UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Visor 3D 2.0

Presentado por Jose Manuel Moral Garrido
en Universidad de Burgos — 30 de enero de 2018
Tutores: José Francisco Díez Pastor, Álvar Arnaiz González

Índice general

Índice de figuras

Índice de tablas

Apéndice A

Plan de Proyecto Software

A.1. Introducción

Cabe mencionar que no he realizado una dedicación a tiempo completo al trabajo de final de grado, sino que ha sido una dedicación parcial. Esto es debido a que me encuentro trabajando al mismo tiempo, por lo tanto, en la mayoría de los *sprint* se han cerrado la mayor parte de las tareas el mismo día ya que mi horario laboral cambia cada semana. Por lo tanto, el día o los días que tengo libres los aprovecho al completo avanzando en el proyecto.

A.2. Planificación temporal

A continuación, detallaremos los diferentes objetivos que se han establecido para cada *sprint* y, a su vez, el progreso obtenido en los mismos. Para ello hemos utilizado una metodología ágil denominada *SCRUM* [3]. Emplearemos a su vez el gestor de tareas provisto por GitHub y generaremos gráficos *burndown* para el seguimiento de los *sprint*, los cuales son provistos por la extensión *ZenHub*.

Sprint 1 - 18–9-2017/24-09-2017

En este *sprint* se pretenden instalar las herramientas necesarias para la realización del proyecto, así como la lectura y comprensión de la *memoria* y *anexos* del proyecto de partida [4]. A su vez, se pretende probar los métodos creados de la aplicación de partida para comprobar su correcto funcionamiento.

Podemos observar el progreso del *sprint* en la figura A.1

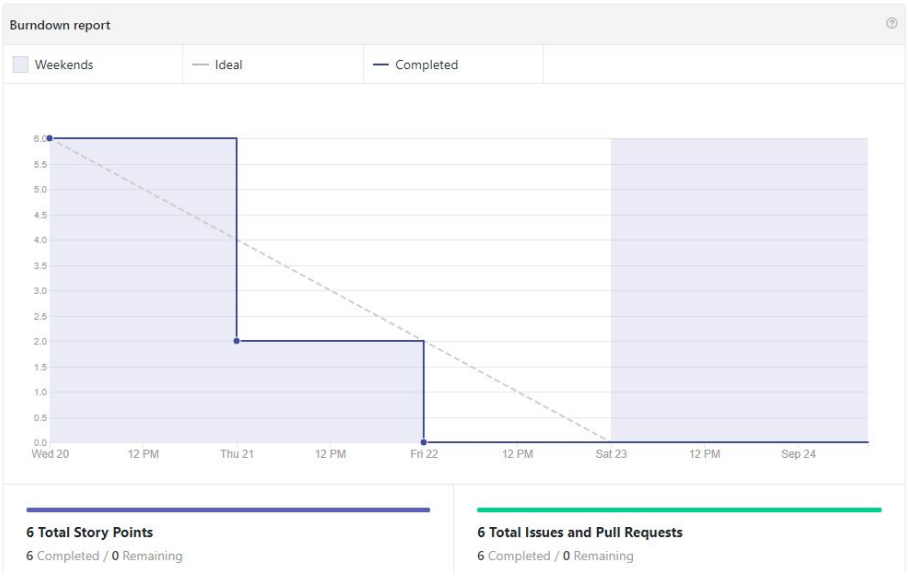

Figura A.1: Progreso en el *sprint* 1.

Sprint 2 - 25-09-2017/01-10-2017

En este *sprint* se pretende subir a mi repositorio propio toda la información necesaria para continuar el proyecto y solucionar los fallos encontrados la semana anterior en los distintos métodos de la aplicación. Además se prentende conocer la estructura completa del proyecto con el fin de agilizar las tareas en el momento de la búsqueda de los puntos de la aplicación a corregir o modificar.

Podemos observar el progreso del *sprint* en la figura A.2

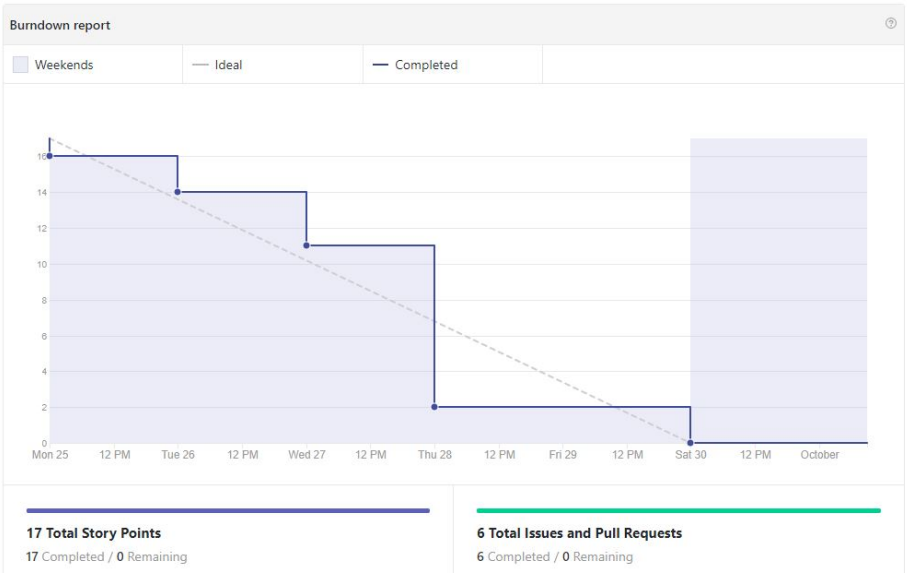

Figura A.2: Progreso en el *sprint* 2.

Sprint 3 - 02-10-2017/08-10-2017

En este *sprint* se pretende cambiar la manera que tiene la aplicación de cotejar los roles de los usuarios (mediante base de datos) a ser cotejados y asignados mediante la *API* de UBUVirtual e intentar interar la aplicación en un servidor público como es *Heroku* [5].

Podemos observar el progreso del *sprint* en la figura A.3

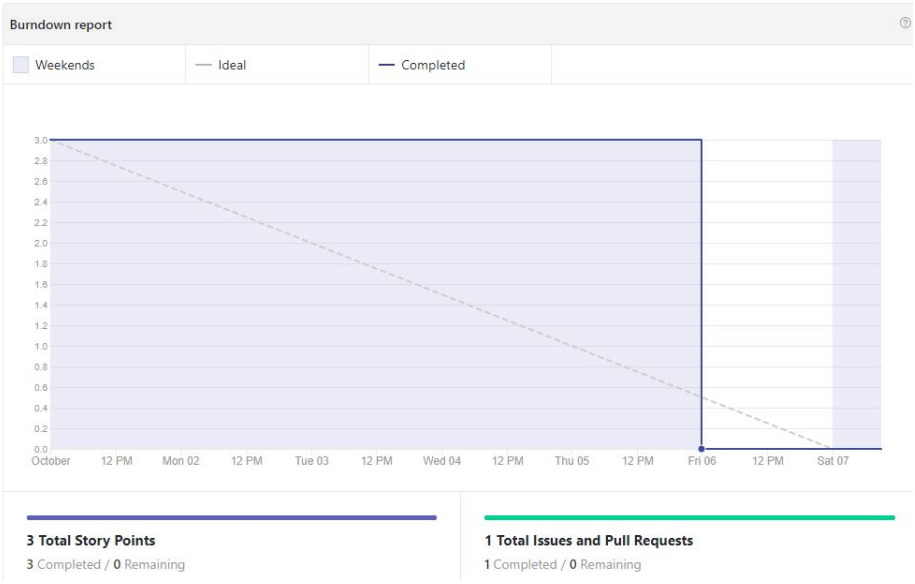

Figura A.3: Progreso en el *sprint* 3.

Sprint 4 - 09-10-2017/15-10-2017

En este *sprint* se pretende continuar con el intento de integración de la aplicación en *Heroku*. A su vez, se pretende albergar los modelos de manera privada para que los alumnos no puedan acceder a ellos nada más que mediante la plataforma. También instalaremos *Moodle* de manera local para poder realizar pruebas sin depender de un tutor que pueda facilitarnos asignaturas, asignación de roles, subida de recursos, etc.

Podemos observar el progreso del *sprint* en la figura A.4

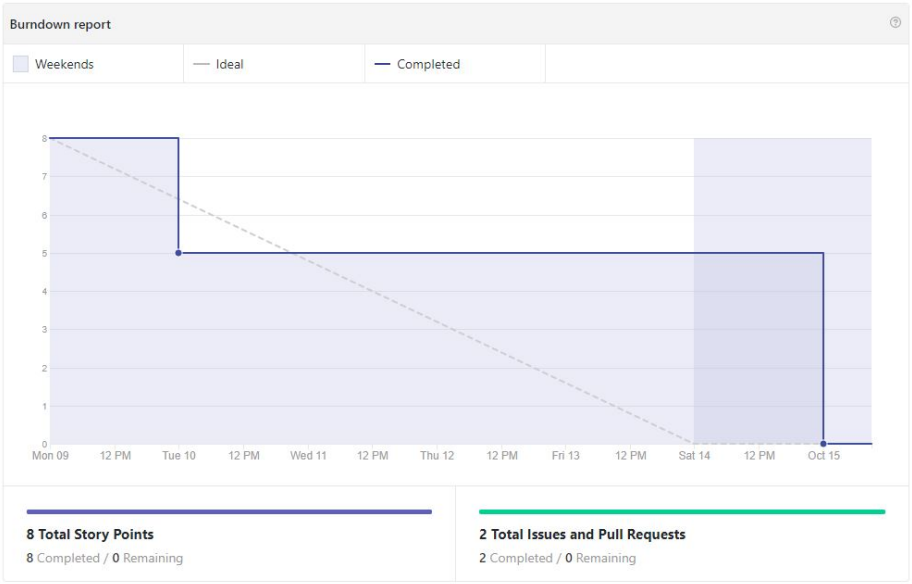

Figura A.4: Progreso en el *sprint* 4.

Sprint 5 - 16-10-2017/22-10-2017

En este *sprint* se pretende instalar de nuevo *Moodle*, ya que el instalado en el *sprint* anterior no funcionaba correctamente. A su vez continuamos con la integración de la aplicación en *Heroku* (tarea que se está alargando por dos motivos: errores en la estructura de la aplicación y que nos encontramos a la espera de que la UBU nos proporcione un servidor que cumpla ciertos requisitos).

Podemos observar el progreso del *sprint* en la figura A.5

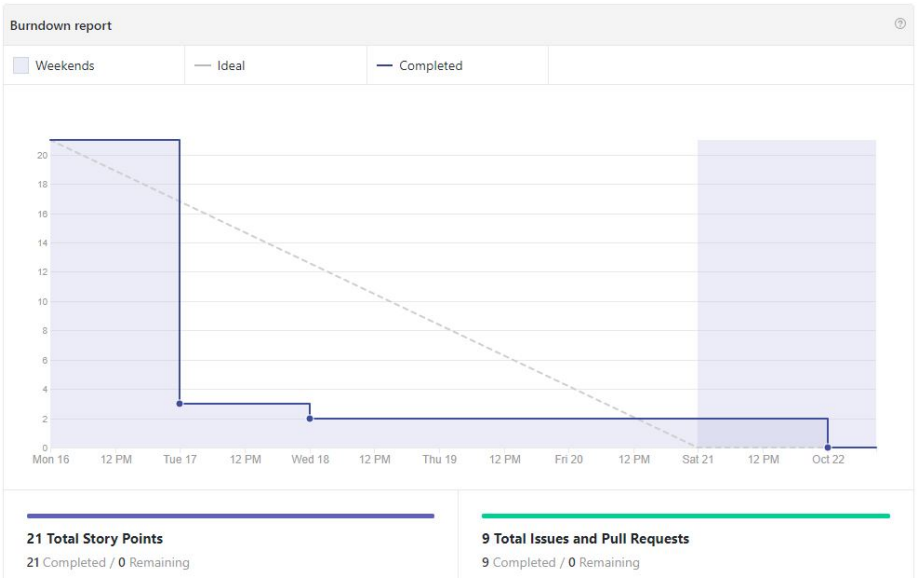

Figura A.5: Progreso en el *sprint* 5.

Sprint 6 - 24-10-2017/29-10-2017

En este *sprint* se pretende revertir la aplicación a un punto anterior ya que podemos decir que no esperábamos que la UBU nos proporcionara un servidor para la ejecución de nuestra aplicación. Esta vuelta atrás la realizaremos de manera manual ya que si la realizamos mediante los *commit*, perderemos unos cambios que no queremos perder. Este cambio manual también involucra cambiar las dependencias que eran necesarias para el lanzamiento de la aplicación en Heroku (*sprint 5*), ya que en este *sprint* hemos sustituido un servidor público como es Heroku por el proporcionado por la Universidad de Burgos. Por otra parte, tendremos que ejecutar la aplicación en el servidor provisto por la *UBU* (*arquimedes*), realizando a su vez una comparativa de los distintos servidores posibles para desplegar nuestra *API*

Podemos observar el progreso del *sprint* en la figura A.6

Figura A.6: Progreso en el *sprint* 6.

Sprint 7 - 30-10-2017/06-11-2017

En este *sprint* se pretende trabajar los aspectos relacionados con la seguridad de los modelos subidos al servidor. Con esto queremos decir que en este *sprint* nos dedicaremos a realizar una encriptación de los modelos para que únicamente los usuarios autorizados sean capaces de visualizar el modelo tal y como es. Esto se realiza con el fin de conservar la privacidad de los modelos ya que estos son únicos. La encriptación se realizará en el momento de la subida del modelo a la aplicación alojada en el servidor, y justo en el momento de visualizar el modelo se realizará su desencriptación. La encriptación se hará para los modelos *PLY* tanto en formato **binario** como en **ascii**, mientras que la desencriptación se hará únicamente desde formato *ascii* para así ahorrar tiempo, complejidad y la programación de dos desencriptadores diferentes. A su vez, se realizará un estudio de los tiempos de carga de los modelos debido a las operaciones realizadas para proceder con su encriptación y desencriptación.

En este *sprint* no se ha conseguido integrar el *script* de desencriptación en el cargador de modelos *PLY*, por lo que será una tarea prioritaria en el siguiente *sprint*.

Podemos observar el progreso del *sprint* en la figura A.7

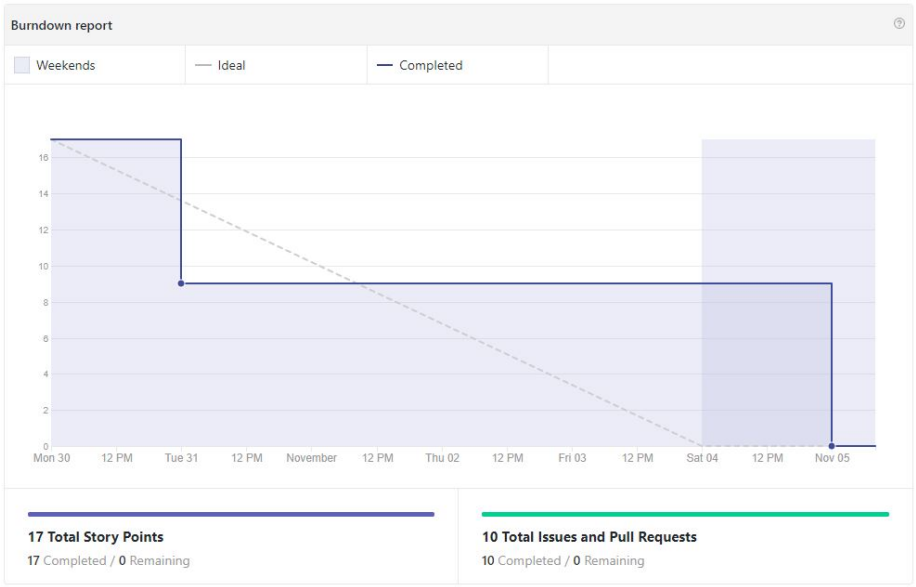

Figura A.7: Progreso en el *sprint* 7.

Sprint 8 - 06-11-2017/12-11-2017

En este *sprint* trataremos de terminar de hacer funcionar la funcionalidad de encriptación y desencriptación de nuestro visor, ya que en el *sprint* anterior tuvimos problemas con el tema de los números en coma flotante, lo cual será mencionado en el *Manual del Programador* D.4. A su vez, dedicaremos este *sprint* a documentar al completo los pasos avanzados hasta el momento, así como solucionar los errores pertinentes a la hora de compilar LaTeX. Con el fin de mejorar los tiempos de carga del visor así como la precisión de los modelos a la hora de encriptarlos y desencriptarlos, se estudiarán diferentes métodos de encriptación (vértices, caras, etc). También realizaremos la configuración *VPN* correspondiente para poder conectarnos al servidor proporcionado por la Universidad de Burgos desde otra red diferente a la de la misma.

No hemos conseguido comprobar el funcionamiento de la aplicación en el servidor «Arquímedes» debido a que no se han instalado correctamente las herramientas requeridas para el funcionamiento de nuestra *API*, lo cual será objetivo para el siguiente *sprint*.

Podemos observar el progreso del *sprint* en la figura A.8

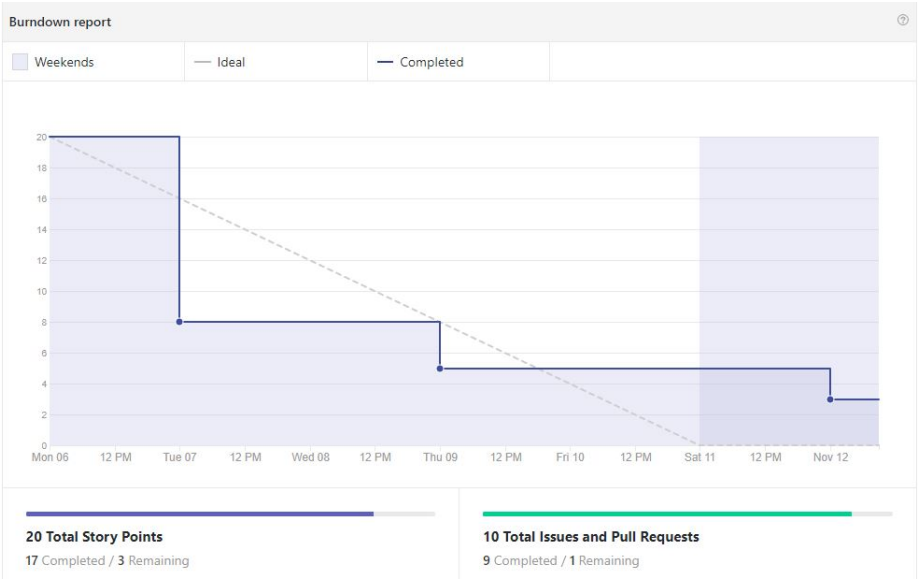

Figura A.8: Progreso en el *sprint* 8.

