UNIVERSIDAD COMPLUTENSE MADRID

Facultad de Informática

Aplicación web de soporte al Aprendizaje-Servicio: gestión de ofertas y demandas de servicio

Web application for supporting Service Learning: management of service offers and service requests

Autores

Daniela-Nicoleta Boldureanu (Grado en Ingeniería del Software)
Victoria Gnatiuk Romaniuk (Grado en Ingeniería Informática)
Jesús Sánchez Granado (Grado en Ingeniería Informática)

Directores

Simon Pickin
Manuel Montenegro Montes

Curso 2020/2021

Índice general

1. Tecnologías utilizadas **7**
1.1. Node.js . . . 7
1.2. Angular . . . 7
1.3. GitKraken . . . 8
1.4. Pivotal tracker . . . 9
1.5. LaTeX . . . 9
1.6. MySQL . . . 9
1.7. Modelio . . . 10
1.8. MySQL Workbench . . . 10

2. DAO **11**

3. Conclusiones y trabajo futuro **13**
3.1. Introducción . . . 13
3.2. Objetivos cumplidos . . . 13
3.3. Problemas Encontrados . . . 14
3.4. Trabajo Futuro . . . 15

4. Conclusions and future work **17**
4.1. Introduction . . . 17
4.2. Objectives completed . . . 17
4.3. Problems Found . . . 18
4.4. Future work . . . 19

5. Contribución **21**
5.1. Jesús Sánchez Granado . . . 21

Índice de figuras

1.1. GitKraken: Información de una rama de desarrollo 8
1.2. Pivotal tracker: Estadísticas de historias 9

Capítulo 1

Tecnologías utilizadas

A continuación, se hablará sobre las tecnologías utilizadas explicando brevemente qué son y los motivos por los que han sido seleccionadas.

1.1. Node.js

Node.js [1] es un entorno de ejecución asíncrono dirigido por eventos. Funciona a base de promesas, es decir, funciones que devolverán un resultado en algún momento del futuro. Las promesas se pueden encadenar una tras otra, recibiendo cada una el resultado de la anterior.

Esta forma de conseguir concurrencia es distinta a la manera más común, que es utilizando cierres de exclusión mutua o candados. Los candados funcionan de la siguiente manera: si durante la ejecución de un programa concurrente un elemento es compartido por varios hilos, el resultado dependerá de las operaciones y el orden en que se lleven a cabo las mismas en dicho elemento. Para evitar que dos hilos accedan de manera simultánea a un mismo recurso, hay que "bloquear" ese recurso utilizando candados, llegando a la situación conocida como exclusión mutua, pero cuando hay varios procesos concurrentes puede darse la situación de que el proceso A esté esperando a que el proceso B libere un recurso, y al mismo tiempo B espera que A libere un recurso. Esta situación se conoce como *deadlock* o interbloqueo.

Node.js tiene un modelo de concurrencia asíncrono en el cual el programador no tiene que manejar directamente cierres de exclusión mutua o candados, por lo que no se puede alcanzar el estado de deadlock. Tampoco maneja explícitamente hilos de ejecución, lo que previene condiciones de carrera. Esto hace de Node.js una elección adecuada para desarrollar sistemas escalables.

Otro motivo para continuar con esta tecnología es que Daniela ya tenía conocimiento previo de este entorno y es una tecnología que venía impuesta por el proyecto.

1.2. Angular

Angular [2] es un *framework* para la construcción de SPA (*Single Page Application*) o aplicaciones de página única que utiliza HTML y Typescript. Angular sigue el patrón modelo-vista-controlador, el cual consiste en separar la aplicación en tres partes:

