Back to Race

26/02/2022

Joaquín Jiménez Torralbo

David Sánchez Morales

Miguel Ángel Pérez Palma

IES FRANCISCO DE LOS RÍOS

ACCESO A DATOS

1. Visión general

El siguiente proyecto es solo la parte del Back-end de un proyecto más ambicioso, que consiste en el desarrollo de una aplicación que permita al taller Jiménez Ortiz premiar la fidelidad de las aseguradoras que recomienden el uso de dicho taller a sus asegurados.

La aplicación consiste en asignar a cada reparación que se haga en el taller, gracias a la recomendación de una aseguradora, una serie de puntos. Estos puntos se acumularan en dicha aseguradora que posteriormente podrán ser intercambiados por diferentes regalos ofrecidos dentro de un catálogo.

2. Objetivos

El objetivo principal es desarrollar una ApiRes que permita conectar el Front-end con la base de datos. De esta manera se atomiza el proyecto, con la idea de poder actualizar la aplicación sin necesidad de poner en riesgo la integridad del proyecto, ya que, solo será necesario intervenir en una de las diferentes partes que componen el conjunto de la aplicación.

Como subobjetivos se han establecido los siguientes

1. Almacenar la información necesaria en una base de datos de forma persistente.
2. Acceder a dicha información a través de la nube.
3. Tener una capa de seguridad entre el Front-end y el Back-end
4. Realizar de forma automática la gestión de los puntos.

