PRÁCTICA FINAL PROGRAMACIÓN III

jLLM

CURSO 2023/2024

Fecha límite de entrega: 13/12/2023

Facultad de Ciencias - Universidad de Salamanca

INTRODUCCIÓN

Una empresa se dispone a realizar una copia barata del famoso ChatGPT de OpenAI, para eso le contrata a usted, estudiante de Programación III, con el objetivo de implementar una aplicación similar a lo que ofrecía en sus primeras versiones ChatGPT, pero en Java, inicialmente de consola y con poco presupuesto. Lo llamarán **jLLM** (de ***j**ava **L**arge **L**anguage **M**odels*) para no tener litigios con OpenAI. Los requisitos principales de la aplicación y su funcionalidad se pueden resumir en los siguiente puntos:

- **Conversación con un LLM**: el prototipo dispondrá de una opción para establecer una conversación con un LLM, que será más o menos hablador, dependiendo del objeto que se cargue en la inicialización del programa. Un buen diseño de POO será clave para extender el asistente en el futuro, implementando nuevos LLM que surjan, cambiar la UI y todo esto sin necesidad de cambiar el resto de la aplicación. Por defecto se deberán implementar al menos dos LLM de los que siguen:
  - **FakeLLM**: se limitará a contestar con frases predefinidas de una lista que reside en memoria. La frase se escogerá al azar y con una lógica que define el desarrollador de la aplicación (e.g., si el mensaje contiene "hola" contestar con un saludo, etc.). Se deja a libre elección del estudiante esta parte.
  - **RandomCSVLLM**: al igual que el anterior se limitará a contestar con frases aleatorias de distintas temáticas que obtiene de un fichero CSV. Este CSV tiene como primer campo el tipo de frase, el segundo la longitud y el tercero la propia frase. Se os proporciona un ejemplo en Studium. Se debe controlar si hay líneas inválidas en el fichero y descartarlas. Ejemplo:

    ```
    refran,31,¡A quien madruga Dios le ayuda!
    saludo,31,Mis saludos perfect@ deconocid@
    ```
  - **SmartLLM [Extra]** se podrá implementar de forma adicional otro LLM más para el apartado "*One more thing*" de la *keynote* que quiere dar la empresa (se detalla más adelante). Esta parte será opcional y se valorará positivamente.
- **Gestión de conversaciones CRUD**: al igual que sucedía en ChatGPT, el prototipo dispondrá de varias opciones para consultar y eliminar conversaciones pasadas. Téngase en cuenta que una conversación es una colección de mensajes ordenados por tiempo.

- **Persistencia**: el programa, al igual que ocurre con ChatGPT, deberá ser capaz de recuperar el estado de la última ejecución. Para este fin se utilizará el mecanismo de serialización en Java para persistir la información del modelo. Se cargará en el inicio de la aplicación y se guardará al salir de esta.
- **Exportación/Importación**: ya que se distribuirá en Europa y para facilitar el cumplimiento de la ley de protección de datos, el programa deberá permitir exportar e importar todas las conversaciones y sus respectivos mensajes en formato JSON o XML. Esta parte también deberá ser extensible ya que en el futuro se pretende incluir la posibilidad de exportar en otros formatos.
- **Interfaces de usuario**: el prototipo deberá contar con una interfaz por consola pero también estar abierto a extensión ya que en futuro se quieren implementar otras interfaces (GUI, voz, etc). Un buen diseño de POO es vital en esta parte. Se deberá implementar al menos una de las siguientes interfaces de usuario:
    - Interfaz de consola simple: Con un menú para distintas opciones. Será la opción por defecto de la aplicación.
    - Interfaz de consola con TTS **[Extra]**: Esta interfaz incluirá el uso de bibliotecas externas que permitirán narrar cada una de las opciones, además de mostrar la información por pantalla como la versión anterior. Otra característica más para la keynote de la empresa y su parte *"One more thing"*
- **Aplicación robusta y resistente a fallos**: la funcionalidad del prototipo debe de haber sido probada manualmente, revisando toda la funcionalidad implementada (no se realizarán tests unitarios automáticos ya que no han sido objeto de esta asignatura). Si una característica no funciona la empresa no la valorará.
- **Arquitectura MVC**: un punto clave del desarrollo de la aplicación es que siga las guías de diseño del estilo arquitectónico Modelo Vista Controlador tratadas a lo largo del curso en la asignatura.

