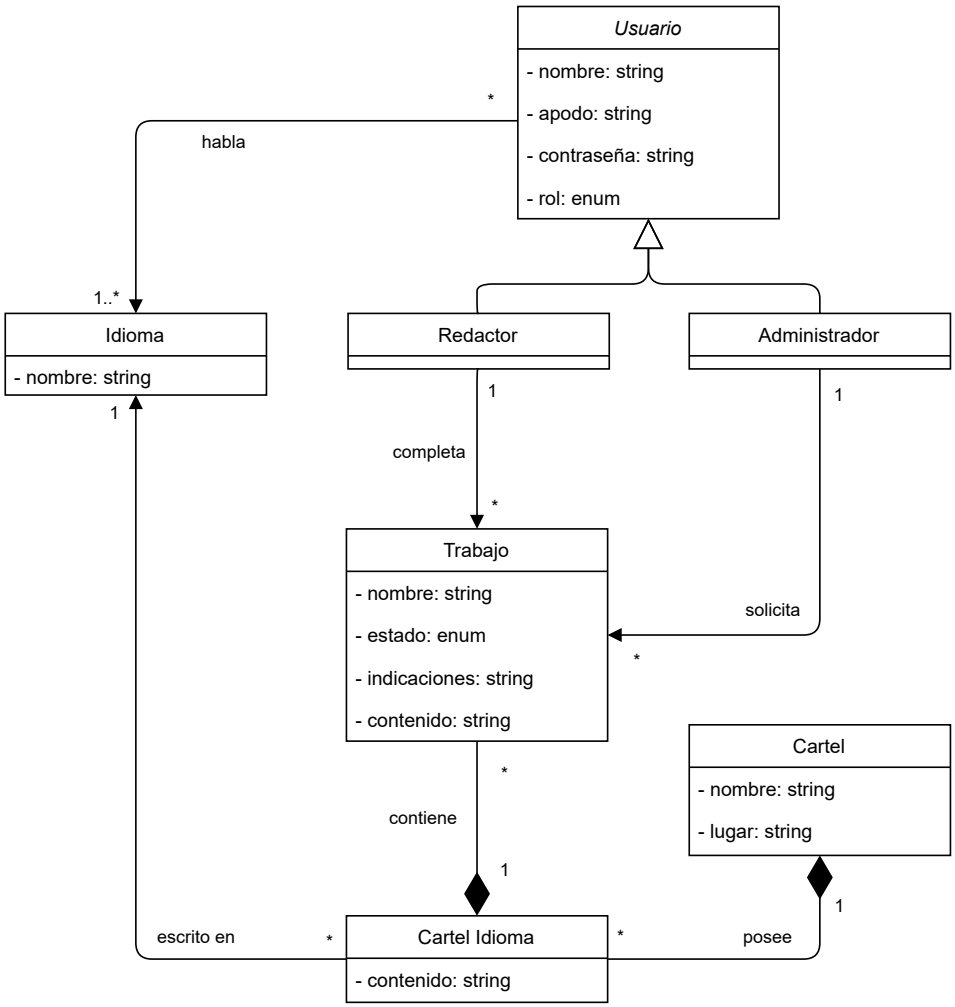

Figura 4.6: Modelo del dominio

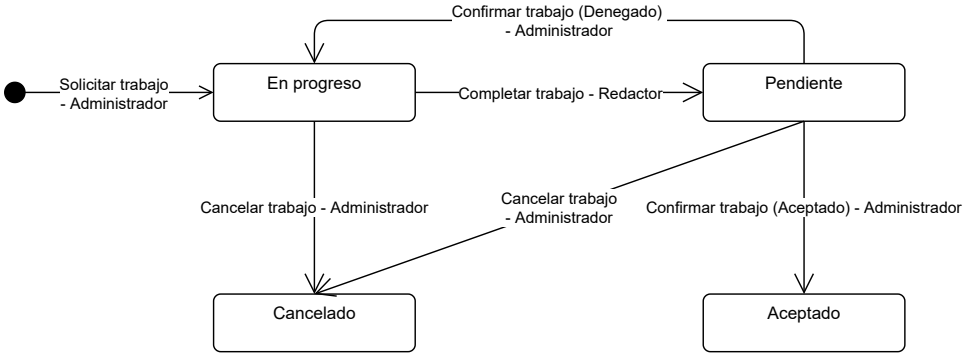

Figura 4.7: Máquina de estados de la entidad “Trabajo”

En la figura 4.6 se muestran sus atributos y relaciones. La entidad principal es "Trabajo", la cual tendrá asignada el administrador solicitante y el redactor encargado. También está asociada a "Cartel Idioma", una entidad que poseerá el contenido de un cartel en un idioma concreto, por ello estará a su vez unida a "Cartel".

Para terminar de comprender el modelo se debe centrar la atención en el estado de "Trabajo". Desde su creación, un trabajo pasa por una serie de estados dependiendo de la operación que se aplica sobre él. En la figura 4.7, haciendo uso de una máquina de estados, se muestran las diferentes fases de un trabajo en función del caso de uso ejecutado.

4.2.2. Diseño de la base de datos

La base de datos es uno de los componentes más importantes que posee la aplicación, ya que será la encargada de almancenar y recuperar todos los datos del sistema. Es por ello que precisa de un diseño adecuado, el cual albergue las entidades y relaciones obtenidas en la sección 4.2.1.

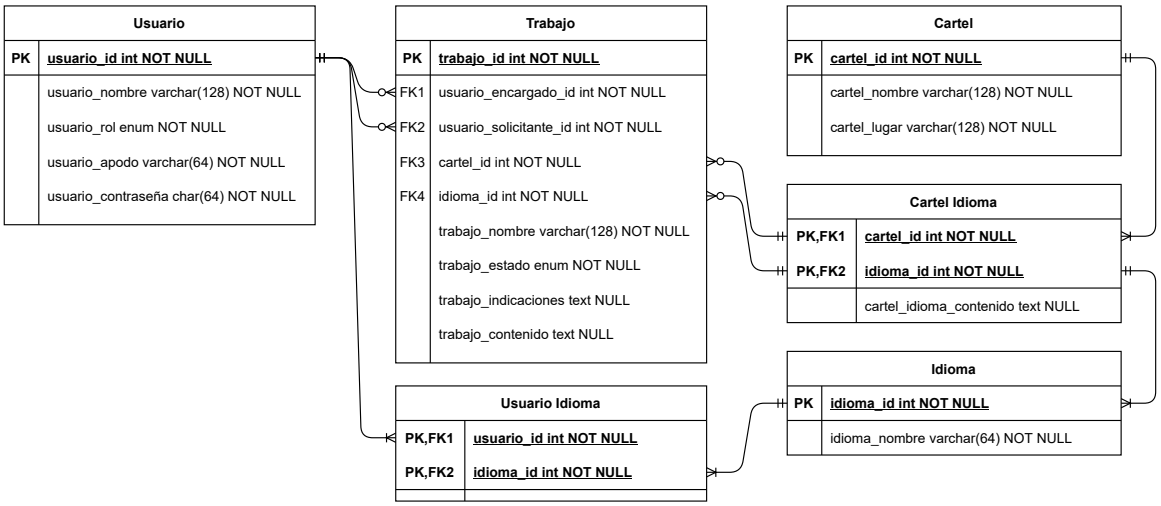

Figura 4.8: Esquema de la base de datos

En la figura 4.8 se encuentra el esquema al que se ha llegado, una base de datos de tipo relacional. Con la notación usada en el esquema se reflejan las relaciones y el resto de propiedades que posee la base de datos.