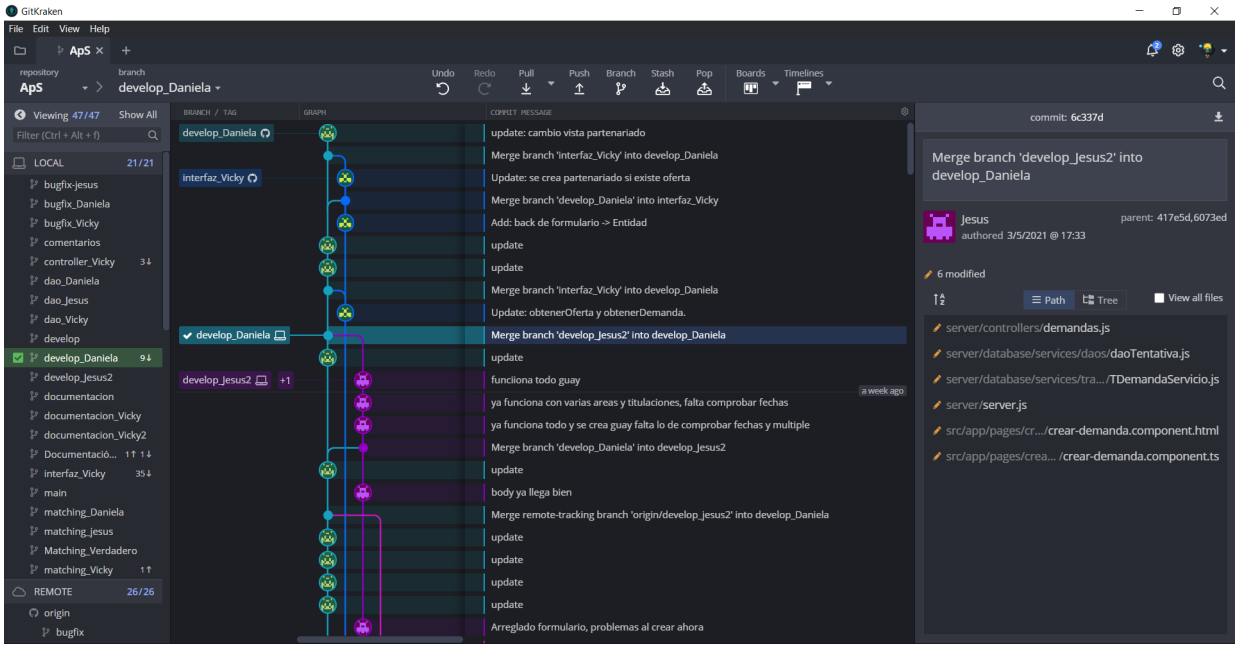

Figura 1.1: GitKraken: Información de una rama de desarrollo

- Modelo: Es la piedra angular del patrón, se encarga de manejar los datos y la lógica de la aplicación.
- Vista: Es la parte que se le muestra al usuario.
- Controlador: Es la parte que se encarga de comunicar a la vista y al modelo. El controlador recibe los eventos de interacción del usuario a través de la vista y se los pasa al modelo, el cual hace las operaciones necesarias y devuelve los resultados al controlador, quien se los pasa a la vista para mostrárselos al usuario.

El uso de Angular venía impuesto por el trabajo realizado con anterioridad y, aunque es una tecnología con la que ningún miembro del equipo estaba familiarizado, es cierto que el diseño de aplicación de página única hace mucho más liviana la ejecución de la aplicación por parte del usuario, al no tener unos tiempos de espera tan grandes como los que tendría al cargar de nuevo cada página. Esto hace de Angular una buena elección para un trabajo de esta índole.

1.3. GitKraken

Gitkraken es un cliente de Git con interfaz de usuario gráfica la cual se puede conectar a distintas plataformas de Git. Gitkraken hace de intermediario entre el usuario y el repositorio de Git, el cual en este caso está alojado en Github. Hemos escogido esta interfaz para nuestro control de versiones porque permite trabajar desde Windows sin necesidad de conocer los comandos de Git. A diferencia de otros clientes de Git, tiene una representación gráfica muy intuitiva que permite ver la distribución de las ramas, los *commits* y su evolución, como se puede observar en la Figura 1.1. Además, permite resolver los conflictos generados al mezclar las distintas ramas de desarrollo dentro de la propia aplicación de una manera bastante sencilla. Esto, sumado a la experiencia previa de Victoria con la aplicación, ha hecho que sea seleccionada como herramienta de control de versiones.

