UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Plataforma web para una IA generadora de melodías musicales

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Balaguer López, Pablo

Tutor/a: Molina Marco, Antonio

CURSO ACADÉMICO: 2022/2023

Agradecimientos

De manera general, agradezco a mis padres, hermano y amigos cercanos el apoyo dado durante toda esta etapa universitaria.

Agradezco la oportunidad que la beca PROMOE me dio de estudiar en Montreal, donde realicé este proyecto, ya que me ha hecho crecer como persona y como profesional.

De manera especial, quiero agradecer toda la amistad, acogimiento y cariño que me otorgaron mis compañeros de piso canadienses en Montreal, Ethan y Zayne. Gracias por ayudarme en mis peores momentos y siempre ver algo bueno en mí. Gracias por ser mi familia canadiense.

Resumen

Este proyecto consiste en el desarrollo e implementación de una plataforma web diseñada para dar acceso al usuario a una Inteligencia Artificial (IA) generadora de melodías musicales. A pesar de que este proyecto se presente como un trabajo académico, tiene también una proyección comercial, ya que la plataforma web desarrollada permite acceder a los diversos proyectos de IA desarrollados en el Laboratorio Quosséça. Este laboratorio, el cual es dirigido por el profesor Gilles Pesant, se encuentra en la Universidad Politécnica de Montreal (Canadá), y está especializado en IA y programación mediante restricciones.

En el laboratorio existían diversos proyectos enfocados en la generación de melodías musicales mediante algoritmos de IA, pero no tenían alcance a un público general, quedándose recluidos en el ámbito académico. Así nació la necesidad de crear una plataforma web desde la que cualquier usuario externo pudiese acceder a estos proyectos, pudiese entrenar el algoritmo seleccionado con sus propios ficheros MIDI, pudiese seleccionar las restricciones a aplicar a la futura melodía generada y posteriormente reproducirla.

La metodología empleada para el desarrollo de la plataforma web ha sido una metodología en cascada. El *frontend* de la plataforma se ha desarrollado en HTML, CSS, el *framework* Bootstrap y JavaScript. El *backend* relacionado con la lógica de la plataforma se ha realizado mediante el *framework* Flask y Python, mientras que el correspondiente al almacenamiento de datos se ha realizado mediante el *framework* .NET de C# (creación de una API REST).

Como resultado final, se ha logrado crear una plataforma web *responsive* dotada de todas las funcionalidades necesarias posibles para generar melodías musicales a través de los proyectos desarrollados en el Laboratorio Quosséça.

Palabras clave: plataforma web, melodía musical, MIDI, restricciones, HTML, CSS, Flask, JavaScript, API REST

Abstract

This project consists in the development and implementation of a web platform designed to provide access to users to an AI music melody generator. Although this project is presented as an academic endeavour, it also has a commercial projection, due to the developed web platform allows access to various AI projects developed at the Quosséça Laboratory. This laboratory, directed by Professor Gilles Pesant, is located at Polytechnique Montreal (Canada) and specializes in AI and constraint programming.

Within the laboratory, there were several projects focused on generating music melodies using AI algorithms, but they lacked reach to a general audience and remained inside of the academic sphere. This gave rise to the need to create a web platform from which any external user could access these projects, train the selected algorithm with their own MIDI files, choose the constraints to apply to the future generated melody and subsequently play it.

The methodology employed for the development of the web platform was a waterfall methodology. The frontend of the platform has been developed using HTML, CSS, Bootstrap framework and JavaScript. The backend related to the platform's logic has been implemented using the Flask framework and Python, while the data storage has been implemented using the .NET framework of C# (API REST creation).

As a final result, a responsive web platform equipped with all the necessary functionalities to generate music melodies through the projects developed at the Quosséça Laboratory has been successfully created.

Keywords: web platform, music melody, MIDI, constraints, HTML, CSS, Flask, JavaScript, API REST

Definiciones y abreviaturas

API REST: APIs de naturaleza REST (Representational State Transfer). Es una interfaz de comunicación que usa el protocolo HTTP para obtener/publicar datos o ejecutar operaciones. Las acciones más comunes son POST (publicar datos), GET (obtener datos), PUT (editar datos) y DELETE (borrar datos).

Capa de presentación: parte visual o interfaz gráfica que interactúa directamente con el usuario.

Capa lógica (de negocio): parte encargada de proporcionar la lógica y funcionalidad a la aplicación. Hace de intermediaria entre la capa de presentación y la de persistencia.

Capa de persistencia (de datos): parte encargada de gestionar y almacenar los datos.

Backend: parte de la aplicación que se encarga de la correcta gestión y almacenamiento de datos, así como de la lógica interna de la aplicación.

Frontend: parte de la aplicación que se encarga de la interfaz visual y de la interacción del sistema con el usuario.

CRUD: acrónimo de *Create*, *Read*, *Update* & *Delete* en inglés. Las operaciones CRUD (Crear, Leer, Actualizar y Borrar) son las operaciones básicas que se efectúan sobre una Base de Datos.

Framework: es un marco de trabajo el cual está estandarizado y tiene unas guías y reglas a seguir.

HTTP (Hypertext Transfer Protocol): protocolo de comunicación y transferencia de datos en la World Wide Web.

Responsive: técnica de diseño web que busca una correcta visualización de una página independientemente del dispositivo que la cargue.

UML (Unified Modeling Language): lenguaje de modelado estandarizado para documentar un sistema.

MIDI (Musical Instrument Digital Interface): los ficheros de tipo MIDI son archivos especiales que almacenan información relacionada con el tono, velocidad… de una pista musical.

Programación mediante restricciones: paradigma utilizado para resolver problemas de combinatoria mediante la declaración de restricciones y la búsqueda de una solución factible que cumpla con estas.

NN (Neural Network): red neuronal en castellano. Es una de las arquitecturas utilizadas en *machine learning*. Está formada por 1 o más capas, las cuáles son un conjunto de nodos que producen un resultado. El modelo más común es una capa la cual recibe una serie de *inputs*. A estos *inputs*, se le aplican unos pesos, y de la multiplicación de los *inputs* por sus pesos se obtiene un *output*, el cual es el resultado. Comparando el resultado obtenido con el esperado, se van ajustando estos pesos, de manera que la red neuronal llegue a un punto que pueda calcular *outputs* de manera correcta haciendo uso de unos *inputs*, sin necesidad de tener el valor esperado.

RNN (Recurrent Neural Network): red neuronal recurrente en castellano. Es un tipo de NN, la cual utiliza datos secuenciales o series periódicas temporales. A diferencia de las NN, el *output* que se genera partir de unos *inputs* sí que tiene en cuenta lo que la RNN ha generado anteriormente como *output*. Es decir, cuando la RNN genera un resultado, este resultado se agregará a los *inputs* para la generación del resultado siguiente. Así pues, a diferencia de las NN, los *inputs* y los *outputs* en la RNN no son independientes el uno del otro. Este tipo de NN son utilizadas para el procesamiento de Lenguaje Natural.

RL (Reinforcement Learning): aprendizaje por refuerzo en castellano. En RL, existe un agente el cual interactúa en un entorno, y este aprende a tomar acciones óptimas para poder así maximizar la recompensa acumulada a lo largo de las acciones que toma. Aquí, el agente toma una acción (A) partiendo de un estado (S), y en función de lo buena o mala que ha sido esa acción (A) partiendo de ese dicho estado (S), genera una recompensa (R), y pasa a un nuevo estado (S'). Así sucesivamente.

DDQN (Double Deep Q-Network): es un algoritmo de RL. Es una versión mejorada de DQN (Deep Q-Network). Básicamente este algoritmo da un valor a cada posible acción que el agente puede tomar dependiendo en qué estado se encuentre actualmente. DDQN utiliza dos redes neuronales, una la principal y la otra la objetivo. La principal se encarga de elegir la mejor acción en función de los valores estimados, mientras que la objetivo se encarga de evaluar el valor de la acción seleccionada.

CP model (Constraint Programming model): modelo de programación por restricciones en castellano. Es un paradigma, donde las relaciones que se dan entre las variables se expresan en términos de restricciones. En CP, las variables tienen unos dominios (valores que pueden tomar), y unas restricciones (restringen los dominios de cada variable). Cuando un valor de dentro de un dominio se le asigna a una variable, hay que tener en cuenta que todas las restricciones del modelo se están cumpliendo. Si esto no fuera así, se violaría alguna restricción. Un modelo de programación por restricciones se resuelve cuando todas las variables del modelo tienen un valor asignado (y este valor está dentro del dominio que cada variable tiene), y a la vez todas las restricciones del modelo se cumplen.

Token: es una unidad individual de una secuencia, en otras palabras, es un símbolo que se considera como una entidad separada dentro de una secuencia. Una secuencia podría ser un texto (un conjunto de caracteres, los cuales cada uno de ellos conforman un *token*) o una melodía musical (cada unidad de tiempo, ya bien represente un silencio o una nota, se puede considerar como un *token*). El concepto de *token* se utiliza mucho en el procesamiento de lenguaje natural.

Índice de contenido

Índice de figuras

Índice de tablas

1. Introducción

A día de hoy, es innegable que para que un producto tenga alcance debe de tener presencia en Internet. En los últimos años, la World Wide Web (WWW) ha experimentado un avance impresionante. Desde la expansión de la conectividad global hasta el aumento exponencial de contenido en línea y la adopción generalizada de servicios basados en la nube, la WWW ha transformado nuestra forma de comunicarnos, compartir información y acceder al conocimiento. Si se pretende vender algo, necesita estar en Internet, para que así pueda estar al alcance de todos con tan solo un par de *clicks*.

El laboratorio Quosséça es un centro de trabajo dirigido y creado por el profesor Gilles Pesant, doctor en Inteligencia Artificial (IA) y programación por restricciones. En este laboratorio, el cual se encuentra dentro de la Universidad Politécnica de Montreal (Canadá), estudiantes del máster de IA y postulantes a doctorado vienen a desarrollar sus proyectos académicos. En el laboratorio se investiga acerca de la Inteligencia Artificial y la programación por restricciones, donde combinando estos dos campos se da lugar a proyectos tan interesantes como IAs generadoras de música.

Dentro del laboratorio, se han llevado a cabo diversos proyectos los cuales incluso han sido premiados por convenciones académicas pero que en ningún momento se han presentado y ofrecido a un público general para darles alcance y un uso práctico. Así pues, se decidió crear una plataforma web la cual albergara dos de estos proyectos (ambos relacionados con IAs para la generación de melodías), donde el usuario general podría acceder y generar melodías de forma totalmente gratuita y sin copyright. Ambos proyectos que se ofrecen permiten entrenar la IA con melodías que el usuario escoja y luego aplicar restricciones a la generación de la melodía.

Este proyecto engloba la creación de una plataforma web que permitirá dar alcance y presencia a los proyectos desarrollados en el laboratorio Quosséça relacionados con la generación de melodías mediante la IA. Además de esto, desde el laboratorio también se quería hacer un especial énfasis en facilitar lo máximo posible al usuario el uso de estos algoritmos, por lo que la página web deberá de ser dinámica, dirigida y clara para el usuario general. Así, además de darles alcance a un mayor público, también se asegura la utilidad y la facilidad de uso de estos proyectos.

Las distintas fases que abarcan el desarrollo del producto se irán exponiendo conforme se avance en la memoria, llegando a ver el producto final y viendo cómo los objetivos principales se cumplen correctamente.

1.1 Motivación

Cuando se me concedió la beca PROMOE a Montreal, tuve que buscar un profesor el cual me pudiese dirigir un proyecto de fin de grado para realizar mi estancia en Canadá. Durante mi búsqueda di con un profesor cuyo campo de investigación, a parte de la IA y la programación por restricciones, era la generación de melodías musicales mediante algoritmos. Esto era idílico para mí, ya que juntaba dos de mis pasiones, la música y la ingeniería informática. Contacté con Gilles Pesant, el profesor del cual hablo, para poder hacer allí una estancia y hacer un proyecto relacionado con este campo de estudio. Él me comentó cuál era el problema que tenían con sus proyectos y me ofreció realizar una plataforma web mediante la cual un usuario medio pudiese hacer uso de estos. Por lo tanto, decidí encargarme del proyecto y ayudarles en el desarrollo de la plataforma.

Mi idea era poder desarrollar un algoritmo haciendo uso de IA para poder generar melodías musicales, pero mi *background* técnico no era lo suficientemente avanzado. Sin embargo, desarrollando la plataforma web podría conocer de cerca estos proyectos y entender cuál era su funcionamiento, así como mejorar mis aptitudes en desarrollo web.

Así pues, vi una oportunidad de mejorar y conocer más campos de la Ingeniería Informática, así como de desarrollarme personalmente y profesionalmente. La realización de este proyecto implicaba mudarme a Montreal durante un tiempo y experimentar un ambiente de trabajo totalmente distinto al que había conocido aquí en Europa, así como programar una plataforma web desde cero y sin previa concepción de cómo diseñarla o de qué arquitecturas había que usar. Era un reto y había que jugársela.

1.2 Objetivos

Como previamente se ha mencionado, el objetivo global de este trabajo era desarrollar una plataforma web que diera cabida a los proyectos desarrollados en el laboratorio Quosséça, teniendo en todo momento en mente que la plataforma desarrollada tenía que facilitar y guiar al usuario mediante su uso.

A pesar de que este trabajo fue realizado para la Universidad Politécnica de Montreal, no existía previamente ningún esbozo de lo que se quería ni ningún punto de partida, por lo que se desarrolló desde cero.

Por consiguiente, se decidieron que estos serían los principales objetivos a cumplir para un correcto desarrollo de la aplicación:

- Diseñar una plataforma web con un correcto funcionamiento y un adecuado dinamismo que guíe al usuario durante el proceso

    - Desarrollar el *frontend* en HTML, CSS y JS
    - Desarrollar un *frontend responsive* haciendo uso del *framework* Bootstrap
    - Mantener la estructura y dotarla de consistencia para darle una mayor facilidad al usuario
    - Desarrollar el *backend* de la aplicación haciendo uso del *framework* Flask y Python

- Dotar de una interfaz gráfica intuitiva para la selección de restricciones a aplicar a la futura melodía generada

- Permitir al usuario subir sus propias MIDI Files para entrenar la IA que escoja

    - Dejarle crear y borrar las MIDI Files que el usuario quiera
    - Cuando la MIDI File esté compuesta por más de 1 pista, dejarle escoger cuál será la que actúe como melodía

- Gestionar la información de usuarios

    - Dotar a la aplicación de la gestión básica de usuarios (creación, modificación y borrado de usuarios)
    - Gestionar la información de los usuarios mediante una base de datos relacional desde la que accederemos a través de una API REST

1.3 Impacto esperado

La puesta en marcha de esta plataforma web tendrá repercusión tanto para el laboratorio como para los usuarios que hagan uso de esta, así como relación con los Objetivos de Desarrollo Sostenible (ver Anexo 2). Así pues, el impacto que realizará en cada uno de ellos será:

Laboratorio

Debido a que la plataforma web dotará de un mayor alcance a los proyectos desarrollados dentro del laboratorio, este ganará reconocimiento y los desarrolladores que han participado en cada uno de los proyectos tendrán mayor visibilidad. Además, los proyectos desarrollados aquí ganarán utilidad, ya que acercamos su acceso a un usuario medio y le permitimos hacer uso de este *software* de manera fácil y gratuita.

Clientes

Los clientes podrán hacer uso de este *software* de manera totalmente gratuita, a diferencia de otros productos similares que son de pago o tienen versiones gratuitas muy limitadas. Asimismo, la música que podrán generar será sin *copyright*, algo que es de gran ayuda para creadores de contenido. Adicionalmente, el usuario también podrá amoldar la melodía a su gusto, haciendo uso de las restricciones que se pueden aplicar sobre la melodía a generar.

1.4 Metodología

Para el desarrollo eficaz de la plataforma, se ha seguido una metodología en cascada, la cual tiene un enfoque secuencial en vez de iterativo (Kumar Pal, 2022). Esta metodología se caracteriza por no poder pasar de fase hasta que la anterior no haya acabado completamente. A pesar de que es una de las metodologías menos populares, para este proyecto casaba debido a que los principales requerimientos y objetivos estaban definidos desde el principio y era claro que estos no iban a cambiar.

Figura 1. Fases de la metodología en cascada. Fuente: elaboración propia.

A continuación, explicaré y pondré en contexto las fases de la Figura 1:

1. **Análisis:** durante esta fase, se hace uso de las reuniones con el cliente y se modelan diagramas de casos de uso en UML para capturar los requerimientos necesarios para llevar a cabo el desarrollo del proyecto. Para ello, se habla con el cliente y se diseñan casos de uso junto a él para recopilar todos los requisitos que esta aplicación debe cumplir. En este caso, se tuvieron diversas reuniones con el profesor del laboratorio, Gilles Pesant, y se acordó qué requisitos indispensables serían necesarios para el correcto funcionamiento.

2. **Diseño:** el objetivo de esta etapa es trasladar los requerimientos que hemos recopilado en la anterior etapa en un formato que posteriormente pueda ser programado. Aquí, tuvimos en cuenta todo lo hablado y desarrollamos bocetos y *mockups* de cómo debería de ser la plataforma web, para enseñar un prototipo al cliente de funcionalidades y estructuración de esta. También se definió la arquitectura del proyecto.

3. **Implementación:** la parte de Implementación consiste en codificar el prototipo previamente creado y crear finalmente la plataforma web, dotándola de una estructuración y una funcionalidad real. Básicamente, aquí se crea la plataforma web. La arquitectura que debe de ser y los lenguajes que se utilizan para codificarla habían sido previamente acordados con el cliente en la fase de diseño, otorgándole así el producto que había pedido.

4. **Validación:** durante esta etapa se valida el producto desarrollado. Se crean casos de prueba para los casos de uso establecidos. Se comprueba que las conexiones funcionan, que las especificaciones se cumplen y que todo funciona como se espera. Se realizaron pruebas manuales para ver que todo funcionaba correctamente y después se enseñó al cliente.

5. **Mantenimiento:** la última etapa del ciclo de vida del *software* consiste en su mantenimiento hasta el final de su vida útil. Esta fase consiste en corregir errores que no se han descubierto mediante la etapa de validación, mejorar funcionalidades que pueden quedar obsoletas y adaptar el *software* a las necesidades del momento. Cuando hablemos sobre cosas a mejorar del proyecto, podremos ver futuros cambios que sufrirá la plataforma web para su correcto mantenimiento.

1.5 Estructura

El proyecto divide su explicación en 10 partes. A continuación se proporciona un breve resumen del contenido de cada parte.

- **1. Introducción:** en este apartado se explica qué ha motivado al alumno a realizar el proyecto, qué objetivos se pretenden conseguir con el proyecto y qué impacto se espera de este. Seguido de esto se explica con sencillez la metodología aplicada, las colaboraciones que se han tenido y los proyectos de IA que se han utilizado.

- **2. Estado del arte:** en este capítulo se exploran las distintas aplicaciones que tiene el mercado para la generación musical mediante IA. Se recopila información sobre estas soluciones para hacer de nuestra solución la más completa y competitiva.

- **3. Análisis del problema:** se analiza cuál es el problema a tratar y se recopila información sobre lo que tiene que incluir la solución. Así pues, se recogen los Requisitos Funcionales y No Funcionales que debe tener nuestra plataforma mediante los Casos de Uso y se crea el modelo de datos. Posteriormente, se presenta cuál es la solución negociada con el cliente y se organiza un plan de trabajo para llevarla a cabo.

- **4. Diseño de la solución:** tras recabar toda la información necesaria relativa a la solución se pasa a la fase de diseño. Primero, se diseña a grandes rasgos la arquitectura de la solución. Después, se detalla la solución de manera más concreta separándola por las 3 capas que posee: de presentación, lógica y de datos.

- **5. Tecnologías utilizadas:** en este apartado se presentan las tecnologías utilizadas para implementar la solución. Aquí se describirán las herramientas generales que se han utilizado para el desarrollo de la solución, así como los lenguajes y *frameworks* utilizados para el *frontend* y *backend* de la solución.

- **6. Implementación de la solución:** una vez descritas las tecnologías utilizadas para la implementación, se explica cómo se ha implementado la plataforma web. Así pues, se explica y se dan ejemplos de código del marco de trabajo principal Flask. Seguido a esto se detallan particularidades que ha requerido nuestra solución y como hemos resuelto ciertos problemas que han surgido.

- **7. Pruebas:** en este capítulo se describen las pruebas manuales utilizadas para validar la solución. Se comentan las diversas pruebas y se explican, concluyendo finalmente con las pruebas de aceptación, donde el cliente final nos da el visto bueno a la plataforma.

- **8. Producto final:** se presentan capturas de pantalla del producto final de forma estática. Uno de los grandes fuertes de la plataforma es su dinamismo, pero no puede ser mostrado en este trabajo de documentación.

- **9. Conclusiones:** el alumno reflexiona sobre el trabajo hecho, complicaciones, adversidades, problemas y soluciones. El alumno comenta cómo este proyecto le ha hecho mejorar profesional y personalmente.

- **10. Trabajos futuros:** en este capítulo se listan ciertos aspectos de mejora que podría incluir la plataforma. Así pues, da cabida a continuar con la mejora del proyecto y seguir desarrollándolo.

1.6 Colaboraciones

El desarrollo de la plataforma web (diseño y funcionalidad) así como de la API REST que se utilizará para la gestión de usuarios ha sido realizado en totalidad por mí. Sin embargo, como esta plataforma web tiene como finalidad hacer uso de 2 proyectos de IA, explicaremos en esta sección en qué consisten, cómo funcionan y quiénes fueron sus creadores.

Ambos proyectos fueron desarrollados dentro del Laboratorio Quosséça por estudiantes del máster de Inteligencia Artificial. Estos proyectos combinan algoritmos de IA y la programación por restricciones. Ambos proyectos están relacionados con la generación de melodías musicales, aunque se diferencian en arquitectura e implementación.

RL-TUNER

RL-Tuner (Lafleur, Chandar, & Pesant, 2022) es un algoritmo para la creación de melodías musicales sin base armónica. Este proyecto ha sido desarrollado por Daphné Lafleur, estudiante de la Universidad Politécnica de Montreal. La siguiente imagen (Figura 2) es un resumen del proyecto desarrollado, y a través de ella explicaremos los diversos aspectos que cubre el proyecto para poder entenderlo a rasgos generales.

