# Desarrollo de un front-end para DuoCode

**Julián F. Calleja da Silva**

**Johana Gabriela Ferreira Yagua**

**José Carlos Valera Villalba**

**Grado en Ingeniería Informática**

**Facultad de Informática**

**Departamento de Sistemas Informáticos y Computación**


**Trabajo fin de grado**

Dirigida por

**Enrique Martín Martín**

**Adrián Riesco**

Madrid, 2015

# Resumen

El proyecto "Desarrollo de un front-end para Duocode" tiene como objetivo el desarrollo de una aplicación web para el aprendizaje de lenguajes de programación. Esta aplicación permite a los usuarios aprender lenguajes de programación a partir de alguno que ya sepan mediante la superación de distintas lecciones y temas. Al empezar, solo algunas lecciones de cada tema están disponibles; el resto se van desbloqueando a medida que se van superando las anteriores.

Para conseguir una estructura clara, los temas en DuoCode consisten en una serie de lecciones. Asimismo, las lecciones se componen de una colección de ejercicios, que se basan en un enunciado en el lenguaje que el usuario conoce y que tendrá que ser traducido al lenguaje que quiere aprender.

A medida que el usuario va resolviendo los ejercicios propuestos, su puntuación va aumentando. Además, el usuario dispone de vidas, las cuales se restarán cuando la respuesta no sea correcta. Esto hace que la aplicación cuente con un tipo de aprendizaje más entretenido, haciéndolo ver como un juego. Otra herramienta de la que dispone Duocode es que permite marcar ejercicios como favorito para tenerlos accesibles y poder consultarlos en cualquier momento.

Además, como un fragmento de código en un lenguaje específico puede escribirse de distintas maneras, una parte de la aplicación está dedicada a los **candidatos**. Si un usuario falla en la resolución de un ejercicio pero cree que su solución es correcta, tiene la posibilidad de enviar su ejercicio como candidato. Al hacer esto, dicha solución pasa a ser evaluada por otros usuarios. Si obtiene los suficientes votos positivos y un usuario administrador la da por válida, pasa a ser solución correcta de ese ejercicio a partir de ese momento; por el contrario, si se vota negativamente esta solución se descartará y no podrá volver a proponerse.

Para acceder como usuario no hay que registrarse en la web ya que incluye un inicio de sesión con **Facebook** y con **Google+**. Por ello, lo único necesario para utilizar la aplicación es darle los permisos de acceso a la información básica del perfil de usuario de la correspondiente red social. Asimismo, DuoCode cuenta también con la posibilidad de compartir en Facebook el éxito tras superar una lección.

Por último, DuoCode es una herramienta útil no solo para los estudiantes, también para los docentes, pues su base de datos guarda incluso información sobre los envíos realizados por los estudiantes. De esta manera se puede hacer un seguimiento de la evolución de los usuarios y detectar aspectos problemáticos que se deban reforzar en el aula.

**Palabras clave:** Web, REST, AngularJS, base de datos, Bootstrap, aprendizaje, lenguajes de programación, traducción.

# Abstract

The project "Desarrollo de un front-end para DuoCode" aims to develop a web application for learning programming languages. This application allows users to learn programming languages from the ones they already know by overcoming different lessons and subjects. At the beginning, only a few lessons of each subject are available; the other ones will be unlocked as the above are completed.

To get a clear structure, subjects in DuoCode are a series of lessons. Also, lessons consist of a collection of exercises, which are based on a statement in the language that the user already knows and that will have to be translated into the language he wants to learn.

As the user solves the exercises, his score increases. In addition, the user has lives, which are subtracted when the answer given is not correct. This gives the application an enjoyable type of learning, making it look like a game. DuoCode allows the user to mark exercises as favorite to keep them accessible and to consult them at any time.

Furthermore, as a snippet in a specific language may be written in different ways, a part of the application is dedicated to **candidates**. If a user fails to resolve an exercise but thinks his solution is right, he has the option of sending his exercise as a candidate. Thus, the solution becomes evaluated by other users. If it gets enough positive votes and an administrator considers it valid, it becomes a right solution for this exercise; on the other hand, if it gets negative votes this solution will be discarded and cannot be proposed again.

In order to log in, the user does not have to register on the website because it includes a login with **Facebook** and **Google+**. Therefore, the only thing needed to use the application is to grant access permissions to basic user profile information of the corresponding network. Moreover, DuoCode has the ability to share on Facebook the achievements after overcoming a lesson.

Finally, DuoCode is a useful tool not only for students but also for teachers, as its database also stores information about the submissions made by the students.

**Keywords:** Web, REST, AngularJS, database, Bootstrap, learning, programming languages, translation.

# Índice

# 1. Introducción

En este proyecto hemos realizado un servicio Web REST y un *front-end* que lo usa, para crear un proyecto de aprendizaje colaborativo de lenguajes de programación llamado DuoCode. Este proyecto ha permitido a los alumnos que lo desarrollan profundizar en tecnologías y paradigmas que no se aprenden a lo largo del grado o que se dan como meras referencias; asimismo, ha permitido afianzar otros conocimientos sobre programación y planificación de proyectos. A continuación presentamos el estado del arte, las influencias tecnológicas y nuestros objetivos.

## 1.1. Estado del arte

En la actualidad hay numerosas maneras de aprender online. Para empezar, existen recursos típicos donde el que quiere aprender lee y realiza poca interacción. El líder por antonomasia de esta modalidad es Wikipedia [19], una enciclopedia libre y editada colaborativamente en múltiples idiomas. Ante dudas concretas hay sitios de preguntas y respuestas como Stack Overflow [18], donde los programadores se ayudan mutuamente en base a cuestiones concretas. Asimismo, muchas universidades usan *entornos virtuales de aprendizaje* como Moodle [15], que facilitan la gestión completa de las asignaturas. También existen cursos online, los llamados MOOC (acrónimo en inglés de *Massive Open Online Course*) donde puedes aprender distintas materias. Estos requieren que se suba material como vídeos con explicaciones detalladas. Suelen estar enfocados a la obtención de algún tipo de diploma o certificado. Ejemplos de este enfoque son Coursera [5], EdX [7] y Miriada X [13].

Otra manera de enfocar la enseñanza, sobre todo cuando quieres ampliar algo sobre lo que ya tienes una base, es la realización de ejercicios. Durante bastantes años se han usado los *jueces online* o *correctores automáticos*. Hay un análisis general sobre este tema, donde se estudia la viabilidad de un proyecto colaborativo para enseñar lenguajes de programación [20]. En él se explica como normalmente este tipo de sistemas, aplicados a lenguajes de programación, se basan en la ejecución de unos casos de prueba para comprobar la corrección de los programas del usuario. La desventaja de estos sistemas es que exige que los instructores desarrollen previamente estas pruebas, tarea poco grata. Además, solamente te indican si el ejercicio está bien o mal pero no te indica donde está el fallo.

Saliéndonos del mundo de la programación hay otros ejemplos, como Duolingo [6] que permiten aprender un idioma a partir de otro que ya sabes previamente. Sus características más relevantes son:

- Plantear el aprendizaje como un juego, haciendo que el usuario gane puntos y experiencia según va avanzando. Además incluye vidas, que hacen centrar la atención en la tarea presente para no tener que reiniciar el nivel.
- Guardar información sobre los fallos del usuario para intentar repetir esas preguntas y que aprenda los conceptos de manera definitiva.
- Incluir prácticas con tiempo y la posibilidad de certificar el nivel del idioma mediante tests online.

Hay otras alternativas como Bussu [4], donde los usuarios hacen simultáneamente de alumnos y profesores e interactúan entre ellos en una red social con el objetivo de aprender otro idioma.

## 1.2. Influencias tecnológicas

Antes de definir cómo llevar a la práctica este proyecto, quisimos ver desde alto nivel qué tipo de sistema queríamos desarrollar. Todas las asignaturas que nombraremos a continuación son las correspondientes al plan de estudios de la Universidad [16, 1].

Durante la carrera hemos aprendido, entre otras cosas, lenguajes de programación que nos permiten hacer software ejecutable en ordenadores de sobremesa, en asignaturas como Fundamentos de la Programación o Tecnologías de la Programación. Aunque éste podría ser un enfoque válido, no era el que queríamos seguir. Nos parece que podríamos hacer algo mucho más rápido de probar y usar por primera vez, sin la necesidad de instalar pesado software adicional.

En otras asignaturas como Software Corporativo aprendimos a montar y configurar un sistemas de gestión de contenidos, también llamados CMS (acrónimo en inglés de Content Management System) y con ello montamos una Web. Este enfoque presenta más ventajas, pues permite a cualquiera que tenga un navegador de Internet acceder a nuestra plataforma. Sin embargo, los CMS actuales no permiten hacer cosas tan concretas y específicas como lo que queríamos hacer. Además, dejaríamos sin usar la mayoría de las características de estos y pensamos que no usaríamos todo el potencial que ofrecen estas plataformas.

También cursamos Aplicaciones Web, donde aprendimos a desarrollar una Web desde el principio. Realizar un desarrollo de este tipo nos parecía sumamente interesante, pues nos permitiría un alto grado de flexibilidad con la manera en que queremos realizar el proyecto, y permitiría a los potenciales usuarios probar y usar la plataforma de manera rápida. Además el uso de llamadas asíncronas nos puede permitir hacer la Web plenamente interactiva y fluida ante las acciones del usuario. Por esto decidimos utilizar HTML, CSS y JavaScript en nuestro proyecto. En esta asignatura también nos apoyábamos en conocimientos adquiridos en otras como Bases de Datos y Ampliación de Base de Datos, donde aprendimos a diseñar e implementar bases de datos relacionales y a usarlas desde las aplicaciones. Su uso permite establecer interconexiones (relaciones) entre los datos (guardados en las distintas tablas), y tiene como principal ventaja evitar duplicidades de los datos, y garantizar la integridad de los datos relacionados entre si. Con todos estos conocimientos podemos hacer un sitio Web interactivo y que guarde una gran variedad de datos. Por todo esto decidimos usar una base de datos de este tipo.

También sentíamos particular inclinación por hacer que el contenido estuviera disponible en dispositivos móviles, bien sea adaptando la Web o a través de una aplicación móvil. Algunos estábamos matriculados en la asignatura Programación de Aplicaciones para Dispositivos Móviles, y vimos una oportunidad para poner de manifiesto lo que se podría aprender en esa asignatura.

Por último investigamos la utilidad de implementar un servicio Web, una tecnología que utiliza un conjunto de protocolos y estándares para intercambiar datos entre aplicaciones. Lo más parecido a este enfoque que hemos estudiado es el uso de *sockets* en Ampliación de Sistemas Operativos y Redes. Su uso permite abstraer y separar la manipulación de los

datos de la representación de la interfaz del usuario, pudiendo incluso tener varias interfaces para la misma aplicación. Por todo ello decidimos usar este enfoque. Investigamos varias maneras de implementar el servicio Web:

- La llamada a procedimiento remoto (RPC, del inglés *Remote Procedure Call*) es un protocolo que permite a un ordenador ejecutar código de manera remota en otro. Vimos que hay diversas maneras de realizar un RPC, basándose en distintas especificaciones, siendo una de las más populares SOAP.
- SOAP (acrónimo de *Simple Object Access Protocol*) es un protocolo que define cómo diferentes ordenadores pueden comunicarse a base del intercambio de mensajes XML. Sus principales desventajas son que se apoya en un descriptor de los servicios disponibles, que hay que actualizar con cada nueva funcionalidad añadida. Además, usa XML, un lenguaje de marcado que ha perdido fuelle frente al más ligero JSON. Por último, no todos los lenguajes de programación ofrecen facilidades para el uso de este tipo de servicio Web.
- REST (acrónimo de *Representational State Transfer*) es un protocolo cliente/servicor sin estado que usa HTTP y los métodos de este protocolo (GET, POST, PUT, DELETE, ...) para consultar y modificar los distintos recursos. Con frecuencia a este tipo de sistemas se les llama RESTful. Permiten mucha libertad a la hora de desarrollarlo y, como para consumir este tipo de servicio Web solo hace falta hacer peticiones HTTP, es ampliamente soportado por numerosos lenguajes de programación. Además, permite ofrecer los datos en JSON. Por todo ello decidimos usar esta tecnología para nuestro proyecto.

## 1.3. Propuestas y objetivos

Gracias a la investigación de todo lo expuesto anteriormente, fijamos nuestras ideas. Nos decidimos a hacer un proyecto para aprender lenguajes de programación, similar a Duolingo. Como estará centrado en código de programación, decidimos llamarlo DuoCode, pues aprenderás lenguajes de programación nuevos a partir de otros que ya sabes.

Las principales características que queríamos que tuviese disponibles el usuario son:

- Se podrá seleccionar el idioma de programación que sabes y el que quieres aprender. El sistema te mostrará código en el lenguaje que dominas y te pedirá rellenarlo en el que no.
- Habrá una clara organización con código agrupado por temas.
- El usuario tendrá una puntuación que irá aumentando según avance. Además, los ejercicios estarán agrupados y tendrán una serie de vidas para completarlos. Esto favorecerá que el usuario se fidelice con la aplicación a través de la gamificación.
- Cuando el usuario falle un ejercicio pero no esté de acuerdo, podrá proponerlo como solución. Con la ayuda de la comunidad y de los moderadores, estos se podrán incorporar como soluciones en un lenguaje de programación determinado.
- Se podrán guardar ejercicios favoritos para poder consultarlos de nuevo.

- Habrá una interacción con las redes sociales. El usuario se podrá autenticar utilizando alguna de ellas y compartir sus resultados en esta.

Basándonos en las conclusiones extraídas del apartado anterior, nuestro objetivos para el proyecto son:

- Desarollar un sistema Web REST para el aprendizaje de la programación. Para ello nos planteamos los siguientes pasos:
  - Especificar los distintos recursos REST y qué información recibe y produce para cada uno de los métodos HTTP (GET, POST, PUT Y DELETE).
  - Diseñar una base de datos relacional para soportar el servicio, especificando las distintas tablas y sus relaciones.
  - Implementar y probar el servicio REST.
- Desarrollar una (o dos) interfaces gráficas que usen el servicio que acabamos de describir:
  - Diseñar la interfaz usando HTML/CSS, usando datos de prueba.
  - Conectar la interfaz con el servicio Web mediante llamadas asíncronas y JavaScript, de manera que use los datos reales proporcionados por el servicio REST. Al uso de esta técnica se le suele llamar AJAJ (acrónimo en inglés de *Asynchronous JavaScript And JSON*).
  - En caso de tener tiempo diseñar la interfaz de la aplicación para Android y realizar su implementación, o adaptar la Web para su visualización en dispositivos móviles.

El resto de la memoria se estructura como sigue: en la sección 3 presentamos nuestro servicio web, mientras en la sección 4 se detalla el *front-end* que lo usa. La sección 5 describe los requisitos y la base de datos usada en el proyecto. La sección 6 presenta el manual de usuario y la sección 7 concluye y presenta algunas líneas de trabajo futuro. Por último, el apéndice A describe detalladamente los requisitos y el apéndice B la guía de instalación.

# 2. Introduction

In this project we have implemented a REST web service and a *front-end* which uses it, for creating a collaborative learning project of programming languages called DuoCode. This project has allowed the students that developed it to go in depth into technologies and paradigms that are not taught during the degree or that are merely referenced; additionally, it has allowed them to reinforce other knowledge about programming and planning projects. In the rest of the section we will introduce in the state of the art, the technological influences and our objectives.

## 2.1. State of the art

Nowadays there are numerous ways of learning online. First, there are typical resources where learners read and perform few interactions. The leader per excellence in this modality is Wikipedia [19], an open and collaborative multi-language encyclopedia. To assist in solving specific doubts there are sites for questions and answers like Stack Overflow [18], were programmers help each other to solve particular problems. Additionally, many universities use *virtual learning environments* like Moodle [15], which make easier the full management of subjects. There are also online courses, called MOOC (Massive Open Online Course) where the users can learn different subjects. These require material like videos with detailed explanations to be uploaded. Usually the purpose of these courses is to obtain some sort of diploma or certificate. Examples of this approach are Coursera [5], EdX [7], and Miriada X [13].

Another learning approach, specially when you want to expand something you already have a base of, is by solving exercises. *Online judges* or *automated assessment system* have been used for many years. There is a general analysis about the topic, where the viability of a collaborative project for teaching programming languages is studied [20]. It explains that normally this type of systems, applied to programming languages, is based on the execution of several test cases to check the correction of the user's programs. The disadvantage of these systems is that they require the trainers to develop these tests in advance, a tough task. Besides, they only point out whether there is an error or not but they do not show where the mistake is.

Besides the world of programming there are other examples, such as Duolingo [6] which allows users to learn a language based on others that they already know. Its most relevant features are:

- It makes learning similar to a game, where users earn points and experience as they progress. Furthermore it includes lives, which encourage users to focus their attention on the current task to avoid having to restart the level.
- It stores information regarding failures of the users to try to repeat these questions and force them to learn the concepts in a definitive manner.
- It includes timed practices and the possibility of certifying the level on the language through online tests.