Sprint 9 - 13-11-2017/19-11-2017

En este *sprint* realizaremos la prueba no realizada en el *sprint* anterior (prueba de ejecución de la aplicación en el servidor «Arquímedes»). También manipularemos la interfaz gráfica de la aplicación cambiando el estilo de ciertas ventanas al mismo tiempo que modificando y ampliando su funcionalidad. A su vez crearemos nuevas pantallas en nuestra aplicación con el fin de aumentar su funcionalidad y facilitar el uso de la misma al usuario (introducción de migas de pan, icono sugerentes y fáciles de comprender, etc). Añadiremos un apartado completo de ejercicios en el que está pensado que interaccione únicamente el profesor y consista en añadir diferentes soluciones a ejercicios propuesto por el mismo, pudiendo albergar una plantilla de cada uno de los ejercicios de cada modelo disponible con el fin de facilitar la enseñanza y corrección.

Para este *sprint* no hemos conseguido realizar los siguientes puntos:

- Autocarga de datos (anotaciones y medidas) en el inicio del visor de ejercicios.
- Documentación del Manual de Usuario (ya que no hemos completado las pantallas que se corresponden con la interfaz de usuario).

Podemos observar el progreso del *sprint* en la figura A.9

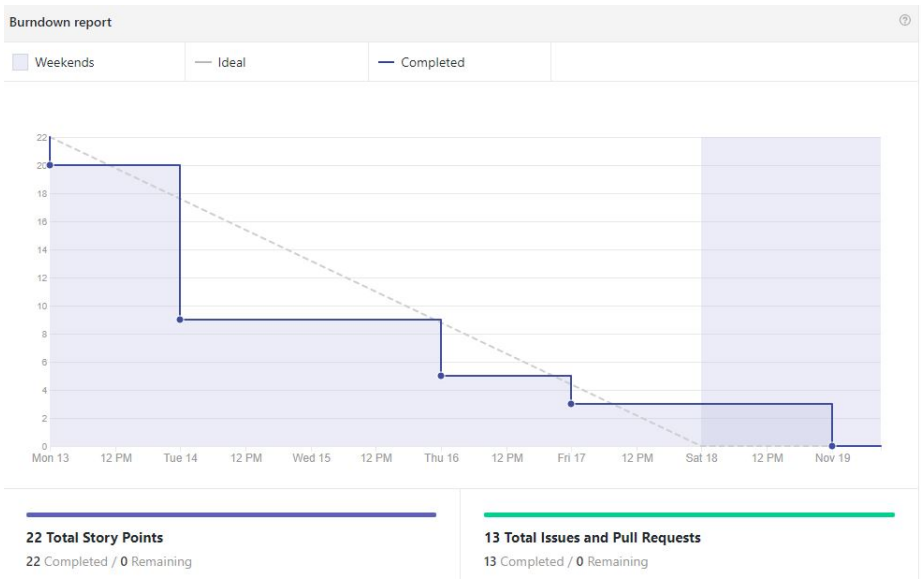

Figura A.9: Progreso en el *sprint* 9.

Sprint 10 - 20-11-2017/26-11-2017

En este *sprint* realizaremos la prueba no realizada en el *sprint* anterior (autocarga de anotacione sy medidas, así como la documentación del Manual de Usuario, pero sin incluir las imágenes de las vistas ya que no tenemos aún las versiones finales de las mismas). Además, trataremos de realizar la configuración del servidor de la Universidad de Burgos con el fin de ejecutar nuestra aplicación en el. No hemos podido avanzar mucho en este *sprint* debido a mi dedicación parcial al proyecto debido s estar trabajando al mismo tiempo.

Para este *sprint* no hemos conseguido realizar los siguientes puntos:

- Llevar a cabo la configuración correspondiente que nos permita ejecutar nuestra *API* en el servidor proporcionado por la Universidad de Burgos.

Podemos observar el progreso del *sprint* en la figura A.10

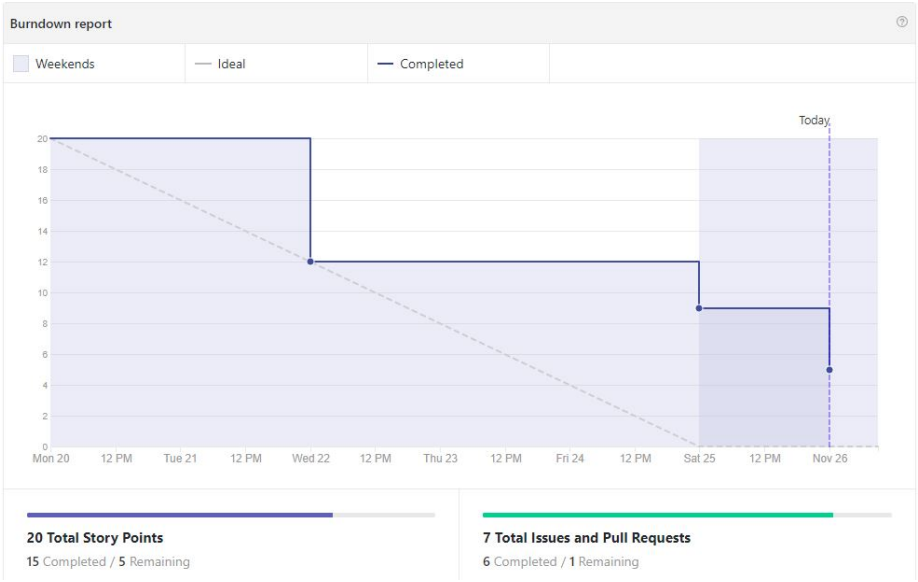

Figura A.10: Progreso en el *sprint* 10.

Sprint 11 - 27-11-2017/03-12-2017

En este *sprint* realizaremos la configuración del servidor «Arquímedes» para posibilitar la ejecución de nuestra aplicación. A su vez, corregiremos errores encontrados en los diferentes botones de la aplicación (carga de puntos, confirmación de eliminación de ejercicio, cancelación de ejercicio ya empezado, etc). También corregiremos los errores acaecidos en la memoria y anexos del proyecto.

Aunque lo hemos intentando fehacientemente, no hemos conseguido realizar la configuración de «Arquímedes» debido a la aparición de diversos fallos dicha configuración, como es la pérdida de la imágenes en la ejecución. de nuestras aplicación.

Podemos observar el progreso del *sprint* en la figura A.11

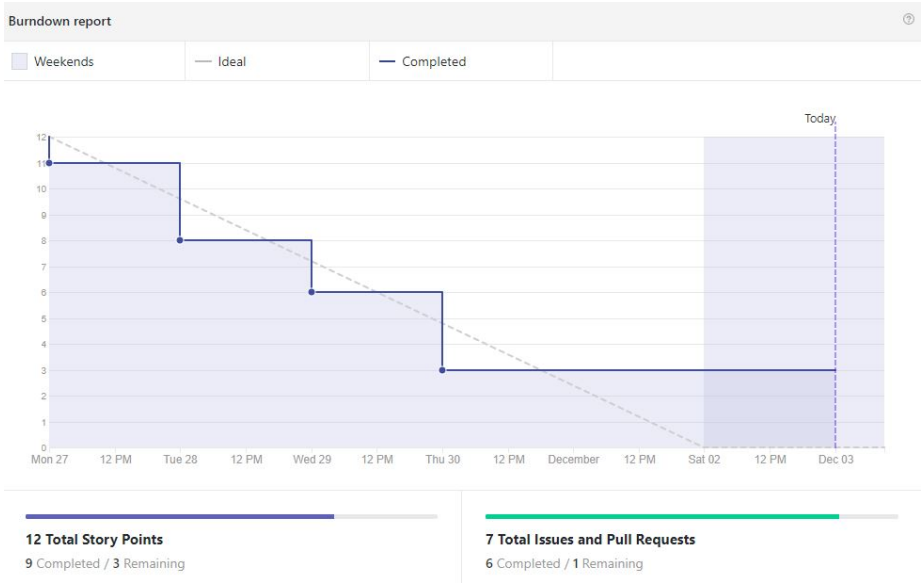

Figura A.11: Progreso en el *sprint* 11.

Sprint 12 - 05-12-2017/10-12-2017

En este *sprint* daremos prioridad a la configuración de nuestro servidor, aunque nos encontramos con dificultades las cuales no es seguro que puedan ser resueltas. A su vez, realizaremos correcciones en el código (refactorización, cambio de nombres, etc), así como cambios en la interfaz con el fin de que el usuario se sienta cómodo con la aplicación y faciliten su entendimiento. Dentro de estos cambios cabe resaltar el cambio de los colores en las esferas importadas pertenecientes a ejercicios de alumnos, así como cambios en los nombre de las etiquetas de las mismas.

Para este *sprint* no hemos conseguido realizar los siguientes puntos:

- Crear botón de *reset* para restablecer los datos de partida del profesor.
- Llevar a cabo la configuración correspondiente que nos permita ejecutar nuestra *API* en el visor «Arquímedes» (será único objetivo del siguiente *sprint*).

Podemos observar el progreso del *sprint* en la figura A.12

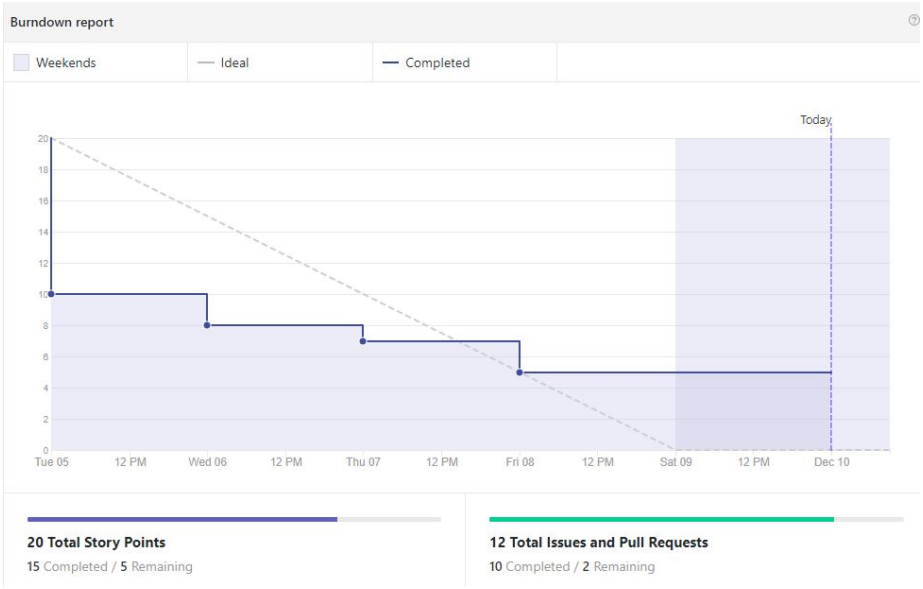

Figura A.12: Progreso en el *sprint* 12.

Sprint 13 - 13-12-2017/17-12-2017

En este *sprint* nos centraremos únicamente en informarnos a fondo acerca de la «librería» de Apache (*mod wsgi*) con la que deseamos conseguir ejecutar nuestra aplicación en el servidor «Arquímedes». A su vez, deshabilitaremos el botón de *logueo* para evitar errores, así como realizaremos los cambios necesarios para solucionar los errores acaecidos en la importación de los mismos datos repetidas veces. Tambié dejaremos solucionada la tarea pendiente de *resetear* los datos del profesor en el visor de ejercicios.

Podemos observar el progreso del *sprint* en la figura A.13

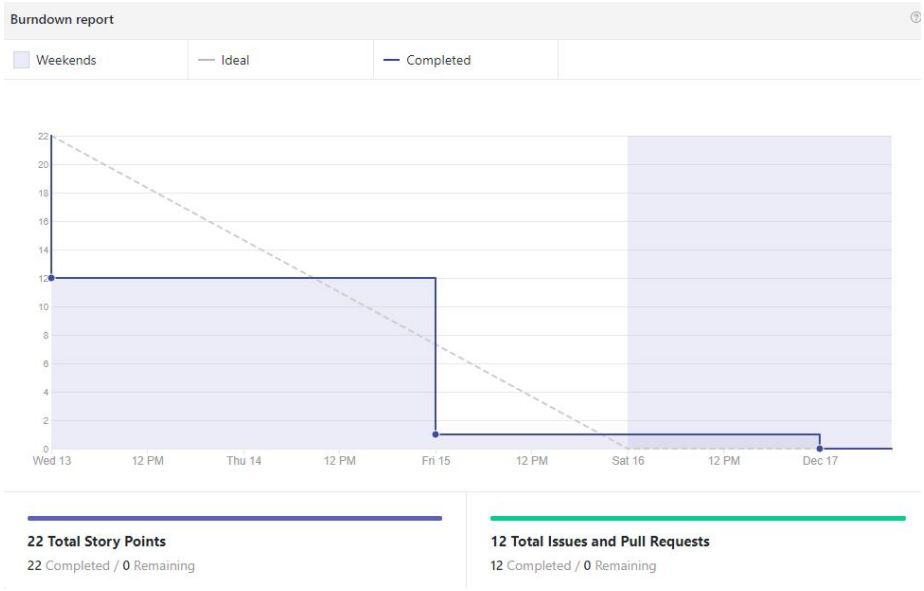

Figura A.13: Progreso en el *sprint* 13.

Sprint 14 - 18-12-2017/24-12-2017

En este *sprint* nos centraremos en la ejecución de nuestra *API* en el servidor «Arquímedes» para lo cual generaremos un *script* que facilite el trabajo de configuración del servidor a nuestro operador. Además, incluiremos expresiones regulares con el fin de limitar los caracteres introducidos por lo alumnos y profesores a la etiquetas de anotaciones y medidas. A su vez, dedicaremos tiempo a la búsqueda de imágenes con las que podamos decorar nuestra interfaz gráfica y así dejar pulida la misma.

Podemos observar el progreso del *sprint* en la figura A.14

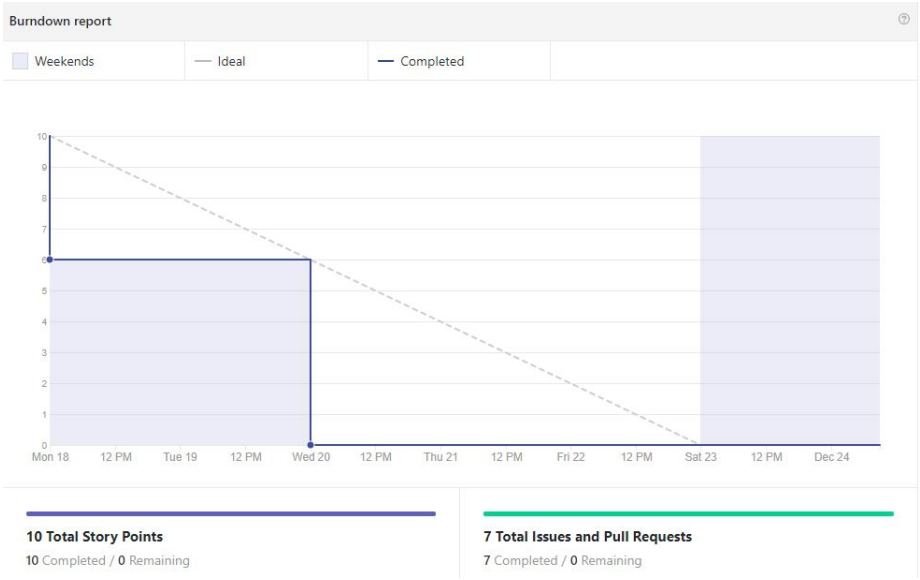

Figura A.14: Progreso en el *sprint* 14.

Sprint 15 - 08-01-2018/14-01-2018

En este *sprint* continuaremos pendientes de la configuración de «Arquímedes» para ejecutar nuestra aplicación debido a que durante finales del mes de Diciembre no se ha trabajado en ello (operador del servidor). A su vez nos centraremos en realizar pruebas y corregir errores encontrados, así como elaborar la documentación del proyecto.

Podemos observar el progreso del *sprint* en la figura A.15

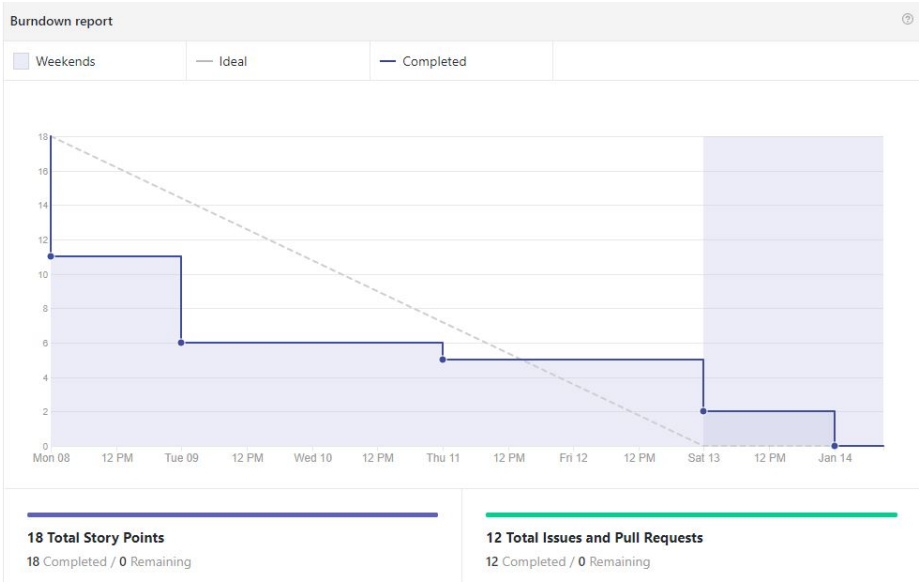

Figura A.15: Progreso en el *sprint* 15.

