



Universidad
Zaragoza

1542

2021/2022

4º Grado en Ingeniería Informática
Sistemas y Tecnologías Web

Memoria UnizApp

Héctor Bara Lles (778097) 778097@unizar.es
Jaime Conchello Bauto (776012) 776012@unizar.es
Víctor Hernández Fernández (717044) 717044@unizar.es
Diego Marco Beisty (755232) 755232@unizar.es

Zaragoza, España
24 de mayo de 2022

Índice

1. Resumen del proyecto	3
2. Acceso a la aplicación	3
3. Propuestas similares	5
3.1. Filtrador de EducaWeb	5
3.2. Buscador especializado de Notas de Corte	5
3.3. Buscadores de diferentes medios nacionales	5
4. Arquitectura de alto nivel	6
4.1. Vista de módulos de frontend	6
4.2. Vista de módulos de backend	9
4.3. Carga de datos de Zaguan	10
5. Modelo de datos	13
6. API Orientada a recursos	15
7. Implementación	17
7.1. Backend	17
7.1.1. Controlador User	17
7.1.2. Controlador Erasmus	17
7.1.3. Controlador Grades	17
7.1.4. Controlador GradeProfile	18
7.2. Frontend	18
7.2.1. Modelo de navegación y analíticas	19
8. Despliegue del sistema	23
8.1. Despliegue local	23
8.2. Despliegue de producción	24
9. Validación	25
9.1. Validación backend	25
9.2. Validación frontend	26
10. Problemas encontrados	27
10.1. Problemas encontrados en backend	27
10.2. Problemas encontrados en frontend	27
11. Problemas potenciales	29
11.1. Problemas potenciales backend	29
11.2. Problemas potenciales frontend	29
12. Distribución de tiempo	30
12.1. Diagrama de Gannt	30
12.2. Tiempo y esfuerzo invertido	30
13. Conclusiones	31

14. Valoración personal	31
14.1. Diego	31
14.2. Héctor	31
14.3. Jaime	31
14.4. Víctor	31
15. Anexo	33
15.1. Desarrollos opcionales	33
15.2. Vistas de aplicación web	35
15.3. Diagrama de Gannt	43
15.4. Gestión del proyecto, Scrum	44
15.4.1. Roles	44
15.4.2. Reuniones	44
15.4.3. Pila de producto	45
15.4.4. Sprints	45

1. Resumen del proyecto

El objetivo principal de UnizApp es facilitar a los estudiantes la elección de una carrera universitaria en la Universidad de Zaragoza.

Actualmente, la información que necesita un estudiante para poder elegir una carrera está disgregada y no es completa. Por un lado, las notas de corte de las carreras se publican en un PDF que no permite hacer filtrados por nota o buscar rápidamente una carrera concreta. Por otro lado, la información sobre las carreras se encuentra disponible en la web de la universidad pero no se muestran datos estadísticos que le sirvan a los estudiantes para hacerse una idea sobre la dureza de la carrera y lo que se pueden esperar de ella. Además, la única forma de conocer la opinión que tienen otras personas sobre una carrera es mediante el boca a boca.

Para solucionar este problema, UnizApp integra esta información permitiendo que los estudiantes puedan saber rápidamente a qué carreras pueden acceder con su nota de corte y puedan conocer una carrera concreta mediante los comentarios de otros estudiantes que la han cursado y métricas objetivas de la carrera. El otro objetivo que persigue UnizApp es dar a conocer a los estudiantes el programa de movilidad Erasmus de una forma atractiva. Este programa de movilidad no se conoce hasta el ecuador de la carrera y se quiere informar a los alumnos para que vean en un mapa todos los destinos que ofrece la universidad.

Esta aplicación está dirigida sobre todo a estudiantes pre-universitarios pero también a estudiantes universitarios para que puedan conocer mejor su carrera y comentar su experiencia. UnizApp será la puerta de entrada para muchos estudiantes que se sientan desorientados en su entrada a la universidad. Esperamos que facilitar una herramienta para filtrar carreras por nota de corte, conocer la experiencia de otros estudiantes y ver información objetiva de interés de cada carrera les ayude en su elección y se consiga una gran difusión.

2. Acceso a la aplicación

Backend Heroku:

- URL: <https://unizapp-backend.herokuapp.com/>
- Nombre app: unizapp-backend
- Usuario: 776012@unizar.es
- Contraseña: Unizapp2022!

Frontend Heroku:

- URL: <https://unizapp.herokuapp.com/>
- Nombre app: unizapp
- Usuario: 776012@unizar.es

- Contraseña: Unizapp2022!

GitHub:

- Organización: <https://github.com/STW-Enjoyers>
- Frontend: <https://github.com/STW-Enjoyers/frontend>
- Backend: <https://github.com/STW-Enjoyers/backend>

Cuenta administrador de Unizapp:

- Usuario: fabra@gmail.com
- Contraseña: fabra

Cuenta usuario normal Unizapp:

- Usuario: usuario@gmail.com
- Contraseña: usuario

3. Propuestas similares

Existen varios portales web que tienen funciones similares a la que ofrece nuestra aplicación, con mayor o menor diferencia. La mayoría de estas webs tienen un carácter más general, mostrando las notas de corte a nivel nacional, por lo que nuestra aplicación tiene la principal diferencia de ser especializada sobre la Universidad de Zaragoza. Entre los sitios web con mayor similitud son:

3.1. Filtrador de EducaWeb

En este caso la página ofrece un filtrado a nivel nacional, pero no cuenta con foro para las carreras, estadísticas ni permite filtrar por cupos (mayores de 25, etc). Tampoco cuenta con información de Erasmus. Aún así, es la más completa de esta sección.

Enlace: [Filtrador de EducaWeb](#)

3.2. Buscador especializado de Notas de Corte

En esta página web se ofrece un filtrado a nivel nacional, con los precios de matrícula, incluyendo universidades privadas. No tiene carreras ni estadísticas ni cupos, ni se filtra por notas de corte. Además, no parece del todo fiable, por ejemplo si buscamos nuestra carrera, el enlace que ofrece a la web de la facultad, es una página antigua que al parecer no se edita desde 2011, como podemos ver en la Figura 1.



Figura 1: Web a la que redirige el buscador de notas de corte.

Enlace: [Página especializada en notas de corte](#)

3.3. Buscadores de diferentes medios nacionales

Estas webs sólo sirven para buscar las carreras que te interesan, sin links a las mismas, sólo información (Tampoco permiten filtrar por nota de corte, sólo por universidad o nombre de grado).

Enlace: [Buscador de ElPeriódico](#)

Enlace: [Buscador de Antena3](#)

4. Arquitectura de alto nivel

En este apartado se presenta el diseño y la organización del código con una vista de módulos para frontend y otra para backend.

En cada vista se incluye una breve descripción de la responsabilidad de cada módulo.

4.1. Vista de módulos de frontend

La Figura 2 representa la vista primaria de frontend.

Más adelante, en la Figura 3 se presenta la vista del paquete app, que se separa de la vista primaria para tener mayor claridad en el diseño.

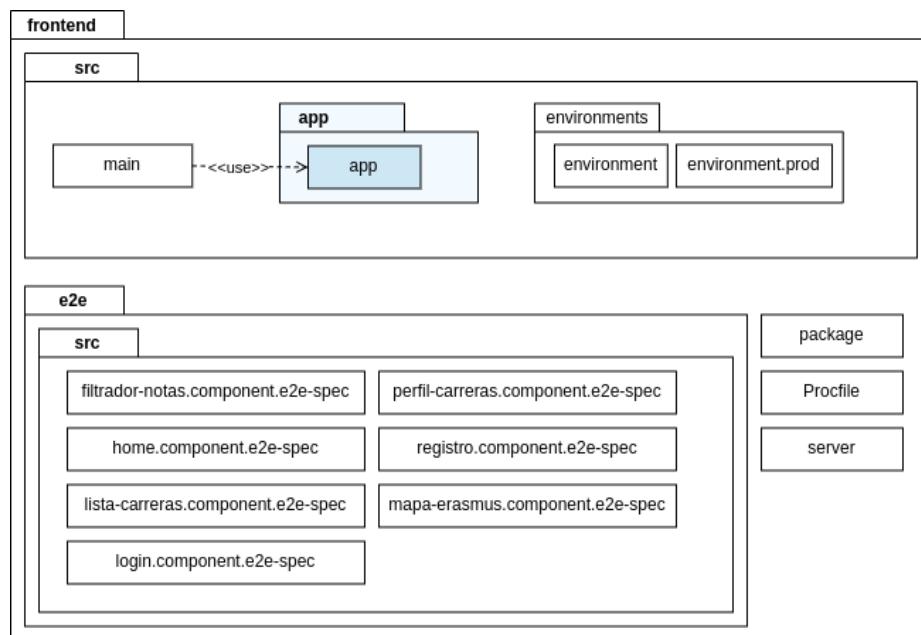


Figura 2: Vista primaria, frontend

- **main**

Módulo obligatorio del framework Angular que hace de punto de entrada de la aplicación.

- **app**

Es el paquete principal que contiene toda la funcionalidad de la aplicación de Unizapp.

- **environments**

Este paquete contiene las configuraciones utilizadas por la aplicación tanto en el despliegue local como en el despliegue de producción.

- **e2e**

Este paquete contiene los módulos de testing con Protractor, que se utilizan para validar el frontend.

■ server

Permite desplegar el frontend en Heroku.

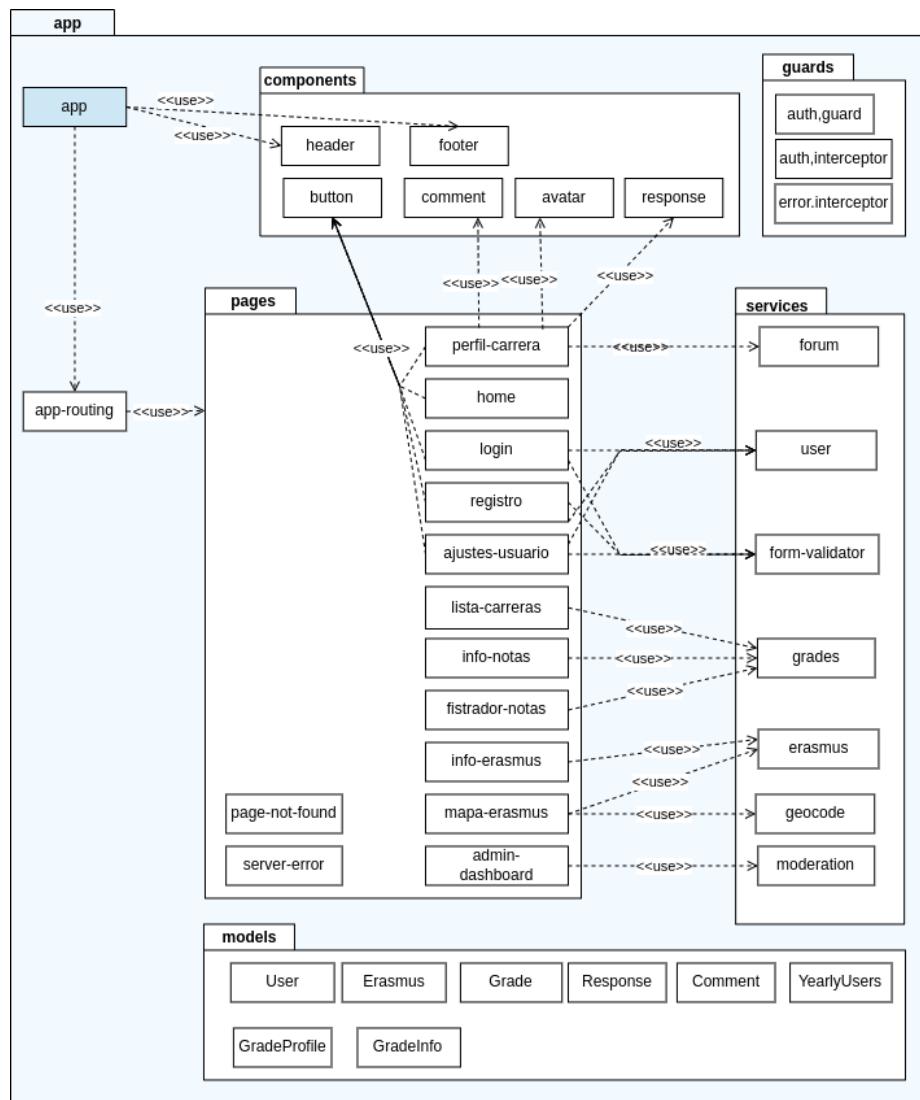


Figura 3: Vista app, frontend

■ app

Módulo raíz en la jerarquía de componentes. Incluye el routing de la aplicación junto con el header y footer.

■ app-routing

Contiene la configuración de la navegación entre las páginas de la aplicación.

