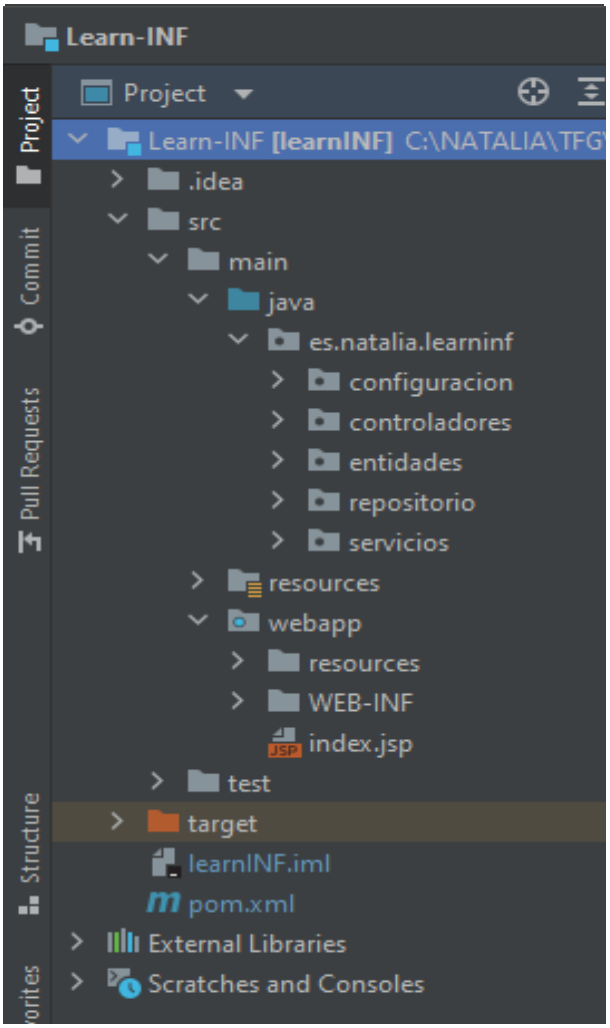
1.4. Parte experimental

1.4.3. Implementación y pruebas

- **Desarrollo de la aplicación con los lenguajes y programas elegidos**

En el anterior punto hemos visto las distintas aplicaciones y lenguajes elegidos y donde han sido implementados estos en la estructura del proyecto. Por ello, vamos a pasar directamente a la explicación del código fuente en sí.



La estructura del proyecto completo sería la siguiente:

El código estará alojado en la carpeta **src**, dentro de esta en **main** se encuentra la carpeta **java, resources** y **webapp** donde está toda la estructura Java y los archivos que contienen las vistas jsp, estilos y demás códigos de la aplicación. En la carpeta **test** encontraremos las pruebas unitarias realizadas con JUnit.

Dentro de la carpeta **target** generada con cada despliegue que hacemos de la aplicación encontraremos entre otros archivos el **.war** del proyecto.

También encontraremos el archivo **pom.xml**, donde definimos las los datos del artefacto, sus propiedades y dependencias necesarias para usar ciertas funciones en nuestro proyecto.

➔ **Carpeta java:** Esta carpeta recolecta todos los archivos con extensión Java del proyecto. La estructura dentro de esta es recogida a su vez por un package, y dentro de este se divide en varias carpetas.

La carpeta de **configuración** simplemente especifica la ruta inicial y la extensión de los archivos de las vistas de la aplicación, definiéndolo como archivos .jsp haciendo que más adelante no tengamos que especificarlo.

La carpeta de **entidades** es donde se definen las tablas de la base de datos. Cada una cuenta con lo que serían los atributos de la entidad, su tipo de dato, las relaciones que guardan con