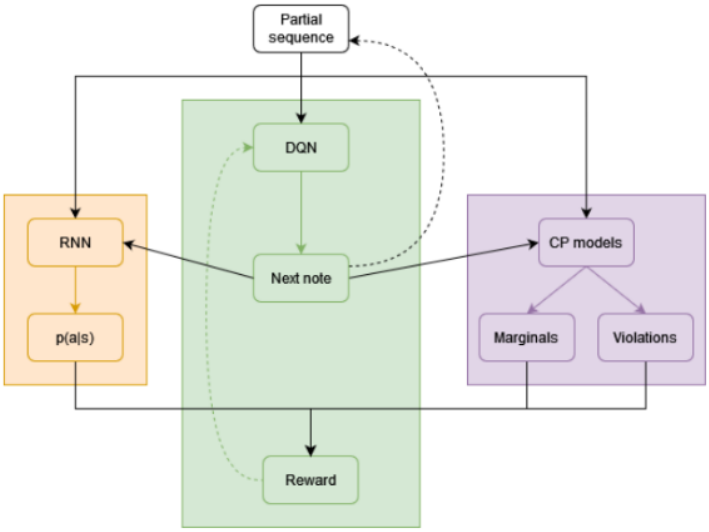

Figura 2. Resumen de la arquitectura de la mejora de RL-Tuner. Fuente: (Lafleur, Chandar, & Pesant, 2022).

Para poder entender ciertos conceptos, se recomienda acceder al apartado de 'Definiciones y Abreviaturas', donde se explican.

Como previamente se ha introducido, con este algoritmo se crean melodías sin base armónica. Para entender el funcionamiento de este, partimos de una secuencia parcial de la melodía (la que hasta el momento se ha generado). Básicamente esto implica que tenemos parte de la melodía generada, desde el *token* $_{t = o}$ hasta el *token* $_{t = (t - 1)}$.

Esta secuencia de *tokens* alimenta al algoritmo DQN (en versiones mejoradas del proyecto se utiliza el algoritmo DDQN, que tiene una funcionalidad similar), a la RNN y al modelo de Programación mediante Restricciones.

En primer lugar, el algoritmo DQN, partiendo del estado en el que se encuentra (la secuencia de *tokens* que tiene ya generada) decide tomar una acción (produce la siguiente nota). Así pues, el algoritmo DQN produce el *token* $_{t = t}$.

Este nuevo *token* generado se añadirá a la secuencia hasta ahora creada, para la siguiente iteración que produzca la arquitectura. Aparte de esto, este nuevo *token* que se ha generado se le proporciona a la RNN y al modelo de Programación mediante Restricciones.

Ahora, la RNN, teniendo en cuenta la secuencia generada desde $t = 0$ hasta $t = (t-1)$ y el nuevo *token* $_{t = t}$, generará un vector con las probabilidades que tiene el *token* $_{t = t}$ de tomar todos sus posibles valores. Es decir, un *token* puede tomar un dominio de valores y lo que se calcula a partir de la secuencia ya generada y del *token* generado por el algoritmo DQN es la probabilidad que tendría de tomar cada uno de esos valores.

El modelo de Programación por Restricciones también recibe los mismos *inputs* que la RNN. Aquí, también se calculan las probabilidades marginales de que el *token* $_t$ tome los valores que están incluidos en su dominio, así como de cuántas restricciones del modelo viola tomando el valor que el algoritmo DQN ha generado para el *token*. A mayor número de restricciones violadas, peor. Hay que recalcar que aquí, en el modelo de Programación por Restricciones, es donde aplicamos conocimiento sobre el dominio. Es decir, es aquí donde nosotros escribimos restricciones para que la generación de la melodía musical tenga sentido y siga pautas musicales. Algunos ejemplos serían definir que el último *token* tome el valor de la tónica (primera nota de una escala), o que los saltos entre *tokens* (notas) no fueran mayores de, por ejemplo, 6 semitonos. En el *paper* que explica RL-Tuner se explica con mayor profundidad esta parte.

Una vez tenemos estos *outputs* por parte de la RNN y del modelo de Programación por Restricciones, se genera una recompensa. La recompensa se calcula a partir de estos *outputs*, y esta determinará cómo de bien se ha escogido el valor para el *token* $_t$ producido por el algoritmo DQN. Así pues, dependiendo del valor de la recompensa, la política seguida por el algoritmo DQN para escogen un valor para el siguiente *token* a generar se modificará, optimizándose conforme se va generando la melodía.

Para el siguiente *token* a generar, se iterará de nuevo sobre el proceso que se acaba de explicar.

Cuando se entrena el algoritmo RL-Tuner, las MIDI Files que se necesitarán serán solo aquellas que tengan melodía, es decir, que estén compuestas solo por una pista. Posteriormente, se entenderá el porqué de esta explicación.

CMT_CPBP

CMT_CPBP (Virasone, 2022) es un algoritmo para la creación de melodías musicales con base armónica. Este proyecto fue desarrollado por Virasone Manibod, estudiante de la Universidad Politécnica de Montreal. Procederemos a explicarlo brevemente a través de la siguiente imagen (Figura 3).

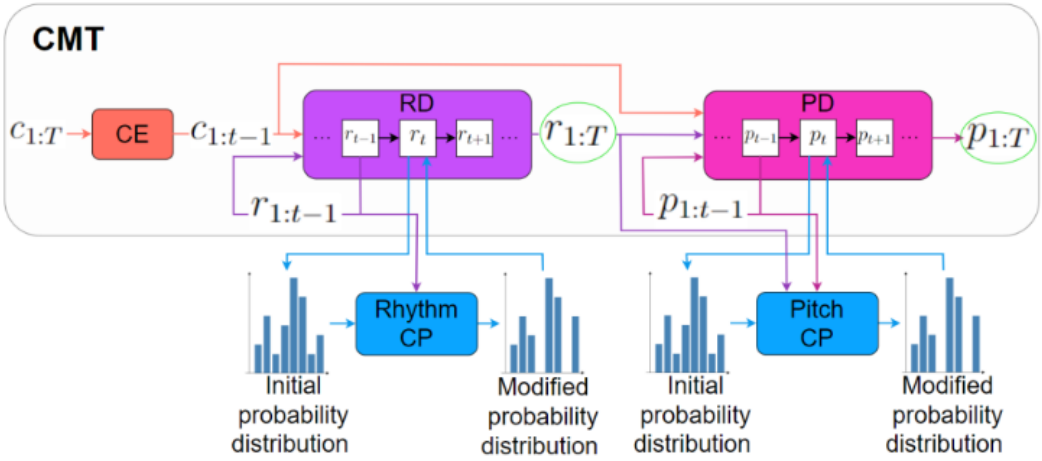

Figura 3. Resumen de la arquitectura CMT_CPBP. Fuente: (Virasone, 2022).

Este proyecto está compuesto de un Chord Encoder (Codificador de acordes), de un Rythm Decoder (Decodificador de ritmo) y un Pitch Decoder (Decodificador de tono). Para mejorar la eficacia del proyecto se añaden modelos de Programación por Restricciones tanto en el decodificador de ritmo como en el decodificador de tono.

Cabe destacar que este proyecto no solo genera una melodía compuesta por *tokens* que representen notas, sino que paralelamente genera *tokens* que representen notas y *tokens* que representen el tiempo. Así pues, en el $\boldsymbol{t} = \boldsymbol{x}$ tendremos un *token* representando la nota que se está tocando o un silencio mientras que tendremos otro token representando si la nota se acaba de poner en marcha, si es una nota que se está arrastrando desde atrás o si es un silencio. Se puede observar en la Figura 4.

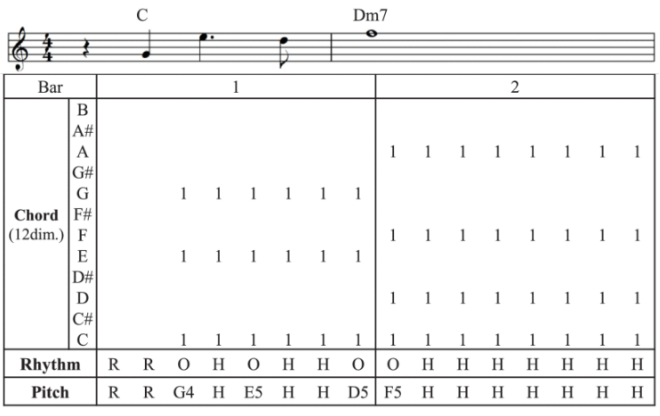

Figura 4. Representación gráfica del funcionamiento de los tokens. Fuente: (Choi, Park, Heo, Jeon, & Park, 2021).

Los *inputs* que recibe la arquitectura son los acordes de la melodía con la que se está entrenando. Estos acordes, pasan por el codificador de acordes, codificándose en un cierto formato y así pudiendo alimentar al decodificador de ritmo. El decodificador de ritmo generará los *tokens* para el ritmo, y cada *token* generado alimentará al modelo de Programación por Restricciones del ritmo. Una vez este calcule las probabilidades del valor del *token* de ritmo en función de las restricciones que tenga y el dominio posible de valores de ese *token*, le devolverá la probabilidad al decodificador de ritmo para la siguiente iteración. Esto hará que el decodificador de ritmo mejore su eficacia. Esto ocurrirá hasta que toda la secuencia de ritmo se genere.

Una vez generada toda la secuencia de ritmo, se alimentará el decodificador de tono con la secuencia de acordes y la secuencia de ritmo. El proceso anteriormente explicado ocurrirá exactamente igual para el decodificador de tono, y finalmente generará una secuencia de tonos. Así pues, una vez generadas las secuencias de ritmo y tono obtendremos finalmente la melodía con base armónica.

2. Estado del arte

La rápida evolución de la IA y la facilidad de uso con la que se presenta ha hecho que esta tecnología sea adoptada y utilizada por un público general. Actualmente, todo producto tiene presencia en la web, por lo que la IA no iba a ser menos. Existen numerosas plataformas web donde se nos permite acceder a diferentes IAs y hacer uso de ellas.

Una de las áreas más importantes donde la IA presenta un crecimiento exponencial es en la creación de arte. Un ejemplo destacado es Dall-E, una plataforma web que genera imágenes con tan solo proporcionarle una descripción de cómo deberían de ser estas. Otro ejemplo sería ChatGPT, otra plataforma web que nos genera letras de canciones a partir de un par de palabras.

En este trabajo nos centramos en plataformas web que ofrezcan IAs que sean capaces de generar melodías musicales. El gran desarrollo experimentado en este campo ha sido fruto de las grandes ventajas que este ofrece al usuario, tales como proporcionar melodías musicales gratuitas, creativas y libres de derechos de autor. De esta forma, el usuario puede hacer uso de ellas para la creación de contenido, buscar inspiración o simplemente para disfrutarlas. A continuación, presentamos diversos proyectos actuales relacionados con la generación musical.

2.1 Magenta Studio

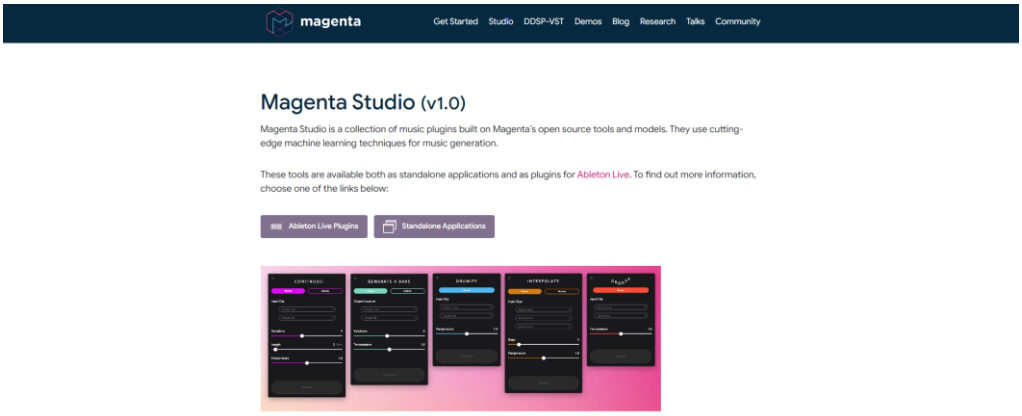

Figura 5. Página web de Magenta Studio. Fuente: elaboración propia.

IA desarrollada por Google's Magenta. Magenta Studio nos proporciona una colección de *plugins* para Ableton (*software* de producción musical) o aplicaciones de escritorio para la generación de melodías. Este *software* tiene la opción de generar melodías a partir de MIDI Files que nosotros le proporcionamos. Se nos permite modelar cómo generar nuestras melodías, pero de una forma breve. Como se puede observar, carece de una aplicación web donde directamente el usuario puede generar las melodías.

2.2 AIVA

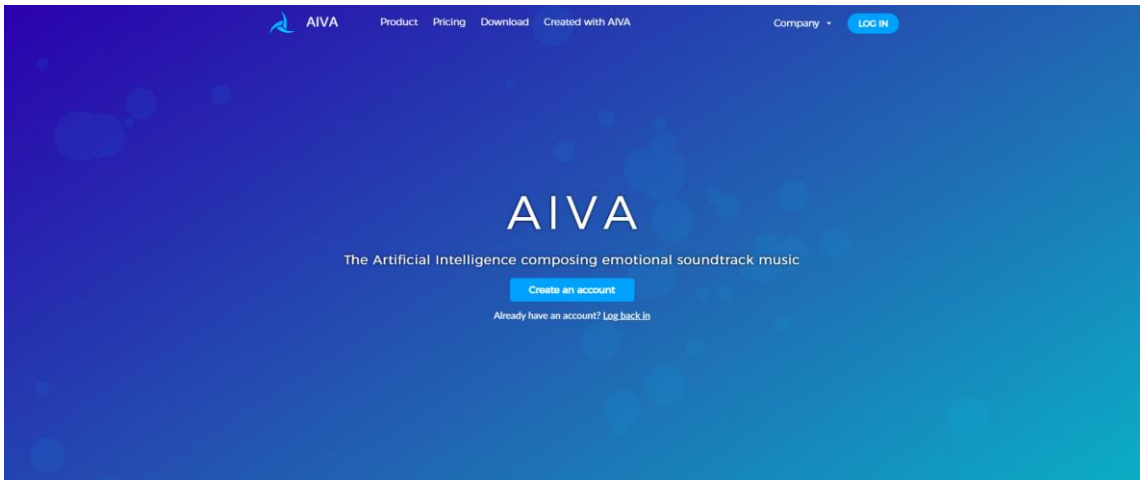

Figura 6. Página web de AIVA. Fuente: elaboración propia.

IA desarrollada por AIVA. Nos ofrece una aplicación integrada dentro de la plataforma web desde donde podremos generar melodías. Nos permite establecer un estilo predeterminado para la composición, la tonalidad, el tiempo... incluso permite interactuar con la melodía generada, presentándonosla junto a una interfaz de piano. Ofrece una subscripción mensual gratuita pero muy limitada, siendo casi obligatorio pagar por su subscripción.

2.3 Ecrett Music

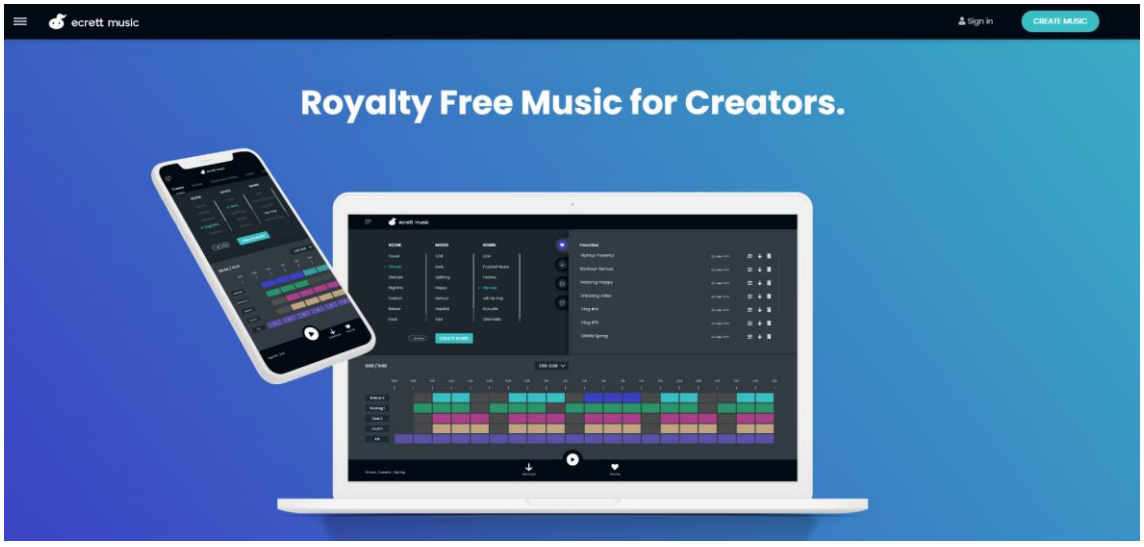

Figura 7. Página web de Ecrett Music. Fuente: elaboración propia.

IA desarrollada por Ecrett Music. Las funcionalidades ofrecidas son muy parecidas a las que AIVA ofrece. Nos permite generar música seleccionando qué género queremos, cuál es el enfoque sentimental que le queremos dar y en qué tipo de escenario se tendría que dar esta melodía. Después, nos permite modificar la melodía modificando las pistas que esta tiene. No nos permite generar música a partir de MIDI Files que nosotros proporcionemos, y el plan gratuito que posee es bastante escaso.

2.4 Propuesta

Se han analizado las plataformas anteriores para poder recabar información de los aspectos que una plataforma web para la generación de melodías musicales debería de tener. Así pues, con estos datos, hemos creado una tabla comparativa donde exponemos cada una de las plataformas junto con las características que cumplen (Tabla 1).

Tabla 1. Comparativa de las diferentes plataformas. Fuente: elaboración propia.

Plataforma web /Características	Generar a partir de tus MIDI	Generar sin necesidad de MIDI	Completamente gratuita	Acceder desde la web	Adaptar la generación	Interfaz para ver la melodía
Magenta Studio						
AIVA						
Ecrett Music						
Nuestra plataforma						

Como podemos observar en la Tabla 1, es indispensable que desde la plataforma web se pueda acceder a la IA, sin necesidad de descargar una aplicación de escritorio para poder utilizarla. Este aspecto es clave, y como podemos observar, prácticamente todas lo cumplen salvo Magenta Studio. Otras de las funcionalidades que debería de tener sería la opción de generar la melodía sin necesidad de proporcionar ningún fichero, a partir del algoritmo, así como adaptar la generación al gusto del usuario. Estas funcionalidades son recogidas por todas las plataformas que permiten al acceso del algoritmo desde la página.

Sin embargo, podemos observar cómo no todas son completamente gratuitas, y, aún peor, hay ciertos productos que no permiten generar la melodía a partir de nuestros propios ficheros. En nuestro desarrollo, permitiremos acceder a todas las funcionalidades de manera gratuita, así como dar la opción al cliente de generar sus melodías en base a las pistas que este elija. Un aspecto que no cumplimos desde nuestra plataforma es el disponer de una interfaz para ver la melodía generada, e, incluso, modificarla una vez se ha generado. Esta funcionalidad es verdaderamente útil para el usuario, por lo que en futuros aspectos de mejora será incluida. Finalmente, nuestra propuesta cumple con 5 de las 6 funcionalidades que detectamos como claves, justificando así su creación y mejorando lo ya existente.

3. Análisis del problema

Tras finalizar el estudio realizado sobre el estado del arte, donde hemos analizado cómo se encuentra el mercado actual para el tipo de producto que estamos desarrollando, pasaremos a identificar los requerimientos y funcionalidades que nuestro producto tiene que incluir para no solo equipararse a lo que actualmente hay, sino para ofrecer una solución más completa.

Previamente, se ha creado una tabla (Tabla 1) donde se comparaban los productos teniendo en cuenta las principales funcionalidades que se esperaban de ellos. Como se puede observar, nuestra solución presentaba el mayor número de funcionalidades esperadas. Así pues, para la recopilación de requisitos deberemos tener en cuenta estas funcionalidades, para adaptar los requisitos a lo que se espera de ellas.

Por lo tanto, para el correcto análisis del problema, primero se recopilarán los requisitos. Se comenzará por recopilar los requisitos funcionales y no funcionales, utilizando el estándar IEEE 830 para ello. Tras esto, se presentarán los diagramas de casos de uso y el modelo de datos utilizado.

Para finalizar, se introducirá la solución propuesta y cuál ha sido la planificación de esta en base a la metodología utilizada.

3.1 Especificación de requisitos

La especificación de requisitos se encuentra dentro de la fase de Análisis, siendo esta la primera fase del desarrollo del producto y una de las más importantes, sobre todo si se emplea una metodología en cascada. Esto se debe a que el desarrollo del producto se hace de manera secuencial y no iterativa, por lo que una vez completada esta fase no se podrá volver atrás para recopilar más requisitos. Por lo tanto, una correcta especificación es esencial para dar con un desarrollo óptimo. Para la recopilación de estos se tuvieron diversas reuniones con el profesor Gilles Pesant, y se estableció qué requisitos eran indispensables para el proyecto.

3.1.1 Requisitos funcionales

La siguiente tabla recopila los requisitos funcionales de la aplicación.

Tabla 2. Recopilación de requisitos funcionales. Fuente: elaboración propia.

Identificación del requisito	Nombre del requisito
RF01	Dirigirse website universidad
RF02	Dirigirse website laboratorio
RF03	Dirigirse RRSS universidad
RF04	Crear usuario
RF05	*Loguear* usuario
RF06	Recuperar información usuario
RF07	*Desloguear* usuario
RF08	Borrar usuario
RF09	Seleccionar *checkpoint* – No entrenar IA
RF10	Seleccionar subir MIDI – Entrenar IA
RF11	Subir MIDI
RF12	Borrar MIDI
RF13	Seleccionar melodía de MIDI
RF14	Generar MIDI sólo melodía
RF15	Generar MIDI melodía y acordes
RF16	Crear carpeta MIDIs sólo melodía
RF17	Crear carpeta MIDIs melodía y acordes
RF18	Seleccionar longitud melodía
RF19	Seleccionar restricciones tono consecutivas
RF20	Seleccionar restricciones duración consecutivas
RF21	Seleccionar restricciones tono separadas
RF22	Seleccionar restricciones duración separadas
RF23	Guardar restricciones
RF24	Borrar restricciones
RF25	Obtener información RL-Tuner
RF26	Obtener información CMT_CPBP
RF27	Escoger proyecto RL-Tuner
RF28	Escoger proyecto CMT_CPBP
RF29	Descargar MIDI generada
RF30	Reproducir MIDI generada

A continuación, detallaremos cada requisito funcional siguiendo una plantilla del estándar IEEE 830. Cabe destacar que en esta plantilla incluimos los requerimientos no funcionales a los que el requerimiento funcional está asociado. Más abajo se detalla cuáles son estos requerimientos no funcionales.

Tabla 3. Requisito funcional RF01. Fuente: elaboración propia.

Identificación del requisito	RF01
Nombre del requisito	Dirigirse website universidad
Descripción	El sistema ofrecerá la posibilidad de acceder a la website de la universidad en la cual se encuentra el laboratorio. Simplemente es una redirección para que el usuario conozca la organización en la que se encuentra el laboratorio que desarrolla el proyecto.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Media

Tabla 4. Requisito funcional RF02. Fuente: elaboración propia.

