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

Índice

1. Introducción

Este proyecto empezó con el objetivo de permitir 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. Todo ello sin olvidarnos de los valiosos conocimientos ya afianzados que seguro serán de utilidad. Nos planteamos realizar una *front-end* para un proyecto de aprendizaje colaborativo de lenguajes de programación en base a otros que ya sabes, llamado DuoCode. A continuación pasaremos a explicar el estado actual del sector, luego detallaremos las tecnologías que nos han influenciado y usaremos, y finalmente definiremos los objetivos del proyecto.

1.1. Revisión del 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 [**?**], una enciclopedia libre y editada colaborativamente en múltiples idiomas. Ante dudas concretas hay sitios de preguntas y respuestas como Stack Overflow [**?**], donde programadores se ayudan mutuamente en base a cuestiones concretas. Asimismo muchas universidades usan plataformas digitales como Moodle [**?**], que permiten publicar apuntes de las diferentes asignaturas. También existen cursos online, los llamados MOOC (acrónimo en inglés de Massive Open Online Course) donde puedes aprender a distancia materias. Estos requieren que se suba material como vídeos con explicaciones detalladas. Suelen estar enfocados a la obtención de algún tipo de diploma o certificación. Requieren un esfuerzo constante y duradero por parte del profesorado, que tiene que actualizar el material académico disponible y corregir los distintos trabajos de los alumnos.

Otra manera de enfocar la enseñanza, sobre todo cuando quieres ampliar algo sobre la que ya tienes una base, es a base 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 [**?**]. 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.

Saliéndonos del mundo de la programación, hay otros ejemplos, como Duolingo [**?**] 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 usuarios 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.
- Guarda información sobre los fallos del usuario para intentar repetir esas preguntas y que aprenda los concepto de manera definitiva.
- Incluye prácticas con tiempo y la posibilidad de certificar el nivel del idioma mediante tests online.

Hay otras alternativas como Bussu [**?**], 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 [**?**]. Este plan de estudios sigues las pautas del Acuerdo del Consejo de Universidades publicado en el B.O.E [**?**].

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 haber sido 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 consumirlas desde las aplicaciones Su uso permite establecer interconexiones (relaciones) entre los datos (guardados en las distintas tablas), y tiene como principal ventajas 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 el curso de realización de la asignatura en 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 que hemos dado a este enfoque 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 es un acrónimo en inglés de Simple Object Access Protocol y es un protocolo que define como 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. Usa XML un lenguaje de marcado que ha perdido fuelle frente al más ligero JSON. Además no todos los lenguajes de programación ofrecen facilidades para el uso de este tipo de servicio Web.
- REST 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 recurso. 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. Además permite ofrecer los datos en JSON. Por esto 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 un proyecto colaborativo para aprender lenguajes de programación, similar a Duolingo. Como iba a estar 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:

- Poder 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 tendrá una serie de vidas para completarlos. Esto favorecerá que el usuario se fidelice con la aplicación.
- Cuando el usuario falle un ejercicio pero no esté de acuerdo, podrá proponerlo como candidato a válido. Con la ayuda de la comunidad y de moderadores, estos se podrán incorporar como soluciones en un lenguaje de programación determinado.
- Se podrá guardar ejercicios favoritos para poder consultarlos de nuevo.

- Habrá una interacción con las redes sociales. Se podrá hacer login con alguna de ellas y compartir tus resultados en esta.

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

- Desarollar un sistema Web RESTful para el aprendizaje de la programación. Para ello nos planteamos los siguientes pasos:
  - Especificar los distintos recursos REST, y que 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 AJAX (acrónimo en inglés de Asynchronous JavaScript And XML).
  - 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.

2. 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* (Remote Procedure Call), *SOAP* (Simple Object Access Protocol) y *RESTful* (Representational State Transfer).

2.1. 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*

2.2. Recursos

Uno de los conceptos más importantes de una API *REST* 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 teníamos clara la arquitectura de nuestro servicio web, definimos todos los *recursos* que eran necesarios para desarrollar *DuoCode.*Aunque hay una sección 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 como correctos. 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 envio 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 que utilizaríamos y comenzamos a pensar qué tecnologías íbamos a usar para la implementación del servicio web.