Sprint 16 - 15-01-2018/21-01-2018

En este *sprint* trataremos de corregir los errores más relevantes de la memoria y anexos y completar éstos en la medida de lo posible. A su vez, realizaremos una reorganización de los diagramas de casos de uso para una mejor visibilidad y comprensión. Así mismo, corregiremos errores de vulnerabilidad en nuestra aplicación. Como último objetivo de este *sprint*, cambiaremos la estética de la página de *login*.

Podemos observar el progreso del *sprint* en la figura A.16

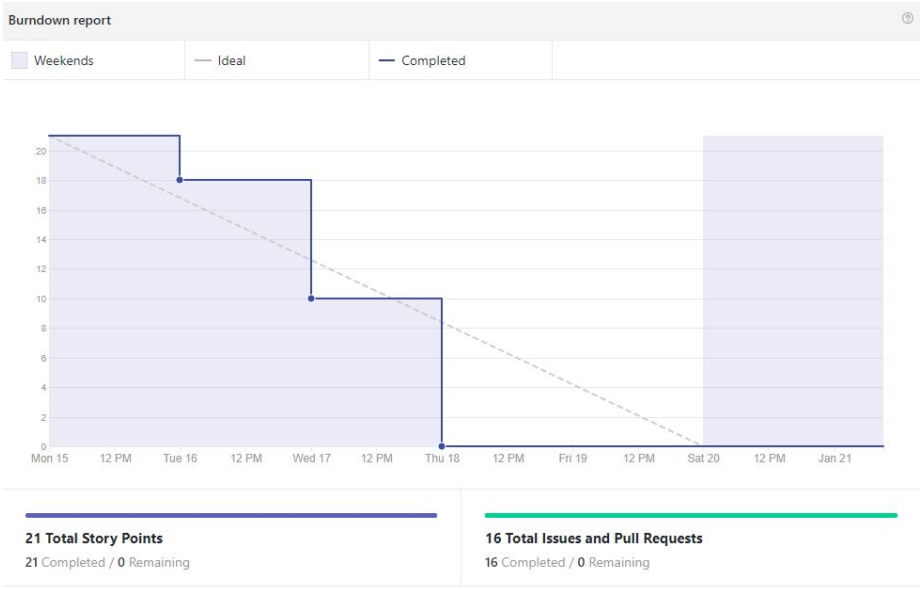

Figura A.16: Progreso en el *sprint* 16.

Sprint 17 - 22-01-2018/28-01-2018

En este *sprint* solucionaremos los errores en la carga de los ejercicios (se intentan cargar los puntos en el modelo antes de que éste se encuentre completamente cargado). Posteriormente, corregiremos errores en la importación de medidas ya que el color obtenido no es el correcto dependiendo de los roles. Refiriéndonos a la estética de la aplicación, cambiaremos el tono de la barra de navegación para mejor visibilidad y mejoraremos la distribución de los mensajes de error de la página de *login*. Por último, realizaremos la documentación referente al estudio de viabilidad y los diagramas de secuencia para la especificación de diseño.

A.3. Estudio de viabilidad

En este apartado explicaremos los distintos escenarios en los que aplicar nuestro proyecto para lograr su futuro desarrollo.

Viabilidad económica

A continuación, analizaremos la viabilidad económica de nuestro proyecto detallando los gastos que hubiera supuesto en los diferentes niveles de una empresa.

Coste de personal

Los recursos humanos empleados en nuestro proyecto han sido el alumno y los tutores. En primer lugar, el alumno ha empleado 20 semanas a media jornada (20 horas semanales). Suponiendo que el alumno tiene un salario mensual bruto de 700€ , al que restándole los gastos de seguridad social del alumno nos deja 656.04€ netos, lo cual ha sido obtenido a partir de[1]:

- Desempleo: 1.55 %
- Formación Profesional: 0.03 %
- Contingencias Comunes: 4.7 %

Esto hace un total de un 6,28 % para el trabajador, por lo que multiplicándolo por el número de meses que ha requerido el proyecto hace un total de 3280.2€ por parte del alumno.

Para calcular el coste de las reuniones con los tutores se conoce que el sueldo base de un ayudante de doctor es de 1815,61 euros mensuales o 21787,32 anuales[2]. En total se imparten 24 créditos con un coste anual por crédito de 907,805€, por lo que teniendo dos tutores que son ayudantes de doctor tendremos:

$$(907,805€ * 0,5 créditos) + (907,805€ * 0,5 créditos) = 907,805€$$

Por lo que el total de costes sería de:

$$3280,2€ + 907,805€ = 4188,005€$$

Coste de hardware

En primer lugar, mencionamos que no se ha necesitado de ningún equipo informático adicional para la correcta realización del proyecto. No obstante, deberemos calcular el coste *hardware* de los equipo utilizados en nuestro proyecto.

[1] http://www.seg-social.es/Internet_1/Trabajadores/CotizacionRecaudaci10777/Basesytiposdecotiza36537/index.htm

[2] http://www.ubu.es/sites/default/files/portal_page/files/pdi_laboral_2017_2.pdf

En este caso, se ha desarrollado nuestra aplicación únicamente en un equipo portátil y estimando su valor en 1400€ y suponiendo una vida útli de 10 años y 20 semanas de proyecto, tenemos un coste de amortización de 54€ aproximadamente:

$$\frac{1400€}{10\ años} * \frac{1\ año}{52\ semanas} * 20\ semanas \approx 54€$$

Coste de software

Este proyecto es una segunda versión y se ha desarrollado empleando plataformas *software* libre para todo lo imprescindible como son las herramientas de documentación, navegadores, *IDE*, etc. Lo único distinto con respecto a su versión anterior ha sido la integración de la aplicación en un servidor proporcionado por la Universidad de Burgos, el cual tiene un SO gratuito, por lo que el coste *software* es nulo.

Coste total

Así, realicemos un cálculo del coste total del proyecto:

Viabilidad legal

Como se mencionó en la versión anterior a nuestro proyecto ([4]), uno de los datos potencialmente sensibles que manejará la aplicación serán los modelos a visualizar. Por ello, tendremos que estar atentos a los posibles cambios con el tipo de licencia que puedan poseer dichos modelos. No por el tipo de fichero ni porque dicho tipo sea propietario –pues es un estándar abierto–, sino por la preferencia de mantener los mencionados modelos a buen recaudo. Además de lo anterior, también almacenaremos unos pocos datos de carácter personal (por el momento solo el correo electrónico institucional). A su vez, cabe mencionar que la imágenes utilizadas para la estética de la aplicación tienen licencia con derecho a reutilización y a reutilización con modificaciones.

Software

En este apartado no se han realizado variaciones de la versión anterior, por lo que incluiremos tal cual la documentación de la primera versión del proyecto [4].

Pasamos ahora a analizar la licencia que deberíamos tener en nuestro proyecto. Tendremos que observar las licencias de los elementos que hemos

utilizado para su realización. Podemos observar las diferentes licencias de los componentes del proyecto en **??**.

Analizando la tabla, vemos que el conjunto de las licencias está en MIT, Apache 2.0, BSD-2-Clause, BSD-3-Clause, BSD. Las licencias BSD y es más restrictiva que su hija la BSD-3-Clause, y esta a su vez más restrictiva que la BSD-2-Clause. Simplemente se diferencian en no poder emplear el nombre de la licencia para promover productos derivados sin permiso y en la obligatoriedad de mencionar la autoría del software en el que te bases para realizar tu proyecto. Pero para más simplicidad, evitaremos incluirlas en la figura A.17. De izquierda a derecha, las hemos ordenado de más permisivas a más restrictivas.

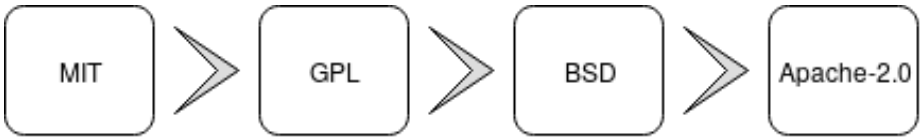

Figura A.17: Compatibilidad entre licencias del proyecto

Apéndice B

Especificación de Requisitos

B.1. Introducción

Se expondrán en esta sección los distintos objetivos que la aplicación debe lograr, así como los requisitos que debe cumplir.

B.2. Objetivos generales

Como objetivos generales de nuestro proyecto tendremos:

- Diferenciar en nuestra aplicación entre los diferentes roles de usuario, cada uno con sus distintas funcionalidades.
- Añadir restricciones en el uso de caracteres para anotaciones, medidas y ejercicios.
- Facilitar la navegabilidad durante el uso de la aplicación.
- Creación de una interfaz que permita al profesor facilidad para corregir ejercicios de alumnos y detectar copias entre ellos.
- Proporcionar seguridad para los modelos debido a su unicidad a la hora de ser alojados en el servidor web.
- Desplegar nuestra aplicación en un servidor web.

B.3. Catalogo de requisitos

A continuación, listaremos el conjunto de requisitos funcionales extraídos a partir de los objetivos generales del proyecto.

Requisitos funcionales

Los requisitos funcionales se han sacado a partir de un diagrama de casos de uso hecho desde cero aunque ciertos requisitos coinciden con los de dicha versión anterior, por lo que serán iguales.

- **RF-1 Visor de modelos**: la aplicación debe ser capaz de visualizar un modelo.
  - **RF-1.1 Gestión de anotaciones**: la aplicación debe ser capaz de manejar anotaciones sobre el modelo.
    - **RF-1.1.1 Añadir anotación**: el usuario debe poder añadir una anotación.
    - **RF-1.1.2 Eliminar anotación**: el usuario debe poder eliminar una anotación.
    - **RF-1.1.3 Editar anotación**: el usuario debe poder editar la etiqueta de una anotación.
    - **RF-1.1.4 Seleccionar anotación**: el usuario debe poder seleccionar una anotación.
    - **RF-1.1.5 Deseleccionar anotación**: el usuario debe poder deseleccionar una anotación.
  - **RF-1.2 Gestión de medidas**: la aplicación debe ser capaz de manejar medidas sobre el modelo.
    - **RF-1.2.1 Añadir medida**: el usuario debe poder añadir una medida.
    - **RF-1.2.2 Eliminar medida**: el usuario debe poder eliminar una medida.
    - **RF-1.2.3 Editar medida**: el usuario debe poder editar la etiqueta de una medida.
    - **RF-1.2.4 Seleccionar medida**: el usuario debe poder seleccionar una medida.
    - **RF-1.2.5 Deseleccionar medida**: el usuario debe poder deseleccionar una medida.

- **RF-1.3 Exportar puntos**: la aplicación debe ser capaz de exportar las anotaciones y medidas que se encuentren actualmente en el visor.
- **RF-1.4 Importar puntos**: la aplicación debe ser capaz de importar las anotaciones y medidas que se encuentren actualmente en el visor.
- **RF-2 Visor de ejercicio**: la aplicación debe ser capaz de visualizar un ejercicio guardado.
  - **RF-2.1 Gestión de anotaciones**: la aplicación debe ser capaz de manejar anotaciones sobre el modelo.
    - **RF-2.1.1 Añadir anotación**: el usuario debe poder añadir una anotación.
    - **RF-2.1.2 Eliminar anotación**: el usuario debe poder eliminar una anotación.
    - **RF-2.1.3 Editar anotación**: el usuario debe poder editar la etiqueta de una anotación.
    - **RF-2.1.4 Seleccionar anotación**: el usuario debe poder seleccionar una anotación.
    - **RF-2.1.5 Deseleccionar anotación**: el usuario debe poder deseleccionar una anotación.
  - **RF-2.2 Gestión de medidas**: la aplicación debe ser capaz de manejar medidas sobre el modelo.
    - **RF-2.2.1 Añadir medida**: el usuario debe poder añadir una medida.
    - **RF-2.2.2 Eliminar medida**: el usuario debe poder eliminar una medida.
    - **RF-2.2.3 Editar medida**: el usuario debe poder editar la etiqueta de una medida.
    - **RF-2.2.4 Seleccionar medida**: el usuario debe poder seleccionar una medida.
    - **RF-2.2.5 Deseleccionar medida**: el usuario debe poder deseleccionar una medida.
- **RF-2.3 Restaurar datos**: la aplicación debe ser capaz de restaurar los datos iniciales del ejercicio previo a la carga del ejercicio del alumno.
- **RF-2.4 Cancelar ejercicio**: la aplicación debe ser capaz de cancelar y salir de un ejercicio en curso.

- **RF-2.5 Exportar puntos**: la aplicación debe ser capaz de exportar las anotaciones y medidas que se encuentren actualmente en el visor.
- **RF-2.6 Importar puntos**: la aplicación debe ser capaz de importar las anotaciones y medidas que se encuentren actualmente en el visor.
- **RF-3 Listar modelos**: la aplicación debe ser capaz de mostrar de una pasada los diferentes modelos disponibles.
  - **RF-3.1 Eliminar modelos**: el usuario debe ser capaz de eliminar modelos.
- **RF-4 Listar ejercicios**: la aplicación debe ser capaz de manejar un listado de ejercicios de cada modelo.
  - **RF-4.1 Gestión de ejercicios**: la aplicación debe ser capaz de manejar medidas sobre el modelo.
    - **RF-4.1.1 Añadir ejercicio**: el usuario debe poder añadir un ejercicio.
    - **RF-4.1.2 Eliminar ejercicio**: el usuario debe poder eliminar un ejercicio.
    - **RF-4.1.3 Modificar ejercicio**: el usuario debe poder modificar un ejercicio.
    - **RF-4.1.4 Editar nombre ejercicio**: el usuario debe poder editar el nombre de un ejercicio.
    - **RF-4.1.5 Guardar ejercicio**: el usuario debe poder guardar un ejercicio modificado.
- **RF-5 Subir modelos**: la aplicación debe ser capaz de manejar un listado de ejercicios de cada modelo.

Requisitos no funcionales

Estos requisitos coinciden con los de la versión anterior a excepción de uno. Aun así, mencionaremos todos para que quede más claro.

- **RNF-1 Usabilidad**: el conjunto de elementos visuales de la interfaz deben ser intuitivos y conocidos por el usuario medio, permitiendo un rápido aprendizaje.
- **RNF-2 Mantenibilidad**: la aplicación debe desarrollarse siguiendo alguna técnica que permita facilidad de mantenimiento e incorporación de nuevas características, así como corrección de errores.

- **RNF-3 Soporte**: la aplicación debe poder emplearse sobre un amplio conjunto de navegadores.
- **RNF-4 Internacionalización**: la aplicación debe estar diseñada para soportar diferentes idiomas.
- **RNF-5 Control de acceso**: la aplicación debe soportar control de acceso de usuarios.
- **RNF-6 Despliegue**: la aplicación debe estar desplegada de manera que sea accesible sin necesidad de una ejecución local.

B.4. Especificación de requisitos

Actores

En nuestro caso solamente tendremos dos actores, que serán el alumno y el profesor(administrador).

Diagrama de casos de uso

A continuación se mostrará el diagrama de casos de uso de nuestro proyecto por niveles en las figuras B.1, B.2 y B.3:

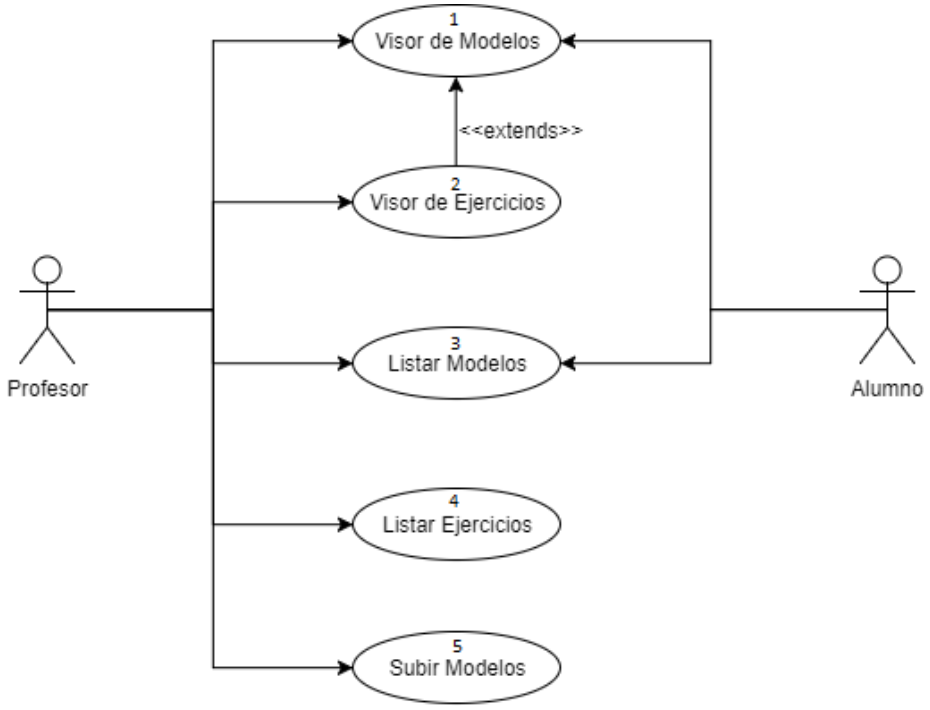

Figura B.1: Nivel 1 del diagrama de casos de uso.