GUÍA DE DISEÑO E IMPLEMENTACIÓN

A continuación se proporcionan una serie de guías de diseño que facilitarán el desarrollo de la aplicación.

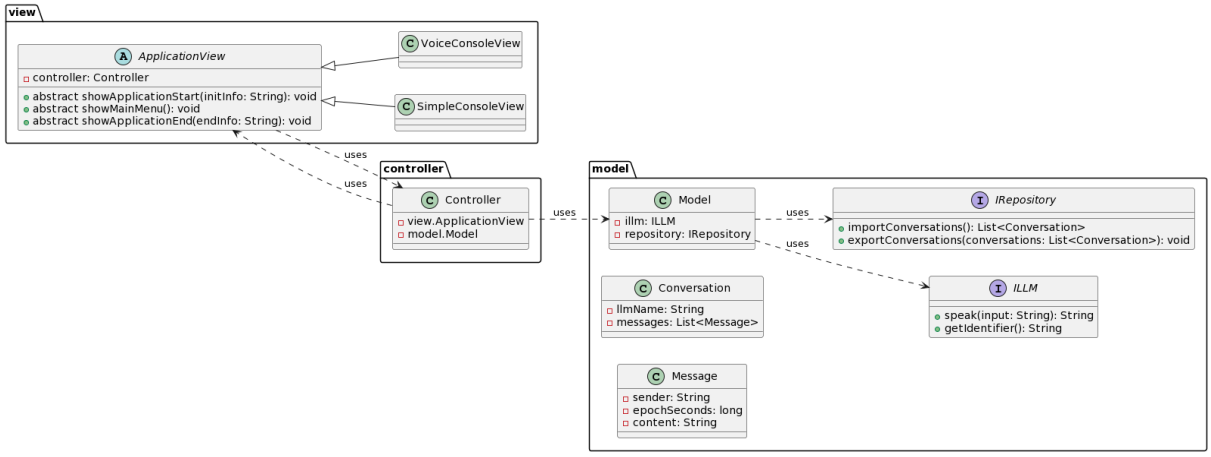
Si no comprende algo o necesita alguna aclaración, escríbalo por el foro de dudas de Studium habilitado para tal fin, resolverá su duda y la de sus compañeros.

Arquitectura MVC:

La aplicación deberá presentar una arquitectura MVC. Esta se deberá armar en el inicio de la aplicación estableciendo los objetos que tomarán parte en la aplicación. Siempre la aplicación trabajará con las clases base *View*, *Model* y *Controller*. La instancia de las clases concretas será lo que cambiará dependiendo de los parámetros pasados por consola en String[] args. Para armar el MVC se podrá utilizar el método main.

- ☐ **Revise el ejemplo de MVC con herencia e interfaces proporcionado.**

Un diagrama de UML general. no completo y de alto nivel de las clases principales de la aplicación podría ser el siguiente:



En el esquema no se muestran todas las clases del proyecto (esto se lo dejamos a usted), pero sí una guía de cuáles podrían ser las principales. Nótese que el controlador tiene una referencia a una clase abstracta y que la clase Model tiene atributos de tipos que son Interfaces. Esto significa que el controlador solo emplea los métodos de dicha superclase *ApplicationView* y el modelo sólo podrá hacer uso de los métodos de las interfaces *IRepository* y *ILLM*. Esto abstrae dichas partes de la aplicación y hace que se puedan sustituir fácilmente. Las clases concretas que ocupen dichos objetos serán configuradas en la inicialización del programa en el *main*, consiguiendo que la lógica de negocio de la aplicación se mantenga intacta, a pesar de cambiar dichos objetos.

En el diagrama anterior se muestra que las clases heredan directamente de *ApplicationView*, sin embargo la clase de voz podría implementarse siendo una clase hija de la UI de consola normal (sería una “clase nieta” de *ApplicationView*), ya que añade comportamiento a la misma. No obstante queda a su elección dicho diseño.