■ pages

Este paquete contiene los componentes que implementan las interfaces gráficas de la aplicación.

perfil-carrera

Componente que implementa el perfil de una carrera, es decir, las estadísticas de la carrera junto con una sección de comentarios donde los usuarios logeados pueden interactuar.

home

Componente que implementa la página inicial de Unizapp. Muestra información textual sobre las principales funcionalidades principales de la aplicación .

login

Componente que contiene el formulario de login de la aplicación.

registro

Componente que contiene el formulario de registro de la aplicación.

ajustes-usuario

Componente que implementa un formulario que permite que el usuario actualice su nombre de usuario o su contraseña.

lista-carreras

Componente que contiene el listado con las principales carreras ofertadas por Unizar en las ciudades de Zaragoza, Huesca, Teruel y la localidad de la Almunia. Cada carrera tiene un hiperenlace que permite navegar a la página de perfil de la carrera.

info-notas

Componente que implementa estadísticas e información relevante sobre notas de corte.

filtrador-notas

Componente que implementa un filtro de notas de corte que permite descubrir las carreras a las que se puede acceder con una nota de corte.

info-erasmus

Componente que implementa las estadísticas e información relevante sobre los Erasmus que oferta Unizar.

mapa-erasmus

Componente que implementa el mapa que muestra los destinos Erasmus ofertados por Unizar.

admin-dashboard

Componente que implementa una interfaz solo accesible por el administrador, que permite moderar los comentarios que se publican en los perfiles de las carreras. También contiene gráficas de utilidad para el administrador.

■ services

Este paquete contiene todos los servicios que se comunican con el backend o que realizan alguna tarea que no se refleja en una interfaz gráfica.

forum

Servicio que consume el recurso /gradeProfile de la API del backend.

user

Servicio que consume el recurso /user de la API del backend.

form-validator

Servicio que gestiona la validacion de los formularios de login, registro y ajustes de usuario.

grades

Servicio que consume el recurso /grades de la API del backend.

erasmus

Servicio que consume el recurso /erasmus de la API del backend.

geocode

Servicio que traduce nombres de paises por coordenadas consultando el fichero countries.csv.

moderation

Servicio que consume los recursos de la API del backend relacionados con la moderación de comentarios.

■ models

Paquete que contiene los modelos utilizados para representar entidades concretas utilizadas tanto por los componentes como por los servicios. En el diagrama no se han dibujado las relaciones de uso que tiene este paquete para mantener un diseño más claro.

■ components

Paquete que contiene componentes reutilizables que son usados por los módulos del paquete pages.

■ guards

Paquete que contiene módulos que interceptan los mensajes HTTP que son enviados entre el frontend y backend para insertar automáticamente el token JWT, detectar códigos HTTP de error, etc.

4.2. Vista de módulos de backend

La Figura 4 representa la vista de los módulos de backend. En particular, se han diseñado e implementado la siguiente arquitectura:

- **app**: Módulo raíz en la jerarquía. Se encarga de iniciar y configurar la aplicación y de redirigir al módulo *routes* las peticiones que recibe.
- **app-api**
 - **routes/index**: Se encarga de dirigir las peticiones al controlador encargado de su procesamiento.

- **controllers/erasmusController:** Contiene las funciones para acceder a la información sobre plazas ofertadas y asignadas para alumnos Erasmus.
 - **controllers/gradesController:** Contiene las funciones para acceder a la información sobre las notas de corte del último año y de cualquier año desde el curso 2013. Se incluye también una función para obtener el histórico de notas de corte de un cierta carrera.
 - **controllers/gradeProfileController:** Contiene las funciones para gestionar el perfil de un carrera, esto incluye obtener información sobre la tasa de egresados de una carrera, publicar mensajes y respuestas en el foro, dar y cancelar "me gusta.^a los mensajes. Además, cuenta con las funciones para que el administrador pueda aceptar o cancelar aquellos mensajes del foro que no han sido verificados todavía.
 - **controllers/userController:** Contiene las funciones para registrarse, loguearse, obtener el perfil de un usuario, cambiar la contraseña y/o nombre de usuario. El administrador cuenta con funciones para *banear* a un usuario, obtener el número de nuevos usuarios mensuales y las carreras con mas comentarios.
 - **models/db:** Se encarga de conectarse a MongoDB Atlas, mediante variables de entorno detecta si se ha lanzado en local o en Heroku para cargar la contraseña,
 - **models/erasmusSchema:** Contiene el modelo del conjunto de datos Erasmus.
 - **models/gradeProfileSchema:** Contiene el modelo de datos de los perfiles de carreras.
 - **models/gradeSchema:** Contiene el modelo del conjunto de datos de notas de admisión.
 - **models/userSchema:** Contiene el modelo de datos de la información sobre usuarios.
- **config/jwtHelper:** Se encarga de configurar JSON Web Token
 - **config/passport:** Configura el módulo passport

4.3. Carga de datos de Zaguan

A continuación, se quiere describir como se trabaja con el Repositorio Institucional de Documentos de la Universidad de Zaragoza (Zaguan). Al comienzo del proyecto una de nuestras principales dudas era como gestionar la comunicación con Zaguan, ya que las peticiones tenían un tiempo de respuesta variable dependiendo del momento cuando se realizaban. Tras una primera aproximación donde se intentó obtener los datos al vuelo y se comprobó que no era viable, se decidió cargar los datos solicitados en la base de datos para así reducir los tiempos de respuesta. En la Figura 5 se presenta el diagrama de secuencia de la petición GET /api/grades/2017, se ha tomado como ejemplo para mostrar la interacción entre la aplicación, Zaguan y la base de datos ya que es idéntica para el resto de casos.

Cuando un usuario realiza una petición para obtener las notas de corte del curso 2017 ocurre lo siguiente: Primero, el backend consulta a la base de datos para comprobar si esos datos están almacenados, en cuyo caso se los devolverá al usuario y se dará por finalizada la interacción. En caso de que los datos no estuvieran guardados, se realizará una petición desde el backend a Zaguan para

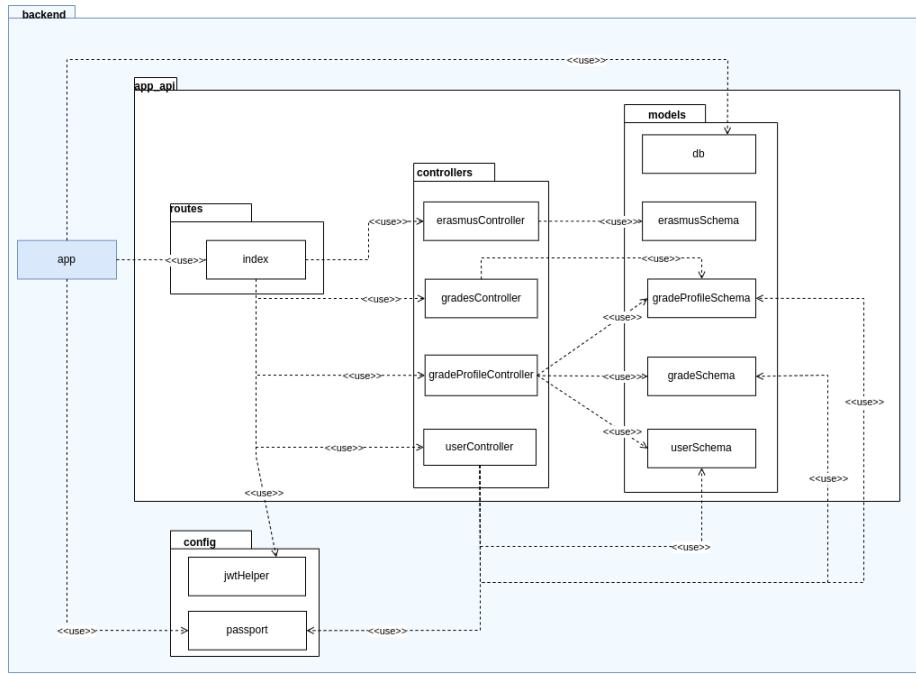


Figura 4: Vista primaria, backend

obtener los datos. Cuando se obtenga la respuesta se devolverán los datos al clientes y seguidamente se almacenaran en la base de datos. De este modo, la siguiente petición que solicite las notas de corte del curso 2017 obtendrá una respuesta en un tiempo menor ya que se obtendrá el resultado directamente desde la base de datos.

El principal problema de esta aproximación es mantener los datos actualizados. Para ello, se trabaja con cron para actualizar los datos existentes o cargar nuevos datos si, por ejemplo, se detecta que han aparecido los notas de corte de un nuevo curso.

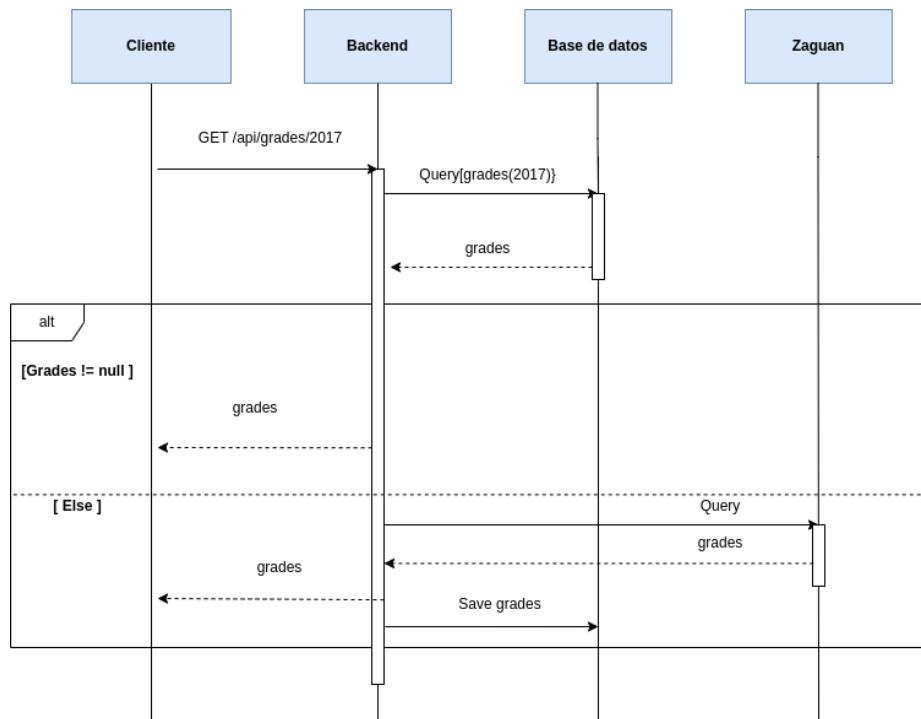


Figura 5: Diagrama de secuencia de la petición `GET /api/grades/2017`

5. Modelo de datos

El modelo de datos de la aplicación es relativamente sencillo, ya que se divide únicamente en cuatro esquemas, los cuales se pueden ver en la Figura 6. A continuación, se va a resumir el papel de cada esquema y de sus atributos.

- **Grade:** Representa cada titulación de la Universidad de Zaragoza. Su campo “nota” representa la nota de corte, “estudio” es el nombre del grado, “curso” contiene el año de los datos del grado, “cupo” se refiere al tipo de plaza que representa (General, Deportistas, Mayores de 25...) e “idCarrera” es el id que se genera a partir del “estudio” y la “localidad”. Los demás campos se entienden bien.

- **Erasmus:** Representan las distintas plazas de erasmus ofertadas, el atributo “in_out” dicta si es una plaza ofertada para que vayan desde Unizar a otras universidades (OUT), o dese otras universidades a unizar (IN). El idioma pedido para la plaza es “idioma”, “ofertadas” y “asignadas” son el número de plazas ofertadas y asignadas respectivamente, “área” es el área del estudio (Bellas Artes, Humanidades...) y “curso” es el año del acuerdo.

Los demás atributos tienen que ver con la localización, “centro” es el centro de Unizar que tiene el acuerdo con la universidad extranjera del campo “universidad” del país del campo “pais”.

- **GradeProfile:** Representa el perfil del grado, para relacionar el perfil de grado con su respectivo grado, tiene “idCarrera”.

También cuenta con (Señalado en la Figura 6, con *3) contadores de comentarios y de comentarios borrados, para aumentar la eficiencia de algunas funcionalidades de administrador.

Este es el esquema más complejo, ya que a su vez tiene varios subdocumentos:

- Graduated: Contiene las estadísticas de egresados, siendo “average” la nota media de los graduados y “graduated”, “changed” y “abandoned” el número de alumnos graduados, cambiados de carrera, o que han abandonado la carrera respectivamente.

- Comments: Contiene los comentarios del perfil. Cada uno de estos comentarios tiene el nombre de usuario y su id (“username”, “userid”), también el cuerpo del comentario, que es “body”.

Para gestionar los likes o upVotes, tiene el campo “upVotes” que es el número de estos y “upVotedUsers” (Señalado en la Figura 6, con *1), que es un array de id de usuarios para gestionar que no haya varios upVotes del mismo usuario.