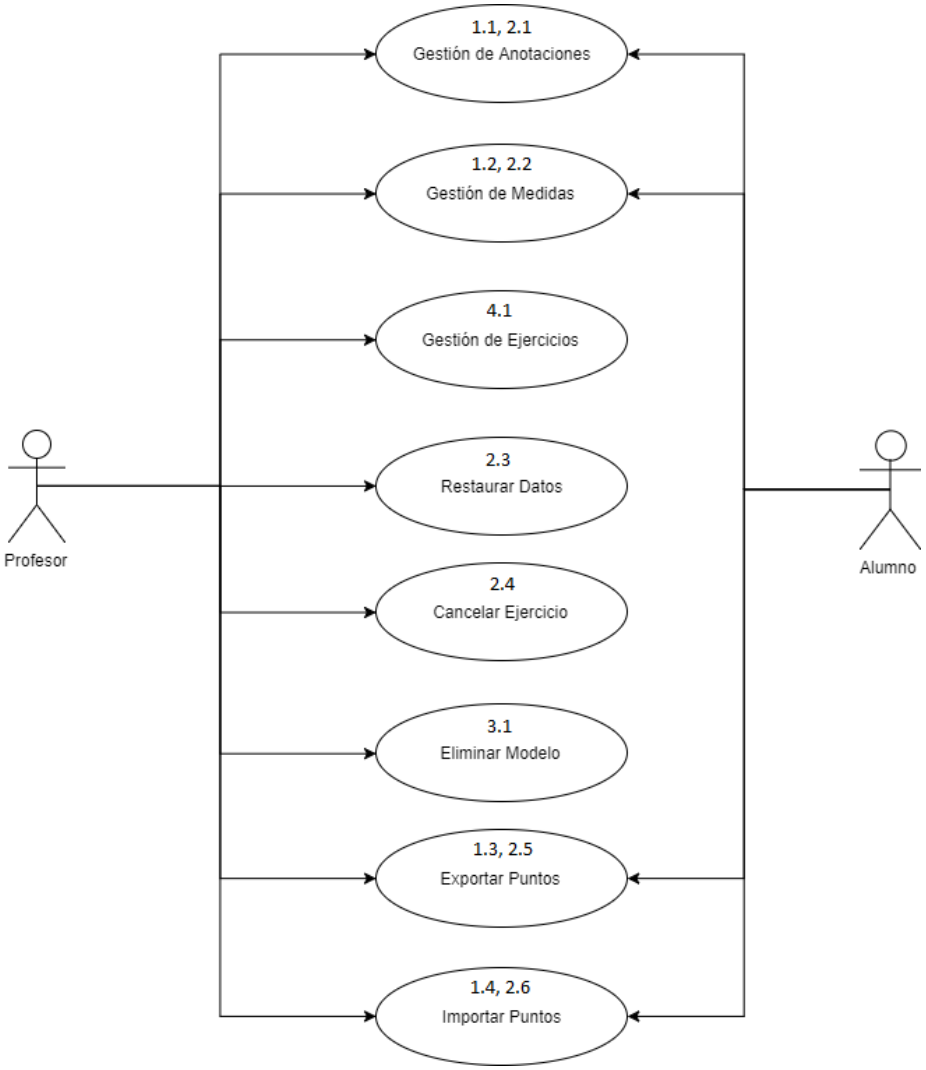

Figura B.2: Nivel 2 del diagrama de casos de uso.

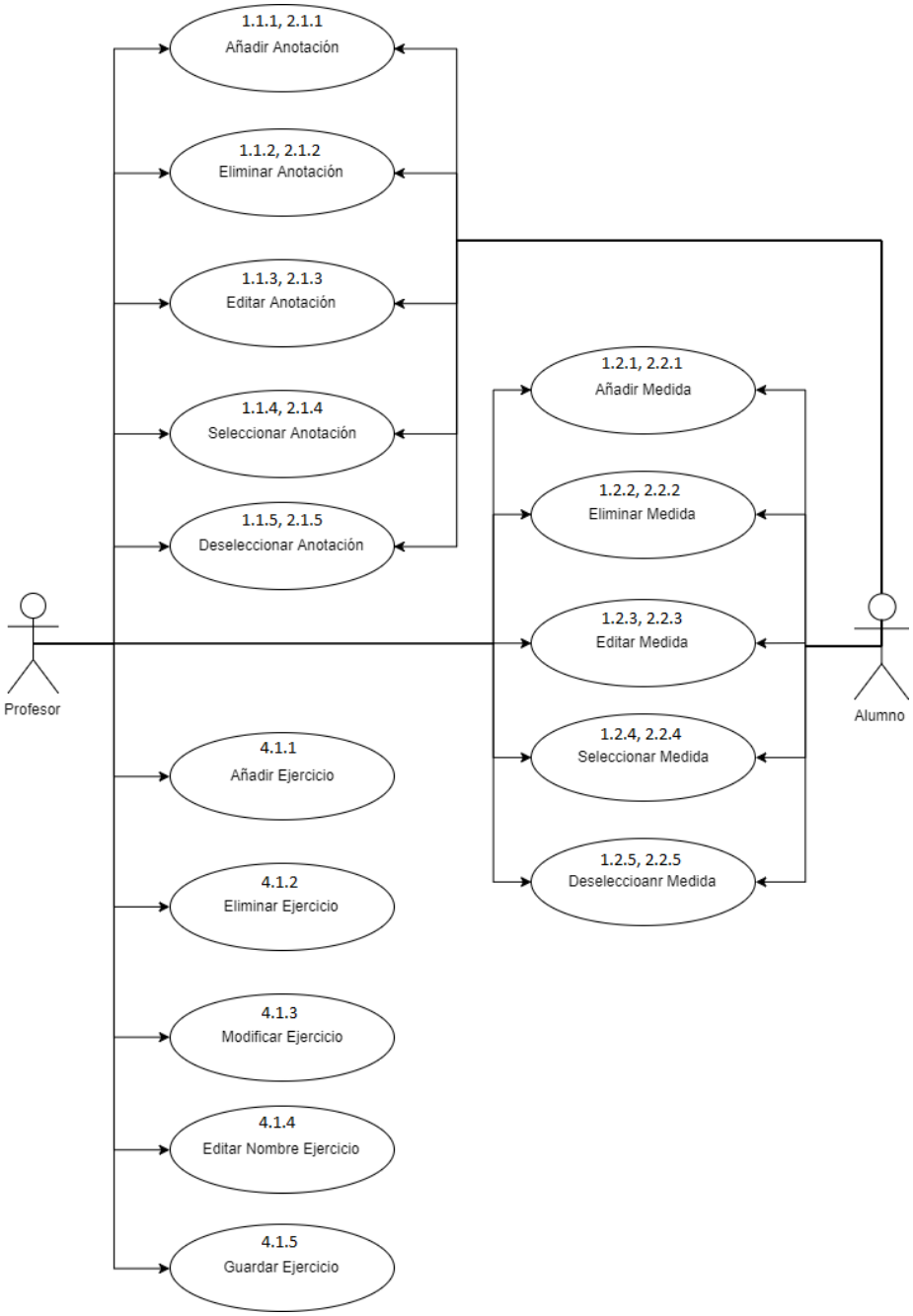

Figura B.3: Nivel 3 del diagrama de casos de uso.

CU-1	**Visor de modelo**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.1, RF-1.2, RF-1.3, RF-1.4
Descripción	Permite al usuario visualizar un modelo y realizar operaciones sobre el mismo.
Precondiciones	• El usuario ha seleccionado un modelo.
Acciones	1. El usuario abre un modelo. 2. Se muestran el modelo sin anotaciones ni medidas. 3. Se da la posibilidad de gestionar tanto anotaciones como medidas.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.1: CU-1 Visor de modelos

CU-1.1	**Gestión de anotaciones**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1, RF-1.1.1, RF-1.1.2, RF-1.1.3, RF-1.1.4, RF-1.1.5
Descripción	Permite gestionar las anotaciones (añadirlas, eliminarlas, etc.).
Precondiciones	• El usuario tiene un modelo abierto.
Acciones	1. Se muestran las anotaciones del modelo visualizado. 2. Se muestran las opciones de añadir, eliminar, editar la etiqueta de y deseleccionar las anotaciones.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.2: CU-1.1 Gestión de anotaciones

CU-1.1.1	**Añadir anotación**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.1
Descripción	Permite al usuario añadir una anotación al modelo.
Precondiciones	-
Acciones	1. El usuario pincha en añadir anotación. 2. El usuario pincha sobre el modelo en la zona donde quiere crear una anotación. 3. Una esfera es añadida en el modelo para representar la anotación. 4. Un elemento aparece en un menú para representar la etiqueta de la anotación.
Postcondiciones	• Se añade una anotación al modelo.
Excepciones	-
Importancia	Alta

Tabla B.3: CU-1.1.1 Añadir anotación

CU-1.1.2	**Eliminar anotación**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.1
Descripción	Permite al usuario eliminar una anotación.
Precondiciones	• Que exista una anotación en el modelo. • (opcional) Que una anotación se encuentre seleccionada.
Acciones	1. Seleccionar una anotación (opcional). 2. Pulsar sobre borrar anotación. Si se ha realizado anterior, saltamos el siguiente paso. 3. Seleccionar una anotación (opcional). 4. Se borra la anotación.
Postcondiciones	• Se elimina la anotación seleccionada.
Excepciones	-
Importancia	Media

Tabla B.4: CU-1.1.2 Eliminar anotación

CU-1.1.3	**Editar anotación**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.1
Descripción	Permite editar la etiqueta de una anotación.
Precondiciones	• Que haya anotaciones.
Acciones	1. Seleccionar una anotación. 2. Pulsar sobre editar anotación. 3. Se muestra un diálogo con la etiqueta actual de la anotación. 4. El usuario edita el texto. 5. Pulsar sobre guardar. 6. El texto de la anotación cambia.
Postcondiciones	• La anotación modificada cambia su texto.
Excepciones	-
Importancia	Media

Tabla B.5: CU-1.1.3 Editar anotación

CU-1.1.4	**Seleccionar anotación**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.1
Descripción	Permite seleccionar una anotación resaltándola.
Precondiciones	• Tiene que existir al menos una anotación.
Acciones	1. El usuario selecciona una anotación en el modelo (opción 1). 2. El usuario selecciona una anotación en la lista lateral (opción 2). 3. La anotación se destaca tanto en el modelo como en la lista lateral.
Postcondiciones	
Excepciones	-
Importancia	Media

Tabla B.6: CU-1.1.4 Seleccionar anotación

CU-1.1.5	**Deseleccionar anotación**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.1
Descripción	Permite al usuario deseleccionar una anotación.
Precondiciones	• Que haya al menos una anotación seleccionada.
Acciones	1. Pinchar sobre una anotación en el modelo (opción 1). 2. Pinchar sobre una anotación en la lista lateral (opción 2). 3. Pinchar sobre deseleccionar todo en la lista lateral (opción 3). 4. Si se ha realizado opcion 1 o 2, se deselecciona dicha anotación. 5. Si se realiza opción 3, se deseleccionan todas las anotaciones.
Postcondiciones	
Excepciones	-
Importancia	Media

Tabla B.7: CU-1.1.5 Deseleccionar anotación

CU-1.2	**Gestión de medida**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1, RF-1.2.1, RF-1.2.2, RF-1.2.3, RF-1.2.4, RF-1.2.5
Descripción	Permite gestionar las medidas (añadirlas, eliminarlas, ...).
Precondiciones	• El usuario tiene abierto un modelo.
Acciones	1. Se muestran las medidas del modelo visualizado. 2. Se muestran las opciones de añadir, eliminar, editar la etiqueta de y deseleccionar las medidas.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.8: CU-1.2 Gestión de medidas

CU-1.2.1	**Añadir medida**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.2
Descripción	Permite al usuario añadir una medida al modelo.
Precondiciones	-
Acciones	1. El usuario pincha en añadir medida. 2. El usuario pincha sobre el modelo en la zona donde quiere crear una anotación. 3. El usuario pincha sobre el modelo en la posición donde quiere que vaya el otro extremo de la medida. 4. Dos esferas y una raya aparecen en el visor para representar la medida. 5. Un elemento aparece en un menú para representar la etiqueta de la medida, que será la etiqueta y las unidades.
Postcondiciones	• Se añade una medida al modelo.
Excepciones	• No se añaden los dos puntos de la medida.
Importancia	Alta

Tabla B.9: CU-1.2.1 Añadir medida

CU-1.2.2	**Eliminar medida**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.2
Descripción	Permite al usuario eliminar una medida.
Precondiciones	• Que exista una medida en el modelo. • (opcional) Que una medida se encuentre seleccionada.
Acciones	1. Seleccionar una medida (opción 1). 2. Pulsar sobre borrar medida. 3. Seleccionar una medida (opción 2). 4. Se borra la medida.
Postcondiciones	• Se elimina la medida seleccionada.
Excepciones	-
Importancia	Media

Tabla B.10: CU-1.2.2 Eliminar medida

CU-1.2.3	**Editar medida**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.2
Descripción	Permite editar la etiqueta de una medida.
Precondiciones	• Que haya medidas.
Acciones	1. Seleccionar una medida. 2. Pulsar sobre editar medida. 3. Mostrar un diálogo con la etiqueta actual de la medida. 4. El usuario edita el texto. 5. Pulsar sobre guardar. 6. El texto de la medida cambia.
Postcondiciones	• La medida modificada cambia su texto.
Excepciones	-
Importancia	Media

Tabla B.11: CU-1.2.3 Editar medida

CU-1.2.4	**Seleccionar medida**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.2
Descripción	Permite seleccionar una medida resaltándola.
Precondiciones	• Tiene que existir al menos una medida.
Acciones	1. El usuario selecciona una medida en el modelo (opción 1). 2. El usuario selecciona una medida en la lista lateral (opción 2). 3. La medida se destaca tanto en el modelo como en la lista lateral.
Postcondiciones	-
Excepciones	-
Importancia	Media

Tabla B.12: CU-1.2.4 Seleccionar medida

CU-1.2.5	**Deseleccionar medida**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1.2
Descripción	Permite al usuario deseleccionar una medida.
Precondiciones	• Que haya al menos una medida seleccionada.
Acciones	1. Pinchar sobre una medida en el modelo (opción 1). 2. Pinchar sobre una medida en la lista lateral (opción 2). 3. Pinchar sobre deseleccionar todo en la lista lateral (opción 3). 4. Si se ha realizado opcion 1 o 2, se deselecciona dicha medida. 5. Si se realiza opción 3, se deseleccionan todas las medidas.
Postcondiciones	-
Excepciones	-
Importancia	Media

Tabla B.13: CU-1.2.5 Deseleccionar medida

CU-1.3	**Exportar punto**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1
Descripción	Permite exportar los puntos que se estén visualizando.
Precondiciones	-
Acciones	1. Pulsar sobre exportar puntos. 2. Un archivo se descarga.
Postcondiciones	• Un archivo nuevo con nuestros puntos se almacena. • Si el archivo es exportado por un alumno, quedará reflejado en archivo almacenado.
Excepciones	-
Importancia	Media

Tabla B.14: CU-1.3 Exportar puntos

CU-1.4	**Importar punto**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-1
Descripción	Permite al usuario cargar puntos creados previamente.
Precondiciones	• Tener un archivo correctamente formado con puntos para el modelo.
Acciones	1. Pulsar sobre importar puntos. 2. Seleccionar el archivo deseado. 3. Pulsar sobre abrir. 4. Las anotaciones y medidas se añaden. 5. El archivo importado contendrá información de quién la ha realizado.
Postcondiciones	• Se añaden añaden nuevas anotaciones y medidas.
Excepciones	• El archivo está mal formado. • El archivo no contiene anotaciones y medidas.
Importancia	Media

Tabla B.15: CU-1.4 Importar puntos

CU-2	**Visor de ejercicios**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.1, RF-2.2, RF-2.3, RF-2.4, RF-2.5, RF-2.6, RF-1
Descripción	Permite al usuario visualizar un ejercicio y realizar operaciones sobre el mismo.
Precondiciones	• El usuario ha seleccionado un ejercicio.
Acciones	1. El usuario abre un ejercicio. 2. Se muestran el ejercicio con las anotaciones y medidas guardadas. 3. Se da la posibilidad de gestionar tanto anotaciones como medidas y además incluir ejercicios de los alumnos.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.16: CU-2 Visor de ejercicios

CU-1.1	**Gestión de anotaciones**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2, RF-2.1.1, RF-2.1.2, RF-2.1.3, RF-2.1.4, RF-2.1.5
Descripción	Permite gestionar las anotaciones (añadirlas, eliminarlas, etc.).
Precondiciones	• El usuario tiene un ejercicio abierto.
Acciones	1. Se muestran las anotaciones del ejercicio visualizado. 2. Se muestran las opciones de añadir, eliminar, editar la etiqueta de y deseleccionar las anotaciones.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.17: CU-2.1 Gestión de anotaciones

CU-1.1.1	**Añadir anotación**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.1
Descripción	Permite al usuario añadir una anotación al ejercicio.
Precondiciones	-
Acciones	1. El usuario pincha en añadir anotación. 2. El usuario pincha sobre el ejercicio en la zona donde quiere crear una anotación. 3. Una esfera es añadida en el ejercicio para representar la anotación. 4. Un elemento aparece en un menú para representar la etiqueta de la anotación.
Postcondiciones	• Se añade una anotación al ejercicio.
Excepciones	-
Importancia	Alta

Tabla B.18: CU-2.1.1 Añadir anotación

CU-2.1.2	**Eliminar anotación**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.1
Descripción	Permite al usuario eliminar una anotación.
Precondiciones	• Que exista una anotación en el ejercicio. • (opcional) Que una anotación se encuentre seleccionada.
Acciones	1. Seleccionar una anotación (opcional). 2. Pulsar sobre borrar anotación. Si se ha realizado anterior, saltamos el siguiente paso. 3. Seleccionar una anotación (opcional). 4. Se borra la anotación.
Postcondiciones	• Se elimina la anotación seleccionada.
Excepciones	-
Importancia	Media

Tabla B.19: CU-2.1.2 Eliminar anotación

CU-2.1.3	**Editar anotación**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.1
Descripción	Permite editar la etiqueta de una anotación.
Precondiciones	• Que haya anotaciones.
Acciones	1. Seleccionar una anotación. 2. Pulsar sobre editar anotación. 3. Se muestra un diálogo con la etiqueta actual de la anotación. 4. El usuario edita el texto. 5. Pulsar sobre guardar. 6. El texto de la anotación cambia.
Postcondiciones	• La anotación modificada cambia su texto.
Excepciones	-
Importancia	Media