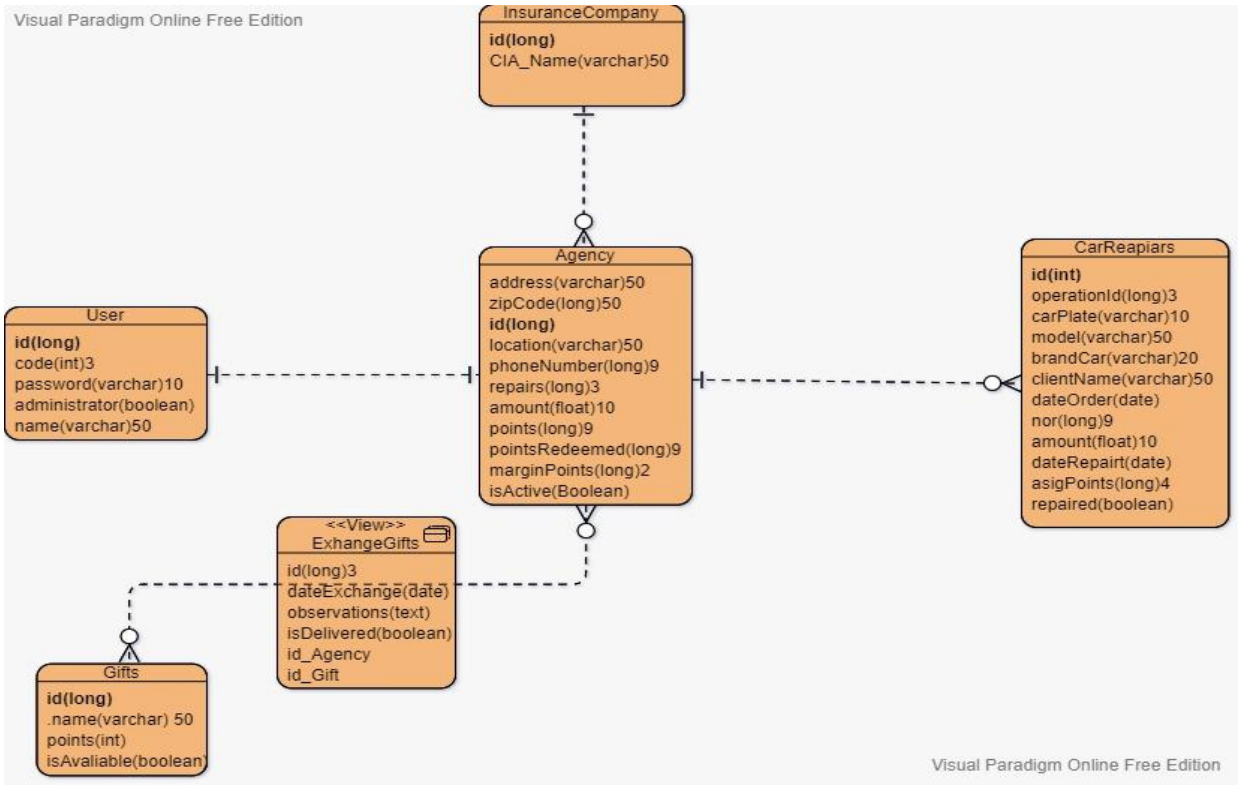
3. Especificaciones

3.1. Base de datos

Para almacenar la información, de forma persistente, que requiere la aplicación para su correcto funcionamiento se ha decidido usar la tecnología Postgresql, ya que permite el despliegue de esta en la nube a través de los servidores de Horoku de manera gratuíta. Aunque las pruebas iniciales se han desarrollado con la tecnología MariaDB de forma local, una vez superadas las primeras pruebas de funcionamiento se decidió dar el salto a un servidor en la nube, aunque se podía haber continuado con MariaDB desde un servidor AWS, al final, el equipo opto por migrar a Postgresql ya que permitía el acceso de forma indefinida y gratuita. Evitando de esta manera la responsabilidad de mantener un servicio propio continuamente activado.

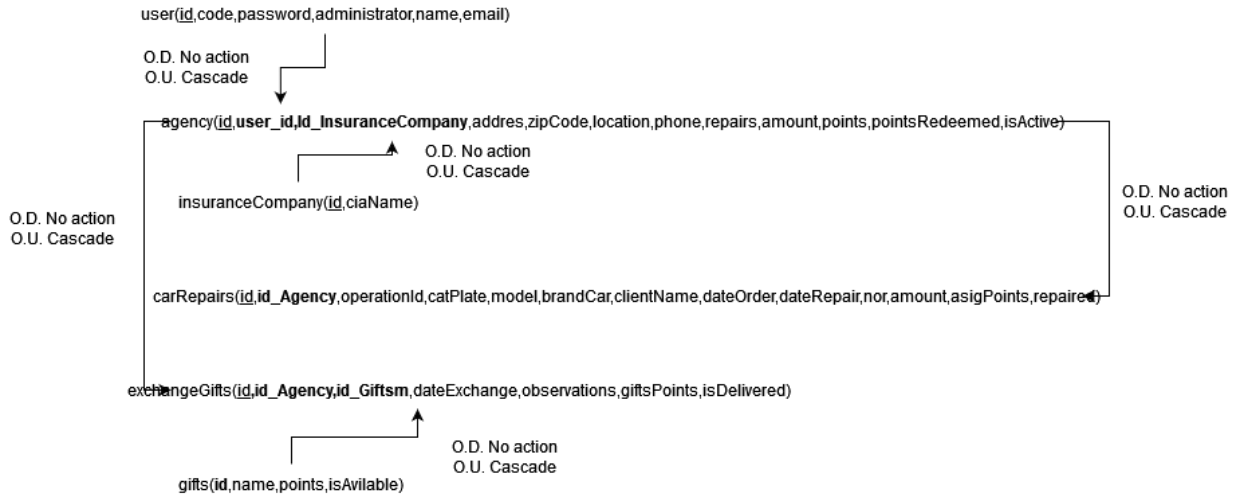
En cuanto al diseño de la base de datos finalmente se han establecido 5 entidades de las que se han obtenido 6 tablas. La entidad usuario hace referencia a todos aquellos que vayan a usar la aplicación, de esta se obtiene una entidad débil, en la que se almacena los datos de las agencias (relación 1,1). Esta división entre usuario y agencia se ha decidido por la necesidad de diferenciar entre los administradores de la aplicación y los usuarios convencionales.