2.3. Implementación

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

2.3.1. 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, se usará de ejemplo la clase **EjerciciosResource**:

```
@Path("ejercicios")
public class 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.

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

- *@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*.

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

- *@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 newEjer tomaría el valor del parámetro tk, perteneciente a la cabecera de la petición HTTP.

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

- *@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.

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

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

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

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

2.3.2. 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`

2.3.3. 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 'select by id' 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 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.

3. Front-end

En esta sección describiremos como 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.

3.1. Protitipo

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 CSS también llamadas hojas de estilo, 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 [**?**] 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 de 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* [**?**], 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 [**?**], un librería 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 que lenguaje está escrito el código, pero bajo nuestras pruebas resulta suficiente con la detección automática que tiene la librería. 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). Hicimos 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 (el nuestro y otros de librerías como Bootstrap) 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.

3.2. Conexión con el servicio REST

Para conectar la interfaz visual con el servicio Web, como hemos comentado en la introducción realizamos llamadas asíncronas. 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 como realizarlo:

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

- Otra manera de realizar esto es usando una librería como jQuery [**?**], 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 se crece de tamaño. Por ello tampoco nos pareció oportuno usarlo para nuestro proyecto.

- Hay *frameworks* de JavaScript, que quieren simplificar el desarrollo de aplicaciones Web y para ello usan el patrón de modelo-vista-controlador en el lado del cliente. Uno de los más probado, con una amplia documentación, y un gran respaldo de la comunidad es AngularJS [**?**]. 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 que no esté todo mezclado e ilegible, y permite extender a HTML, para crear componentes reutilizables. Además permite consumir servicios Web REST de manera muy fácil.

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*.