Tabla B.20: CU-2.1.3 Editar anotación

CU-2.1.4	**Seleccionar anotación**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.1
Descripción	Permite seleccionar una anotación resaltándola.
Precondiciones	• Tiene que existir al menos una anotación.
Acciones	1. El usuario selecciona una anotación en el ejercicio (opción 1). 2. El usuario selecciona una anotación en la lista lateral (opción 2). 3. La anotación se destaca tanto en el ejercicio como en la lista lateral.
Postcondiciones	-
Excepciones	-
Importancia	Media

Tabla B.21: CU-2.1.4 Seleccionar anotación

CU-2.1.5	**Deseleccionar anotación**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.1
Descripción	Permite al usuario deseleccionar una anotación.
Precondiciones	• Que haya al menos una anotación seleccionada.
Acciones	1. Pinchar sobre una anotación en el ejercicio (opción 1). 2. Pinchar sobre una anotación en la lista lateral (opción 2). 3. Pinchar sobre deseleccionar todo en la lista lateral (opción 3). 4. Si se ha realizado opcion 1 o 2, se deselecciona dicha anotación. 5. Si se realiza opción 3, se deseleccionan todas las anotaciones.
Postcondiciones	-
Excepciones	-
Importancia	Media

Tabla B.22: CU-2.1.5 Deseleccionar anotación

CU-2.1	**Gestión de medidas**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2, RF-2.2.1, RF-2.2.2, RF-2.2.3, RF-2.2.4, RF-2.2.5
Descripción	Permite gestionar las medidas (añadirlas, eliminarlas, ...).
Precondiciones	• El usuario tiene abierto un ejercicio.
Acciones	1. Se muestran las medidas del ejercicio visualizado. 2. Se muestran las opciones de añadir, eliminar, editar la etiqueta de y deseleccionar las medidas.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.23: CU-2.2 Gestión de medidas

CU-2.2.1	**Añadir medida**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.2
Descripción	Permite al usuario añadir una medida al modelo.
Precondiciones	-
Acciones	1. El usuario pincha en añadir medida. 2. El usuario pincha sobre el ejercicio en la zona donde quiere crear una anotación. 3. El usuario pincha sobre el ejercicio en la posición donde quiere que vaya el otro extremo de la medida. 4. Dos esferas y una raya aparecen en el visor para representar la medida. 5. Un elemento aparece en un menú para representar la etiqueta de la medida, que será la etiqueta y las unidades.
Postcondiciones	• Se añade una medida al ejercicio.
Excepciones	• No se añaden los dos puntos de la medida.
Importancia	Alta

Tabla B.24: CU-2.2.1 Añadir medida

CU-2.2.2	**Eliminar medida**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.2
Descripción	Permite al usuario eliminar una medida.
Precondiciones	• Que exista una medida en el ejercicio. • (opcional) Que una medida se encuentre seleccionada.
Acciones	1. Seleccionar una medida (opción 1). 2. Pulsar sobre borrar medida. 3. Seleccionar una medida (opción 2). 4. Se borra la medida.
Postcondiciones	• Se elimina la medida seleccionada.
Excepciones	-
Importancia	Media

Tabla B.25: CU-2.2.2 Eliminar medida

CU-1.2.3	**Editar medida**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.2
Descripción	Permite editar la etiqueta de una medida.
Precondiciones	• Que haya medidas.
Acciones	1. Seleccionar una medida. 2. Pulsar sobre editar medida. 3. Mostrar un diálogo con la etiqueta actual de la medida. 4. El usuario edita el texto. 5. Pulsar sobre guardar. 6. El texto de la medida cambia.
Postcondiciones	• La medida modificada cambia su texto.
Excepciones	-
Importancia	Media

Tabla B.26: CU-2.2.3 Editar medida

CU-2.2.4	**Seleccionar medida**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.2
Descripción	Permite seleccionar una medida resaltándola.
Precondiciones	• Tiene que existir al menos una medida.
Acciones	1. El usuario selecciona una medida en el ejercicio (opción 1). 2. El usuario selecciona una medida en la lista lateral (opción 2). 3. La medida se destaca tanto en el ejercicio como en la lista lateral.
Postcondiciones	-
Excepciones	-
Importancia	Media

Tabla B.27: CU-2.2.4 Seleccionar medida

CU-2.2.5	**Deseleccionar medida**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2.2
Descripción	Permite al usuario deseleccionar una medida.
Precondiciones	• Que haya al menos una medida seleccionada.
Acciones	1. Pinchar sobre una medida en el ejercicio (opción 1). 2. Pinchar sobre una medida en la lista lateral (opción 2). 3. Pinchar sobre deseleccionar todo en la lista lateral (opción 3). 4. Si se ha realizado opcion 1 o 2, se deselecciona dicha medida. 5. Si se realiza opción 3, se deseleccionan todas las medidas.
Postcondiciones	-
Excepciones	-
Importancia	Media

Tabla B.28: CU-2.2.5 Deseleccionar medida

CU-2.3	**Restaurar datos**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2
Descripción	Permite al usuario restaurar los valores de anotaciones y medidas iniciales (previamente guardados).
Precondiciones	• Que un ejercicio tenga importado los datos del alumno.
Acciones	1. Pinchar en el botón de restaurar. 2. Se eliminarán todas las anotaciones y medidas que no pertenezcan al ejercicio previamente guardado (Si no hay ejercicio hecho, se borra todo).
Postcondiciones	-
• Se recupera el ejercicio guardado. **Excepciones**	-
Importancia	Alta

Tabla B.29: CU-2.3 Restaurar datos

CU-2.3	**Cancelar ejercicio**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2
Descripción	Permite al usuario cancelar un ejercicio abierto y salir de el.
Precondiciones	• Tener un ejercicio abierto.
Acciones	1. Pinchar en el botón de cancelar. 2. Si existen cambios sobre ele ejercicio de partida, sale un diálogo preguntado si deseamos guardar los cambios. 3. Si guardamos lo cambios, el ejercicio se actualiza. 4. Si no guardamos los cambios, el ejercicio no se modifica. 5. Si cancelamos, nos quedamos en el visor de ejercicios.
Postcondiciones	-
• Se vuelve a la lista de ejercicios. **Excepciones**	-
Importancia	Alta

Tabla B.30: CU-2.4 Cancelar ejercicio

CU-2.5	**Exportar punto**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2
Descripción	Permite exportar los puntos que se estén visualizando.
Precondiciones	-
Acciones	1. Pulsar sobre exportar puntos. 2. Un archivo se descarga.
Postcondiciones	• Un archivo nuevo con nuestros puntos se almacena.
Excepciones	-
Importancia	Media

Tabla B.31: CU-2.5 Exportar puntos

CU-2.6	**Importar punto**
Versión	2.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-2
Descripción	Permite al usuario cargar puntos creados previamente.
Precondiciones	• Tener un archivo correctamente formado con puntos para el ejercicio.
Acciones	1. Pulsar sobre importar puntos. 2. Seleccionar el archivo deseado. 3. Pulsar sobre abrir. 4. Las anotaciones y medidas se añaden. 5. Si las anotaciones eran de un alumno, se importan con un color de esfera distinto.
Postcondiciones	• Se añaden nuevas anotaciones y medidas.
Excepciones	• El archivo está mal formado. • El archivo no contiene anotaciones y medidas.
Importancia	Media

Tabla B.32: CU-2.6 Importar puntos

CU-3	**Listar modelos**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-3.1
Descripción	Permite al usuario ver los modelos disponibles.
Precondiciones	• Haber accedido a la aplicación correctamente.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Modelos.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.33: CU-3 Listar modelos

CU-3.1	**Eliminar modelos**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-3
Descripción	Permite al usuario eliminar modelos.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Modelos. 3. Elegir el modelo a eliminar
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.34: CU-3.1 Eliminar modelos

CU-4	**Listar ejercicios**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-4.1
Descripción	Permite al usuario listar los ejercicios.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Ejercicios. 3. Elegir un modelo para ver sus ejercicios.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.35: CU-4 Listar ejercicios

CU-4.1	**Gestión de ejercicios**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-4, RF-4.1.1 ,RF-4.1.2, RF-4.1.3, RF-4.1.4, RF-4.1.5
Descripción	Permite al realizar operaciones sobres los ejercicios.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Ejercicios. 3. Elegir un modelo para ver sus ejercicios. 4. Elegir la opción correspondiente.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.36: CU-4.1 Gestión de ejercicios

CU-4.1.1	**Añadir ejercicios**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-4.1
Descripción	Permite añadir un ejercicio.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Ejercicios. 3. Elegir un modelo para ver sus ejercicios. 4. Elegir la opción de añadir.
Postcondiciones	• Se abre el visor de ejercicios para dicho modelo.
Excepciones	-
Importancia	Alta

Tabla B.37: CU-4.1.1 Añadir ejercicios

CU-4.1.2	**Eliminar ejercicios**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-4.1
Descripción	Permite eliminar un ejercicio.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Ejercicios. 3. Elegir un modelo para ver sus ejercicios. 4. Elegir la opción de eliminar (botón de papelera). 5. Elegimos en el diálogo emergente si queremos seguir adelante.
Postcondiciones	• Se elimina el ejercicio si elegimos que se elimine en la comprobación.
Excepciones	
Importancia	Alta

Tabla B.38: CU-4.1.2 Eliminar ejercicios

CU-4.1.3	**Modificar ejercicios**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-4.1
Descripción	Permite Modificar un ejercicio.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Ejercicios. 3. Elegir un modelo para ver sus ejercicios. 4. Elegir la opción de modificar (botón de llave inglesa). 5. Se abre el ejercicio guardado correspondiente.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.39: CU-4.1.3 Modificar ejercicios

CU-4.1.4	**Editar nombre ejercicios**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-4.1
Descripción	Permite editar el nombre de un ejercicio.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Ejercicios. 3. Elegir un modelo para ver sus ejercicios. 4. Elegir la opción de modificar (botón de lápiz). 5. Se un cuadro de edición del nombre del ejercicio.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.40: CU-4.1.4 Editar nombre ejercicios

CU-4.1.5	**Editar nombre ejercicios**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	RF-4.1
Descripción	Permite guardar un ejercicio modificado.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Ejercicios. 3. Elegir un modelo para ver sus ejercicios. 4. Elegir la opción de modificar (botón de llave inglesa). 5. Se abre el visor de ejercicios. 6. Si se realizan modificaciones se pulsa el botón de guardar para salvar el ejercicio.
Postcondiciones	-
Excepciones	-
Importancia	Alta

Tabla B.41: CU-4.1.5 Guardar ejercicios

CU-5	**Subir modelo**
Versión	1.0
Autor	Jose Manuel Moral Garrido
Requisitos asociados	
Descripción	Permite subir modelo.
Precondiciones	• Haber accedido a la aplicación correctamente. • Tener rol de Profesor.
Acciones	1. Entrar en la aplicación. 2. Elegir la opción de Subir Modelo.
Postcondiciones	-
• Elegir modelo a subir. **Excepciones**	-
Importancia	Alta

Tabla B.42: CU-5 Subir modelo

Apéndice C

Especificación de diseño

C.1. Introducción

A continuación especificaremos la manera en la que hemos organizado cada uno de los elementos del proyecto y detallaremos las razones de por qué lo hemos hecho así. La información mostrada hasta el apartado **??** es idéntica a la primera versión de nuestro proyecto ya que no se han realizado modificaciones al respecto [4].

C.2. Diseño de datos

Este apartado junto con el diagrama de clases del apartado C.4 se ha sacado de la primera versión de nuestro proyecto puesto que no se han realizado cambios. A su vez, se han modificado los diagramas de flujo sustituyéndolos por diagramas de secuencia para conocer el *Proceso de login*, *Selección de un Modelo* y *Selección de un Ejercicio*.

Puntos

Necesitamos una convención para que nuestro sistema de importar y exportar puntos funcione correctamente. El archivo *JSON* (ver código C.1) se compone por:

- `filename` El nombre del modelo al que pertenece.
- `annotations` Una lista con las diferentes anotaciones.

Código C.1: Esquema JSON

```
{"filename": "a filename",
 "annotations": [annotation, ..., annotation],
 "measurements": [measurement, ..., measurement]}

annotation = {"tag": "a tag",
                          points: [point]}

measurement = {"tag": "a tag",
                           points: [point, point]}

point = {"x": float,
                "y": float,
                "z": float}
```

- `measurements` Una lista con las medidas.
- `timestamp` Una marca de tiempo de cuando se ha validado en el servidor.
- `checksum` Un código de comprobación (*md5* para ser concretos.)

Cada una de las anotaciones se compone por una etiqueta y un punto. Las medidas son una etiqueta y dos puntos. Finalmente, los puntos son tres números de tipo *float* (`x`, `y`, `z`).

La estructura de los archivos *JSON* puede verse en el código C.1.

C.3. Diseño procedimental

En el momento que el servidor está en línea, éste puede aceptar peticiones de inicio de sesión. Cuando un usuario lo solicita, es redirigido a la página de *login*, en el que tendrá que introducir su correo y contraseña de UBUVirtual. Tras ello, el servidor buscará en su base de datos de usuarios si éste está presente en ella. Si no está, redirigiremos al usuarios a la página de *login* anteriormente mencionada. Si pertenece, entonces el servidor lanza una consulta a la *API* de UBUVirtual, para saber si también se encuentra reconocido como usuario en dicha plataforma y, en caso afirmativo, conocer su rol en una asignatura en concreto. Si no consta el correo, o si la contraseña

es incorrecta, se redirige a la página de *login*. En caso de que ambas preguntas sean correctas, dependiendo del rol que dicho usuario ocupe, tendrá la posibilidad de acceder únicamente a visualizar un modelo (rol de alumno) o podrá acceder a visualizar un modelo o un ejercicio (rol de profesor). En el caso de que se quiera visualizar un modelo, se redirigirá a la estantería de modelos desde la cual se elegirá el modelo a visualizar. En caso de querer consultar un ejercicio (únicamente el profesor), se realizará la misma mecánica pero esta vez se redirigirá a la estantería de ejercicios y, desde allí, se elegirá el ejercicio que corresponda.

En la figura C.1 se aprecia cómo sucede el proceso de *login*.

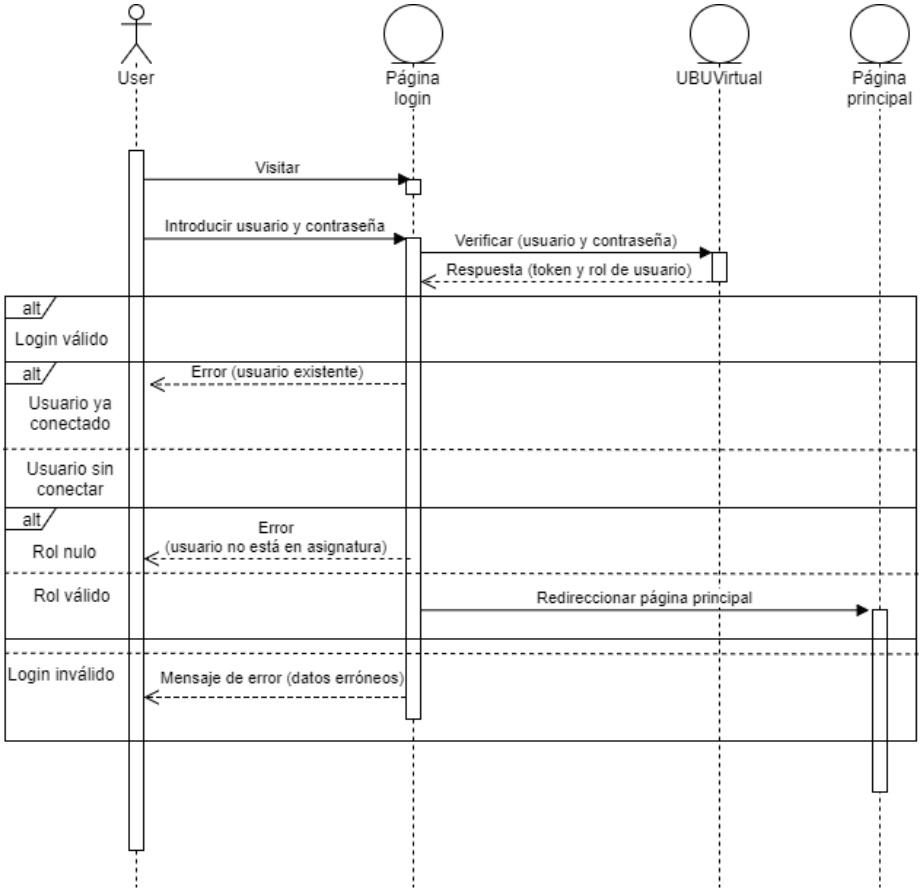

Figura C.1: Proceso de *login*

Por ejemplo, si el siguiente proceso que queremos realizar es elegir el modelo, se seguirá el diagrama de la figura C.2.