Dado que el objeto principal de la aplicación es, que los usuarios puedan dar de alta coches para obtener puntos y que estos a su vez sean intercambiados por regalos. Se ha entendido que el centro de la base de datos será la entidad de agencia, desde la cual partirán todas las relaciones. Teniendo una relación (1,n) con las reparaciones de los coches y una (n,m) con los regalos, que dará lugar a una tabla que podrá ser usada como registro histórico de los regalos pedidos por la agencia. A su vez cada agencia dependerá de una compañía de seguros matriz con una relación (1,n).



Modelo entidad-relación

Como aspecto a reseñar del paso a tablas se ha optado por establecer todas las actualizaciones en cascada con el objetivo de que los cambios en una determinada tupla se transmitan al resto de tuplas relacionadas. En el caso del borrado de una tupla se ha optado por no realizar ninguna acción, dado que el cliente prefería conservar un histórico de las operaciones realizadas; es por esto que se decidió establecer un campo booleano en la entidad agencia que indique si esta se le permite el acceso a la aplicación o se a dado de baja de la misma. La misma idea se ha implementado en la tabla regalos, donde un booleano indica si un regalo puede ser intercambiado o no, con la idea de facilitar la administración para los productos estacionales o retirar del catálogo aquellos de los que no se disponga stock



Paso a tablas de la base de datos.

3.2. Tecnologías

Para crear la aplicación que implementará el servicio API Restful, hemos utilizado Java 15 y hemos trabajado en el IDE Eclipse principalmente. Hemos trabajado en un entorno Maven modular.

Para realizar el mapeado de la base de datos, se ha utilizado uno de los ORM (Object Relational Mappings) más populares: Spring Boot. Este a su vez integra el framework Hibernate con JPA.

Para almacenar la información, se ha utilizado una base de datos PostgreSQL desplegada en un servidor externo de Heroku.

Para el diseño, hemos utilizado principalmente Visual ParaDigm para el diagrama de clases y modelo entidad relación.

Con respecto a la comunicación, hemos utilizado WhatsApp, Discord y Git para el control de versiones.

La documentación pública de la API se ha desarrollado mediante la librería Swagger.

En la siguiente lista se resumen las tecnologías empleadas:

- Entornos de Desarrollo:
  - ➢ Java.
  - ➢ Eclipse.
  - ➢ Maven.
- Mapeo de Entidades:
  - ➢ Spring Boot (ORM).
  - ➢ Hibernate JPA.
  - ➢ PostgreSQL.
  - ➢ Heroku.
  - ➢ MariaDB.
- Trabajo en Equipo:
  - ➢ Git.
  - ➢ Discord.
  - ➢ WhatsApp.
  - ➢ Google Drive.
- Diseño:
  - ➢ Visual ParaDigm Online.
  - ➢ InVision.
- Otras tecnologías:
  - ➢ Cloudinary.
  - ➢ Log4Java.

- Swagger.
- Postman.

3.3. Puntos de acceso

Los diferentes clientes que quieran utilizar los servicios de la API deberán acceder a ella mediante unos puntos de acceso específicos. Estos puntos, basados en la tecnología HTTP, tendrán cada uno una función única, haciendo que los clientes no tengan libre albedrío a la hora de interactuar con la API.

Como en cualquier otro servicio de estas características, los tipos de peticiones recibidos vía HTTP pueden ser de varios tipos:

- GET: Sirve para recibir información de la base de datos.
- POST: En nuestro caso, hemos utilizado las peticiones POST no solo para crear registros en la base de datos, sino que también la hemos empleado para actualizar registros, por lo que nuestra API no necesita peticiones PUT.
- PUT: Normalmente se utilizan para actualizar registros. Nuestra API no utiliza este tipo de peticiones.
- DELETE: Utilizamos estas peticiones para eliminar registros de la base de datos.