A continuación explicaremos en detalle varias de las características de AngularJS y como las hemos usado en nuestro proyecto. Para empezar, este *framework* 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, logo, 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, y tenerlo separado 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 donde 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 si 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', que 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 como hemos conseguido que los distintos controladores compartan datos entre ellos. Para esto usamos lo que AngularJS llama 'Providers' [**?**], que permite crear servicios y objetos especiales, que puedes añadir a tus controladores. Estos solo se cargan la primera vez, y las siguientes se devuelve 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*.

3.3. Integración con redes sociales

Una vez conectados con los datos reales, nos dispusimos a permitir que los distintos usuarios pudieran ver cierto contenido sin estar *logueados* 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 [...]ucm.es, y esta está gestionada por Google, de manera que todas las personas que van a corregir este trabajo pueden acceder y probarlo. En segundo lugar queríamos que pudiera ser accesible usando una cuenta de Facebook [**?**], pues es una de las redes sociales más populares en España y en el mundo.

Cada red social pone a disposición de los desabolladores librerías 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 librerías 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 librerías como la que usamos, HelloJS [**?**], que permiten obtener los datos para autenticar a un usuario de una manera sencilla, y funciona para multitud de redes sociales. Integramos esto con la aplicación que ya habíamos creado con AngularJS, y hicimos que si estás *logueado* 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 real, y obtener ciertos datos básicos del usuario, como su nombre, o un link 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 *logueado* con Facebook. Para ello usamos la librería que ponen a disposición para tales fines [**?**].

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.

4. 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.

4.1. 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

4.2. Base de datos

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

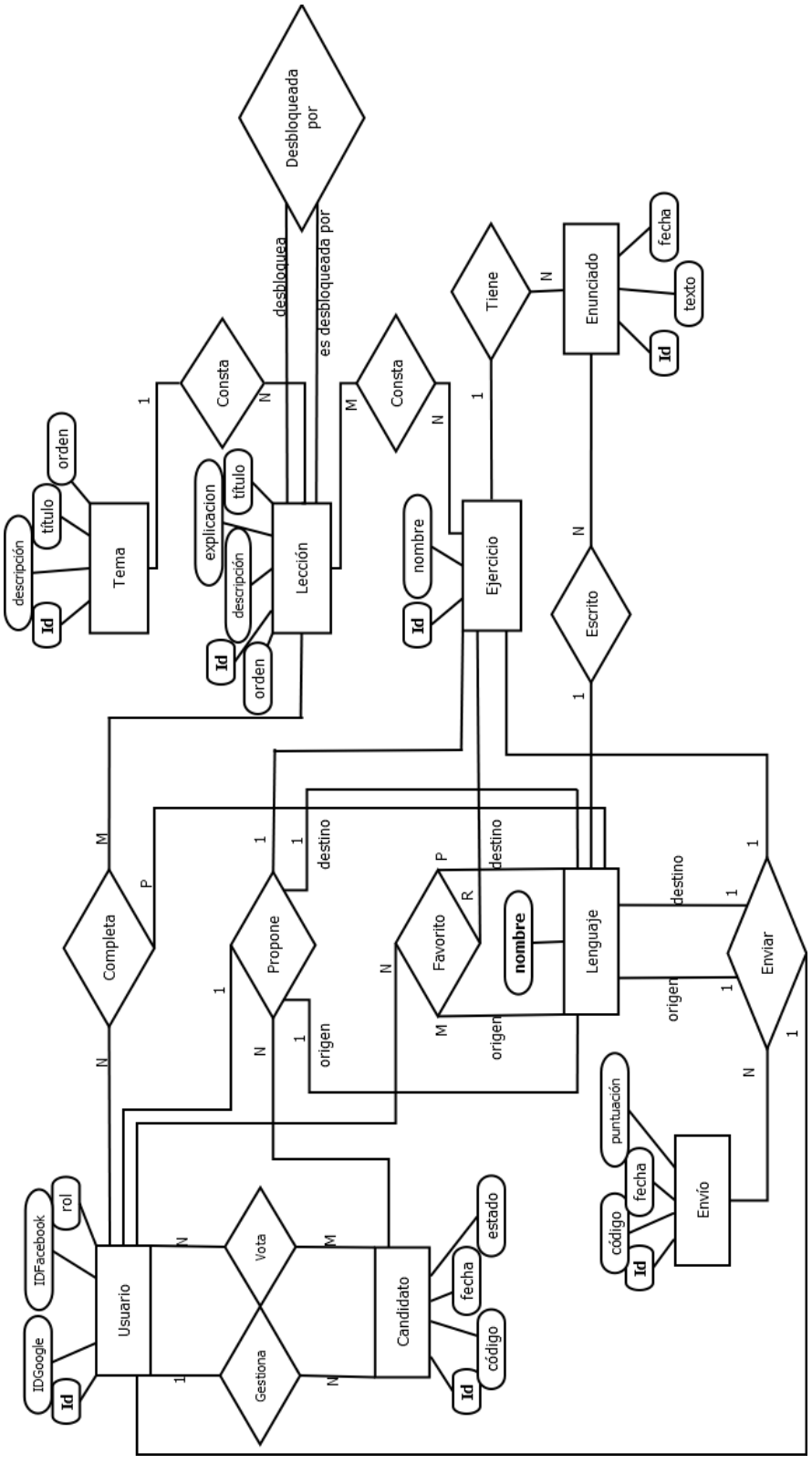

Figura 1: Modelo entidad-relacion

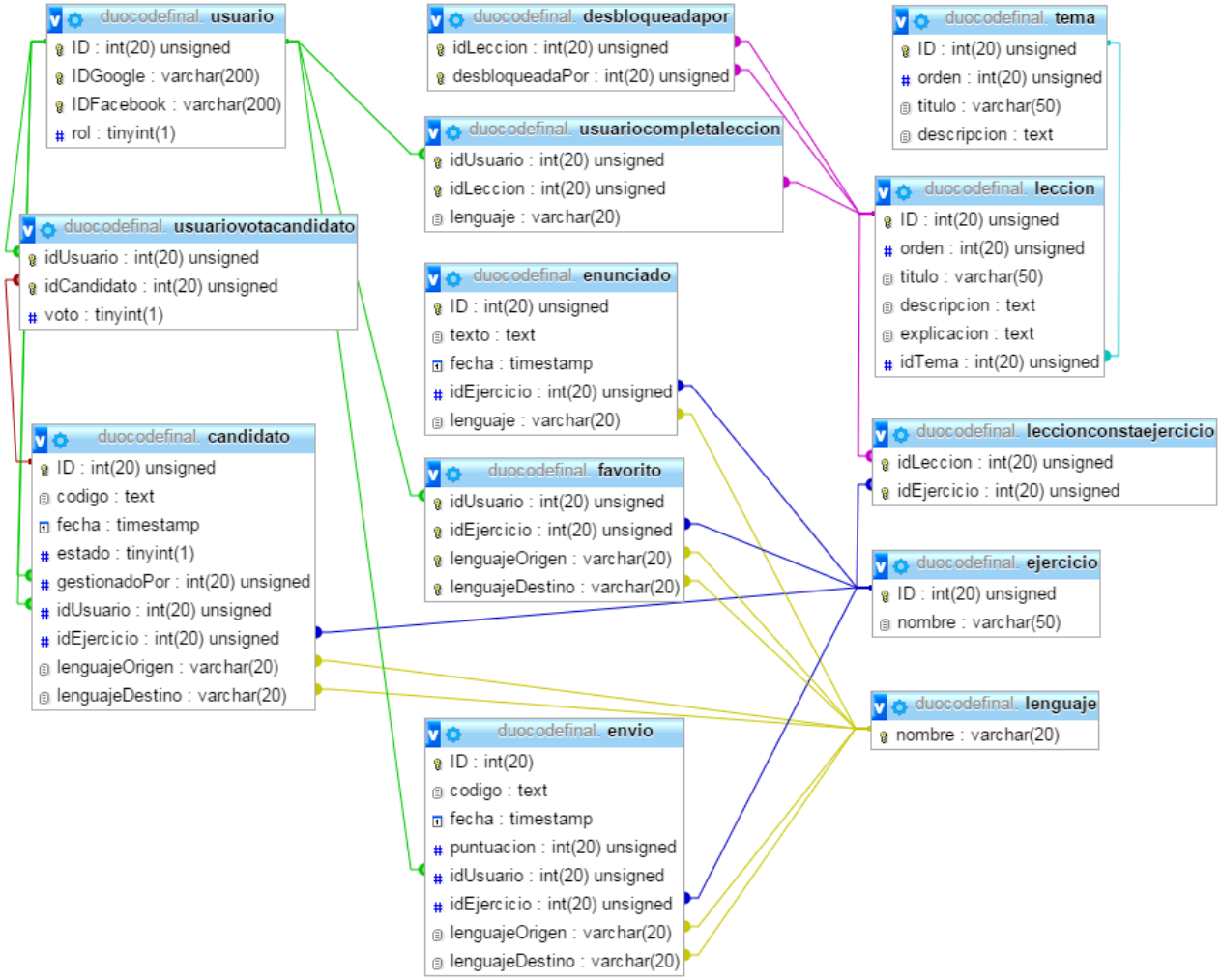

Figura 2: Modelo relacional

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, primera división 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, segunda división 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.
Lenguaje	Lenguaje en el que se ha completado dicha lección, ya que una lección puede ser completada por el mismo usuario en distintos lenguajes.

- 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.

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. *traducido*
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. *traducido*
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.

Referencias

[1] N. M. Oliet, Y. O. Mallén, and A. V. López. *Estructuras de datos y métodos algorítmicos.* Prentice Practica. Prentice Hall, 2004.

A. Especificación de requisitos

En esta sección se exponen todos los requisitos existentes 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 en específica, un PUT para modificarla y un DELETE para eliminarla. En estos requisitos obtenemos el ID del candidato 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 de desarrollador

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 una 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*. Ña 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" keyst
  ```