Paquete view:

ApplicationView: se trata de una clase abstracta que tiene los siguientes métodos abstractos y atributos:

Tendrá un atributo:

- **`Controller`** `controller.`

Tendrá como mínimo los siguientes métodos abstractos:

- `void` **`showApplicationStart`**`(String initInfo): da un mensaje de bienvenida`
- `void` **`showMainMenu`**`(): inicia el menú principal de la aplicación.`
- `void` **`showApplicationEnd`**`(String endInfo): finaliza la aplicación ordenadamente.`

La funcionalidad de los métodos abstractos la implementaran las clases que heredan de View, en este caso podrían ser las siguientes:

- **Interfaz de Consola Simple:** será una clase que hereda de *View* y que implementa los métodos anteriores, elegid el nombre que prefiráis. Será la encargada de mostrar los mensajes de inicio de la aplicación y de iniciar el menú con las opciones del programa al usuario. Gestiona toda la interacción con el usuario y las opciones de la aplicación que son:

    - **Nueva conversación**: esta opción permite mantener una conversación con el modelo cargado actualmente. Se deberá crear una nueva conversación en el modelo e ir añadiendo los mensajes a medida que se intercambian mensajes. Para salir de esta opción se detectará la cadena **`/salir`** y se volverá el menú principal y se dará por finalizada la conversación.
      Para implementar esta parte se podría crear inicialmente un objeto conversación actual en el modelo al que se le irán añadiendo mensajes, tanto los escritos por el usuario como los obtenidos como retorno de pasarlos al agente. Queda a su elección como implementar esta lógica, siempre y cuando cumpla con el MVC.

      ```
      Yo [13/11/23: 20:22:22]: Quiero que me hables de ti

      Agent [13/11/23: 20:23:00]: Por supuesto, soy…

      (…)

      Yo [13/11/23: 20:22:22]: /salir

      MENÚ PRINCIPAL …
      ```

- **Menú CRUD**: se deberán implementar las siguientes operaciones CRUD:
  - **Listar conversaciones**: se podrán listar las conversaciones mantenidas. El listado será similar al siguiente:

```
1. EpochUnixSecondsInicio | Número de mensajes | Primeros 20 caracteres del primer mensaje
2. EpochUnixSecondsInicio | Número de mensajes | Primeros 20 caracteres del primer mensaje
3. EpochUnixSecondsInicio | Número de mensajes | Primeros 20 caracteres del primer mensaje
```

    Seguidamente, se le indicará si quiere mostrar todos los mensajes de una de las conversaciones o salir. Si elige lo primero, se mostrarán ordenados dichos mensajes como se produjeron:

```
Conversación del DD:MM:AA HH:MM:SS:
Yo [11/11/23: 20:22:22]: <mensaje>
Agent [11/11/23: 20:23:00]: <mensaje>
(…)
```

  - **Eliminar conversación**: Esta es una opción en combinación con la anterior de tal forma que se pueda eliminar la conversación elegida por un número mostrado en el listado.

- **Menú exportación**: se deberá poder exportar todas las conversaciones y sus mensajes a un fichero en formato JSON o XML (output.json, output.xml) dependiendo del objeto IRepository que esté cargado. También se deberá poder importar de un fichero con el nombre input.xml o input.json con el mismo formato colocado en la carpeta /jLLM del escritorio del usuario.

- **Interfaz de Consola Con Voz:** tendrá la misma funcionalidad que la anterior pero además reproduce las opciones de menú principal con la biblioteca TTS proporcionada. Reproduce las opciones al mostrar el menú y al seleccionar una opción concreta del menú. El resto de información no es necesario que se narre.

La instanciación de una u otra clase en el inicio dependerá de si se pasa por parámetro en el main el parámetro -voz o no. Se deberá comprobar los argumentos pasados por consola a la hora de construir el MVC.

Paquete controller:

El controlador será un mero intermediario y orquestador en esta aplicación. Se ocupará principalmente de:

- La inicialización de la aplicación para devolverla a su estado anterior si existiera, indicando al modelo que restaure la información e indicando a la vista que muestre un mensaje con el resultado de dicha operación.
- Indicar a la vista que muestre el menú principal.
- Recibir órdenes desde la vista y trasladarlas al modelo, devolviendo los retornos correspondientes o propagando las excepciones correspondientes.
- Finalizar la aplicación indicando al modelo que guarde el estado de la misma y finalmente indicando a la vista que muestre un mensaje si todo ha ido bien o se ha producido algún error.