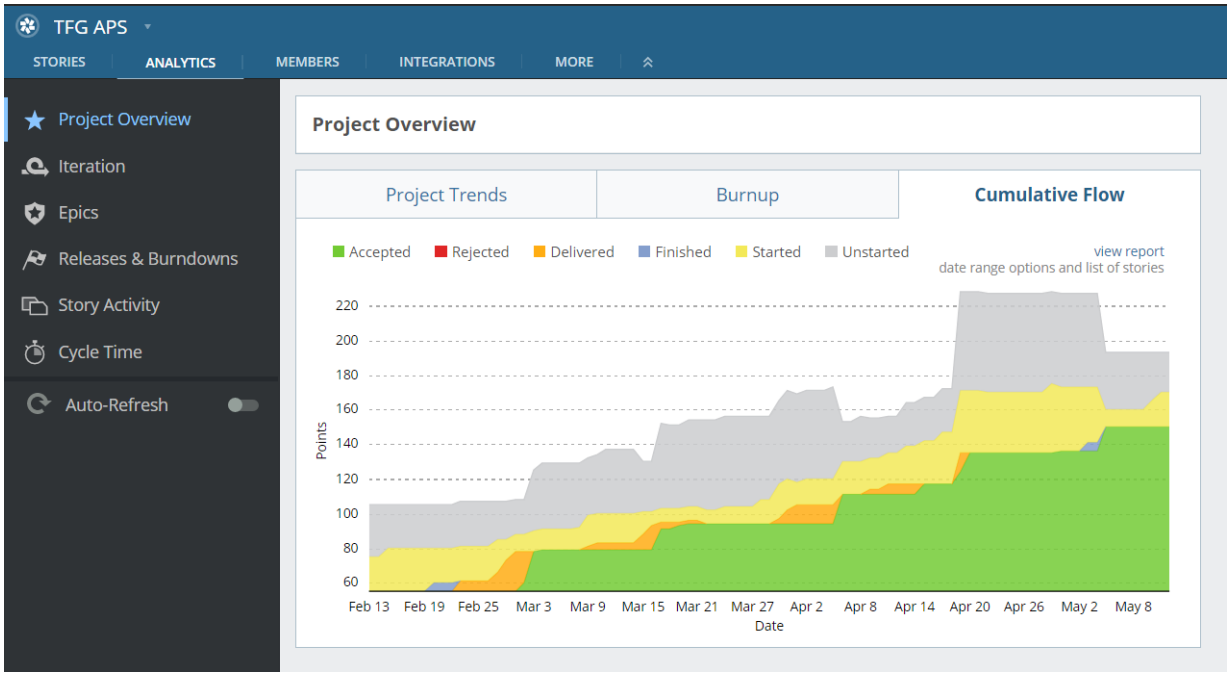

Figura 1.2: Pivotal tracker: Estadísticas de historias

1.4. Pivotal tracker

Pivotal tracker es una herramienta de *product planning* y administración de tareas diseñada para equipos de desarrollo que siguen metodologías de diseño ágiles. Esta herramienta permite crear historias de usuario y asignarles una puntuación del 1 al 5 indicando su dificultad y/o tiempo invertido en dichas tareas. Además, permite cambiar el estado de las tareas (empezado, finalizado, en revisión, etc.) y cualquier cambio en el estado de dichas tareas se informa por correo de manera automática a quien esté involucrado en ella. También permite ver las tareas completadas y rechazadas y generar gráficos indicando el esfuerzo realizado, como el que podemos ver en la figura 1.2. Esta tecnología fue sugerida por Victoria y nos ha facilitado mucho tanto la organización como el seguimiento de nuestros avances.

1.5. LaTeX

LaTeX es un lenguaje de creación de documentos escritos utilizado comúnmente en el mundo académico, que es una de las principales razones por la que lo hemos escogido para redactar nuestra memoria, a pesar de que ningún integrante del grupo tuviera experiencia previa con ello. A diferencia de otros procesadores de texto, como Microsoft Word o LibreOffice Writer, se escribe el texto plano y se formatea dicho texto con etiquetas.

1.6. MySQL

Aunque nuestro proyecto continúa el trabajo realizado por David Jiménez del Rey, el cual ya contaba con un sistema gestor de bases de datos, dicho sistema era MongoDB. Como los datos que se iban a manejar en la aplicación eran en su mayoría relacionales, se tomó la decisión de utilizar MySQL para la base de datos. Dado que todos los componentes

del grupo tenían experiencia previa en bases de datos SQL fue un cambio bien recibido.

1.7. Modelio

Modelio es un entorno de modelado *open-source* el cual permite trabajar con un amplio rango de modelos y diagramas. Dado que ya se contaba con experiencia previa en esta herramienta por parte de todos los miembros del equipo, se ha escogido para realizar los modelos de datos necesarios para la aplicación.

1.8. MySQL Workbench

MySQL Workbench es una herramienta para diseño, desarrollo y administración de bases de datos relacionales. Cuenta con funcionalidades de validación de esquemas y modelos y promueve las mejores prácticas de los estándares de modelado de datos. También promueve los estándares de diseño específicos de MySQL para evitar errores al generar esquemas relacionales o creando bases de datos MySQL. Por estos motivos, junto con su relativa simplicidad, es por lo que se ha elegido esta herramienta para hacer los diagramas de entidad-relación.

Capítulo 2

DAO

Tras cambiar la base de datos de MongoDB por una relacional, también era necesario reimplementar la lógica de accesos a la base de datos, por lo que se crearon cuatro DAOs que se encargarían de las operaciones de cada una de las cuatro áreas definidas en el modelo de entidad-relación.

El DAO es un patrón de diseño que trata de proporcionar una interfaz para la comunicación con una base de datos u otro sistema de persistencia de datos. Esta interfaz se encarga de llevar a cabo las operaciones CRUD, es decir creación, lectura, actualización y eliminación de datos y además asegura la independencia entre la lógica de la aplicación y la capa de negocio.

Aunque no eran estrictamente necesarios, dado que en Javascript no hace falta declarar el tipo de los objetos, se decidió crear objetos *transfer* para así tener más documentados los campos de cada tipo de objeto. Un *transfer* o *Data Transfer Object* es un objeto cuya única función es guardar la información de cierto objeto y permitir su acceso y manipulación. De esta forma, si hay algún problema, este se detectará cuanto antes y evitará que la aplicación falle repentinamente en fases más avanzadas de la ejecución.

Los objetos *transfer* contienen simplemente los atributos deseados de cada tipo de objeto además de las funciones *get* y *set* para poder acceder y actualizar la información de dichos atributos. En conjunto con los DAO, los *transfer* ayudan aún más a la separación de capas de negocio y lógica.

