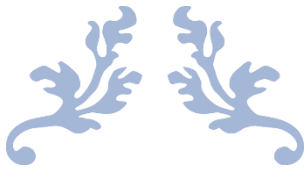


PROYECTO DE COMPUTACIÓN II

Memoria del proyecto

Álvaro Álvarez-Barriada Azaustre

Carlos Delgado Rodríguez

Javier Rodríguez González

Índice

Introducción ... 2
La España vaciada ... 2
Nuestra propuesta ... 3
Funcionalidades ... 3
Casos de uso ... 4
Vistas del programa ... 4
Login ... 4
Registro ... 5
Pantalla principal ... 6
Resultados de búsqueda ... 6
Detalle de oferta inmobiliaria ... 7
Detalle de oferta de trabajo ... 8
Favoritos ... 8
Planificación ... 10
Jornada laboral ... 10
Análisis de costes ... 10
Fuentes de datos ... 11
Planificación temporal ... 12
Resumen técnico ... 13
Front-end ... 13
Back-end ... 15
Conclusiones ... 16
Bibliografía ... 16
Anexos ... 17
Manual de instalación ... 17
Manual de usuario ... 17
Guía de fallos comunes ... 18
Ausencia del fichero *.env* ... 18
Ausencia de artisan key ... 18
Archivos obsoletos ... 18
Enlace a GitHub ... 18

Introducción

El problema conocido como "la España vaciada" consiste en la situación en la que se encuentra gran parte de la población rural de nuestro país actualmente, con una demografía escasa y en peligroso descenso y con una carencia de infraestructura, empleo y servicios que provoca que este problema se perpetúe.

Uno de los efectos de la despoblación se ve reflejado en el mercado inmobiliario, en el que es cada vez más frecuente encontrar casas y locales comerciales en venta de pueblos de la España vaciada.

Para enfrentarse a este problema, se ha desarrollado una aplicación web en forma de "recomendador", es decir, un programa que realiza búsquedas en base a una serie de parámetros y aconseje al usuario una zona en la que emprender un negocio o irse a vivir.

Con este programa se pretende incentivar el traslado a los pueblos, donde se busca recuperar las tasas de empleo y ocupación de antaño, con el fin último de recuperar la calidad de vida que, hasta hace medio siglo, existía en estas zonas tan castigadas demográficamente en la actualidad.

La España vaciada

Como se ha comentado anteriormente, este problema de despoblación a gran escala en el mundo rural no sólo es en gran medida peligroso para la subsistencia de los pueblos, sino que, además, no existen predicciones de mejora a corto plazo.

Esto se debe a que, sobre todo en municipios pequeños, la población joven decide irse a vivir a núcleos poblacionales más concurridos, como Madrid, Barcelona y otras capitales de provincia, con el fin de obtener mejores oportunidades laborales o académicas. En muchos casos, estos jóvenes no vuelven a sus pueblos de origen, sino que se quedan en estas ciudades, provocando que la población de los mismos disminuya notablemente.

Sin embargo, la población en España, sigue aumentando y ya son un 60% los municipios con menos de 1000 habitantes. A su vez, ocupan el 40% del territorio nacional, agrupándose sobre todo en Castilla y León y Extremadura.

Ante esta situación, han surgido alternativas que intentan dar una solución, como la Ley de Desarrollo Territorial y de Lucha contra la Despoblación, que intenta activar la generación de actividad económica y garantizar la prestación de servicios públicos, además de la labor de la Confederación de Centros de Desarrollo Rural (COCEDER), que ha desarrollado la plataforma volveralpueblo.org, que cuenta con un banco de casas, tierras y negocios con los que pretenden evitar que el colectivo de personas que vive en el medio rural quede excluido de unos niveles mínimos de bienestar.

Con este proyecto, los nuevos pobladores podrán alquilar, comprar u obtener una propiedad en el medio rural y, gracias a la plataforma, obtener un seguimiento personalizado, además de recibir formación y apoyo de parte de la misma, con el fin de lograr la mejor adaptación posible a su nueva vida.

Sin embargo, pese a todos los esfuerzos realizados hasta la fecha, la situación sigue empeorando, por lo que se ha decidido utilizar este proyecto para intentar mitigar los efectos de la España vaciada.

Nuestra propuesta

El programa realizado – tal y como ya se ha comentado – es una herramienta que, en base a unos criterios de búsqueda, recomienda una ubicación a la que mudarse, proporcionando información sobre ofertas en dicho lugar con el fin de facilitar el traslado a los usuarios.

A continuación, se detallan algunos aspectos relativos a esta propuesta.

Funcionalidades

La función principal del proyecto es recomendar propiedades en zonas de despoblación en base a unas condiciones que el cliente haya elegido a modo de filtro.

También muestra información adicional tanto de los pisos como de las ofertas de trabajo, donde el usuario puede encontrar detalles sobre sendas ofertas.

Además, cuenta con un apartado de *login* para que el cliente pueda acceder a sus datos, como, por ejemplo, una lista de posibles viviendas favoritas. Existe un apartado dedicado al registro de usuarios, donde se puede crear una cuenta introduciendo una serie de datos.