Desglose de los diferentes ENDPOINT:

- GET:
  - /users: Devuelve todos los usuarios.
  - /users/available: Devuelve todos los usuarios disponibles a los que se le puede asignar una agencia.
  - /users/elements{elements}/page/{page}}: Recibe un número de elementos y un número de página. Devuelve usuarios paginados.
  - /users/name/{name}/elements/{elements}/page/{page}: Recibe un número de página y un nombre. Devuelve usuario por nombre paginado.
  - /users/admins/elements{elements}/page/{page}: Devuelve todos los usuarios administradores paginados.
  - /users/agencies/elements{elements}/page/{page}: Devuelve todos los usuarios con agencia paginados.
  - /agencies: Devuelve todas las agencias.
  - /agencies/elements{elements}/page/{page}: Recibe nº de página. Devuelve agencias paginadas.
  - /agencies/username/{username}/elements{elements}/page/{page}: Recibe nº de página y nombre agencia. Devuelve agencias paginadas y filtradas por nombre.

- /agencies/active/{active}/elements{elements}/page/{page}: Devuelve todas las agencias que coincidan con el parámetro isActive paginadas, pudiendo ser las que estén o no activas en función de lo que se reciba.
- /carRepairs: Devuelve todos las reparaciones de vehículos,
- /carRepairs/id/{id}:Devuelve la reparación correspondiente a dicho id.
- /carRepairs/elements{elements}/page/{page}: Recibe nº de página. Devuelve las reparaciones de vehículos paginadas.
- /carRepairs/operation/{operation}/elements{elements}/page/{page}: Recibe un nº de operación, devuelve las reparaciones del vehículo con ese nº paginadas.
- /carRepairs/carPlate/{carPlate}/elements{elements}/page/{page}: Recibe una reparación, devuelve reparaciones del vehículo con esa matrícula paginadas.
- /carRepairs/clientName/{clientName}/elements{elements}/page/{page}: Recibe el nombre del cliente. Devuelve una lista de las reparaciones de ese cliente paginadas.
- /carRepairs/ini/{ini}/end{end}/elements{elements}/page/{page}: Recibe una fecha inicio y final de orden y filtra devolviendo las reparaciones que se incluyan paginadas.
- /carRepairs/min/{min}/max/{max}/elements{elements}/page/{page}: Recibe una cantidad mínima y máxima de puntos y filtra devolviendo las que se incluyan paginadas.
- /carRepairs/repaired/{t-f}/elements{elements}/page/{page}: Recibe un boolean para filtrar los coches que en el campo reparado coincidan, devolviendo una lista de reparaciones paginada.
- /insuranceCompany: Devuelve todas las compañías de seguros.
- /insuranceCompany/elements{elements}/page/{page}: Recibe nº de página. Devuelve todas las compañías de seguros paginadas.
- /insuranceCompany/CIA_Name/{CIA_Name}: Recibe nombre de compañía: Devuelve la compañía con ese Nombre.
- /insuranceCompany/CIA_Name/{CIA_Name}elements{elements}/page/{page} :recibe un número de filas y un número de página, devuelve una lista con el número de elementos especificados y desde el elemento pertinente.
- /insuranceCompany/id/{id}: Recibe el id de la compañía y devuelve la compañía con ese id.
- /gifts: Devuelve todos los regalos.
- /gifts/elements{elements}/page/{page}: Recibe nº de página, Devuelve todos los regalos paginados.
- /gifts/name/{name}/elements{elements}/page/{page}: Recibe nº de página y el nombre del regalo, Devuelve todos los regalos con ese nombre paginados
- /gifts/avaliable/{t-f}/elements{elements}/page/{page}: Recibe un boolean y devuelve los regalos que coincidan en el campo disponible paginados.

- ➢ /exchangeGifts/gift/{id_gift}/elements{elements}/page/{page}: Recibe un regalo, devuelve los canjes de dicho regalo paginados.
- ➢ /exchangeGifts/isDelivered{t-f}/elements{elements}/page/{page}: Recibe un boolean y devuelve los regalos intercambiados según su estado.
- ➢ /exchangeGifts/agency/{agency}/elements{elements}/page/{page}: Recibe una agencia y filtra los canjes de esa agencia paginados.