Del diagrama 4.8 cabe destacar la forma en la que se ha decidido guardar la información del cartel en distintos idiomas. El enfoque del que se ha hecho uso consiste en almacenar en la tabla "Cartel Idioma" toda la información del cartel que ha sido traducida, manteniendo en "Cartel" aquellos datos que no sean sensibles al idioma. De esta forma, el esquema se queda claro y las búsquedas son eficientes [19].

4.2.3. Diseño del *back-end*

El *back-end* será la parte del sistema encargada de satisfacer las peticiones del cliente y de manejar los datos guardados en la base de datos. Ambas comunicaciones se encuentran reflejadas en la figura 4.9.

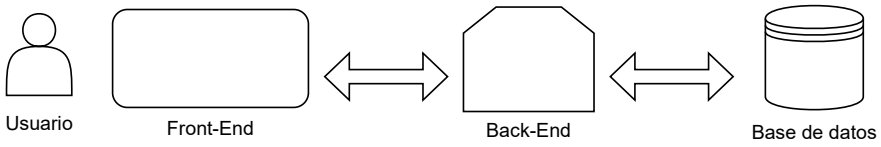

Figura 4.9: Diagrama del *back-end*

Las comunicaciones con el *front-end* van a ser posibles con el uso de la arquitectura REST (*REpresentational State Transfer*). Se basa en el empleo de solicitudes al servidor, que hace el cliente a través del protocolo HTTP (*Hypertext Transfer Protocol*) [30]. El servidor implementa operaciones acordes a las solicitudes esperadas, que ejecutarán métodos de recuperación, creación, modificación o borrado de datos. En la figura 4.1 se muestran las rutas de las operaciones disponibles, acompañadas de los usuarios que pueden acceder a ellas.

URI	GET	POST	PUT	DELETE
/auth				
/auth/login				
/usuarios				
/usuarios/:id				
/carteles				
/carteles/:id				
/carteles/:id/trabajos				
/carteles/:id/trabajos/:id				

Pública Usuarios registrados Administrador No disponible

Tabla 4.1: Rutas y restricciones del servidor

Los objetivos que tienen cada una de las rutas de la figura 4.1 son los siguientes:

- "/auth": comprobar la autenticación del usuario.
- "/auth/login": solicitar el inicio de sesión.
- "/usuarios": recuperar todos los usuarios registrados en el sistema y crear nuevos.
- "/usuarios/:id": recuperar, editar y eliminar el usuario indicado en la ruta.
- "/carteles": recuperar todos los carteles registrados en el sistema y crear nuevos.

- "/carteles/:id": recuperar, editar y eliminar el cartel indicado en la ruta.
- "/carteles/:id/trabajos": recuperar todos los trabajos registrados en el sistema y crear nuevos para el cartel indicado en la ruta.
- "/carteles/:id/trabajos/:id": recuperar y editar el trabajo solicitado para el cartel indicado en la ruta.

En general, con REST se consigue flexibilidad y robustez, ya que las peticiones al servidor no poseen estados y pueden hacerse desde una página web, una aplicación móvil o un programa de escritorio.

4.2.4. Diseño del *front-end*

Como se ha mencionado anteriormente, el *front-end* comprende la interfaz que empleará el cliente para comunicarse con el sistema. Su diseño debe estar enfocado al usuario que empleará la página, por ello se deberán tomar decisiones que favorezcan la usabilidad del sitio web.

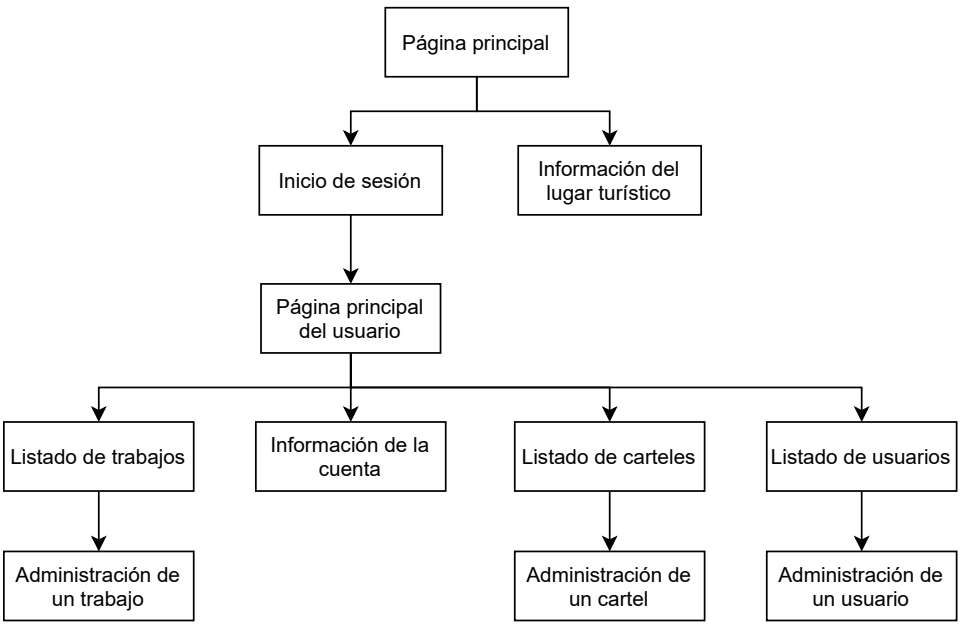

Figura 4.10: *Sitemap* de la aplicación web

Como primer paso se deben definir las distintas páginas que compondrán el sistema. En la figura 4.10 se muestra un *sitemap* o mapa del sitio web, donde se representan las interfaces existentes y la manera en la que están relacionadas.

Con las interfaces delimitadas se han realizado *mockups* de cada una. Por un lado, se ha llevado a cabo un diseño de todas las páginas para pantallas de ordenador y,

por otro lado, un diseño de las interfaces "Página principal" e "Información del lugar turístico" adaptado a móviles. Se ha tomado la decisión de recortar el alcance móvil para asegurar la obtención de un sistema funcional, adaptando únicamente las vistas que empleará el turista.

(a) *Mockup* web **(b)** *Mockup* móvil

Figura 4.11: *Mockups* de "Información del lugar turístico"

En la figura 4.11 se presentan los diseños de la ventana con la información de un lugar. El resto de páginas pueden ser vistas visitando los siguientes enlaces: diseño web[2] y diseño móvil. Estos *mockups* sirven para visualizar los componentes, su distribución y la funcionalidad encontrada en cada lugar del sitio. Además, facilitan las labores de programación, durante las cuales se completarán los diseños obtenidos añadiendo formas, sombreado, colores, etc.

4.3. Desarrollo

El desarrollo constituye la sección con más peso en la memoria y la que más tiempo ha consumido de manera general. Se va a pasar a detallar las decisiones tomadas a la hora de usar las tecnologías elegidas, y en cuanto a la forma en la que se ha estructurado el código[3].