También, para la gestión de borrado de comentarios y baneos están los campos “visible” (true o false) y “status” (vacío, banned o deleted).

Por último, tiene el atributo de fecha de creación (“date”), “adminCheck” para gestionar si ha sido moderado o no por un administrador y “responses”, que es un array de respuestas, que son exactamente iguales que los comentarios, pero sin el campo “responses”.

- **User:** Representa a los usuarios del sistema, con el atributo “email” único, aunque en el código también se ha hecho que “username” (nombre de usuario) no se repita.

La contraseña está implementada por “password”, con su “saltSecret” para aumentar la seguridad, “regen” (Señalado en la Figura 6, con *4) es un atributo auxiliar que hace que antes de guardar un usuario sólo se encripte la contraseña si el usuario se registra o si cambia la contraseña.

Los atributos “admin” y “banned” representan si un usuario es administrador o si ha sido baneado respectivamente. También hay un campo del día en el que se registró el usuario que se genera automáticamente al crear el usuario: “registerDate”.

Por último el atributo más complejo es “comments” (En la Figura 6 está señalado con un ***2**) que contiene datos de los comentarios del usuario, que serán del formato [idComentario] si es un comentario e [idComentario, idRespuesta], si es una respuesta.

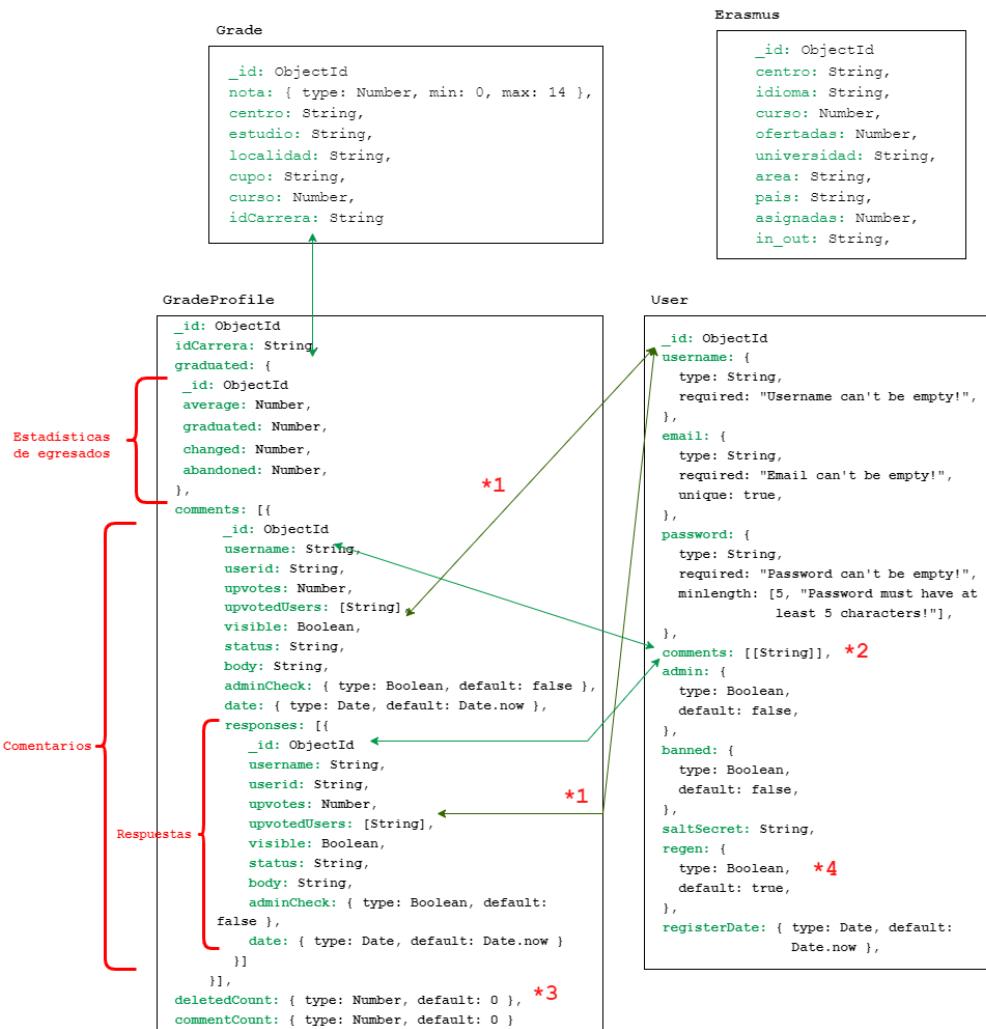


Figura 6: Modelo de datos de la aplicación

6. API Orientada a recursos

El backend de nuestra aplicación expone una API orientada a recursos, a partir del módulo index, que se puede ver en la Figura 4. En esta sección se va a explicar que papel desempeña cada petición, para ver más a fondo lo que devuelven las peticiones y los parámetros que necesitan: <https://unizapp-backend.herokuapp.com/api-docs/>

En la Tabla de los Cuadros 1 y 2 se pueden ver todas las rutas con su descripción y si necesitan token de auth para funcionar correctamente.

Lista de rutas			
Tipo	Ruta	Auth?	Descripción
GET	/grades/last	No	Esta llamada permite obtener las notas de corte del último curso
GET	/grades/{year}	No	Esta llamada permite obtener las notas de corte de un curso determinado
GET	/grades/historical/{degree}	No	Esta llamada permite obtener el histórico de notas de corte de una carrera determinada
GET	/erasmus/in	No	Esta llamada permite obtener el número de alumnos Erasmus por país que vienen a estudiar a Unizar
GET	/erasmus/out	No	Esta llamada permite obtener el número de alumnos de Unizar que se van a estudiar a cada país
POST	/user/register	No	Esta llamada permite registrar un usuario.
POST	/user/login	No	Esta llamada permite hacer login a un usuario.
GET	/user/{userid}/username	Sí	Esta llamada permite modificar el nombre de usuario del usuario que la realiza.
POST	/user/{userid}/password	Sí	Esta llamada permite modificar la contraseña del usuario que la realiza.
GET	/user/{userid}/profile	Sí	Esta llamada permite ver el perfil de un usuario.

Cuadro 1: Tabla con la lista de rutas de la API.

Tipo	Ruta	Auth?	Descripción
GET	/gradeProfile/{degreeId}/info	No	Esta llamada permite ver un perfil de carrera.
POST	/gradeProfile/{degreeId}/comment	Sí	Esta llamada permite a un usuario comentar en un perfil de carrera.
POST	/gradeProfile/{degreeId}/comment/{commentId}/reply	Sí	Esta llamada permite a un usuario responder a un comentario.
POST	/gradeProfile/{degreeId}/comment/{commentId}/cancelUpVote	Sí	Esta llamada permite a un usuario cancelar el upVote a un comentario.
POST	/gradeProfile/{degreeId}/comment/{commentId}/upVote	Sí	Esta llamada permite a un usuario hacer upVote a un comentario.
POST	/gradeProfile/{degreeId}/comment/{commentId}/reply/{replyId}/cancelUpVote	Sí	Esta llamada permite a un usuario cancelar el upVote a una respuesta.
POST	/gradeProfile/{degreeId}/comment/{commentId}/reply/{replyId}/upVote	Sí	Esta llamada permite a un usuario hacer upVote a una respuesta.
GET	/gradeProfile/comments/check	Sí	Esta llamada permite a un administrador consultar los mensajes que no han sido moderados
POST	/gradeProfile/{degreeId}/comment/{commentId}/verify	Sí	Esta llamada permite a un administrador aceptar un comentario no moderado
POST	/gradeProfile/{degreeId}/comment/{commentId}/response/{responseId}/verify	Sí	Esta llamada permite a un administrador aceptar una respuesta no moderada
DELETE	/gradeProfile/{degreeId}/comment/{commentId}/delete	Sí	Esta llamada permite a un administrador marcar como borrado un comentario.
DELETE	/gradeProfile/{degreeId}/comment/{commentId}/response/{responseId}/delete	Sí	Esta llamada permite a un administrador marcar como borrado una respuesta.
GET	/user/{userid}/ban	Sí	Esta llamada permite a un administrador banear a un usuario.
GET	/user/{userid}/yearly	Sí	Esta llamada permite a un administrador obtener el número de nuevos usuarios mensuales durante el último mes
GET	/gradeProfile/conflictives	Sí	Esta llamada permite a un administrador ver las carreras con más comentarios borrados.
GET	/gradeProfile/commented	Sí	Esta llamada permite a un administrador ver las carreras con más comentarios.

Cuadro 2: Tabla con la lista de rutas de la API.

7. Implementación

En este apartado se exponen detalles que se considera que han sido relevantes para la implementación del sistema. Se incluyen técnicas de implementación, seguridad, gestión de errores entre otros aspectos.

7.1. Backend

7.1.1. Controlador User

La implementación del módulo de Usuarios, se ha basado principalmente en el uso de jwt y de passport, que han sido configurados en ficheros dentro del módulo config. Se van a explicar a continuación para entender mejor el controlador.

- **passport.js:** En este fichero se define una localStrategy, que busca un usuario con el email dado y si existe, verifica la contraseña, con una función verifyPassword de userSchema, que se explicará más adelante.
- **jwtHelper.js:** En este fichero se crea una función que verifica el token dado y lo decodifica, para meter en req._id el id del usuario. Gracias a ello, si en la ruta ponemos esta función antes que la función del controlador (Por ejemplo: `get(jwtHelper.verifyJwtToken, controlUser.changeName)`), a esa función del controlador le llegará req._id para que pueda usarlo para buscar el usuario (O lo que necesite).

También, para gestionar el cifrado (genSalt y hash) y validación (compareSync) de las contraseñas, algunas funciones dentro de userSchema (No de controller) usan bcrypt para ese fin. Además, contiene una función en la que se generan los tokens de jwt (jwtGen), a partir del id de usuario, secret y tiempo de expiración.

En esta sección se ha hablado más de funciones de ayuda al controlador que al controlador en sí, ya que la implementación de las funciones de dentro del controlador no tienen mucha complicación. Cabe destacar el uso de la función jwtGen anteriormente nombrada para generar el token devuelto en register y login. Además del checkeo de si se es admin para las distintas funciones que necesitan privilegios de administrador y de algunos pipelines muy sencillos usados para hacer joins (\$group) entre gradeProfile y grade con sus id.

7.1.2. Controlador Erasmus

La implementación del módulo Erasmus se ha basado principalmente en el desarrollo de un *pipeline* que utiliza \$match para filtrar entre estudiantes que vienen a estudiar a Unizar o estudiantes que se van a estudiar fuera, \$group para agrupar los datos por país y \$sum para obtener el total de plazas asignadas u ofertadas. Para poder consultar esta información, se han definido dos funciones, expuestas al exterior, que se corresponden con llegadas o salidas de estudiantes.

Adicionalmente se configura el cron para actualizar o cargar nuevos datos provenientes de Zaguan sobre el programa Erasmus.

7.1.3. Controlador Grades

En la implementación del módulo de notas de admisión destaca la obtención de datos de Zaguan, ya comentada anteriormente e ilustrada en el diagrama de secuencia de la Figura 5. Además, un

aspecto que fue necesario desarrollar en este controlador, fue la generación de un identificador único para el par (estudio,localidad). Se quería generar una cadena única para una carrera impartida en una localidad, independientemente del curso, para así utilizarla como parámetro para acceder al foro de una carrera. Como en el frontend de la aplicación únicamente se muestra en el listado las notas del último año, no se podía usar el *ObjectId* de MongoDB, ya que al cambiar de curso académico, los foros de los años anteriores quedarían inaccesibles.

La solución implementada consiste en emplear el módulo *crypto* para generar un hash, con el algoritmo md5, en hexadecimal de los campos estudio y localidad de cada nota de corte que se añade a la base de datos. El resto de funcionalidad desarrollada en este módulo, no presenta ninguna gran diferencia respecto al resto de controladores. Para poder consultar esta información, se han definido tres funciones, expuestas al exterior, que se corresponden con: obtener las notas del último año, obtener las notas de un año concreto y obtener el histórico de notas de corte de una carrera.

7.1.4. Controlador GradeProfile

Como en el apartado anterior, destaca la obtención de datos de Zaguan para el perfil de carrera (Datos de egresados), además del uso de cron para automatizar la actualización y obtención de esos datos. Este módulo es el más grande de todos con diferencia, ya que las funciones de crear comentarios, respuestas, upvotes, aunque no sean muy complicadas de implementar, necesitan muchas comprobaciones y búsquedas para encontrar el dato exacto a modificar (o añadir). Además el tratamiento de los datos de egresados que ofrece Zaguan es menor granularidad, por lo que hay que sacar medias de notas e introducirlas a la BD.

Por último, para la función que devuelve los comentarios sin verificar por el administrador (checkComments) se necesita hacer un pipeline y bastante tratamiento de datos. Las otras funciones de administrador de este módulo (de borrar y verificar comentarios: deleteComment, verifyComment), siguen una lógica parecida a las funciones de los upVotes.