Los cuatro DAO que se crearon a partir del diagrama entidad-relación son:

- `DAOColaboracion`: Se encarga de manejar toda la información relacionada con los proyectos y los partenariados, desde sus participantes, ya sean profesores o estudiantes, hasta los mensajes y archivos asociados a estos proyectos o partenariados. Este DAO se llama así porque tiene como piedra angular la clase `Colaboración`. Esta clase fue creada para hacer de padre de las clases `Partenariado` y `Proyecto` y así evitar la repetición de métodos y atributos similares. Utiliza los *transfer* `TColaboracion`, `TPartenariado` y `TProyecto`.

- `DAOComunicacion`: Se encarga de manejar toda la información relacionada con todas las formas de comunicación disponibles, desde los mensajes y los *uploads* que se pueden intercambiar durante las distintas fases de un partenariado o proyecto hasta los *emails* o las *newsletter* a las que se pueden suscribir los usuarios. Por lo tanto utiliza los *transfer* `TUpload`, `TMensajes`, `TMail` y `TNewsletter`

- `DAOTentativa`: Se encarga de manejar toda la información relacionada con ofertas y demandas y sus relaciones con la titulación local ofrecida por la universidad, las

áreas de servicio y las necesidades sociales que pudiera tener la demanda. Al igual que antes, se creó una clase padre llamada `Anuncio` para evitar la repetición de atributos en las clases `Oferta` y `Demanda` y en sus derivadas. Este DAO también se encarga de las iniciativas, que son propuestas de proyecto realizadas por un estudiante a la espera de que se le dé el visto bueno, y de los mensajes y *uploads* que pudieran tener tanto la oferta como la demanda. Para poder llevar a cabo esta función, este DAO utiliza los *transfer* `TIniciativa`, `TOfertaServicio`, `TAnuncioServicio` y `TDemandaServicio`.

- `DAOUsuario`: Se encarga de manejar los datos pertenecientes a las distintas clases de usuario, que son: profesor interno, profesor externo, estudiante interno, estudiante externo, *admin*, socio comunitario y oficina ApS. Además de estas clases, también interactúa con los respectivos padres de cada una de ellas y con las titulaciones locales, áreas de conocimiento y universidades que son necesarias para completar los atributos de los profesores. Para ello utiliza los *transfer* `TAdmin`, `TEntidad` (que deberá cambiarse en un futuro por `TSocioComunitario`), `TUsuario`, `TProfesor`, `TOficinaAPS`, `TEstudiante`, `TProfesorExterno`, `TProfesorInterno`, `TEstudianteInterno` y `TEstudianteExterno`.

Se ha intentado "estandarizar" los DAO para evitar que en un futuro haya que actualizarlos con regularidad, pero debido a la posibilidad de nuevas funcionalidades o de cambios en el modelo de datos, esto es imposible. Por este motivo, es posible que sea necesario actualizarlos si la aplicación sufre cambios.

Capítulo 3

Conclusiones y trabajo futuro

3.1. Introducción

En esta última sección se hablará sobre el estado actual del proyecto y sus objetivos, así como de las mejoras que se podrían realizar sobre el mismo en un futuro.

Dado que este TFG es una continuación del proyecto de David Jiménez del Rey, se comenzará exponiendo las principales diferencias del proyecto actual con este último.

- La base de datos: además de haber cambiado la base de datos de MongoDB a una base relacional, se han realizado varios cambios en el modelo como queda explicado en la sección **??**.
- La capa de acceso a datos: siguiendo el cambio en la base de datos, se han creado DTO y DAO para llevar a cabo la lógica de accesos a la BD como queda explicado en la sección 2.
- Implantación de un sistema de *matching*: el proyecto del año pasado trataba las ofertas y las demandas como simétricas, es decir, teniendo los mismos atributos, pero esto no es así, por lo que se ha diseñado un sistema de *matching* como queda explicado en la sección **??**.
- Cambios en los formularios: debido a los cambios registrados en el modelo de datos, ha sido necesario crear algunos formularios o adaptar otros ya existentes. Esto queda mejor explicado en la sección **??**.

3.2. Objetivos cumplidos

A continuación, se repasarán los objetivos de este trabajo y su completitud:

- *Construir unas bases sólidas del proyecto, creando un modelo de dominio que aclara los conceptos implicados en la aplicación y un modelo de datos que enriquece el modelo de dominio y plasma como la aplicación gestiona la información.*
  Este objetivo ha sido cumplido y los diagramas de dominio y de datos se pueden ver en las Figuras **??** y **??**, respectivamente.
- *Crear un modelo relacional que muestre la estructura de la base de datos, facilitando su entendimiento y manejo a los futuros desarrolladores del proyecto.*

Este objetivo también ha sido cumplido y el diagrama resultante se puede observar en la Figura **??**

- *Crear una base de datos relacional compleja y rica en detalles.*
  Este objetivo ha sido completado en su mayoría, aunque debido a la terminología empleada es posible que haya que renombrar alguna tabla. Esto se explicará con más detalle en la sección Trabajo Futuro.