4.3.1. Estructura del proyecto

Como se ha mencionado en la sección 2.3.3, se ha decidido guardar el código del cliente y del servidor en una *monorepo*. Razón por la cual se ha empleado el conjunto de herramientas *Nx*, las cuales simplifican las labores de creación y mantenimiento de una *monorepo*.

[2]Las páginas destinadas a un rol de usuario determinado, se indican poniendo el nombre del rol en el diseño abajo a la izquierda. Aquellas que no posean un nombre, pueden ser accedidas por cualquier rol.

[3]Todo el código del proyecto ha sido alojado en un repositorio privado en GitHub. Se puede visualizar el contenido del repositorio en este enlace.

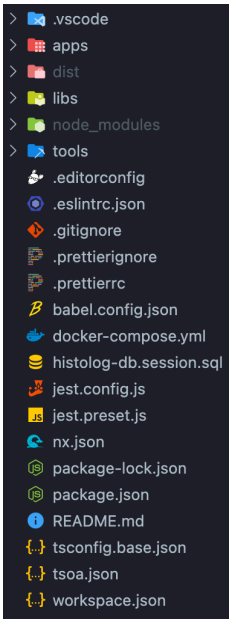

Figura 4.12: Estructura del proyecto

En la figura 4.12 se muestra la raíz del proyecto generado con las herramientas *Nx*. La mayoría de archivos que la conforman son archivos de configuración, creados por defecto al generar el proyecto. Esta es una de las muchas ventajas que ofrecen las *monorepos*, ya que la configuración definida puede ser compartida por todos los proyectos.

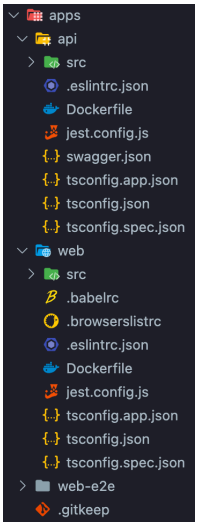

Figura 4.13: Estructura de la carpeta *apps*

Dentro de la carpeta *apps* se almacena el código del *front-end*, en el directorio *web*, y del *back-end*, en el directorio *api*, como se aprecia en la figura 4.13. Ambas carpetas

poseen el código fuente y una serie de archivos de configuración. Estos últimos se han usado en el caso de que se haya precisado una extensión de la configuración global para cada proyecto.

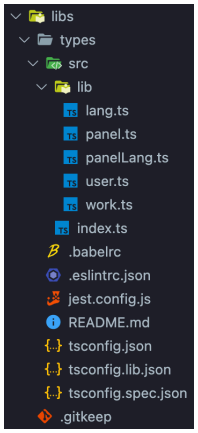

Figura 4.14: Estructura de la carpeta *libs*

Código 4.1: Interfaz del Usuario del archivo *user.ts*

```
1  export enum Rol {
2      ADMINISTRADOR = 'Administrador',
3      REDACTOR = 'Redactor',
4  }
5
6  export interface IUser {
7      id: number;
8      nombre: string;
9      rol: Rol;
10     apodo: string;
11     idiomas: ILang[];
12 }
13
14 export interface IUserLogin {
15     apodo: string;
16     clave: string;
17 }
```

Lo último destacable de los archivos de la raíz es la carpeta *libs*. Se trata de una carpeta en la que se puede crear código destinado a ser usado por cualquier proyecto del directorio *apps*. En la figura 4.14 se encuentra reflejado su contenido, que en este caso se ha empleado para almacenar tipos. En cada archivo se han definido una serie de interfaces empleadas en la comunicación entre el cliente y el servidor. De esta manera,

se consigue al mismo tiempo consistencia en ambos códigos y robustez. En 4.1 se encuentra un trozo del archivo *user.ts*, donde se exportan el enumerado de roles, el tipo de un objeto Usuario y los datos requeridos para la operación de inicio de sesión.

4.3.2. Implementación del *back-end*

El código del servidor ha sido repartido en distintas carpetas, como muestra la figura 4.15, para facilitar la comprensión del mismo. El punto de partida de la aplicación se da en el archivo *main.ts*. En él se ejecuta el servidor de *Express*[4] y se realiza la conexión con la base de datos. Tras esto, el servidor queda a la espera de la llegada de llamadas del cliente, que ejecutarán el resto de la funcionalidad.

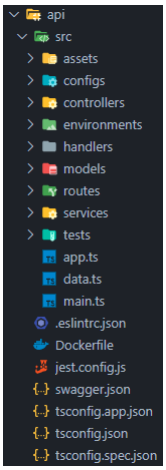

Figura 4.15: Estructura del *back-end*

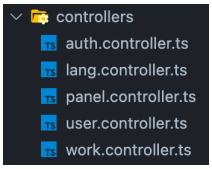

Figura 4.16: Estructura de la carpeta *controllers* del *back-end*

Estas llamadas llegan como peticiones HTTP interceptadas por la interfaz REST. Esta interfaz ha sido implementada en la carpeta *controllers* y, como se ve en la figura 4.16, se han creado distintos archivos para cada una de las rutas definidas en la sección 4.2.3. Dentro de cada archivo se han definido funciones que ejecutarán las acciones correspondientes a la operación elegida para esa ruta.

[4]Configurado en el archivo *app.ts*.

Código 4.2: Función para recuperar un usuario del controlador *user.controller.ts*

```
/**
* Devuelve los detalles de un usuario registrado.
* @param userId ID del usuario
* @returns Datos del usuario
*/
@Security({ jwt: [Rol.ADMINISTRADOR, Rol.REDACTOR],
    restriction: [] })
@Get('{userId}')
public async getUser(@Path() userId: number): Promise<IUser> {
    return this.userService.get(userId).then((user) => {
        return {
            id: user.id,
            nombre: user.nombre,
            rol: user.rol,
            apodo: user.apodo,
            idiomas: user.idiomas,
        };
    });
}
```

Al haber hecho uso de *tsoa* para crear las rutas, estos controladores incluirán una serie de anotaciones. Cada fichero contiene una clase con la anotación *@Route('path')*, donde *path* es la ruta base para acceder al resto de operaciones que se definen en la clase. En el código 4.2 se encuentra una de las funciones del archivo *user.controller.ts*. Comenzando desde el inicio, se encuentra un comentario que explica su funcionamiento, el cual será usado para generar la documentación con *Swagger*. La anotación *@Security* determina las condiciones que se deben cumplir para acceder a la ruta. En este caso, *jwt* comprobará si el usuario ha iniciado sesión y si posee uno de los roles indicados. Por otro lado, *restriction* verá si, en el caso de tratarse de un usuario que no sea administrador, la información del usuario que está intentando recuperar es la de su cuenta. Esta última medida permite evitar que aquellos usuarios sin suficientes privilegios accedan a información que no les corresponde. Siguiendo con las anotaciones, se define *@Get("{userId}")*, que establece el tipo de petición HTTP y el nombre de la ruta. Por último, con *@Path()* se pueden extraer los parámetros incluidos en la ruta, como es el caso del identificador del usuario en este ejemplo.