There are other alternatives like Bussu [4], where users take simultaneously the roles of students and teachers and interact with each other in a social network with the objective of learning another language.

## 2.2. Technological influences

Before defining how to carry out this project, we wanted to plan from a high-level perspective which type of system to develop. All the subjects that are about to be named are the ones corresponding the University curriculum [16, 1].

During our degree we have learned, among other things, programming languages to write executable software for desktop computers, in subjects such as Fundamentals of Programming or Computer Programming Technology. We chose not to follow this approach, even though it could be appropriate for our project. We were certain that we could come up with a solution that would be much faster to test and use for the first time, without the need to install heavy software.

In other subjects such as Enterprise Software we learned to build and configure a CMS (Content Management System) and build a web page with it. This approach introduces more advantages, since it allows anybody with a web browser to access our platform. However, current CMSs do not allow to do such specific things as we would like. Besides, we would leave unused most of its features and we would not use them up to their full potential.

We also took web applications, where we learned to develop a web from scratch. We were interested in carrying out a development of this kind, since it would allow a high degree of flexibility, and potential users could use the platform quick and easily. Furthermore, the use of asynchronous calls can allow us to make a fully interactive and fluid web page to user interactions. For these reasons we decided to use HTML, CSS, and JavaScript in our project. In this subject we rely on knowledge gain on others such as Databases and Advanced Databases, where we learned to design and implement relational databases and use them from applications. Their use allows to establish interconnections (relations) between data (stored in different tables), and and whose main advantages are avoiding data duplicity and securing the integrity of related data. With this knowledge we can make fully interactive websites that stores a wide variety of data. For all of these facts we decided to use this kind of database.

We also had a desire for making the content mobile friendly, either by adapting the website or or by making a mobile application. Some of us were enrolled on the subject Application Programming for Mobile Devices, and we saw an opportunity for showing what we could achieve.

Lastly, we researched the utility of implementing a web service, a technology that uses a set of protocols and standards to exchange data between applications. The more similar approach that we had studied is the use of sockets in Operating Systems and Network Extension. Their use allows to abstract and split data manipulation from the graphical user interface, been able of even having multiple interfaces for the same application. For all of these we decided to follow this approach. We researched several ways of implementing a web service:

- RPC (Remote Procedure Call) a protocol that allows one computer to execute code in another one in a remote location. We saw that there a several ways of implementing RPC, using different specifications, been SOAP one of the most popular.

- SOAP (Simple Object Access Protocol) a protocol that defines how different computers can compunicate by exchanging XML messages. Their principal disadvantages are that it relies on a service descriptor, that you have to update with every new functionality. Moreover, it uses XML, a markup language that has lost popularity facing the more light JSON. Lastly, not all programming languages offer utilities for integrating this kind of Web service.
- REST (Representational State Transfer) a stateless client/server protocol that uses HTTP and its methods (GET, POST, PUT, DELETE, ...) to check and modify the different resources. Frequently these are also called RESTful. They allow a lot of freedom when developing and, as programs only need to make HTTP requests to consume these web services, it is widely supported by numerous programming languages. Furthermore, they allow to present the data in JSON format. For these reasons these we decided to use this technology for our project.

## 2.3. Objectives and proposals

Through the previous research we grounded our ideas. We decided to do a project for learning programming languages, similar to Duolingo. As it will be oriented to programming code, we decided to call it DuoCode, since you will learn new programming language from others that you already know.

The main features that we want to make available to the user are:

- Select the programming language that they already know and the one you want to learn. The system will show code in the language that the user already masters and will ask for the translation into the one the user wants to learn.
- The code will have a clear organization by subjects.
- Users will have a score that will rise as they progress. Furthermore, exercises will have a set of lives to be completed. This will favor the user's loyalty throw gamification.
- When users fail an exercise but they do not agree, they will be able to propose it as a solution. With the help of the community and moderators, the system will be able to add solutions in a specific programming language.
- Exercises can be stored as favorites to check them again.
- There will be interaction with social networks. The user will be able to authenticate using some of them and share the results on them.

Based on the conclusions extracted from the previous section, our objectives for the project are:

- To develop a REST Web service for learning programming languages. We intend to follow these steps:
    - Specify the different REST resources and what information they receive and produce for each of the HTTP methods (GET, POST, PUT, and DELETE).

    - Design a relational database to support the service, specifying the different tables and their relations.
    - Implement and test the REST service.

- Develop at least one graphic interface using the service described above:

    - Design the interface using HTML/CSS, using test data.
    - Connect the interface with the web service through asynchronous calls and JavaScript, using the real data provided by the REST service. The use of this technique is often called AJAJ (Asynchronous JavaScript And JSON)
    - Develop an application for Android, or depending on the time constrains adapt the website for its visualization on mobile devices.

The rest of the memory is structured as follows: in Section 3 we introduce our web service, while in Section 4 the front-end that uses it is detailed. Section 5 presents the requirements and the database used on the project. Section 6 introduces the user's manual and Section 7 concludes and presents some lines of the future work. Lastly, Appendix A describes in detail the requirements and the appendix B the installation guide.

# 3. Servicio web

Con el fin de que nuestro proyecto no consistiese en una única aplicación web y se pudiesen desarrollar tanto una aplicación móvil como una aplicación de escritorio en el futuro, necesitábamos implementar un sistema *interoperable* de acceso al contenido de nuestra base de datos.

Un **servicio web** era nuestra solución, así que buscamos información sobre los estándares más empleados y nos centramos en tres: *RPC*, *SOAP* y *RESTful*.

## 3.1. Arquitectura del servicio web

Cada integrante del grupo investigó a fondo un estándar en concreto y tras una puesta en común nos decantamos por un servicio web tipo *RESTful* porque es ampliamente usado en la actualidad, es un protocolo ligero y cuenta con los siguientes puntos fuertes:

- **Sin estado**:

  Cada petición por parte del cliente ha de contener toda la información necesaria para poder procesarla, sin depender de contenidos almacenados en el servidor. Por esto todo el proceso de autentificación y el estado de la sesión se lleva a cabo en el cliente.

- **Cliente-Servidor**:

  *REST* es una arquitectura *cliente-servidor* que nos permite diferenciar dos componentes: la lógica de negocio (servidor) y la lógica de presentación(cliente).

  Gracias a esta arquitectura podemos desarrollar cada componente de manera independiente, siempre y cuando se mantengan coherentes con la interfaz. También se simplifican las tareas de mantenimiento ya que tanto la ampliación como la corrección del sistema se puede hacer por partes.

- **Protocolo HTTP**:

  El servidor proporciona al cliente una interfaz uniforme basada en *URLs* para acceder a los recursos de nuestro sistema. El intercambio de información se realiza mediante peticiones *HTTP*.

  Cada verbo *HTTP* (Get, Post, Put y Delete) tiene su equivalente operación *CRUD* (Create, Read, Update y Delete) y método *SQL* (Insert, Select, Update y Delete). Se puede apreciar el mapeo en la siguiente tabla:

| **Peticiones HTTP** | **Operaciones CRUD** | **Método SQL** |
|---|---|---|
| *POST* | *CREATE* | *INSERT* |
| *GET* | *READ* | *SELECT* |
| *PUT* | *UPDATE* | *UPDATE* |
| *DELETE* | *DELETE* | *DELETE* |

## 3.2. Recursos

Uno de los conceptos más importantes de una API *RESTful* es la existencia de *recursos*. Un recurso es un objeto de un tipo determinado, con datos asociados y que cuenta con un conjunto de métodos que operan sobre él (Get, Post, Put y Delete en el caso de nuestro proyecto).

Una vez que tuvimos clara la arquitectura de nuestro servicio web, definimos todos los *recursos* que eran necesarios para desarrollar *DuoCode*. Aunque en el apéndice '*Especificación de requisitos*' donde se habla detalladamente sobre cada recurso, se pueden ver resumidos en la siguiente lista.

- **Temas**: Recurso encargado de gestionar los conceptos más comunes de programación a los que el usuario se enfrenta (instrucciones de control, funciones, algoritmos iterativos, algoritmos recursivos, clases...). Cada tema consiste en una agrupación de lecciones con distintos niveles de dificultad.
- **Lecciones**: Recurso encargado de gestionar las distintas lecciones que conforman un tema. Una lección consta de un grupo de ejercicios con un nivel de dificultad similar (dentro del tema 'clases' encontraríamos las lecciones 'sintaxis', 'herencia', 'polimorfismo',...).
- **Ejercicios**: Recurso encargado de gestionar los distintos ejercicios que conforman una lección. Un ejercicio consta de una breve descripción y de una lista de enunciados que lo resuelven.
- **Enunciados**: Recurso encargado de gestionar los distintos enunciados que implementan un ejercicio. Un enunciado contiene un fragmento de código en un lenguaje de programación concreto que corresponde a la solución de un ejercicio.
- **Lenguajes**: Recurso encargado de gestionar todos los lenguajes con los que se puede trabajar en DuoCode.
- **Candidatos**: Recurso encargado de gestionar enunciados propuestos por el usuario que no han sido calificados todavía como correctos o incorrectos. Los candidatos también pueden ser moderados por los usuarios.
- **Usuarios**: Recurso encargado de gestionar los usuarios registrados en la aplicación, la información personal, las lecciones que ha completado, los puntos que tiene, los ejercicios favoritos y los envíos realizados.
- **Envios**: Recurso encargado de gestionar los envíos de soluciones por los usuarios de cada ejercicio que realizan. La información de cada envío resulta útil para realizar estadísticas y encontrar fallos reiterativos a la hora de aprender un lenguaje de programación nuevo.

Con todos los recursos definidos pudimos diseñar la base de datos y comenzamos a pensar qué tecnologías íbamos a usar para la implementación del servicio web.

## 3.3. Implementación

Actualmente hay una gran diversidad de frameworks para implementar una API RESTful usando cualquier tecnología. Los lenguajes con los que más cómodos nos sentíamos programando eran PHP y Java así que centramos la investigación a estos dos.

Después de analizar ambas propuestas y ver el código de ejemplos sencillos nos dimos cuenta de que la curva de aprendizaje iba a ser menos pronunciada si lo hacíamos con Java.

### 3.3.1. Jersey

La API para servicios RESTful en Java es la *JAX-RS*, definida en el *JSR 311*, y el framework que usamos es *Jersey*, ya que es su implementación más estándar y existe mucha documentación disponible.

Cada uno de los recursos definidos anteriormente se implementaron en Java siguiendo la siguiente sintaxis, para la cual se usará de ejemplo la clase **EjerciciosResource**:

- *@Path('ejercicios')*: Especifica la ruta de acceso relativa para una clase, un recurso o un método. En nuestro ejemplo indica la ruta relativa del recurso 'Ejercicios' a la que hacer las peticiones HTTP.

```
@Path('ejercicios')

public class EjerciciosResource {
        .
        .
        .
}
```

- *@GET*: Método que se ejecuta cuando hay una petición HTTP GET sobre el recurso al que pertenece.
- *@Produces(MediaType.APPLICATION_JSON)*: Especifica el tipo de MIME que el recurso produce y envía al cliente. En nuestro caso el tipo que produce nuestro método es *JSON*.

```
    @GET
    @Produces(MediaType.APPLICATION_JSON)
    public Ejercicios getEjercicios() {
        return new Ejercicios(this.ejercicioMapper.findAll());
    }
```

- *@POST*: Método que se ejecuta cuando hay una petición HTTP POST sobre el recurso al que pertenece.

- *@Consumes(MediaType.APPLICATION_JSON)*: Especifica los tipos de medios de petición aceptados. En nuestro caso el tipo que acepta nuestro método es *JSON*.
- *@HeaderParam('token')*: Enlaza el parámetro a un valor de cabecera HTTP. En nuestro caso el parámetro `'token'` del método newExercise tomaría el valor del parámetro `'tk'`, perteneciente a la cabecera de la petición HTTP.

```
@POST
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public ErrorYID newExercise(@HeaderParam('tk') String token){
   .
   .
   .
}
```

- *@Path('{id}')*: En este caso @Path es la ruta relativa de un método. Se concatena con el Path definido en la clase y el resultado sería `/ejercicios/{id}`
- *@PathParam('id')*: Enlaza el parámetro `'idParam'` del método a un segmento de ruta. En este caso `'idParam'` tomaría el valor de `'id'` que aparece en la ruta a la que se realiza la petición.

```
@GET
@Path("{id}")
@Produces(MediaType.APPLICATION_JSON)
public Ejercicio getEjercicio(@PathParam("id") int idParam) {...}
```

- *@DELETE*: Método que se ejecuta cuando hay una petición HTTP DELETE sobre el recurso al que pertenece.

```
@DELETE
@Path("{idEjercicio}")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public ErrorSimple deleteEjercicio(...){...}
```

- *@PUT*: Método que se ejecuta cuando hay una petición HTTP PUT sobre el recurso al que pertenece.

```
@PUT
@Path("{idEjercicio}")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public ErrorSimple putEjercicio(...){...}
```

### 3.3.2. Advanced Rest Client

Para ir testeando la API y comprobar el correcto funcionamiento de los métodos usamos **Advance Rest Client**, un complemento de Google Chrome que permite hacer peticiones HTTP a una URL y muestra los datos que devuelve la petición, así como si ha fallado o ha saltado alguna excepción. Advance Rest Client se puede buscar y descargar en la siguiente dirección: `https://chrome.google.com/webstore/category/apps`.

### 3.3.3. Acceso a la base de datos

El acceso a la base de datos era otro de los puntos delicados a la hora de desarrollar nuestro servicio web. No queríamos tener código SQL repartido por todo el servicio así que decidimos hacer uso de patrones clásicos de acceso a datos.

Durante el curso pasado en la asignatura de 'Ampliación de bases de datos' nos explicaron el patrón *Data Mapper* y pensamos que podría ser útil en nuestro proyecto. El patrón *Data Mapper* consiste en construir una capa de componentes (*Mappers*) que traducen los objetos de la aplicación a la representación de la base de datos y viceversa. De esta manera se desacopla el código de acceso a datos de los objetos de dominio. La introducción de distintos tipos de objetos de dominio provoca la introducción de sus correspondientes *Mappers*, y esto provoca que exista mucho código duplicado (los métodos insert, update, delete, findByID tienen un código muy similar entre dos objetos de dominio). La solución es abstraer todo el código común de los *Data Mappers* a una clase **'AbstractMapper'** para facilitar el mantenimiento de la aplicación y reducir el código duplicado.

La clase *AbstractMapper* es una clase abstracta que implementa las operaciones *SQL* más comunes y de la que heredan todos los *Mappers* de cada objeto de dominio. Se puede ver un ejemplo de la implementación de un 'findById' a continuación.

```
public T findById(int id) {
        Connection con = null;
        PreparedStatement pst = null;
        ResultSet rs = null;
        T result = null;
        try {
                con = ds.getConnection();
                pst = con.prepareStatement(getFindIdSQL());
                pst.setInt(1, id);
                rs = pst.executeQuery();
                if (rs.next()) {
                        result = buildObject(rs);
                }
        } catch (SQLException e) {
                ...
        } finally {
                ...
        }
        return result;
}
```

Todos los *Mappers* de nuestro proyecto están almacenados en la carpeta 'src/java/mappers' y se pueden ver online en la siguiente dirección: `https://github.com/jucallej/DuoCode/tree/master/src/java/mappers`.

La principal ventaja del uso del patrón *Data Mapper* es que el código SQL se queda centrado en los distintos mappers creados, facilitando el mantenimiento y la flexibilidad del sistema.

# 4. Front-end

En esta sección describiremos cómo realizamos la interfaz gráfica que usa el servicio Web REST creado previamente. En primer lugar realizamos un prototipo con HTML/CSS, luego conectamos ese prototipo con los datos reales, y por último integramos varias redes sociales en la aplicación.

## 4.1. Prototipo

Antes de conectar la interfaz con los datos del servicio Web decidimos hacer un prototipo con datos de prueba. Lo hicimos usando HTML, un lenguaje de marcado para la elaboración de páginas Web, y hojas de estilo CSS, que sirven para definir la presentación de un documento HTML. Hacer interfaces visuales coherentes y vistosas puede ser complejo y laborioso si se decide partir desde cero. En la actualidad se suele usar *frameworks* como Bootstrap [3] para tener una base sólida. Sus principales características son:

- Permite realizar interfaces con diseño adaptable, también llamadas *responsive*, que permiten ajustar la visualización de la Web para distintos dispositivos como móviles, tabletas y ordenadores. Esto también se puede realizar con la última versión de CSS, pero es más complejo y su uso es menos directo. Boostrap dispone de 12 columnas, que puedes distribuir entre el contenido de diferente manera según las características del dispositivo.
- Dispone de una serie de clases CSS que permiten hacer rápidamente elementos HTML vistosos como botones, tablas, etc.
- También tiene una serie de componentes reutilizables creados con CSS como iconos, barras de menús, cuadros de alerta, insignias, barras de progreso, etc.