- *Implementar cuatro DAO que realicen la lógica de acceso y gestión de datos, encapsulando el acceso a la base de datos. Crear transfers que permitan estructurar y manejar de forma sencilla los datos de la BD.*
  Este objetivo también ha sido completado, aunque, como ya se ha dicho con anterioridad, es prácticamente imposible crear un DAO "perfecto" y es posible que se deba modificar en un futuro si se agregan funcionalidades nuevas a la aplicación.

- *Implementar un sistema de matching entre las ofertas de servicio planteadas por un profesor y las demandas de servicio planteadas por un socio comunitario que determina que porcentaje de encaje tienen.*
  Este objetivo también ha sido completado y el sistema de *matching* queda explicado en la sección 7.

- *Adaptar las páginas de registro y perfil del usuario al nuevo sistema, e implementar formularios para la creación de ofertas, demandas y partenariados.*
  Este objetivo no ha sido totalmente completado. Aunque los formularios de registro y perfil de usuario han sido actualizadas, y se han creado formularios para la creación de demandas y ofertas, en el caso de los partenariados, se plantean tres formas para poder crear un partenariado: *match* entre una oferta y una demanda ya creadas, un profesor decide respaldar una demanda, y un socio comunitario decide aceptar una oferta. Cada una de estas formas requeriría dos formularios, uno para el profesor y otro para el socio comunitario. Se ha hecho el formulario de *match* para el profesor y se ha empezado el desarrollo de la parte de *back-end* para el formulario de la parte del socio comunitario. Esto se debe a la complejidad de trabajar con Angular para un equipo sin experiencia previa y a la falta de tiempo.

- *Corregir bugs encontrados en el proyecto precedente.*
  Este objetivo se da como completado dado que la gran mayoría de bugs encontrados fueron arreglados al principio del proyecto, y aunque quedaban algunos, al modificar los formularios y tener que hacer cambios en las interfaces, los bugs ya serán corregidos en la reimplementación de estos formularios.

3.3. Problemas Encontrados

A continuación, se describirán las dificultades y problemas encontrados durante la realización de este proyecto.

La principal dificultad ha residido en la complejidad de trabajar con Node.js y con Angular, y la relativa poca experiencia previa del equipo con estas tecnologías. El equipo comenzó a familiarizarse con las tecnologías antes de comenzar el curso, pero aun así eran tecnologías complejas para quienes no habían trabajado con ellas previamente. Además, al dedicarse la mayoría del tiempo a la especificación y diseño de la base de datos y al diseño e implementación de las funcionalidades de *back-end* , cuando se cambió a trabajar

en el *front-end* , en el que se usa Angular, hubo que volver a "aprender" a trabajar con Angular.
A pesar de las dificultades aquí comentadas, se ha aprendido que Angular y Node.js son tecnologías muy acertadas para el diseño de aplicaciones web, tanto por la arquitectura SPA que ofrece Angular como por la concurrencia alcanzada con Node.js.

3.4. Trabajo Futuro

Aunque se hayan completado la mayoría de los objetivos que tenía este proyecto, la aplicación aún no está lista para su despliegue y uso público, pero con otro año más de trabajo debería estar lista. A continuación, se enumeran algunas mejoras para el proyecto con este fin:

- Adaptar la interfaz y terminar los formularios: Como se ha explicado anteriormente al repasar los objetivos, todavía quedan formularios por hacer para cubrir todos los casos en los que se puede crear un partenariado, los cuales serían: el formulario de *match* del socio comunitario, el formulario para que un profesor pueda respaldar una demanda sin haber creado todavía una oferta, el formulario para que el socio comunitario aceptara la propuesta de dicho profesor, el formulario para que un socio comunitario pudiera aceptar una oferta sin haber creado todavía una demanda y el formulario para que el profesor diera su visto bueno a esta proposición. Tras esto, el partenariado se encontraría en el estado EN_NEGOCIACIÓN, y habría que hacer otros dos formularios (uno para el profesor y otro para el socio comunitario) para poder pasar al estado ACORDADO. Además, para poder crear un proyecto a partir de un partenariado, serían necesarios, al igual que en el caso anterior, otros dos formularios más.
  Debido a los cambios que ha sufrido la base de datos y a la implementación de los DAO, algunas vistas de la aplicación han dejado de ser funcionales. Será necesario cambiar los controladores para que en lugar de utilizar la lógica de acceso a base de datos de MongoDB utilicen la lógica proporcionada por el nuevo *back-end* .

- Implementar un sistema de mensajería: Dado que la aplicación la van a poder usar tanto profesores como estudiantes y socios comunitarios y la comunicación será bastante importante, tanto en los periodos de definición del proyecto como cuando se empiece a trabajar en él, consideramos necesario que se implemente un sistema de mensajería o chat para hacer lo más eficiente posible la comunicación entre los usuarios. Incluso se podría implementar un sistema de foros para que los usuarios compartieran temas relacionados con el ApS o incluso para que los estudiantes pudieran ayudarse unos a otros en cuanto a temas algo más generales.