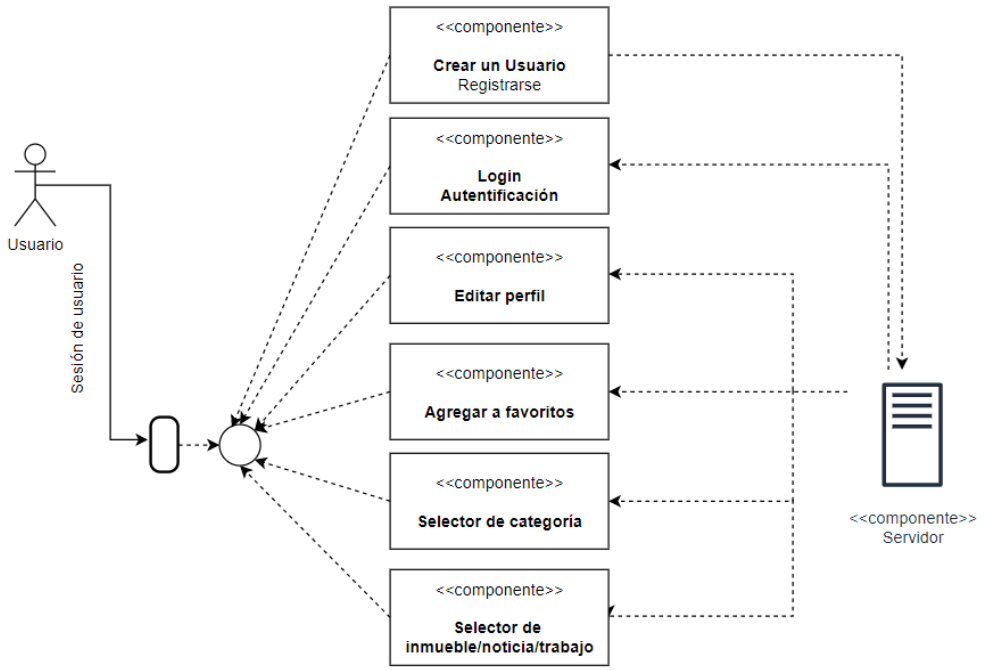


Figura 1: diagrama funcional

Casos de uso

El funcionamiento de este programa está pensado para que el usuario, de una forma sencilla pueda buscar ofertas inmobiliarias y de trabajo en una zona despoblada, reflejado en el siguiente diagrama de casos de uso:

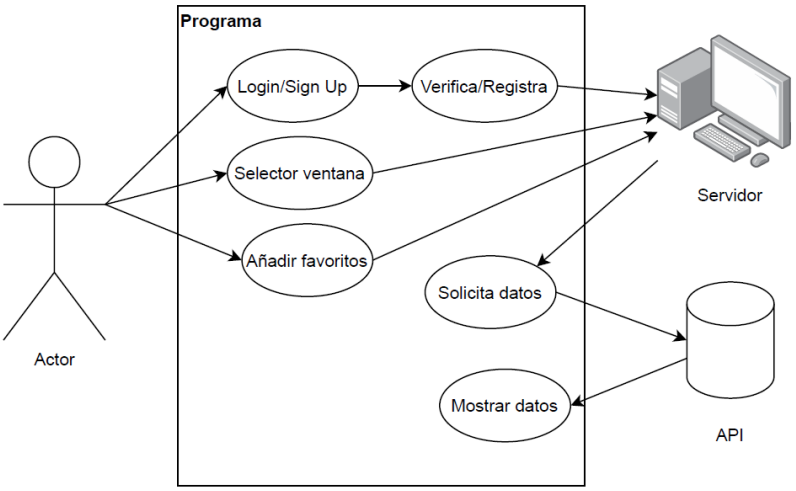


Figura 2: diagrama de casos de uso del usuario

De tal forma, cuando un usuario accede al programa, puede iniciar sesión, registrarse o realizar una búsqueda. Además, en caso de haber iniciado sesión podrá añadir a favoritos las ofertas de inmuebles y trabajos que le resulten interesantes.

Vistas del programa

A continuación, se muestran las interfaces desarrolladas para la aplicación, desde su primera versión a modo de *mockup*, realizado con el software *Figma*, hasta la versión final presente en el programa.