Identificación del requisito	RF02
Nombre del requisito	Dirigirse website laboratorio
Descripción	El sistema ofrecerá la posibilidad de acceder a la website del laboratorio. Simplemente es una redirección que ofrece la plataforma web para que el usuario pueda conocer en profundidad en qué consiste el laboratorio, quiénes son los dirigentes y qué campos de estudia alberga.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Media

Tabla 5. Requisito funcional RF03. Fuente: elaboración propia.

Identificación del requisito	RF03
Nombre del requisito	Dirigirse RRSS universidad
Descripción	El sistema ofrecerá la posibilidad de acceder a las distintas RRSS de la universidad, siendo estas YouTube, Facebook y Twitter. Simplemente es una forma de promocionar la universidad, así como de facilitar al usuario el seguimiento de esta.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Baja

Tabla 6. Requisito funcional RF04. Fuente: elaboración propia.

Identificación del requisito	RF04
Nombre del requisito	Crear usuario
Descripción	El sistema ofrecerá la posibilidad de crear un usuario para poder proseguir con el funcionamiento de la plataforma. El usuario tendrá que aportar un nombre de usuario, un correo electrónico y una contraseña. Si tanto el nombre de usuario como el correo electrónico no ha sido almacenado todavía, se creará este usuario y posteriormente se podrá *loguear* con estas credenciales.
Requerimiento no funcional	RNF01, RNF02, RNF03, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 7. Requisito funcional RF05. Fuente: elaboración propia.

Identificación del requisito	RF05
Nombre del requisito	*Loguear* usuario
Descripción	El sistema ofrecerá la posibilidad de que un usuario se identifique para proseguir con el funcionamiento de la plataforma. Esto solo se podrá conseguir si previamente se ha creado una cuenta de usuario. Para *loguearse* tendrá que proporcionar sus credenciales; usuario y contraseña.
Requerimiento no funcional	RNF01, RNF02, RNF03, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 8. Requisito funcional RF06. Fuente: elaboración propia.

Identificación del requisito	RF06
Nombre del requisito	Recuperar información usuario
Descripción	Una vez *logueado* el usuario, este podrá recuperar la información asociada a su cuenta y al estado en el que se encuentra durante el proceso de generación musical. Así pues, podrá recuperar su nombre de usuario y el correo asociado a la cuenta. Dependiendo de si el usuario ha seleccionado el entrenamiento o no de la IA, podrá también recuperar un listado de las MIDI Files que ha subido a la plataforma.
Requerimiento no funcional	RNF01, RNF02, RNF03, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 9. Requisito funcional RF07. Fuente: elaboración propia.

Identificación del requisito	RF07
Nombre del requisito	*Desloguear* usuario
Descripción	Una vez el usuario esté *logueado*, podrá *desloguearse* de la cuenta. Esto le llevará a comenzar el proceso de nuevo, ya que tendrá que efectuar el *login* para proseguir con la generación de melodías musicales.
Requerimiento no funcional	RNF01, RNF02, RNF03, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 10. Requisito funcional RF08. Fuente: elaboración propia.

Identificación del requisito	RF08
Nombre del requisito	Borrar usuario
Descripción	Una vez el usuario esté *logueado*, este podrá borrar su cuenta de la plataforma. Esta cuenta se borrará de la base de datos en la que se almacena, por lo que ya no podrá *loguearse* con esa cuenta.
Requerimiento no funcional	RNF01, RNF02, RNF03, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 11. Requisito funcional RF09. Fuente: elaboración propia.

Identificación del requisito	RF09
Nombre del requisito	Seleccionar checkpoint – No entrenar IA
Descripción	Una vez el usuario esté *logueado*, este podrá seleccionar si quiere entrenar la IA o quiere proseguir con una IA previamente entrenada (uso del *checkpoint*). La selección del *checkpoint* permite continuar con la generación y deshabilita la opción de proporcionar de MIDI Files al sistema. Si previamente se habían subido MIDI Files y luego se selecciona esta opción, estas MIDI Files se desecharán.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 12. Requisito funcional RF10. Fuente: elaboración propia.

Identificación del requisito	RF10
Nombre del requisito	Seleccionar subir MIDI – Entrenar IA
Descripción	Una vez el usuario esté *logueado*, este podrá seleccionar si quiere entrenar la IA o quiere proseguir con una IA previamente entrenada (uso del *checkpoint*). Si selecciona subir sus ficheros MIDI, es porque el usuario quiere entrenar la IA con los MIDI que este proporcione. Al seleccionar esta opción se deshabilita la opción de continuar hasta que el usuario suba como mínimo un fichero MIDI.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 13. Requisito funcional RF11. Fuente: elaboración propia.

Identificación del requisito	RF11
Nombre del requisito	Subir MIDI
Descripción	El sistema permitirá al usuario subir las MIDI Files que este quiera para entrenar la IA. El sistema se encargará que solamente ficheros con extensión '*.midi*' puedan ser subidos, ya que es el único tipo de fichero con el que se puede entrenar el algoritmo. Se mostrará una lista gráfica editable con las MIDI Files subidas por el usuario.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 14. Requisito funcional RF12. Fuente: elaboración propia.

Identificación del requisito	RF12
Nombre del requisito	Borrar MIDI
Descripción	El sistema permitirá al usuario borrar las MIDI Files que previamente hayan sido subidas. Las MIDI Files subidas se muestran en la plataforma mediante un listado gráfico editable. Desde este se podrán modificar y, por lo tanto, borrar.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 15. Requisito funcional RF13. Fuente: elaboración propia.

Identificación del requisito	RF13
Nombre del requisito	Seleccionar melodía de MIDI
Descripción	El usuario, una vez haya subido las MIDI Files, continuará con el entrenamiento y generación de la melodía. Si dentro de este listado de MIDI Files existe alguna MIDI File con más de una pista (es decir, con 2 o más voces), tendrá que escoger qué pista actuará como melodía.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 16. Requisito funcional RF14. Fuente: elaboración propia.

Identificación del requisito	RF14
Nombre del requisito	Generar MIDI sólo melodía
Descripción	El usuario, una vez haya subido las MIDI Files, continuará con el entrenamiento y generación de la melodía. Si dentro de este listado de MIDI Files existe alguna MIDI File con más de una pista (es decir, con 2 o más voces), tendrá que escoger qué pista actuará como melodía. De esta elección, se tendrá que extraer una MIDI File que contenga tan solo 1 pista, la cual será la melodía.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 17. Requisito funcional RF15. Fuente: elaboración propia.

Identificación del requisito	RF15
Nombre del requisito	Generar MIDI melodía y acordes
Descripción	El usuario, una vez haya subido las MIDI Files, continuará con el entrenamiento y generación de la melodía. Si dentro de este listado de MIDI Files existe alguna MIDI File con más de una pista (es decir, con 2 o más voces), tendrá que escoger qué pista actuará como melodía. De esta elección, se creará una MIDI File con dos pistas, la principal será la melodía escogida y la secundaria serán los acordes extraídos de la MIDI File anterior.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 18. Requisito funcional RF16. Fuente: elaboración propia.

Identificación del requisito	RF16
Nombre del requisito	Crear carpeta MIDIs sólo melodía
Descripción	El usuario, una vez haya subido las MIDI Files, continuará con el entrenamiento y generación de la melodía. Se tendrán que separar las MIDI Files que contienen una única pista (sólo melodía) con las MIDI Files que contienen 2 pistas (melodía y acordes). Esto se debe a que cada algoritmo se entrena con un tipo de MIDI File o con otro. Así pues, dentro del sistema donde se despliegue la aplicación, se creará y actualizará por usuario una carpeta que contenga las MIDI Files con solo melodía.
Requerimiento no funcional	RNF03, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 19. Requisito funcional RF17. Fuente: elaboración propia.

Identificación del requisito	RF17
Nombre del requisito	Crear carpeta MIDIs melodía y acordes
Descripción	El usuario, una vez haya subido las MIDI Files, continuará con el entrenamiento y generación de la melodía. Se tendrán que separar las MIDI Files que contienen una única pista (sólo melodía) con las MIDI Files que contienen 2 pistas (melodía y acordes). Esto se debe a que cada algoritmo se entrena con un tipo de MIDI File o con otro. Así pues, dentro del sistema donde se despliegue la aplicación, se creará y actualizará por usuario una carpeta que contenga las MIDI Files con melodía y acordes.
Requerimiento no funcional	RNF03, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 20. Requisito funcional RF18. Fuente: elaboración propia.

Identificación del requisito	RF18
Nombre del requisito	Seleccionar longitud melodía
Descripción	El sistema permitirá al usuario decidir cuál es la longitud de la melodía a generar. Esta puede tomar los siguientes valores, teniendo en cuenta un tiempo de 4 por 4: 8, 16 y 32.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 21. Requisito funcional RF19. Fuente: elaboración propia.

Identificación del requisito	RF19
Nombre del requisito	Seleccionar restricciones tono consecutivas
Descripción	El sistema permitirá al usuario decidir qué restricciones se aplican sobre el tono que toman las notas en los compases consecutivos que hayan sido seleccionadas de la melodía a generar. Las restricciones podrán ser: todas las notas el mismo tono, todas las notas un tono distinto, incrementar el tono durante estos compases consecutivos, decrementar el tono durante estos compases consecutivos, aumentos de tono por intervalo de 3ª mayor o aplicar silencios.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 22. Requisito funcional RF20. Fuente: elaboración propia.

Identificación del requisito	RF20
Nombre del requisito	Seleccionar restricciones duración consecutivas
Descripción	El sistema permitirá al usuario decidir qué restricciones se aplican sobre la duración de las notas en los compases consecutivos que hayan sido seleccionadas de la melodía a generar. Las restricciones podrán ser: todas las notas la misma duración, todas las notas una duración distinta, acelerar el tiempo (hacer cada vez las notas más cortas) o desacelerar el tiempo (hacer cada vez las notas más largas).
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 23. Requisito funcional RF21. Fuente: elaboración propia.

Identificación del requisito	RF21
Nombre del requisito	Seleccionar restricciones tono separadas
Descripción	El sistema permitirá al usuario decidir qué restricciones se aplican sobre el tono que toman las notas de los compases separados que hayan sido seleccionadas de la melodía a generar. Las restricciones podrán ser: los tonos de las notas de los compases separados serán los mismos o serán traspuestos un intervalo de 3ª mayor.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 24. Requisito funcional RF22. Fuente: elaboración propia.

Identificación del requisito	RF22
Nombre del requisito	Seleccionar restricciones duración separadas
Descripción	El sistema permitirá al usuario decidir qué restricciones se aplican sobre la duración que toman las notas de los compases separados que hayan sido seleccionadas de la melodía a generar. Las restricciones podrán ser: la duración de las notas es idéntica, se duplica la duración de las notas o se reduce a la mitad la duración de las notas.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 25. Requisito funcional RF23. Fuente: elaboración propia.

Identificación del requisito	RF23
Nombre del requisito	Guardar restricciones
Descripción	El sistema debe de permitir al usuario guardar las restricciones que seleccione sobre la melodía a generar. Estas restricciones serán representadas gráficamente indicando qué restricción es y sobre qué compases afecta. Estas restricciones luego se pasarán al modelo de IA para que las aplique al generar la melodía.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 26. Requisito funcional RF24. Fuente: elaboración propia.

Identificación del requisito	RF24
Nombre del requisito	Borrar restricciones
Descripción	El sistema debe de permitir al usuario borrar las restricciones sobre la melodía a generar que ya ha seleccionado. Las restricciones seleccionadas para borrar serán borradas de la lista en la que están guardadas, por lo que no se pasarán al modelo de IA.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 27. Requisito funcional RF25. Fuente: elaboración propia.

Identificación del requisito	RF25
Nombre del requisito	Obtener información RL-Tuner
Descripción	El sistema proporcionará información sobre el algoritmo RL-Tuner al usuario. Le proporcionará un enlace al repositorio de GitHub, así como información resumida sobre este.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Media/Baja

Tabla 28. Requisito funcional RF26. Fuente: elaboración propia.

Identificación del requisito	RF26
Nombre del requisito	Obtener información CMT_CPBP
Descripción	El sistema proporcionará información sobre el algoritmo CMT_CPBP al usuario. Le proporcionará un enlace al repositorio de GitHub, así como información resumida sobre este.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Media/Baja

Tabla 29. Requisito funcional RF27. Fuente: elaboración propia.

Identificación del requisito	RF27
Nombre del requisito	Escoger proyecto RL-Tuner
Descripción	El sistema dará la opción al usuario de escoger el algoritmo RL-Tuner para la generación de la melodía musical. Esta opción será disponible siempre y cuando se haya seleccionado usar un checkpoint o se tengan MIDI Files con 1 pista (melodía).
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF05, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 30. Requisito funcional RF28. Fuente: elaboración propia.

Identificación del requisito	RF28
Nombre del requisito	Escoger proyecto CMT_CPBP
Descripción	El sistema dará la opción al usuario de escoger el algoritmo CMT_CPBP para la generación de la melodía musical. Esta opción será disponible siempre y cuando se haya seleccionado usar un *checkpoint* o se tengan MIDI Files con 2 pistas (melodía + acordes).
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF05, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 31. Requisito funcional RF29. Fuente: elaboración propia.

Identificación del requisito	RF29
Nombre del requisito	Descargar MIDI generada
Descripción	El sistema dará la opción al usuario de descargar la MIDI File que se ha generado tras todo el proceso.
Requerimiento no funcional	RNF01, RNF02, RNF03, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

Tabla 32. Requisito funcional RF30. Fuente: elaboración propia.

Identificación del requisito	RF30
Nombre del requisito	Reproducir MIDI generada
Descripción	El sistema dará la opción al usuario de reproducir sin necesidad de descargar la MIDI File que se ha generado tras todo el proceso.
Requerimiento no funcional	RNF01, RNF02, RNF04, RNF06, RNF07
Prioridad del requisito	Alta

3.1.2 Requisitos no funcionales

La siguiente tabla recopila los requisitos no funcionales de la aplicación.

Tabla 33. Recopilación de requisitos no funcionales. Fuente: elaboración propia.

Identificación del requisito	**Nombre del requisito**
RNF01	Interfaz de la aplicación con facilidad de uso
RNF02	Interfaz de la aplicación guiada
RNF03	Seguridad de la información
RNF04	Desplegable desde cualquier navegador
RNF05	Disponibilidad del sistema
RNF06	Emplear tecnologías actuales
RNF07	Eficiencia del sistema

A continuación, detallaremos cada requisito no funcional siguiendo una plantilla del estándar IEEE 830.

Tabla 34. Requisito no funcional RNF01. Fuente: elaboración propia.

Identificación del requisito	RNF01
Nombre del requisito	Interfaz de la aplicación con facilidad de uso
Descripción	La interfaz gráfica que presente la aplicación tiene que ser amigable e intuitiva para el usuario, sin que la facilidad de uso se vea comprometida.
Prioridad del requisito	Alta

Tabla 35. Requisito no funcional RNF02. Fuente: elaboración propia.

Identificación del requisito	RNF02
Nombre del requisito	Interfaz de la aplicación guiada
Descripción	La interfaz gráfica que presente la aplicación tiene que guiar al usuario durante todo el proceso de generación de una melodía musical. De esta forma, para llegar al objetivo final el usuario no debe de dudar de lo que está haciendo.
Prioridad del requisito	Alta

Tabla 36. Requisito no funcional RNF03. Fuente: elaboración propia.

Identificación del requisito	RNF03
Nombre del requisito	Seguridad de la información
Descripción	A pesar de que el sistema no va a almacenar información sensible del usuario, este tiene que garantizar la seguridad de los datos almacenados del usuario.
Prioridad del requisito	Alta

Tabla 37. Requisito no funcional RNF04. Fuente: elaboración propia.

Identificación del requisito	RNF04
Nombre del requisito	Desplegable desde cualquier navegador
Descripción	La interfaz gráfica del sistema tiene que poderse utilizar y visualizarse correctamente desde cualquier navegador web. De esta forma, la usabilidad del sistema no se ve afectada por el explorador que el usuario utilice.
Prioridad del requisito	Alta

Tabla 38. Requisito no funcional RNF05. Fuente: elaboración propia.

Identificación del requisito	RNF05
Nombre del requisito	Disponibilidad del sistema
Descripción	El sistema tiene que estar disponible y en funcionamiento los 7 días de la semana, las 24 horas al día. Así, no limitamos al usuario del uso de la plataforma.
Prioridad del requisito	Alta

Tabla 39. Requisito no funcional RNF06. Fuente: elaboración propia.

Identificación del requisito	RNF06
Nombre del requisito	Emplear tecnologías actuales
Descripción	El sistema debe de ser desarrollado utilizando tecnologías actuales y modernas para una mayor adaptación al entorno actual y para poder dar pie a una escalabilidad del sistema si se requiere. El hecho de usar tecnologías modernas implica que es más sencillo escalar estas y adaptarlas al nuevo entorno.
Prioridad del requisito	Alta

Tabla 40. Requisito no funcional RNF07. Fuente: elaboración propia.

Identificación del requisito	RNF07
Nombre del requisito	Eficiencia del sistema
Descripción	El sistema debe funcionar lo más eficientemente posible. De esta manera, mejorará la usabilidad del usuario ya que se eliminan o reducen los tiempos de espera.
Prioridad del requisito	Alta

3.1.3 Casos de uso

Para proseguir con el análisis del problema, se recogerán los Requisitos Funcionales en los casos de uso del usuario. Así pues, ahora presentaremos qué interacciones tiene el usuario con el sistema de manera general, especificando en qué situación podrá ejercer ciertas funcionalidades previamente descritas.

Para el correcto entendimiento de los casos de uso cabe destacar ciertos aspectos fundamentales. Existen algunos casos de uso que se extenderán de otros casos de uso mediante la etiqueta <<*extends*>>. Estos casos de uso solo tienen sentido si el caso de uso base del que se extienden se ha realizado, básicamente porque extienden su funcionalidad. Para el funcionamiento del caso base los casos extendidos son opcionales.

Otro aspecto fundamental es el recogido por la etiqueta <<*include*>>. Los casos de uso con la etiqueta <<*include*>> forman una relación directa entre 2 casos de uso, donde el para realizar el caso base necesitaremos previamente realizar el caso de uso con la etiqueta. Para el funcionamiento del caso base los casos incluidos son obligatorios, ya que el caso de uso base está incompleto sin la realización de los incluidos.

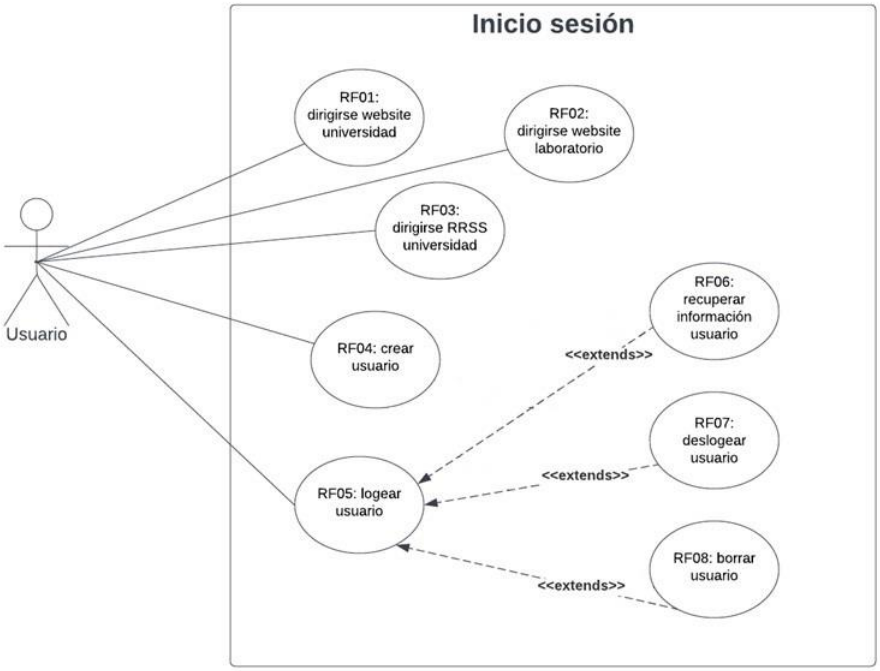

Figura 8. Diagrama de caso de uso Login (inicio de sesión). Fuente: elaboración propia.

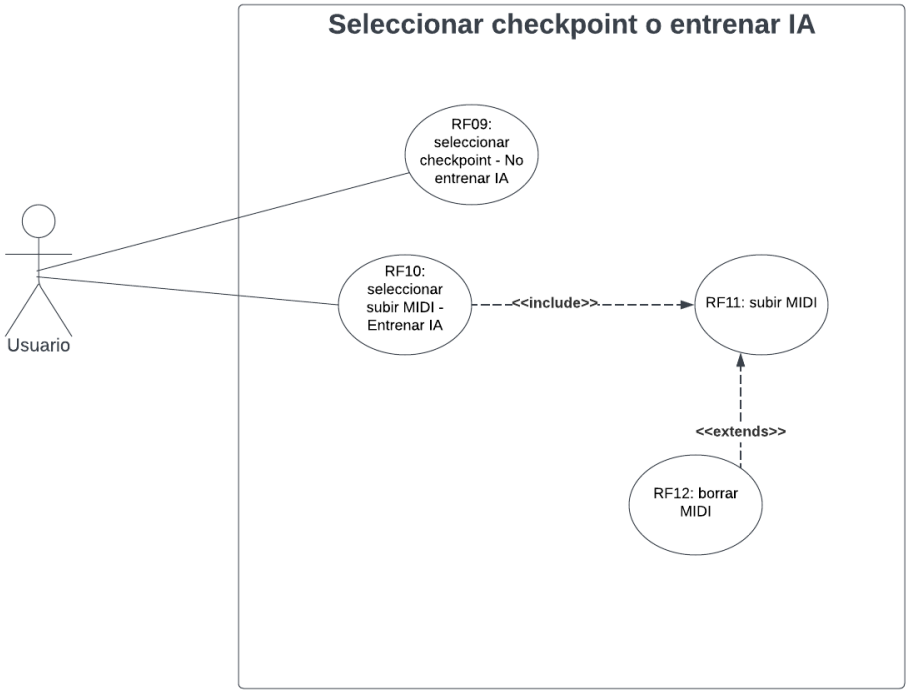

Figura 9. Diagrama de caso de uso entrenar IA o seleccionar checkpoint. Fuente: elaboración propia.