- Agregar parámetros opcionales en los formularios: Actualmente, algunos de los parámetros que ha de introducir el usuario en los formularios para crear ofertas y demandas podrían dejarse como opcionales. Esto podría interferir con el sistema de *matching*, pero no se ha tenido tiempo para tenerlo en cuenta.
  Además, el año de las fechas de la oferta debería ser opcional. En caso de que el profesor no introdujera año, eso significaría que la oferta es vigente de manera cíclica todos los años en los periodos de tiempo que tenga establecidos. De nuevo, esto debería haber sido implementado pero no se ha tenido tiempo. Los campos que deben ser obligatorios en los formularios han sido marcados con asteriscos.

- Implementar un "*matching* manual": Puede darse el caso de que un usuario quiera respaldar/aceptar una oferta o demanda, teniendo él vigente una oferta o demanda que sería adecuada para la situación, pero la aplicación no ha hecho el *match* todavía. En este caso, la aplicación debería preguntar al usuario si tiene vigente una oferta o demanda que corresponda con la que quiere respaldar/aceptar y, en caso afirmativo, le pediría seleccionar la oferta o demanda en cuestión. Tras este *match* el atributo dummy debería quedar a false.

- Mejorar el método de *matching* encargado de buscar coincidencias entre textos: El algoritmo empleado en este proyecto para el procesamiento del lenguaje natural, podría ser sustituido por alguna de las librerias NLP ya existentes. De esta forma se podrían obtener métricas de similitud de texto tales como las de Soft Cosine Measure.

- Implementar nuevos usuarios: En las fases finales de este proyecto han surgido algunos usuarios nuevos, los cuales todavía no han podido ser implementados. Estos usuarios son:

  - Colaborador: Tendrá los mismos permisos que un profesor externo pero carecerá de la capacidad de calificar a estudiantes.
  - tutor CA: Este usuario corresponde a los tutores de los centros asociados a la UNED y por ello tendrá los mismos permisos que un profesor interno.

  Con estos dos nuevos usuarios, cambia también la jerarquía de clases en lo que a profesores respecta. Ahora colaborador y profesor externo heredan de la clase Promotor externo, y profesor interno y tutor CA heredan de la clase promotor interno. Tanto promotor interno como promotor externo heredan de promotor, y aunque estos cambios se pueden ver en el modelo de datos, todavía no han sido realizados en la base de datos, por lo que deberán realizarse estos cambios.

- Añadir a la izquierda de los campos de los formularios una opción para obtener más información sobre cada uno de ellos.

- Implementar la obtención de las coincidencias entre las titulaciones que el socio comunitario ha propuesto como posibles y las titulaciones a las que pertenecen la/s asignatura/s identificadas en la oferta como asignatura/s objetivo.

Capítulo 4

Conclusions and future work

4.1. Introduction

This last section will be about the actual state of the project and its objectives, as well as the improvements that could be made in this project in the future.

Since this project is a continuation of David Jimenez del Rey's project the main differences between the two projects are going to be explained.

- The database: besides changing the database from MongoDB to a relational database, various changes have been made in the model as explained in section **??**.
- The data access layer: following the changes made in the database, DTO and DAO have been created to handle the database access logic as explained on section 2.
- Implementation of a matching system: last year's project treated offers and demands as if they were symmetrical, that is, as if they had the same attributes, which was not the case, so a matching system was designed as explained in section **??**.
- Changes in the forms: because of the changes that the data model underwent, new forms have been needed and old ones have been modified. This is better explained in section **??**.

4.2. Objectives completed

Next, the objectives of this project and its completion will be reviewed:

- *Build a solid foundation for the project, creating a domain model that clarifies the concepts involved in the application and a data model that enriches the domain model and reflects how the application manages information.*
  This objective has been met and the data and domain diagrams can be seen on Figures **??** and **??**, respectively.
- *Create a relational model that shows the structure of the database, facilitating its understanding and management by future project developers.*
  This objective has also been met and the resulting diagram can be seen on Figure **??**

- *Create a complex and detailed relational database.*
  This objective has been mostly completed, but, due to the terminology used some tables on the database may need to be renamed. This will be explained in more detail in the future work section.

- *Implement four DAOs that perform the logic of access and data management, encapsulating the access to the database. Create transfers that allow the database data to be structured and managed in a simple way.*
  This objective has also been met, even though, as it has been said before, it is impossible to create a "perfect" DAO and it may be modified in the future if new functionalities are added to the application.

- *Implement a system of matching between the offers proposed by a teacher and the demands solicited by a community partner that determines what percentage of fit they have.*
  This objective has also been met and the matching system is explained in chapter **??**.