Login

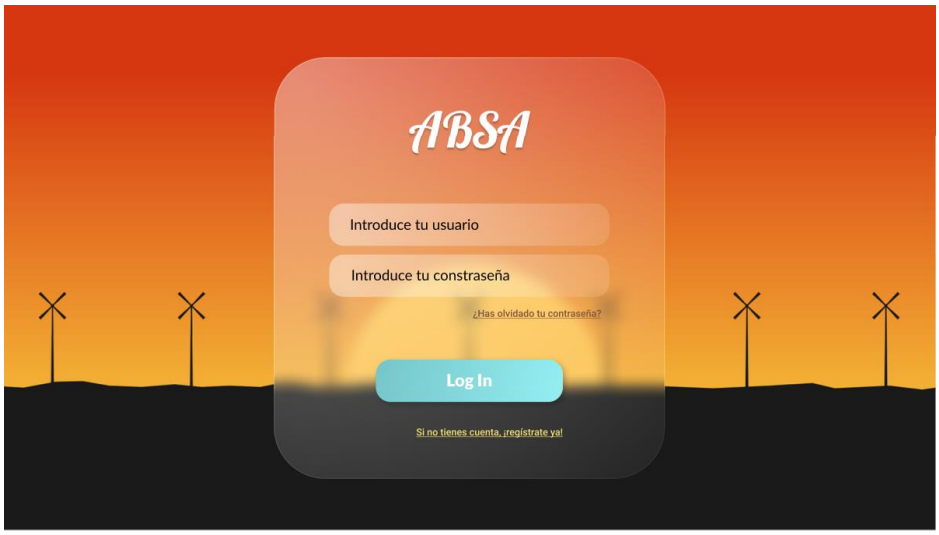


Figura 3: mockup de la ventana de login

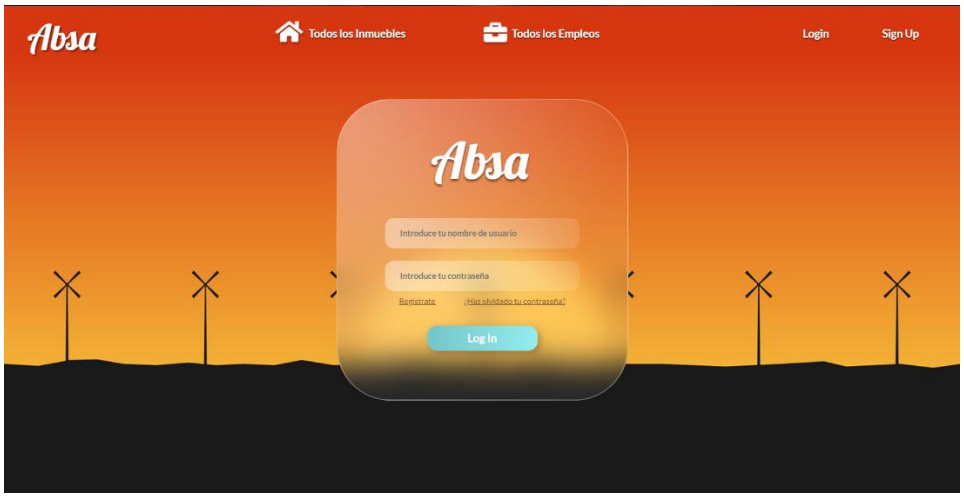


Figura 4: versión final de la ventana de login

Registro

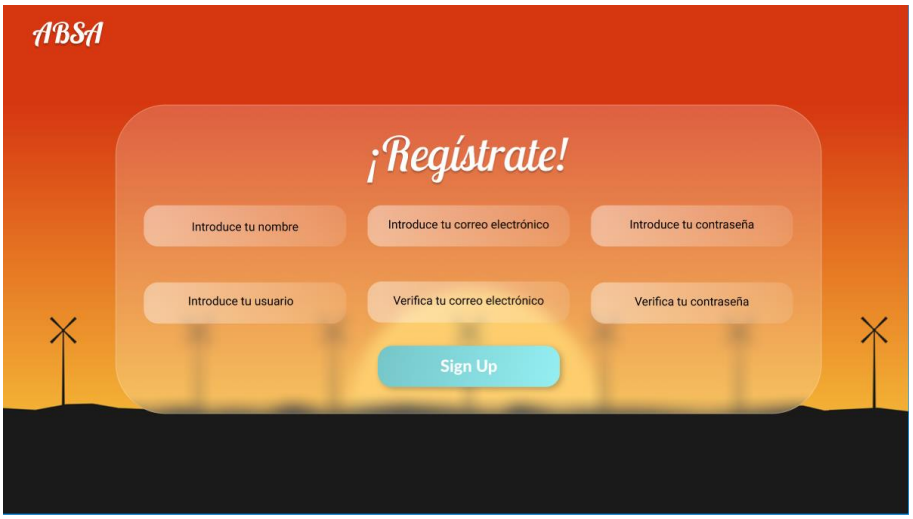


Figura 5: mockup de la ventana de registro

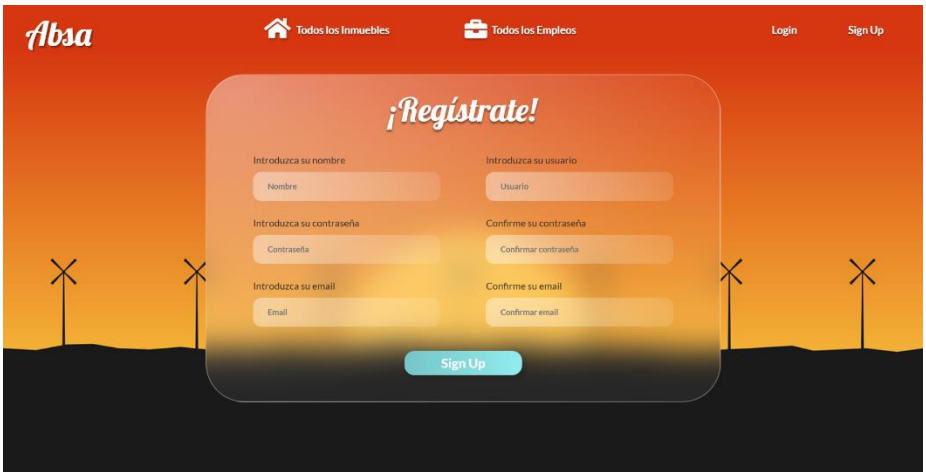


Figura 6: versión final de la ventana de registro

Pantalla principal

Figura 7: mockup de la ventana de pantalla principal

Figura 8: versión final de la ventana de pantalla principal

Resultados de búsqueda

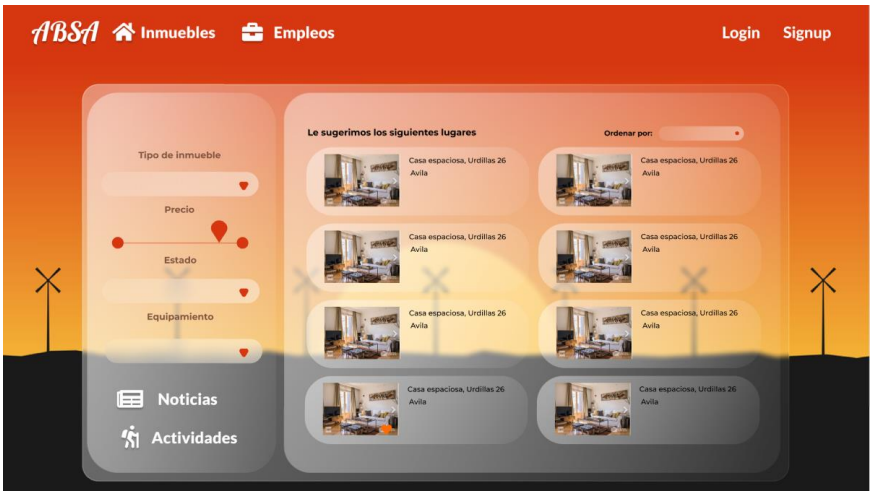


Figura 9: mockup de la ventana de resultados de búsqueda

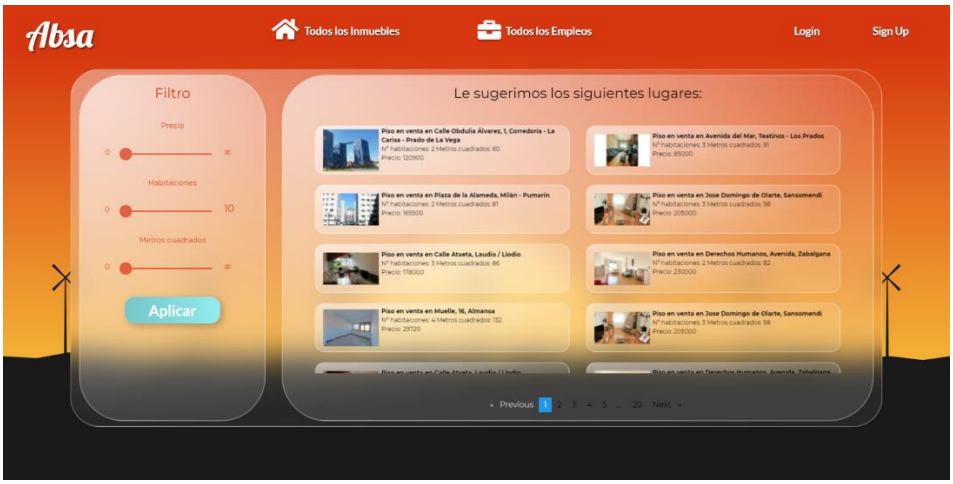


Figura 10: versión final de la ventana de resultados de búsqueda

Detalle de oferta inmobiliaria

Figura 11: mockup de la ventana de detalle de oferta inmobiliaria

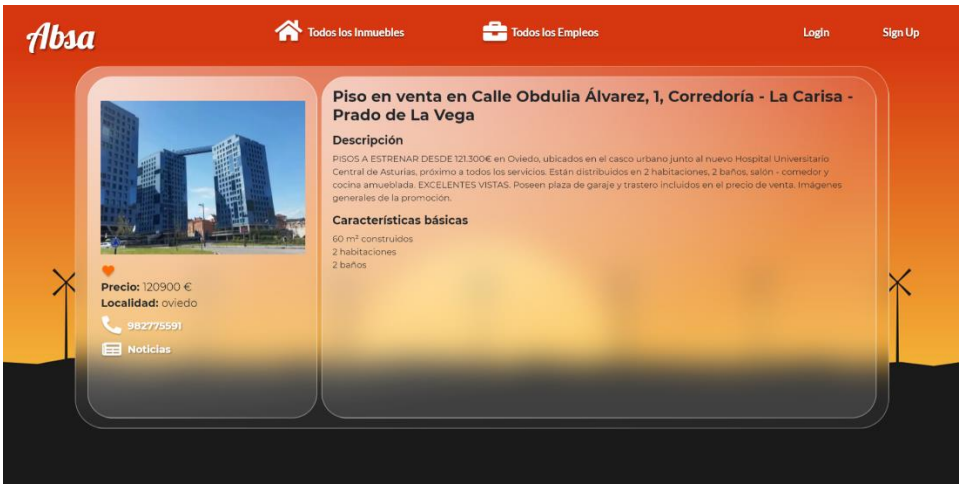


Figura 12: versión final de la ventana de detalle de oferta inmobiliaria

Detalle de oferta de trabajo

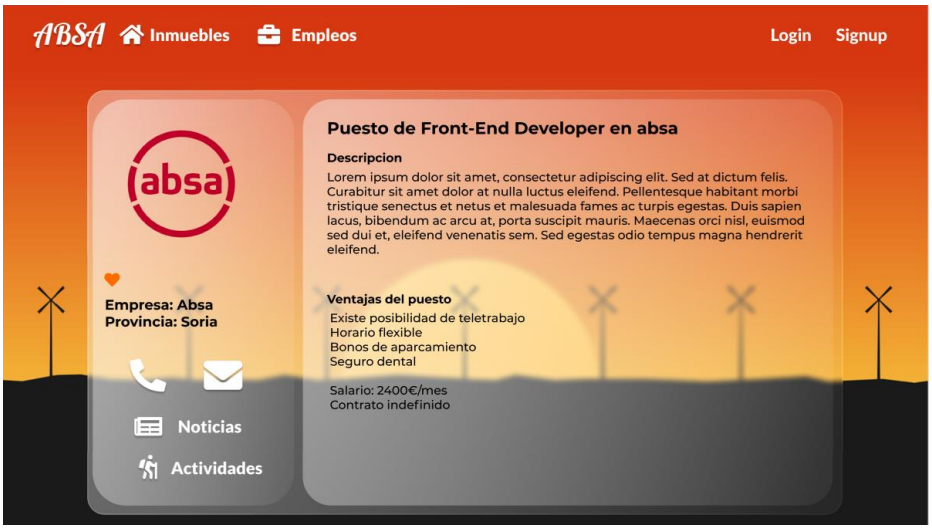


Figura 13: mockup de la ventana de detalle de oferta de trabajo

Figura 14: versión final de la ventana de detalle de oferta de trabajo

Favoritos

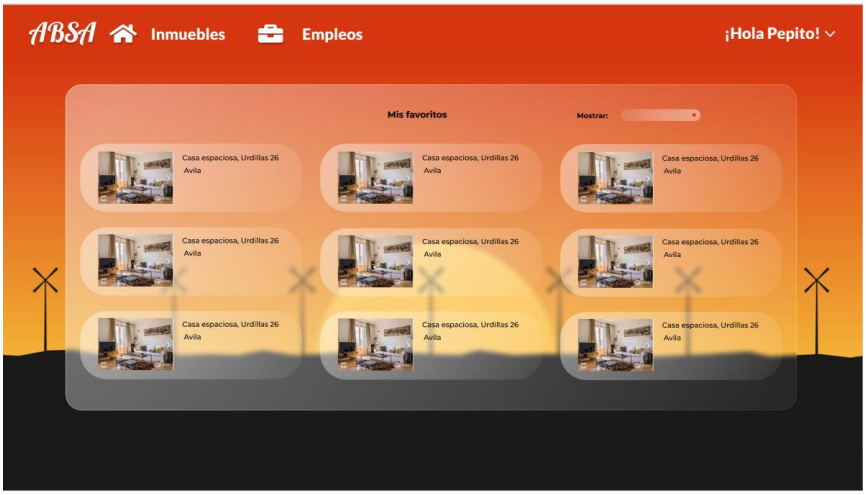


Figura 15: mockup de la ventana de favoritos

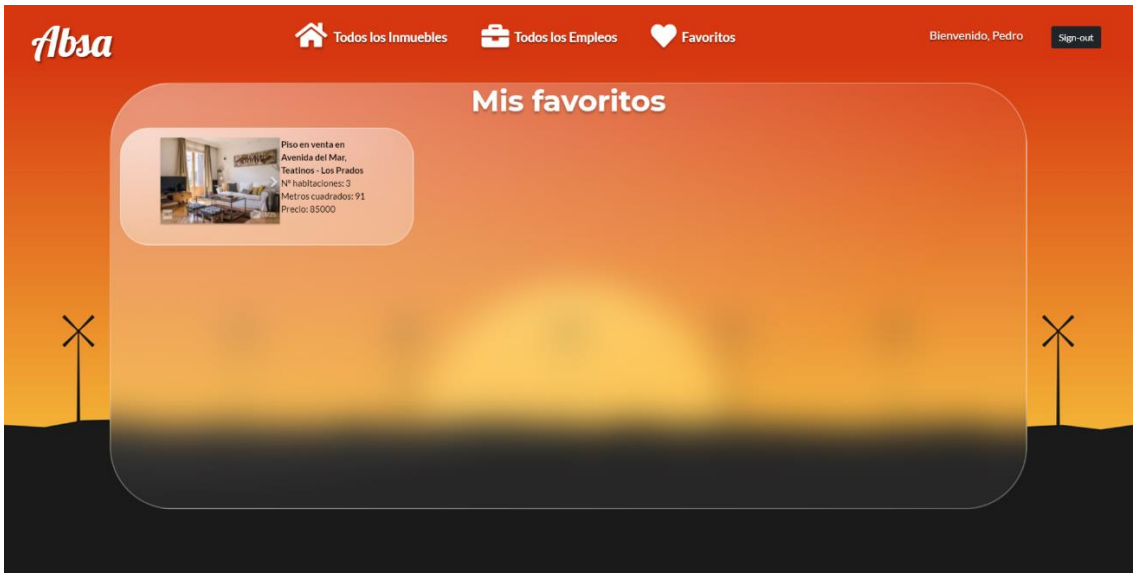
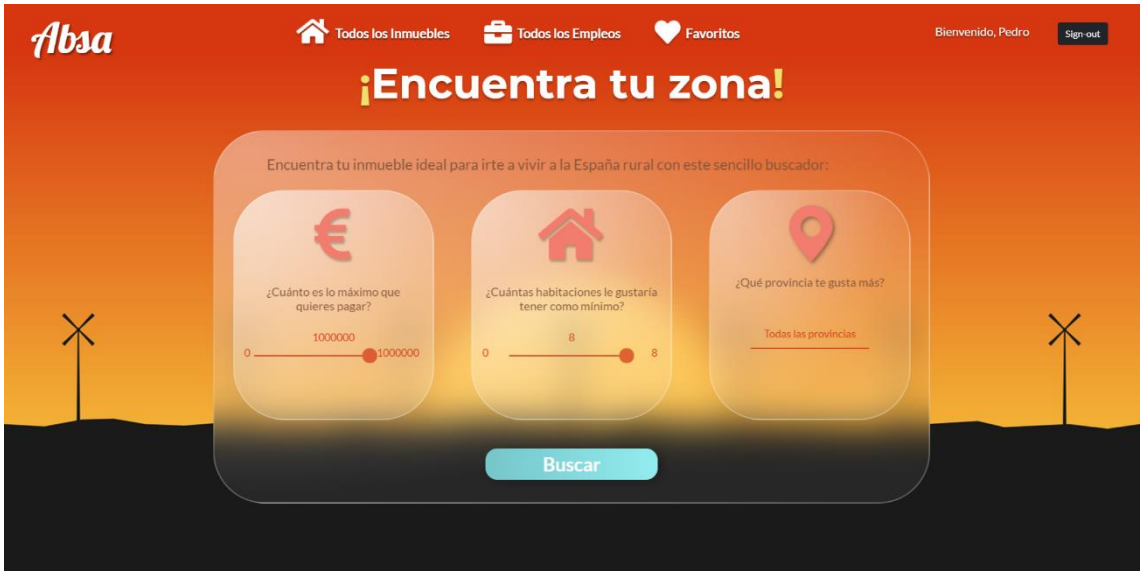


Figura 16: versión final de la ventana de favoritos



Extra: variante de la página principal con la sesión iniciada

Planificación

En este apartado se detalla la lista de tareas a realizar, el tiempo necesario para completarlas y sus fechas de inicio y finalización, así como los cambios ocurridos durante el desarrollo y otras incidencias, además de otras tareas propias de la planificación, como la fijación de una jornada laboral o la elaboración de un presupuesto.