7.2. Frontend

El frontal de la aplicación *Unizap* se ha implementado con el framework Angular. Todo el código asociado a interfaces gráficas se ha implementado utilizando componentes. El resto del código, asociado a tareas que no utilizan GUIs, se han implementado como servicios. Haciendo esta separación se evitan problemas de acoplamiento ya que cada servicio puede ser reutilizado por cualquier componente.

Respecto a los componentes, se han agrupado respetando la cohesión del código. Es decir, los componentes que implementan una página web completa, se han agrupado en el directorio pages/. El resto de componentes implementan todas partes de una página como un botón o un formulario, se han agrupado bajo el directorio /components.

Respecto a los servicios, en su mayoría consumen cada uno un recurso distinto de la API del backend, o implementan alguna funcionalidad como la geolocalización o la validación de formularios.

Otro aspecto a destacar es que se ha aplicado una gestión de errores utilizando la clase GlobalErrorHandler, que proporciona Angular y que permite gestionar todos los errores lanzados por la aplicación.

En la Figura 7 se observa como GlobalErrorHandler gestiona tanto errores generados por componentes de Angular como respuestas HTTP inválidas devueltas por el backend (ServerErrorInterceptor). Una vez que se captura un error, esta clase puede loguear el error por consola y notificarlo al

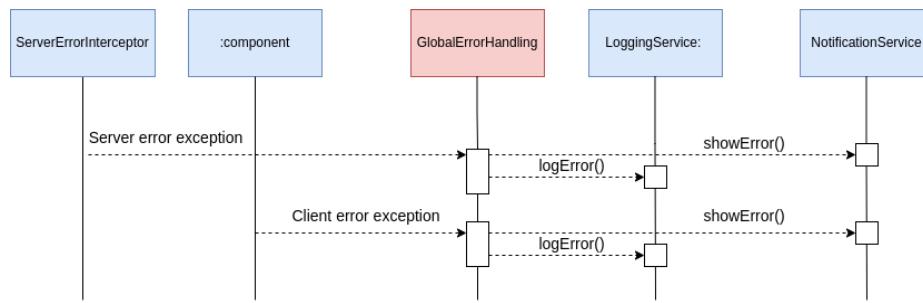


Figura 7: Diagrama de secuencia. Gestión de errores con Angular.

servicio de notificación.

Finalmente, destacar que se han utilizado guards para injectar tokens JWT en la peticiones HTTP al backend y para detectar y procesar códigos HTTP de error.

7.2.1. Modelo de navegación y analíticas

Todas las páginas tienen un header completamente responsive con la navegación de toda la aplicación tanto para usuarios registrados como no registrados, y un footer con un pequeño resumen de las páginas comunes y de una descripción de nuestro equipo. De principio a fin:

Inicio (Figura 11): Es la pantalla principal, por lo que su funcionalidad es dar la bienvenida y ayudar al usuario a encontrar las principales herramientas de la aplicación. No requiere ninguna fuente de datos ni conecta con ningún servicio externo, ya que su contenido es estático.

- **Uso:** Tiene tres descripciones y enlaces mediante botón a los apartados comunes más importantes, que son Carreras, Filtrador de Notas y Erasmus. Al final tiene un enlace a la web de unizar.
- **Analítica:** Esta página no contiene ninguna analítica.

Filtrador de Notas de Corte (Figura 12): Esta es una de las principales herramientas de la aplicación, la cual permite ver todas las notas de corte de la Universidad de Zaragoza y filtrar mediante nota máxima y cupo.

- **Uso:** En la parte superior tiene un `input` para especificar la nota máxima por la que queremos filtrar, y un `select` con los diferentes cupos que la Universidad de Zaragoza concede. Debajo, se encuentra la tabla con la información de carreras, notas de corte, localidad y cupo correspondiente.
- **Analítica:** Esta página no contiene ninguna analítica.

Información sobre Notas de Corte (Figura 13): De forma complementaria, esta página contiene información y datos relevantes sobre lo que es la nota de corte, y muestra una gráfica con las 5 notas de corte más altas por localidad.

- Uso: En la parte superior tiene dos secciones de texto que no son interactivas, y en la parte inferior tiene una gráfica con 4 botones para cambiar la localidad de consulta.
- Analítica: Contiene una gráfica con las 5 carreras con más nota de corte por comunidad. Permite conocer las carreras más exigentes de forma sencilla, y comparar si es igual en todas las localidades.

Lista de Carreras (Figura 14): Esta página tiene una lista de todas las carreras ofertadas en cada localidad, y permite buscar una carrera mediante un filtrador dinámico. De forma individual, la lista permite acceder al perfil de cada carrera y consultar su información asociada.

- Uso: En la parte superior se puede ver un carrusel para cambiar entre localidades, y por encima del carrusel se encuentra un *input* para poder filtrar por nombre. El filtro es general, por lo que se sigue aplicando al cambiar la localidad en el carrusel. En la parte inferior se muestra una tabla estilizada con las carreras ofertadas en la localidad que contienen el texto especificado en el filtro, y cada carrera mostrada es un enlace al perfil de esa carrera, al hacer *click* en su nombre.
- Analítica: Esta página no contiene ninguna analítica

Perfil de Carrera (Figura 15): Junto al filtrador de notas y al mapa de Erasmus, esta página completa las herramientas importantes de la aplicación. El perfil de la carrera se forma de manera dinámica mediante el ID de una carrera seleccionada en la lista de carreras, y contiene dos gráficas con el rendimiento académico del último año y un histórico de las notas de corte de esa carrera. Debajo de las gráficas, la página contiene un foro de comentarios y respuestas a primer nivel que permite a los usuarios compartir sus opiniones, ideas o experiencias, y responder a otros usuarios. Estos comentarios muestran un avatar predefinido, el nombre del usuario que ha escrito el comentario, el contenido del comentario, las respuestas asociadas si procede, y el número de *likes* que han expresado el resto de usuarios.

- Uso: En la parte superior se encuentran dos gráficas con estadísticas de la carrera, las cuales no son interactivas y solo son informativas. En la parte inferior, podemos ver el foro de comentarios y la parte de uso para el usuario. Esta sección muestra los comentarios escritos hasta la fecha en esa carrera, y el usuario puede ver las respuestas a los comentarios. Si el usuario quiere escribir un comentario, solo es necesario que ingrese con su cuenta y rellene el campo en la parte superior de la sección. Una vez escrito, solo hace falta pulsar el botón 'Publicar' y la aplicación insertará nuestro comentario. En el caso de querer responder a otro comentario, el usuario puedes pulsar en el texto Responder del comentario y se habilitará un campo similar al de comentario, pero que quedará enlazado como respuesta al comentario elegido. Si el número de comentarios es muy grande, el foro habilita un botón de 'Mostrar más comentarios' y se cargarán al final de los mostrados. En el caso de que al usuario le guste mucho un comentario, puede darle un *like* pulsando el botón en forma de corazón, y este quedará registrado en el comentario. Si por alguna razón se retracta, puede volver a pulsarlo y se quitará el *like*. Los *likes* se quedan guardados entre sesiones, por lo que cada usuario solo puede dar *like* a un comentario una vez.
- Analítica: Contiene dos gráficas importantes, una en la parte superior izquierda que muestra el rendimiento académico de los actuales alumnos en el último año. Esto permite ver a golpe de vista el ratio de estudiantes que se gradúan, que cambian de carrera o que abandonan. La otra

gráfica en la parte superior derecha muestra el histórico de notas almacenadas en la aplicación, y permite ver de forma muy clara la tendencia que ha tenido la nota de corte de esa carrera en el tiempo. Estas dos visualizaciones pueden ser clave para una persona que este buscando su carrera ideal.

Mapa del proyecto Erasmus (Figura 16): Esta es otra de las herramientas principales de nuestra aplicación. Contiene un mapa centrado en Europa con la información de las plazas ofertadas tanto en Unizar como en otros países del continente. Para aclarar el contenido, el mapa contiene una leyenda con la explicación de los colores mostrados en los elementos interactivos.

- Uso: El mapa permite hacer *Zoom*, y contiene elementos interactivos dibujados con forma de círculo que, al pulsarlos, muestran una etiqueta con el nombre del país y las plazas ofertadas.
- Analítica: Se puede consultar la información de cada país y de sus plazas ofertadas. Además, se puede ver la relación de plazas ofertadas a través del tamaño de los círculos dibujados.

Información sobre proyecto Erasmus (Figura 17): De forma complementaria, esta página contiene información y datos relevantes sobre lo que es el proyecto Erasmus, y muestra dos gráficas con las plazas ofertadas a cada país para estudiar en Unizar, y con las plazas ofertadas por país para estudiar fuera de España siendo estudiante de Unizar.

- Uso: En la parte superior tiene dos secciones de texto que no son interactivas, y en la parte inferior tiene dos gráficas estáticas. Esta página es solo informativa.
- Analítica: Contiene dos gráficas con que sintetizan de forma visual la información numérica del Mapa de Erasmus. Una gráfica permite ver las plazas internas de Unizar para estudiantes extranjeros, y la otra gráfica permite ver las plazas externas por país para estudiantes de Unizar. Estas gráficas permiten buscar y comparar entre países de forma muy cómoda.

Login de usuario (Figura 18): Página necesaria para la identificación de un usuario. Si el usuario quiere ingresar en la aplicación tiene que estar registrado y facilitar su correo y su contraseña.

- Uso: Contiene un formulario con dos *inputs*, uno para el correo del usuario y otro para la contraseña. El campo de correo dara error si el dato no coincide con el patrón de correo electrónico, y el campo de contraseña tiene el dato ingresado escondido al ser información privada. Una vez llenado el formulario, se puede validar mediante el botón Entrar. Si la validación tiene éxito, redirigirá a nuestros datos de usuario, y si da error, el formulario nos avisará de ello.
- Analítica: Esta página no contiene ninguna analítica

Registro de usuario (Figura 19): Junto al Login, esta página es básica para poder darse de alta en la aplicación. El usuario puede registrarse especificando un correo electrónico, un nombre de usuario y una contraseña.

- Uso: Contiene un formulario con cuatro *inputs* y un captcha para verificar que el usuario no es un robot. Para registrarse, es necesario llenar los datos con un correo electrónico válido, un nombre de usuario único, una contraseña con al menos 5 caracteres, y repetir correctamente la contraseña para evitar errores de escritura. Para poder validar el registro con el botón Registrar, es necesario completar un reCaptcha que verificará que no somos robots. Si el usuario o el correo

que hemos registrado ya existe, el formulario nos avisará del error y podremos insertar otro.

- Analítica: Esta página no contiene ninguna analítica

Ajustes de usuario (Figura 20): Esta página muestra el usuario y el correo con el que se ha ingresado, y le permite cambiar su nombre de usuario y su contraseña.

- Uso: Contiene dos formularios para cambiar el nombre de usuario y la contraseña. Para cambiar el usuario, solo es necesario llenar el *input* de Usuario y pulsar el botón de Actualizar Usuario. Si el usuario que hemos registrado ya existe, el formulario nos avisará del error y podremos insertar otro, si no, nos lo cambiará al instante y lo notificará. En caso de que decidamos cambiar la contraseña, solo necesitaremos llenar los dos *inputs* de Contraseña actual y Nueva contraseña de forma correcta y pulsar el botón Actualizar contraseña. Si la contraseña actual no coincide con la insertada, o la nueva contraseña no cumple los parámetros, dará error y el formulario nos notificará de ello.

- Analítica: Esta página no contiene ninguna analítica

Panel de administrador (Figura 21): Esta es una página exclusiva para los administradores de la aplicación. Contiene tres gráficas con información sobre usuarios registrados anualmente y sobre las carreras con más comentarios y las carreras más conflictivas. Además, permite a los administradores moderar comentarios y usuarios pudiendo realizar diferentes acciones.

- Uso: En la parte superior, contiene tres gráficas que muestran información relativa a usuarios y carreras. Estas gráficas no son interactivas. En la parte inferior se encuentra la sección de moderación, que es la parte interesante para un administrador o moderador. Esta sección tiene una especie de carrusel con los comentarios que aun no han sido moderados, y tres botones para poder Verificar un comentario, es decir, darle el visto bueno, para poder Borrar un comentario y que no se muestre su contenido, o para poder *Banear* a un usuario y que no pueda acceder ni se muestre ninguno del contenido que ha generado. En caso de que no haya comentarios sin moderar, el carrusel mostrará un mensaje de aviso y no se mostrarán los botones de moderación
- Analítica: La primera gráfica en la parte superior muestra la cantidad de usuarios registrados cada mes del año en curso. Esta gráfica permite ver si nuestra aplicación de verdad tiene un impacto mayor en épocas de admisiones y de cambios en las notas, o si el registro de usuarios decae algún mes específico. Las dos gráficas siguientes contienen las carreras con mayor número de comentarios y las carreras con mayor número de comentarios borrados. Esto nos permite ver las carreras más populares entre los usuarios y las carreras donde se originan más problemas o comentarios negativos, de forma que se pueda tener mayor control sobre ella y aumentar la moderación

8. Despliegue del sistema

8.1. Despliegue local

Para desplegar el backend en local hay que realizar lo siguiente:

- Clonar el repositorio: `git clone https://github.com/STW-Enjoyers/backend.git`
- Ejecutar: `npm install`
- Ejecutar: `npm start`

No es necesario cargar información previamente en la base de datos, ya que hemos añadido el fichero `config/config.json` con la contraseña para conectarse a MongoDB Atlas. Aunque no es una buena práctica de seguridad, hemos decidido hacerlo así para simplificar el despliegue.