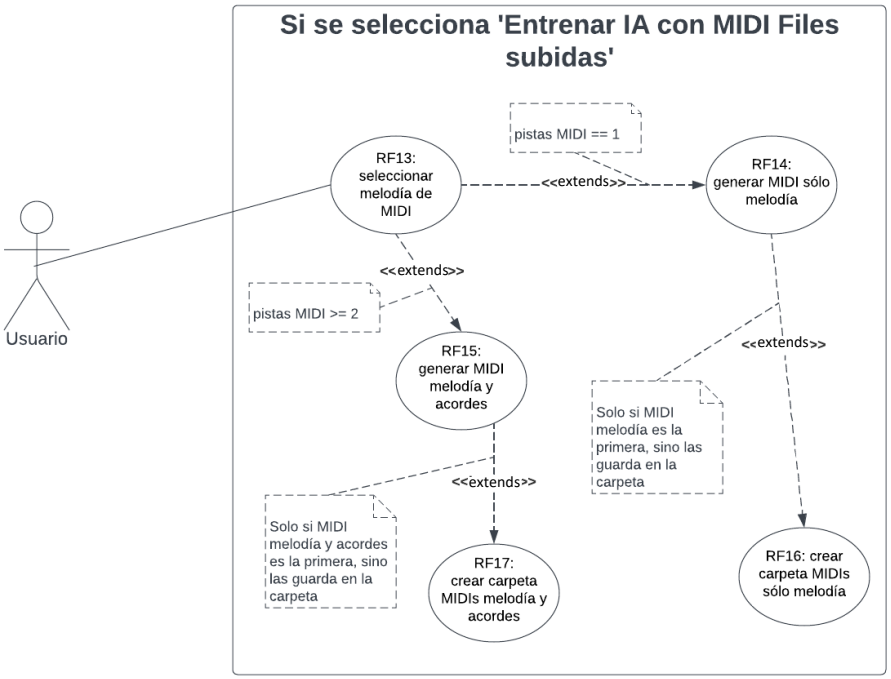

Figura 10. Diagrama de caso de uso selección de melodía si se entrena IA. Fuente: elaboración propia.

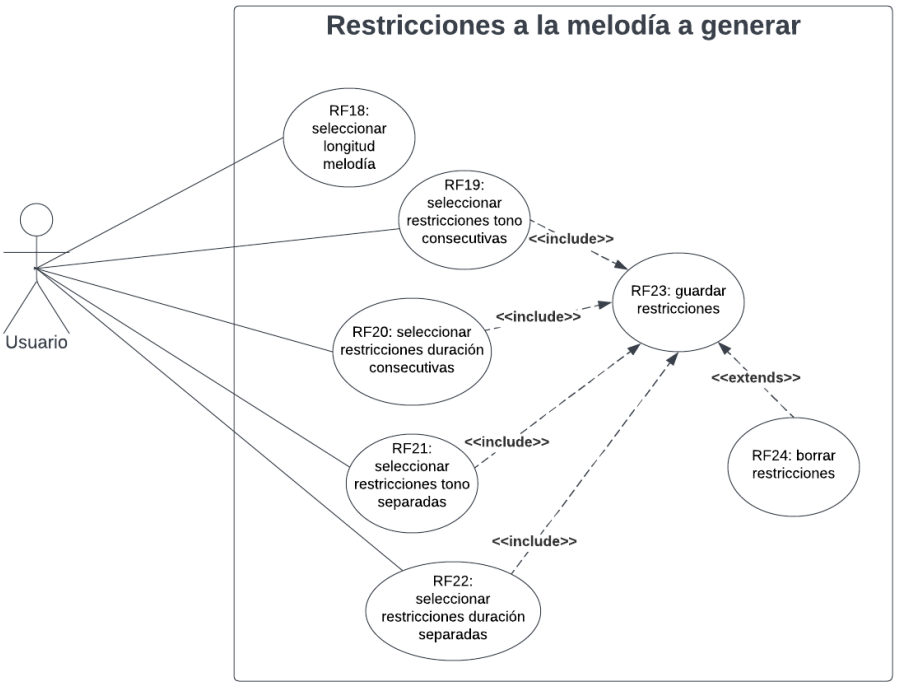

Figura 11. Diagrama de caso de uso restricciones a la melodía a generar. Fuente: elaboración propia.

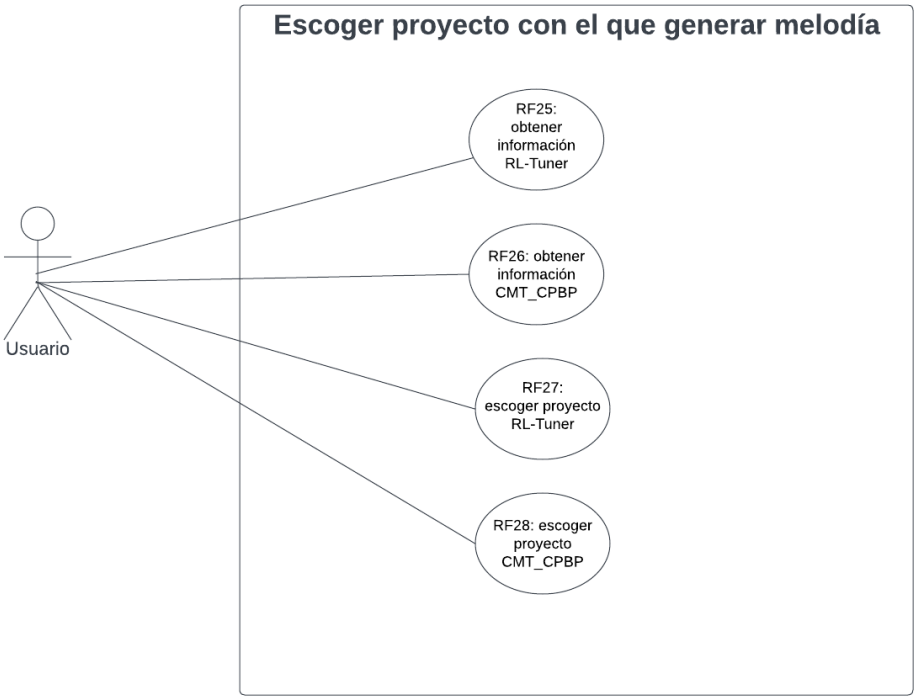

Figura 12. Diagrama de caso de uso escoger proyecto. Fuente: elaboración propia.

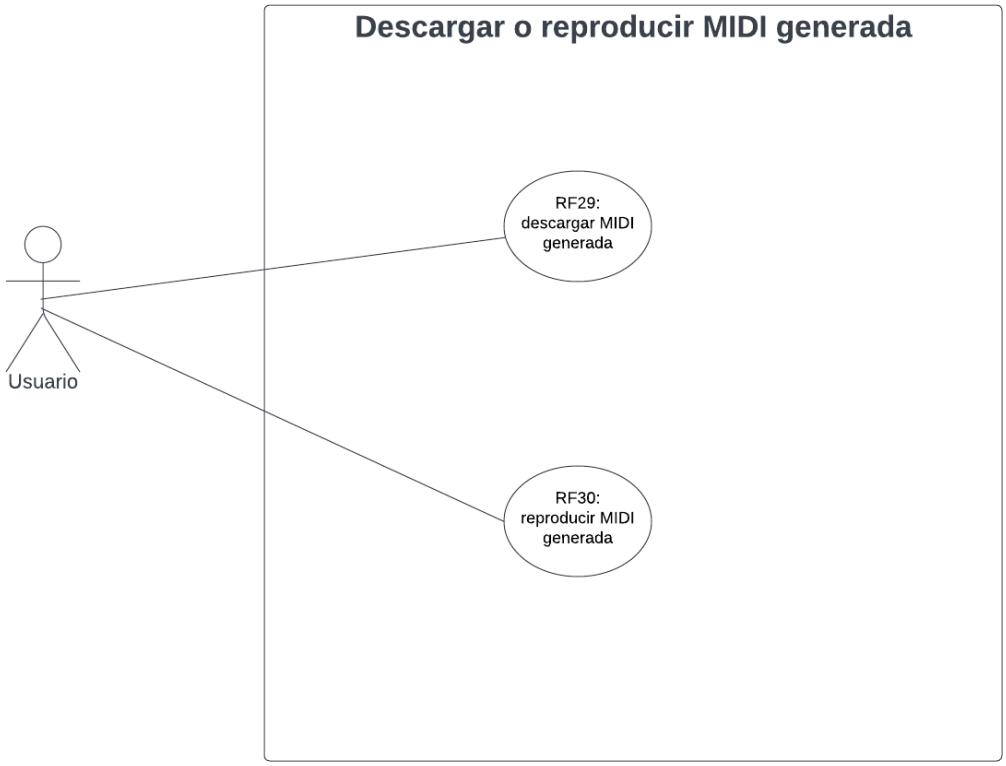

Figura 13. Diagrama de caso de uso descargar o reproducir MIDI generada. Fuente: elaboración propia.

3.1.4 Modelo de datos

El modelado de datos representa la estructura estática de la solución que se le quiere proponer al problema. En nuestra plataforma web no necesitamos almacenar datos más allá que los necesarios para el *login* del cliente y alguna característica que se explicará de este posteriormente. Tanto las MIDI Files que subirán los clientes en caso de entrenamiento como las restricciones a la melodía no se van a almacenar, ya que esta información simplemente se añadirá a la información que tiene la IA para poder ser entrenada y/o generar la melodía.

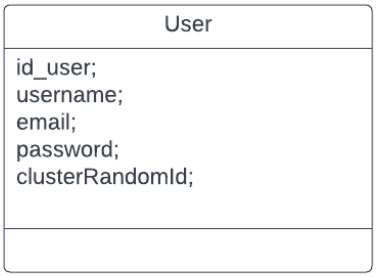

Figura 14. Diagrama de clase. Fuente: elaboración propia.

Como se ha comentado, el modelo de datos solo requiere almacenar una mínima información sobre el cliente. El ID del cliente será necesario para listar a los clientes en la base de datos, mientras que el usuario, *email* y contraseña serán necesarios para crear la cuenta y posteriormente poder *loguearse*. Como se puede observar, el último atributo comprende un ID aleatorio para el clúster.

Debido a que no se ha abordado la parte de arquitectura todavía, no se ha introducido la función del clúster en este proyecto. Sin embargo, para entender el por qué de este atributo, se introducirá en esta sección.

El cliente, para entrenar y/o generar la melodía mediante el uso de uno de los dos proyectos, necesitará enviar el trabajo a un clúster que la universidad utiliza para proyectos de gran computación, ComputeCanada. El clúster de ComputeCanada es un clúster creado por el gobierno canadiense para procesar proyectos que requieren una gran computación. Este clúster es ofrecido a las universidades de Canadá, así como a ciertas empresas para poder ser usado en sus investigaciones o trabajos (Witt, 2019). Así pues, los proyectos del laboratorio que requerían de gran computación (como los dos proyectos que ofrecemos en la plataforma) enviaban sus trabajos al clúster y luego recibían la respuesta de este, con el cálculo ya generado. Como era obvio, y para también dotar de rapidez a la plataforma, un usuario localmente no puede hacerse cargo del cómputo del algoritmo para entrenarlo o para que genere la melodía. Así pues, los cómputos se envían a este clúster al que el laboratorio tiene acceso. El ID del clúster que cada usuario tiene simplemente es una identificación del trabajo enviado, para que así, una vez terminado, este usuario sea capaz de recibir el resultado.

3.2 Solución propuesta

Tras haber identificado y especificado los requisitos, casos de uso y el modelo de datos a utilizar, se puede comentar la solución negociada con el cliente. Primero, se negoció que las fases por las que se pasará para el desarrollo de la solución serán las marcadas por el apartado de plan de trabajo, que recoge las fases de la metodología en cascada que estamos usando para la plataforma.

Tras esto, se acordó con el cliente que el *frontend* de la aplicación se realizará con HTML, CSS y el *framework* de CSS denominado Bootstrap. Para dotar de dinamismo al *frontend* se utilizará JavaScript y las librerías que se necesiten. El *backend* de la aplicación se programará en Python, usando el *framework* de Flask. Para la base de datos, se creará una API en C# utilizando el módulo .NET. La conexión sistema-base de datos se realizará mediante peticiones HTTP

3.3 Plan de trabajo

El diagrama de Gantt que se presenta a continuación (Figura 15) sigue la metodología en cascada que se ha mencionado anteriormente. Como se puede observar, se recogen 4 de las 5 fases que incluye la metodología, ya que la fase de Mantenimiento no se planifica.

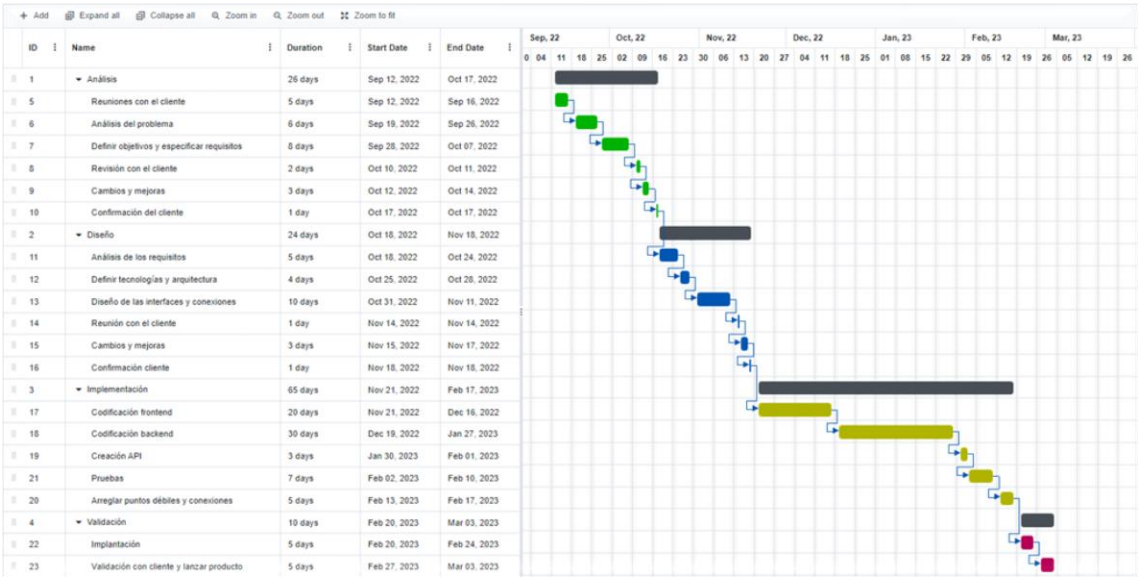

Figura 15. Diagrama de Gantt. Fuente: elaboración propia.

El diagrama de Gantt presenta las fases por las que hemos pasado para desarrollar el producto junto con las tareas que cada una tiene. Como se puede contemplar, tanto las fases como las tareas de cada fase son representadas de manera secuencial, presentando dependencias. Esto es así por el tipo de metodología que utilizamos, donde hasta no acabar una fase o tarea no se puede pasar a la otra.

En la fase de Análisis, las principales tareas fueron reunirse con el cliente para recopilar información, analizar el problema y definir los objetivos, así como acotar los Requisitos Funcionales. Una vez hecho esto, se concertó otra reunión con el cliente para revisarlos, cambiar lo necesario y confirmarlo.

Seguidamente se pasó a la fase de Diseño, donde la primera tarea fue analizar profundamente los Requisitos Funcionales recopilados. Posteriormente, se definió la arquitectura a seguir y las tecnologías a utilizar. En base a esto se realizaron bocetos de la plataforma y se diseñaron las futuras conexiones de su arquitectura. Para cerrar esta fase se tuvo otra reunión con el cliente para discutir lo diseñado, aplicar cambios puntuales y confirmar su satisfacción.

La tercera fase fue la fase de Implementación y fue la más costosa de todas. Esta tuvo una duración de 65 días y consistió en la codificación del *frontend* y el *backend* de la plataforma, de la creación de la Rest API, de la conexión de la Rest API con la base de datos relacional SQL Express interna y de la realización de pruebas manuales para comprobar que el funcionamiento total era el correcto. Debido a que el programador tenía que aprender desde 0 diversos lenguajes y *frameworks,* surgieron una multitud de errores durante la implementación. De esta manera se justifica el por qué de tan larga duración. Cuando ya se realizaron todas las tareas que previamente hemos comentado antes, se arreglaron ciertos puntos débiles de la plataforma.

En la fase final, la de Validación, se puso en marcha la plataforma y se el mostró al cliente el producto final. Este la revisó cuidadosamente hasta que dio el visto bueno y consideró que se podía lanzar el producto.

4. Diseño de la solución

En el apartado anterior hemos recopilado todos los Requisitos Funcionales que se le exigen a nuestra aplicación, así como diseñado todos los casos de uso que describen la interacción entre Sistema - Usuario y el modelo de datos necesario para el correcto funcionamiento.

Teniendo en cuenta todo esto, en este apartado describiremos qué decisiones hemos llevado a cabo para diseñar una solución que satisfaga todo lo recogido en el análisis del problema a resolver.

Primeramente, describiremos cuál es la arquitectura llevada a cabo. Posteriormente, se detallará el diseño de forma más efusiva, separándolo por las 3 capas que lo componen.

4.1 Arquitectura del sistema

La arquitectura diseñada sigue las pautas de una arquitectura de 3 capas. Como podemos observar, tenemos 3 partes las cuales comprenden el cliente, el servidor web y la API desde la que se accede a la base de datos. Estas 3 partes se dividen en capa de presentación, capa lógica y capa de datos.

La capa de presentación está formada por el cliente, el cual cuando acceda a la plataforma web visualizará esta cargando mediante el navegador toda la información asociada al despliegue *frontend* y la comunicación con el servidor web.

La capa lógica está formada por el servidor web, y es aquí donde la funcionalidad interna del sistema se lleva a cabo. Esta sirve como intermediaria entre el cliente y la base de datos. Cuando el cliente realiza alguna petición sobre el sistema, esta se procesa en el servidor web y o bien se responde o bien se contacta con la base de datos, para así una vez recuperada esa información devolvérsela al cliente.

Un aspecto a destacar y totalmente distinto de las arquitecturas multicapa que se suelen ver es la aparición del clúster. Como previamente se ha introducido, el laboratorio tiene acceso a un clúster del gobierno canadiense, al cual envían los trabajos que requieren de mucho cómputo para así resolverlos en un tiempo razonable. Así pues, cuando el usuario busque entrenar el algoritmo o simplemente generar la música, este trabajo costoso se enviará desde el servidor web al clúster y se esperará a su respuesta. Una vez recibida, se ofrecerá al cliente el resultado, que viene siendo una MIDI File con la melodía generada. Esto se abordará solamente de manera conceptual, ya que por problemas burocráticos el acceso al clúster durante el desarrollo del proyecto estaba vetado, y simplemente se puso en escena, sin llegar a implementar ninguna conexión con este hasta nuevo aviso del gobierno para poder acceder a ComputeCanada.

Para terminar con la introducción de la capa lógica, las conexiones que se realizan entre servidor web y la API donde se alberga la base de datos se hacen mediante el protocolo HTTP. Las conexiones entre capa de presentación y capa lógica también se realizan mediante HTTP, pero se hace uso de otras herramientas proporcionadas por el framework Flask.

Como punto final, la capa de datos engloba todo lo relacionado con la persistencia de la información del sistema. Aquí encontraremos una base de datos a la cual se accede mediante llamadas a una Rest API, la cual almacena información sobre el usuario.

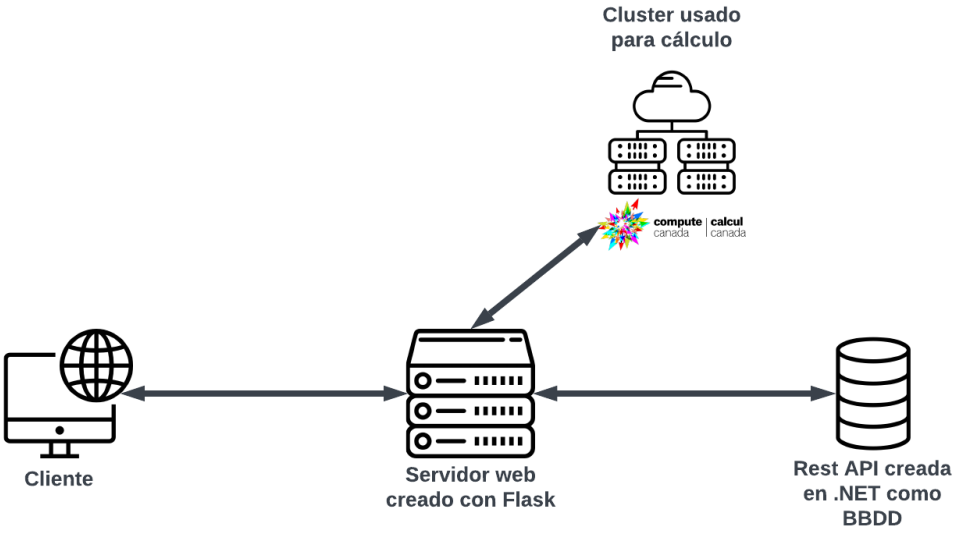

Figura 16. Arquitectura del sistema. Fuente: elaboración propia.

4.2 Diseño detallado

Aquí detallaremos de manera aproximada lo introducido anteriormente dividiéndolo por capas, pero sin llegar a desarrollar completamente la solución diseñada, ya que esta parte la dejaremos para el capítulo que trata sobre el desarrollo de la solución propuesta. Así pues, explicaremos en grosso modo en qué consiste cada capa, cómo está formada y cómo interactúa con las otras.

Es importante resaltar que las explicaciones sobre la capa de presentación y la capa lógica se basarán en los casos de uso que previamente hemos creado. Así pues, se verá una notable sinergia entre las explicaciones de estas.

Para la capa de datos, nos basaremos en el modelado de datos del apartado 3.1.4, donde describíamos qué datos del usuario necesitamos.

4.2.1 Capa de presentación

Previamente mencionábamos que la capa de presentación estaba formada por el cliente, representando el mecanismo de interacción entre el usuario y el sistema. El objetivo de esta capa es comunicar al usuario con el sistema de una forma amigable, así como capturar toda la información que este transmite al sistema a través de esta.

Cuando el cliente acceda al dominio donde se encuentra la plataforma web, este la visualizará cargando mediante el navegador utilizado toda la información asociada a la parte visual de la plataforma.

Así pues, el cliente desde su navegador web (Google Chrome es el recomendado como principal por las distintas ventajas que ofrece) accede al dominio de la plataforma. Una vez realizada la conexión, el navegador descargará los ficheros que el servidor web le proporcione para poder desplegar correctamente el *frontend* de la plataforma, estando estos compuestos en nuestro diseño por ficheros HTML, CSS y JavaScript. Cuando ya consiga visualizarlo, toda interacción que se tenga con el sistema se enviará a este mediante las conexiones descritas dentro del fichero JavaScript, desde el cual se conecta la capa de presentación con la capa lógica. Estas conexiones se realizarán mediante el protocolo HTTP, para acceder así a los *endpoints* de nuestro servidor web o mediante herramientas proporcionadas por el *framework* Flask (las cuales pueden ser usadas desde ficheros HTML), usado para construir la funcionalidad del servidor web.