Jornada laboral

La jornada laboral se ha establecido como 3.5 horas diarias de lunes a viernes (5 días), generando que durante una semana se pueda trabajar 17.5 horas como máximo por trabajador.

Análisis de costes

Al estar utilizando programas con licencia de uso gratuita para el desarrollo de la aplicación, el presupuesto de software es nulo, con lo que solo se debe tener en cuenta el costo de los trabajadores.

Por su parte, el desarrollo de la aplicación necesita de mínimo 3 informáticos, con un tiempo total de desarrollo estimado en 52.5 horas de trabajo por semana (total del grupo).

Si contamos con que el sueldo medio de un informático es de 57€ brutos (45€ netos) por hora de trabajo, el precio de la aplicación se establece en 57156.75€ brutos, siendo un total de 45123.75€ netos.

Por tanto, el salario de bruto de cada trabajador es de 19052.25€.

Tabla 1: análisis de precios y número de horas

Precio/hora (bruto)	57,00 €
Precio/hora (neto)	45,00 €
Total horas	334,25 horas

Tabla 2: análisis de los salarios

Salarios a pagar	
Con IVA	57.156,75 €
Sin IVA	45.123,75 €

Tabla 3: salario de cada trabajador

Sueldo por trabajador	
Con IVA	19.052,25 €
Sin IVA	15.041,25 €

Fuentes de datos

Para poder realizar el proyecto, se establecieron una serie de fuentes de datos de las cuales se intentaría extraer la información, tales como webs de inmobiliaria o APIs públicas.

Originalmente, se valoró utilizar las siguientes fuentes:

- API de Idealista: para obtener los datos de las viviendas disponibles en las diferentes zonas de despoblación con información de precios, habitaciones, etc.
- API IPV (índice de precio de vivienda): para obtener el precio de la vivienda por zonas, que se usará para identificar los lugares en los cuales el precio del terreno es bajo o alto.