Para desplegar el frontend en local hay que realizar lo siguiente:

- Clonar el repositorio: `git clone https://github.com/STW-Enjoyers/frontend.git`
- Ejecutar: `npm install`
- Ejecutar: `npm start`

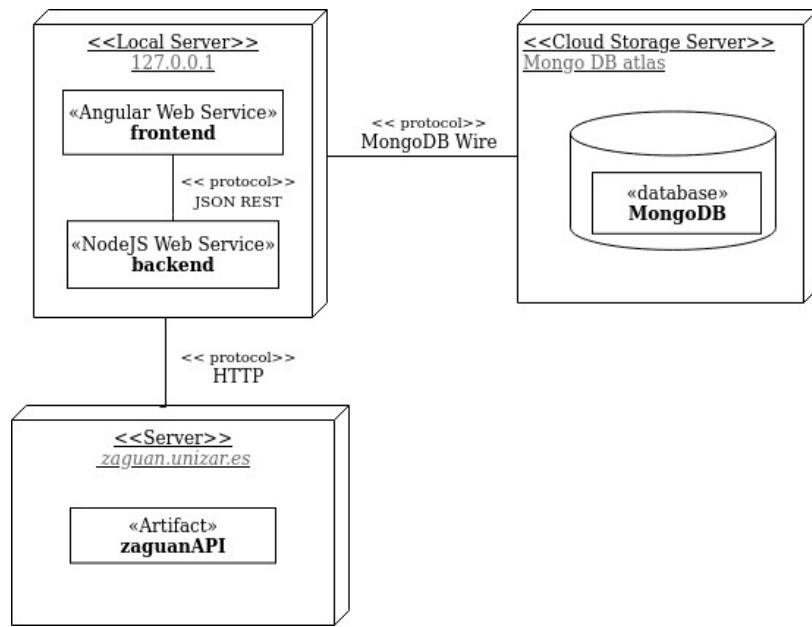


Figura 8: Despliegue local de Unizapp.

8.2. Despliegue de producción

El despliegue es accesible desde la url <https://unizapp.herokuapp.com/>. El despliegue del backend se encuentra en <https://unizapp-backend.herokuapp.com/>

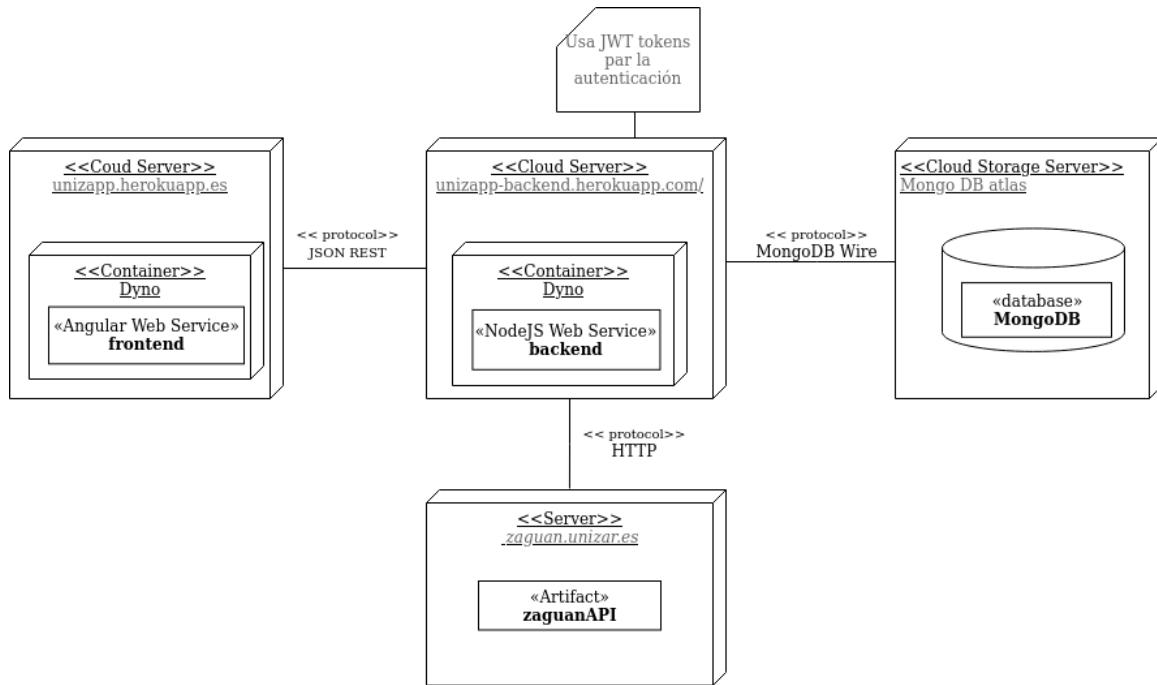


Figura 9: Despliegue en producción de Unizapp.

9. Validación

9.1. Validación backend

La validación del backend se realizó mediante test de Postman. La batería de tests se puede encontrar en nuestro repositorio del backend en la carpeta postman.test: [Link al fichero](#).

La clave de los test de Postman es el uso de las variables globales, por ejemplo, para poder probar tanto en heroku como en local, en el primer test hay un Pre-request script en el que se define la url a la que hacer las peticiones, como se puede ver en la Figura 10. Hay 3 tests (change username back y same y deleteComment) que, en función del tiempo de respuesta de la base de datos, pueden fallar

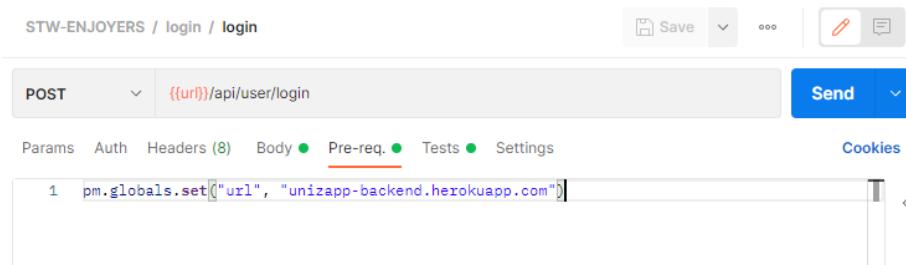


Figura 10: Uso de variables globales postman

En total se han hecho 173 tests, como son muchos y la mayoría son prácticamente iguales se van a comentar sólo los más importantes (El último resultado de la ejecución de los tests está en [la misma carpeta](#)).

- Para probar el login se usaron cuatro direcciones diferentes.
 - La petición correcta, que tenía un test que verifica que el status (Status test) era correcto, otro que comprueba que el id que se devolvía era el correcto (Constant id) y otro que ponía el token devuelto en una variable global para probar peticiones que necesitaban auth (Given token).
 - La petición que daba un email sin registrar, que también tenía un test que comprobaba el status (Status test) y otro que se aseguraba que el mensaje de error era el correcto (Message test, para mensajes de error).
 - La petición que daba una contraseña incorrecta, con Status test y Message test.
 - La petición que daba email y contraseña vacía, con Status test y Message test.
- Para probar el **register** se usó un método muy parecido, pero se quitó el test que registraba usuarios correctamente para no llenar la base de datos de usuarios de prueba.
- Para probar **profile** se hicieron pruebas de Status test, pero también se hicieron algunos nuevos, como comprobar que los campos del perfil devuelto son como los esperados en el caso de la petición correcta. Se probaron otros casos, no dando token de auth o dando uno incorrecto, para estos se comprobó que el campo auth devuelto era false. (También se probó dando un id incorrecto, con el formato de Message test)

- Los test de **info** de gradeProfile no tienen nada especial. Pero el de **comment** (Para el que se probaron los fallos igual que en profile), tiene de especial que la petición buena guarda en una variable global el id del nuevo comentario para poder hacer luego los tests de **reply** sobre él. El test de la petición correcta de **reply**, nombrado anteriormente, comprueba que el reply se ha hecho sobre el comentario guardado en la variable global, y guarda tanto el comentario como la respuesta para comparar los cambios hechos cuando se hagan las pruebas de **upVote** y **cancelUpVote** sobre ellos.
- Los test de **upVote** y **cancelUpVote** además de la petición buena, que comprueba que ha subido o bajado el número de upVotes respectivamente y de las pruebas de Status, de Auth y de Message Test (para campos incorrectos), se han hecho Message test de pruebas en las que se le daba upVote a un comentario ya upVoted por el usuario (igual con cancelUpVote)
- Para los tests de **changeUsername** y **changePassword** no se hizo nada nuevo, se desarrolló un test para devolver al estado anterior el username y el password para que se pudieran volver a ejecutar las pruebas sin problemas.
- Para los tests en los que se necesitaban privilegios de admin, se probó primero (Con Status y Message test) a hacer las peticiones sin privilegios de admin y luego (las correctas) a hacerlos con un token de admin. También se hicieron test en los que se usaban métodos de petición incorrectos, para comprobar que saltaba 501 (HttpNotImplemented).
- También se probó que los usuarios baneados no pudieran comentar, responder, dar upVote ni cancelUpVote, mediante Status test y Body comment test (Que es parecido a Message test, pero comprobando que el mensaje dice “USUARIO BANEADO”).
- Por último, para las peticiones de **grades** y **erasmus**, se hicieron algunos test interesantes, como ver en **erasmus** (tanto in como out) si las plazas ofertadas eran mayor a 0. Ver que en **grades/last** el último curso era 2021, probar que en **grades/{year}** el curso es el del año que se pasa y un Message test para un año en el que no hay datos. También se comprueba en **grades/historical** que todos los valores devueltos del histórico de notas de corte son de la carrera que se pide.

9.2. Validación frontend

El frontal de Unizapp se ha validado principalmente con Protractor. Los tests de navegador que proporciona esta librería nos han permitido saber de forma rápida el estado del proyecto. De esta forma hemos evitado subir commits a GitHub cuando encontrábamos algún bug en el sistema.

Los tests se centran en comprobar que el HTML mantiene la estructura, que ciertos comportamientos como el filtro de notas o el filtro de carreras funcionan correctamente y que la navegación es correcta.

También se ha hecho una automatización con GitHub Actions que se explica en el apartado 15.1. Destacar que interacciones complejas como la moderación de comentarios o la publicación de respuestas se ha validado de forma manual y queda para el futuro automatizarlo en tests de protractor.

10. Problemas encontrados

A continuación se describen los principales problemas encontrados durante el desarrollo de la aplicación, tanto en backend como en frontend.

10.1. Problemas encontrados en backend

Zaguan: Tal y como se ha comentado previamente en la Sección 4.3, uno de los principales problemas al principio del trabajo era consultar a Zaguan. Además de los tiempos de respuesta variable, se desconocía si la fuente de datos contaba con una API para realizar consultas o por si el contrario se tendría que utilizar Web Scraping. Investigando, se descubrió que existía una API [1] que permitía consultar indicando patrones que debían cumplirse en los nombres de los documentos devueltos. De esta forma, y conociendo los identificadores que se emplean para nombrar cada uno de los documentos que nos interesaban (notas de admisión (DS003), Erasmus (DS009) e información de graduados (DS007)), se ha podido consultar Zaguan sin la necesidad de emplear Web Scraping.

Open Api: Se encontraron problemas para añadir la autenticación necesaria para realizar acciones de administrador o consultar el perfil privado de un usuario desde Swagger. En una primera aproximación, y para poder continuar con el desarrollo, se optó por emplear el cuerpo de la petición para enviar el token. Una vez que el resto de la funcionalidad se tenía implementada, se investigó como solucionar este problema y se detectó que se estaba trabajando con una versión incompatible con la sintaxis que se estaba tratando de utilizar para añadir el token de autenticación. Como solución, se actualizó toda la sintaxis para Swagger a la versión 3.0.1 de OpenAPI y se incorporó el token en la cabecera.

Logs con Winston en Heroku: Otra dificultad que se encontró fue a la hora de guardar los logs creados por Winston en Heroku. Mientras que en local se generaban los ficheros sin problemas, en Heroku era necesario hacer uso de un plugin para poder gestionarlos de la misma forma. En un inicio, se intentó trabajar con Papertrail [2]. El problema fue que al intentar usar la versión gratuita se indicaba que había que verificar la cuenta del administrador de la aplicación en Heroku mediante la introducción de una tarjeta de crédito. Se intentó probar con el resto de plugins que se mostraban en Heroku, pero todos tenían el mismo requisito (incluido para el nivel gratuito.)