Código 4.3: Traducción de la recuperación de un usuario del archivo *routes.ts*

```
app.get('/api/v1/usuarios',
    authenticateMiddleware([{"jwt":["administrador"]}]),
    function UserController_getUsers(request: any,
        response: any, next: any) {
```

```
        const args = {
            lang: {"in":"query","name":"lang","dataType":"string"},
        };

        let validatedArgs: any[] = [];
        try {
            validatedArgs =
                getValidatedArgs(args, request, response);
        } catch (err) {
            return next(err);
        }

        const controller = new UserController();

        const promise = controller.getUsers.apply(controller,
            validatedArgs as any);
        promisezandler(controller, promise,
            response, undefined, next);
    }
);
```

Todas las características que se acaban de explicar de *tsoa*, junto a otras empleadas en otras funciones, consiguen acelerar y simplificar la creación del *back-end*. Esto se debe a que *tsoa*, a través de la ejecución de un comando, genera un archivo[5] con la declaración de todas las rutas mediante funciones de *Express*. La traducción del código 4.2 anterior será la del código 4.3. Este fichero exporta una función que se añade al servidor de *Express* para que haga uso de las rutas generadas.

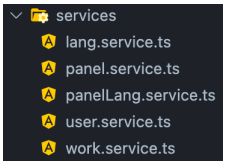

Figura 4.17: Estructura de la carpeta *services* del *back-end*

Código 4.4: Función para recuperar un usuario del archivo *user.service.ts*

```
public async get(userId: number): Promise<Usuario> {
    return await this.userRepository.findOne(userId);
}
```

En el código 4.2, dejando a un lado las anotaciones, se llama a la función encargada

[5]El fichero *routes.ts* almacenado en la carpeta *routes*.

de ejecutar la acción pedida. Esta función se encuentra definida en los archivos de la carpeta *services*, que se pueden visualizar en la figura 4.17. Este directorio contiene la lógica de negocio de la aplicación y es aquí donde se interactúa con la base de datos. Siguiendo con el ejemplo de recuperación de un usuario, en el código 4.4 se encuentra el contenido de la función que devuelve el usuario con el identificador especificado. Se trata de un código muy simple gracias a la siguiente tecnología empleada, *TypeORM*.

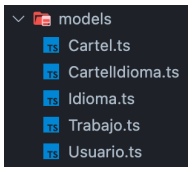

Figura 4.18: Estructura de la carpeta *models* del *back-end*

Con *TypeORM* se consigue un acceso y manejo del contenido de la base de datos muy sencillo. Haciendo uso de un archivo de configuración[6], en el que se especifica la información de conexión a la base de datos, *TypeORM* es capaz de conectarse y ofrecer un catálogo de funciones con las que acceder a los datos. También genera los esquemas de la base haciendo uso, nuevamente, de anotaciones. En el código 4.5 se encuentra el modelo del usuario, guardado junto al resto de modelos en la carpeta *models*, cuyo contenido se muestra en la figura 4.18. Mediante estas anotaciones se han definido los identificadores, columnas, relaciones y demás opciones para conseguir el esquema alcanzado en la sección 4.2.2.

Código 4.5: Modelo del usuario del archivo *Usuario.ts*

```
1  @Entity()
2  export class Usuario {
3      @PrimaryGeneratedColumn()
4      id: number;
5
6      @Column('varchar', { length: 128 })
7      nombre: string;
8
9      @Column({ type: 'enum', enum: Rol })
10     rol: Rol;
11
12     @Column('varchar', { length: 64 })
13     apodo: string;
14
15     @Column('varchar', { length: 64 })
16     clave: string;
```

[6]El fichero *orm.config.ts* almacenado en la carpeta *config*.

```
17
18      @ManyToMany(() => Idioma, {
19          eager: true,
20          nullable: false,
21          onDelete: 'CASCADE',
22      })
23      @JoinTable()
24      idiomas: Idioma[];
25  }
```

De la figura 4.15 quedan algunas carpetas por mencionar. Por un lado *handlers*, en la que se han incluido archivos con funcionalidad de apoyo. Uno de los archivos es *auth.handler.ts*, donde se definen las funciones que ejecuta la anotación *@Security*, empleada por *tsoa*. En este archivo se hace uso del *token* enviado por la petición HTTP para comprobar si se trata de un usuario registrado y para conocer el nivel de privilegios que tiene en función de su rol. Por ello, en la generación del *token*[7] se almacena toda la información del usuario. De esta manera, cuando convenga, se podrá recuperar el rol del usuario que envíe el *token* y se podrá verificar si tiene los permisos suficientes para ejecutar la acción. El otro archivo se trata de *error.handler.ts* y en él se implementa el código encargado de generar los errores, en el cual se establece un código HTTP de error y un mensaje.

En el directorio *environments* de la carpeta 4.15 residen los archivos con las variables de entorno. Hay dos archivos, uno que se usa cuando la aplicación se ejecuta en un entorno de desarrollo y el otro que se emplea cuando está en producción.

Por último, la carpeta *tests* y el archivo *data.ts* contienen los archivos con las pruebas unitarias y datos de ejemplo, respectivamente. Ambos archivos se explican con profundidad en la sección 4.4.

4.3.3. Implementación del *front-end*

En la parte del cliente se ha empleado React a la hora de crear la interfaz. En la figura 4.19 aparece la estructura del código fuente que posee la aplicación *web*. El punto de entrada es el fichero *main.tsx*, el cual se va a emplear para explicar las tecnologías empleadas y las decisiones que se han aplicado.

En *main.tsx* se definen las principales rutas privadas y públicas a las que se podrá acceder, el cual ha sido posible gracias al uso de *React Router*. En el directorio *routes* de la figura 4.20 se definen todas las rutas de las que se compone la aplicación y, para cada una de ellas, se establece el componente que renderiza y los roles de los usuarios que pueden acceder a ella, entre otros parámetros.

Los componentes que ejecuta cada ruta se han repartido las tres carpetas de la figura 4.21. La primera es *pages*, en ella se almacenan las vistas principales de la aplicación

[7] Al efectuarse el inicio de sesión en el archivo *auth.controller.ts*.

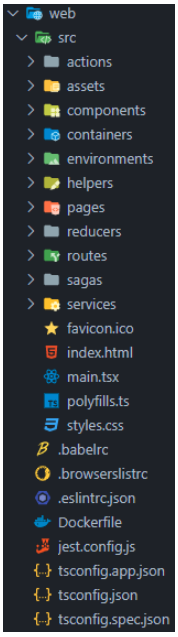

Figura 4.19: Estructura del *front-end*

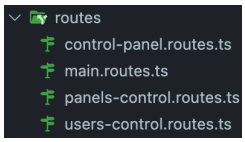

Figura 4.20: Estructura de la carpeta *routes* del *front-end*

y pueden hacer uso de otros componentes. Cabe resaltar que la página del panel de control también incluirá definiciones de otras rutas. Se ha tomado esta decisión para distinguir la parte administrativa del resto de la aplicación. La carpeta *containers* almacena todas las vistas utilizadas en el panel de control y, al igual que ocurre con los componentes de *pages*, también incluyen otros componentes. El directorio *components* tendrá componentes reutilizables empleados en toda la aplicación. Aquí se encuentran los formularios, botones, barras de navegación y demás utilidades empleadas en las vistas.