Paquete model:

Existirá una clase que se corresponde con el modelo y que debe tener, al menos, los siguientes atributos:

ILLM: se trata de un objeto que implementa la interfaz ILLM que será proporcionado por el constructor al modelo a la hora de crear el Model del MVC en el método main. Esto dependerá de una opción recibida por String[] args. Esta interfaz tiene 2 métodos: uno denominado speak que admite un String como argumento y que retorna un String como retorno y otro llamado getIdentifier que devuelve una String con el identificador del LLM. El modelo solo se comunica con objetos de este tipo. Se deberán implementar objetos que implementen esta interfaz como se indicaba en la introducción, podéis darles los nombres que queráis.

IRepository: se trata de un objeto que implementa la interfaz IRepository que permitirá exportar o importar las conversaciones del modelo. Solo tiene dos métodos denominados import, que retorna una lista de conversaciones, y export, que exporta una lista de conversaciones que hay en Model. Los detalles de implementación los tiene la clase concreta. Se deberán desarrollar objetos que implementen esta interfaz como se indicaba en la introducción, podéis darles los nombres que queráis.

Clases del modelo de dominio:

Una clase POJO que represente un mensaje Mensaje:

- Emisor
- Fecha en UnixEpoch: segundos transcurridos desde el inicio de la época Unix hasta este momento. Ver la clase Instant de Java 8 y su método getEpochSecond().
- Contenido

Una clase POJO que represente una conversación con, al menos, los siguiente campos:

- Nombre del LLM empleado en la conversación: el identificador
- Mensajes: un listado de mensajes
- Fecha inicio en segundos
- Fecha fin en segundos

Una clase Frase para el RandomCSVLLM

Resto de clases que se vean necesarias para el desarrollo de la lógica de negocio de la aplicación. **El proyecto no está limitado a las clases anteriores, solo indica que al menos deberían existir dichas clases o similares**.

NOTAS IMPORTANTES SOBRE EL CICLO DE VIDA DE LA APLICACIÓN

Ejecución de la aplicación y configuración de MVC:

Al ejecutar la aplicación por consola, el usuario podrá pasarle parámetros que definirán cómo se ejecutará la aplicación. Como sabe, los parámetros pasados se encontrarán en String[] args del método main. Deberá controlar si se pasan los siguientes parámetros:

java -jar jllm.jar repository model vista

Donde repository podrá ser:

> **xml**: el IRepository del modelo será en ese caso una clase que implementa la exportación a XML.
>
> **json**: el IRepository del modelo será en esta caso una clase que exporte a JSON. Será la opción por defecto.

Donde model podrá ser:

> **fake**: será el LLM fake y aleatorio

csv: será el LLM que saca los valores de un CSV situado en la carpeta de la aplicación. Si no se encuentra el fichero se mostrará un error en la inicialización.

smart: opcional, si se hubiera implementado

Donde vista podrá ser:

consola: opción por defecto de la aplicación.

voz: opcional si se hubiera implementado.

De este modo se deberá armar en el *main* un MVC que depende de esos parámetros de entrada. En el caso de la vista crear la vista correspondiente y en el caso del modelo crear el modelo pasándole dos objetos que implementen IRepository y ILLM respectivamente y de acuerdo a los parámetros indicados al ejecutar la aplicación.

Inicialización de la aplicación:

La aplicación deberá intentar cargar su estado de un fichero binario llamado jLLM.bin que se encuentra en la carpeta de la aplicación, llamada jLLM y que se encuentra en la carpeta jLLM del escritorio del usuario (se deberá crear esta carpeta manualmente antes del uso de la aplicación). El programa comprobará si existe dicho fichero y de ser así notifica al usuario cuantas conversaciones se han cargado con un mensaje, si no existe el fichero notifica al usuario indicando que no ha encontrado un fichero y que parece que es la primera ejecución del programa. Utilizar el mecanismo de serialización de Java para este fin.

Finalización de la aplicación:

La aplicación deberá volcar la información que tenga almacenada en el modelo, en este caso las conversaciones a un fichero que pueda cargar en el inicialización que se ha mencionado anteriormente. Utilizar el mecanismo de serialización de Java para este fin.

NOTAS SOBRE RECURSOS NECESARIOS EN LA APLICACIÓN

Generación de un número aleatorio: clase Random.

Gestión de tiempos: clase Instant y DateTimeFormatter

JSON y XML: bibliotecas disponibles en los apuntes.

Investigad cómo utilizarlas.