Por todo esto decidimos usar este *framework* en nuestro proyecto.

Como los usuarios principales de la aplicación serán desarrolladores, queríamos que se sintieran cómodos con el diseño. Además queríamos que tuviera personalidad y fuera reconocible. Por ello decidimos usar la paleta de colores *Monokai* [14], disponible en la mayoría de editores de texto orientados a código.

A la hora de mostrar el código que el usuario debe traducir, queríamos que se resaltara la sintaxis del lenguaje para facilitar al usuario su lectura. Para ello usamos `highlight.js` [11], un biblioteca JavaScript que hace precisamente esto. Para ello hay que poner el código de la siguiente manera:

```
<pre><code> código a resaltar <pre><code>
```

Opcionalmente se le puede especificar en qué lenguaje está escrito el código, pero bajo nuestras pruebas resulta suficiente con la detección automática que tiene la biblioteca. Se puede especificar la paleta que colores que quieres que use para resaltar el código, y para mantener la coherencia usamos de nuevo *Monokai*.

Teniendo todo esto en cuenta realizamos el prototipo usando diversos documentos HTML, y un único archivo CSS con todo lo necesario para dar estilo a estos. Nos aseguramos de que todo el diseño fuera correctamente visible en varios tipos de dispositivos (nos centramos principalmente en móviles y ordenadores). Desarrollamos varios comportamientos personalizados, como una barra lateral con la información del usuario que se queda fija en ordenadores y dispositivos con alta resolución (no hace *scroll* con el resto del contenido), pero que mueve y se coloca al final cuando estás en un dispositivo móvil. Todos los archivos CSS están dentro de la carpeta 'css', al igual que todos los archivos JavaScript están dentro de la carpeta 'js'. El resultado se puede ver reflejado en el producto final, pues el diseño lo conservamos intacto.

## 4.2. Conexión con el servicio REST

Para conectar la interfaz visual con el servicio Web realizamos llamadas asíncronas, como hemos comentado en la introducción. Nos planteamos hacer una Web con una única página, que se va modificando con estas peticiones. Con JavaScript, el lenguaje de programación del lado del cliente para interactuar con el HTML y CSS, hay varias maneras de realizar estas llamadas asíncronas, y hacer que la interfaz reaccione correctamente a las acciones del usuario. Por ello nos planteamos cómo realizarlo:

- Se pueden hacer de manera nativa con JavaScript, sin ninguna biblioteca externa, pero hay que tener en cuenta que la manera de realizarlo varía según el navegador, por lo que realizar una petición de esta manera que soporte los navegadores más usados genera un código largo y tedioso. Realizar así muchas llamadas puede convertirse en algo complejo y poco mantenible. Además una vez recibidos los datos, modificar el HTML (a través de modificaciones del DOM, o 'Modelo de Objetos del Documento') con JavaScript se convierte en una tarea compleja y poco abarcable para un proyecto de relativo tamaño.

- Otra manera de realizar esto es usando una biblioteca como jQuery [12], que simplifica mucho la realización de peticiones asíncronas y la modificación del documento, además de incluir muchas funciones útiles. Sin embargo la modificación intensiva del documento HTML, sigue siendo algo que se puede complicar bastante en cuanto crece el tamaño. Por ello tampoco nos pareció oportuno usarlo para nuestro proyecto.

- Hay *frameworks* de JavaScript, que simplifican el desarrollo de aplicaciones Web mediante el uso del patróns modelo-vista-controlador en el lado del cliente. Uno de los más probados, con una amplia documentación y un gran respaldo de la comunidad es AngularJS [2]. Permite adjuntarle a un elemento HTML un controlador JavaScript. Además permite hacer que la vista represente los valores de los atributos del controlador asociado y, si estos cambian, la vista cambiará automáticamente. Ayuda a separar el código JavaScript y permite extenderlo a HTML para crear componentes reutilizables. Por último, permite consumir servicios Web REST de manera sencilla.

Tras ver las características de AngularJS y comprobar que se adaptaría muy bien a nuestro proyecto, decimos usarlo. Usamos varias características avanzadas de este *framework* como a continuación explicaremos en detalle. Para empezar, permite hacer que una parte del HTML cambie dinámicamente según la URL donde estemos. Esto permite que todos las vistas compartan ciertos elementos como los menús superiores, logotipo, fondos etc. y dependiendo de la ruta URL actual que se muestre una vista distinta (ej. selección de lenguajes, selección de lección, valorar candidatos, etc.). Permite a los usuarios guardar un link en los favoritos del navegador o compartirlo, y que este guarde la vista exacta donde está el usuario. Además cambiar de vista se simplifica significativamente, pues solamente hay que cambiar de URL (usando *links* normales de HTML, por ejemplo). Esto es un comportamiento que usan las páginas en las que el servidor te devuelve distintos HTML, pero que páginas basadas en peticiones asíncronas para todo no suelen tener. Esto también nos permite separar el HTML por las distintas vistas. En nuestro caso dentro de la carpeta 'parts' tenemos los distintos fragmentos de HTML que son específicos de cada vista. Cada uno de ellos tiene asociado un controlador diferente. La manera de indicar dónde se va a insertar el HTML extra dependiendo de la URL es usando el atributo *ng-view* de la siguiente manera:

```
<div ng-view></div>
```

Configuramos los fragmentos de HTML que le corresponde a cada URL, en un archivo JavaScript que llamamos `'duocode.js'`, así:

```
duocodeApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/', {
        templateUrl: 'parts/lenguajes.html',
      }).
      when('/lenguajes', {
        templateUrl: 'parts/lenguajes.html',
      }).

      [...]

}]);
```

Asociamos los distintos controladores, poniéndolos en los fragmentos de HTML:

```
ng-controller="LeccionesController"
```

Y creando dicho controlador en el archivo arriba mencionado `'duocode.js'`:

```
duocodeApp.controller('LeccionesController', ['$scope', '$http', [...],
    function ($scope, $http, [...] ) {

        [...]

}]);
```

En nuestra aplicación tenemos varias vistas que muestran la información del usuario (su nombre, foto, puntos conseguidos, etc.), pero no todas lo hacen. Una manera poco eficiente de hacer esto sería duplicar el código para aquellas que sí muestran dicha información. Sin embargo la manera más efectiva y adecuada sería crear un componente reutilizable, de manera que lo usen los que quieran mostrar esta información. AngularJS permite este último enfoque, haciendo que un fragmento de HTML pueda ser mostrado por varias vistas. Lo hace permitiéndonos extender los elementos HTML disponibles. Para ello debemos indicar en el archivo `'duocode.js'` qué fragmento es reutilizable:

```
duocodeApp.directive('infoUsuario', function() {
  return {
    restrict: 'E',
    templateUrl: 'parts/info-usuario.html'
  };
});
```

Una vez hecho esto podemos usar en las vistas que correspondan el siguiente elemento HTML que acabamos de crear:

```
<info-usuario></info-usuario>
```

La última característica que creemos importante explicar es cómo hemos conseguido que los distintos controladores compartan datos entre ellos. Para esto usamos lo que AngularJS llama 'Providers' [17], que permiten crear servicios y objetos especiales que puedes añadir a tus controladores. Estos solo se cargan la primera vez; las siguientes se devuelven los mismos datos. De esta manera varios controladores pueden tener acceso por ejemplo al mismo usuario y no tener que hacer varias peticiones al servicio Web REST por los mismos datos. Lo pusimos en un archivo separado llamado `'providers.js'`.

```
duocodeProviders.factory('usuarioServicio', ['$http', [...],
  function ($http, [...]) {

  [...]

  return $http.get(rutaApp+'usuarios/' + usuarioData.idUsuario);
}]);
```

Con todo esto conseguimos hacer que la interfaz usara los datos reales que le proporcionaba el servicio Web REST. En este punto usábamos un usuario predefinido, y suponíamos que el usuario estaba siempre *logueado*.

## 4.3. Integración con redes sociales

Una vez conectados con los datos reales, nos dispusimos a impedir que los distintos usuarios pudieran ver cierto contenido sin estar autenticados y permitirles hacerlo usando varias redes sociales. En primer lugar queríamos que fuera posible usar nuestra aplicación con una cuenta de Google. Todos los profesores y alumnos tienen una dirección de correo del tipo <usuario>@ucm.es, y esta está gestionada por Google, de manera que todos ellos pueden acceder al servicio. En segundo lugar queríamos que pudiera ser accesible usando

una cuenta de Facebook [8], pues es una de las redes sociales más populares en España y en el mundo.

Cada red social pone a disposición de los desarrolladores bibliotecas JavaScript que permiten obtener un campo único para identificar al usuario, y un *token* que permite acceder a ciertos datos del usuario y autenticarlo durante un periodo limitado de tiempo. El problema es que estás bibliotecas no funcionan más que con su propio servicio. También se puede obtener este campo para identificar al usuario y el *token* haciendo una serie de peticiones a sus servidores e intercambiando algunos datos (es el servidor de la red social el que pide autorización al usuario). Este enfoque tiene el mismo problema, pues es distinto para cada red social. Para solucionar esto hay bibliotecas como la que usamos, HelloJS [10], que permiten obtener los datos para autenticar a un usuario de una manera sencilla, y que funcionan para multitud de redes sociales. Integramos esto con la aplicación que ya habíamos creado con AngularJS, e hicimos que si estás autenticado siempre se mande el código que identifica al usuario y el *token* por la cabecera de cada petición al servicio Web REST. Es este el encargado de verificar que esto es valido, y obtener ciertos datos básicos del usuario, como su nombre, o un enlace a su foto de perfil.

Además, para que el usuario se implicara más, queríamos que pudiera compartir sus avances. Consideramos que Google+ tiene poco uso, por eso solo permitimos compartir cuando estás autenticado con Facebook. Para ello usamos la biblioteca que pone a disposición de los desarrolladores para tales fines [9].

Para último, recordar que para usar estos servicios de las distintas redes sociales es necesario registrarse como desarrollador en los distintos portales de las compañías.

# 5. Requisitos y base de datos

En esta sección hablaremos del método que utilizamos para abarcar la fase de desarrollo de los requisitos y de la creación de la base de datos.

## 5.1. Requisitos

Esta es la primera fase que abarcamos al empezar con el proyecto. Después de discutir las ideas sobre cómo queríamos que fuese el servicio web, empezamos a redactar los requisitos.

La redacción de los requisitos nos ha facilitado mucho la tarea de la implementación ya que así no teníamos que improvisar mientras escribíamos código y cuando lo necesitábamos recurríamos a ellos. Hemos hecho pequeñas modificaciones a medida que nos encontrábamos con algo distinto a lo que nos habíamos imaginado o que no habíamos tenido en cuenta antes de empezar a implementar.

Para hacerlos, los organizamos según los recursos que utilizaríamos (por ejemplo: temas, lecciones, ejercicios, usuarios...) y cada uno de ellos tiene un máximo de cuatro partes (GET, POST, PUT y DELETE) debido a que utilizamos una API REST y trabajamos con estos métodos de petición. Indicamos en todos qué se tiene que recibir por Payload y por Header (opcionalmente) y cuál será la respuesta que recibiremos.

La especificación de los requisitos se encuentra en el Apéndice A

## 5.2. Base de datos

A la vez que íbamos definiendo los requisitos, fuimos dándonos cuenta de las entidades y atributos que necesitaríamos en la base de datos y formamos el modelo entidad-relación (figura 1). Una vez conseguido esto, definimos las tablas y elaboramos el modelo relacional (figura 2).