otras entidades y métodos específicos de constructores, getters y setters. Gracias a la extensión de JPA definir estas relaciones ha sido más rápido y sencillo.
Las carpeta de **repositorios** define interfaces java para usar la funcionalidad del módulo DeltaSpike, la cual ayuda con lógicas ya predefinidas básicas para las consultas a la base de datos. Para usar esta necesitamos la clase java de Entity Manager, todas las interfaces java tendrán la relación extend con esta clase para usar sus funcionalidades. Son consultas simples, como búsquedas por ID.
La carpeta de **servicios** recoge los métodos que realizan consultas, eliminaciones, creaciones o modificaciones a la BD, y se pueden relacionar con repositorios. También tiene consultas predefinidas gracias a DeltaSpike, como los findAll o findBy.
En la carpeta de **controladores** encontramos las funciones de enrutado, que relaciona las vistas con los servicios y proporciona estos al usuario. Por lo general suele haber un controlador por vista o conjunto de vistas relacionadas. Dependiendo la función que queramos realizar podremos obtener datos para mostrar en las vistas o recoger datos de formularios para trabajar con ellos gracias a los servicios que hayamos creado.

➔ **Carpeta resources:** En esta encontraremos archivos necesarios para el uso de ciertas dependencias o servicios como DeltaSpike o las funcionalidades de hibernate.

➔ **Carpeta webapp:** En esta guardaremos todas las vistas, hojas de estilo CSS, archivos JavaScript, imágenes y demás fuentes necesarias para la visualización de la aplicación. En este caso, las vistas están realizadas con extensión .jsp para la visualización de datos que se pasarán mediante los modelos del controlador.

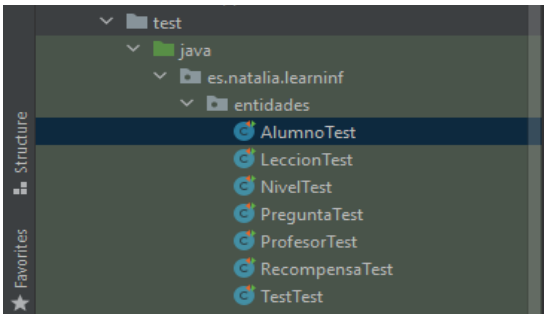
➔ **Carpeta test:** Aquí se guardan los test realizados mediante clases Java y el framework de testing JUnit.

El código que guardan estas carpetas está por completo comentado en cada método y aplicación de los servicios en los controladores, como están las entidades formadas, etc. Por ello, creo que es mucho más sencillo entenderlo mediante las explicaciones que hay en estos archivos, pero de una manera breve explicaré el flujo que puede llevar la aplicación en la situación de crear una cuenta de usuario básica y lo que se puede encontrar hasta llegar al final de un test.
Siendo un nuevo usuario tendremos que crearnos una cuenta en la plataforma. Rellenando el formulario, el cual tiene validación mediantes las funcionalidades de HTML5, haríamos un POST desde el controlador para la creación de la cuenta y también una sesión para guardar los datos del usuario. Una vez creada, nos llevaría a la vista de perfil, donde podremos cerrar

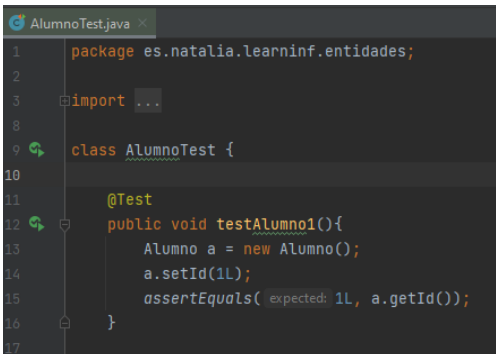
sesión cuando queramos (destruimos la sesión del controlador) e ir a mundos donde empezaremos a navegar por los distintos menús hasta la lección.
Todos los menús están controlados. Primero de que haya una sesión activa, y segundo que los puntos que tiene el usuario en su cuenta sean los necesarios para su paso a los distintos niveles y mundos. Todos los datos mostrados por pantalla serán resultado de consultas de los servicios y métodos GET para mostrar los datos a través de los modelos.
Al llegar al test se recogerán las opciones seleccionadas a través de Javascript y Ajax, para el conteo de aciertos. Dependiendo del resultado se mostrará un resultado u otro en la pantalla final, otorgando una recompensa y desbloqueando nuevos niveles.
Por último si en caso de no ser un alumno, si no que quisiéramos ingresar como profesor el login sería similar, solo que para acceder a estos perfiles sólo disponemos de la opción por correo. Al ser profesor, el perfil de este será distinto ya que nos mostrará, aparte de su información personal, un listado de alumnos que están asociados a este a través de su código. El profesor tiene acceso a todos los recursos de la web. Para controlar que no haya fallos, la única opción deshabilitada es la de mandar el test para obtener los puntos. Los profesores no pueden darse de alta en la plataforma, ya que de manera realista sería muy fácil para un alumno tener información que no les correspondiera. En caso de querer dar de alta a un profesor nuevo tendría que ser de manera interna en la base de datos.
Algunas normas a destacar serían, por ejemplo, el no poder repetir un test que ha sido superado con éxito, pero si se puede visualizar la lección. Las recompensas se pueden visualizar en otra vista de la aplicación, si pasas el ratón por encima de ellas te dirán dónde las has obtenido. También se comprueba siempre que haya una sesión activa siempre que sea necesaria para evitar excepciones por datos nulos, y a su vez se controlará que tipo de sesión es, si es una sesión de alumno o de profesor para tener accesos y perfiles distintos.