- *Adapt the registration and user profile pages to the new system, and implement forms for the creation of offers, demands and partnerships.*
  This objective has not been totally completed. Even though registration and user profile pages have been actualized, and new forms have been made for the creation of demands and offers, in the partnership case, three ways of creating a partnership exist: match between an offer and a demand created beforehand, a teacher decides to support a demand, and a community partner decides to support an offer. Each of these ways needed two forms, one for the teacher and another for the community partner. The form for the teacher in the match way has been made and the development of the Back-End for the community partner form has begun. This is due to the complexity of working with Angular for a team without previous experience in said technology and due to the lack of time.

- *Correct bugs found in the previous project.* This objective is given as completed on account of most of the bugs being fixed in the early stage of the project and, even though some of them remained, since the interfaces and forms will be modified, the rest of the bugs will be fixed in the reimplementation of said parts.

4.3. Problems Found

Next, the difficulties and problems found during this project will be described.

The main difficulty has been the complexity of working with Node.js and Angular, and the team's relative lack of previous experience with these technologies. The team started working with these technologies before the scholar year started, but even then, the technologies were very complex. Furthermore, since most of the time was spent on the design and development of the database and the back-end functionalities, when the team started working in Front-End, which is where Angular is used, they had to "relearn" Angular.
Despite the difficulties, it has been learned that Angular and Node.js are very successful technologies for the design of web applications, both by the SPA pattern offered by Angular as the concurrency achieved with Node.js.

4.4. Future work

Even though most of the objectives of this project have been completed, the application is not ready yet for its deployment and public use, but with another year of work, it should be. Next, some improvements for the project are enumerated:

- Adapting the interface and finishing the forms: As it has been explained before while reviewing the objectives, there are still forms to do to cover all the cases in which a partnership can be created: the match form for the community partner, the form needed for a teacher to support a demand without having made an offer yet, the form needed for a community partner to accept the proposal of said teacher, the form needed for a community partner to accept an offer whitout having created a demmand yet and the form needed for the teacher to accept the proposal of said community partner. After this the partnership would be in "EN_NEGOCIACIÓN" state, and it would take two more forms (one for the teacher, the other for the community partner) so it could advance to " ACORDADO" state. Furthermore, two more forms would be needed for the creation of a project from a partnership.
  Due to the changes in the database and the DAO implementation, some pages are no longer functional. It will be necessary to change the controllers in order to use the new database access logic instead of the old one, which was prepared for MongoDB.

- Implement a messaging system: Since the application is going to be used by teachers, students and community partners and communication will be key, both in the definition of a project and in the realization, we deem necessary the implementation of a messaging or chatting system in order to make the communication between users as efficient as possible. A forum system could even be implemented so the users could share information related to ApS or even for students to be able to help each other in more general topics

- Add optional parameters in the forms: Now, some of the parameters that the user needs to introduce in the forms so he can create demands or offers could be optional. This might interfere with the matching system, but there was no time to implement countermeasures in this case.
  Another improvement would be letting the year set in the offer's dates be optional. In case that the year was left empty, this would mean that the offer is available every year in the periods of time set. Again, this should have been implemented but there was not enough time. the parameters that should be mandatory in the forms have been marked with an asterisk.

- Implement a "manual matching": Sometimes, an user may want to accept/support an offer or a demand, having already made an offer or a demand that would be adequate for the situatiom, but without the application having done the match yet. In this case, the application should ask the user if he has an offer or demand that corresponds with the one he wants to accept/support and, if so, it would ask him to select it. After this match, the dummy attribute should be put to false.

- Upgrade the matching method tasked with searching for coincidences in texts: The natural language processing algorithm used in this project could ba substituted by one of the existing NLP libraries. This way we could obtain metrics like the Soft Cuisine Measure.

- Implement new users: In the final stages of this project some new users have emerged, those users have not been implemented yet. the users are:

- Colaborator: this user will have the same permissions as an external teacher, but he will not be able to qualify students.
- CA tutor: this user corresponds to the tutors of the centers associated with UNED and will have the same permissions as an internal teacher.

The arrival of these new users also changes the class hierarchy for the teachers. Now colaborator and external teacher inherit from the class external promoter, and internal teacher and CA tutor inherit from the class internal promoter. Both promoters inherit from the class promoter, and even though these changes can already be seen in the data model, have yet to be implemented on the database.

Capítulo 5

Contribución

A continuación, se detallan las contribuciones de cada uno de los componentes del grupo ordenados por orden alfabético.

5.1. Jesús Sánchez Granado

La primera fase del proyecto era principalmente una fase de investigación, así que en esta fase todos hemos leído la memoria de David Jiménez del Rey y se ha aprovechado para preparar el entorno de trabajo y crear el repositorio de Github al cual se irán subiendo los avances en el proyecto. Además, de manera conjunta se realizaron pruebas de robustez en la aplicación web con el objetivo de intentar encontrar *bugs* o posibles mejoras.