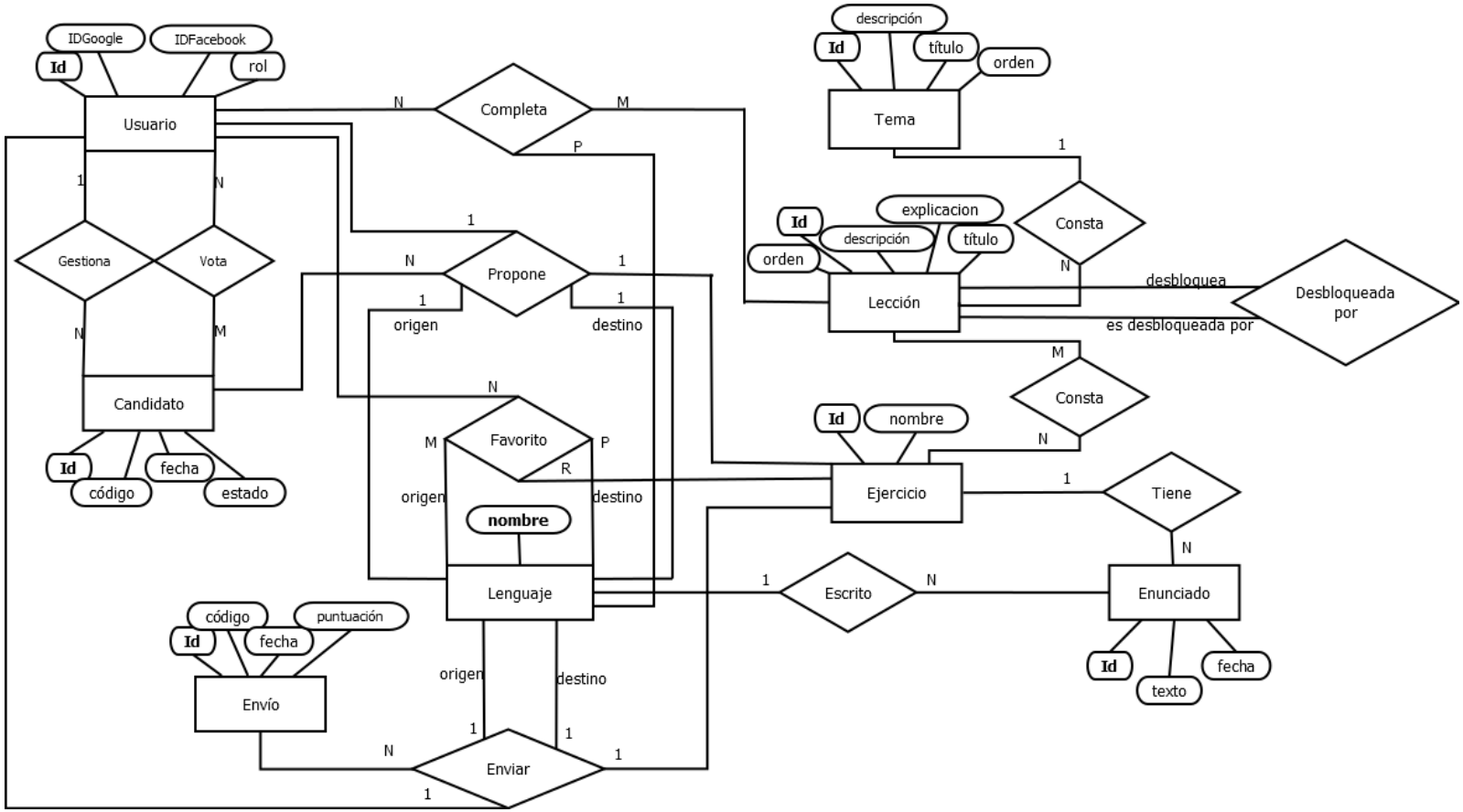



Figura 1: Modelo entidad-relacion

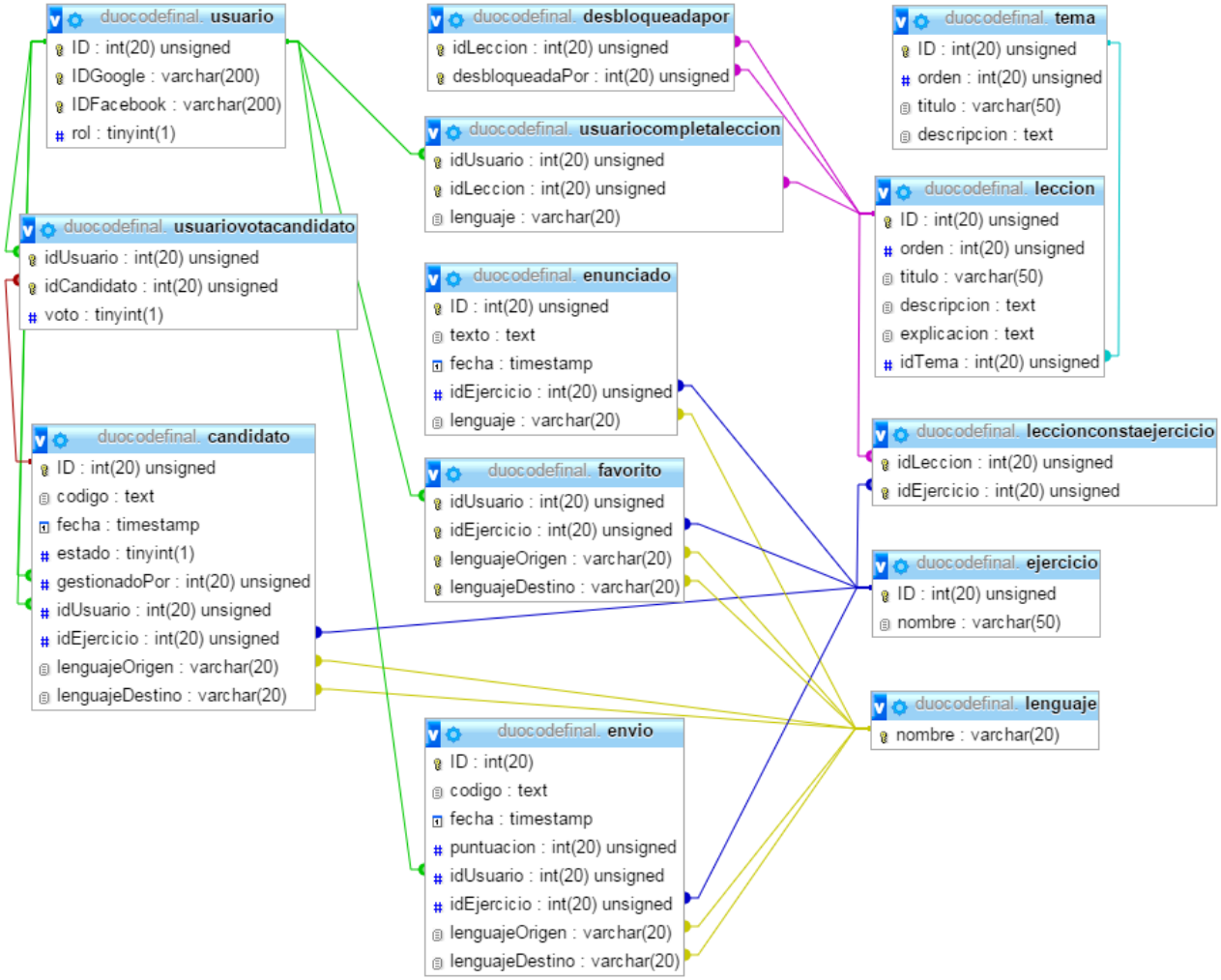



Figura 2: Modelo relacional

### Estructura de las tablas

**Usuario:** Tabla que contiene los datos de los usuarios. No hace falta guardar el nombre u otros datos personales ya que los obtenemos de la aplicación con la que haya iniciado sesión.

| Nombre | Descripción |
|---|---|
| ID | ID que le asigna la aplicación al usuario. |
| IDGoogle | ID que le asigna Google. Es opcional ya que el usuario puede iniciar con Facebook. |
| IDFacebook | ID que le asigna Facebook. Es opcional ya que el usuario puede iniciar con Google. |
| rol | Indica si el usuario es administrador (1) o no (0). |

**UsuarioVotaCandidato:** Tabla que contiene todos los votos de los usuarios a los candidatos.

| Nombre | Descripción |
|---|---|
| idUsuario | ID del usuario que vota. |
| idCandidato | ID del candidato al que se quiere votar. |
| voto | Puede tomar los valores 0 o 1 dependiendo de si el voto es positivo (1) o negativo (0). |

**Candidato:** Tabla que contiene la información de los candidatos enviados por los usuarios de la aplicación.

| Nombre | Descripción |
|---|---|
| ID | Identificador único para el candidato. |
| código | Texto propuesto como posible solución del ejercicio. |
| fecha | Fecha en la que se realiza la propuesta. |
| estado | Puede ser No Gestionado (0), Aceptado (1) o Rechazado (-1). |
| gestionadoPor | ID del usuario administrador que finalmente acepta o rechaza el candidato. Inicialmente se encuentra vacío. |
| idUsuario | ID del usuario que propone el candidato. |
| idEjercicio | ID del ejercicio que resuelve dicho candidato. |
| lenguajeOrigen | Lenguaje en el que se encuentra el enunciado del ejercicio. |
| lenguajeDestino | Lenguaje en el que está la posible solución. |

**Tema:** Tabla que contiene la información de los temas, primer nivel para organizar los ejercicios según a qué van dedicados. Los temas se conforman de una colección de lecciones.

| Nombre | Descripción |
|---|---|
| ID | ID propio del tema |
| orden | Orden en el que queremos que aparezca el tema. |
| título | Nombre que distingue a los temas. |
| descripción | Texto breve que describe el tema. |

**Lección:** Tabla que contiene la información de las lecciones, segundo nivel para la organización. Las lecciones se conforman de una serie de ejercicios y están incluidas en temas.

| Nombre | Descripción |
|---|---|
| ID | Identificador único para la lección. |
| orden | Orden en el que queremos que aparezca la lección. |
| título | Nombre que distingue a las lecciones. |
| descripción | Texto breve que describe la lección. |
| expliciación | Texto largo que sirve como introducción a la lección. |
| idTema | ID del tema al que pertenece esta lección. |

**UsuarioCompletaLeccion:** Tabla que contiene la relación de las lecciones con los usuarios que las han completado.

| Nombre | Descripción |
|---|---|
| idUsuario | ID del usuario que ha completado la lección. |
| idLección | Lección que ha sido completada. |
| lenguaje | Lenguaje en el que se ha completado dicha lección, ya que una lección puede ser completada por el mismo usuario en distintos lenguajes. |

**DesbloqueadaPor:** Tabla que contiene la dependencia entre lecciones. Hay lecciones a las que solo se puede acceder si se han superado otras anteriormente.

| Nombre | Descripción |
|---|---|
| idLección | ID de la lección a la que queremos asignar dependencia. |
| desbloqueadaPor | Lección que debe ser superada para poder acceder a la mencionada anteriormente. |

**Ejercicio:** Tabla que contiene la información de los ejercicios existentes.

| Nombre | Descripción |
|---|---|
| ID | Identificador asignado por la aplicación para cada ejercicio. |
| nombre | Nombre con el que se diferenciará de los demás. |

**LeccionConstaEjercicio:** Tabla que contiene la relación donde se asignan los ejercicios a las correspondientes lecciones. Un mismo ejercicio puede estar en varias lecciones a la vez.

| Nombre | Descripción |
|---|---|
| idLección | ID de la lección a la que se añade un ejercicio. |
| idEjercicio | ID del ejercicio añadido a la lección. |

**Lenguaje:** Tabla que contiene los lenguajes que podrán aprender los usuarios.

| Nombre | Descripción |
|---|---|
| nombre | Nombre del lenguaje. |

**Enunciado:** Tabla que contiene la información de los enunciados. Un enunciado es un código con un correspondiente lenguaje que forma parte de un ejercicio. Un ejercicio puede tener varios enunciados en varios idiomas.

| Nombre | Descripción |
|---|---|
| ID | ID que identifica al enunciado. |
| texto | Código, en lenguaje de origen, que tendrá que ser traducido. |
| fecha | Fecha en la que fue añadido el enunciado. |
| idEjercicio | ID del ejercicio al que corresponde este enunciado. |
| lenguaje | Lenguaje en el que está escrito el código de dicho enunciado. |

**Favorito:** Tabla que contiene la relación entre los usuarios y los ejercicios favoritos de estos.

| **Nombre** | **Descripción** |
|---|---|
| idUsuario | ID del usuario que añade un ejercicio a favorito. |
| idEjercicio | ID del ejercicio que ha sido marcado como favorito. |
| lenguajeOrigen | Lenguaje del enunciado que ha sido marcado como favorito. |
| lenguajeDestino | Lenguaje que el usuario quería aprender al marcar este ejercicio como favorito. |

**Envío:** Tabla que contiene la relación entre los usuarios y todos los ejercicios que han realizado en la aplicación.

| **Nombre** | **Descripción** |
|---|---|
| ID | ID correspondiente a cada envío. |
| código | Código respuesta del usuario. |
| fecha | Fecha en la que se ha resuelto el ejercicio. |
| puntuación | Puntuación obtenida. |
| idUsuario | Usuario que ha realizado el envío. |
| idEjercicio | Ejercicio resuelto. |
| lenguajeOrigen | Lenguaje inicial del enunciado. |
| lenguajeDestino | Lenguaje en el que el usuario ha enviado el ejercicio. |

# 6. Manual de usuario

En esta sección se muestra el funcionamiento de la aplicación web desde el punto de vista de un usuario.

## 6.1. Lenguajes

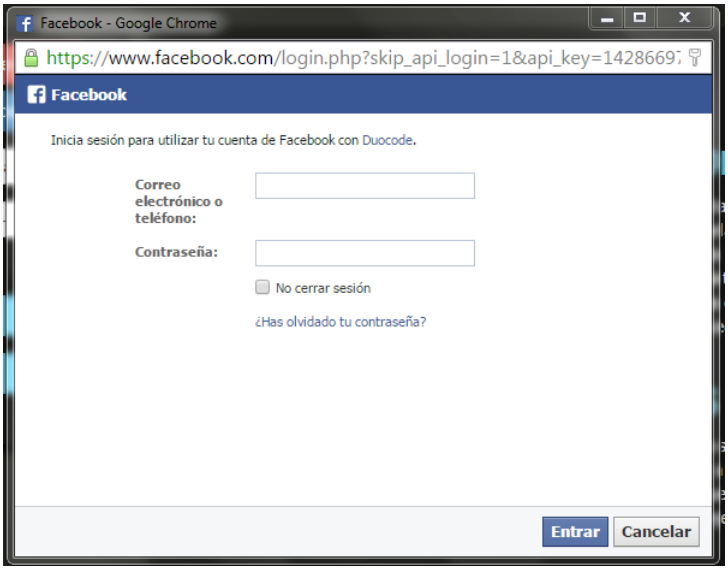
Al acceder al inicio de la web se nos muestran un par de listas con lenguajes de programación, donde podremos elegir cuál sabemos y cuál queremos aprender, en ningún caso será posible elegir la misma opción en ambos lados. En la parte superior se encuentra la barra de navegación que estará accesible en todo momento.


Una vez pulsado el botón de "¡Comenzar!", pasamos a la página de los temas.

## 6.2. Temas

En esta sección nos encontramos con dos partes diferenciadas. Por una parte, tenemos el login/información de usuario y por otra todos los temas disponibles con una breve explicación sobre estos.


Tenemos la posibilidad de iniciar sesión tanto con Google como con Facebook.




En ambos casos se pedirá permiso al usuario para acceder a su información de perfil. Después de iniciar sesión, en la información de usuario aparecerá el nombre y la imagen de su perfil de red social, su puntuación total en DuoCode, los lenguajes seleccionados anteriormente, su número de ejercicios favoritos y candidatos enviados en la aplicación.


Una vez seleccionado un tema, pasamos a las lecciones.

## 6.3. Lecciones

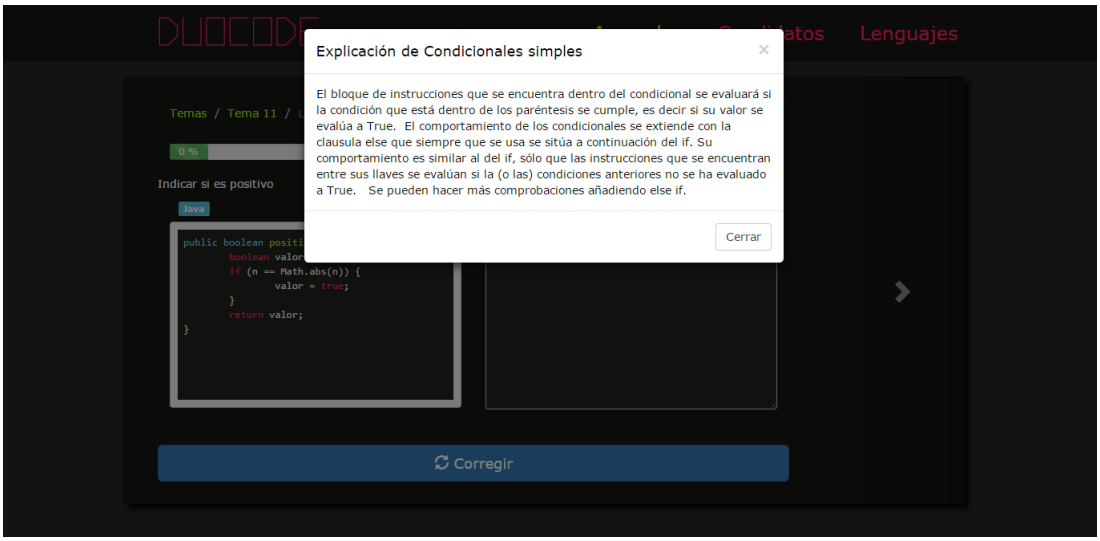
Al seleccionar un tema accedemos a sus correspondientes lecciones, de las cuales podemos ver un pequeño resumen. Si el nombre de la lección aparece en blanco significa

que esta es dependiente de otra, es decir, tienes que superar antes otra lección para poder acceder a esta.


Una vez seleccionada una lección, pasamos a los ejercicios.

## 6.4. Ejercicios

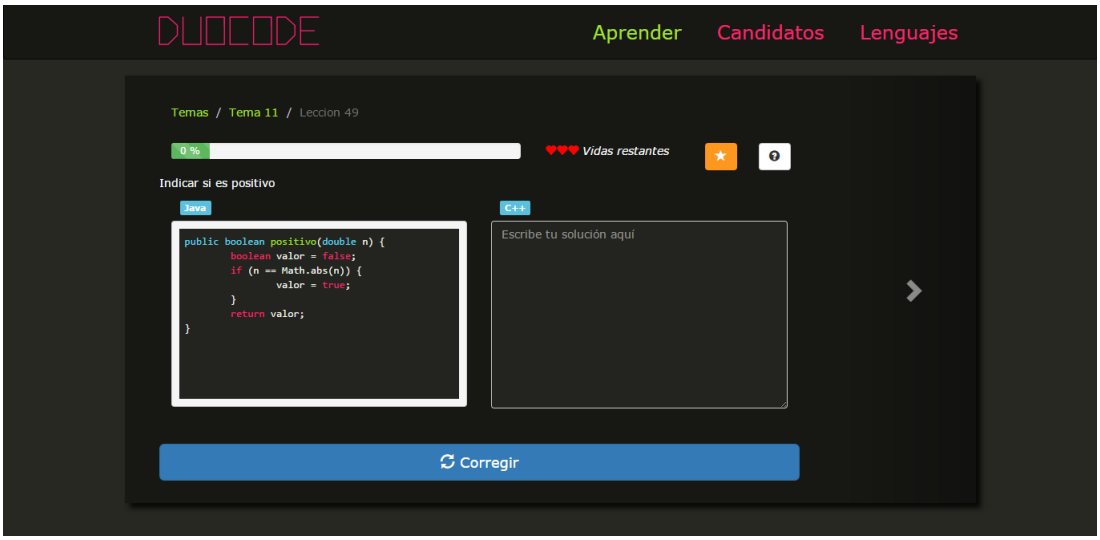
En esta parte lo primero que nos encontramos es un *pop-up* con una explicación de la lección, al que se podrá acceder nuevamente a lo largo de esta.




Al cerrar el *pop-up* empezamos con los ejercicios. En todo momento encontramos varios elementos en la parte superior: una barra en la que se indica el porcentaje que se ha completado de la lección, las vidas restantes, un botón para añadir a favorito el ejercicio actual y un botón de ayuda con el que se muestra el *pop-up* del principio.

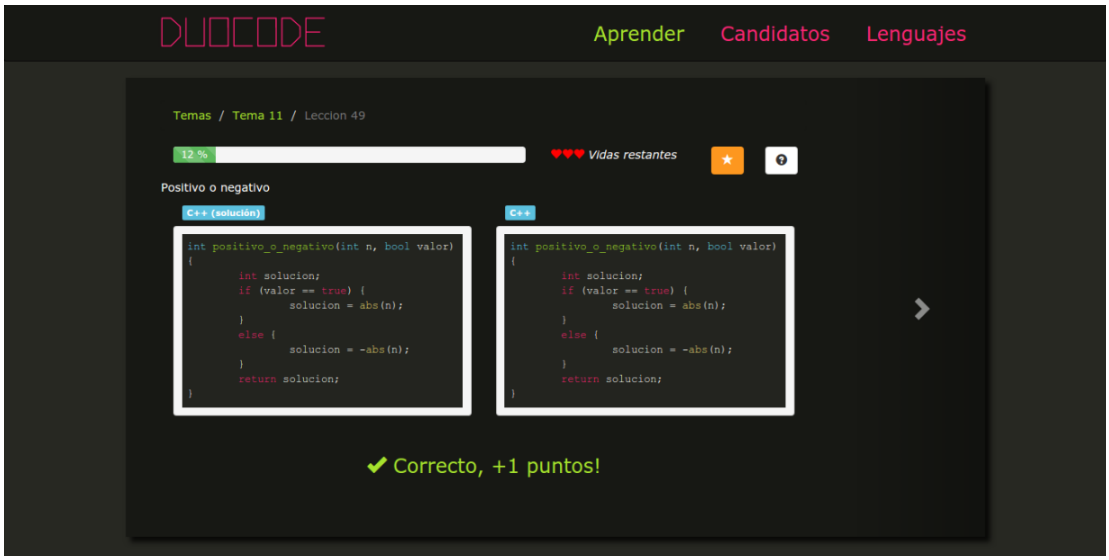
Para cada ejercicio se muestra su título y dos cuadros de código, uno con el del enunciado del ejercicio y otro vacío que es donde el usuario tendrá que resolver el ejercicio.