ONE MORE THING

Esta sección es para estudiantes que quieran profundizar en la realización de la práctica y es totalmente opcional y voluntaria, si no tiene interés no siga leyendo y salte a la siguiente sección. Se valorará muy positivamente su implementación y os dará la oportunidad de aplicar vuestros conocimientos a algo en boga.

Extras para la keynote de la empresa, en concreto su última parte "*One more thing…*":

- **Interfaz con voz TTS**: se implementará otra view que, al mismo tiempo que muestra la información de las opciones de la vista, narra dichas opciones empleando una biblioteca de Text To Speech (TTS) que interactúa con el TTS nativo del SO subyacente. La biblioteca que se propone utilizar es la siguiente:
  - https://github.com/jonelo/jAdapterForNativeTTS/tree/master

  En el apartado de **releases** del repositorio, en la sección de **assets** podéis encontrar el Jar que debéis importar en el proyecto para utilizar la biblioteca:
  - https://github.com/jonelo/jAdapterForNativeTTS/releases

  En el README del repositorio es posible ver las formas de obtener las distintas voces de vuestro sistema. En esta parte se permite que no sea independiente de plataforma y que hagáis lo necesario para que se obtenga una voz TTS. En caso de no obtener ninguna voz, el programa no emitirá ninguna narración. Revisad los requisitos de cada sistema operativo también disponibles en el README del proyecto. Si tenéis dudas de cómo abordar esta parte planteadlas en el foro o por correo electrónico. Es posible que haya que realizar pausas entre narraciones.

- **SmartLLM**: se debe implementar un ILLM que utilice una biblioteca externa que se comunique con un LLM desplegado con el software Ollama. Para instalar este software hay 3 opciones[1]: Linux, macOS o WSL2 de Windows (los que tengáis windows os tendréis que instalar WSL2 y Ubuntu). La instalación de Ollama la tiene aquí: https://ollama.ai/download Una vez instalado Ollama, se deberá instalar un modelo, en nuestro caso escogeremos **mistral**. Para instalarlo con las última versión bastará utilizar los siguientes comandos:

[1] Hay una cuarta opción, inocua para tu sistema y muy sencilla, ejecutarlo en un Google Colab como este, es la opción externa. Utilizad esta opción si vuestro sistema no tiene mucha RAM.

```
# Esta orden inicia el servidor en segundo plano

ollama serve &

# Esta orden descarga, ejecuta e instala el modelo (ojo son 4GB de espacio y se necesitan incluso 8GB de RAM)
En los equipos de las aulas podéis probarlo.

ollama run mistral

# Una vez cargado podremos salirnos de la interfaz interactiva pulsando en Ctrl+D o

# bye. OJO El servidor sigue encendido en el host 127.0.0.1 puerto 11434.

# Para ver si se está ejecutando en Ubuntu o WSL2 podéis utilizar

pgrep ollama

# Y para matar el proceso (por el ID que devuelve el comando anterior)

kill PIDNUMBERDEVUELTO
```

Por tanto, ahora solo quedaría añadir la biblioteca y ver cómo funciona. Esta biblioteca simplemente permite hacer llamadas y consumir la API que expone Ollama con el servidor que despliega. El esquema de funcionamiento es el siguiente:

La página del repositorio es la siguiente, pero os dejamos un ejemplo ya que funciona.

https://github.com/amithkoujalgi/ollama4j

Podéis bajar el jar de aquí en la columna download, clicando en jar:

https://s01.oss.sonatype.org/#nexus-search;quick~ollama4j

Esta biblioteca tiene otras dependencias como Jackson, SLF4J, etc. **Tenéis una carpeta en studium con todos los jar que deberéis añadir para la sección "One more thing"**. Tenéis un ejemplo aquí de cómo usarlo:

```
String host = "http://localhost:11434/";
OllamaAPI ollamaAPI = new OllamaAPI(host);
String msg = "Hola, ¿cómo estás?";
String response = ollamaAPI.ask("mistral",msg);
System.out.println(response);
```

Aseguraros de que funciona de forma aislada (en un proyecto de ejemplo) y luego ya integrarlo en la aplicación. La generación puede tardar, dadle tiempo.