El estado de los datos usados por los componentes es manejado a través de *Redux*, cuya implementación se encuentra repartida en las carpetas de la figura 4.22. Cada uno de los componentes hace uso de las funcionalidades que ofrece *Redux*, bien para recuperar los estados de los objetos que usan, o bien a la hora de ejecutar las acciones apropiadas cuando sucede algún evento. De manera adicional, ha sido necesario emplear *Redux-Saga*, ya que las ejecuciones asíncronas no están soportadas por defecto en *Redux*. Este hecho no haría posible las llamadas al *back-end*. Se va a hacer uso de un ejemplo para ilustrar el funcionamiento y la forma en la que se han implementado estas dos tecnologías.

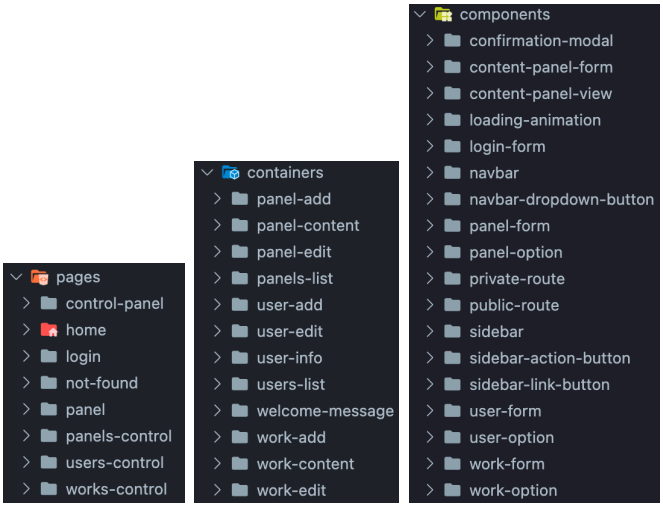

Figura 4.21: Estructura de las carpetas *pages*, *containers* y *components* del *front-end*

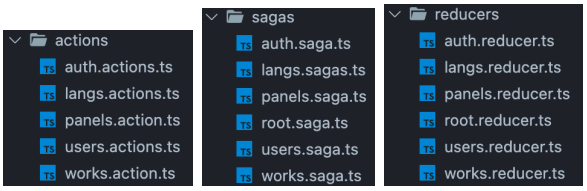

Figura 4.22: Estructura de las carpetas *actions*, *sagas* y *reducers* del *front-end*

Código 4.6: Componente del listado de usuarios del archivo *users-list.tsx*

```
1 const dispatch = useDispatch();
2
3 const users: IUser[] = useSelector(usersSelectors.users);
4 const usersTransaction: Transaction =
5     useSelector(usersSelectors.transaction);
6
7 useEffect(() => {
8     dispatch(usersActions.getAll());
9 }, [dispatch]);
```

Al igual que con la sección 4.3.2, el ejemplo va a consistir en la recuperación de todos los usuarios. Se da comienzo a esta acción en el componente con el listado de usuarios[8], en él se ejecuta el código 4.6. Como se ve, primero recupera el estado de los objetos de la lista de usuarios y de la transacción de la petición. Este último objeto contendrá el estado de la transacción en todo momento, es decir, si está pendiente, si se ha realizado con éxito y los errores que hayan ocurrido si ha fallado la ejecución. Tras esto, la función *useEffect* ejecutará el código en su interior una vez se muestre el

[8]Componente *users-list* de la carpeta *containers*.

componente. En él se dispara la acción para recuperar todos los usuarios[9].

Código 4.7: Función *handleGetAll* del archivo *users.saga.ts*

```
function* handleGetAll(): Generator {
    try {
        const response: IUser[] = yield* call(usersService.getAll);
        yield put(
            usersActions.getAllResponse({
                error: undefined, result: response
            })
        );
    } catch (error) {
        yield put(
            usersActions.getAllResponse({
                error: error.message, result: undefined
            })
        );
    }
}
```

Una vez se solicite la acción, toma el control *Redux-Saga*. Este se va a encargar de ejecutar la parte asíncrona que comprende la llamada a la API. En el código 4.7 está definida la función *handleGetAll*[10], que recuperará la lista de usuarios del *back-end* y activará la acción *getAllResponse* con los resultados de la llamada al servidor. Todas las llamadas al *back-end*, incluida la del código 4.7, han sido definidas en la carpeta *services*, donde ha sido necesario el uso de *axios*.

Código 4.8: Función *usersReducer* del archivo *users.reducer.ts*

```
case UsersActionType.GetAll:
    return { ...state, transaction: Transaction.start() };
case UsersActionType.GetAllResponse:
    if (action.payload.error) {
        return {
            ...state,
            transaction: Transaction.finish(action.payload.error),
        };
    } else {
        return {
            ...state,
            users: action.payload.result,
```

[9]Definida en el archivo *users.actions.ts* de la carpeta *actions*.
[10]Definida en el archivo *users.saga.ts* de la carpeta *sagas*.

```
13              transaction: Transaction.finish(),
14          };
15      }
```

A la misma vez que se estaba dando la ejecución anterior de *Redux-Saga*, se seguía el orden natural de *Redux*. Una vez activada la acción *getAll*, se ejecuta el primer *case* del trozo de código 4.8, que se encuentra en la función *usersReducers*[11]. En él simplemente se informa que la transacción ha comenzado. Más tarde, una vez *Redux-Saga* acciona *getAllResponse* con la respuesta de la API, se vuelve a ejecutar la función *usersReducers*, pero esta vez el segundo *case*. En este caso, se vuelve a actualizar el estado de la transacción y se modifica el valor de los usuarios, siempre y cuando la petición al *back-end* haya sido exitosa.

4.4. Pruebas

Durante el desarrollo, especialmente el del *back-end*, las pruebas han sido muy necesarias para conseguir un correcto funcionamiento de la aplicación. Al comienzo, cuando se empezaron a implementar las primeras rutas, se emplearon pruebas manuales. Estas eran sencillas y permitían conocer la existencia de errores rápidamente. Una vez el proyecto avanzó y se crearon la mayoría de las rutas, se pasó a la creación de pruebas unitarias, que facilitaron la inclusión de nuevas características sin miedo a la aparición de errores. A continuación, se detallan las herramientas que han sido usadas.

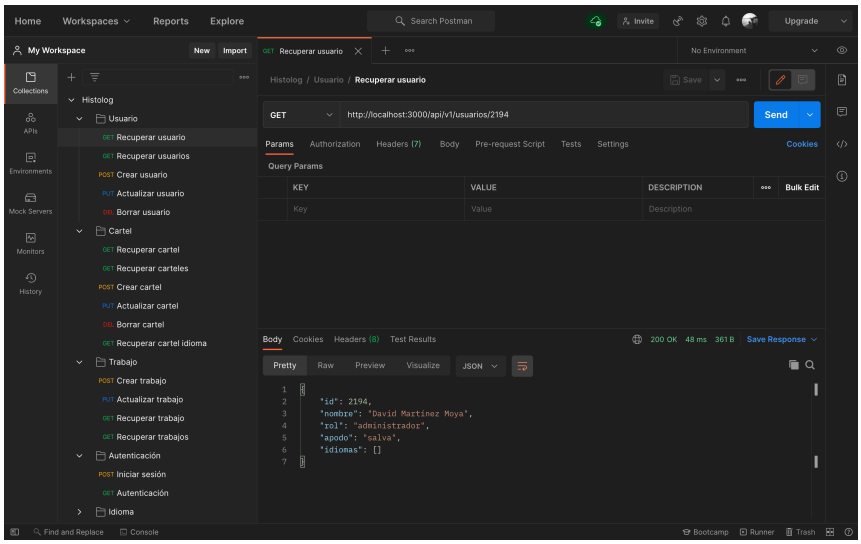

Figura 4.23: Espacio de trabajo de *Postman*

Para las pruebas manuales ha sido empleada la herramienta de *testing* de API REST

[11]Definida en el archivo *users.reducer.ts* de la carpeta *reducers*.