Por último está el botón de "Corregir" con el cual enviamos el ejercicio a ser corregido.




Después de corregir el ejercicio hay dos posibilidades:

- Solución correcta: En este caso se muestra un mensaje con la puntuación obtenida.



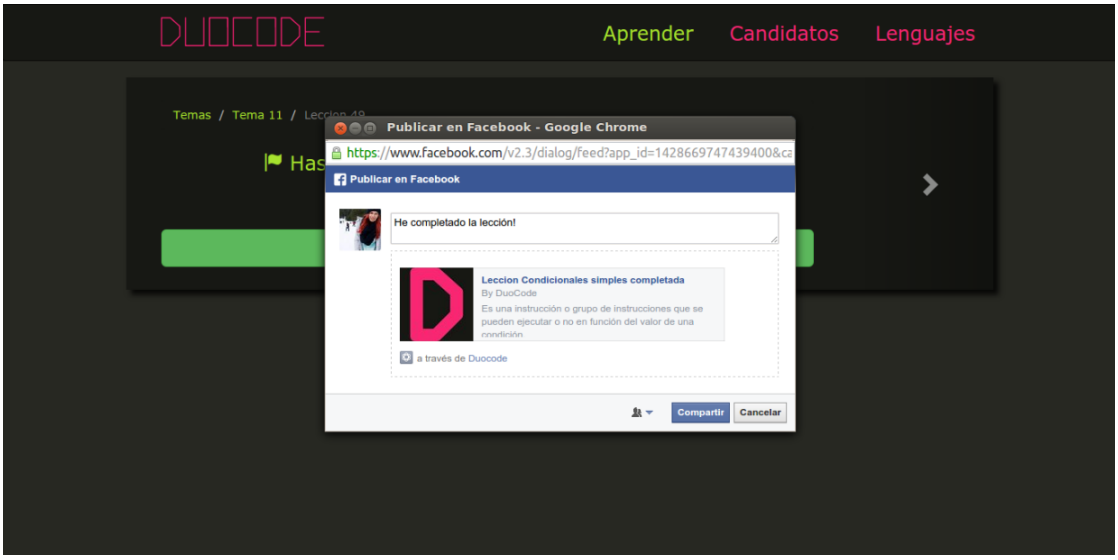

- Solución incorrecta: En este caso se resta un corazón, se muestra un mensaje de incorrecto y la puntuación obtenida. Donde estaban anteriormente el código de enunciado y la solución del usuario ahora se muestra el enunciado y la solución correcta. También se da la posibilidad al usuario de enviar su ejercicio como candidato si piensa que la solución que dio era correcta; si el usuario no sabe qué son los candidatos se encuentra también un botón con un *pop-up* explicativo.



Hay dos posibilidades de acabar las lecciones:

- Sin vidas: Una lección puede acabar si se acaban los corazones de los que dispone el usuario, es decir, si falla tres veces. Se muestra un botón mediante el cual el usuario puede volver a intentar la misma lección.


- Lección completada: Al acabar la lección con vidas, ésta se da como superada y se da la opción de compartir en Facebook el resultado. A partir de aquí podemos volver a las lecciones para seguir aprendiendo.




## 6.5. Mis favoritos

Desde la información de usuario se puede acceder a los favoritos del usuario. De cada ejercicio se muestra su título, los lenguajes que estaban seleccionados al marcar el ejercicio como favorito y un botón que al seleccionarlo muestra los códigos de origen y destino y la opción de desmarcarlo como favorito.

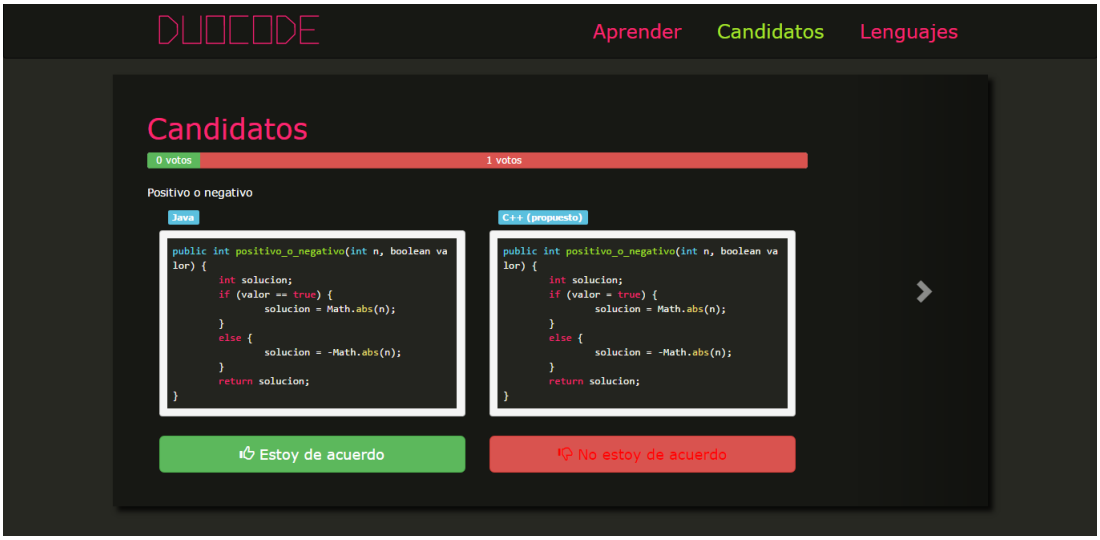

## 6.6. Mis candidatos

Desde la información de usuario se puede acceder a los candidatos del usuario. De cada ejercicio se muestra su título, los lenguajes que estaban seleccionados al enviar el ejercicio como candidato y un botón que al seleccionarlo muestra el código del enunciado, la solución

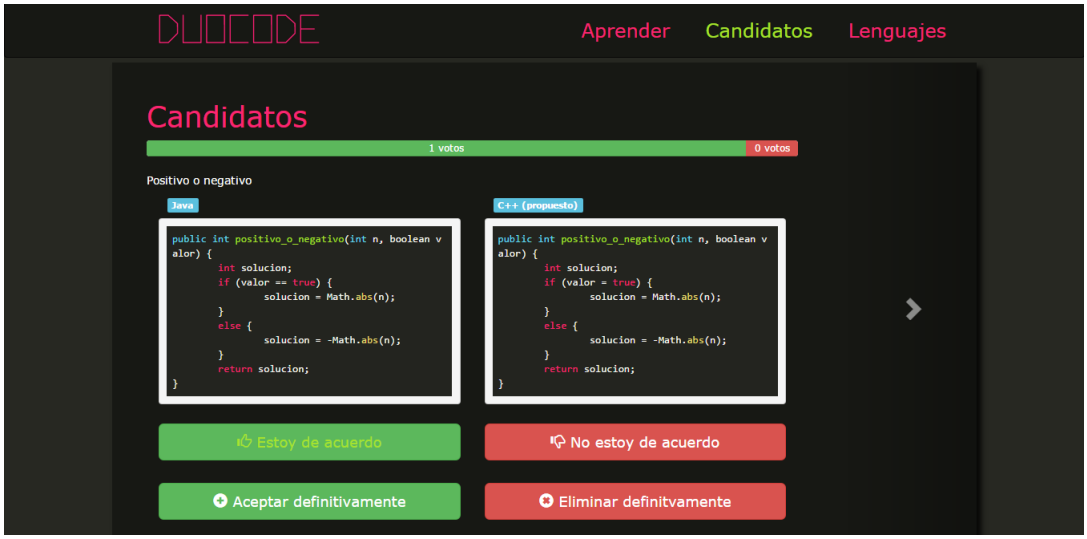
propuesta por el usuario y la opción de eliminar dicho candidato. Si el candidato ha sido aceptado o rechazado se indica en la parte superior.


## 6.7. Candidatos

Desde la barra de navegación tenemos acceso al apartado "Candidatos" en el cual se muestran los candidatos propuestos por los demás usuarios. De cada uno se muestra una barra con la cantidad de votos positivos y negativos que ha recibido, el nombre del ejercicio, el código del enunciado, el código propuesto a evaluación y dos botones en los que se puede votar si se está de acuerdo o no.




Además, si el usuario es administrador se muestran otros dos botones adicionales mediante los cuales se acepta o rechaza un candidato definitivamente. Si se acepta pasa a ser solución válida del ejercicio, si se rechaza se elimina. En cualquiera de los casos ya no se encontrará en la lista de candidatos a evaluar.




Si ya se han evaluado todos los candidatos existentes se muestra un mensaje al usuario informándole de ello y un botón que le dirige a los temas para que siga aprendiendo.

# 7. Conclusiones

En este proyecto hemos desarrollado *DuoCode*, un sistema de aprendizaje colaborativo de lenguajes de programación mediante traducciones, tomando como referencia un lenguaje conocido o incluso el lenguaje natural, en el caso de que el usuario no tenga conocimientos previos de programación.

Durante nuestro paso por la universidad tuvimos que realizar diversos proyectos para asignaturas que nos aportaron conocimientos sobre temas específicos. Con el desarrollo de *DuoCode* pudimos poner en práctica todos los conceptos aprendidos durante nuestra formación en un único proyecto. Además, nos sirvió para aprender tecnologías que no se enseñan en la universidad y cuentan con mucha demandada actualmente como *AngularJS*, *BootStrap*, *Jersey* o *PhoneGap*.

Optamos por un proceso de desarrollo ágil, dividido en sprints de una media de dos semanas de duración que finalizaban con una reunión de revisión con los tutores. Ocasionalmente nos reunimos los alumnos para tratar problemas específicos de diseño o implementación. Esta metodología de desarrollo también fue nueva para nosotros, ya que en la mayoría de las prácticas y proyectos que realizamos en la universidad seguían sistemas tradicionales como el modelo de cascada o el modelo en espiral. El resultado es satisfactorio y nos sirvió para ver cómo se desarrolla software en la actualidad, pues un gran porcentaje de las empresas de desarrollo de aplicaciones móviles y webs usan metodologías ágiles.

Además aprendimos a desarrollar un proyecto en equipo usando *Git*, un sistema de control de versiones (*VCS*) muy demandado y exigido por cualquier empresa de desarrollo. En este caso la curva de aprendizaje sí que fue elevada, algúnos de los componentes no habían usado nunca un *VCS* y al principio resultó algo complicado, pero mereció la pena el tiempo que invertimos en su aprendizaje y a día de hoy volveríamos a elegirlo sin duda.

El tema que más tiempo nos quitó fue la configuración del servidor *GlassFish* para que usase el protocolo `HTTPS` y poder cifrar el intercambio de información. Tuvimos varios problemas para configurarlo correctamente por nuestra falta de conocimientos sobre el tema, además no encontramos mucha documentación actual por internet. Por todo eso decidimos cambiar el servidor en el que alojar el servicio web por un servidor TomCat, ya que cuenta con las mismas características que *GlassFish* y hay mucha más documentación online. Esta decisión supuso un acierto y nos permitió seguir desarrollando usando el protocolo `HTTPS`.

El balance final del desarrollo de *DuoCode* es positivo. Nos ha aportado conocimientos durante el desarrollo y es una herramienta de aprendizaje a través de la cual los futuros estudiantes de informática podrán aprender un lenguaje nuevo o incluso aprender un lenguaje desde cero. Varios colegios e institutos españoles están empezando a implantar la programación como una asignatura obligatoria, y herramientas como *DuoCode* ayudarán a los alumnos a practicar los conceptos aprendidos durante las clases.

El trabajo presentado en este proyecto nos ofrece una buena base para futuras extensiones que nos permitan mejorar su usabilidad y generalidad. El uso de patrones de diseño como el *Abstract Mapper* o el framework *AngularJS* permitieron desarrollar un código mantenible y extensible para futuros desarrolladores. Aunque la versión actual de *DuoCode* es estable y cumple toda la funcionalidad descrita al comienzo de la memoria, también hay mucho trabajo futuro que desarrollar:

- Integración de **insignias digitales**. Una 'insignia digital' es el registro en línea de un logro, cualidad, calidad o interés. Se puede encontrar más información en el siguiente *enlace*: `https://support.mozilla.org/es/kb/que-es-una-insignia`
- Integración de DuoCode en el ambiente universitario y valorar las insignias conseguidas.
- Mostrar qué parte de código está mal en cada corrección.
- Intercalar preguntas en distintos lenguajes.
- Agrupar lenguajes según el paradigma de programación: Lenguajes imperativos, funcionales, lógicos, declarativos, etc.
- Coloreado de la sintaxis en tiempo real.
- Posibilidad de cambiar la paleta de colores en el diseño.
- Refinamiento de la versión móvil.
- Normalización de la base de datos.

# 8. Conclusions

In this project we have developed *DuoCode*, a new way for learning how to code. It consists of a platform where users can learn new programming languages by translating small pieces of code from a known one. It can even be the natural language if the user has never studied any programming language before.

During our years in the university we have acquired knowledge about different areas related to computer science. Developing *DuoCode* allowed us to apply all our knowledge into a single project. Moreover, we also learned new technologies that helped us growing as engineers, such as *AngularJS*, *BootStrap*, *Jersey* or *PhoneGap*.

First of all we decided to use an agile development methodology. Every sprint lasted 2 weeks and ended with a meeting where we talked about what we did. It was a new way of developing for us; we were used to apply traditional methodologies such as *Waterfall* or *Spiral Development*. It was a great decision, since we have had very good results and we have learned how to use one of the most popular ways of development nowadays.

We have also learned how to work in team by using *Git*, the most common distributed version control. It took us some weeks to learn *Git*, it is not simple and has a lot of features. However, it was very useful once we learned how to use the basic commands.

We had a lot of problems trying to configure SSL on *GlassFish* in order to use `HTTPS` protocol. It was probably the task that took most of our time so we decided to switch to *TomCat*. *TomCat* is similar to *GlassFish* but it has much more documentation. Finally, we could configure the server to use the `HTTPS` protocol.

We are proud of the final result. We have learned new technologies and *DuoCode* is a good way for future students to learn the basics of coding in a new language. Nowadays a lot of schools are interested in teaching how to code. *DuoCode* is a very useful platform where students can practice and apply their knowledge.

*DuoCode* constitutes a good base for future extensions, so it can be extended on many ways. We have used design patterns such as *Abstract Mapper* or *AngularJS* framework to obtain a maintainable code, making it easier to future developers to extend their project. Although the actual version of *DuoCode* is stable, a lot of new features can be implemented:

- Integrate *Open Badges*: a new online standard to recognize and verify learning. More info can be found at `http://openbadges.org/`
- Show which parts of the code are wrong.
- Intercalate questions in different languages.
- Group languages with same programming paradigm.
- Highlight syntax in real time.
- Allow the user to change the interface colors.
- Improve the mobile version.
- Implement database normalization.

# 9. Tareas desarrolladas por los estudiantes

Se presentan a continuación las tareas desarrolladas por cada uno de los estudiantes.

## 9.1. Julián F. Calleja da Silva

Todos nos hemos ayudado y colaborado durante este proyecto pero en esta sección explicaré las tareas que yo, Julián F. Calleja da Silva, específicamente he realizado.