Agregaciones en MongoDB: En este caso, más que tratarse un problema en si, al no haber trabajado nunca con MongoDB, nos resultó al comienzo complicado implementar *pipelines*. Pese a ello, tras consultar ejemplos y guías, se ha podido desarrollar la funcionalidad requerida sin mayor dificultad.

10.2. Problemas encontrados en frontend

Header: Realizar un header no es difícil, pero si se busca separar componentes, que tenga desplegables, que sea dinámico al cambio de datos de usuario y que sea completamente responsive la cosa se puede complicar mucho. Dado que es un elemento que va creciendo según van completándose otros apartados, ha sido un componente con un desarrollo de principio a fin de proyecto, pasando por infinidad de cambios y retoques.

Filtrado dinámico de tablas: Una interacción que ha dado algún quebradero de cabeza es que los filtros de Filtrador-Notas y el filtro de Lista-Carreras se compartían. Por ello, si se usaba un filtro en una de las páginas y no se quitaba, creaba problemas en la visualización de resultados en el resto

de tablas con filtro ya que se seguía filtrando de forma invisible.

Mapa de Erasmus: Un problema que se ha presentado en el mapa del programa Erasmus es que no teníamos ninguna referencia en coordenadas de los países participantes, por lo que se han añadido de forma complementaria a la información obtenida de Zaguan, y después se han relacionado.

11. Problemas potenciales

En esta sección se van a analizar los problemas que podrían darse en un futuro con la implementación y despliegue actuales, tanto por parte del backend como del frontend.

11.1. Problemas potenciales backend

El mayor problema que podría darse a largo plazo sería que la **fuente de datos** de la que obtenemos toda la información relevante de la aplicación cambiara, lo que podría llegar a romper la aplicación dependiendo de los cambios que se hicieran. Para arreglarlo habría que hacer un refactor de código bastante importante.

Otro problema, y el más probable, es que falle la **base de datos** en algún momento, ya que siendo una BD gratuita en Cloud, sus capacidades son limitadas y haciendo algunas pruebas al principio del proyecto se sobrecargaba. Además, sólo tiene 512 MB de almacenamiento, ahora ocupa un porcentaje ínfimo de esa capacidad, pero a la larga llegaría a ser un problema.

Por último, como se ha nombrado en la sección 10, no se ha conseguido el guardado de **logs** en heroku, por lo que si quisieramos ver logs antiguos, no podríamos.

11.2. Problemas potenciales frontend

Hay varios problemas que pueden originarse que implicarían tener que replantear varias decisiones de diseño, o incluso modificar partes importantes de la aplicación. El mayor problema de todos es que un año se modifique el sistema de evaluación o que cambie completamente la prueba. Aunque el filtrador funcionaría correctamente ya que solo depende de las notas de corte del último año, todas las estadísticas almacenadas y mostradas dejarían de tener valor. En caso de que cambiase la prueba, es posible que se tuviese que repasar todo, teniendo en cuenta nombres, cupos, localidades o campus, incluso comentarios desfasados.

Otro posible problema de menor escala es que si aparece una carrera nueva que no tiene ninguna estadística, las gráficas puede que den algún problema de visualización.

Por otra parte, en el mapa de Erasmus se muestran las plazas externas e internas ofertadas mediante un círculo dinámico que crece en función del número de plazas de un país o su oferta. Esto puede ser problemático ya que si un año hay un crecimiento notable en un país, algunos círculos pueden solapar otros o pueden salirse del país correspondiente, dificultando la interacción con el mapa. Además, si entra un país nuevo al programa, es posible que de problemas ya que no se encontrará la referencia de coordenadas de ese país, y habrá que añadirla manualmente.

12. Distribución de tiempo

12.1. Diagrama de Gannt

Para facilitar la visualización, el diagrama de Gannt se puede consultar en la Figura 23 en el apartado de Anexo.

Es notable la diferencia entre el desarrollo frontend y backend, siendo el primero algo intermitente, donde siempre hay arreglos y cambios, y siendo el segundo un proceso mucho mas lineal, únicamente cambiando aquellas llamadas que tienen errores o a las cuales les falta algún dato puntual.

12.2. Tiempo y esfuerzo invertido

El frontend ha sido desarrollado por Diego y Victor, el backend por Héctor y Jaime.

Tiempo invertido de forma individual por cada integrante del equipo:

	Diego	Héctor	Jaime	Víctor
Febrero	5'5 horas	5'5 horas	5'5 horas	5'5 horas
Marzo	12'5 horas	12'5 horas	14 horas	9'5 horas
Abril	36'5 horas	25 horas	25 horas	28'5 horas
Mayo	15'5 horas	27'5 horas	26 horas	30'5 horas

Esto resulta en un esfuerzo individual de:

	Diego	Héctor	Jaime	Víctor
Total	70 horas	70,5 horas	70,5 horas	74 horas

Enlace: Documento con el resumen de tiempo invertido

13. Conclusiones

Este proyecto ha sido sin lugar a dudas un caso de éxito en el que los integrantes de S.T.W Enjoyers hemos estado totalmente involucrados desde el primer momento.

No solo se ha conseguido realizar una gestión impecable de proyecto aplicando Scrum, sino que además se han conseguido superar la principales dificultades técnicas que teníamos al principio. Como por ejemplo la sincronización de la base de datos con los datos publicados en Zaguan, o la adaptación que algunos hemos tenido que hacer para utilizar nuevos frameworks de desarrollo. Otro aspecto a destacar ha sido la excelente comunicación de equipo que hemos tenido, utilizando Whatsapp, discord y los issues de github.

Todo esto ha generado que podamos tener un producto que cumple con los requisitos que fueron planteados al principio del desarrollo y que creemos que puede resultar de gran utilidad para los alumnos preuniversitarios y universitarios que utilicen Unizapp.

14. Valoración personal

14.1. Diego

“Este proyecto deja un impacto muy positivo en mi paso por Ingeniería Informática. El trabajo y la comunicación en equipo ha sido gratificante. Además hemos podido completar todas las funcionalidades del proyecto en el tiempo planificado. Personalmente he conseguido desenvolverme en un framework nuevo y he aprendido un montón de conceptos que estoy seguro que me servirán de aquí en adelante.”

14.2. Héctor

“Estoy satisfecho con lo que ha conseguido hacer mi equipo en apenas 3 meses de desarrollo. La metodología Scrum seguida ha funcionado muy bien, incluso con la mitad de integrantes del equipo sin experiencia previa en Agile, llegando al final de los sprints con casi todas las funcionalidades acabadas (Con integración incluida). También siento que he aprendido mucho tanto de Node como de Mongo, de hecho, echando la vista atrás veo que podría haber quedado un código más eficiente y limpio sabiendo lo que sé ahora. Finalmente, de este proyecto me llevo la experiencia de trabajar el stack MEAN con un buen equipo y la espinita clavada de no haber aprendido más sobre los pipelines de mongo.”

14.3. Jaime

“Este trabajo me ha permitido conocer mas a fondo muchas tecnologías de las que únicamente había escuchado o leído pero nunca había utilizado y que pienso que pueden ser muy útiles en mi futuro profesional. Además, gracias a mis compañeros que propusieron utilizar la metodología Scrum, he podido conocer una nueva forma de trabajo, que en nuestro caso ha funcionado muy bien para llevar al día el proyecto. En definitiva, valoro positivamente tanto la aplicación desarrollada como la gestión y comunicación que hemos tenido en todo momento entre los miembros del equipo”

14.4. Víctor

“Aprender un nuevo lenguaje siempre es un reto, pero estoy contento con mi desarrollo y lo que he aprendido desde el inicio del proyecto. Aunque el primer encuentro con Angular ha sido duro, al final

he conseguido sobreponer los problemas y he conseguido implementar las ideas y las funcionalidades propuestas y planificadas. De este trabajo me llevo el increíble trabajo en equipo, la comunicación y la interacción con mis compañeros y la experiencia de un nuevo proyecto y un nuevo lenguaje, y también me llevo la autocrítica de los aspectos donde puedo mejorar para los siguientes proyectos.

15. Anexo

15.1. Desarrollos opcionales

- **Protractor y Github actions.** Se ha hecho una automatización de los tests de Protractor para que se ejecuten cada vez que se hace un commit o un pull request a la rama principal del repositorio de frontend. El código se adjunta a continuación:

```

name: CI

on:
  push:
    branches:
      - testing
      - main
  pull_request:
    branches:
      - main

jobs:
  build:

    runs-on: ubuntu-18.04

    steps:
      - uses: actions/checkout@v1
      - name: Use Node.js 16.13.0
        uses: actions/setup-node@v1
        with:
          node-version: 16.13.0
      - name: Install dependencies
        run: npm install
      - name: Lint
        run: npm run lint
      - name: Build
        run: npm run build -- --prod
      - name: Update webdriver
        run: node node_modules/protractor/bin/webdriver-manager update
      - name: Update Chrome # See https://stackoverflow.com/q/63651059/419956
        run: |
          wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub \
            | sudo apt-key add -
          sudo sh -c 'echo "deb http://dl.google.com/linux/chrome/deb/ stable main" >> \
            /etc/apt/sources.list.d/google-chrome.list'
          sudo apt-get update
          sudo apt-get --only-upgrade install google-chrome-stable
      - name: Run angular app
        run: npm run start-prod &
      - name: E2E tests

```

```
run: npm run e2e -- --configuration=ci
```

Como se observa en el código, se ejecutan los siguientes pasos de forma secuencial:

1. Se prepara para utilizar la versión 16.13 de Node.
2. Se actualizan todos los paquetes utilizados por la aplicación.
3. Se ejecuta un analizador de código estático.
4. Se construye la aplicación en modo producción.
5. Se actualiza webdriver.
6. Se actualiza la versión de Chrome.
7. Finalmente se ejecuta la aplicación Angular y en otro hilo se ejecutan los tests de protractor.

Destacar que en un principio no se incluyó el paso referido a la actualización de Chrome a la última versión. Sin embargo, este paso es imprescindible puesto que el webdriver no se ejecuta con ninguna versión de Chrome distinta a la última. También fue el paso más tedioso de configurar al no encontrar prácticamente documentación.

- **Testing con Postman.** El testing con postman (Desarrollo opcional) se explica en la Sección 9.2.
- **Captcha.** Para la implementación del Captcha, se ha usado la herramienta reCaptcha facilitada por Google Developers para realizar la validación. Este proceso ha seguido dos pasos:
 1. Antes de tocar código, es necesario que registremos nuestro captcha en la plataforma de Google Developers. Para ello, es necesario darse de alta como desarrollador con una cuenta de Gmail y después acceder al panel de administrador de reCAPTCHA para registrar un nuevo sitio web con reCAPTCHA v2. En la propia página te solicita que te des de alta para poder interactuar con los servicios, así que no debería ser complicado. Una vez registrado el sitio web y seleccionado la versión de reCAPTCHA deseada, podemos ver que la herramienta generará dos claves propias para nuestro Captcha, una siteKey y una secretKey. Principalmente, nos interesa la siteKey.
 2. Una vez Google ya sabe que queremos poner un captcha en nuestra web, tenemos que codificarlo. Primero de todo, instalamos el paquete de 'ng-recaptcha' para Angular, actualmente en la versión 9.0.0, y acto seguido necesitamos hacer varios cambios en nuestra aplicación. El proceso se puede hacer de varias formas, de forma más directa o parametrizada, y existen infinidad de tutoriales y guías para poder realizar fácilmente el proceso. Como ejemplo más detallado, alguna guía consultada es Dev.to (usada en esta aplicación) y TheCodeHubs. Ambas siguen un proceso similar:
 - a) En app.modules.ts, importamos el módulo de ng-recaptcha.
 - b) En app.component.ts, o la página que queramos añadirlo, tenemos que añadir la

lógica del captcha.

- c) En app.component.html, o la página que queramos añadirlo, añadimos el reCAPTCHA mediante la etiqueta `<re-captcha>`. La respuesta puede hacerse de diferentes formas.

15.2. Vistas de aplicación web

En este apartado se describen las vistas de la aplicación, los servicios con los que se conectan las vistas y se incluye al final el mapa de navegación de Unizapp.