- Infojobs: para buscar ofertas de empleo.
- Tripadvisor: para identificar actividades de ocio a realizar en las zonas recomendadas.

Sin embargo, debido a varios problemas, como la denegación de uso de diferentes APIs, la imposibilidad de *scrapear* alguno de los sitios webs dadas sus medidas de seguridad o por alteraciones en la planificación, surge la necesidad de cambiar las fuentes.

En el caso de idealista, se pidió – sin respuesta – permiso para utilizar su API. Tras haber intentado realizar *web scraping* sin éxito, se encontró que esta página está protegida por una empresa experta en seguridad *anti scraping*. Por este motivo, se optó por extraer la información del sitio web de Fotocasa.

Por otra parte, Tripadvisor denegó la utilización de su API, lo cual, sumado a cambios en la planificación del programa, se decidió eliminar este apartado del proyecto con el fin de enfocarse más en ofrecer información sobre el mercado inmobiliario.

Por último, en el caso de Infojobs se optó por utilizar una herramienta especializada en *screen scraping*, UiPath Studio, para extraer la información. Aunque se consiguió obtener información a través de este medio, fue imposible integrarlo en un script de Python o adherirlo al programa de forma alguna, por lo que se optó por buscar información de empleo en otras webs, como, por ejemplo, Infoempleos.

Tabla 4: relación de fuentes de datos sustituidas y motivos

Fuente original	Fuente sustituta	Motivo de cambio
API Idealista	Fotocasa (*scraper*)	Denegación de API e imposibilidad de *scrapear*
API Tripadvisor	-	Denegación de API y cambio de planificación
API Infojobs	Infoempleos (*scrap*)	Imposibilidad de integrar el *scraper* en Python

Planificación temporal

Originalmente, el desarrollo de este proyecto estaba contemplado para 19 semanas, desde febrero hasta junio.