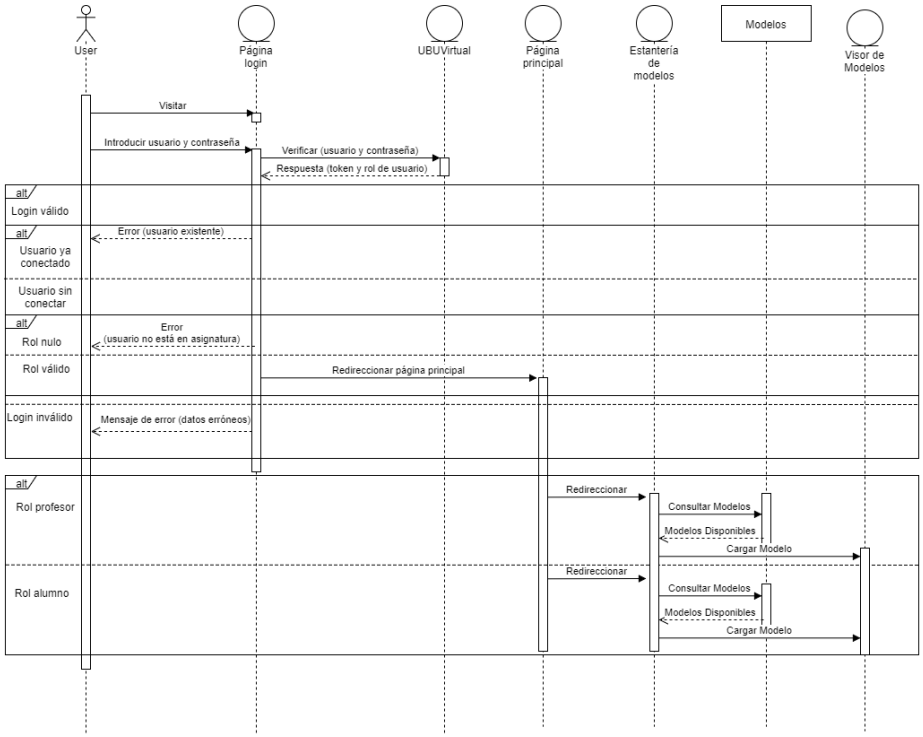

Figura C.2: Selección de un modelo

Si somos profesor, podremos a su vez elegir un ejercicio tal y como se aprecia en la figura C.3.

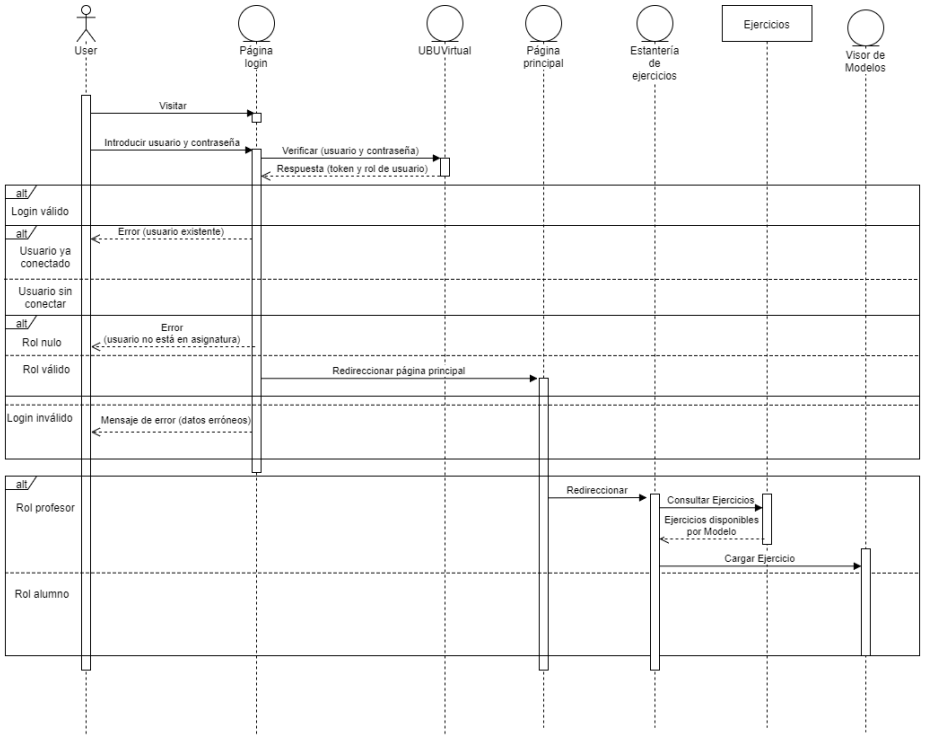

Figura C.3: Selección de un ejercicio

C.4. Diseño arquitectónico

En la figura C.4 el diagrama de clases que define la parte JavaScript del proyecto. La clase «`Utils`» aunque separada del resto, sí está relacionada con las demás clases, aunque para facilitar la comprensión hemos evitado las uniones. Dependen de ella las clases «`AnnotationTool`», «`MeasurementTool`», «`PointManager`» y «`Scene`».

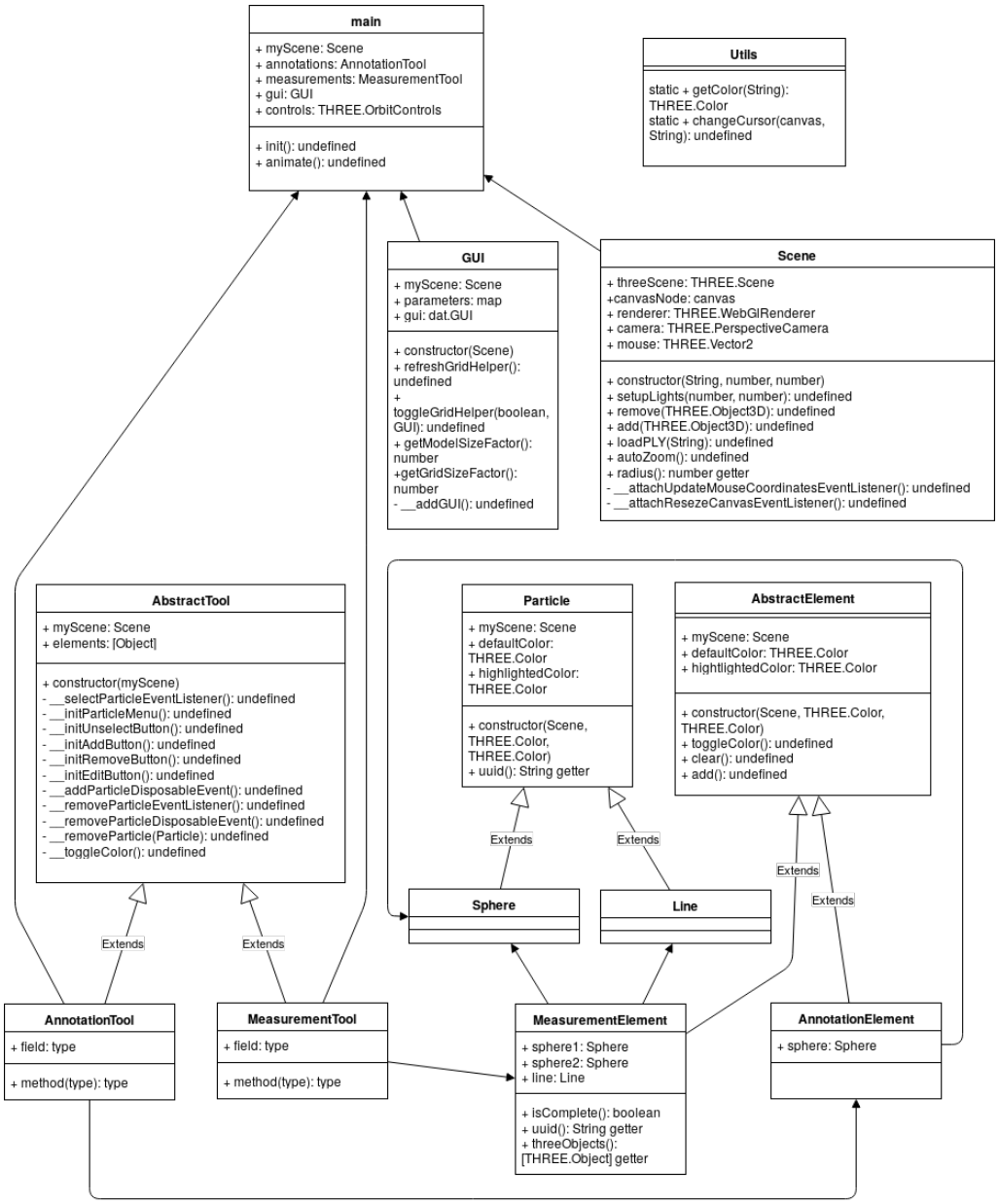

Figura C.4: Diagrama de clases de parte JavaScript

Sin embargo, no añadiremos el diagrama de paquetes JavaScript, puesto que solamente existe uno.

Apéndice D

Documentación técnica de programación

D.1. Introducción

Primero de todo mencionar que los siguientes apartados se han sacado del Manual del Programador del proyecto predecesor (consultar [4]) ya que, de no ser así, tendríamos que referenciar continuamente y la persona interesada podría tener problemas para el despliegue o modificación. En este apartado mostraremos la información necesaria para cualquier persona que quiera ejecutar nuestra aplicación o realizar modificaciones en la misma.

D.2. Manual de despliegue

Vamos a ver los pasos a realizar para desplegar la aplicación desde cero. No obstante y con el fin de facilitar el despliegue, tenemos un par de *scripts* para dicha tarea: podemos emplear `«install_run.bat»` para Windows, además de `«install_run.sh»` para Linux. Si la ejecución del *script* lanza algún error, siempre podemos realizar una instalación paso a paso. A su vez, trataremos de desplegar nuestra aplicación en el servidor «Arquímedes» con su configuración previa (ver sección D.6).

Lanzar un terminal o consola

Para comenzar, lo primero que tendremos que hacer es ejecutar una consola, algo realmente fácil. En Windows por ejemplo, pulsando `«Windows + R»` podemos ejecutar un comando. En este caso, queremos lanzar la consola

de comandos, así que teclearemos «cmd» en el recuadro de texto y pulsaremos ejecutar. Así de fácil. En versiones recientes de Windows, también podremos utilizar la «Powershell» si así lo deseamos. En el menú de inicio podemos buscar la aplicación como tal. En Linux lanzar el terminal incluso suele estar asociado a un atajo de teclado: «Ctrl + Alt + T». En caso contrario, podemos buscar en el menú de aplicaciones, suele estar nombrado como «Terminal», otras veces como «XTerm», etc. Finalmente, en Mac se llama «Terminal».

Instalando Python

Los componentes en que se basa el servidor web necesitan Python, que será nuestra primera parada. Para ello, iremos a la página de descargas de Python, y nos descargaremos la versión «3.6.4»[1] para nuestro sistema operativo. Tras la descarga, procederemos a su instalación. Durante el proceso en Windows, se preguntará si desea añadir Python al PATH del sistema. Si desea utilizar los comandos tal cual se escriben aquí, debe aceptar esta opción. Si no, en vez de escribir «python3» o «pip3», deberá introducir la ruta absoluta de los binarios.

Creando un entorno virtual

Tras completar el paso anterior, vamos a instalar algunos componentes. Si ya tenemos más instalaciones de Python, o si simplemente queremos que nuestra instalación siga limpia, deberemos hacer un pequeño rodeo creando un entorno virtual sobre el que recaigan los componentes. Insistimos, si has leído estos casos y no te sientes identificado, puedes saltarse esta explicación.

```
$ python3 -m venv mi_venv
```

Lo que hace el código arriba descrito[2] es crear un entorno virtual en la carpeta actual bajo la carpeta visor. Una vez creado, queremos que las dependencias se instalen en él así que necesitamos activarlo primero. Para ello, escogemos la opción según nuestro sistema operativo, escogiéndola de la tabla D.2 y nuestra consola para ejecutarla en un terminal.

[1] https://www.python.org/downloads/release/python-364/

[2] Es posible que el comando «python3» no sea reconocido en ocasiones; sería posible que el alias real para la instalación sea «python».

Posix	bash/zsh	`$ source mi_venv/bin/activate`
	fish	`$ . mi_venv/bin/activate.fish`
	csh/tcshq	`$ source mi_venv/bin/activate.csh`
Windows	cmd.exe	`C:\> mi_venv\Scripts\activate.bat`
	Powershell	`PS C:\> mi_venv\Scripts\Activate.ps1`

Ahora, si instalamos las dependencias se hará dentro de este entorno, sin «manchar» nuestra instalación original. Cuando necesitemos salir de dicho entorno virtual, teclearemos «`deactivate`», o simplemente cerraremos el terminal.

Instalando las dependencias Python

En la carpeta raíz del proyecto, encontramos un archivo «`requirements.txt`». A éste debemos hacer mención con el comando `pip3`[3]:

```
$ pip3 install -r <ruta proyecto>/requirements.txt
```

Con las dependencias ya instaladas, tenemos que dar permisos de acceso a los usuarios que deseemos.

Dando permisos a usuarios

Si queremos que un usuario pueda utilizar la aplicación, tenemos que incluirlo en la base de datos. Con el fin de facilitar añadir usuarios a dicha tabla, tenemos un método del que nos podemos aprovechar en el módulo «`MyApp`», llamado «`import_users_to_db`». Para llamarlo, desde la línea de comandos debemos seleccionar la carpeta «`MyApp`» mediante el comando «`cd`». Como argumento al mismo, escribiremos la carpeta hacia el cual nos queremos dirigir. Si lo que deseamos es ir un directorio hacia arriba, escribiremos como parámetro «`..`».

Si la consola nos indica por ejemplo que estamos sobre la carpeta MyApp del proyecto, subiremos una posición escribiendo:

```
$ cd ..
```

Si la consola nos indica que estamos en la carpeta que contiene el proyecto (típicamente «`3D-Viewer-v2.0`»), entonces podemos iniciar el terminal interactivo de python mediante:

```
$ python3
```

[3] es posible que también necesitemos escribirlo sin el «`3`».

Dentro del mismo, podremos importar el módulo «MyApp» mediante:

```
>>>import MyApp
```

Ahora, emplearemos el método que queríamos, «import_users_to_db». Podemos hacerlo sin parámetro:

```
>>>MyApp.import_users_to_db()
```

Y, entonces buscará el archivo «instance/users_to_import.csv» por defecto. Si por el contrario, deseamos que busque otro archivo (siempre dentro de la carpeta «instance»), se lo podemos pasar como parámetro:

```
>>>MyApp.import_users_to_db('otro_archivo')
```

De esta manera, los usuarios dentro de este archivo serán importados a la base de datos para otorgarles permiso de acceso. El archivo que vamos a emplear para decirle los usuarios, tiene que tener un formato especial: *CSV*. Éste es realmente sencillo de realizar mediante un programa de hoja de cálculo como «Excel», «Numbers» o «Calc», por mencionar los más conocidos. Simplemente llenaremos las celdas de la siguiente manera:

email	**nombre**
email@mail.com	pepito
secret@mail.com	007 Agent

Tabla D.1: Ejemplo de *CSV*

Como se ve, en cada columna hay un tipo de dato, y el nombre de dicho dato se determina mediante la cabecera de dicha columna. Las celdas inferiores a la misma contendrán los datos de dicho valor para cada uno de los usuarios. Lo mismo pasa con el resto de las columnas.

Pista: para introducir estos datos en la hoja de cálculo correspondiente, normalmente se pueden pegar directamente desde una tabla en un navegador, etc.

Una vez los datos en la hoja de cálculo el programa correspondiente tendrá un pequeño asistente para conseguir exportar a *CSV* los valores introducidos. Dicha orden suele encontrarse en el menú «Archivo», bajo la opción «Exportar...», «Exportar como...», «Guardar como...», etc. Normalmente podremos emplear los parámetros por defecto del asistente,

pero no nos olvidemos de guardar el archivo en la carpeta «instance» con el nombre adecuado.

Nota: En el momento de redactar esta sección, el único campo necesario es el *email*, pero para saber a ciencia cierta cuáles son los campos por almacenar, conviene echar un vistazo al *script* de creación en «MyApp/sql_scripts/schema.sql».

Lanzando el servidor

Ahora, en la carpeta raíz del proyecto, encontraremos el fichero «runserver.py» al que haremos referencia con la orden:

```
$ python3 <ruta proyecto>/runserver.py
```

Et *voilà*: el servidor está lanzado, y el terminal nos indica la dirección que debemos introducir para consultar el contenido: «localhost:5000» o «127.0.0.1:5000» como se prefiera. Por fin debería ver la interfaz de la aplicación.

D.3. Estructura de directorios

Para comenzar, mostraremos los archivos existentes en la raíz del proyecto. Cabe mencionar que la estructura de la versión anterior del visor ha sido modificada ya que no seguía una estructura correcta para su despliegue en un servidor. Por ello, la aplicación en sí se encuentra en la raíz.

- **.gitignore** Posee los diferentes elementos que el sistema *GIT* debe evitar revisar.
- **activate_this.py** Archivo que deberá ser movido por el operador del servidor a la carpeta `mi_venv/bin/` con el fin de que Apache adquiera las dependencias de instaladas en el entorno virtual.
- **config.py** Archivo de configuración predeterminada. Estos parámetros se verán sobrescritos por los añadidos en `instance/config.py`.
- **host_config.txt** Configuración de Apache para la máquina proporcionada como servidor.
- **install_run.bat** *Script* para Windows que genera un entorno virtual, lo registra, instala las dependencias Python, y lanza el servidor.

- **install_run.sh** El mismo *script*, pero para Linux.
- **jsconfig.json** Archivo de configuración para *VSCode*.
- **MyApp.wsgi** Archivo que sirve de configuración para que Apache (*mod_wsgi*) sepa que tiene que ejecutar.
- **package.json** Define las dependencias y comandos disponibles llamándolo desde npm (Node).
- **requirements.txt** Requisitos de los módulos para Python.
- **rollup.config.js** Configuración para el *plugin* que empaqueta el código.
- **runserver.py** *Script* que carga el módulo principal para lanzar el servidor.
- **runtime.txt** Fichero que contiene el entorno de ejecución de la aplicación.
- **tyings.json** Configuración IntelliSense de Visual Studio Code.

A continuación, mostraremos las diferentes carpetas dentro de la raíz del proyecto, pasando a su posterior despliegue:

- Documentación
- instance
- mi_venv
- MyApp
- node modules
- Resources
- src
- typings

Entraremos a partir de ahora en detalle.