- POST:
  - ➢ /users: Recibe un usuario. Crea un usuario y devuelve su id generado.
  - ➢ /agencies: Recibe una agencia.. Crea una agencia y devuelve su id generado.
  - ➢ /carRepairs: Recibe una reparación. Crea una reparación y devuelve su id generado.
  - ➢ /insuranceCompany: Recibe una compañía de seguros. Crea una compañía de seguros y devuelve su id generado.
  - ➢ /gifts: Recibe un regalo un archivo de imagen. Crea un regalo y devuelve su id generado. Además inserta la imagen en Cloudinary y modifica el regalo para añadirle el ID público asociado.
  - ➢ /exchangeGifts: Recibe un exchangeGift, lo inserta y devuelve el id generado.

- DELETE:
  - ➢ /users: Recibe un usuario. Elimina el usuario.
  - ➢ /agencies: Recibe una agencia. Elimina la agencia.
  - ➢ /carRepairs: Recibe una reparación. Elimina la reparación.
  - ➢ /insuranceCompany}: Recibe una compañía de seguros. Elimina la compañía de seguros.
  - ➢ /gifts: Recibe un regalo, Elimina el regalo.
  - ➢ /exchangeGift: Recibe un exchangeGift y lo elimina.

3.4 Seguridad

La api representa una capa de seguridad más de la aplicación. Su principal función es establecer una barrera que impida a un posible atacante acceder a la base de datos directamente, teniendo que realizar las peticiones a través de una capa intermedia entre la base de datos y el usuario.

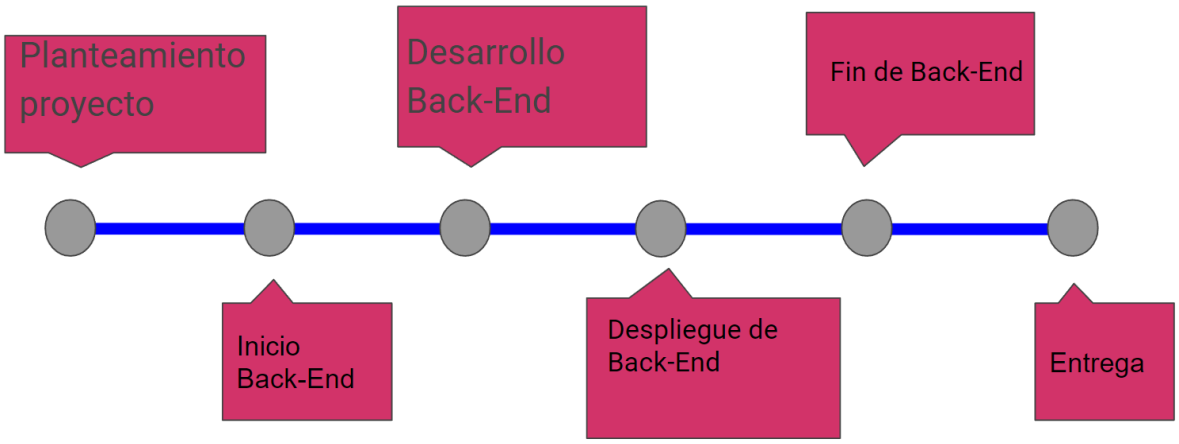
Como toda tecnología tiene sus ventajas e inconvenientes. Por un lado existe la desventaja de tener que estar conectado a internet para que la aplicación pueda funcionar, tanto para usuarios como para administradores; pero a su vez se obtiene la ventaja de que la base de datos se encuentra en un tercer sitio al que no se puede acceder si el ordenador o el móvil es vulnerable a un ataque informático.

Con respecto al uso de XML representa un salto cualitativo de gran envergadura. Ya que se ha pasado de un simple fichero que puede ser abierto dentro de un documento de texto con la información legible a simple vista. A una base de datos ubicada en otro dispositivo la cual sólo es accesible mediante peticiones http. A su vez tanto la lógica de negocio como la estructura de la base de datos no se encuentran dentro de la aplicación que está en manos del usuario, si no que se ha desarrollado en un programa aparte que no está funcionando dentro de su máquina, por lo que se hace más difícil el acceso a dicha información.