  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 modo desarrollador

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

Tenemos que seguir una serie de 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.

C. Manual de usuario

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

C.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.

Figura 3: Lenguajes

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

C.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.

Figura 4: Temas

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

Figura 5: Inicio de sesión con Google

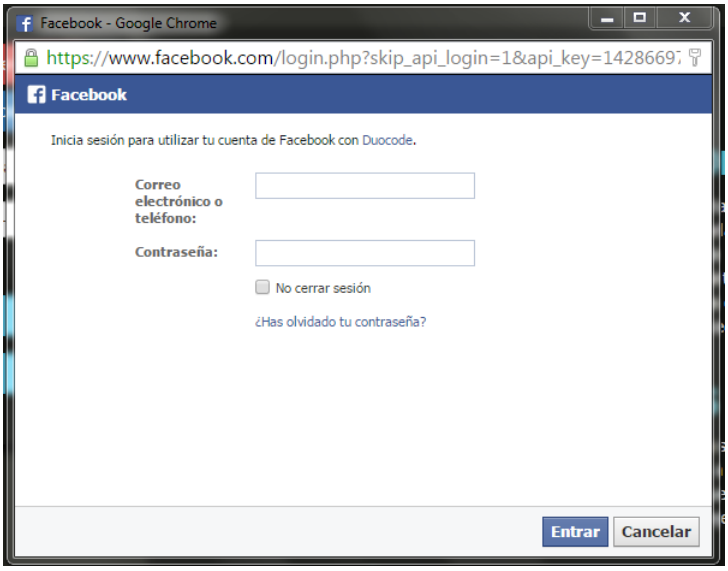

Figura 6: Inicio de sesión con Facebook

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.