En el primer borrador de planificación, se estableció la siguiente distribución de tareas y sus respectivos tiempos.

PROPUESTA TABLA FINAL	Horas Individuales	Fecha inicio	Fecha fin	Coste	Autor
4º Desarrollo del programa final		07-01-2021	29-01-2021	0	Todos
Creación de la interfaz	10	07-01-2021	14-02-2021	10	Álvaro
Método limpieza	1	07-01-2021	07-01-2021	1	Carlos
Tokenización	1	07-01-2021	07-01-2021	1	Carlos
Stemming	1	08-01-2021	08-01-2021	1	Carlos
Stopwords	1	08-01-2021	08-01-2021	1	Carlos
Destokenización	1	11-01-2021	11-01-2021	1	Carlos
Creación de TF-IDF	1	11-01-2021	11-01-2021	1	Carlos
Creación de modelos	1	12-01-2021	12-01-2021	1	Carlos
Controlador de la interfaz (Javier)	10	08-01-2021	15-01-2021	10	Javier
Controlador de la interfaz (Carlos)	6	12-01-2021	15-01-2021	6	Carlos
Controlador de la interfaz (Álvaro)	6	15-01-2021	15-01-2021	2	Álvaro
5º Funcionalidades finales		18-01-2021	26-01-2021	0	Todos
Posibilidad de cargar archivos (individualmente)	2	18-01-2021	18-01-2021	2	Álvaro
Posibilidad de elegir algoritmo	1	19-01-2021	19-01-2021	1	Álvaro
Previsualización del nº de archivos utilizados	2	18-01-2021	18-01-2021	2	Javier
Posibilidad de guardar el modelo entrenado	1	19-01-2021	19-01-2021	1	Álvaro
Posibilidad de guardar el diccionario de entrenamiento	2	18-01-2021	18-01-2021	2	Carlos
Entrenamiento	4	19-01-2021	20-01-2021	4	Javier
Previsualización de resultados de aprendizaje del modelo	3	19-01-2021	20-01-2021	3	Carlos
Posibilidad de cargar modelos	1	20-01-2021	20-01-2021	1	Álvaro
Posibilidad de cargar diccionario (Javier)	1	20-01-2021	20-01-2021	1	Javier
Posibilidad de cargar diccionario (Álvaro)	1	20-01-2021	20-01-2021	1	Álvaro
Seleccionar algoritmo y diccionarios propios (Javier)	1	21-01-2021	21-01-2021	1	Javier
Seleccionar algoritmo y diccionarios propios (Carlos)	1	21-01-2021	21-01-2021	1	Carlos
Seleccionar algoritmo y diccionarios propios (Álvaro)	1	21-01-2021	21-01-2021	1	Álvaro
Clasificacion	4	21-01-2021	25-01-2021	4	Carlos
Previsualización de resultados de clasificación	2	25-01-2021	26-01-2021	2	Carlos
6º Documentación - Javier	6	22-01-2021	27-01-2021	6	Javier
6º Documentación - Álvaro	6	22-01-2021	27-01-2021	6	Álvaro
6º Documentación - Carlos	4	26-01-2021	27-01-2021	4	Carlos

Figura 17: planificación inicial del proyecto

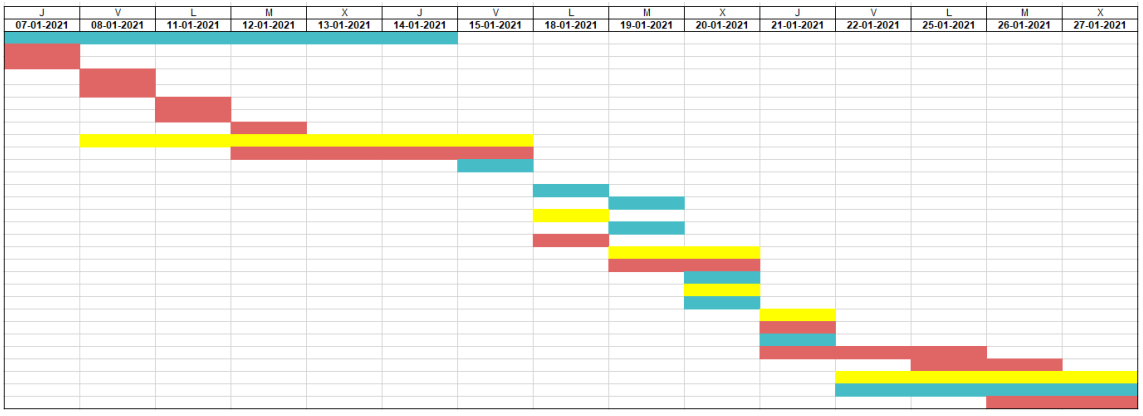


Figura 18: diagrama de Gantt original

Sin embargo, a lo largo del desarrollo surgieron distintos contratiempos que se vieron reflejados como cambios en la planificación, obteniendo, en la última entrega del calendario propuesto, las siguientes fechas y tareas.