- Cuando investigamos los diferentes enfoques para realizar servicios web, me encargué de SOAP, vi sus ventajas y desventajas e incluso hice un pequeño ejemplo para entender mejor su funcionamiento.
- Hicimos una reunión para ponernos de acuerdo en cual era el mejor enfoque para el servicio web antes de presentar los resultados a los profesores. Me encargué de organizarla y presidirla.
- Hice un ejemplo simple de servicio web REST usando el IDE eclipse y el *framework* Spring. Consistía en hacer el factorial de un número. El objetivo era poder comparar distintos IDEs y distintos *frameworks* con los que poder realizar el servicio web.
- Me encargué de revisar las tablas de la base de datos y comprobé que estaban como mínimo normalizadas hasta la forma normal de *Boyce-Codd*.
- Aporté la clase AbstractMapper explicada en la sección 3.3.3 y amplié su funcionalidad, permitiendo que el método `'insert'` devuelviese un entero con el valor del campo autoincremental de la tabla en caso de que exista y $-1$ en caso contrario. Además añadí todo lo necesario para usar la base de datos desde el proyecto REST.
- Creé varias clases que representan el modelo de la aplicación, sus correspondientes `'mappers'` para acceder a la base de datos descrita en la sección 3.3.3 y las funciones para los distintos verbos de los recursos REST (GET, POST, PUT y DELETE). En concreto implementé:
  - Todo lo relacionado con los ejercicios, incluyendo la lista de los enunciados de los distintos lenguajes de programación correspondientes un ejercicio en su GET.
  - Todo lo relacionado con los enunciados, incluyendo un método en su `'mapper'` para obtener todos los enunciados de un ejercicio específico.
  - Todo lo relacionado con los envíos, incluyendo un método en su `'mapper'` para obtener todos los envíos de un usuario. Además creé la clase `'Puntuador'`, que encapsula la capacidad de evaluar un fragmento de código en un lenguaje concreto para un ejercicio específico.
- Realicé lo necesario para asegurarnos que el usuario se ha autenticado correctamente y es quien dice ser. Para ello creé un paquete llamado `'autentificacion'`, que contiene clases que preguntan a los servidores de Google y Facebook por los datos de un usuario (nombre, enlace a su foto de perfil, etc.) y otra clase con métodos estáticos que se encarga de comprobar, una vez el usuario está correctamente autenticado, si existe en nuestra base de datos, y en caso contrario lo crea.

- Me encargué de que la visualización de la web se adaptara correctamente a distintos dispositivos, teniendo en cuenta principalmente dispositivos móviles y ordenadores de sobremesa. Para ello usé el sistema de columnas de Bootstrap.
- Creé los botones que usamos en la web. Además del texto elegí los iconos más adecuados para la acción que representan. Tanto para el estilo como los iconos usé las clases CSS disponibles en Bootstrap.
- Hice la barra que muestra el progreso de los ejercicios completados en una lección. Para ello usé componentes de Bootstrap y los configuré para que tuvieran el color y la apariencia deseados.
- Hice parte de las vistas que muestran la información del usuario. En concreto me encargué de la imagen de perfil, el nombre, los puntos conseguidos hasta el momento y los enlaces a sus favoritos y sus candidatos.
- Busqué una biblioteca JavaScript que resaltara la sintaxis de los distintos lenguajes de programación. Descargué `'highlight.js'` y la configuré adecuadamente.
- Realicé la vista donde los usuarios pueden votar las soluciones propuestas a candidatos. Además integré una barra que muestra gráficamente los votos a favor y en contra de los usuarios.
- Realicé la vista que muestra que no hay más candidatos disponibles para votar y te invita a seguir realizando ejercicios.
- Me encargué de realizar las peticiones al servicio web para saber los ejercicios que corresponden a una lección.
- Controlé cuándo un usuario ha terminado todos los ejercicios necesarios para completar la lección. En ese caso se muestra una pantalla de felicitación y se le invita a seguir aprendiendo.
- Realicé la función que se encarga de marcar un ejercicio como favorito y la asocié al botón correspondiente. Además, hice que este cambiara de color según su estado.
- Realicé la función que se encarga de crear un candidato a partir de una solución incorrecta y la asocié al botón correspondiente.
- Conecté la vista para votar candidatos con el servicio web e hice que el botón para pasar al siguiente candidato tuviera la funcionalidad deseada.
- Realicé la función que se encarga de votar a un candidato, la asocié a los botones correspondientes, hice que al votar se actualizase la barra que muestra los votos a favor y en contra y cambié los colores de los botones para votar positiva o negativamente según la acción del usuario.
- Añadí toda la estructura genérica para permitir al usuario autenticarse. Para ello usé la biblioteca HelloJS. Además, me encargué de que cuando el usuario está autenticado se mande en cada petición su identificador y su *token*. Por último hice que el componente que se usa cuando quieres información del usuario devuelva uno vacío con el rol $-1$ cuando no hay un usuario autenticado.
- En la memoria me encargué de hacer la introducción (sección 1) y su traducción al inglés (sección 2).

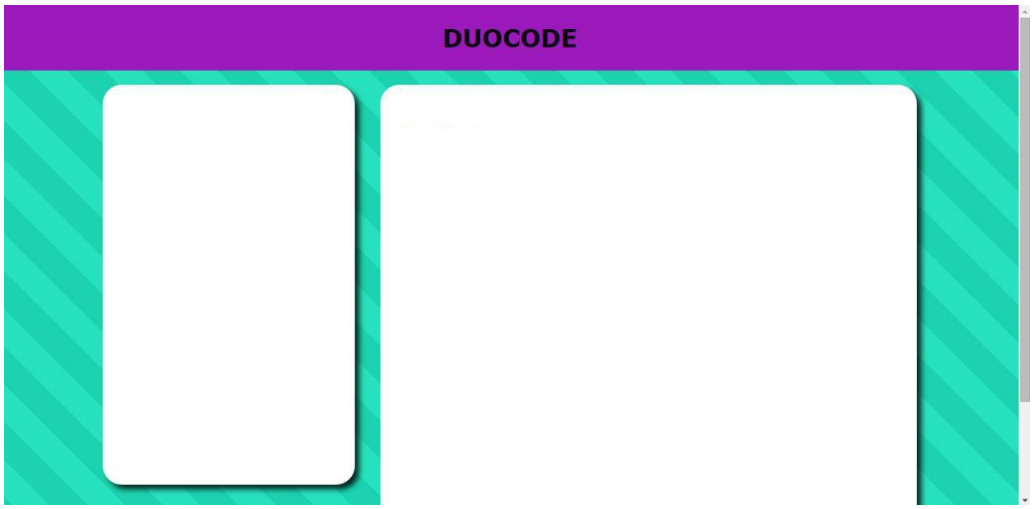
- En la memoria me encargué de escribir sobre el *front-end* (sección 4).
- Realicé una lista con las tareas más importantes realizadas durante el proyecto y me encargué de organizar una reunión y presidirla. En ella revisamos cada tarea, nos acordamos de quién la realizó y lo apuntamos para poder especificarlo en la memoria.

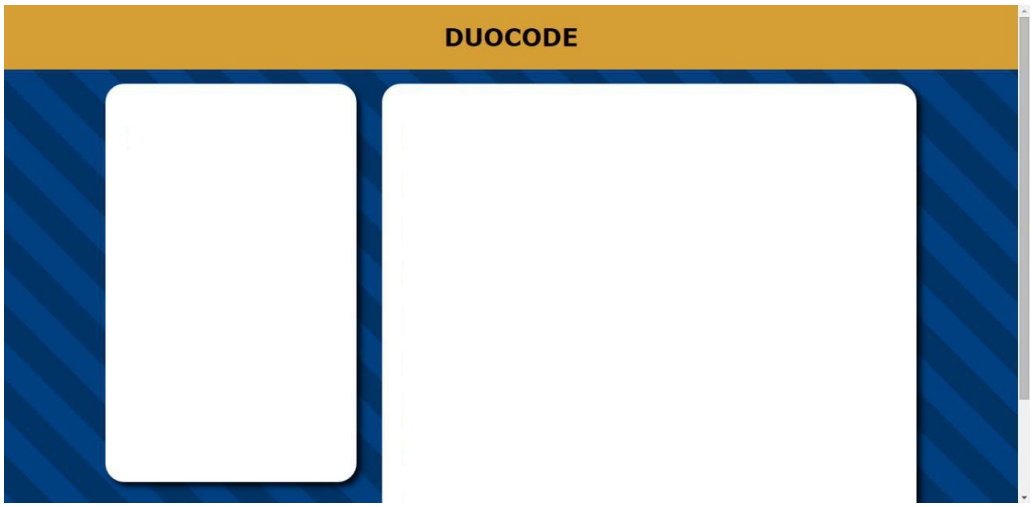
## 9.2. Johana Gabriela Ferreira Yagua

Todos nos hemos ayudado y colaborado durante este proyecto pero en esta sección explicaré las tareas que yo, Johana Gabriela Ferreira Yagua, específicamente he realizado.

- Al empezar el proyecto, una de las principales cosas que teníamos que decidir era qué tipo de servicio web utilizar. Yo me encargué de buscar información sobre RCP y sus ventajas y desventajas para compararlas con SOAP y REST.
- Intenté hacer dos ejemplos de servicio web REST, uno utilizando Spring con Eclipse y otro con Jersey para comparar cuál era más sencillo de utilizar. En ambos casos el ejemplo utilizado fue el algoritmo de factorial.
- Hicimos una reunión, presidida por mí, para crear un diseño inicial del modelo entidad-relación de la base de datos.
- Me encargué de pasar el diseño del modelo entidad-relación a ordenador y de hacer los cambios necesarios.
- Después de tener el modelo entidad-relación, creé la base de datos, las tablas con sus correspondientes atributos y las relaciones con `phpMyAdmin`.
- Hice la población de la base de datos con información válida para realizar las pruebas.
- Creé varias clases que representan el modelo de la aplicación, sus correspondientes `'mappers'` para acceder a la base de datos descrita en la sección 3.3.3 y las funciones para los distintos verbos de los recursos REST (GET, POST, PUT y DELETE). En concreto implementé:
  - Todo lo relacionado con los usuarios, incluyendo el GET de un usuario específico, que aporta toda la información registrada del usuario en la aplicación (historial de ejercicios, candidatos propuestos, ejercicios favoritos, lecciones completadas y votos a otros candidatos). Aparte del `'mapper'` de usuario, existen otros dos `'mappers'`, uno para las lecciones que ha completado el usuario y otro para los votos que ha realizado a distintos candidatos.
  - Todo lo relacionado con los candidatos, incluyendo en el PUT la diferenciación si se trata de un usuario común o de un administrador: en el primer caso el usuario solo puede modificar el voto a positivo o negativo; en el segundo, además de tener esa capacidad, puede gestionar el ejercicio dándolo por válido o por rechazado.
  - Todo lo relacionado con los lenguajes de programación soportados por la herramienta.
- Realicé los prototipos en papel para tener una idea de cómo queríamos que fuese el aspecto visual de la aplicación y su estructura interna.
- Realicé los primeros ejemplos con HTML y CSS con la estructura anteriormente definida e hice que la sección de información del usuario fuese fija para que estuviese presente en todo momento.

- Hice pruebas con distintas combinaciones de colores y fondos, por ejemplo:




  Finalmente, pensé en probar con la paleta de colores `Monokai`, ya que es muy utilizada en programación; esta fue la definitiva.

- Incorporé con Bootstrap un pop-up explicativo al iniciar cada lección, el cual está accesible una vez iniciada la sesión por si el usuario quiere volver a verlo. También hice un pop-up explicando qué significa enviar un ejercicio como candidato cuando se da esta opción.

- Hice la vista de favoritos del usuario. Para que no saliese demasiada información en esta vista, al principio solo se muestra el título de los ejercicios marcados como favoritos y sus correspondientes lenguajes de programación en los que fueron marcados. Usé la opción `Collapse` de Bootstrap para mostrar y ocultar el código de los ejercicios cuando el usuario seleccione uno de ellos.

- Hice la vista de candidatos de un usuario basándome en la de favoritos para buscar consistencia visual en la web.

- Hice la vista de lenguajes que se muestra al entrar en la web. Para que un usuario no pueda elegir el mismo lenguaje de programación en las listas "Lenguaje que sé" y "Lenguaje que quiero aprender" utilicé AngularJS.
- Me encargué de realizar las peticiones al servicio web para saber las lecciones que corresponden a un tema.
- Me encargué de realizar las peticiones al servicio web para obtener la información del usuario:
  - Nombre y foto.
  - Número de candidatos enviados.
  - Número de ejercicios marcados como favorito.
- Me encargué de realizar las peticiones al servicio web para obtener los candidatos de un usuario.
- Me encargué de realizar las peticiones al servicio web para obtener los favoritos de un usuario.
- Hice que en el pop-up explicativo de las lecciones saliera la información correspondiente.
- Añadí la función de iniciar sesión con Google+ y con Facebook mediante la estructura creada por Julián F. Calleja da Silva.
- Añadí la función de compartir con Facebook el logro tras superar una lección.
- En la memoria me encargué de hacer el resumen y la lista de palabras clave y su traducción al inglés.
- En la memoria me encargué de hacer la sección sobre los requisitos y la base de datos (sección 5).
- En la memoria me encargué de hacer el manual de usuario (sección 6).
- En la memoria me encargué de hacer el apéndice sobre la especificacion de requisitos (apéndice A).

## 9.3. José Carlos Valera Villalba

Todos nos hemos ayudado y colaborado durante este proyecto pero en esta sección explicaré las tareas que yo, José Carlos Valera Villalba, específicamente he realizado.

- Cuando investigamos los diferentes enfoques para realizar servicios web, me encargué de buscar información sobre REST, vi sus ventajas y desventajas frente a *SOAP* y *RCP*.
- En la reunión para decidir qué tipo de servicio web implementaríamos compartí la información encontrada sobre REST y mi punto de vista. Discutimos los pros y los contras de cada tecnología y al final decidimos que REST era lo que más se ajustaba a nuestras necesidades.
- Hice un ejemplo simple de servicio web REST usando el IDE *Netbeans* y el *framework Jersey*. Consistía en devolver el factorial de un número, pasándole el número en la cabecera de la petición *HTTP* y en la *query*. El objetivo era poder comparar distintos *IDEs* y *frameworks* con los que realizar el servicio web. *NetBeans* era una buena opción por su integración de *Git* y toda la documentación online que hay para desarrollar servicios web usando este entorno.
- Hicimos una reunión para decidir el *framework* y el IDE que íbamos a usar y yo fui el organizador. Al final decidimos usar *Jersey* como framework y *Netbeans* como IDE.
- Me encargué de crear el proyecto, con la estructura básica de todas las clases que necesitaríamos y lo subí a github.
- Descargué el framework *Jersey* y lo integré en el proyecto. Completé las clases creadas con la sintaxis de *Jersey* y comprobé su correcto funcionamiento con datos sencillos usando 'Advanced Rest Client', una aplicación para testear servicios web tipo REST.
- Creé varias clases que representan el modelo de la aplicación, sus correspondientes '`mappers`' para acceder a la base de datos explicada en la sección 3.3.3 y las funciones para los distintos verbos de los recursos REST (GET, POST, PUT y DELETE). En concreto implementé:
  - Todo lo relacionado con la gestión de los favoritos. La implementación del '`mapper`' necesario y los métodos del recurso REST.
  - Todo lo relacionado con las lecciones. La implementación del '`mapper`' necesario y los métodos del recurso REST.
  - Todo lo relacionado con los temas. La implementación del '`mapper`' necesario y los métodos del recurso REST.
- Implementé la vista de las lecciones en HTML. Desarrollé el controlador adecuado en *AngularJS* y las funciones necesarias para la gesión de las lecciones bloqueadas de cada usuario.
- Implementé la vista de los temas en HTML. Desarrollé el controlador adecuado en *AngularJS* y la función necesaria para que los temas apareciesen ordenados correctamente.

- Implementé la vista de los ejercicios en HTML cuando la respuesta del usuario ha sido correcta e incorrecta. Desarrollé en el controlador adecuado en AngularJS las funciones necesarias para mostrar en la vista el resultado de la corrección del ejercicio.
- Implementé la vista de los ejercicios cuando te has quedado sin vidas. En el controlador de ejercicios creé las funciones necesarias para llevar la cuenta de las vidas y que cuando se quedase a cero notificase al usuario.
- Me encargué de configurar los servidores (*TomCat* para el servicio web y el Apache que viene en *XAMPP*) para que usasen el protocolo HTTPS y proteger el tráfico de datos.
- Implementé las peticiones HTTP al servicio web para obtener la lista de lenguajes disponibles en nuestro sistema.
- Implementé las peticiones HTTP al servicio web para obtener todos los temas disponibles en nuestro sistema.
- Me encargué de que ciertas partes de la aplicación solo fuesen visibles cuando el usuario está logueado. Por ejemplo 'compartir en Facebook' solo se mostraría cuando el usuario se ha identificado usando su cuenta de Facebook. Para ello usé la clase `ng-hide` de AngularJS y desarrollé las funciones necesarias para gestionar la visibilidad en los controladores.
- Me encargué del desarrollo de la aplicación móvil. Usé *PhoneGap* y creé la aplicación Android usando la clase 'WebView', que lo que hace básicamente es cargar una vista web en el móvil optimizada para su correcta visualización. *PhoneGap* también gestiona los touch events, por lo que el programador no los tiene que tener en cuenta durante el desarrollo.
- En la memoria me encargué de escribir la sección sobre el servicio web, donde explico principalmente la arquitectura usada y el framework que elegimos para implementar la API REST.
- En la memoria me encargué de escribir la sección del manual de instalación, donde explico todo el software necesario para extender el proyecto y los pasos a seguir para desplegar *DuoCode* en un sistema Linux.
- En la memoria me encargué de escribir las conclusiones obtenidas después de finalizar el desarrollo del proyecto en inglés y español. También redacté la lista de posibles mejoras que se pueden implementar en el futuro.