3.6. Log4Java

Para tener un control de errores adecuado se ha optado por tener un histórico de los errores que se produzcan durante la ejecución del programa. Para ello se decidió que estos errores fueran almacenados en un fichero mediante la tecnología Log4j2. Dicha herramienta permite que cuando se produzca una incidencia prevista en el control de errores de la Appi, ésta sea almacenada. De esta manera, además de conservar un histórico de las peticiones realizadas, facilitará enormemente la detección de errores de funcionamiento del código y agilizando su reparación.

1. Planificación Temporal



1. Planteamiento del proyecto: Los componentes del equipo se reúnen con el empresario para detallar la propuesta del proyecto y resolver aquellos puntos clave necesarios para el inicio del proyecto y se diseña un esquema de Entidad Relación y se asigna un reparto de tareas provisional.

2. Inicio de Back-End: Se estructura el proyecto con el uso Maven, se desarrolla el modelo, se asignan dependencias y se implementa GIT para desarrollar el proyecto en labor de equipo.

3. Desarrollo de Back-End: Se crean los repositorios, se desarrollan los controladores y los servicios, se implementa el uso de BD con MariaDB el proyecto.

4. Despliegue de Back-End: Se asignan los Endpoints, creamos una Base de datos con el uso de PostgreSQL, empieza el testeo en PostMan para el funcionamiento correcto del proyecto y se despliega el proyecto con Heroku

5. Fin de Back-End: Para finalizar el equipo centra sus esfuerzos en documentar el código, documentar la API con el uso de swagger, implementar un almacenamiento de control de errores con Log4Java, implementar un sistema de reducción de almacenamiento de base de datos con el uso de Cloudinary y una comprobación generalizada del funcionamiento correcto del proyecto.

2. Aspectos a reseñar

5.1. Documentación de la API

Para ayudar a los desarrolladores que en el futuro tengan que modificar el código, nuestro equipo ha decidido usar SWAGGER para documentar la API Restful. Para ello, se han documentado todos los métodos desarrollados dentro de los controladores.

5.2. Almacenamiento de fotos

Para reducir la carga interna de la base de datos implementada en Heroku, el equipo ha tomado la decisión de crear un repositorio externo que se conectará a un servidor externo al de la API para almacenar los archivos multimedia, en este caso, las imágenes de los regalos.

Para este servicio, hemos utilizado el servidor en línea gratuito Cloudinary. Para ello, hemos necesitado de descargar dependencias específicas para establecer la conexión con este servicio.

Existen 2 métodos específicos para la gestión de los archivos multimedia en línea:

- Upload: Almacena en la nube el fichero, y devuelve una ID pública con la información de la imagen, la cual posteriormente se utilizará para obtener la URL real y también para poder eliminarla. Este método se encarga de insertar y actualizar imágenes.
- Delete: Elimina un fichero en la nube filtrando mediante una ID pública.

De esta forma, el Front-End enviará un fichero vía HTTP al Back-End y este la insertará en la nube de forma automática.

5.3 Control de errores

Para evitar que la api deje de funcionar o se almacene en la base de datos información errónea se ha implementado un sistema de control de errores que permiten prevenir estos fallos. Para ello se han creado dos clases ,ServiceException que hereda de Exception y RecordNotFoundException que hereda de Exception; la primera sirve para controlar errores genéricos y la otra para errores concretos que afectan a la id.

El control de errores se ha implementado en el servicio, que lanza una excepción al controlador. En el controlador dentro del catch, para que el servicio avise cuando se ha

realizado una petición errónea, se devuelve una respuesta que indique al usuario que ha habido un fallo interno pero sin que le ofrezca más información de la necesaria.

3. Reparto de Tareas

6.1. GitHub

En el ámbito del reparto de tareas, para organizar el trabajo de cada miembro del equipo se ha usado la función de organización de proyectos de Github, la cual es muy similar a la que se puede encontrar en Trello, en la cual se crean Issues que se asignan a uno de los miembros del equipo.

Se ha organizado en 3 tablas.

- Por Hacer: En este punto se añaden todos los Issues que se están preparando para desarrollar en cada uno de los spring, se le asigna a cada issue uno de los miembros del equipo según se ha detallado en los daily meeting.

- En proceso: Esta es la fase intermedia en la cual se introducen los issues que actualmente se encuentran en desarrollo.