En la segunda fase del proyecto Jesús ha arreglado algunos *bugs*. Uno de ellos era un mal redireccionamiento de un enlace. Otro, que más que un *bug* era una mejora, pues al editar una iniciativa siempre pedía aceptar los términos y condiciones, cosa que se consideró innecesaria.
Algunos de estos *bugs* se han dejado por hacer pues al tener que realizar cambios en la interfaz, los *bugs* ya serían arreglados al reimplementar las vistas pertinentes.

En la tercera fase del proyecto, Jesús se ha encargado de crear la estructura de las tablas **profesorinterno_oferta, estudiante_iniciativa, estudiante_proyecto, upload, upload-anuncioservicio, mail, uploads-colaboracion, mensaje, mensaje-anuncioservicio y mensaje-colaboracion**. En dicha estructura quedaban especificados los nombres y tipos de sus campos, sus restricciones, las claves foráneas y quedaban definidas también sus relaciones, siempre y cuando fueran con tablas de este grupo ya nombrado.
Una vez se crearon las tablas restantes y se unificó el formato de las mismas, Jesús realizó las conexiones entre las tablas que aún no habían sido relacionadas. Esto se debe a que, para repartir la creación de tablas, estas se dividieron en grupos, y había algunas que se relacionaban con tablas de otros grupos, por lo que no se podían relacionar hasta tener la base de datos completa.

En la cuarta fase del proyecto Jesús se ha encargado de desarrollar el DAO llamado `Comunicacion`, el cual implementa el acceso a la base de datos para el grupo de tablas que se puede ver en la Figura **??**. Para poder crear este DAO, antes ha necesitado crear los *transfer* `TUpload, TMensajes, TMail` y `TNewsletter`. Estos *transfer* se encargan, respectivamente, de almacenar los datos de los elementos *upload*, mensaje, *mail* y *newsletter* además de permitir modificarlos. Este DAO contiene las funciones CRUD, es decir, las

necesarias para crear, leer, actualizar y eliminar elementos en las respectivas tablas de la base de datos, en total unas 22 funciones.
Esto se debe a que tanto los mensajes como los *uploads* pueden pertenecer o a una colaboración o a un anuncio de servicio, por lo que cada uno tiene dos funciones distintas para su creación, una para cada caso. Además, se han hecho también funciones para obtener todos los mensajes y todos los *uploads* de una determinada colaboración o de un determinado anuncio de servicio.
Terminado el DAO en cuestión, Jesús creó los *transfers* `TColaboracion, TProyecto` y `TPartenariado`, los cuales eran necesarios para la creación del DAO `Colaboracion`. Además, en dicho DAO hizo las funciones CRUD del *transfer* `TColaboracion`. Tras esto, junto con Victoria, ambos adaptaron la funcionalidad de *login* al sistema actual.

En la quinta fase del proyecto Jesús ha creado el método que se encarga de comprobar si el periodo de disponibilidad de la demanda coincidía con el cuatrimestre o cuatrimestres especificados por el profesor en la oferta de servicio. También se cubrieron los casos en los que se consideraba que había un "*antimatching*", situaciones en las que, aunque otros atributos coincidieran, el *match* no se llevaría a cabo. Por ejemplo, si los periodos elegidos para la definición del proyecto escogidos por el profesor y por el socio comunitario no fueran compatibles, aunque todos los demás atributos lo fueran, no habría *match*.
Tras esto, y una vez Daniela y Victoria terminaron sus respectivos métodos, Jesús hizo la función "maestra" que agrupaba todos los métodos de *matching* hechos anteriormente, después de modificarlos ligeramente para que fuera más fácil hacer los cálculos de los resultados, teniendo en cuenta los respectivos pesos de los atributos. Una vez se ha calculado el porcentaje de *matching*, si este es mayor que el 50 %, se guarda el resultado en la base de datos con el fin de notificar a los usuarios pertinentes. Esta función además permite configurar los pesos de los atributos para así hacerla más versátil a la hora de priorizar qué están buscando los usuarios, de momento esto se ha hecho creando un fichero llamado `configuracion.txt` para que la función lea los valores en él descritos y los asigne.

En la sexta fase del proyecto Jesús ha implementado el formulario para la creación de una demanda de servicio. Esto, además de la creación de un fichero *crear-demanda.component.html* y un fichero *crear-demanda.component.scss* para mostrar la vista al usuario y de un fichero *crear-demanda.component.spec.ts* y un fichero *crear-demanda.component.ts* para encargarse de la lógica asociada al formulario, ha supuesto además cambios en los *controllers* y en el fichero de rutas. Además de esto, ha cambiado algunos campos en los formularios para hacerlos más legibles para el usuario.

Bibliografía

[1] David Herron. *Node.js Web Development: Server-side development with Node 10 made easy, 4th Edition.* Packt Publishing Ltd, 2018.

[2] Greg Lim. *Beginning Angular 2 with Typescript.* CreateSpace Independent Publishing Platform, 2017.