# Referencias


[1] Acuerdo del Consejo de Universidades, (B.O.E. 4 de agosto de 2009). http://www.boe.es/boe/dias/2009/08/04/pdfs/BOE-A-2009-12977.pdf.

[2] AngularJS. https://angularjs.org/.

[3] Bootstrap. http://getbootstrap.com/.

[4] Bussu. https://www.busuu.com/es/.

[5] Coursera. https://es.coursera.org/.

[6] Duolingo. https://es.duolingo.com/.

[7] EdX. https://www.edx.org/.

[8] Facebook. https://es-es.facebook.com/.

[9] Facebook SDK para JavaScript. https://developers.facebook.com/docs/javascript.

[10] HelloJS. http://adodson.com/hello.js/.

[11] highlight.js. https://highlightjs.org/.

[12] jQuery. https://jquery.com/.

[13] Miriada X. https://www.miriadax.net/.

[14] Monokai. http://www.colourlovers.com/palette/1718713/Monokai.

[15] Moodle. https://moodle.org/.

[16] Plan de Estudios del Grado de Ingeniería Informática de la universidad Complutense. https://informatica.ucm.es/estudios/2014-15/grado-ingenieriainformatica-planestudios.

[17] Providers de AngularJS. https://docs.angularjs.org/guide/providers.

[18] Stack Overflow. http://stackoverflow.com/.

[19] Wikipedia. https://es.wikipedia.org.

[20] A. R. (coordinador). Estudio de viabilidad de un entorno de aprendizaje colaborativo de lenguajes de programación. Informe del PIMCD 97/2014, 2015.

# A. Especificación de requisitos

En esta sección se exponen todos los requisitos existentes del servicio web REST de nuestra aplicación, explicando la función de cada uno de ellos, sus parámetros de entrada y su respuesta.

## A.1. localhost/duocode/rest/temas

En este apartado se explica cómo hacer un GET para obtener todos los temas y un POST para crear un nuevo tema.

**REQ01 – GET:** Devuelve la lista de todos los temas existentes en modo URL. No hace falta enviarle ninguna información.

- Response:

```
{‘temas’: [‘localhost/duocode/rest/temas/1’,
‘localhost/duocode/rest/temas/2’]}
```

**REQ02 – POST:** Crea un tema nuevo. Por una parte le pasamos por Payload la información del tema que queremos crear (título, descripción y orden en el que queremos que se muestre):

- Payload:

```
{‘titulo’: ‘Bucles’, ‘descripcion’: ‘En este tema veremos bucles while’,
‘orden’: ‘2’}
```

  Por otra parte le enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error” e “id”. Si la respuesta en el campo “error” es afirmativa significa que algo ha fallado y no ha podido terminar la operación correctamente, en este caso el id será “-1”, ya que no ha sido asignado ninguno; si es negativa todo ha ido correctamente y el id será el asignado a este nuevo recurso. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{‘error’: ‘no’, ‘id’: ‘4’}
{‘error’: ‘si’, ‘id’: ‘-1’}
```

## A.2. localhost/duocode/rest/temas/idTema

En este apartado se explica cómo hacer un GET para obtener un tema en específico, un PUT para modificarlo y un DELETE para eliminarlo. En estos requisitos obtenemos el ID del tema mediante la URL.

**REQ03 – GET:** Devuelve los datos y las lecciones que tiene el tema.

- Response:

```
{‘titulo’: ‘Bucles’, ‘descripcion’: ‘En este tema veremos bucles while’,
‘fechaCreacion’: ‘17/11/2014’, ‘lecciones’:
[‘localhost/duocode/rest/lecciones/1’, ‘localhost/duocode/rest/lecciones/2’],
 ‘orden’: ‘2’}
```

**REQ04 – PUT:** Modifica el tema en su totalidad.

- Payload:

```
{‘titulo’: ‘tituloTema’, ‘descripcion’: ‘descripcionTema’,
‘orden’: ‘int con el orden en el que lo quieres mostrar’}
```

  Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error”. En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{‘error’: ‘no’}
```

**REQ05 – DELETE:** Borra el tema completamente. Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error”, en caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{‘error’: ‘no’}
```

## A.3. localhost/duocode/rest/lecciones/

En este apartado se explica cómo hacer un GET para obtener todas las lecciones y un POST para crear una nueva lección.

**REQ06 – GET:** Devuelve la lista de todas las lecciones existentes en modo URL. No hace falta enviarle ninguna información.

- Response:

```
{‘lecciones’: [‘localhost/duocode/rest/lecciones/1’,
‘localhost/duocode/rest/lecciones/2’]}
```

**REQ07 – POST:** Crea una nueva lección. Por una parte le pasamos por Payload la información de la lección que queremos crear (título, descripción, explicación que aparecerá al inicio de ésta, orden en el que queremos que se muestre, ID del tema al que pertenecerá, un array de ejercicios que compondrán la lección y otro array de lecciones de las que depende para ser desbloqueada):

- Payload:

```
{‘titulo’: ‘titulo lección’, ‘descripcion’: ‘descripción lección’,
‘explicacion’: ‘explicación detallada’, ‘orden’: ‘4’, ‘idTema’: ‘1’,
‘idEjercicios’: [‘8’, ‘14’], ‘leccionesDesbloqueadoras’: [‘1’, ‘2’]}
```

  Por otra parte le enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error” e “id”. Si la respuesta en el campo “error” es afirmativa significa que algo ha fallado y no ha podido terminar la operación correctamente, en este caso el id será “-1” ya que no ha sido asignado ninguno; si es negativa todo ha ido correctamente y el id será el asignado a esta nueva lección. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{‘error’: ‘no’, ‘id’: ‘3’}
{‘error’: ‘si’, ‘id’: ‘-1’}
```

## A.4. localhost/duocode/rest/lecciones/idLeccion

En este apartado se explica cómo hacer un GET para obtener una lección específica, un PUT para modificarla y un DELETE para eliminarla. En estos requisitos obtenemos el ID de la lección mediante la URL.

**REQ08 – GET:** Devuelve los datos de la lección.

- Response:

```
{‘titulo’: ‘título de la lección (ej. Bucles fácil)’,
‘descripcion’: ‘descripción de la lección (ej. bucles para practicar)’,
‘explicacion’: ‘explicación detallada’, ‘fechaCreacion’: ‘17/11/2014’,
‘ejercicios’: [‘localhost/duocode/rest/ejercicios/2’,
‘localhost/duocode/ejercicios/3’],
‘leccionesDesbloqueadoras’: [‘1’, ‘2’], ‘orden’: ‘3’}
```

**REQ09 – PUT:** Modifica una lección. Aparte de cambiar los datos de una lección, en este método también podemos marcar como completada la lección para un usuario en un determinado lenguaje, por lo que tenemos dos posibles Payloads:

- Payload para modificar lección:

```
{’leccion’ : {‘titulo’: ‘titulo lección’, ‘descripcion’:
‘descripción lección’, ‘explicacion’: ‘explicación detallada’,
‘idEjercicios’: [‘1’, ‘2’], ‘leccionesDesbloqueadoras’: [‘2’, ‘3’],
‘orden’: ‘2’, ‘idTema’: ‘1’}}
```

- Payload para marcar como lección completada para un usuario:

```
{‘idUsuarioCompletaLeccion’: ‘3’, ‘lenguajeCompletadoLeccion’: ‘Java’,
‘leccion’: {‘titulo’: ‘titulo lección’,
‘descripcion’: ‘descripción de la lección’,
‘explicacion’:‘explicación detallada’, ‘idEjercicios’: [‘1’, ‘2’],
‘leccionesDesbloqueadoras’: [‘2’, ‘3’], ‘orden’: ‘2’, ‘idTema’: ‘1’}}
```

  Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error”. En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{‘error’: ‘no’}
```

**REQ10 – DELETE:** Borra la lección completamente. Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

  ```
  {'idUsuario': '2', 'token': 'token', 'network': 'network'}
  ```

  La respuesta que obtenemos está conformada por "error". En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

  ```
  {'error': 'no'}
  ```

## A.5. localhost/duocode/rest/ejercicios

En este apartado se explica cómo hacer un GET para obtener todos los ejercicios y un POST para crear un nuevo ejercicio.

**REQ11 – GET:** Devuelve la lista de todos los ejercicios existentes en modo URL. No hace falta enviarle ninguna información.

- Response:

  ```
  {'ejercicios': ['localhost/duocode/rest/temas/ejercicios/1',
  'localhost/duocode/rest/ejercicios/2']}
  ```

**REQ12 – POST:** Crea un ejercicio nuevo. Por una parte le pasamos el nombre y los enunciados (un mismo ejercicio puede tener varios enunciados porque cada uno corresponde a distintos lenguajes de programación):

- Payload:

  ```
  {'nombre': 'nombreDelEjercicio', 'enunciados': ['1', '2']}
  ```

  Por otra parte le enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

  ```
  {'idUsuario': '2', 'token': 'token', 'network': 'network'}
  ```

  La respuesta que obtenemos está conformada por "error" e "id". Si la respuesta en el campo "error" es afirmativa significa que algo ha fallado y no ha podido terminar la operación correctamente, en este caso el id será "-1" ya que no ha sido asignado ninguno; si es negativa todo ha ido correctamente y el id será el asignado a este nuevo recurso. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

  ```
  {'error': 'no', 'id': '4'}
  {'error': 'si', 'id': '-1'}
  ```

## A.6. localhost/duocode/rest/ejercicios/idEjercicio

En este apartado se explica cómo hacer un GET para obtener un ejercicio en específico, un PUT para modificarlo y un DELETE para eliminarlo. En estos requisitos obtenemos el ID del ejercicio mediante la URL.

**REQ13 – GET:** Devuelve los datos del ejercicio, los enunciados que tiene y el nombre del lenguaje asociado a cada uno.

- Response:

```
{‘nombre’: ‘nombreDelEjercicio’, ‘fechaCreacion’: ‘17/11/2014’,
‘enunciados’: [{‘enunciado’: ‘localhost/duocode/rest/enunciados/5’,
‘nombreLenguaje’: ‘Java’}, {‘enunciado’:
‘localhost/duocode/rest/enunciados/8’, ‘nombreLenguaje’: ‘C++’}]}
```

**REQ14 – DELETE:** Borra un ejercicio completamente. Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error”. En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{‘error’: ‘no’}
```

## A.7. localhost/duocode/rest/enunciados

En este apartado se explica cómo hacer un GET para obtener todos los enunciados y un POST para crear un nuevo enunciado.

**REQ15 – GET:** Devuelve la lista de todos los enunciados existentes en modo URL. No hace falta enviarle ninguna información.

- Response:

```
{‘enunciados’: [{‘enunciado’: ‘localhost/duocode/rest/enunciados/1’,
‘nombreLenguaje’: ‘Java’}, {‘enunciado’: ‘localhost/duocode/rest/enunciados/2’,
‘nombreLenguaje’: ‘C++’}]}
```

**REQ16 – POST:** Crea un enunciado nuevo. Por una parte le pasamos el lenguaje, el código y el ID del ejercicio correspondiente:

- Payload:

```
{‘nombreLenguaje’: ‘Java’, ‘codigo’: ‘código del enunciado’,
‘idDelEjercicioQueResuelve’: ‘1’}
```

Por otra parte le enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

La respuesta que obtenemos está conformada por “error” e “id”. Si la respuesta en el campo “error” es afirmativa significa que algo ha fallado y no ha podido terminar la operación correctamente, en este caso el id será “-1” ya que no ha sido asignado ninguno; si es negativa todo ha ido correctamente y el id será el asignado a este nuevo enunciado. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{‘error’: ‘no’, ‘id’: ‘4’}
{‘error’: ‘si’, ‘id’: ‘-1’}
```

## A.8. localhost/duocode/rest/enunciados/idEnunciado

En este apartado se explica cómo hacer un GET para obtener un enunciado en específico, un PUT para modificar un enunciado y un DELETE para eliminarlo. En estos requisitos obtenemos el ID del candidato mediante la URL.

**REQ17 – GET:** Devuelve los datos del enunciado.

- Response:

```
{‘fechaCreacion’: ‘17/11/2014’, ‘codigo’: ‘código del enunciado a resolver’,
‘nombreLenguaje’: ‘Java’, ‘idDelEjercicioQueResuelve’: ‘1’}
```

**REQ18 – PUT:** Modifica un enunciado.

- Payload:

```
{‘nombreLenguaje’: ‘Java’, ‘codigo’: ‘código del enunciado’,
‘idDelEjercicioQueResuelve’: ‘1’}
```

Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{'idUsuario': '2', 'token': 'token', 'network': 'network'}
```

La respuesta que obtenemos está conformada por "error". En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{'error': 'no'}
```

**REQ19 – DELETE:** Borra un enunciado completamente. Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{'idUsuario': '2', 'token': 'token', 'network': 'network'}
```

La respuesta que obtenemos está conformada por "error". En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{'error': 'no'}
```

## A.9. localhost/duocode/rest/lenguajes/

En este apartado se explica cómo hacer un GET para obtener todos los lenguajes y un POST para crear un nuevo lenguaje.

**REQ20 – GET:** Devuelve la lista de todos los lenguajes existentes. No hace falta enviarle ninguna información.

- Response:

```
{'lenguajes': [{'nombre': 'Java'}, {'nombre': 'C++'}]}
```

**REQ21 – POST:** Crea un lenguaje nuevo. La única información necesaria es el nombre.

- Payload:

```
{'nombre': 'Python'}
```

Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error” y “nombreConfirmacion”. Si la respuesta en el campo “error” es afirmativa significa que algo ha fallado y no ha podido terminar la operación correctamente; si es negativa todo ha ido correctamente y el “nombreConfirmacion” será el asignado a este nuevo lenguaje. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{‘error’: ‘no’, ‘nombreConfirmacion’: ‘Python’}
```

## A.10. localhost/duocode/rest/candidatos/

En este apartado se explica cómo hacer un GET para obtener todos los candidatos y un POST para crear un nuevo candidato.

**REQ22 – GET:** Devuelve la lista de todos los candidatos existentes en modo URL. No hace falta enviarle ninguna información.

- Response:

```
{‘candidatos’: [‘localhost/duocode/rest/candidatos/1’,
‘localhost/duocode/rest/candidatos/2’]}
```

**REQ23 – POST:** Crea un nuevo candidato asociado al usuario que solicita la petición. Por un lado le enviamos el código del candidato, el ID del ejercicio que resuelve, el lenguaje en el que está escrito el candidato y el lenguaje del enunciado:

- Payload:

```
{‘codigo’: ‘codigoDelCandidato’, ‘idEjercicio’: ‘4’,
‘nombreLenguajeDestino’: ‘Java’, ‘nombreLenguajeOrigen’: ‘C++’}
```

  Por otra parte le enviamos por Header el ID del usuario, el token y el network, que nos sirven para saber qué usuario es el que ha enviado el candidato:

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error” e “idCandidato”. Si la respuesta en el campo “error” es afirmativa significa que algo ha fallado y no ha podido terminar la operación correctamente, en este caso el id será “-1” ya que no ha sido asignado ninguno; si es negativa todo ha ido correctamente y el id será el asignado a este nuevo candidato.

- Response:

```
{‘error’: ‘no’, ‘id’: ‘4’}
```

## A.11. localhost/duocode/rest/candidatos/idCandidato

En este apartado se explica cómo hacer un GET para obtener un candidato en específico, un PUT para modificarlo y un DELETE para eliminarlo. En estos requisitos obtenemos el ID del candidato mediante la URL.

**REQ24 – GET:** Devuelve los datos de un candidato, incluidos los votos que tiene:

- Response:

```
{'idEjercicio': 'localhost/duocode/rest/ejercicios/4',
'nombreLenguajeOrigen': 'Java', 'nombreLenguajeDestino: 'C++',
'codigo': 'código del candidato', 'idUsuarioCreador': '2',
'fechaCreacion': '17/11/2014', 'votos': [{'idUsuarioVoto': '8',
'voto': 'pos'}, {'idUsuarioVoto':'5', 'voto': 'neg'}]}
```

**REQ25 – PUT:** Excepcionalmente no modifica todo el candidato, sino que sirve para que un usuario pueda votar, modificar el voto o eliminarlo (si se vuelve a votar positivo o negativo el voto se anula). Se envía el ID del usuario y el voto (1 si es positivo y 0 si es negativo).

- Payload:

```
{'votar': {'idUsuario': '6', 'voto': '1'}}
```

  La respuesta que obtenemos está conformada por "error". En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente.