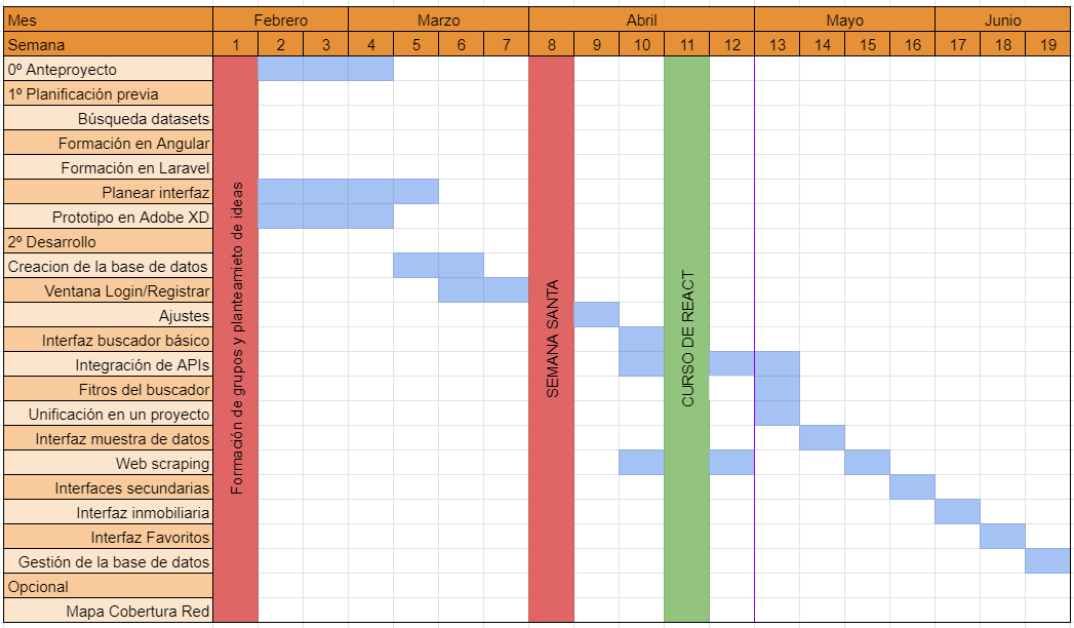


Figura 19: diagrama de Gantt y tareas actualizadas

En cuanto a las tareas sugeridas, en algunos casos se añadieron funcionalidades durante la fase de desarrollo – motivo por el que no aparecen en el diagrama – o se eliminaron debido al resultado de reuniones del equipo de desarrolladores, en las que se decidió cambiar el rumbo del proyecto y prescindir de apartados que resultaban imposibles de realizar, poniendo por ejemplo aquellos que dependían de APIs externas cuyo uso nos fue denegado.

Por otra parte, durante el desarrollo surgió un imprevisto que supuso una pérdida de valorada en dos semanas de trabajo, tras la cual el grupo se vio obligado a realizar un esfuerzo extra con el fin de poder seguir la planificación que había propuesto en la última reunión con el profesor.

En definitiva, la planificación de este proyecto ha sido cambiante, en parte debido a cambios en la idea a desarrollar, en parte por diversos problemas, por lo que, pese a que se ha realizado buscando la mayor precisión, es inevitable que contenga algunas discrepancias con el trabajo final.

Resumen técnico

En este apartado se explica brevemente cómo funcionan los dos *frameworks* utilizados en el desarrollo del proyecto.

Front-end

Para la parte de interfaces y experiencia con el usuario, *front-end*, se ha utilizado el *framework* Angular, que permite crear componentes a partir de fragmentos de código HTML con los cuales se puede exponer la información de una manera cómoda y fácil de entender, ya que estos se pueden reutilizar en distintos ficheros y cuantas veces se quiera.

Es por esto por lo que desarrollar la parte de *front-end* con Angular resulta eficiente debido a que, al crear un componente, se puede volver a usar más adelante, lo que evita duplicidad de bloques de código.

El framework, funciona de manera intuitiva con componentes. Estos se crean en la carpeta app y cada componente es una capeta con cuatro archivos.

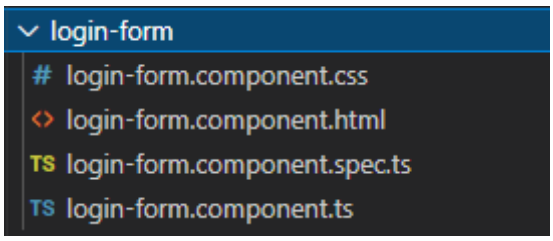


Figura 20: ficheros que constituyen un componente

Esta carpeta consta de cuatro archivos, teniendo cada uno de ellos una utilidad diferente. El archivo *.css* contiene el estilo del componente, que permite mejorar la estética del mismo. El fichero *.html* incluye el contenido de la página al que se aplica el estilo. Por su parte, el archivo *.ts* es el script del componente, en *typescript*, que permite interactuar con el *back-end*.

Los recursos de la aplicación se encuentran en la carpeta *assets* del proyecto de Angular, que además contiene todos los ficheros multimedia (imágenes, vídeos, sonidos, etc.) que pueda requerir el programa.

Para mostrar el resultado de la unión de diferentes componentes se encarga archivo *app.module.ts*, ya que gestiona la conectividad entre los mismos, su exposición e interacción con otros componentes. Además, se declaran los diferentes componentes para ser reconocidos globalmente por el *framework.*

En este archivo de control se predefinen las rutas de conexión entre los diversos componentes, haciendo uso de la librería *routes*, que permite crear un hilo de rutas desde el que se puede decidir la jerarquía e importancia de los diversos componentes.

```
import { MainpageComponent } from './components/mainpage/mainpage.component';
import { AuthHeaderInterceptor } from './shared/auth-header.interceptor';
import { FavoritosComponent } from './components/favoritos/favoritos.component';
import { OfertasComponent } from './components/ofertas/ofertas.component';
import { InfoComponent } from './components/info/info.component';
import { ComboboxComponent } from './components/combobox/combobox.component';
import { OfertastrabajoComponent } from './components/ofertastrabajo/ofertastrabajo.component';
import { InfojobsComponent } from './components/infojobs/infojobs.component';
import { NgxPaginationModule } from 'ngx-pagination';

      You, a month ago • Parte Angular copiada
const rutas: Routes = [
  {
    path: '', pathMatch:'full', redirectTo: 'Mainpage'
  },
  {
    path: 'Mainpage', component: MainpageComponent
  },
  {
```

Figura 21: fichero app.module.ts

Los archivos *.ts* de cada componente (*typescript*), constituyen el nexo entre el archivo de control del documento (*app.module.*ts) y cada componente.

En los ficheros de este tipo se declara cada clase de componente y se añade toda la interactividad de la ventana, además de permitir la comunicación con el *back-end* mediante las llamadas a las APIs.

```
@Component({
  selector: 'app-login-form',
  templateUrl: './login-form.component.html',
  styleUrls: ['./login-form.component.css']
})
export class LoginFormComponent implements OnInit {
```

Figura 22: fichero .ts de un componente

Back-end

Para implementar funcionalidades en el programa, se decidió utilizar el *framework* de Laravel, el cual se ocupa de todo el manejo de datos de la aplicación y se comunica con la parte de *front-end* (Angular) para poder mostrar, editar, añadir y borrar datos para la aplicación.

Laravel se encarga de gestionar la base de datos y de permitir la interacción del *front-end* con la misma mediante la API (Interfaz de Programación de Aplicación). Gracias a ella, la aplicación obtiene una capa de seguridad y, en caso de externalizar los datos, se puede convertir la API en pública.

Este *framework*, a su vez, permite la creación y manejo de contenidos de la base de datos creando un registro de los cambios que suceden en sus tablas, permitiendo obtener copias de seguridad y facilitar las migraciones de las mismas y su generación mediante *seeders*.