- Finalizado: Como su nombre indica aquí se traspasan los issues de la tabla en proceso que han sido finalizados, se cierra el issue, y en el momento en el que se termina un sprint se archivan todos los issues.

6.2. Asignación de Tareas

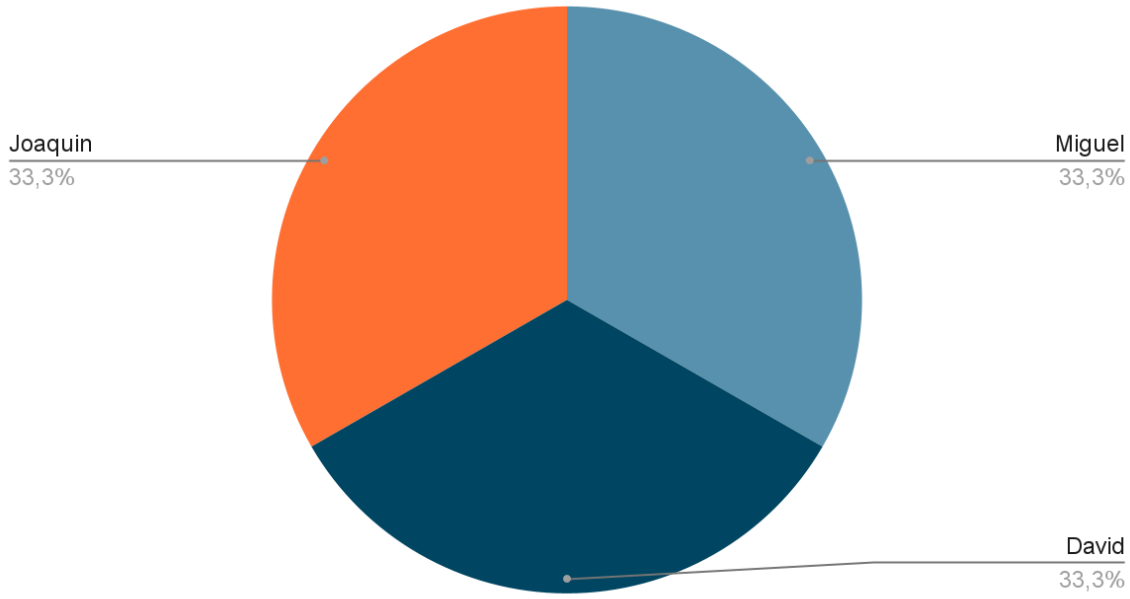
En el reparto de tareas nuestro equipo ha sido muy equitativo, según se decidió en la primera semana cada uno de los integrantes del equipo se encargaría de desarrollar 2 modelos y sus respectivas clases.

- Miguel Angel:
  - ➢ Modelos, Controlador, Repositorio y Servicio de ExchangeGift y User.
  - ➢ Clase Swagger y su correcto funcionamiento.
  - ➢ Endpoints de ExchangeGift y User.
  - ➢ Diseño de la Base de Datos.
  - ➢ Diagrama de Clases.

- David:
  - ➢ Modelos, Controlador, Repositorio y Servicio de ExchangeGift y Gift y Regalo.
  - ➢ Clase Cloudinary y su correcto funcionamiento.
  - ➢ Endpoints de Gift y Agency
  - ➢ Testeos en Postman.
  - ➢ Deploy a Heroku.
  - ➢ Diseño de la Base de Datos.

- Joaquin:
  - ➢ Modelos, Controladores, Repositorios y Servicios de CarRepair e InsuranceCompany.
  - ➢ Clase Log4Java y su correcto funcionamiento.
  - ➢ Control de Excepciones y Errores en el Servicio.
  - ➢ Endpoints de CarRepair e InsuranceCompany
  - ➢ Diseño de la Base de Datos.

6.3. Porcenta



4. Enlaces

7.1. GitHub

DavidSM24/Taller_AADD (github.com)

7.2. Servidor

https://talleraadd.herokuapp.com

7.3 Swagger

Swagger UI (talleraadd.herokuapp.com)