Figura 7: Información de usuario

Una vez seleccionado un tema, pasamos a las lecciones.

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

Figura 8: Lecciones

Una vez seleccionada una lección, pasamos a los ejercicios.

C.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 ésta.

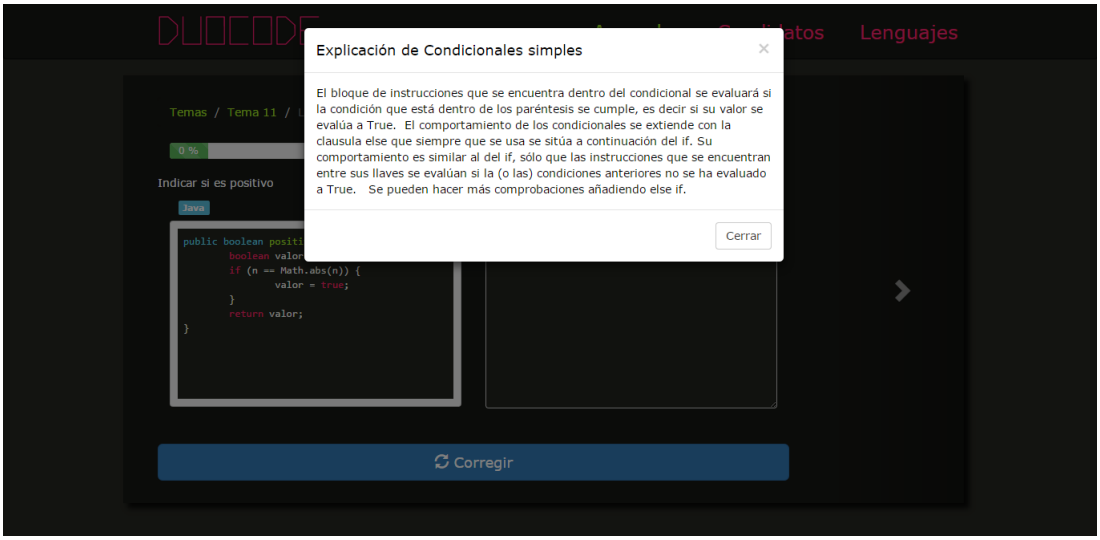

Figura 9: Pop-up al iniciar una lección

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.

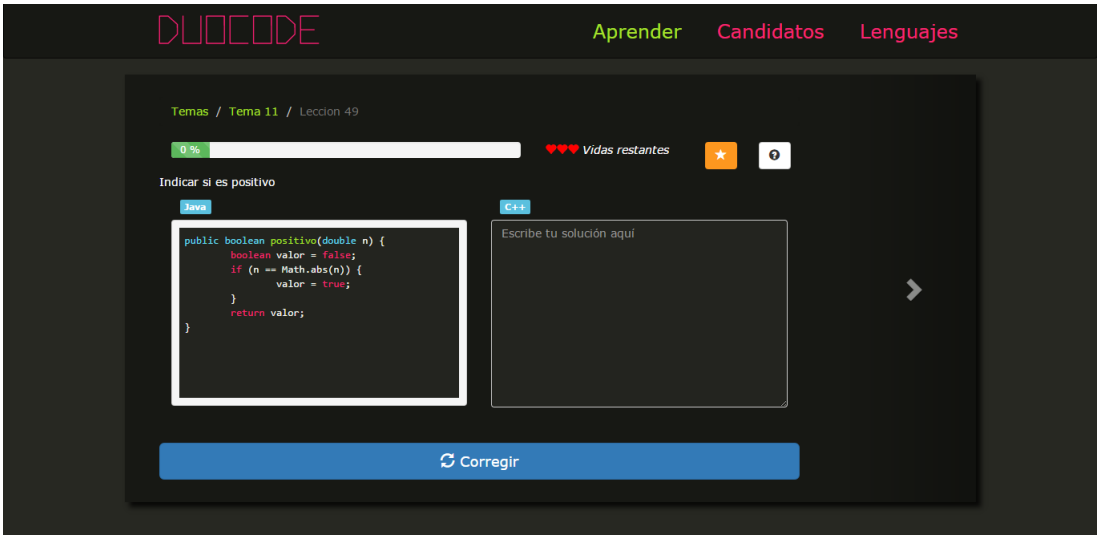

Figura 10: Ejercicio

Después de corregir el ejercicio hay dos posibilidades:

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

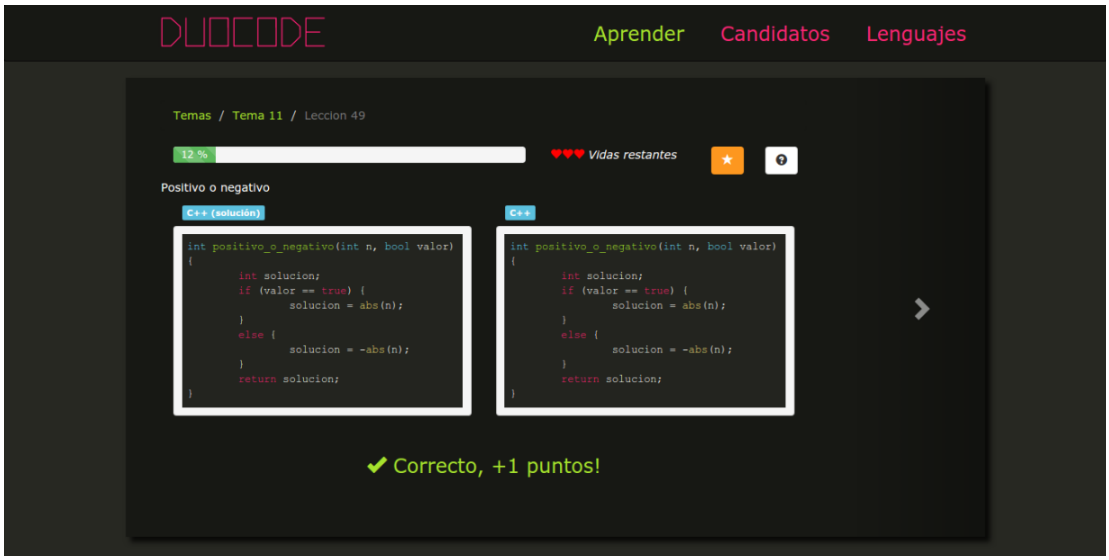

Figura 11: Ejercicio correcto

- 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 y si el usuario no sabe qué son los candidatos se encuentra también un botón con un pop-up explicativo.

Figura 12: Información de usuario

Figura 13: Información de usuario

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.

Figura 14: Información de usuario

- 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.

Figura 15: Información de usuario

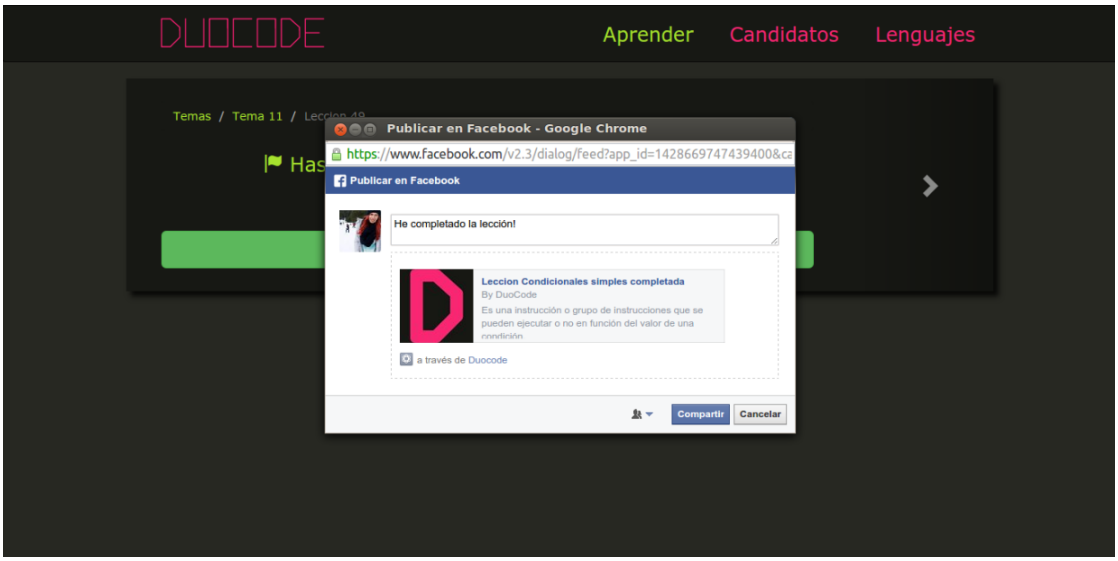

Figura 16: Información de usuario

C.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.