1. Al acceder por primera vez a Unizapp, nos encontramos con la vista de la figura 11. Esta vista no se conecta con ningún servicio del backend y se utiliza para orientar al usuario.
2. Si se hace click en la opción Notas del header, se despliegan varias opciones. Al pulsar en la opción llamada *Filtrador*, accedemos a la vista de la figura 12. Esta vista permite filtrar notas de corte y se conecta con el recurso `/grades` del backend.
3. Si pulsamos en la segunda opción del desplegable llamada *Información*, accedemos a la vista de la figura 13. Esta vista muestra estadísticas sobre notas de corte y se conecta, al igual que la vista anterior, al recurso `/grades`.
4. Si pulsamos en la opción de la cabecera llamada *Carreras*, podremos acceder a la vista de la figura 14 que nos permite ver todas las carreras de Zaragoza, Huesca, Teruel, La Almunia y también nos permite acceder al perfil de cada carrera. Esta vista se conecta con el recurso `/grades` del backend.
5. Si en esta misma vista pulsamos en una carrera del listado mostrado, podremos acceder a la vista de la figura 15. Esta vista muestra estadísticas, una sección de comentarios a modo de foro y permite compartir el enlace a la misma página por correo. Se conecta con el recurso `/gradeProfile`.
6. Si en el header se pulsa sobre la opción Erasmus, se desplegarán dos opciones. Si clicamos en la que se llama *Mapa*, podremos ver la vista de la figura 16. Esta vista utiliza el recurso `/erasmus` del backend.
7. Si pulsamos en la segunda opción **Información**, podremos ver información relacionada con los erasmus. Esta vista, figura 17, se conecta con el recurso `/erasmus`.
8. Cuando nos logeamos, registramos o cambiamos los ajustes de usuario accedemos a las vistas figuras 18, 19 y 20 respectivamente. Estas tres vistas se conectan con el recurso `/user` del backend.
9. Finalmente, si nos logeamos con una cuenta de administrador, podremos observar que al pul-

sar en el nombre de usuario de la cabecera, aparece una nueva opción llamada *Dashboard*. Si pulsamos en esta opción podremos acceder a la vista de la figura 21. En esta vista, se pueden ver estadísticas de interés para el administrador y además se pueden moderar los comentarios publicados en los perfiles de las carreras. Esta vista se conecta con el recurso `/gradeProfile` y `/user` del backend.



Figura 11: Vista de Inicio

The screenshot shows the "Filtrador-Notas" (Filter Notes) page. It features a table with columns for "Carrera" (Degree Program), "Nota de corte" (Cutoff Score), "Localidad" (Location), and "Cupo" (Admission Capacity). The table lists various programs from Zaragoza and Huesca, such as Programa conjunto Física-Matemáticas (FisMat), Programa conjunto en Matemáticas-Ingeniería Informática, Medicina, Biomedicina, Odontología, and more. At the bottom, there's a pagination bar showing pages 1 through 60.

Figura 12: Vista de Filtrador-Notas



Figura 13: Vista de Info-Notas

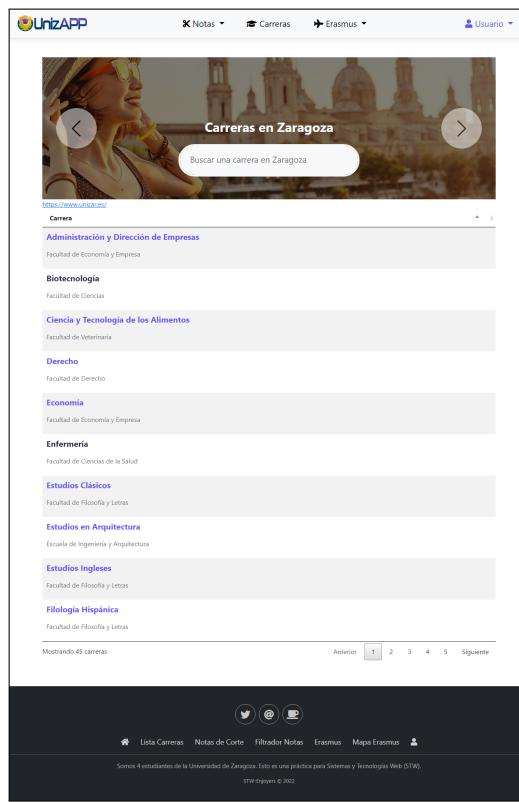


Figura 14: Vista de Carreras

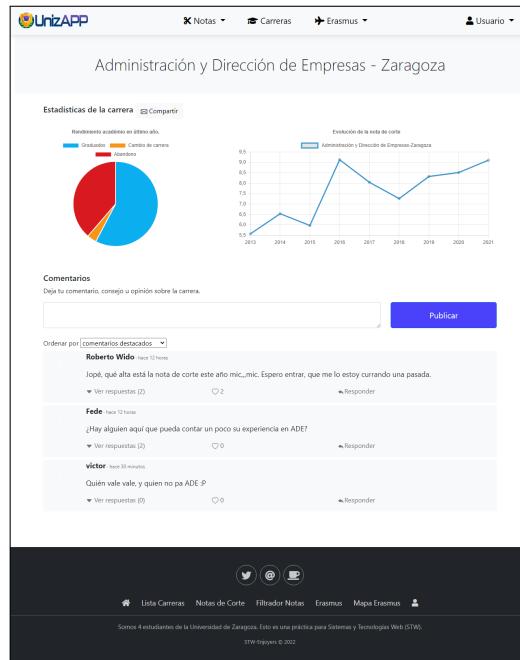


Figura 15: Vista de Perfil-Carrera

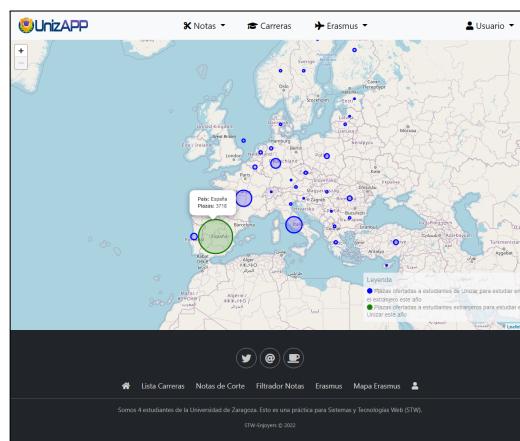


Figura 16: Vista de Mapa-Erasmus

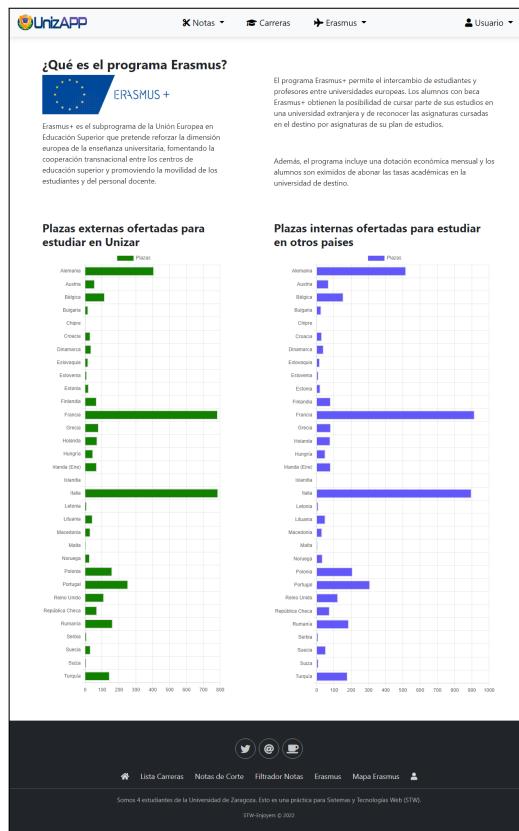


Figura 17: Vista de Info-Erasmus

Iniciar sesión

Correo
Escribir el correo

Contraseña
Escribir la contraseña

Entrar

¿No estás registrado/a todavía? [Regístrate.](#)

Figura 18: Vista de Login

The screenshot shows the 'Registrar' (Register) page of the UnizAPP web application. At the top, there are input fields for 'Correo electrónico' (Email), 'Usuario' (Username), and 'Contraseña' (Password). Below these are fields for 'Repetir contraseña' (Repeat password) and a 'No soy un robot' (I'm not a robot) checkbox with a reCAPTCHA interface. A large blue 'Registrar' (Register) button is at the bottom. At the very bottom of the page, there is a link '¿Ya estás registrado/a? Iniciar sesión' (Already registered? Log in).

Figura 19: Vista de Registro

The screenshot shows the 'Ajustes usuario' (User Settings) page. It displays the current user information: 'Usuario: vitor' and 'Correo: vitor@gmail.com'. There are two main sections: 'Usuario' (User) and 'Contraseña actual' (Current password). The 'Usuario' section has a 'Actualizar usuario' (Update user) button. The 'Contraseña actual' section has fields for 'Contraseña actual' (Current password) and 'Nueva contraseña' (New password), both with eye icon password inputs, and an 'Actualizar contraseña' (Update password) button.