Postman. Como aparece en la figura 4.23, en la parte izquierda se ha organizado el espacio de trabajo incluyendo en la carpeta *Hitolog* otras carpetas relacionadas con el tipo de entidad que participa en las peticiones que contiene. A la derecha se muestra un ejemplo de petición, en este caso el de recuperar un usuario, donde se muestra el resultado de la petición abajo y los parámetros de la petición en la parte superior.

Por otro lado, las pruebas unitarias han sido posibles gracias al uso conjunto de *Jest* y de *SuperTest*. Con la primera tecnología se han definido cada una de las pruebas, la segunda ha hecho posible la creación de peticiones al servidor. Estas pruebas verifican el funcionamiento de la aplicación al hacer llamadas al servidor, es decir, automatizan las pruebas manuales, que hasta este momento se hacían con *Postman*. Los archivos con las pruebas se han almacenado en la carpeta *tests* del proyecto *api* 4.15. También ha sido necesario la creación del archivo *data.ts*, que contiene datos de ejemplo y que está localizado en el proyecto *api*.

4.5. Despliegue

De cara a un posible despliegue público de la aplicación, se han creado diferentes archivos de configuración para ejecutar todas las aplicaciones necesarias de manera que se obtenga un sistema funcional. Para ello, se ha empleado la tecnología *Docker*, con la cual se puede replicar el entorno favorable para la ejecución de cada aplicación en cualquier máquina. De esta forma, se han creado contenedores que ejecutan el *back-end*, el *front-end* y la base de datos *PostgreSQL*. Los dos primeros contenedores han sido configurados en ficheros *Dockerfile*, cada uno localizado en las raíces de su proyecto correspondiente. Y se han unido los dos proyectos, junto a la base de datos, en el archivo *docker-compose.yml*, guardado en la carpeta raíz de la *monorepo*.

Con toda esta configuración se consigue arrancar todo el sistema mediante el uso de un solo comando. Además se asegura su correcto funcionamiento en cualquier máquina. Por último, es necesario señalar la utilidad que ha tenido *Docker* durante el desarrollo, con el despliegue del contenedor de la base de datos.

4.6. Documentación

Las peticiones que ofrece el servidor han sido documentadas mediante el uso de *Swagger*. Como se muestra en la figura 4.24, esta herramienta documenta todas las operaciones de las que se pueden hacer uso, en cada una de ellas informa del tipo de datos de entrada y de salida. Además, permite realizar peticiones a cada una de las rutas para probarlas.

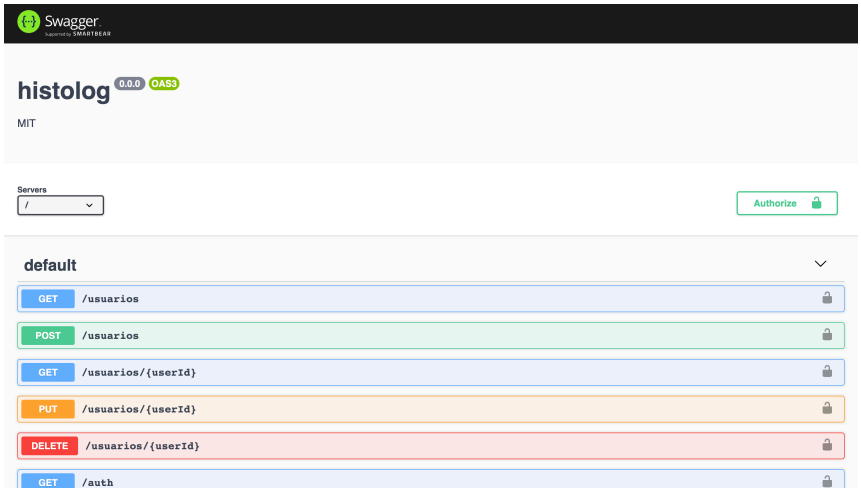

Figura 4.24: Pantalla de inicio de *Swagger*

5. Conclusiones y vías futuras

5.1. Conclusiones

Después de todas las etapas acontecidas a lo largo del trabajo, se puede considerar que se ha alcanzado un producto similar al especificado. Se ha conseguido crear una aplicación *web* que cumple con la propuesta establecida en la sección 2.2.

Se ha desarrollado una interfaz simple, usable y accesible para los turístas y los encargados de gestionar la información del sistema. Se ha tratado de suministrar todo tipo de ayudas a los usuarios de la aplicación, de manera que el uso de la misma sea satisfactorio. Se han marcado todas las consecuencias de las acciones que los usuarios pueden hacer sobre la interfaz, denotando los diferentes errores que pueden surgir durante su uso. Se ha aplicado un diseño adaptable de la interfaz, para que pueda ser usada en cualquier tipo de dispositivo.

Aunque el proyecto no ha sido desplegado en un entorno real, este puede ser usado como base de lo que podría ser un sistema perfectamente funcional. Más aún, al no existir *software* similar al descrito en este TFG, los diseños, modelos e implementaciones del sistema aquí descritos pueden ser tomados en cuenta como un punto de inicio de algo más grande. De esta manera, pueden surgir nuevas tecnologías similares que consigan el propósito que en este trabajo se ha hecho notar desde el inicio: hacer uso de la tecnología para buscar una nueva forma de facilitar la transmisión de la cultura y la historia que ofrece cualquier destino turístico.

5.2. Vías futuras

En cuanto a las vías futuras, se van a distinguir algunos de los objetivos que se deberían cumplir a corto plazo, para lograr un sistema funcional. También se van a discutir algunas de las nuevas características que podrían integrarse en un futuro.

Las características a mejorar del proyecto presentado son las siguientes:

- Mejorar del editor de texto de los redactores, de manera que no sea necesario el uso de *markdown*.

- Añadir barras de búsqueda a los listados de usuarios, carteles y trabajos. Empleando búsqueda perezosa para que no se recuperen a la vez todas las entidades de la aplicación.

- Mejorar la comunicación del administrador con los redactores y viceversa, en cuanto a la forma en la que se debe llevar a cabo el trabajo.
- Implementar notificaciones que informen a los usuarios de determinados eventos, por ejemplo, que el redactor sea notificado de que se le ha asignado un nuevo trabajo.

De cara a la implementación de nuevas funcionalidades pueden tomarse en cuenta las siguientes:

- Añadir toda la funcionalidad necesaria para la implementación de audioguías, inluyendo un nuevo rol que se encargue de crearlas.
- Crear más funcionalidad de los carteles, como una propiedad de visibilidad con la que el administrador pueda ocultar o mostrar un cartel del público.
- Incluir funcionalidades para subir imágenes a los carteles.
- Crear una *Progressive Web App* (PWA) a partir de la aplicación *web*, con la que facilitar el acceso del usuario a la página.

Bibliografía

[1] B. Aston. A brief history of JavaScript. *Medium*. URL https://medium.com/@_benaston/lesson-1a-the-history-of-javascript-8c1ce3bffb17.