Documentación

En ella tendremos el conjunto de documentaciones generadas para el proyecto, cuyas carpetas son:

- **javascript** En ella tenemos la documentación generada para la parte Javascript, esto es, toda la lógica e interfaz del visor.
- **LaTex** Contiene la memoria y anexos del proyecto.
- **python** Posee la documentación generada mediante Sphinx[4] para el servidor.

instance

Aunque no es un directorio incluido en el repositorio, debemos mencionarlo, puesto que en él incluiremos algunas configuraciones propias de la instancia que estemos desplegando o desarrollando y que no queramos incluir en la versión que subamos al servidor. Algunos archivos de utilidad que meteremos son:

- **config.py** En este archivo incluiremos algunos parámetros de configuración que debemos sobrescribir, como el estado de `«DEBUG»`, o el secreto de aplicación `«SECRET_KEY»`.
- **users_to_import.csv** Es la ruta por defecto para el archivo *CSV* que nos permite importar la lista de usuarios a los que queremos dar permiso.

mi_venv

Entorno virtual en e que se instalarán las dependencias correspondientes para la posterior ejecución de la aplicación.

MyApp

En él está la aplicación en sí misma, casi toda la lógica de negocio y toda la visualización. Dicha carpeta contiene:

- **sql_scripts** Incluye los *scripts* de utilidad para la base de datos.

[4] http://www.sphinx-doc.org/en/stable/

- **static** Directorio para cargar elementos estáticos de las páginas.
  - **images** Contiene las imágenes utilizadas en la aplicación.
  - **js** *Scripts* que importaremos. Aquí tendremos el código JavaScript una vez transpilado, y por tanto, gran parte de la lógica del visor.
  - **uploads** En este directorio terminan los modelos, además de los *PNG* como sus miniaturas.A su vez contiene:
    - **exercises** Incluye los ejercicios realizados por el profesor que servirá de ayuda a la hora de corregir.
- **templates** En él se encuentran las plantillas en formato jinja2[5].
- **translations** En este directorio encontraremos los ficheros plantilla para poder realizar la traducción con Babel.
- **babel.cfg** Configuración con la que Babel busca las etiquetas identificadoras.
- **forms.py** Contiene constructores de los diferentes formularios.
- **___init___.py** Constructor del módulo.
- **messages.pot** Binario con las traducciones de Babel ya realizadas
- **read_write_ply.py** Archivo que contiene el código necesario para posibilitar la lectura de archivos *.PLY*.
- **User.py** Clase necesaria para la gestión de usuarios.
- **users.db** Base de datos que alberga la información de los usuarios admitidos.
- **views.py** Contiene los diferentes *endpoints* de nuestro servidor.

node modules

Contiene el conjunto de dependencias *Node*.

[5] http://jinja.pocoo.org/

Resources

Contiene algunos recursos que no sabíamos donde meter, pues no pertenecen estrictamente a ninguna de las partes anteriores. Entre sus elementos se encuentran los archivos «fuente» de los diagramas, o los generadores de informes para los archivos *PLY*, que posiblemente integremos en un futuro sobre el servidor.

src

Código fuente de JavaScript sin compilar.

typings

Archivos de configuración para nuestro *IDE* (Visual Studio Code).

D.4. Manual del programador

Instalando dependencias Node

Para labores de desarrollo, necesitaremos tener instalado Node y algunas dependencias extra. Podremos si tecleamos desde la carpeta raíz del proyecto el siguiente comando:

```
$ npm install
```

Éste y todos los comandos que sean con npm, deberán ser ejecutados desde la misma ruta.

Transpilación JavaScript

Durante el desarrollo del proyecto, separamos el código JavaScript en diferentes fuentes, que requieren para funcionar de realizar una transpilación a un estándar que comprendan los navegadores actuales. Aunque éstos soportan gran parte de las características de *ES6* (*ES2015*), estándar en el que hemos escrito el fuente original, hay elementos que todavía no soportan, como los *import* entre las diferentes clases. Adicionalmente, aprovechamos para hacer el archivo destino más pequeño, en un proceso denominado *minifying*. Para ello, vamos a empaquetar el código de una manera más adecuada, y nos ayudaremos de rollup:

```
$ npm run build
```

Si por ejemplo estamos desarrollando, sería molesto llamar a esta función cada vez que hiciésemos un cambio. Para ello, hemos implementado una configuración que nos permitirá invocar una sola vez a un comando de transpilación, y cada vez que realicemos un cambio automáticamente se generará el fichero destino en formato *minified*. Para hacer esto, tenemos que llamar al comando:

```
$ npm run build dev
```

Construir la documentación JavaScript

Para hacerlo, tenemos otro *script* incluido en la configuración de Node para facilitarnos la tarea. Solamente hay que escribir:

```
$ npm run jsdoc
```

Construir la documentación de Python

Construir la documentación de Python es igual de sencillo:

```
$ npm run pydoc-http
```

Eso sí, hay que tener en cuenta que éste comando no nos avisa de que sigamos algunas pautas recomendables, así que necesitamos de otro comando para revisar la sintaxis:

```
$ npm run pydoc-style
```

Usando el analizador de modelos: eliminando puntos superfluos

Con el fin de mejorar los modelos, añadimos varias funciones a nuestras herramientas con el fin de «recortar» en la medida de lo posible el espacio que ocupan los modelos, además de evitar errores posteriores con los mismos.

Hemos empaquetado toda la funcionalidad en un módulo Python, por lo que abriremos la dicha consola sobre la carpeta `«Resources»`:

```
$ python3
```

A continuación, cargaremos la función a emplear:

```
>>> from ply_utils import process_ply_file
```

Con la función cargada, aplicaremos los parámetros necesarios.

```
>>> process_ply_file('una ruta de archivo', 'encoding')
```

La ruta de archivo puede ser tanto relativa como absoluta. El parámetro de `encoding` es opcional, y si se incluye debe ser `«b'ascii'»`, `«b'binary_little_endian'»` o `«b'binary_big_endian'»`.

Este último parámetro es el que nos permitirá comprimir los archivos *ASCII* en formato binario, más compacto. Así mismo, podremos generar un fichero *ASCII* a partir de uno binario.

Babel - Internacionalización

Como el proceso ya está iniciado, en el manual detallaremos la actualización y no el comienzo. La instalación de Babel es un requisito incluido en `«requirements.txt»`, así que podemos instalarlo junto con el resto de dependencias del proyecto (preferentemente en el entorno virtual):

```
$ pip3 install -r requirements.txt
```

Los ficheros que Babel procesa están determinados por babel.cfg, así que cualquier necesidad extra deberemos reflejarla en dicho archivo. Al añadir o eliminar las cadenas a procesar del código, necesitamos que Babel sepa cuales son sus cambios, así que le pedimos las extraiga:

```
$ pybabel extract -F babel.cfg -k lazy_gettext -o messages.pot .
```

Nota: Ojo al punto final, no es una errata, tiene significado (directorio actual).

Aunque es improbable que dejemos de emplear las sustituciones perezosas (son las lazy_gettext), si dejasen de ser necesarias podríamos quitar la parte `«-k lazy_gettext»` del comando. Si queremos añadir un idioma que todavía no existe, entonces tenemos que crear un fichero inicial del mismo:

```
$ pybabel init -i messages.pot -d translations -l es
```

El parámetro `«-d translations»` dictamina que el nuevo fichero tendrá como destino la carpeta translations, para que tengamos todas las traducciones juntas. El parámetro `«-l es»` determina el lenguaje; en este caso es el castellano, así que si queremos otro diferente tenemos que buscar la abreviatura adecuada[6] según el proyecto que hemos empleado (flask-babel)[7,8], tendríamos que revisar la versión de *CLDR* empleada por Babel.

[6] http://babel.pocoo.org/en/latest/api/languages.html#module-babel.languages

[7] https://github.com/python-babel/flask-babel

[8] https://pythonhosted.org/Flask-Babel/

Si el idioma ya estaba creado, entonces no queremos que nos sobreescriba lo que ya teníamos traducido. Entonces, le pediremos nos actualice las referencias mediante:

```
$ pybabel update -i messages.pot -d translations
```

Añadirá las nuevas cadenas, y aquellas que ya no encuentre, cambiará su marca de `«#:»` a `«#~»`.

El único paso que nos quedará entonces será el de generar los ficheros que emplea Babel:

```
$ pybabel compile -d translations
```

Y eso es todo, ya podemos tener las últimas etiquetas traducidas de forma dinámica.

Instalación y configuración de Moodle como API Rest

Como se mencionó en los aspectos relevantes de la memoria, es necesario configurar *Moodle* para poder utilizarlo como una *API Rest*. A continuación se detalla como configurar la *API*.

En primer lugar debemos tener un usuario con el rol de administrador de la plataforma para poder acceder a la *Administración del sitio* para poder activar los servicios web que por defecto vienen desactivados. Deberemos dirigirnos a *Administración del sitio–Características avanzadas* y habilitar los servicios web. Una vez hecho esto deberemos activar el protocolo *Rest*, que básicamente es el protocolo seguido por una *API Rest*, el cual se accede mediante *Administración del sitio – Extensiones – Servicios Web – Administrar protocolos* y habilitamos dicho protocolo.

A su vez, para que podamos acceder a dichas funcionalidades, además de tener que estar el servicio web y el protocolo activado, lo usuarios deben tener una ficha o *token* el cual los identifique de manera única. Para generar desde nuestra *API* dichos *tokenes* nos dirigiremos a *Administración del sitio – Extensiones – Servicios web – Administrar tokens* y ahí generaremos los tokens para los usuarios. De esta manera, cada usuario tendrá un identificador único para realizar las peticiones correspondientes [1].

Tratamiento de los roles de los usuarios

Como se mencionó en los aspectos relevantes de la memoria, inicialmente se tenía una idea de realizar una comprobación de la base de datos para

obtener los roles de usuario. Posteriormente se decidió que obtener dichos roles directamente de *UBUVirtual* era más correcto.

Para ellos utilizamos las funciones proporcionadas por *Moodle* para realizar peticiones a nuestra *API Rest* (véase la sección de *Conceptos Teóricos*), que en este caso es UBUVirtual. En dicho listado de funciones ([2]) encontramos la función *core enrol get enrolled users*, la cual nos permitirá conocer los usuarios de la asignatura, así como su rol en la misma y más información variada de cada uno de los participantes. Dicha función nos muestra esta información en forma de diccionario *JSON* desde el que buscaremos al usuario correspondiente para así conocer su rol en la asignatura correspondiente. Dicha información nos es presentada con la estructura definida en la figura D.1:

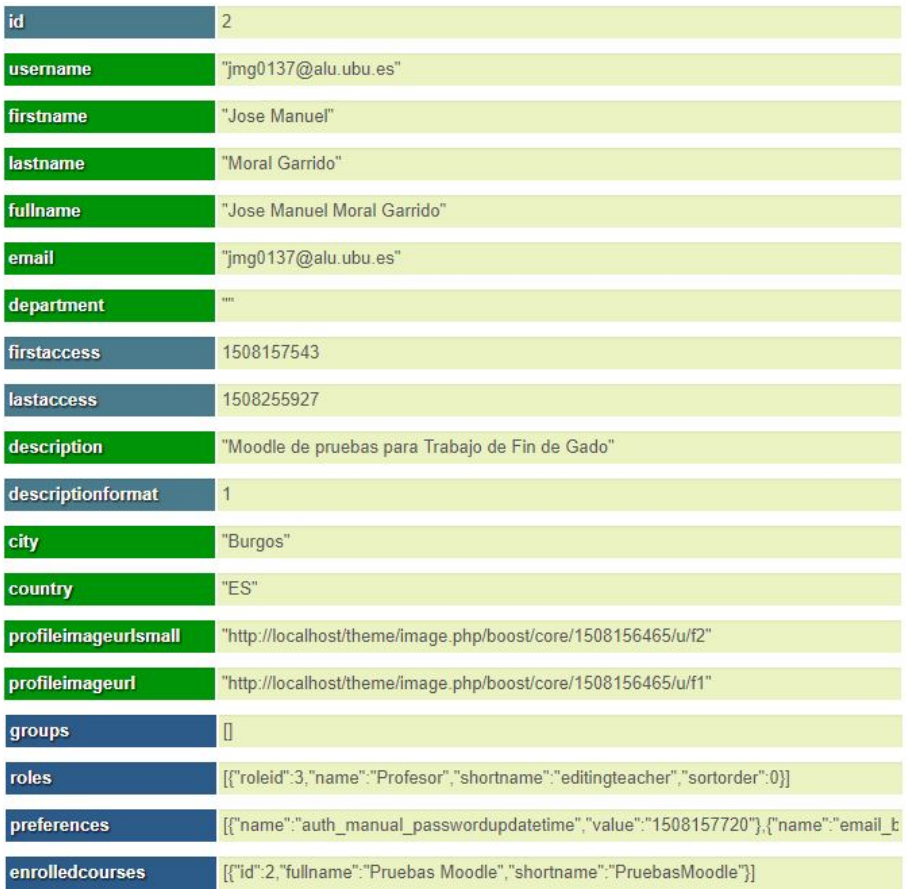

Figura D.1: Estructura de la información proporcionada por la API en formato JSON.

Como se puede apreciar en el campo *roles* nos encontramos con el rol correspondiente del usuario en cuestión, que en este caso es *Profesor* y el id de dicho rol es 3.

Obtención de los modelos

Ya que la idea inicial era la de obtener los modelos a través de *UBUVirtual* mediante recursos, cabe mencionar como conseguimos obtenerlos aunque la idea no prosperase.

Para ello, recurrimos de nuevo a las funciones *API Rest* siendo esta vez la función *mod resource get resources by courses* la elegida [2]. Dicha función nos ofrece la información de los recursos presentes en la *API* de UBUVirtual en los cursos correspondientes. En el caso de no seleccionar un curso en concreto, nos devuelve cada uno de los recursos a los que dicho usuario puede acceder. La información resultante tiene la estructura definida en la figura D.2:

Figura D.2: Estructura de la información proporcionada por la API en formato JSON.

De esta manera podemos acceder al nombre de recurso con su correspondiente extensión y comprobar que es del curso correspondiente mediante el campo *course*.

Pero posteriormente, la Universidad de Burgos nos proporcionó un servidor privado, de nombre «Arquímedes» en el que podemos desplegar nuestra *API* sin necesitar por ello todo lo mencionado anteriormente acerca de los albergar los modelos como recursos de *UBUVirtual*, ya que podremos albergarlos en nuestro servidor.

D.5. Encriptado y desencriptado de los modelos

Para otorgar seguridad a los modelos, hemos tenido que encriptar los mismos de manera que el modelo que se alberga en el servidor esté modificado de tal manera que alguien ajeno a la *API* que quiera obtener datos o modificar los datos guardados de los modelos sea incapaz.

Con el fin de realizar modificaciones en los modelos 3D para así preservar su seguridad y unicidad, se ha decidido generar una secuencia de números aleatorios con una estructura determinada. De esta manera, conociendo la semilla utilizada para la generación de los números, podremos codificar y decodificar nuestros modelos sin miedo a perder datos importantes de los mismos. Para obtener dichos números aleatorios hemos introducido en el código una función la cual dada un número, nos devuelve otro, con lo cual realizando esta operación un cierto número de veces, obtendremos una secuencia de números «aleatorios» (se encuentra entrecomillado porque los valores son aleatorios, pero si se conoce la semilla inicial siempre obtendremos la lista de valores en el mismo orden)[9].

Obtención de los números aleatorios

La manera de obtener números aleatorios en el caso de los vértices es la siguiente: 7 * número aleatorio anterior % 101, mientras que en el caso de las caras obtendremos los valores aleatorios de la siguiente manera: 7 * número aleatorio anterior % 11. Vemos que la diferencia reside en el valor máximo que puede alcanzar el número aleatorio.

Para la modificación de los valores de los vértices

Llegados a este punto tendremos que decidir cómo modificar los valores de los modelos, para lo cual hemos realizado un estudio de los tiempos que

[9] https://cdsmith.wordpress.com/2011/10/10/build-your-own-simple-random-numbers/

tarda el modelo en ser encriptado. La operación a realizar en cada caso será determinada por el tipo de los valores que se vayan a modificar.

Siendo la manera de generar los números aleatorios la mencionada en la sección D.5 y la manera de modificar los valores de los vértices: valor del vértice * (2 * número aleatorio obtenido), estaríamos en la encrucijada de elegir la cantidad de operaciones a realizar debido al gran volumen de datos que queremos modificar, por lo tanto hemos obtenido:

Para los modelos ASCII:

- Si modificamos todos los vértices que componen al modelo obtenemos un tiempo de: 12,0131 segundos
- Si modificamos solamente los vértices que ocupan posiciones pares del modelo (la mitad de operaciones) obtenemos un tiempo de: 10,2731 segundos

Para los modelos Binarios:

- Si modificamos todos los vértices que componen al modelo obtenemos un tiempo de: 2,7190 segundos
- Si modificamos solamente los vértices que ocupan posiciones pares del modelo (la mitad de operaciones) obtenemos un tiempo de: 2,0486 segundos

Una vez conocidos estos datos, decidimos modificar únicamente los valores que ocupan posiciones pares, ya que la encriptación del modelo es suficiente para conservar su seguridad como podemos observar a continuación y el tiempo de codificación es menor:

Teniendo un modelo inicial como el de la figura E.1:

Figura D.3: Modelo de partida.

Obtendremos un modelo encriptado como el mostrado en la figura D.6:

Figura D.4: Modelo de encriptado.

Como se puede apreciar, el modelo encriptado es lo suficientemente difuso como para poder obtener mediciones o datos del mismo en caso de que este fuera robado de la carpeta de almacenamiento de la aplicación. Pero a partir de aquí nos surge el problema relacionado con el redondeo de los decimales, así como de la cantidad de decimales que se devuelven al realizar un *casteo* a otra clase (por ejemplo de *str a float* en *Python*).

Tras realizar las operaciones de codificación del modelo, procedimos a comprobar que los valores obtenidos dividido entre los valores originales nos devolvieran el multiplicando (nuestro número aleatorio D.5) y es aquí cuando nos damos cuenta de que no podemos utilizar los valores en coma flotante de los vértices ya que al multiplicar o dividir, los valores de los mismo son

corrompidos por los redondeos y el número de decimales. Por ejemplo, para un multiplicando de $0,7$, al dividir el valor obtenido entre el valor inicial obtenemos que el multiplicando es 0,7129, con lo que podemos concluir que esta no es una manera viable de encriptar los modelos. A continuación mostramos el modelo de la figura E.1 desencriptado tras modificar sus vértices en la figura D.5:

Figura D.5: Modelo desencriptado utilizando los valores de los vértices.