Figura 20: Vista de Ajustes-Usuario

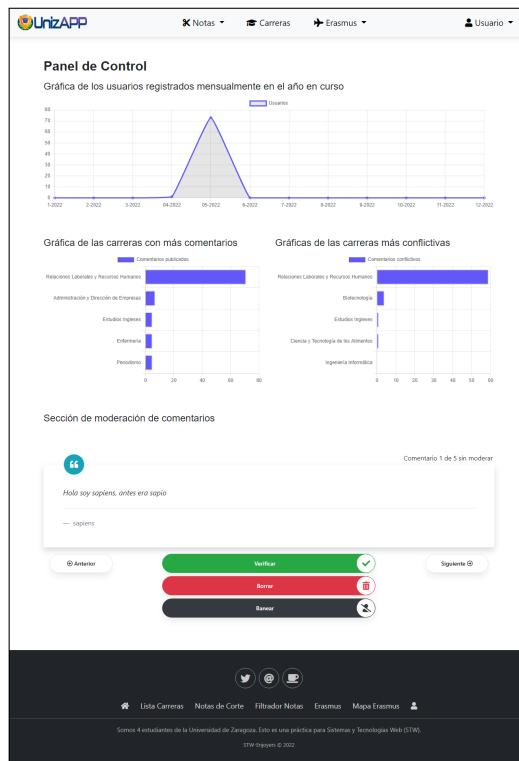


Figura 21: Vista de Admin-dashboard

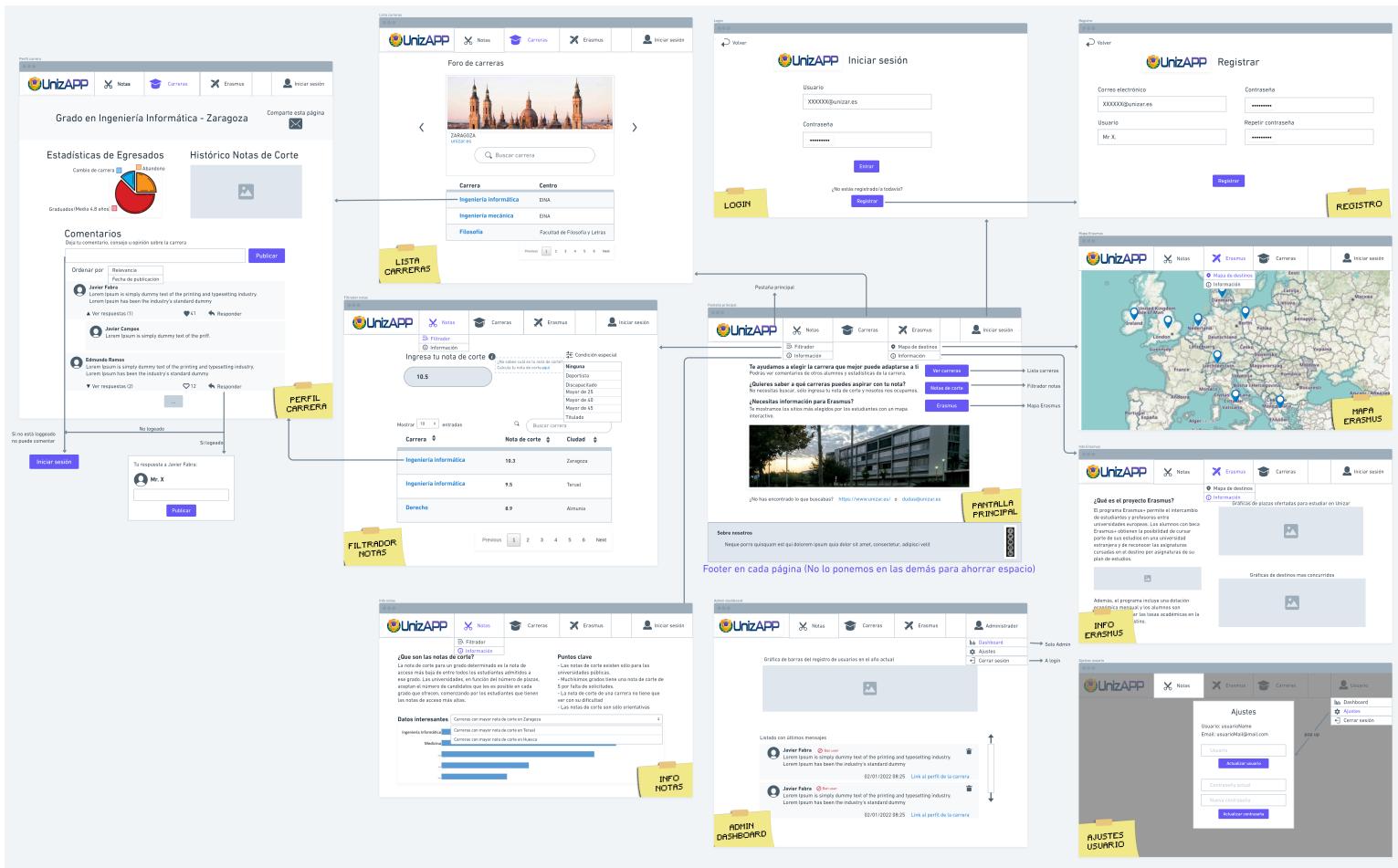


Figura 22: Mapa de navegación seguido

15.3. Diagrama de Gannt

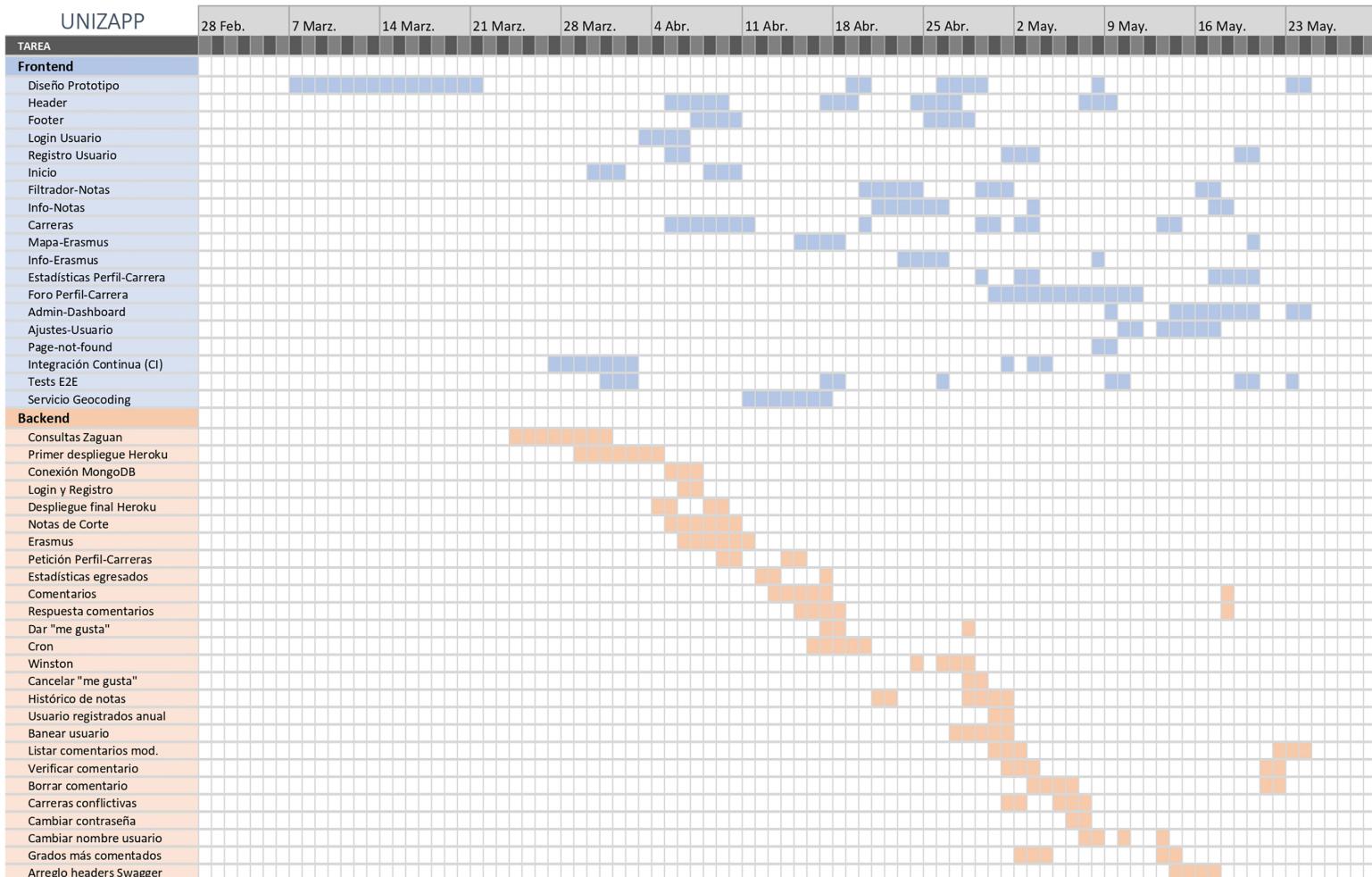


Figura 23: Hoja con el Diagrama de Gannt Backend-Frontend

15.4. Gestión del proyecto, Scrum

La gestión del proyecto se ha realizado utilizando la metodología ágil Scrum.

La motivación principal para utilizar esta metodología concreta ha sido que los integrantes Héctor Bara y Diego Marco ya la habían aplicado en otra asignatura de la carrera. A continuación se presentan los aspectos más relevantes que se han puesto en práctica como roles, reuniones y sprints. También se incluyen diagramas de burnup para visualizar cómo ha sido el rendimiento del equipo en cada sprint.

15.4.1. Roles

Diego Marco ha hecho de Scrum Master, ayudando al equipo a planificar las reuniones de los Sprints y a aplicar conceptos como la estimación de entradas de pila en base a puntos de historia.

El rol de product owner se ha repartido entre todos los integrantes del equipo al no tener un cliente concreto con el que interactuar.

15.4.2. Reuniones

Respecto a las reuniones, se hizo una planificación al principio del proyecto que nos permitiese acabar casi una semana antes de la entrega final para poder dedicarnos a la redacción de la memoria en ese espacio de días. Como se observa en la figura 24, para cada sprint hacemos una reunión de planificación, otra de revisión y una última de retrospectiva.

Hito	Fecha
Planificación proyecto.	23 Marzo
Planificación sprint 1	26 Marzo
Revisión sprint 1	8 Abril
Deadline sprint 1	8 Abril
Retrospectiva sprint 1	9 Abril
Planificación sprint 2	9 Abril
Revisión sprint 2	29 Abril
Deadline sprint 2	29 Abril
Retrospectiva sprint 2	30 Abril
Planificación sprint 3	30 Abril
Revisión sprint 3	20 Mayo
Deadline final	20 de Mayo

Figura 24: Fechas de la reuniones del equipo, aplicando Scrum

15.4.3. Pila de producto

Se adjunta el documento donde hemos ido manteniendo la pila de producto y el reparto de tareas: documento. También se incluye la pila de producto a continuación junto con la estimación en puntos de historia entre paréntesis:

1. Un usuario puede ver información sobre la aplicación. (5)
2. Un usuario puede filtrar las carreras de Unizar según su nota de corte. (40)
3. Un usuario puede ver información estadística sobre notas de corte (8)
4. Un usuario puede buscar carreras de Unizar específicas de Zaragoza, Huesca, Teruel y Almunia. (20)
5. Un usuario puede ver la página de perfil de una carrera (40)
6. Un usuario puede ver información sobre los erasmus. (8)
7. Un usuario puede ver un mapa de destinos erasmus. (13)
8. Un usuario puede registrarse en la aplicación. (13)
9. Un usuario puede loguearse en la aplicación. (13)
10. Un usuario logueado puede comentar en una carrera. (20)
11. Un usuario puede compartir el perfil de una carrera por correo. (20)
12. Un usuario logueado puede cambiar sus datos de perfil (8)
13. Un usuario administrador puede moderar un listado con los últimos mensajes publicados en los perfiles de carrera.(13)
14. Un usuario administrador puede ver una gráfica con el histórico de usuarios que se han dado de alta en el último año (13)
15. Un administrador puede banear a un usuario y borrar comentarios. (20)
16. (Desarrollo opcional): Integración continua en frontend con github actions.(5)
17. Pedir validación de captcha al registrar un usuario (5)
18. Un usuario administrador puede ver una gráfica con las carreras más conflictivas. (8)
19. Un usuario administrador puede ver una gráfica con las carreras más comentadas. (8)

15.4.4. Sprints

Durante el proyecto se planificaron un total de tres sprints. El primero de una duración de 15 días y los dos siguientes se ampliaron a 20 días cada uno. A continuación, para cada sprint se muestra un diagrama de burnup en el que se muestra: en el eje X los días que duró el sprint; en el eje Y, los

puntos de historia; en la línea amarilla, los puntos de historia completados por el equipo; en la línea azul, una línea idónea de puntos de historia completados; en la linea verde, el objetivo de puntos de historia para el sprint.

- **Sprint 1:** Para este sprint se planteó completar las entradas de pila 1, 9, 8 y 16. En total 71 puntos de historia.

Como se observa en la figura 25, acabamos completando 31 puntos de historia. No conseguimos llegar al objetivo planteado porque la mayor parte del tiempo la dedicamos a la configuración inicial del proyecto y al aprendizaje de los frameworks de desarrollo Angular y NodeJS.

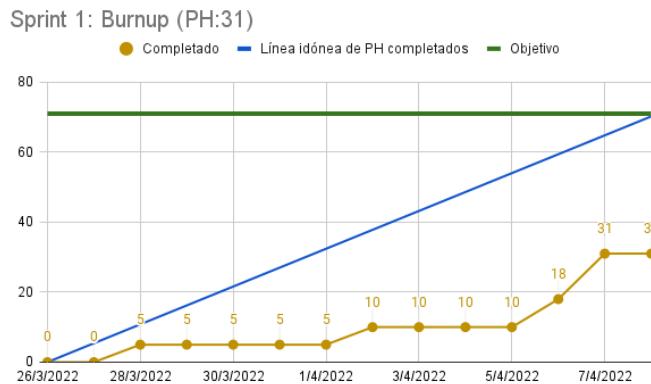


Figura 25: Diagrama de burnup para el sprint 1.

- **Sprint 2:** En este sprint ya no teníamos que configurar nada y podíamos dedicar todo el tiempo del sprint para completar funcionalidades. Es por ello que el objetivo fue más ambicioso que en el primer sprint y decidimos intentar completar 174 puntos de historia. Funcionalidades 7, 6, 4, 3 y 5.

Como se observa en la figura 26, acabamos completando 89 puntos de historia, 58 más que en el sprint anterior. Pese a no conseguir llegar al total planteado, consideramos este sprint un éxito ya que pudimos tener una gran cantidad de funcionalidad desplegada en producción.

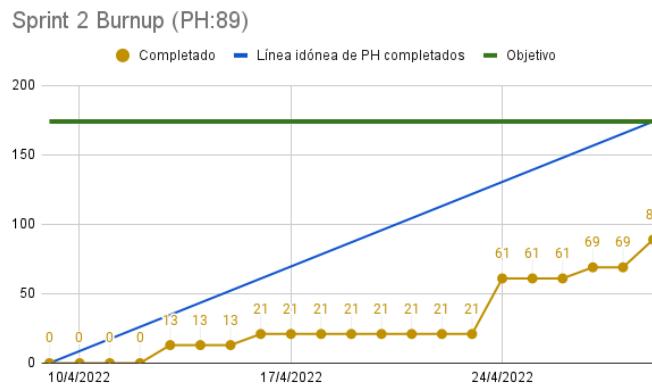


Figura 26: Diagrama de burnup para el sprint 2.

- **Sprint 3:** En este último sprint incluimos toda la funcionalidad restante, un total de 180 puntos de historia. Funcionalidades 2,10,11,12,13,14,15,17,18 y 19. Como se observa en la figura 27, al principio del sprint logramos terminar gran parte de la funcionalidad que no habíamos logrado terminar en el sprint 2. Finalmente, conforme se terminaba el sprint, logramos tener todas las funcionalidades completadas.

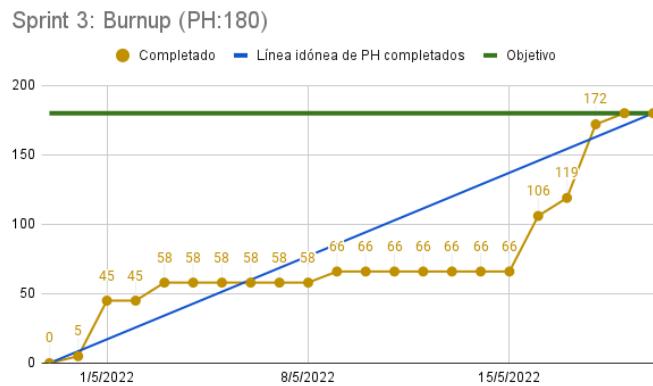


Figura 27: Diagrama de burnup para el sprint 3.

Referencias

- [1] “Search engine api zaguán.” <https://zaguán.unizar.es/help/hacking/search-engine-api>. Accedido: 22-05-2022.
- [2] “Papertrail.” <https://elements.heroku.com/addons/papertrail>. Accedido: 22-05-2022.