[2] Atlassian. Página web de Trello. URL https://trello.com/home.

[3] Auth0. Página web de JWT. URL https://jwt.io/.

[4] J. E. Bigné, X. Font, and L. Andreu. *Marketing de destinos turísticos. Análisis y estrategias de desarrollo*. ESIC, 2000.

[5] Bootstrap. Página web de Bootstrap. URL https://getbootstrap.com/.

[6] Bootstrap. Página web de Bootstrap Icons. URL https://icons.getbootstrap.com/.

[7] X. Canalis. El turismo es el sector que más riqueza aporta a la economía española. *Hosteltur*. URL https://www.hosteltur.com/130893_el-turismo-el-sector-que-mas-riqueza-aporta-a-la-economia-espanola.html.

[8] Dan Abramov. Página web de Redux. URL https://redux.js.org/.

[9] Docker. Página web de Docker. URL https://www.docker.com/.

[10] Facebook. Página web de React. URL https://reactjs.org/.

[11] Facebook Open Source. Página web de Jest. URL https://jestjs.io/.

[12] Figma. Página web de Figma. URL https://www.figma.com/.

[13] C. Gayete. Estudio de los sistemas de información turística en el municipio de oropesa del mar. Análisis y propuestas de mejora. Tesis de pregrado, Universidad Politécnica de Valencia, 2017. URL https://riunet.upv.es/bitstream/handle/10251/89861/GAYETE%20-%20Estudio%20de%20los%20sistemas%20de%20informaci%C3%B3n%20tur%C3%ADstica%20en%20el%20municipio%20de%20Oropesa%20del%20Mar%20%3A%20An....pdf.

[14] Google. Página web de Google Workspace. URL https://workspace.google.com/.

[15] INE. Viajeros y pernoctaciones según categorías en la Región de Murcia, . URL https://www.murciaturistica.es/es/estadisticas_de_turismo?pagina=viajeros-y-pernoctaciones-segun-categorias.

[16] INE. Viajeros no residentes en España según procedencia y destino en la Región de Murcia, . URL https://www.murciaturistica.es/es/estadisticas_de_turismo?pagina=viajeros-no-residentes-en-espana-segun-procedencia-y-destino.

[17] ITREM. Web oficial turismo Región de Murcia. URL https://www.murciaturistica.es/.

[18] kelektiv. Página de Github de bcrypt. URL https://github.com/kelektiv/node.bcrypt.js#readme.

[19] Leumas Naypoka. Multilanguage Database Design in MySQL. URL https://www.apphp.com/tutorials/index.php?page=multilanguage-database-design-in-mysql.

[20] Microsoft. Página web de TypeScript, . URL https://www.typescriptlang.org/.

[21] Microsoft. Página web de VSCode, . URL https://code.visualstudio.com/.

[22] Mozilla. Tutorial de CSS. URL https://developer.mozilla.org/es/docs/Web/CSS/.

[23] Nrwl. Página web de Nx. URL https://nx.dev/.

[24] OpenJS Foundation. Página web de ESLint, . URL https://eslint.org/.

[25] OpenJS Foundation. Página web de Express, . URL https://expressjs.com/.

[26] OpenJS Foundation. Página web de Node.js, . URL https://nodejs.org/en/.

[27] Pablo Rocamora. Plantilla TFG en Latex para la FIUM. URL https://es.overleaf.com/latex/templates/plantilla-tfg-fium/tnckzznxktsr.

[28] Postman. Página web de Postman. URL https://postman.com/.

[29] Prettier. Página web de Prettier. URL https://prettier.io/.

[30] Red Hat. ¿Qué es una API de REST? URL https://www.redhat.com/es/topics/api/what-is-a-rest-api.

[31] Redux-Saga. Página web de Redux-Saga. URL https://redux-saga.js.org/.

[32] Stack Overflow. Encuesta de las bases de datos más usadas, . URL `https://insights.stackoverflow.com/survey/2020#technology-databases`.

[33] Stack Overflow. Encuesta de los frameworks de desarrollo web más usados, . URL `https://insights.stackoverflow.com/survey/2020#technology-web-frameworks`.

[34] The Latex Project. Página web de Latex. URL `https://www.latex-project.org/`.

[35] The PostgreSQL Global Development Group. Página web de PostgreSQL. URL `https://www.postgresql.org/`.

[36] tsoa. Página web de tsoa. URL `https://tsoa-community.github.io/docs/`.

[37] TypeORM. Página web de TypeORM. URL `https://typeorm.io/#/`.

[38] visionmedia. Página de Github de SuperTest. URL `https://github.com/visionmedia/supertest#readme`.

[39] World Wide Web Foundation. History of the Web. URL `https://webfoundation.org/about/vision/history-of-the-web/`.

A. Plantillas de casos de uso

ID:	1
Nombre:	Iniciar sesión
Actor(es)	Usuario
Descripción:	Un usuario con cualquier rol deberá identificarse con su apodo de usuario y contraseña para acceder al resto de las funcionalidades de la página.
Precondiciones:	Ninguna
Postcondiciones:	Ninguna
Flujo principal:	1. El usuario introduce su apodo de usuario y su contraseña. 2. El sistema valida los datos. 3. El usuario es ingresado en el sistema.
Flujos alternativos:	2a) El apodo o la contraseña no son correctos: 1. El sistema indica al usuario que los datos introducidos no son correctos, sin especificar si se trata del apodo o la contraseña. 2. El usuario vuelve al paso 1.

Tabla A.1: Caso de uso para iniciar sesión

ID:	2
Nombre:	Ver información de la cuenta
Actor(es)	Usuario
Descripción:	Un usuario puede comprobar información relevante de su cuenta. Se mostrará su nombre real, su apodo de usuario, el rol que desempeña y los idiomas que domina.

Tabla A.2 – *Continúa en la siguiente página*

Precondiciones:	• El usuario ha iniciado sesión.
Postcondiciones:	Ninguna
Flujo principal:	1. El usuario solicita información de su cuenta. 2. El sistema muestra datos de su cuenta: el nombre real, el apodo de usuario, el rol de la cuenta y los idiomas que domina.
Flujos alternativos:	Ninguno

Tabla A.2: Caso de uso para ver información de la cuenta

ID:	3
Nombre:	CRUD usuario
Actor(es)	Administrador
Descripción:	El administrador controla todas las operaciones de creación, edición y eliminación de los usuarios registrados en el sistema.
Precondiciones:	• El administrador ha iniciado sesión.
Postcondiciones:	• Cuando se elimina un usuario que tenía trabajos pendientes, estos trabajos son cancelados.
Flujo principal:	Crear usuario: 1. El administrador introduce información personal: el nombre real, el rol que desempeña, los idiomas que domina, el apodo de usuario y la contraseña dos veces, como medida de seguridad.. 2. El sistema valida los datos. 3. El administrador es informado de la finalización del proceso.

Tabla A.3 – *Continúa en la siguiente página*

	Modificar usuario: 1. El administrador puede editar la siguiente información: el nombre real, el apodo de usuario y la contraseña, introduciendo dos veces la nueva contraseña. 2. El sistema valida los datos. 3. El administrador confirma la acción, como medida de seguridad. 4. El administrador es informado de la finalización del proceso.
	Ver información del usuario: 1. El administrador solicita información del usuario. 2. El sistema muestra información personal y de inicio de sesión del usuario: el nombre real, el rol, los idiomas que domina y el apodo de usuario.
	Eliminar usuario: 1. El administrador solicita la eliminación de un usuario. 2. El administrador confirma la acción, como medida de seguridad. 3. El usuario es informado de la finalización del proceso.