Basándonos en los casos de uso del apartado 3, presentaremos aquí los bocetos visuales diseñados para la plataforma web. Seguidamente se introducirá el diagrama de navegación de la plataforma web.

Bocetos

Los dos siguientes bocetos (Figura 17 y Figura 18) están asociados al primer diagrama de casos de uso '*Login*'. En el primer boceto (Figura 17) podemos observar que existen partes que nos permiten dirigirnos a la información de la universidad, del laboratorio y de sus redes sociales, aparte de otras funcionalidades extras. Desde aquí el usuario podrá *loguearse*, ya bien mediante el botón de *Login* o mediante el botón de usuario. Cabe recalcar que hasta que el usuario no se *loguea* no se puede proseguir con la funcionalidad de la plataforma. Los requisitos funcionales que se cumplen en la Figura 17 son: RF01, RF02, RF03, RF04 y RF05.

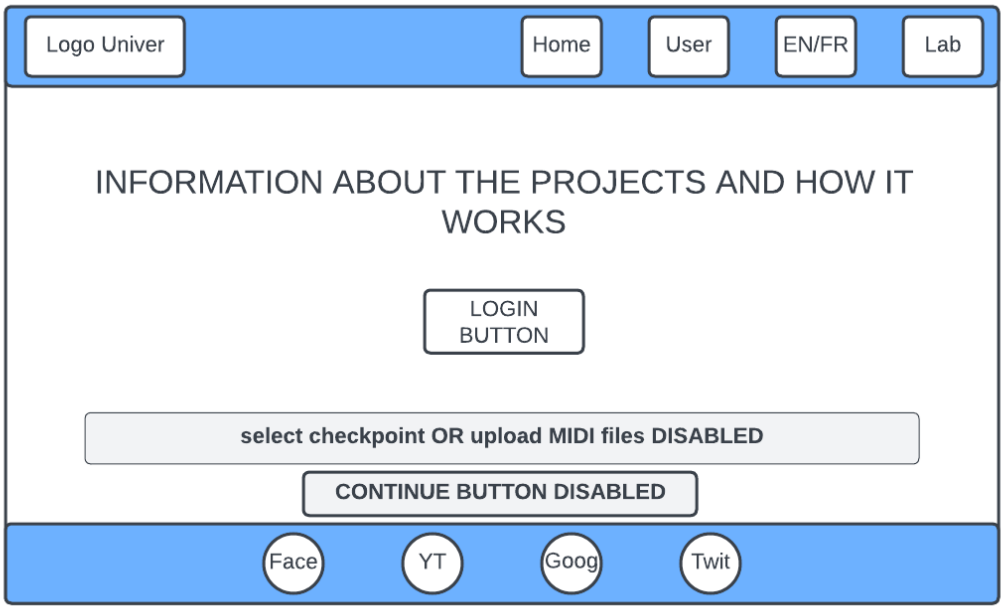

Figura 17. Boceto inicio sesión. Fuente: elaboración propia.

El boceto representado por la figura 18 muestra la pantalla que aparece cuando el usuario aprieta el botón de *Login*. Desde aquí, este podrá *loguearse* o crear una cuenta. Los requisitos funcionales que se cumplen en la Figura 18 son: RF04 y RF05.

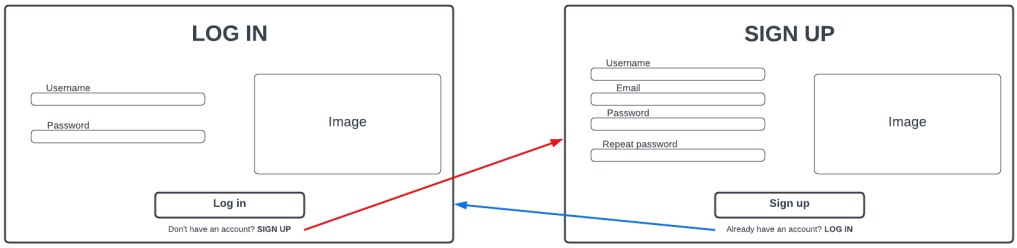

Figura 18. Boceto login/sign up usuario. Fuente: elaboración propia.

En los dos siguientes bocetos (Figura 19 y Figura 20) podremos observar cómo se activan las funcionalidades recopiladas por el diagrama de casos de uso 'entrenar IA o seleccionar *checkpoint*'. Debido a que esta pantalla se accede una vez el usuario se ha *logueado*, cuando presione el botón de usuario accederá a la información de usuario (como se aprecia en la figura 20), así como a las opciones de *desloguearse* y de borrar la cuenta. Esto último estaba recopilado en el diagrama de casos de uso de '*Login*'. También hay que destacar que por la estructuración de la plataforma, prácticamente desde cualquier ventana se podrá acceder a la web de la universidad, la web del laboratorio y a las RRSS de la universidad. Los requisitos funcionales que se cumplen en la Figura 19 son: RF01, RF02, RF03, RF06, RF07, RF08, RF09, RF10, RF11 y RF12.

Cuando presionemos el botón de usuario la Figura 20 se mostrará por encima de la Figura 19, mostrándonos información y funcionalidad relativa al usuario. Los requisitos funcionales que se cumplen en la Figura 20 son: RF06, RF07 y RF08.

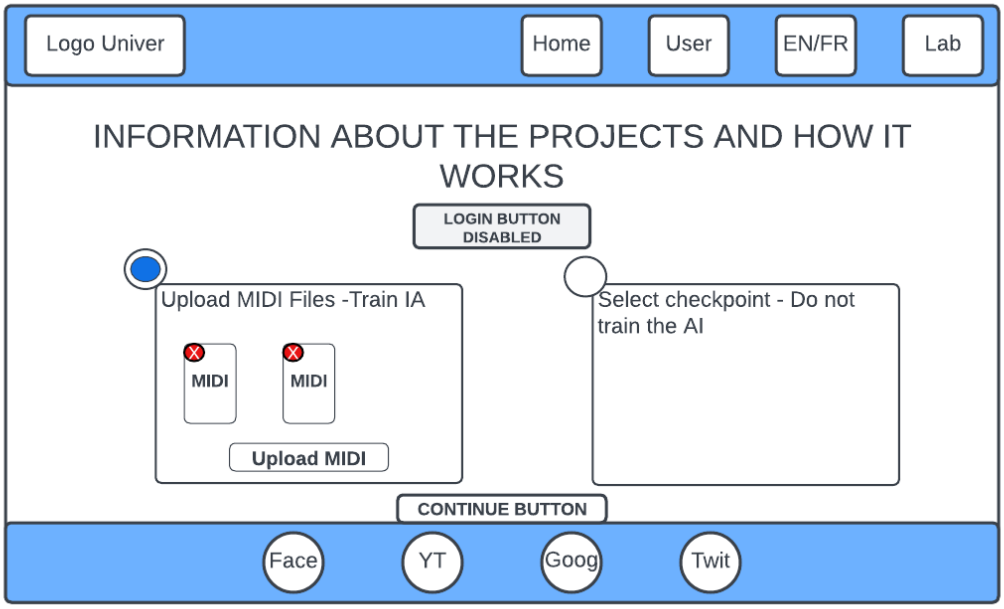

Figura 19. Boceto entrenar IA o seleccionar checkpoint. Fuente: elaboración propia.

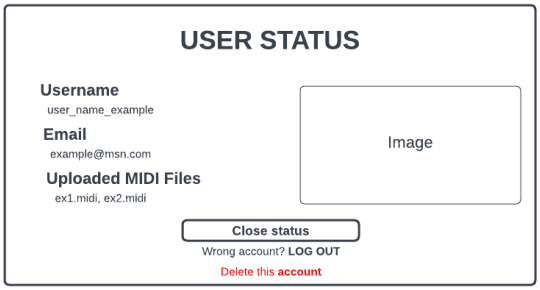

Figura 20. Boceto usuario logueado. Fuente: elaboración propia.

Si se ha seleccionado no entrenar la IA (*checkpoint*), no tendremos MIDI Files subidas, por lo que la siguiente ventana la esquivaremos (Figura 21). Si se decide entrenar la IA con nuestras MIDI Files, tendremos que subirlas al sistema y después escoger qué pista o parte actuará como melodía. Por lo tanto, el sistema mostrará la Figura 21 antes de pasar a seleccionar las restricciones de la melodía a generar. Esta ventana se asocia con el diagrama de casos de uso 'selección de melodía si se entrena IA'. Cabe destacar que los casos de uso relacionados con generar la MIDI y crear su respectiva carpeta se procesan internamente por el sistema, por lo que el usuario no tiene que hacer nada.

Los requisitos funcionales que se cumplen en la Figura 21 son: RF13, RF14, RF15, RF16 y RF17.

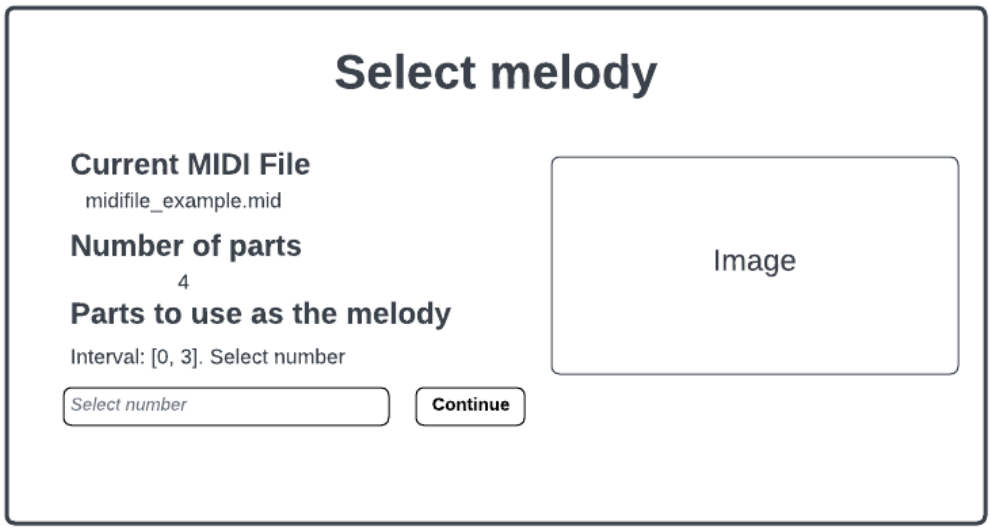

Figura 21. Boceto seleccionar melodía de las diferentes pistas de la MIDI. Fuente: elaboración propia.

La siguiente ventana de la aplicación (Figura 22) permitirá al usuario seleccionar las restricciones que quiere aplicar sobre la melodía a generar. Como se puede observar, este boceto comprende todos los casos de uso recopilados en el diagrama de casos de uso 'restricciones a la melodía a generar'. Una vez se tengan todas las restricciones creadas sobre la melodía a generar, el sistema las guardará y se pasará a la ventana de selección de proyecto, para escoger con qué algoritmo querremos generar nuestra melodía.

Los requisitos funcionales que se cumplen en la Figura 22 son: RF01, RF02, RF03, RF06, RF07, RF08, RF18, RF19, RF20, RF21, RF22, RF23 y RF24.

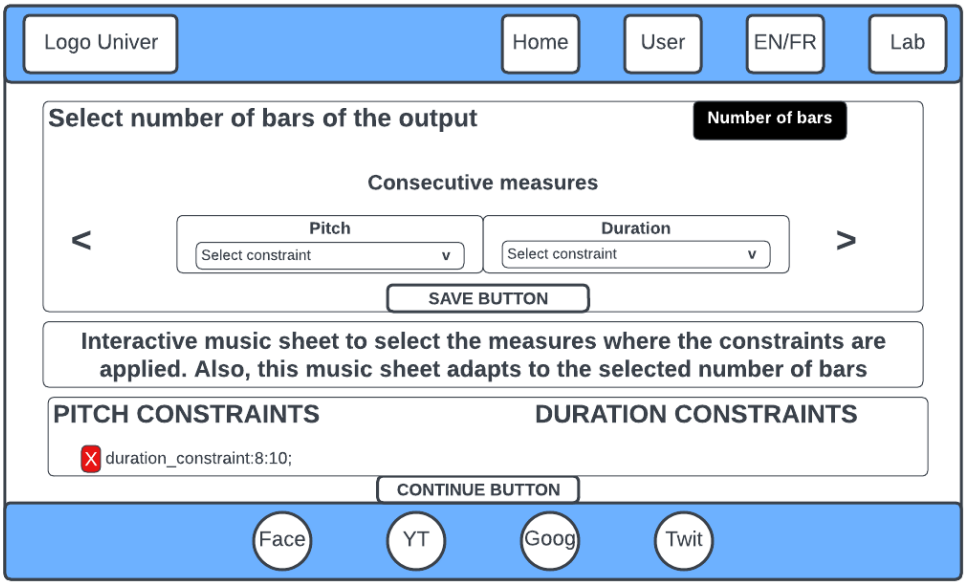

Figura 22. Boceto seleccionar restricciones para la melodía a generar. Fuente: elaboración propia.

En la siguiente ventana (Figura 23), a la cual accederemos una vez hayamos registrado todas las restricciones que queremos aplicar sobre nuestra melodía, podremos obtener información sobre los proyectos, así como escoger uno de estos para generar la melodía. Cuando el proyecto se escoja, el trabajo se enviará al clúster junto a toda la información aportada por el usuario, y este esperará a que llegue el resultado (MIDI File generada). Este boceto comprende todos los casos de uso recopilados en el diagrama de casos de uso 'escoger proyecto'.

Los requisitos funcionales que se cumplen en la Figura 23 son: RF01, RF02, RF03, RF06, RF07, RF08, RF25, RF26, RF27 y RF28.

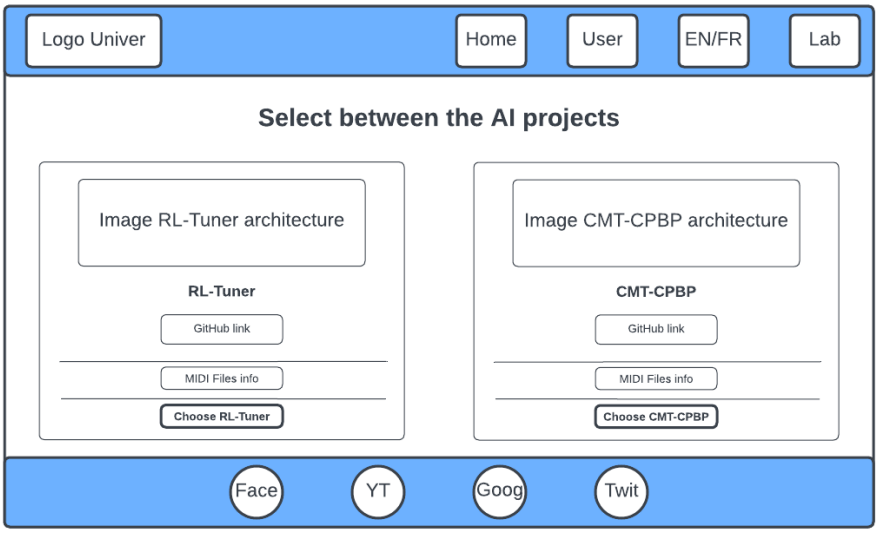

Figura 23. Boceto escoger proyecto. Fuente: elaboración propia.

Finalmente, la Figura 24 nos mostrará la MIDI File generada por el algoritmo de IA escogido. Desde aquí podremos reproducir la MIDI generada, descargarla o volver a generar una nueva MIDI, empezando de nuevo el proceso. Esta última opción nos llevaría a la Figura 19. Este boceto está asociado al último diagrama de casos de uso descrito, 'descargar o reproducir MIDI generada'.

Los requisitos funcionales que se cumplen en la Figura 24 son: RF01, RF02, RF03, RF06, RF07, RF08, RF29 y RF30.

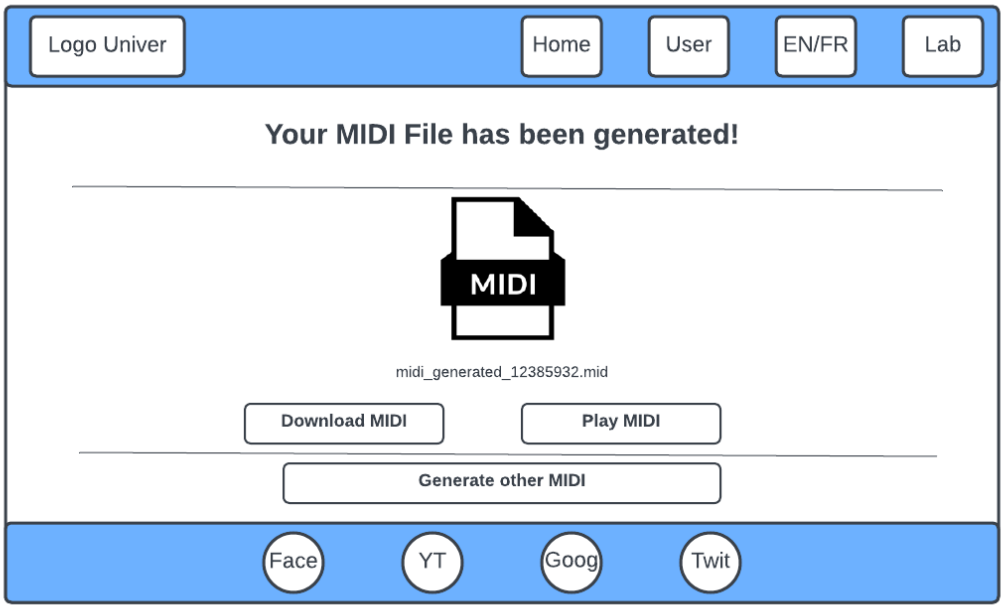

Figura 24. Boceto descargar o reproducir MIDI generada. Fuente: elaboración propia.

Diagrama de navegación

Una vez presentados todos los bocetos que compondrán la plataforma, presentamos el diagrama de navegación por el que está compuesta. Para el correcto entendimiento del diagrama, las flechas sin punta significan que la navegabilidad entre las páginas es bidireccional. Las flechas con punta significan que la navegabilidad entre las páginas es unidireccional. La página estado usuario podrá volver solamente a la página desde la que ha sido accedida.

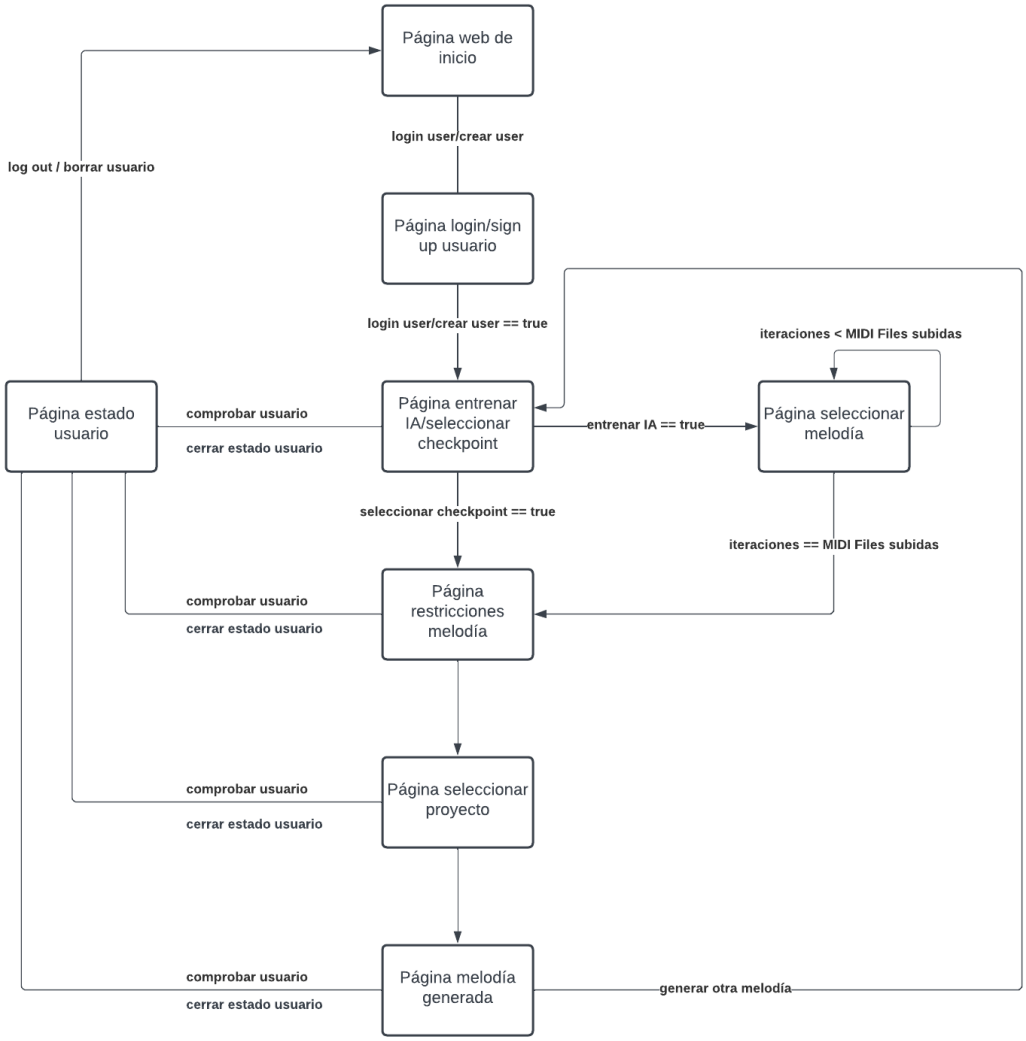

Figura 25. Diagrama de navegación. Fuente: elaboración propia.

4.2.2 Capa lógica

La capa lógica, o también conocida como capa de negocio, es la capa que actúa como intermediaria entre la capa de presentación y la capa de datos. Aparte de esto, es la capa que comprende todas las funcionalidades internas del sistema y que, una vez realizadas, proporciona el resultado de estas al usuario mediante la capa de presentación. Por lo tanto, es en esta capa donde se concentra la gestión interna del sistema.

Anteriormente habíamos introducido que para crear nuestro servidor web haríamos uso del *framework* Flask, desarrollado en Python, así como de Python y ciertas librerías que serían necesarias. Como no disponíamos de ningún dominio ni de ningún lugar web donde desplegar nuestra aplicación, se decidió desarrollarla en el ordenador, utilizando este como servidor web.

Para abordar el diseño de la capa lógica, referenciaremos cada caso de uso y cada funcionalidad para brindar una perspectiva de cómo se desarrollará posteriormente. No obstante, primero hay que explicar la estructuración del proyecto.