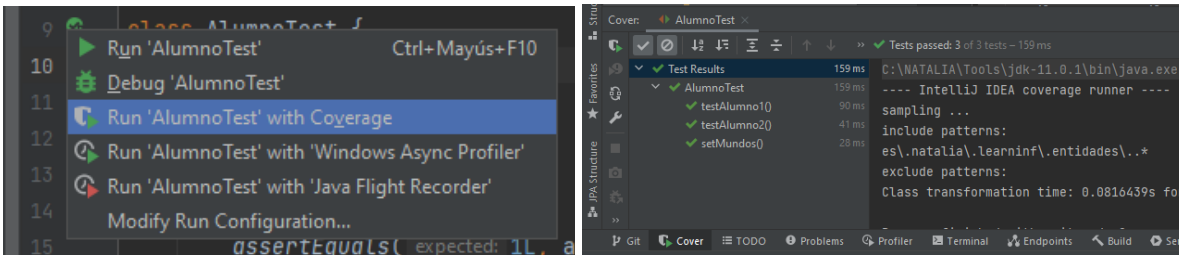
- **Documentación las pruebas realizadas**
    - **Pruebas unitarias de las partes más complicadas del proyecto**

Para realizar las pruebas de testeo de la aplicación utilicé JUnit5. Se trata de un framework para la automatización de las pruebas unitarias o de integración en proyectos.
Gracias a estas bibliotecas con sus anotaciones y métodos nos permite realizar pruebas de integración y asegurarnos que los métodos más importantes que usamos en el proyecto son eficaces y no dan lugar a fallos a la hora de usarlos en la aplicación desplegada.



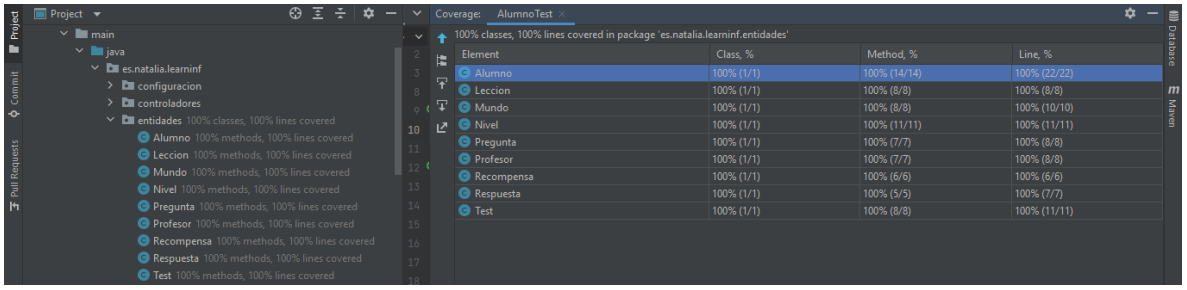
Estas pruebas deben estar guardadas en la carpeta de **test - java** dentro del código fuente de la aplicación, y deberá tener la misma estructura de carpetas que las clases que queramos testear. Una vez tengamos estas, es importante que las clases tengan la anotación de @Test en ellas, ya que con esto seremos capaces de arrancar las pruebas por cada método que queramos comprobar. Cuando queramos comprobar si los test realizados son correctos clickeamos en los iconos que se crearán en estas clases, y seleccionaremos Run with Coverage para comprobar qué porcentaje del código ha sido testeado con éxito.