Tabla A.3 – *Continúa en la siguiente página*

Flujos alternativos:	Crear usuario: 2a) Las contraseñas no coinciden: 1. El sistema informa el error. 2. El administrador vuelve a ejecutar el paso 1. 2a) El apodo de usuario ya está siendo usado: 1. El sistema informa el error. 2. El administrador vuelve al paso 1.
	Modificar usuario: 2a) Las contraseñas no coinciden: 1. El sistema informa el error. 2. El administrador vuelve a ejecutar el paso 1. 2a) El apodo de usuario ya está siendo usado: 1. El sistema informa el error. 2. El administrador vuelve al paso 1.

Tabla A.3: Caso de uso de las operaciones CRUD para los usuarios

ID:	4
Nombre:	CRUD cartel
Actor(es)	Administrador
Descripción:	El administrador controla todas las operaciones de creación, edición y eliminación de los carteles creados en el sistema.
Precondiciones:	• El administrador ha iniciado sesión.
Postcondiciones:	• Cuando se borra un cartel, todos sus trabajos asociados también deben ser eliminados.

Tabla A.4 – *Continúa en la siguiente página*

Flujo principal:	Crear cartel: 1. El administrador indica: el nombre y el lugar asociado al cartel. 2. El sistema valida los datos. 3. El administrador es informado de la finalización del proceso.
	Modificar cartel: 1. El administrador edita: el nombre o el lugar del cartel. 2. El sistema valida los datos. 3. El administrador es informado de la finalización del proceso.
	Ver información del cartel: 1. El administrador solicita información del cartel. 2. El sistema muestra información del cartel: el nombre, el lugar asociado y la información que contiene.
	Eliminar cartel: 1. El administrador solicita la eliminación de un cartel. 2. El administrador confirma la acción, como medida de seguridad. 3. El usuario es informado de la finalización del proceso.

Tabla A.4 – *Continúa en la siguiente página*

Flujos alternativos:	Crear cartel: 2a) El nombre elegido ya está asignado a otro cartel: 1. El sistema informa el error. 2. El administrador vuelve al paso 1. Modificar cartel: 2a) El nombre elegido ya está asignado a otro cartel: 1. El sistema informa el error. 2. El administrador vuelve al paso 1.

Tabla A.4: Caso de uso de las operaciones CRUD para los carteles

ID:	5
Nombre:	Solicitar trabajo
Actor(es)	Administrador
Descripción:	El administrador puede solicitar la ejecución de un trabajo sobre un cartel. Deberá especificar su idioma y, en función del elegido, tendrá que seleccionar el redactor que lo llevará a cabo. Además, puede escribir indicaciones sobre cómo se debe realizar el trabajo.
Precondiciones:	• El usuario ha iniciado sesión. • El administrador ha creado carteles en el sistema. • El administrador ha registrado redactores. • No existe un trabajo en progreso con idioma semejante al que se quiere solicitar.

Tabla A.5 – *Continúa en la siguiente página*

Postcondiciones:	• Se crea un trabajo. • El trabajo pasa a estar en progreso. • El trabajo se asocia al administrador. • El trabajo se asocia al cartel correspondiente. • El trabajo se asocia al redactor encargado de llevarlo a cabo.
Flujo principal:	1. El administrador da un nombre al trabajo, elige un idioma, selecciona uno de los usuarios adecuados para realizarlo y escribe indicaciones sobre lo que se debe hacer. 2. El administrador confirma la acción, como medida de seguridad. 3. El administrador es informado de la finalización del proceso.
Flujos alternativos:	Ninguno

Tabla A.5: Caso de uso para solicitar un trabajo

ID:	6
Nombre:	Ver información del trabajo
Actor(es)	Administrador
Descripción:	Para cada trabajo, el administrador tiene la opción de ver información relevante. Puede visualizar el estado actual, el redactor asignado y el idioma.
Precondiciones:	• El administrador ha iniciado sesión. • El trabajo ha sido creado.
Postcondiciones:	Ninguna

Tabla A.6 – *Continúa en la siguiente página*

Flujo principal:	1. El administrador solicita información del trabajo. 2. El sistema muestra información del trabajo: el nombre, el estado actual, el idioma del trabajo, el usuario encargado de llevarlo a cabo y su contenido.
Flujos alternativos:	Ninguno

Tabla A.6: Caso de uso para ver información del trabajo

ID:	7
Nombre:	Cancelar trabajo
Actor(es)	Administrador
Descripción:	Para cada trabajo en proceso, el administrador puede cancelarlo en caso de que no sea necesario que se lleve a cabo.
Precondiciones:	• El administrador ha iniciado sesión. • El trabajo que se quiere cancelar existe. • El trabajo que se quiere cancelar no está cancelado o completado.
Postcondiciones:	• El trabajo pasa a estar cancelado.
Flujo principal:	1. El administrador solicita la cancelación del trabajo. 2. El administrador confirma la acción, como medida de seguridad. 3. El administrador es informado de la finalización del proceso.

Tabla A.7 – *Continúa en la siguiente página*

Flujos alternativos:	Ninguno

Tabla A.7: Caso de uso para cancelar el trabajo

ID:	8
Nombre:	Confirmar trabajo
Actor(es)	Administrador
Descripción:	Para cada propuesta de trabajo recibida del redactor o traductor, el administrador puede aceptarla o denegarla. En caso de que se deniegue, deberá incluirse una explicación o los cambios que se deben realizar.
Precondiciones:	• El administrador ha iniciado sesión. • El administrador ha creado el trabajo. • El trabajo está pendiente de confirmación.
Postcondiciones:	• El trabajo pasa a estar completado si se acepta, o a seguir en progreso si se rechaza. • El contenido del cartel asociado es modificado por los cambios del trabajo, si se acepta.
Flujo principal:	1. El administrador acepta el trabajo. 2. El administrador es informado de la finalización del proceso.
Flujos alternativos:	1a) El administrador rechaza el trabajo: 1. El administrador indica la razón. 2. El administrador es informado de la finalización del proceso.

Tabla A.8: Caso de uso para confirmar el trabajo

ID:	9
Nombre:	Completar trabajo
Actor(es)	Redactor
Descripción:	Para cada solicitud de trabajo, el redactor podrá hacer uso de herramientas de edición de texto para realizarla y enviarla al administrador. También contará con una herramienta que muestre el contenido del cartel en los idiomas disponibles.
Precondiciones:	• El trabajo ha sido creado por el administrador. • El redactor está asignado para completar el trabajo. • El trabajo está en progreso.
Postcondiciones:	• El trabajo pasa a estar pendiente de confirmación.
Flujo principal:	1. El sistema muestra al redactor información del trabajo: nombre e indicaciones del administrador. 2. El redactor completa el trabajo haciendo uso de herramientas de edición de texto. 3. El redactor confirma la propuesta de trabajo. 4. El redactor es informado de la finalización del proceso.
Flujos alternativos:	Ninguno

Tabla A.9: Caso de uso para completar el trabajo