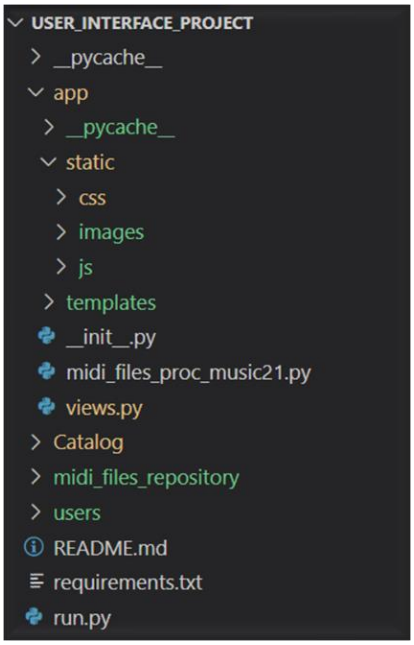

Figura 26. Estructuración del proyecto. Fuente: elaboración propia.

El proyecto está estructurado principalmente por lo contenido en la carpeta '*app*' y lo contenido en la carpeta '*Catalog*'. En la carpeta '*app*' tenemos todo lo relacionado con el desarrollo de la plataforma web, desde el *frontend* hasta el *backend*. El *frontend* estaría compuesto por la carpeta '*static*' y la carpeta '*templates*'.

La carpeta '*static*' hace referencia a toda la información estática del proyecto. Esta abarca las imágenes que utilizamos como adorno de la parte visual de la aplicación (carpeta '*images*'), el CSS utilizado para estilar los ficheros HTML (carpeta '*css*') así como los archivos JavaScript que dotan de dinamismo al *frontend* y establecen las conexiones de la capa de presentación con la capa lógica (carpeta '*js*'). En la carpeta '*templates*' tendremos todos los ficheros HTML utilizados en la plataforma web.

La funcionalidad del *backend* estaría compuesta por el archivo '*views.py*', desde el cual se gestionan todos los *endpoints* de la aplicación, la funcionalidad de la aplicación, la gestión de llamadas a la API, la gestión de llamadas desde la capa de presentación, etc. Cuando este archivo tenga que tratar con las MIDI Files, seleccionar la pista que va a actuar como melodía, crear las carpetas asociadas a estas MIDI Files… hará uso del archivo '*midi_files_proc_music21.py*', un archivo que utiliza una biblioteca desarrollada por el MIT para tratar archivos MIDI.

La carpeta '*__pycache__*' y el archivo '*__init__.py*' son archivos relacionados con el despliegue de la aplicación.

Fuera de la carpeta '*app*', la principal carpeta que debemos tener en cuenta es la carpeta '*Catalog*', donde está desarrollada la API en .NET que accede a la base de datos del sistema, la cual es una base de datos SQL interna en el ordenador. Nuestra aplicación web se conectará con esta mediante llamadas HTTP, descritas en el archivo '*views.py*'.

También a la misma jerarquía que '*app*' encontraremos un repositorio de MIDI Files usado para las pruebas ('*midi_files_repository*'), una carpeta donde almacenamos las MIDI Files en sus respectivas carpetas por cada usuario ('*users*') y el archivo principal desde donde se pone en marcha nuestra aplicación, '*run.py*'.

Una vez explicado en qué consiste esta capa y cuál es la estructura del proyecto, se explicarán las funcionalidades en las que interviene la capa lógica haciendo referencia a los casos de uso.

En el diagrama de casos de uso '*Login*' (Figura 8), el usuario por medio de la capa de presentación tiene que poder *loguearse* o crear un usuario. Una vez hecho esto, tiene que poder gestionar el estado de este usuario, ya bien cerrando sesión o borrándolo de la base de datos. Nuestra arquitectura no permite que la capa de presentación contacte directamente con la capa de datos, por lo que la capa lógica tiene que actuar como intermediario entre estas.

La capa de presentación se comunicará con la capa lógica mediante peticiones HTTP incluidas en los ficheros JavaScript que esta utiliza, proporcionándole los datos necesarios para cumplir con la petición. Así pues, una vez contactada la capa lógica (la petición HTTP contacta con un *endpoint* de esta capa), esta se comunicará con la capa de datos a través del protocolo HTTP, cumpliendo con las peticiones que la capa de presentación ha iniciado. Por lo tanto, desde la capa lógica responderemos la petición de la capa de presentación accediendo a la capa de datos y devolviendo la respuesta que esta nos proporcione.

En el diagrama de casos de uso 'entrenar IA o seleccionar *checkpoint*' (Figura 9), el usuario por medio del *frontend* debe de poder subir ficheros MIDI y borrarlos si no le hicieran falta. Así pues, la capa lógica se encargará de la gestión de la subida de los ficheros MIDI, de su organización y de su posterior borrado si se necesitase.

En la figura 10, relacionada con la gestión de MIDIs, tendremos que seleccionar qué pista queremos que actúe como melodía, así como organizar las carpetas por tipo de MIDI y por usuario. La capa lógica será la encargada de realizar todo este proceso interno, donde se utilizará el archivo '*midi_files_proc_music21.py*' anteriormente mencionado como un módulo en '*views.py*'. Este archivo importa una biblioteca denominada '*music21',* desarrollada por el MIT, la cual nos permite tratar los ficheros MIDI y lograr gestionarlos exitosamente. También, mediante librerías estándar, crearemos las carpetas necesarias para la organización de MIDIs teniendo en cuenta el usuario que está utilizando la aplicación.

Acto seguido, en la figura 11, relacionada con la generación de la melodía, tendremos que guardar dentro del sistema las restricciones seleccionadas por el usuario, así como permitir gestionar dinámicamente esta lista de restricciones. La capa lógica gestiona esto de manera interna, y en un posterior desarrollo, usará este listado de restricciones para que, una vez conectado nuestro servidor web con el clúster, pasarle la información y generar la melodía acorde a los límites impuestos por el usuario.

Aparte del control de todas estas funcionalidades internas, la capa lógica es responsable de proporcionar la navegabilidad de la página web a través de los *endpoints* desde los que se carga cada fichero HTML. Así pues, también se encargará de organizar esta navegabilidad.

4.2.3 Capa de datos

Finalmente, dentro de nuestra arquitectura se encuentra la capa de datos. Esta capa es responsable de proporcionar persistencia a los datos almacenados del sistema. A pesar de que nuestra plataforma no requiere de prácticamente casi almacenamiento de datos, es necesario hacer uso de una base de datos para gestionar los clientes que se den de alta.

La capa de datos se comunica solamente con la capa lógica, y es esta última la que envía peticiones de tipo CRUD (*Create*, *Read*, *Update* & *Delete*) sobre la base de datos. La capa de datos está formada por una API creada en el *framework* .NET, la cual gestiona las peticiones que se envían a esta API accediendo a una base de datos SQL interna. La API se desarrolla siguiendo una arquitectura MVC (*Model-View-Controller*), aislando cada una de las capas con las otras y facilitando el desarrollo de la RestAPI (Kaalel, 2023).

Cuando analizamos el problema en el Capítulo 3, proporcionamos un diagrama de clase donde representábamos la clase Usuario, que es la única clase de la que almacenaremos datos. Dentro de esta proporcionamos qué atributos eran necesarios para el correcto funcionamiento de la aplicación, pero no profundizamos en estos. Ahora, en la fase de diseño de la solución, los hemos especificado correctamente.

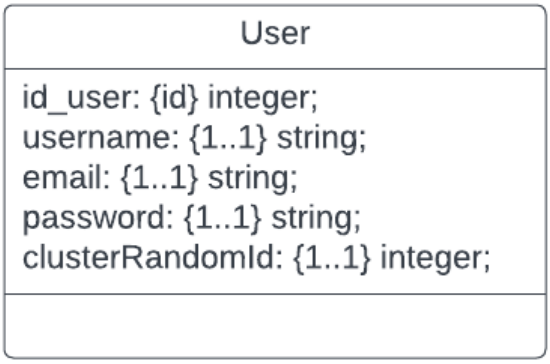

Figura 27. Diseño detallado de clase. Fuente: elaboración propia.

User: (id_user: integer(9), username: varchar(22), email: varchar(22), password: varchar(22), clusterRandomId: integer(9))

CP: {id_user}

VNN: {username}, Unicidad: {username}

VNN: {email}, Unicidad: {email}

VNN: {password}

VNN: {clusterRandomId}, Unicidad: {clusterRandomId}

R1: el atributo clusterRandomId tendrá un número comprendido entre 100000000 y 999999999.

5. Tecnologías utilizadas

En este capítulo introduciremos las tecnologías en las que nos hemos apoyado para desarrollar exitosamente el proyecto.

5.1 Herramientas generales para el desarrollo

Google Chrome 

Google Chrome es uno de los mejores entornos de ejecución que existen para el desarrollo *frontend*, ya que ofrece ciertas ventajas significativas que lo hacen destacar frente a otros entornos de ejecución, tales como su motor de renderizado o sus herramientas de desarrollo integrado.

El motor de renderizado Blink nos permite interpretar y representar el código HTML, aplicar estilos CSS y ejecutar JavaScript en el navegador. Su eficiencia y compatibilidad con los estándares actuales garantiza que la aplicación se despliegue correctamente en diferentes dispositivos y sistemas operativos.

Las herramientas de desarrollo integradas facilitan la depuración de errores y la prueba de la aplicación web. Destaca la consola de desarrollador, la cual nos permite detectar y solucionar errores relacionados con los ficheros JavaScript. También destaca el panel de elementos, para examinar y editar el código HTML y los CSS aplicados a los elementos de la página, así como la facilidad que este entorno nos presta para inspeccionar y modificar el DOM (*Document Object Model*) en tiempo real.

Visual Studio Code 

Visual Studio Code (VS Code) es uno de los editores de código más populares entre los programadores gracias a su facilidad para escribir, editar y depurar código en diversos lenguajes de programación.

Las características que contiene, la posibilidad de añadir extensiones y de personalizarlo al gusto de cada programador hacen de este editor de código uno de los mejores en el mercado.

Las características más destacadas son el resaltado de sintaxis y autocompletado inteligente (proporcionándonos sugerencias de código a tiempo real y permitiéndonos una correcta visualización), la depuración integrada (permitiéndonos depurar aplicaciones), el control de versiones (sincronizándose fácilmente con GitHub), la administración de extensiones, la terminal integrada (para ejecutar comandos), etc.

Este rendimiento y flexibilidad que ofrece VS Code hace de este el editor de código elegido para el desarrollo del proyecto.

GitHub 

GitHub es una plataforma online que permite alojar y compartir proyectos de *software* utilizando el sistema de control de versiones Git. Es una herramienta esencial para desarrolladores y equipos de programación, ya que facilita la colaboración, el seguimiento de cambios y la gestión de proyectos. A pesar de que en este proyecto solo ha sido desarrollado por 1 persona, se ha hecho uso de GitHub para el control de versiones de este y para tener el proyecto almacenado en la nube.

Las 3 características más importantes de GitHub son:

1. Repositorios: son espacios de almacenamiento en línea en los cuales se pueden almacenar y organizar proyectos de *software*. Los repositorios permiten realizar un seguimiento de los cambios realizados en el código.

2. Control de versiones: haciendo uso de Git se puede realizar un seguimiento de las diferentes versiones del código fuente. Esto permite a los desarrolladores trabajar en paralelo en diferentes ramas y fusionar los cambios de forma segura.

3. Colaboración: GitHub facilita la colaboración entre desarrolladores y equipos. Permite a múltiples personas trabajar en el mismo proyecto, realizar revisiones de código, comentar y discutir los cambios propuestos, y trabajar juntos de manera efectiva.

Existen muchas más características importantes por explicar, pero con estas ya podemos afirmar que GitHub es una plataforma esencial para el desarrollo de *software* colaborativo que ayuda a los desarrolladores a trabajar de manera eficiente y efectiva en proyectos de *software*.

5.2 Frontend

HTML 

HTML (*HyperText Markup Language)* es el lenguaje utilizado para crear y estructurar el contenido de las páginas web. Es la base de cualquier página web y define su estructura mediante etiquetas y elementos.

HTML utiliza una sintaxis basada en etiquetas que engloban el contenido y le dan significado. Cada etiqueta tiene un propósito específico y define diferentes elementos. Mediante sus etiquetas, HTML permite enlazar diferentes páginas web mediante enlaces, definir listas, crear formularios interactivos, insertar medios como audio y video, etc.

En resumen, HTML es el lenguaje de marcado fundamental en el desarrollo web utilizado para estructurar y definir el contenido de las páginas web, siendo este un lenguaje fundamental en el desarrollo web el cual es interpretado por los navegadores para mostrar el contenido de forma adecuada al usuario.

CSS 

CSS (*Cascading Style Sheets*) es un lenguaje de hojas de estilo que se utiliza para dar forma a un documento HTML. Su función principal es separar el contenido de un documento web de su presentación visual.

Con CSS, se pueden definir estilos los cuales se aplican a las etiquetas HTML para modificar su aspecto, dándoles un determinado color, tipografía, tamaño, etc. Esto permite personalizar la apariencia de una página web y lograr diseños visualmente atractivos y coherentes. Los estilos se pueden aplicar de forma global a una misma etiqueta a través de hojas de estilo externas enlazadas con el documento HTML o pueden aplicarse estilos de forma individual a las etiquetas HTML usando el atributo *'style'*.

Por lo tanto, CSS es un lenguaje de hojas de estilo que permite controlar la apariencia y el formato de un documento HTML de manera flexible y eficaz.

Bootstrap 

Bootstrap es un *framework* de desarrollo web que proporciona un conjunto de herramientas y estilos predefinidos que facilitan la creación de sitios web y aplicaciones *responsive*. Este *framework* se caracteriza por su sistema de rejilla, el cual permite organizar el contenido de una página en filas y columnas. Esto facilita la creación de diseños flexibles y adaptables a los distintos tamaños de pantalla.

Además de esto, Bootstrap ofrece una amplia gama de componentes y estilos CSS predefinidos, los cuales ayudan a diseñar una página web de manera rápida y dotada de profesionalidad.

JavaScript 

JavaScript es un lenguaje de programación que se utiliza en el desarrollo web para agregar interactividad y dinamismo a las páginas web. Este lenguaje se enfoca en la lógica que la página web debe de seguir, así como de definir su comportamiento.

JavaScript es el responsable de manipular nuestro contenido HTML de forma dinámica, responder a eventos o interacciones que tiene el usuario con nuestra página, realizar cálculos internos o comunicarnos con servidores (comunicación capa de presentación con capa lógica) para recibir o enviar datos.

De los diversos lenguajes de programación que se utilizan para el lado del cliente de una aplicación web este es el más utilizado, con una cuota de mercado del 98.7% (W3Techs, 2023). Su compatibilidad con la mayoría de los navegadores web modernos, su gran número de *frameworks* y bibliotecas y su amplia interoperabilidad con HTML y CSS hacen de este lenguaje el mejor para desarrollar la plataforma web.

5.3 Backend

Python & Flask  

Flask es un *framework* de Python utilizado para el *backend* de plataformas web. Una de las características principales que ofrece Flask es el enfoque minimalista que este tiene, proporcionando principalmente funcionalidades esenciales para construir aplicaciones web. Flask se basa en *endpoints* (URLs que nosotros definimos), donde los clientes envían peticiones HTTP. Cada *endpoint* está asociado a una función escrita en Python que gestiona la solicitud y devuelve la respuesta al cliente. Estas respuestas van desde un texto plano o un objeto JSON hasta la renderización de un fichero HTML o la gestión un archivo (por ejemplo, una MIDI File).

Otra de las ventajas que ofrece es la integración con plantillas HTML. Flask hace uso de un motor de plantillas desarrollado en Python llamado Jinja2, el cual puede generar contenido dinámico en las vistas y pasar datos desde Flask a HTML, así como hacer llamadas a las funciones de Flask desde el código HTML, generando así una bidireccionalidad entre el *frontend* y el *backend* de una manera más sencilla.

Se ha escogido este *framework* debido a que se utiliza Python para otras funcionalidades del *backend*. Python es un lenguaje de programación versátil y simple, con una inmensa comunidad y un gran número de librerías tanto estándar como desarrolladas por la comunidad. Otra de las razones por las que se escoge este lenguaje es por la importancia que tiene dentro del mundo del desarrollo, siendo el lenguaje más usado actualmente, según el índice TIOBE (TIOBE, 2023).

C#, .NET 5.0 & SQLServer  

.NET 5.0 es un *framework* de desarrollo de *software* multiplataforma que permite crear una amplia variedad de aplicaciones. Proporciona un entorno de ejecución y una biblioteca de clases unificada para desarrollar aplicaciones en C#. Se ha escogido este *framework* por su facilidad para desarrollar APIs haciendo uso de ASP.NET Core.

ASP.NET Core es la plataforma de desarrollo web de .NET. ASP.NET Core permite desarrollar APIs RESTful de alto rendimiento y escalables. El principal motivo por el que se ha escogido esta tecnología era por la arquitectura Modelo-Vista-Controlador con la que se desarrolla, así como la facilidad que ofrece el paquete Entity Framework para conectarnos con la base de datos SQL interna.

Para integrar la base de datos SQL con la API desarrollada se hace uso del paquete Entity Framework Core, el cual nos permite definir el modelo de datos a usar y mapearlo a las tablas de la base de datos, así como definir los accesos a la BD.

Para poder hacer uso de una BD SQL interna, se necesita el servidor de BBDD SQLServer. Para conectar la API con la BD se configura la cadena de conexión con la base de datos, permitiendo ya enviar peticiones a la API para acceder a la BD, donde el paquete EF se encargará de generar y ejecutar las consultas SQL.

6. Implementación de la solución

En la fase de análisis habíamos recopilado todos los Requisitos Funcionales y No Funcionales que nuestra aplicación requería mediante reuniones con el cliente y un análisis exhaustivo del problema.

Posteriormente se pasó a la fase de diseño, donde se analizaron los requisitos nuevamente y a partir de estos se definieron las tecnologías a utilizar, la arquitectura a seguir y se desarrollaron bocetos e interfaces para poder tener un prototipado de lo que sería la propuesta final.

En esta fase, la fase de implementación, es donde se concentran el mayor número de esfuerzos en el proyecto. Aquí se convierte todo lo diseñado previamente en una plataforma tangible, con funcionalidades reales y no conceptuales. Es aquí donde se programa el *frontend* de la aplicación, el *backend* que da soporte al servicio, así como la API a la que la plataforma accede para recuperar información sobre los usuarios.

Además de eso, hay que destacar que no todos los lenguajes o *frameworks* utilizados en la plataforma eran dominados por el programador. Durante el grado universitario, se habían visto lenguajes como JavaScript o C#, pero no se había profundizado en ellos. Como agravante, los *frameworks* utilizados eran totalmente desconocidos, así como los lenguajes de HTML y CSS. También hay que destacar que el programador era el único responsable del proyecto y nunca había construido una plataforma web, por lo que el desconocimiento de cómo se estructura, cómo funciona y cómo separar las capas llevó a multitud de fallos que consumieron gran parte del tiempo dedicado a la fase de implementación.

Así pues, se procederá a explicar con ejemplos de código el funcionamiento de Flask, el *framework* principal que gestiona la plataforma web. Posteriormente se introducirá particularidades de la solución y cuáles son los principales problemas que han surgido y qué se ha hecho para solucionarlos.

6.1 Funcionamiento de Flask

En el punto 4.2.2, donde presentábamos la capa lógica, ya se explicó la estructura de carpetas de Flask. Básicamente lo que estaba relacionado con Flask se incluía dentro de la carpeta raíz '*app*', salvo el archivo principal al que se llama cuando ponemos en marcha la aplicación, '*run.py*', encargado de definir la aplicación y dotarla de un puerto donde desplegarse. En la siguiente imagen (Figura 28) observamos el archivo y cuál es el comando necesario para poner en marcha la aplicación.

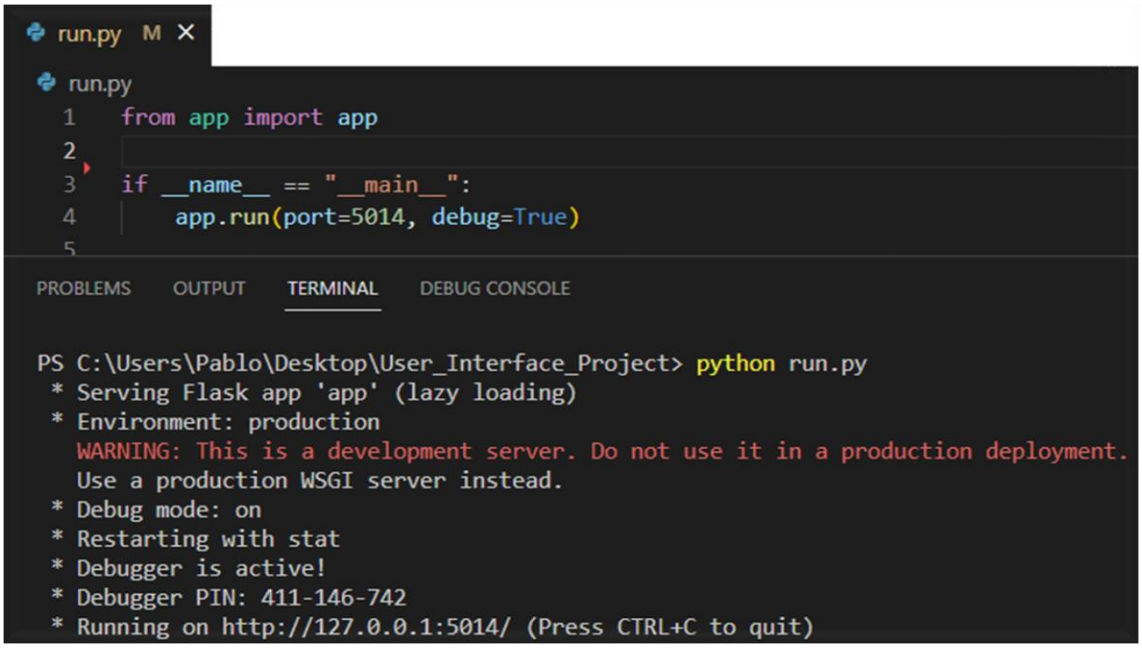

Figura 28. Fichero 'run.py' y comando para poner en marcha la aplicación. Fuente: elaboración propia.

Desde la carpeta raíz '*app*' se puede acceder a las carpetas '*__pycache__*', '*static*' (donde se encuentran los elementos estáticos del programa, como los archivos CSS, JavaScript o las imágenes) y '*templates*' (donde se encuentran los ficheros HTML que componen las vistas). También se puede acceder a los archivos escritos en Python de '*__init__.py*', '*midi_files_proc_music21.py*' y '*views.py*'.

Previamente ya se explicó en profundidad la estructura, por lo que no se volverá a detallar en esta parte del trabajo. No obstante, aquí nos centraremos en el archivo '*views.py*', desde el cual toda la funcionalidad de la aplicación toma forma.