A la hora de generar la API, primero deben crearse los modelos de los datos y sus controladores, para poder tratar la información de las bases de datos como objetos.

Por último, para realizar llamadas a la API, hay que crear dicha función en el controlador y, posteriormente, asignar una referencia a la misma en el archivo *api.php*.

```
/* Ruta para operaciones de Empleos Favoritos */
Route::get("/favsjob", "App\Http\Controllers\FavsjobController@index");
Route::get("/favsjob/{userId}", "App\Http\Controllers\FavsjobController@getFavsByUsuId");
Route::post("/favsjob-addFav", "App\Http\Controllers\FavsjobController@addFav");
```

Figura 23: ejemplo de rutas en el fichero api.php

Conclusiones

En este proyecto se realizó un programa con el fin de intentar atenuar los efectos de la despoblación que está sufriendo la España vaciada mediante un recomendador de viviendas en zonas despobladas.

El proceso de desarrollo ha servido, entre otras cosas, para darse cuenta de lo susceptible que puede ser una planificación a los cambios, obligando al equipo a reorganizarse y adaptarse a las necesidades concretas de cada etapa.

Además, ha supuesto retos importantes como enfrentarse a otro tipo de impedimentos, como la imposibilidad de obtener información de las fuentes que se había previsto usar y la necesidad de suplirlas con medios similares.

En resumen, se ha logrado desarrollar una aplicación que aglutina el uso de APIs, la creación de una propia, la implementación de *web scrapers* y *screen scrapers* y la formación en entornos de trabajo nunca antes utilizados.

Bibliografía

1. CODING POTIONS. 2019. Coding Potions. Angular. *"Cómo crear rutas y componentes de forma sencilla."* [En línea] Coding Potions. [Citado el: 26 de marzo de 2021.] Disponible en: https://codingpotions.com/angular-componentes-routing

2. Google, 2021. Angular.io. *"Documentación."* [En línea] Angular. [Citado el: 6 de abril de 2021.] Disponible en: https://angular.io/guide/contributors-guide-overview

3. Laravel, 2021. Laravel Documentation. *"The PHP Framework For Web Artisans."* [En línea] Laravel.com [Citado el: 15 de mayo de 2021]. Disponible en: https://laravel.com/docs/8.x/readme

4. FEMP. 2017. Federación Española de Municipios y Provincias (FEMP). *"Despoblación: una Ley específica con toda su Financiación".* [En línea] FEMP. [Citado el: 4 de junio de 2021.] Disponible en: http://www.femp.es/comunicacion/noticias/despoblacion-una-ley-especifica-con-toda-su-financiacion

5. COCEDER. 2019. Confederación de Centros de Desarrollo Rural (COCEDER). *"¿Quiénes somos?"*. [En línea] COCEDER. [Citado el: 4 de junio de 2021.] Disponible en: http://volveralpueblo.org/quienes-somos/

6. Anders Jesen, 2020. Youtube UiPath Basics #11 - Screen Scraping with Get Text, [En línea] 15 de Julio de 2020, [Citado el: 9 de Mayo de 2020] https://www.youtube.com/watch?v=MdNVjWWkD98&ab_channel=AndersJensen

Anexos

En este apartado se detallan algunos aspectos relativos a la instalación y utilización del programa, así como dudas frecuentes.

Manual de instalación

Para instalar el programa, el usuario debe descargar el archivo comprimido de la entrega y tener instalado en su equipo los frameworks de Angular y Laravel.

Para poder ejecutar el programa, primero has de descargar una serie de dependencias que, junto con el fichero comprimido adjunto, permitirán su visualización.

Además de Angular y Laravel, el usuario debe tener instalado *Composer*, *Xampp* y la librería *npm,* de *NodeJS*. Una vez cumplidos estos requisitos, se han de seguir los siguientes pasos:

1. Se accede desde una consola del terminal a la carpeta llamada angular.
2. Una vez dentro de la carpeta se ejecuta el siguiente comando:

$$npm\ install$$

3. Una vez ha acabado de instalar todos los paquetes de ese comando se accede a la carpeta llamada proyecto y se ejecutan los siguientes comandos:

$$\begin{gathered} composer\ install \\ npm\ install \\ npm\ install\ ngx-pagination\ --save \\ php\ artisan\ key{:}\ generate \\ php\ artisan\ jwt{:}\ secret \\ php\ artisan\ cache{:}\ clear \\ php\ artisan\ config{:}\ clear \end{gathered}$$

4. Ya cuenta con todos los paquetes instalados, diríjase a la guía de lanzamiento para comprobar cómo ejecutar el programa.

Manual de usuario

Este programa puede lanzarse desde Angular (para visualizar la parte de esta asignatura, *front-end*) o Laravel, en caso de querer una perspectiva complementada del proyecto.

En caso de elegir Angular, el usuario debe abrir un terminal de consola y dirigirse a la carpeta *"angular"* del proyecto. Una vez ahí, debe ejecutar el siguiente comando:

$$ng\ serve$$

Si, por el contrario, desea ejecutar desde Laravel, el usuario deberá acceder, en cambio, a la carpeta "*proyecto*", donde lanzará los siguientes comandos:

$$php\ artisan\ serve$$

Tenga en cuenta que ambos servicios dependen de estar ejecutando la consola en segundo plano, ya que se encuentra dando servicio a la aplicación.

Guía de fallos comunes

El escenario ideal para probar el funcionamiento del programa sería con un ordenador nuevo, sin Angular, Laravel ni ninguna de las librerías instaladas en el proyecto.

Como rara vez se encuentra un ordenador "limpio", se ha elaborado una guía con los fallos más comunes que se pueden encontrar a la hora de ejecutar la aplicación.

Ausencia del fichero *.env*

En caso de localizar un mensaje de error explicando la ausencia de este fichero, el usuario puede duplicar el fichero *.env.example* y renombrarlo como *env*.

Ausencia de artisan key

Si el usuario halla este mensaje de error, debe dirigirse a la carpeta del proyecto y ejecutar el siguiente comando:

$$php\ artisan\ generate{:}\ key$$

Archivos obsoletos

Si algún archivo perteneciente a *@angular/cli* o *vendor* no está debidamente actualizado, el usuario debe dirigirse a la guía de instalación y repetir dichos pasos, que, en lugar de instalar los paquetes, actualizará aquellos que se encuentren en versiones inferiores a la utilizada en el programa.

Enlace a GitHub

https://github.com/javirg1005/HY_RE