REQUISITOS TÉCNICOS Y RECOMENDACIONES

- La aplicación deberá poder ejecutarse en **Linux, Windows y macOS**.
- La aplicación es **multiplataforma** por tanto, no se debe hacer referencia a ningún aspecto de un sistema operativo concreto.
- La aplicación debe presentar una **E/S por consola robusta**.
- Se deben **controlar las posibles excepciones** y gestionarlas correctamente.
- Se deben seguir los **principios de diseño del MVC**, respetando las responsabilidades de cada una de las partes de la aplicación.
- Se debe utilizar una **sintaxis camelCase**, propia de Java.
- **Revisad los proyectos proporcionados** a lo largo del curso y las indicaciones proporcionadas en este enunciado.
- Si se presentan **dudas**, **dificultades** o necesitáis **aclaraciones** sobre el enunciado, lo mejor es **consultarlas con los profesores de la asignatura**.
- Si se trata de *bugs*, **tras haber invertido un mínimo de tiempo y esfuerzo** en resolverlos, los podéis consultar también al profesorado. **Saber cómo depurar el código es una competencia que hay que adquirir en esta asignatura**.

REQUISITOS DE LA ENTREGA EN STUDIUM **(MUY IMPORTANTE)**

Estos requisitos son obligatorios, la ausencia de alguno de ellos supondrá un 0 suspenso en este trabajo.

- **Fecha límite de entrega: 13 diciembre de 2023 23:59.**
- **Proyecto completo en Netbeans**, JDK mínima 17 con las bibliotecas jar incorporadas. Se deberá entregar en un zip con la nomenclatura **Apellido1Apellido2Nombre.zip.**
- **Informe del trabajo en PDF con resultados de la ejecución del programa y unas mínimas explicaciones de cada sección que demuestren la realización del trabajo**. El fichero puede tener el nombre que queráis. Se debe pensar en este informe como un manual de usuario que acompaña el software con explicaciones de la funcionalidad y capturas de pantalla del resultado de la ejecución de las funcionalidades del proyecto.
  Repetimos, el informe es un requisito, sin él no se evalúa la práctica. No se debe incluir un "*copy paste*" del código en el informe (ya lo estáis subiendo con el proyecto). No debe haber capturas de pantalla sin explicación y no se debe incluir de nuevo el enunciado completo. No se evalúa al peso, esto busca demostrar que habéis ejecutado el software y que domináis lo que habéis entregado. Posible esquema de contenidos del informe:
  1. Portada. Nombre,DNI etc.
  2. Introducción: breve explicación
  3. Funcionalidad 1: demostración y explicación
  4. Funcionalidad 2: demostración y explicación
  5. …
  6. Problemáticas, aspectos a mejorar, posibles líneas futuras.

RÚBRICA PARA LA PRÁCTICA

Característica de la aplicación	%	Excelente (9-10)	Notable(7-8)	Suficiente (5-6)	Deficiente (0-4)
MVC *scaffolding*	20%	El proyecto cumple con los conceptos de diseño MVC y no tiene errores.	El proyecto cumple con los conceptos de diseño del MVC, con algún error puntual.	El proyecto cumple con los conceptos de diseño del MVC, pero presenta algunos errores leves.	El proyecto no cumple con los conceptos de diseño del MVC, o presenta errores graves en la utilización de este patrón.
CRUD Conversación	20%	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
CRUD Listar	10%	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
CRUD Eliminar	10%	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
CSV LLM	10%	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
Importación Exportación	10%	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	Solo se presenta una de las dos opciones exportación o importación.	No funciona esta característica o presenta errores graves de diseño.
Serialización	10%	Se presenta esta característica completamente funcional y sin fallos y cumpliendo MVC.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.
Parámetros entrada y configuración MVC	5%	Se presenta esta característica completamente funcional y sin fallos. Con parámetros y configuración del MVC correcta.	Se utilizan parámetros pero existe algún error puntual.	No se utilizan parámetros pero si se construye el MVC en el main.	No funciona esta característica o presenta errores graves de diseño.
Uso de Git continuado	5%	Se presenta un proyecto con uso de Git como control de versiones y con commits continuados que demuestran la evolución del proyecto.			Se presenta un proyecto sin uso de control de versiones Git o con un único commit.
One more thing[2]	extra 10%	Se presenta esta característica completamente funcional y sin fallos.	El proyecto presenta completamente funcional esta característica con algún fallo puntual.	El proyecto presenta esta característica pero con errores o deficiencias.	No funciona esta característica o presenta errores graves de diseño.

[2] Esta puntuación se puede utilizar para compensar errores de la práctica o para llegar a obtener un 1.1 en la práctica.