Este archivo contiene las URLs (*endpoints* de la aplicación) y las vistas asociadas a estas URLs. Desde este archivo se gestionan las rutas de la aplicación y la lógica que hay detrás de cada una. Las vistas en Flask son funciones que se ejecutan cuando se hace una petición, ya bien sea a través de una petición HTTP a la URL que contiene la vista o mediante otras herramientas que proporciona este *framework*. Estas funciones se encargan de la lógica de negocio y generan las respuestas que se envían al cliente, cargan los ficheros HTML que corresponden con la navegación dotada al sistema o ejecutan internamente alguna función.

Cuando se pone en marcha la aplicación Flask, se llama a la ruta principal y se ejecuta la vista a la que esta esté asociada. Como podemos observar en la siguiente imagen (Figura 29), la ruta o URL principal está definida por *'@app.route('/', methods=['GET', 'POST'])'*. El primer campo representa la ruta ('/'), mientras que el segundo campo representa qué tipo de peticiones HTTP se le puede pedir a esta ruta. Todas las rutas tendrán una estructura similar, sin embargo, la vista o función asociada cambiará según lo que se necesite. Debajo de la ruta tendremos una línea de código definiendo la política CORS que nuestra aplicación sigue, pero esto se explicará más tarde.

Finalmente, podemos observar la vista asociada a nuestra URL, *'homepage_not_log()'*. Debajo de esta vista instanciaremos variables globales a sus valores originales y se cargará el fichero HTML vinculado a la primera vista de la página web (*'return render_template('homepage_not_logged.html')'*).

```python
@app.route('/', methods=['GET', 'POST'])
#We add the CORS package to solve the problems of CORS policy when we make Request to the Flask functions
@cross_origin(origin='*')
def homepage_not_log():
    global UserIsLogged
    global sessionUser
    global sessionEmail
    global sessionPassword
    global midiFilesList
    global training
    UserIsLogged = False
    sessionUser = ""
    sessionEmail = ""
    sessionPassword = ""
    midiFilesList = []
    training = True
    return render_template('homepage_not_logged.html')
```

Figura 29. Fichero 'views.py': ruta principal de Flask. Fuente: elaboración propia.

Una vez ejecutada completamente la vista, se nos abrirá el fichero HTML en el navegador. Se puede observar como la URL coincide con la dirección IP de nuestro ordenador (*localhost*), y va seguida del puerto en el que hemos desplegado la aplicación (si se observa la figura 28, esto coincide con lo que la terminal nos comenta). Vemos cómo se muestra la página asociada al fichero HTML que Flask cargaba en la vista y cuál es su diseño final. A la parte derecha de la Figura 29 se observa la estructura de ficheros y los CDNs que se han cargado junto a este fichero HTML. Los ficheros, dentro de la carpeta *'static'*, son el CSS y el JavaScript asociados a este fichero HTML que se proporcionan desde el servidor creado con Flask. Los CDNs son servidores externos los cuales proporcionan contenido al usuario sin tener este que ser proporcionado desde nuestro servidor. Así pues, ciertas librerías de JavaScript y CSS son proporcionadas al usuario mediante los CDNs. Todas las vistas de las URLs de Flask que simplemente carguen un fichero HTML en cierta URL seguirán una estructura parecida, simplemente variando un poco en la lógica que la vista tiene asociada.

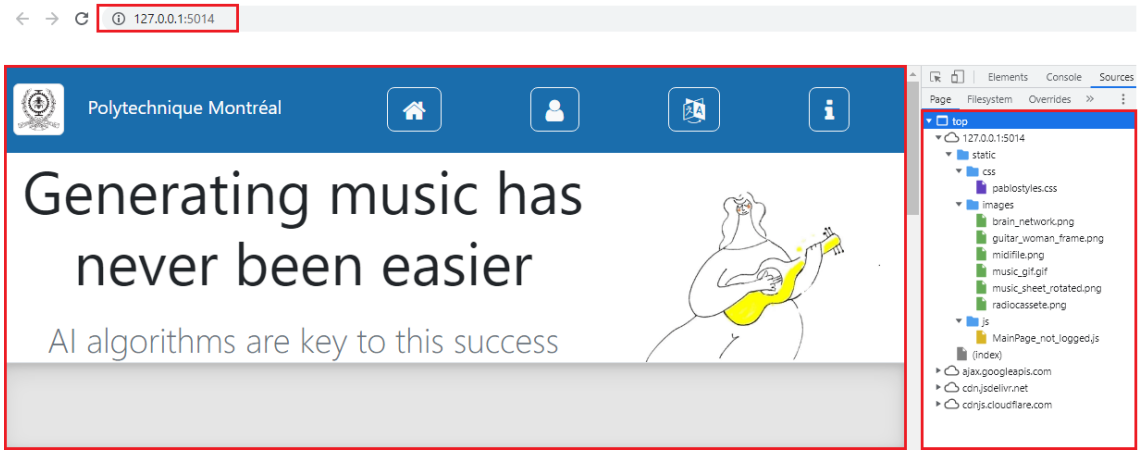

Figura 30. Página principal cargada desde Flask en el navegador. Fuente: elaboración propia.

Se ha mostrado ya como desde Flask cargamos una ruta en el navegador, ejecutamos la vista asociada y desplegamos el fichero vinculado a esta. Sin embargo, esto no siempre ocurre de la misma manera, no es necesario siempre realizar una petición HTTP a un *endpoint* para conseguir esta funcionalidad (cuando se inicia la aplicación se realiza una petición HTTP a la ruta principal *'/'*). Desde HTML se puede conseguir la misma funcionalidad. Esto es posible gracias al motor de plantillas Jinja2, que nos proporciona una bidireccionalidad entre los ficheros HTML y Flask.

En la Figura 31 se observa una captura de pantalla de un fichero HTML con 3 recuadros de colores. El recuadro rojo representa la función que incluimos dentro de una etiqueta HTML para que, una vez se haga *click* sobre este botón, se ejecute la vista asociada a *'cluster'* y se cambie la ruta a la URL con la que está vinculada. La función *'url_for()'* recibe como parámetro una vista, ejecuta esta vista, carga el fichero HTML asociado a esta y cambia la ruta de la página. La vista *'cluster'* la tenemos definida en el archivo *'views.py'*. Así pues, accedemos desde HTML a Flask sin necesidad de una petición HTTP.

Los recuadros amarillo y verde son simplemente demostraciones de cómo desde un fichero HTML hacemos *link* con ficheros CSS o JavaScript que tenemos en la estructura de carpetas de nuestro servidor (rutas tales como *'static/css/pablostyles.css'* o *'static/js/projectChoosing.js'*) o CDNs.

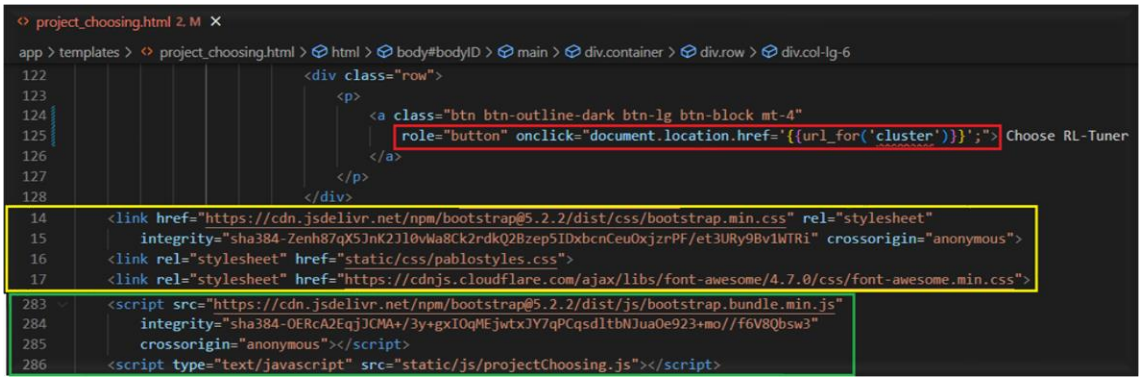

Figura 31. Fichero HTML desde el que se accede a una vista de Flask. Fuente: elaboración propia.

Tras ver cómo desde Flask podemos cargar ficheros HTML para desplegar las páginas de nuestra web y cómo desde estos ficheros HTML podemos acceder a Flask y seguir con la navegación de páginas, procederemos a enseñar cómo desde el JavaScript asociado a un fichero HTML podemos acceder a Flask a través de peticiones HTTP.

El siguiente ejemplo abarcará el Caso de Uso que representa el Requisito Funcional 11, subir MIDI. Por la parte del *frontend,* el cliente subirá una MIDI File a través de un botón representado por una etiqueta HTML. El JavaScript asociado a este fichero HTML estará atento a las entradas de MIDIs. Por cada fichero MIDI que se suba, se enviará este fichero al *backend* mediante una petición HTTP a un *endpoint* de Flask. La vista asociada a este *endpoint* procesará el fichero MIDI y lo almacenará en la carpeta de MIDIs del usuario. Esta carpeta contiene todos los ficheros MIDIs "brutos", es decir, todavía no han sido separados en MIDIs con melodía y MIDIs con melodía y acordes. Cabe destacar que esta funcionalidad solo está disponible si el usuario ha iniciado sesión. Posteriormente, en el apartado 6.2 veremos este Caso de Uso, para explicar en profundidad como funciona la arquitectura de 3 capas que hemos implementado.

Así pues, desde la plataforma web, una vez iniciado sesión el usuario, tenemos la opción de subir los ficheros MIDI para alimentar a la IA que queremos entrenar. Cuando presionamos la opción de subir nuestras propias MIDI, se nos da la opción de subir los ficheros. Se observa que el botón de Continuar, si no percibe ninguna MIDI subida todavía, se deshabilita. Esto es lógico, ya que no podemos permitir entrenar la IA si no tenemos nada con lo que entrenarla.

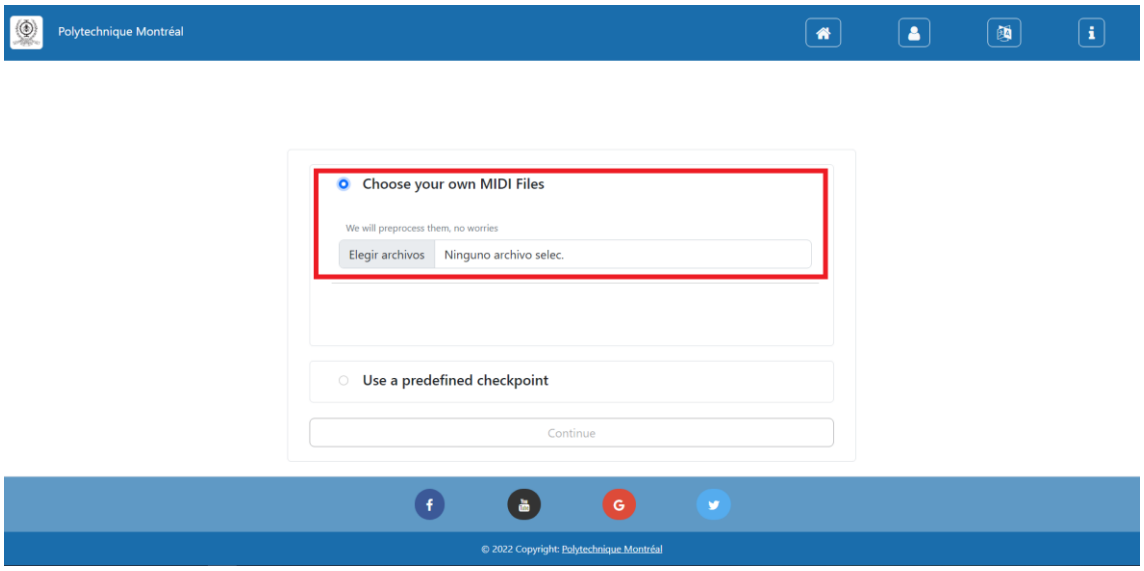

Figura 32. Página web desde la que subimos las MIDI para entrenar la IA. Fuente: elaboración propia.

El JavaScript de la Figura 33 primero guardará en una variable el elemento HTML vinculado a la subida de ficheros de la Figura 32. A esta variable se le añadirá un *listener*, el cual se ejecutará cuando suceda un evento relacionado con la subida de un fichero (*input*). Un *listener*, como su nombre indica, es una función que se asocia a una variable y que está pendiente de cualquier evento que suceda sobre esta. Cuando el evento que suceda sobre la variable sea el recogido por el *listener*, efectuará el código que se encuentre dentro de la función.

Cuando hay una subida de fichero o ficheros, se recorrerán todos estos. Se lanzará una alerta al usuario indicándole qué fichero ha subido y posteriormente se llamará a una función denominada '*sendMIDIFile*'. Esta función creará una petición HTTP al *endpoint* de Flask que gestiona los ficheros MIDI. Como se puede observar, se realiza un POST, ya que estamos pasando el fichero MIDI desde el *frontend* al *backend* para que así posteriormente sea gestionado.

Dentro de esta función podemos observar que se comprueba cuando se envía el último fichero MIDI. Cuando este ya se ha enviado, se comprobará cuantos ficheros MIDI se han subido durante toda la sesión del usuario, para así saber cuántos se necesita procesar y para habilitar o no el botón de Continuar de la Figura 32.

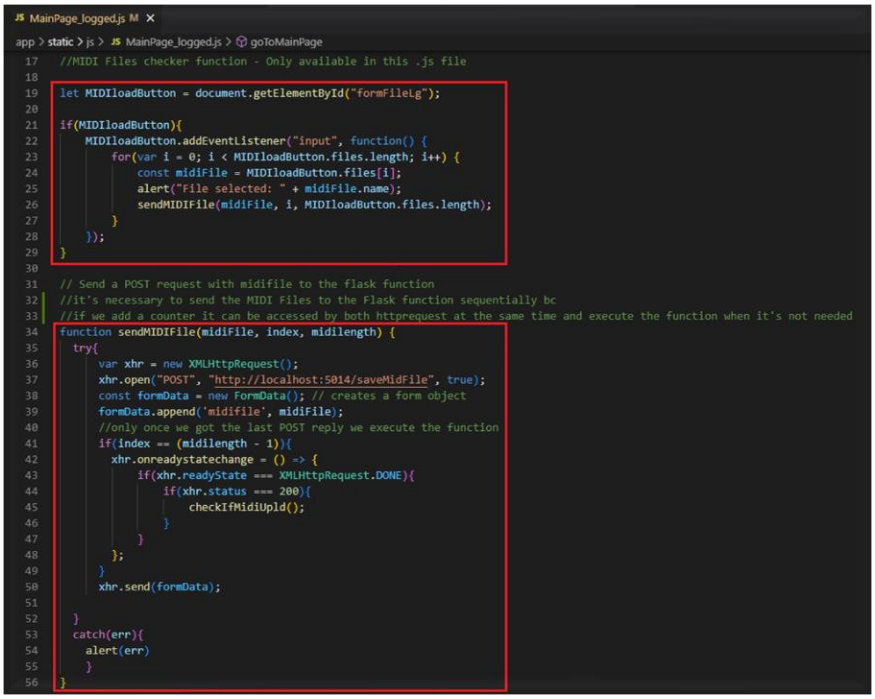

Figura 33. Fichero JavaScript que envía la MIDI File desde el frontend hasta el backend. Fuente: elaboración propia.

Desde Flask, se recibirá esta petición HTTP y se ejecutará la vista asociada a la URL que esta petición HTTP incluía. Aquí podemos observar cómo se guarda el fichero MIDI recibido en la carpeta del usuario. Como simplemente es un *endpoint* que actúa como pura lógica interna de la aplicación, no devuelve ningún HTML a cargar por la página.

Figura 34. Fichero 'views.py': ruta '/saveMidFile' que almacena la MIDI. Fuente: elaboración propia.

Finalmente, en la Figura 35 podemos ver como estos ficheros subidos aparecen en el *frontend*, una vez han sido procesados internamente por Flask y guardados en la carpeta de usuario. La aparición dinámica de las MIDIs en el *frontend* y la posibilidad de borrarlas de la página y de la carpeta de usuario se realiza mediante JavaScript. Dinámicamente crea los objetos y les asigna un hijo, que es el botón de borrar. Este botón de borrar tiene un *listener* que actúa cuando se hace *click* sobre botón. Removerá el objeto del *frontend* y enviará una petición HTTP a Flask para que desde el sistema se remueva el fichero de la carpeta del usuario.

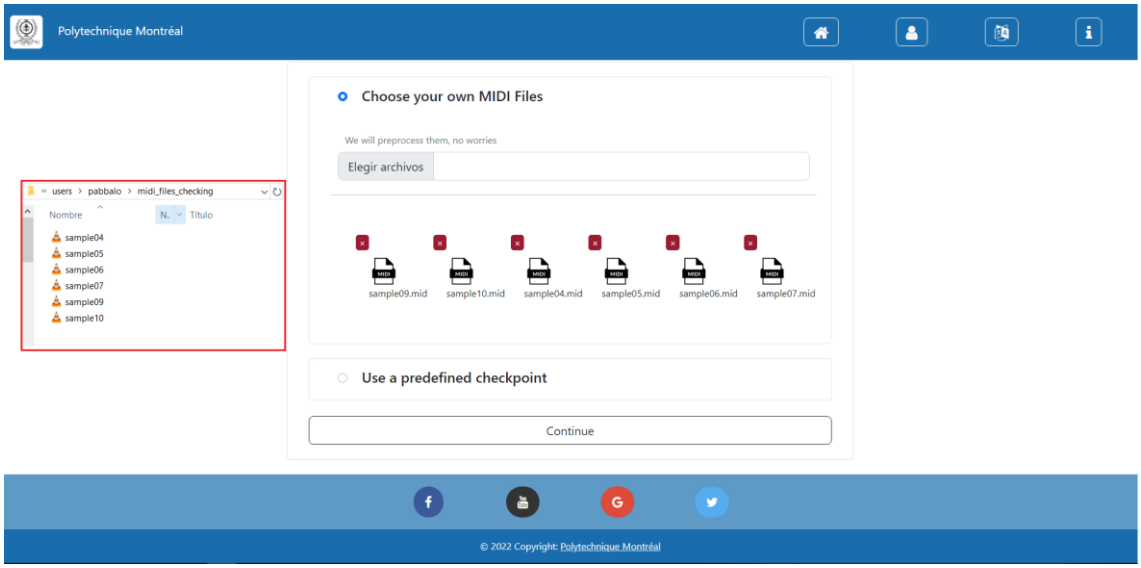

Figura 35. Página web con las MIDI subidas para entrenar la IA. Fuente: elaboración propia.

Para finalizar este apartado es conveniente mostrar cómo el archivo de *'views.py'* hace uso del fichero *'midi_files_proc_music21.py'*, el cual hace uso de la biblioteca desarrollada por el MIT *'music21'*. Esta librería nos dota de funciones que nos permiten acceder a los elementos individuales de una MIDI como las notas, los acordes... también podemos transformar y manipular estos ficheros MIDI, así como generar un nuevo fichero a partir del tratado (Scott Asato Cuthbert, 2023). Debido a la extensión de este archivo y de su importancia, se incluirá en el Anexo 1.

Haciendo referencia a la Figura 35, el botón de Continuar está ahora habilitado. Esto sólo puede ocurrir si se selecciona entrenar IA (y se han subido MIDIs) o si se selecciona el *checkpoint*. Dependiendo de lo seleccionado, el *'url_for()'* de la Figura 36 tendrá un valor u otro. Aquí se muestra el valor que tiene si se entrena la IA.

```
onclick="document.location.href='{{url_for('fileProcessNew', firstprocess=0)}}';"> Continue </button>
```

Figura 36. Elemento HTML del botón Continue si se entrena la IA. Fuente: elaboración propia.

Así pues, esto nos llevará a ejecutar la vista *'fileProcessNew'* en Flask que se precia en la Figura 37. Lo primero a destacar es el módulo *'mfpm'*. Este módulo representa el archivo *'midi_files_proc_music21.py',* y se hace uso de este para acceder a sus funciones. Algunas de sus funciones son *'path_establish(user)'*, la cual nos devuelve una serie de rutas necesarias para crear las carpetas donde guardaremos las MIDIs con melodía y las MIDIs con melodía y acordes, *'delete_folder_if_exists(path)',* la cual nos borra la carpeta de la ruta y *'MIDIConverter(...)',* la cual almacena toda la lógica detrás del tratado de las MIDIs, separando sus pistas, haciendo que nos quedemos con una de las distintas pistas que tiene la MIDI para generar una MIDI con solo melodía, agrupar todas las pistas para crear acordes mediante el método *'chordify()',* ... (se recomienda acceder al Anexo 1 para comprender el funcionamiento).

Básicamente esta vista de Flask recorrerá todas las MIDIs almacenadas en la carpeta del usuario y las procesará haciendo uso del módulo anteriormente mencionado. Si descubre que la MIDI File que se está tratando solo posee una pista, la guardará en la carpeta de MIDIs con sólo melodía.

Si descubre que la MIDI File que se está tratando posee 2 o más pistas, primero nos hará seleccionar cuál queremos que actúe como melodía. De aquí ya generará otra MIDI File con sólo 1 melodía y la guardará en la carpeta de MIDIs con sólo melodía. Después, hará uso de todas las pistas de la melodía para generar la base armónica. Tras esto, creará una nueva MIDI File con 2 pistas, la primera será la melodía que nos ha pedido seleccionar, la segunda pista será la base armónica. Tras esto, almacenará la MIDI File en la carpeta de MIDIs con melodía y acordes. Todo esto se realiza mediante una interfaz gráfica que permite al usuario seleccionar la pista a actuar como melodía. Cuando ya no hayan más MIDIs para tratar, se pasará a la siguiente ventana.

```python
@app.route('/<int:firstprocess>', methods=['GET', 'POST'])
def fileProcessNew(firstprocess):
    global midiFilesList
    global training
    #variable training to True, we will train the algorithm with the MIDI Files we have in the user's folder
    training = True
    #if we train the algorythm
    # we establish our paths
    print('files processing')
    print(type(firstprocess),firstprocess)
    cwd, directory, path_om, path_cm, path_MIDI_garbage = mfpm.path_establish(sessionUser)

    if (firstprocess == 0):
        # we check if the files where we have to store the MIDI files have been created previously, in order to delete them and its content
        mfpm.delete_folder_if_exists(path_om)
        mfpm.delete_folder_if_exists(path_cm)
        mfpm.delete_folder_if_exists(path_MIDI_garbage)

    for f in os.scandir(directory):
        if (f.is_file() and f.name.endswith('.mid')):
            midifileMusicScore, partsMIDIMusicScore = mfpm.MIDIConverter(str(f.path), path_cm, path_om, path_MIDI_garbage)

            # MIDIFile with only Melody
            if partsMIDIMusicScore == 1:
                mfpm.create_folder_if_not_exists(path_om)
                shutil.move(str(f.path), path_om)
            else:
                display_text = "Interval: [0, " + str(partsMIDIMusicScore - 1) + "]"
                melody_name = str(f.path).split("\\")[1]
                return render_template('PartsChoosing.html', PartsRange=display_text, CurrentMelodyNumberParts=partsMIDIMusicScore, CurrentMelodyName=melody_name)

    #we delete the MIDI FIles we could not move (the raw ones with M&C)
    mfpm.delete_folder_if_exists(path_MIDI_garbage)
    return redirect(url_for('MIDIUploadSelected', inputUsername=sessionUser))
```

Figura 37. Fichero 'views.py': vista 'fileProcessNew' que trata las MIDIs. Fuente: elaboración propia.