Figura 17: Información de usuario

C.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.

Figura 18: Información de usuario

C.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.

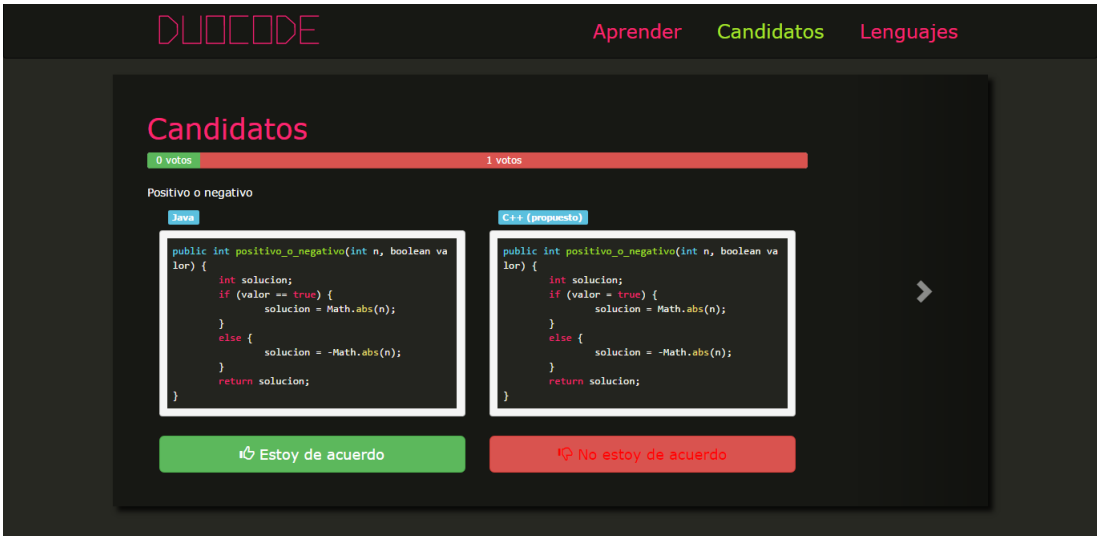

Figura 19: Información de usuario

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 encontraría en la lista de candidatos para evaluar.

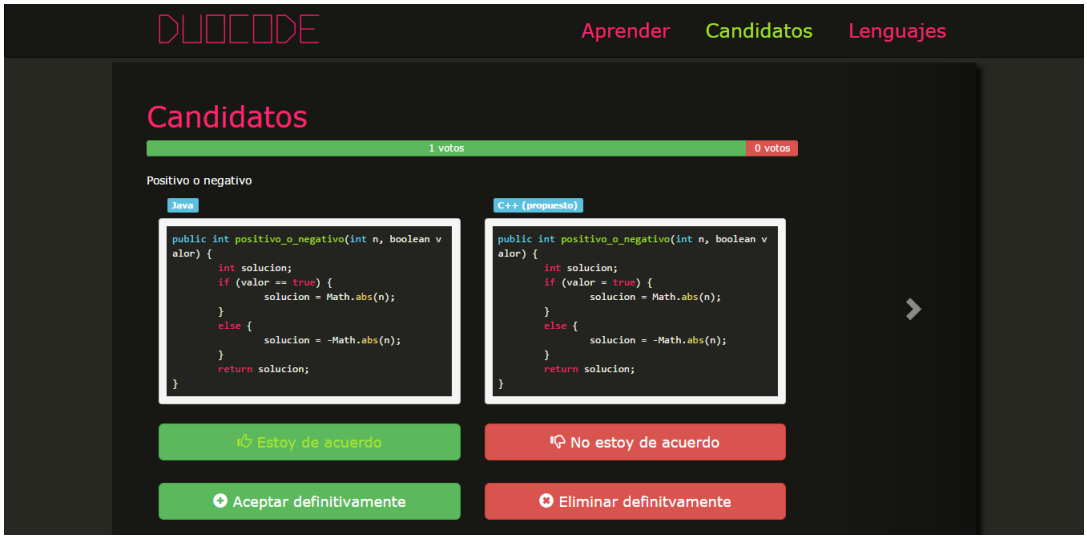

Figura 20: Información de usuario

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.

Figura 21: Información de usuario