Para la modificación de los valores de las caras

Tras el resultado nefasto de modificar los valores de los vértices, decidimos que podríamos alterar los valores de las caras formadas por los vértices, las cuales son enteras y no tendremos el problema de los decimales. En este caso, siendo la manera de generar los números aleatorios la mencionada en la sección D.5 y la manera de modificar los valores de las caras:

valor de la cara * (número aleatorio obtenido * 10) + (número aleatorio obtenido * 100) habiendo también realizado un estudio de los tiempos de codificación de los modelos, teniendo en cuanta que el número de caras en el modelo cogido de ejemplo es 248 999 mientras que el número de vértices es de 126 720:

Para los modelos ASCII:

- Si modificamos todos los vértices que componen al modelo obtenemos un tiempo de: 14,1245 segundos
- Si modificamos solamente los vértices que ocupan posiciones múltiplos de cuatro del modelo (la cuarta parte de operaciones) obtenemos un tiempo de: 11,1153 segundos

Para los modelos Binarios:

- Si modificamos todos los vértices que componen al modelo obtenemos un tiempo de: 2,9409 segundos
- Si modificamos solamente los vértices que ocupan posiciones múltiplos de cuatro del modelo (la cuarta parte de operaciones) obtenemos un tiempo de: 1,9845 segundos

Una vez conocidos estos datos, decidimos modificar únicamente los valores que ocupan posiciones múltiplos de cuatro. Con esta modificación, la encriptación del modelo es suficiente para conservar su seguridad (el modelo no se llega a mostrar en el navegador ya que no es capaz de dibujarlo) y el tiempo de codificación es menor. De este modo obtenemos tanto una mejor codificación en lo relacionado con la seguridad como de precisión de resultados tras la decodificación. Por lo tanto, concluiremos adjudicando a las **caras** la encriptación en lugar de a los **vértices**.

Obtendremos un modelo encriptado como el mostrado en la figura D.6:

Figura D.6: Modelo de encriptado.

D.6. Configuración de «Arquímedes»

Debido a que no tenemos los permisos necesarios para configurar el servidor personalmente, tenemos que depender de u operador que realice las operaciones que le sean indicadas. Esto conlleva una pérdida considerable de tiempo y por ello a llevado, entre otras razones, más tiempo del programado para dicha tarea. Por un lado detallaremos como hemos configurado nuestra máquina personal para posteriormente poder facilitar las instrucciones necesarias a nuestro operador. Por otro lado, incluiremos las modificaciones realizadas para poder adaptar la configuración de nuestra máquina a la del servidor «Arquímedes».

Configuración de nuestra máquina personal

Primero de todo mencionar que la máquina a configurar se compondrá de un sistema operativo *Linux* en su versión 16.04 [10]. Para poder empezar, primero tuvimos que elegir de qué manera queríamos desplegar nuestra aplicación y con qué herramienta, como se mencionó en la sección de *Técnicas y Herramientas* de la Memoria.

Primero instalaremos *Apache* en nuestra máquina con el comando:

```
apt-get install apache2
```

Paso siguiente instalaremos la librería *mod_wsgi*:

```
apt-get install libapache2-mod-wsgi-py3
```

Una vez instalado esto, deberá aparecernos en la dirección `«/var»` una carpeta llamada ***www***. En mi caso he introducido mi aplicación directamente en dicha carpeta, pero no es necesario, simplemente ha sido por comodidad ya que posteriormente podremos referenciar cualquier ruta (teniendo que ser ésta absoluta).

Seguidamente y dependiendo de si estamos desplegando nuestra aplicación en *Windows Linux*, deberemos ejecutar los comandos desde la carpeta raíz del proyecto:

```
python3 -m venv mi_venv
```

```
pip3 install -r requirements.txt
```

Esto es debido a que si trabajamos desde *Windows* nuestro entorno virtual no tendrá la carpeta `«../mi_venv/bin»` y dicho comando creará ese directorio. Este paso se debe a que necesitamos meter en dicho directorio el archivo *activate_this.py* [11]https://github.com/pypa/virtualenv/blob/master/virtualenv_embedded/a situado en la carpeta raíz del proyecto. Dicho archivo contiene la configuración necesaria para que *Apache* pueda obtener las dependencias instaladas en el entorno virtual ya que, por defecto, dichas dependencias deberían estar instaladas en la máquina. nuestra aplicación deberá constar de un archivo con extensión *.wsgi* que será el que le diga a *Apache* lo que debe hacer con nuestra aplicación y cómo hacerlo.

En este archivo (situado en la raíz del proyecto) se deberá incluir nuestra aplicación como variable del sistema, así como realizar la lectura y ejecución del archivo *activate_this.py* mencionado anteriormente. Finalmente

[10] https://es.wikipedia.org/wiki/Ubuntu#Ubuntu_16.04

[11] text

dicho archivo de configuración contendrá la ejecución de nuestro programa (*APP.run*) [12]. Importante mencionar que en la línea de nuestro fichero *.wsgi* `from MyApp import APP as application`, la última parte (*as application*) es necesaria para el *Apache* conozca qué va a ser la aplicación.

Posteriormente nos dirigiremos a la carpeta: `«/etc/apache2/sites-available»` en la que deberemos modificar el archivo *000-default.conf* que contiene la configuración por defecto de *Apache*. Dicha configuración será sustituida por la proporcionada en el archivo *host_config.txt* situado en la carpeta raíz del proyecto. Para comprender mejor este archivo de configuración, mencionaremos qué es cada uno de sus componentes al margen de los comentarios que trae el archivo por defecto:

- **ServerName**: Dirección en la que el servidor se ejecutará.
- **ServerAlias**: Alias de la dirección anterior.
- **ServerAdmin**: Administrador del servidor.
- **DocumentRoot**: Carpeta contenedora de los archivo que componen la aplicación. Aquí pondremos la dirección absoluta de la carpeta *MyApp* incluida en nuestro proyecto.
- **WSGIScriptAlias /«dirección elegida»**: Ejecución en la «dirección elegida»(*/visor3d* por ejemplo ejecutara el contenido de *MyApp.wsgi* para que se pueda acceder desde *ServerName/visor3d*) utilizando la ruta absoluta del fichero *.wsgi* y encontrar las dependencias necesarias.
- **Directory «directorio»**: conjunto de acciones que serán aplicadas al directorio especificado, figurando aquí la dirección absoluta de la ubicación de nuestro proyecto.

Un vez hecho esto, deberemos hacer *reset* al servicio de apache con el comando:

```
service apache2 restart
```

Siguiendo estos pasos, podremos acceder a nuestra aplicación desde nuestro navegador en el *localhost/«nombre dado»* si estamos en la máquina configurada y desde el

[12] https://gist.github.com/LeZuse/4032238

D.7. Compilación, instalación y ejecución del proyecto

D.8. Pruebas del sistema

Apéndice E

Documentación de usuario

E.1. Introducción

E.2. Requisitos de usuarios

E.3. Instalación

E.4. Manual del usuario

En este apartado, se explicará cada una de las funcionalidades disponibles de nuestra aplicación.

Barra de navegación

Este elemento se encontrará en todas las vistas de la aplicación a excepción de la página de *login*. Dicho elemento nos proporcionará una navegación mas rápida y fluida a cada uno de los elementos incluidos en nuestra barra de navegación. A su vez, este elemento nos dotará de una herramienta conocida como *miga de pan(breadcrumb)*. Dicha herramienta nos ayudará a encontrar el camino de vuelta a cualquier pantalla por la que hayamos pasado sin necesidad de volver al inicio y empezar de nuevo. Podemos apreciar en la figura **??** la barra de navegación inicial, y en la figura **??** la barra de navegación con la utilización de *migas de pan*. (IMAGEEEEEEEEEEEEEEEEN) (IMAGEEEEEEEEEEEEEEEEN)

Nuestra barra de navegación diferencia también entre usuarios con diferente rol. Dependiendo de nuestro rol (alumno o profesor), podremos acceder

al apartado de subida de modelos (sección E.4). La diferencia entre las barras de navegación según el rol del usuario se pueden apreciar en las figuras **??** y **??**. (IMAGEEEEEEEEEEEEEEEEN) (IMAGEEEEEEEEEEEEEEEEN)

Página de inicio

Una vez se haya logueado correctamente, según sea su rol en la asignatura correspondiente, se encontrará con dos estructuras diferentes. Por un lado, si su rol es el de **profesor**, se topará con dos bloques que distinguen entre **Modelos** y **Ejercicios** como se representa en la figura **??**. (IMAGEEEEEEEEEEEEEEEEN)

El bloque de **Modelos** nos llevará a la estantería de los modelos en la que podremos elegir qué modelo visualizar, como se explica en la sección **??**. Por otro lado, el bloque de **Ejercicios** nos llevará al listado de los modelos disponibles para la realización de ejercicios, como se muestra en la sección E.4.

Por otro lado, si su rol es el de **Alumno**, se encontrará con un único bloque llamado **Modelos** que nos redirigirá a la estantería de modelos en la que podremos elegir el modelo a visualizar. Sigue la estructura de la figura **??**. (IMAAAGEEEEEEEEEEEEEN)

Repositorio de ejercicios

En esta página podremos observar el listado de los modelos disponibles sobre los que podremos realizar ejercicios. Aquí podremos elegir qué modelo utilizar simplemente pinchando sobre este y automáticamente se abrirá una página como la mostrada en la sección E.4. Es entonces cuando encontraremos el listado de ejercicios disponibles para dicho modelo. Esta página sigue la estructura de la figura **??**. (IMAGEEEEEEEEEEEEEEEEN)

Aunque siga la estructura de la estantería de los modelos, existe una pequeña diferencia. Esta diferencia reside en que desde la estantería de modelos podemos eliminar un modelo y desde la estantería de ejercicios no.

Repositorio de ejercicios para cada modelos

Es en esta página donde encontraremos el listado de ejercicios disponibles para un modelo en concreto. Dicha página tiene la estructura de la figura **??**. (IMAGEEEEEEEEEEEEEEEEN)

Como se puede apreciar, tenemos la imagen del modelo a un lado de la página y al otro el listado de ejercicios disponibles para dicho modelo.

Cuando naveguemos por la lista de ejercicios veremos como se ilumina cada ejercicio al paso del ratón como se muestra en la figura **??**. (IMAGEEEEEEEEEEEEEEEN)

A su vez, se resaltan tres botones en el ejercicio correspondiente con distintas funcionalidades. Por un lado tenemos el botón de **Editar**, el cual nos redirigirá al visor de ejercicios mencionado en la sección E.4. Por otro lado tenemos el botón de **Eliminar** con el nos aparecerá un diálogo de confirmación para la eliminación de dicho ejercicio como el mostrado en la figura **??**. (IMAGEEEEEEEEEEEEEEEEN)

Por último tenemos el botón de **Editar nombre** con el que podremos editar el nombre del ejercicio a nuestro gusto siempre y cuando respetemos las reglas de nombres correspondientes. Cuando pinchemos en dicho botón nos aparecerá un cuadro como el mostrado en la figura **??**, el cual nos permitirá guardar los cambios o cancelar el proceso. (IMAGEEEEEEEEEEEEEEEEN)

Si la convención de nombres no es respetada, nuestra aplicación mostrará una alerta como la de la figura **??**. (IMAGEEEEEEEEEEEEEEEEN)

Visor

Acciones comunes en el visor de modelos

Lo miso que en lo de Alberto.

Visor de ejercicios

Este visor únicamente será visible para los usuarios con rol de **profesor** debido a que aquí se realizarán las plantillas de corrección para determinados ejercicios. Se compondrá de funcionalidades ampliadas para facilitar al usuario la corrección y visualización de ejercicios.

Acciones comunes en el visor de ejercicios

Dicho visor se compondrá de las mismas funcionalidades que en visor de modelos visible para los alumnos (añadir, borrar, editar, etc), con la diferencia que en este no tendremos la posibilidad de importar y exportar modelos. En este caso, los ejercicios realizados por el profesor será almacenados en el servidor, pudiendo editar estos en cualquier momento(como se puede ver en la sección E.4). La diferencia de este visor con respecto al anterior reside en la aparición de tres botones nuevos, uno de «Guardar», otro de «Restablecer

Datos» y otro de «Salir». El visor mencionado tendrá la estructura de la figura **??**. (IMAGEEEEEEEEEEEEEEEEEEEEEEEEEEN)

El botón de guardar simplemente guardará los cambios realizados en las medidas y anotaciones, mientras que el botón de salir nos devolverá a la pantalla de ejercicios para cada modelo en el caso de no haber realizado cambios(sección E.4). Si cuando pulsamos este botón («Salir») se detectan cambios, nos aparecerá un diálogo de confirmación en la pantalla como el mostrado en la figura4 **??**. (IMAGEEEEEEEEEEEEEEEEEEEEEEEEEEN)

Respecto al botón de «Restablecer Datos», su función consiste en que teniendo el profesor un ejercicio hecho, y sobre éste carga el ejercicio de un alumno, posteriormente sea capaz de volver a su ejercicio inicial sin tener que salir del ejercicio y volver a entrar.

De esta manera podremos salir del visor de ejercicios descartando o guardando los cambios, o simplemente cancelar el proceso de salida del ejercicio.

Subir modelos

Finalmente, nos encontramos con la página de subida de modelos, la cual solamente estará visible en la barra de navegación (sección E.4) para los usuarios con rol de **profesor**. Nos dirigiremos al apartado «Subir» en la barra de navegación y nos redirigirá a la página de la figura **??**. (IMAAAAAAGEEEEEEEEEEEN)

A la hora de subir un modelo, pincharemos en «Selección de archivo» y nos aparecerá un diálogo en el que seleccionaremos el modelo con extensión «*.ply*» correspondiente. Tras seleccionar el archivo pulsar en aceptar, pincharemos en el botón de **Subir** y tendremos que esperar a que nos aparezca el cuadro de confirmación como el de la figura **??**. (IMAAAAAAGEEEEEEEEEEEN)

Si no realizamos la espera correspondiente, la cual puede demorarse en función del tamaño del modelo, la subida de dicho modelo podría verse alterada. Esta alteración se debería a que interrumpimos el proceso de encriptación de los modelos (véase la secciónD.4), pudiendo conllevar a fallos en la carga de los modelos.

Por lo tanto, partiendo de un modelos como el de la figura E.1

Figura E.1: Visualización del modelo encriptado correctamente.

Obtendremos un resultado como el de la figura E.2 por no dejar a la aplicación realizar la encriptación completa.

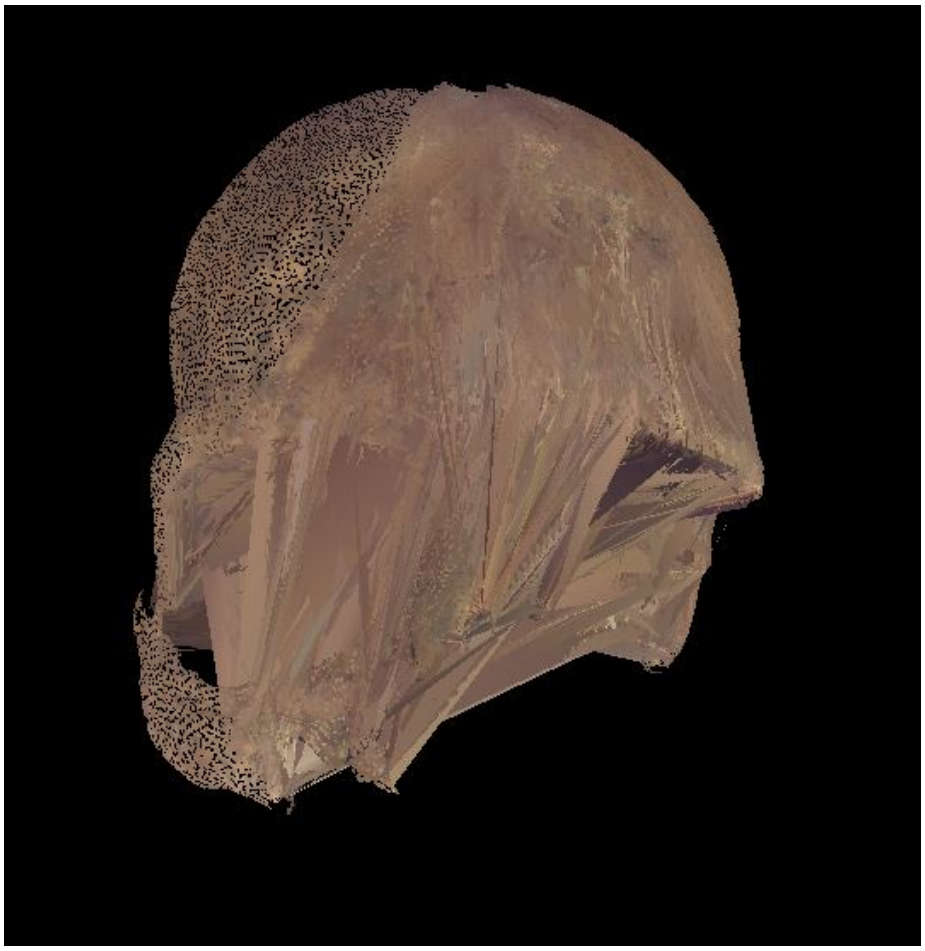

Figura E.2: Visualización del modelo cargado con una encriptación incompleta.

A su vez, podrían darse casos en los que ni siquiera se termine de subir el modelo elegido.

Bibliografía

[1] Fersacom. Como configurar api rest-full en moodle. https://www.fersacom.es/configurar-api-rest-full-moodle/, 2017.

[2] Moodle. Web service api functions — moodle. https://docs.moodle.org/dev/Web_service_API_functions, 2017.

[3] Ken Schwaber and Mike Beedle. *Agile software development with Scrum*, volume 1. Prentice Hall Upper Saddle River, 2002.

[4] Alberto Vivar. 3D viewer: Trabajo fin de grado ingeniería informática. https://github.com/Alberto-Vivar/3D-Viewer, 2016-2017.

[5] Wikipedia. Heroku – wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Heroku, 2017.