- Response:

```
{'error': 'no'}
```

**REQ26 – DELETE:** Borra un candidato completamente.

- Header:

```
{'idUsuario': '2', 'token': 'token', 'network': 'network'}
```

  La respuesta que obtenemos está conformada por "error". En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

```
{'error': 'no'}
```

## A.12. localhost/duocode/rest/usuarios/

En este apartado se explica cómo hacer un GET para obtener todos los usuarios.

**REQ27 – GET:** Devuelve todos los usuarios. A diferencia de otros, este GET solo lo puede hacer un administrador por lo que necesitamos nuevamente del Header.

- Header:

  ```
  {‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
  ```

  La respuesta que obtenemos está conformada por “error” y la lista de usuarios. Si “error” tiene una respuesta afirmativa, significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente y muestra la lista de usuarios mediante su URL. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

  ```
  {‘error’: ‘no’, ‘usuarios’: [‘localhost/duocode/rest/usuario/1’,
  ‘localhost/duocode/rest/usuario/2’]}
  ```

## A.13. localhost/duocode/rest/usuarios/idUsuario

En este apartado se explica cómo hacer un GET para obtener un usuario en específico y un DELETE para eliminarlo. En estos requisitos obtenemos el ID del candidato mediante la URL.

**REQ28 – GET:** Devuelve la información asociada a un usuario. Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario que accede es el mismo del que se da la información:

- Header:

  ```
  {‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
  ```

  La respuesta es la información detallada de toda la sesión del usuario

- Response:

  ```
  {‘nick’: ‘nickDelUsuairo’, ‘leccionesCompletadas’:
  [‘localhost/duocode/rest/lecciones/1’, ‘localhost/duocode/rest/lecciones/2’],
  ‘favoritos’: [{‘ejercicio’: ‘localhost/duocode/rest/ejercicios/5’,
  ‘nombreLenguajeOrigen’: ‘Java’, ‘nombreLenguajeDestino’: ‘C++’},
  {‘ejercicio’: ‘localhost/duocode/rest/ejercicios/7’,
  ‘nombreLenguajeOrigen’: ‘Python’, ‘nombreLenguajeDestino’: ‘C++’}],
  ‘historialEjercicios’: [{‘idEnvio’: ‘1’,
  ‘ejercicio’: ‘localhost/duocode/rest/ejercicios/4’,
  ‘nombreLenguajeOrigen’: ‘Java’, ‘nombreLenguajeDestino’: ‘C++’,
  ‘codigo’: ‘código enviado’, ‘fecha’: ‘17/11/2014’, ‘puntuacion’: ‘2’},
  {‘idEnvio’: ‘2’, ‘ejercicio’: ‘localhost/duocode/rest/ejercicios/9’,
  ‘nombreLenguajeOrigen’: ‘Python’, ‘nombreLenguajeDestino’: ‘Perl’,
  ‘codigo’: ‘código enviado’, ‘fecha’: ‘18/11/2014’, ‘puntuacion’: ‘7’}],
  ‘candidatosPropuestos’: [‘localhost/duocode/rest/candidatos/18’,
  ‘localhost/duocode/rest/candidatos/23’]}
  ```

**REQ29 – DELETE**: Borra un usuario completamente. Enviamos por Header el ID del usuario, el token y el network, que nos sirven para comprobar que el usuario es administrador y tiene permisos para hacer esta operación:

- Header:

  ```
  {‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
  ```

  La respuesta que obtenemos está conformada por “error”. En caso afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente. El único caso de error posible es que el usuario no tenga los permisos necesarios.

- Response:

  ```
  {‘error’: ‘no’}
  ```

## A.14. localhost/duocode/rest/envios

En este apartado se explica cómo hacer un GET para obtener todos los envíos y un POST para crear un nuevo envío.

**REQ30 – GET**: Devuelve todos los envíos para que un administrador pueda tener información de ellos. Como solo puede tener acceso a esto el administrador, es necesaria la siguiente información:

- Header:

  ```
  {‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
  ```

  La respuesta se compone de usuarios con historiales de ejercicios.

- Response:

  ```
  {‘envios’: [{‘idUsuario’: ‘1’, ‘historialEjercicios’: [{‘idEnvio’: ‘1’,
  ‘ejercicio’: ‘localhost/duocode/rest/ejercicios/5’,
  ‘nombreLenguajeOrigen’: ‘Java’, ‘nombreLenguajeDestino’: ‘C++’,
  ‘codigo’: ‘código enviado’, ‘fecha’: ‘17/11/2014’, ‘puntuacion’: ‘2’},
  {‘idEnvio’: ‘2’, ‘ejercicio’: ‘localhost/duocode/rest/ejercicios/7’,
  ‘nombreLenguajeOrigen’: ‘Python’, ‘nombreLenguajeDestino’: ‘Perl’,
  ‘codigo’: ‘código enviado’, ‘fecha’: ‘18/11/2014’, ‘puntuacion’: ‘7’}]},
  {‘idUsuario’: ‘4’, ‘historialEjercicios’: [{‘idEnvio’: ‘1’,
  ‘ejercicio’: ‘localhost/duocode/rest/ejercicios/6’,
  ‘nombreLenguajeOrigen’: ‘Java’, ‘nombreLenguajeDestino’: ‘C++’,
  ‘codigo’: ‘código enviado’, ‘fecha’: ‘17/11/2014’, ‘puntuacion’: ‘2’},
  {‘idEnvio’: ‘2’, ‘ejercicio’: ‘localhost/duocode/rest/ejercicios/5’,
  ‘nombreLenguajeOrigen’: ‘Python’, ‘nombreLenguajeDestino’: ‘Perl’,
  ‘codigo’: ‘código enviado’, ‘fecha’: ‘18/11/2014’, ‘puntuacion’: ‘7’}]}]}
  ```

**REQ31 – PUT:** Corrige un ejercicio y también se comprueba si se ha completado la lección; en caso afirmativo se marca como completada en la base de datos. Por una parte enviamos la URL del ejercicio, el lenguaje del enunciado, el lenguaje de la solución y el código:

- Payload:

```
{‘ejercicio’: ‘localhost/duocode/rest/ejercicios/idEjercicio1’,
‘nombreLenguajeOrigen’: ‘Java’, ‘nombreLenguajeDestino’: ‘C++’,
‘codigo’: ‘código enviado en el lenguaje de destino’}
```

  Por otra parte enviamos el idUsuaro, el token y el network para comprobar que el usuario que envía el ejercicio para corregir es el que tiene la sesión iniciada.

- Header:

```
{‘idUsuario’: ‘2’, ‘token’: ‘token’, ‘network’: ‘network’}
```

  La respuesta que obtenemos está conformada por “error” y “puntuacion”. Si “error” tiene un valor afirmativo significa que algo ha fallado y no ha podido terminar la operación correctamente; en caso negativo todo ha ido correctamente y devuelve la puntuación que ha obtenido el ejercicio al ser corregido. El único caso de error posible es que el usuario no coincida.

- Response:

```
{‘error’: ‘no’, ‘puntuacion’: ‘2’}
```

# B. Manual de instalación

En este apartado se detallan los pasos a seguir para tener instalado **DuoCode** en un sistema *Linux*, con el fin de poder trabajar directamente con los ficheros fuentes y extender el proyecto. La dirección donde podemos descargar todo el código es `https://github.com/jucallej/DuoCode.git`.

El software necesario para trabajar con el proyecto se especifica a continuación.

- **Requisitos generales**
  - Java 7.
  - Git.
  - Navegador Web - Google Chrome.
- **Requisitos para el Servicio Web**
  - Xampp.
  - Tomcat 8.0.
  - NetBeans 8.0.
- **Requisitos para el Front-End**
  - Soporte SSL en Xampp y Tomcat.
- **Requisitos para la Aplicación móvil**
  - Node.Js 0.10.X.
  - Phonegap 5.0.0.
  - Eclipse Luna con plugin para Android.

## B.1. Instalación

Es importante seguir el orden de instalación que aparece a continuación para no tener problemas de dependencias entre programas.

- **Java**

  En primer lugar necesitamos tener instalada una versión de Java. La forma más sencilla es acceder a la web `http://www.java.com/es/` y descargar la última versión.

  Es posible usar el siguiente comando en el termina:

  ```
  $ sudo apt-get install openjdk-7-jdk openjdk-7-jre
  ```

- **Git**

  Para poder descargar todos los ficheros fuentes del proyecto es necesario tener un cliente Git instalado y configurado.

  Una opción es acceder a la web `http://git-scm.com/downloads/guis` y elegir el cliente que queramos.

  Es posible usar el siguiente comando en el termina:

  ```
  $ sudo apt-get install git
  ```

  Una vez instalado git podemos clonar el proyecto **DuoCode** y obtener una copia en nuestro sistema. Si trabajas con un cliente gráfico simplemente hay que pinchar en el botón *clone* y escribir la dirección del repositorio `https://github.com/jucallej/DuoCode.git`.

  También podemos clonar el proyecto directamente desde la línea de comandos:

  ```
  $ git clone https://github.com/jucallej/DuoCode.git
  ```

- **Google Chrome**

  Cualquier navegador es válido pero Chrome cuenta con un plugin (Advance Rest Client) que es necesario para hacer pruebas con el servicio REST. Para instalar el navegador accedemos a la web de Chrome `https://www.google.es/chrome/` y pinchamos en el botón de descarga.

  Es posible usar el siguiente comando en el termina:

  ```
  $ sudo apt-get install google-chrome-stable
  ```

- **XAMPP**

  Es necesario tener instalado un servidor local. XAMPP nos proporciona una base de datos MySQL y un servidor Apache. Para instalarlo solo hay que acceder a la web `https://www.apachefriends.org/index.html` y descargar la última versión disponible.

  Una vez instalado y funcionando accedemos a `http://localhost/phpmyadmin` para importar la Base de datos. Creamos la Base de datos nueva con el nombre 'Duocode', la seleccionamos e importamos el archivo 'Duocode.sql' para que nos cree las tablas y cargue la información del proyecto.

  En la carpeta *'htdocs'*, que se encuentra dentro de la carpeta del servidor XAMPP, es donde tenemos que poner la parte del front-end. Copiamos la carpeta *'duocode'* que contiene el *'index.html'* y todos los scripts y la pegamos en *'htdocs'*. Podemos probar el funcionamiento accediendo a la dirección `'http://localhost/duocode'`.

- **Tomcat**

  Para el servicio web REST necesitamos tener instalado Tomcat 8.0 o superior. Para ello, accedemos a la web `http://tomcat.apache.org/download-80.cgi` (para la versión 8.0), descargamos la última versión para nuestro sistema operativo y lo descomprimimos.

- **NetBeans**

  El IDE que se ha usado para desarrollar el proyecto es NetBeans 8.0. Nos permite integrar el sistema de control de versiones *Git* y los servidores. Se puede descargar desde la web `https://netbeans.org/downloads/`.

  Una vez instalado importamos el proyecto descargado desde el Git y seleccionamos el servidor con el que queremos que funcione, en nuestro caso será el Tomcat descargado anteriormente.

  Las bibliotecas necesarias se importan de manera automática una vez que hemos cargado el proyecto en NetBeans.

- **Certificados SSL**

  Para que tanto el front-end como el servicio web funcionen con *https* es necesario activar SSL en los servidores *Tomcat* y *Apache* del XAMPP.

  Hemos creado nuestros propios certificados SSL y se pueden encontrar en la carpeta *duocode/certificados*. Ñ̃a contraseña es *complutense*.

  Tomcat lo podemos configurar gracias al archivo *server.xml* encontrado en /apache-tomcat-8.0.21/conf/server.xml. Lo abrimos y dentro del elemento:

  ```
  < Service name = 'Catalina' >
  ```

  pegamos estas líneas de código (cambiando la ruta del proyecto):

  ```
  <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
                 maxThreads="150" SSLEnabled="true" scheme="https"
                 secure="true" clientAuth="false" sslProtocol="TLS"
             keystoreType="PKCS12"
             keystoreFile="{Ruta al repositorio}\DuoCode\duocode\certificados\
             mycert.p12" keystorePass="contraseña"/>
  ```

  Para configurar el server Apache que nos proporciona XAMPP primero tenemos que poner los archivos *'mars-server.crt'*, *'mars-server.key'* en la siguiente ruta:

  - *'/opt/lampp/etc/ssl.crt/'* los ficheros con la extensión .crt.
  - *'/opt/lampp/etc/ssl.key/'* el fichero con la extensión .key.

  Una vez tenemos los certificados y la clave en las rutas adecuadas editamos el fichero *httpd-ssl.conf*. Dependiendo de la versión puede tener un aspecto u otro, en nuestro caso hemos definido un nuevo VirtualHost con la siguiente configuración.

  ```
  <VirtualHost _default_:443>

  DocumentRoot "/opt/lampp/htdocs"
  ServerName localhost:443
  ServerAdmin you@example.com
  ErrorLog "/opt/lampp/logs/error_log"
  TransferLog "/opt/lampp/logs/access_log"

  SSLEngine on

  SSLCertificateFile "/opt/lampp/etc/ssl.crt/mars-server.crt"
  ```

```
SSLCertificateKeyFile "/opt/lampp/etc/ssl.key/mars-server.key"
SSLCertificateChainFile "/opt/lampp/etc/ssl.crt/my-ca.crt"

SSLCACertificateFile "/opt/lampp/etc/ssl.crt/my-ca.crt"

<FilesMatch "\.(cgi|shtml|phtml|php)$">
    SSLOptions +StdEnvVars
</FilesMatch>
<Directory "/opt/lampp/cgi-bin">
    SSLOptions +StdEnvVars
</Directory>

BrowserMatch "MSIE [2-5]" \
         nokeepalive ssl-unclean-shutdown \
         downgrade-1.0 force-response-1.0

CustomLog "/opt/lampp/logs/ssl_request_log" \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"

<Directory "/opt/lampp/htdocs">
        Options Indexes
        AllowOverride None
        Allow from from all
        Order allow,deny
</Directory>

</VirtualHost>
```

Una vez configurado podemos acceder a la dirección `https://localhost/duocode`, aunque nos saldrá un mensaje indicando que el certificado no está verificado (es un certificado que hemos creado nosotros) así que lo añadimos como excepción y ya tenemos **DuoCode** instalado.

- **Node.js**

  Para poder trabajar con PhoneGap - Cordova es necesario tener instalado Node.js en nuestro equipo. Podemos hacerlo descargándolo desde la web '`https://nodejs.org/`' o directamente desde un terminal:

```
$ sudo add-apt-repository ppa:chris-lea/node.js
$ sudo apt-get update
$ sudo apt-get install nodejs
```

  Es necesario que la versión sea 0.8+. Podemos comprobarlo tecleando desde el terminal:

```
$ node -v
```

- **PhoneGap**

  PhoneGap nos sirve para crear una app móvil desde el HTML5, CSS y JS de nuestro proyecto. Para instalarlo podemos descargarlo desde la web `http://phonegap.com` o directamente desde un terminal:

```
$ npm install -g phonegap
```

- **Eclipse y Android SDK**

  Gracias al plugin de Android podemos usar Eclipse como IDE para probar la app móvil de DuoCode. Instalamos la última versión de Eclipse descargándolo desde la web `http://eclipse.org`.

  Una vez descargado lo abrimos y vamos a añadir el plugin para Android. Pinchamos en **help - Install New Software** y añadimos la siguiente URL `https://dl-ssl.google.com/android/eclipse/` y seleccionamos **next** hasta completar la instalación.

  Reiniciamos Eclipse y tenemos que especificar la dirección del **Android SDK** que acabamos de descargar para que se actualice y tener Eclipse listo.

  Podemos importar la carpeta del proyecto que encontramos en DuoCode y probarlo con el emulador seleccionando la carpeta del proyecto y pulsando sobre *'Run'*.

## B.2. Desplegado

Finalmente, una vez instalado y configurado todo, podemos desplegar y trabajar sobre el proyecto local.

Tenemos que seguir los pasos que se detallan a continuación:

- **Arranque de XAMPP**

  Para arrancar el servidor Apache y tener acceso a la Base de datos es necesario que *XAMPP* esté funcionando. Para ello abrimos un terminal, nos posicionamos en la carpeta donde se haya instalado *'/opt/lampp/'* y escribimos el siguiente comando:

  ```
  $ ./xampp start
  ```

  Es posible que se quede parado mientras intenta arrancar el servidor Apache porque necesite la contraseña del certificado. Si esto ocurre escribimos en el terminal *'complutense'* y pulsamos Enter.

- **Despliegue del servicio REST**

  Para desplegar el servicio REST abrimos Netbeans y seleccionamos el proyecto DuoCode que importamos durante la instalación. Accedemos a *Run - Build* y después pulsamos con el botón derecho del ratón sobre la carpeta del proyecto y seleccionamos *Deploy*.

  Con estos pasos conseguimos que el servidor Tomcat se inicie y se despliegue el servicio REST.

- **Acceso a DuoCode**

  Si no se ha producido ningún problema durante la instalación y el despliegue podemos acceder a la web `https://localhost/duocode` y probar la aplicación.