Muestra de funcionamiento de testeo

Para este apartado he realizado las pruebas de las clases de **entidades**, ya que al ser la raíz de la creación de los posteriores métodos era importante comprobar que los métodos de creación de estos elementos, setter y getters funcionan apropiadamente y de la manera esperada.



Vista del testing 100% completado en la carpeta entidades

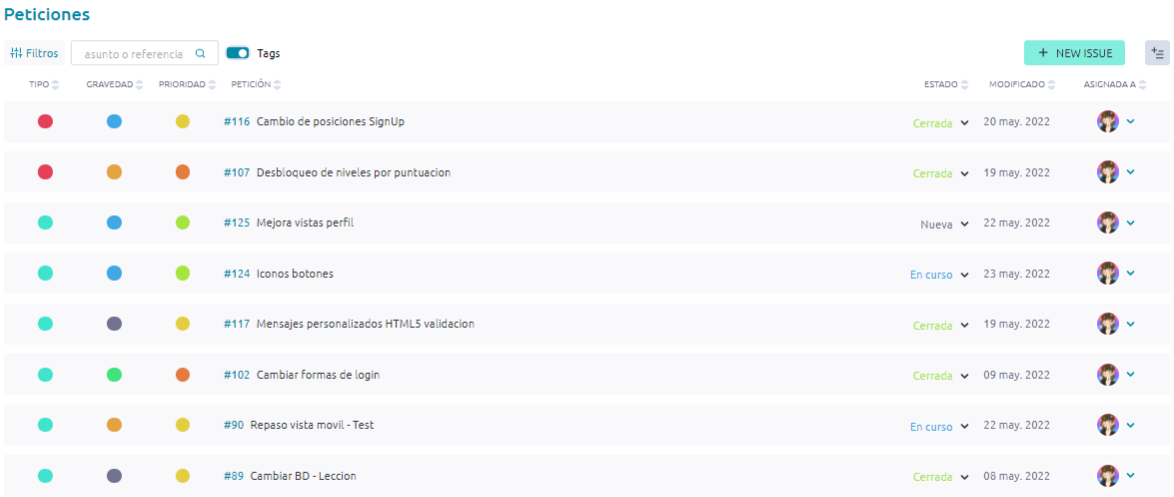
- **Pruebas de integración: Aceptación en las historias de usuario**

Para el seguimiento de las historias de usuario, peticiones y plazos de desarrollo se ha utilizado la plataforma de Taiga.io incorporando todas las necesidades que han surgido a lo largo de la construcción de la aplicación.

En total han sido 6 épicas que dividían y clasificaban las historias de usuario, a grandes rasgos, entre diseño UX, FrontEnd y BackEnd. A la vez que se iban especificando estas

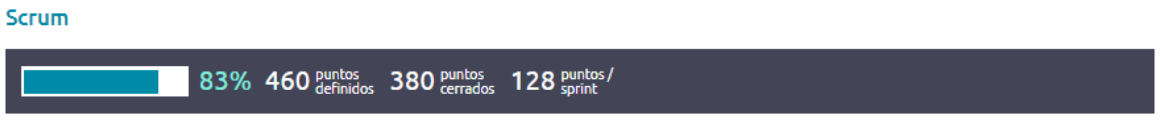
historias y sus tareas también se incorporaron peticiones, la mayoría de mejoras para la aplicación sin ser en exceso prioritarias.

Vista del panel de Épicas en Taiga.io



Vista del panel de Peticiones en Taiga.io. Las de tipo BUG son rojas y las de tipo Mejora son celestes

A cada tarea declarada se le asignaba determinados puntos correspondientes a la parte de la aplicación que tocaban. En total se han definido un total de 460 puntos.



En general la mayoría de las tareas han sido realizadas, algunas han tenido que traspasarse de unos sprints a otros y otras se han vuelto a incorporar pero en forma de petición al ser problemas menores.