6.2 Particularidades de la solución

Acceso a datos desde una arquitectura de 3 capas

Tal y como se mencionó anteriormente en el capítulo 4.1, donde explicábamos la arquitectura del sistema, se sigue un modelo a 3 capas. Recapitulando, las 3 capas están compuestas por la capa de presentación (lo que el usuario ve visualmente), capa lógica o de negocios (la funcionalidad interna del sistema) y la capa de datos (el lugar donde se persiste la información del sistema).

Para que la plataforma web respetase esta arquitectura, desde la capa de presentación no podíamos permitir que hubiese libre acceso a la capa de datos. Para solucionar esto, se tenía que hacer uso del intermediario que actúa entre las dos, la capa lógica.

Por consiguiente, demostraremos esta particularidad de la solución mediante el Caso de Uso que representa el Requisito Funcional 05, *Loguear* usuario.

El usuario, una vez la página principal se ha renderizado en su navegador, para proseguir con la funcionalidad del sistema tiene la opción de iniciar sesión o de crear una cuenta si este todavía no la tuviera. En este Caso de Uso representamos qué tiene que ocurrir si el usuario inicia sesión. Al presionar uno de los diversos botones de inicio de sesión que se despliegan por la página, aparecerá el formulario representado por la Figura 38 donde se le pide al usuario las credenciales de inicio de sesión.

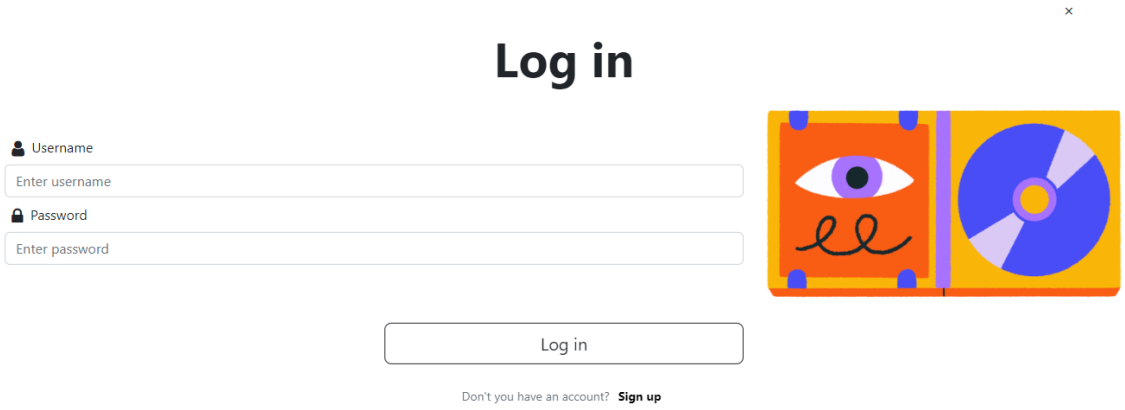

Figura 38. Formulario de Inicio de Sesión. Fuente: elaboración propia.

Lo visualizado en la Figura 38 está dentro de la capa de presentación. Es el *frontend* de la plataforma web, el cual interactúa con el usuario y recibe las entradas que este proporciona. Este fichero HTML que se ha desplegado en el navegador del usuario hace uso del siguiente fichero JavaScript (Figura 39). Este fichero antes de hacer la petición HTTP a la capa lógica, comprueba que los campos tengan valor mediante una función definida en el fichero JavaScript denominada *'inputEmpty()'*.

Si estos campos no presentan valor, se alerta al usuario. Estas comprobaciones (como tantas otras) se realizan para evitar errores inesperados y hacer que el usuario identifique rápidamente el error, mejorando la facilidad de uso de la plataforma.

Una vez se confirma que los campos tienen valor, se realiza una petición HTTP de tipo POST a un *endpoint* de Flask enviándole las credenciales del usuario, las cuales estarán dentro de un objeto JSON. Básicamente, desde la capa de presentación se contacta con la capa lógica para pasarle la información de acceso a la página web. Tras la petición, la capa de presentación se queda esperando a recibir una respuesta por parte de la capa lógica.

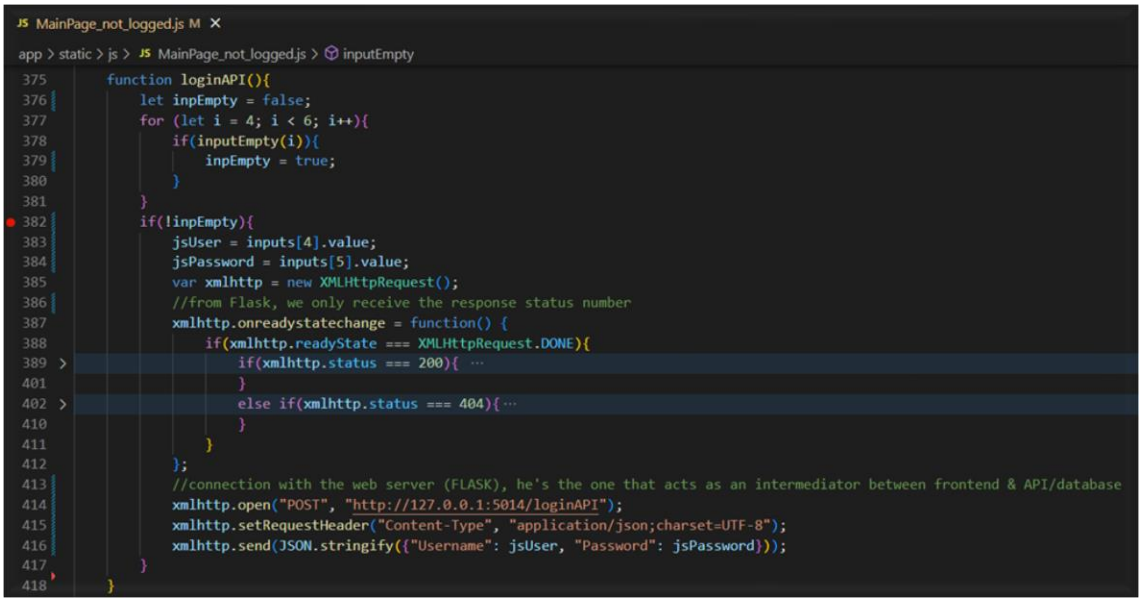

Figura 39. Petición HTTP desde la capa de presentación hasta la capa lógica. Fuente: elaboración propia.

En este momento, ya se tiene la petición realizada a la capa lógica. Ahora desde esta se tiene que tratar este objeto JSON que se ha enviado a través de la petición HTTP, recuperar de este las credenciales y contactar con la capa de datos mediante una petición HTTP para ver si este usuario existe o no. Una vez se envíe la petición a la capa de datos, la capa lógica quedará esperando a una respuesta por parte de la capa de datos donde se confirme si existe el usuario o no.

En la Figura 40 se pude apreciar la lógica antes descrita. La función *'requests.get()'* es la encargada de la petición de tipo GET a la API que representa la capa de datos.

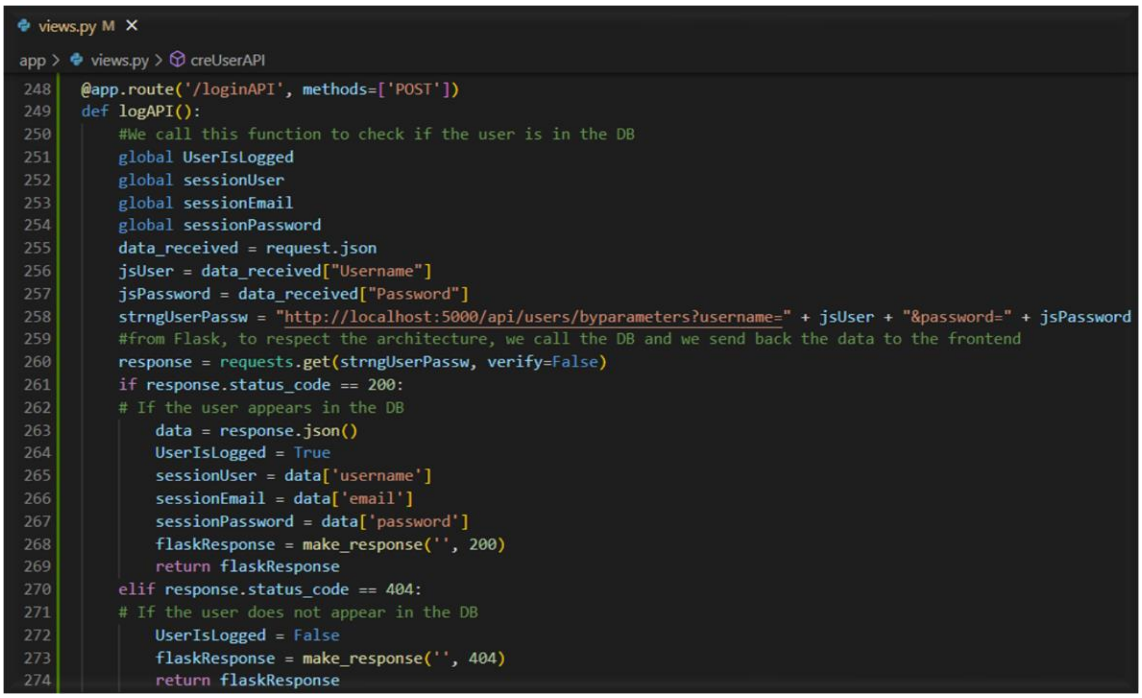

Figura 40. Petición HTTP desde la capa lógica hasta la capa de datos. Fuente: elaboración propia.

La Figura 41 representa la función de la API que recibe la petición desde la capa lógica. Esta función recibe las credenciales y hace la consulta sobre la base de datos relacional SQL interna. Si el usuario existe, devuelve un estado 200 (indicando que ha sido exitosa la petición) junto con un objeto JSON con toda la información del usuario que hay en la base de datos. Si no existe, devuelve un estado 404 (el usuario no existe en la base de datos).

Figura 41. Recepción de la petición HTTP en la capa de datos y respuesta. Fuente: elaboración propia.

Una vez tengamos la respuesta desde la capa de datos, tendremos que llevar esta hasta la capa de presentación pasando por la capa lógica. La respuesta de la capa de datos puede devolver 2 estados, el estado 200 (existe usuario) o el estado 404 (no existe usuario).

Esta respuesta llegará a la capa lógica. En la Figura 40, se puede observar cómo se gestiona el estado de la respuesta. Si la respuesta ha sido exitosa (estado 200), las variables relacionadas con la información del usuario toman el valor de la respuesta de la capa de datos (se puede observar cómo desde el JSON enviado desde la API se toman los valores relacionados con el usuario) y se responde a la capa de presentación con un estado 200.

No obstante, si la respuesta no ha sido exitosa (estado 404), se actualiza la variable *'UserIsLogged'* a falso y se propaga este estado 404 a la capa de presentación.

Finalmente, en la Figura 39, podemos observar que se está esperando la respuesta de la capa lógica en la línea 388. Si la respuesta tiene un estado 200, se cargará la página web correspondiente al usuario *logueado* con su información. Si la respuesta tiene un estado 404, se alertará al usuario de que las credenciales introducidas no corresponden a ningún usuario que actualmente se encuentre en la base de datos. En la Figura 39 no se ve desplegado el código que acabamos de explicar, pero básicamente esto se ha realizado para simplificar las imágenes y centrarnos en la comunicación entre capas.

REST API en .NET

Una particularidad de la solución es la implementación de una REST API desarrollada en el *framework* .NET Core. Durante la fase de diseño, se acordó desarrollar una API en este marco de trabajo debido al patrón de diseño que sigue (Modelo-Vista-Controlador) y a su posible escalabilidad.

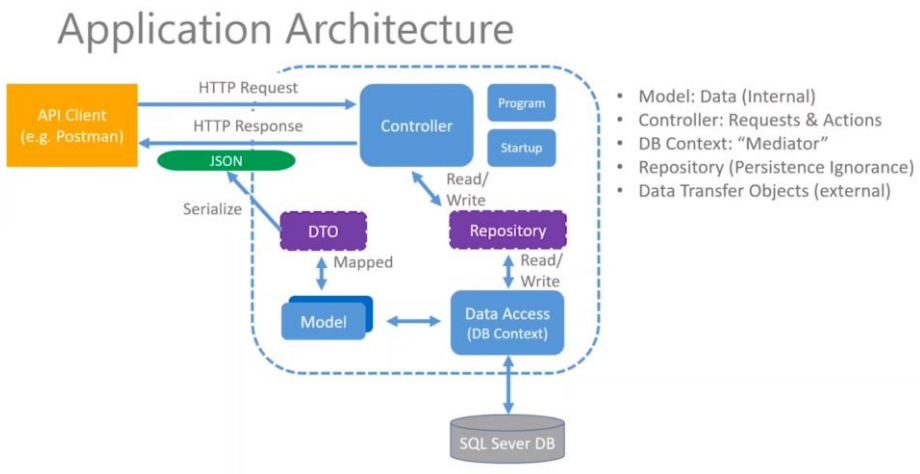

Figura 42. Arquitectura MVC de la REST API de la capa de datos. Fuente: (Jackson, 2020).

En la figura 42 podemos observar la arquitectura de esta API. En rasgos generales, los objetos de datos que queremos tratar (es decir, en nuestro caso el objeto Usuario) se definen como el modelo de datos de la API. Este modelo de datos estará representado en la base de datos SQL interna que tenemos mediante una tabla, así como en la API.

Para poder acceder a la base de datos, los clientes externos realizarán peticiones HTTP al 'Controlador'. Esta clase se encarga de la gestión de las peticiones, y transfiere la información al 'Repositorio'. La clase 'Repositorio' define las consultas sobre la Base de Datos mediante métodos en un nivel alto (C#). Para efectuar estos métodos hace uso de la clase 'Contexto de Base de Datos'. Sobre esta clase se realizarán en un nivel alto las consultas que se recogían en las peticiones HTTP, definidas en un nivel alto por los métodos de la clase 'Repositorio'. La particularidad de esta clase es que traduce estas consultas realizadas en un nivel alto al lenguaje de la Base de Datos que estemos utilizando, accediendo directamente a esta. Una vez la Base de Datos nos proporcione una respuesta en su lenguaje (SQL en nuestro caso), la clase 'Contexto de Base de Datos' recogerá esta respuesta y se la enviará a la clase 'Repositorio'. A su vez, la clase 'Repositorio' se la enviará a la clase 'Controlador', devolviendo esta finalmente la respuesta al cliente externo. La figura 43 muestra la estructura de carpetas de la API.

Así sería la arquitectura que presenta nuestra API, pero este no era el único motivo por el que se había desarrollado junto a este marco de trabajo. La escalabilidad era un punto muy importante valorado por el cliente. Gracias a la arquitectura, si se realizan migraciones de la base de datos de una relacional (SQL en nuestro caso) a una no relacional (MongoDB), esto no supondría mucho cambio en la API. Como se ha comentado, la clase 'Controlador' hace uso de la clase 'Repositorio'. En esta clase 'Repositorio' definimos en un nivel alto las consultas sobre la base de datos. La clase 'Repositorio' está formada por una interfaz (se le denomina el contrato de métodos, que define de forma abstracta qué métodos/consultas se pueden hacer sobre la base de datos) y su implementación (en el caso de la API, es una implementación teniendo en cuenta una base de datos SQL). Si migrásemos la Base de Datos, en vez de cambiar toda la API, solamente habría que cambiar esta implementación y adaptarla a la nueva base de datos, así como cambiar ciertas líneas de código de los archivos principales, tales como la conexión a la Base de Datos interna. Todo lo demás funcionaría perfectamente, ya que se abstrae gracias al patrón de diseño que se sigue.

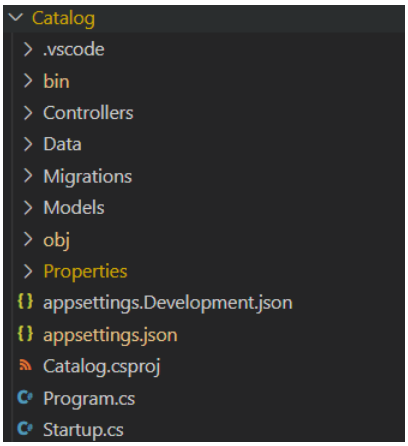

Figura 43. Estructura de carpetas de la API. Fuente: elaboración propia.

Partitura dinámica

Una de las claves de esta plataforma web y de estos algoritmos de IA es que se permite seleccionar qué restricciones aplicar a la melodía a generar. Por ello, era necesario crear una partitura dinámica que permitiese al usuario seleccionar el número de compases que quiere que su melodía tenga (8, 16 o 32 compases). Sobre esta partitura, se requería que el usuario seleccionara los compases y escogiese qué restricción o restricciones aplicar. Dependiendo de la restricción, el usuario debía de poder seleccionar solamente compases consecutivos o seleccionar dos conjuntos de compases. Todo esto debía de ser intuitivo, fácil y sencillo para el usuario. Así pues, mediante HTML, CSS y JavaScript se logró generar esta partitura dinámica que tan necesaria era.

En la Figura 44, podemos observar la partitura dinámica de 16 compases con las restricciones sobre compases consecutivos que ha aplicado el usuario.

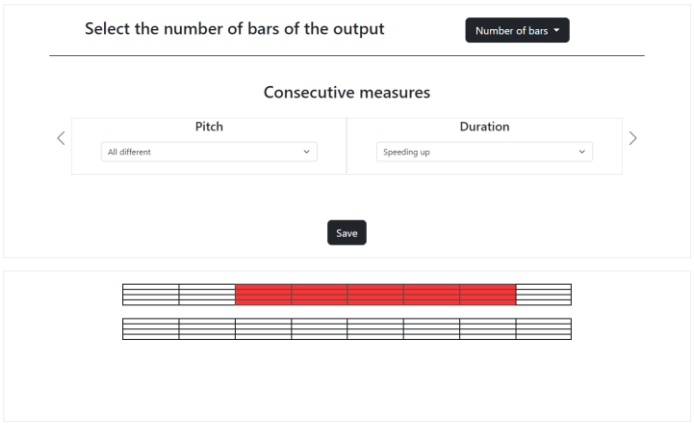

Figura 44. Partitura dinámica – compases consecutivos. Fuente: elaboración propia.

En la Figura 45, podemos observar la partitura dinámica de 32 compases con las restricciones sobre dos conjuntos de compases consecutivos que ha aplicado el usuario.

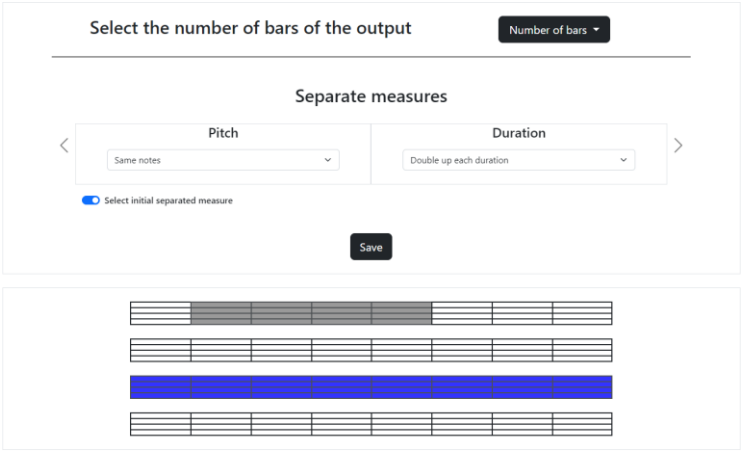

Figura 45. Partitura dinámica – conjuntos de compases consecutivos. Fuente: elaboración propia.

6.3 Problemas encontrados y decisiones tomadas

Durante el desarrollo de la solución se han encontrado problemas de todo tipo, desde problemas de planificación hasta problemas de código.

Los principales problemas de planificación residieron en estimar incorrectamente la curva de aprendizaje. Había que aprender sobre el desarrollo web, sobre nuevos lenguajes y dominar sus *frameworks*. Sumado a eso, los lenguajes que se conocían de antemano no se conocían en profundidad. Por lo tanto, el aprender de o una gran parte de cosas llevó al programador a dedicar un mayor número de horas de las estimadas.

Respecto a los problemas de código, había diseños que tenían que ser implementados de los que no había información o ejemplos previos. Por ejemplo, la creación de una partitura dinámica mediante CSS, HTML y JavaScript no había sido documentada en ningún sitio, por lo que su implementación llevó a cometer un gran número de errores. A parte de estos problemas que están puramente relacionados con el código, hubo otros problemas relacionados con el control de versiones de los paquetes y bibliotecas o con aspectos relacionados con el desarrollo web.

El problema que más tiempo tardó en ser resuelto estaba relacionado con la interacción entre capas debido a un mecanismo de seguridad, *Cross-Origin Resource Sharing Policy* (*CORS Policy*). Este mecanismo controla las solicitudes de recursos entre diferentes dominios. Cuando desde un origen distinto al servidor se solicita un recurso, esta política se activa para determinar si el recurso debe ser entregado o no al origen que no pertenece al dominio del servidor. La aplicación web (Flask) se ejecuta en un determinado dominio, mientras que la API se ejecuta en otro distinto. Cuando desde la capa lógica intentamos acceder a la capa de datos a través de peticiones HTTP a la API, estamos intentando acceder a un recurso desde otro dominio distinto al de la API.

Esta petición era denegada por la política CORS, ya que esta no había sido configurada desde la API. Por lo tanto, para poder acceder a la Base de Datos correctamente se tuvo que configurar la política CORS en la API. La clase *'Startup.cs'* es el archivo principal para poner en marcha la API que tiene acceso a la Base de Datos SQL interna. Por lo tanto, en la configuración de esta teníamos que definir cuál era nuestra política CORS. Como solamente éramos nosotros quienes podía acceder a ella, permitimos peticiones de recursos desde cualquier origen, con cualquier método y con cualquier encabezado.

Figura 46. Configuración de la política CORS. Fuente: elaboración propia.

7. Pruebas

En la fase de Validación descrita en la metodología que se ha utilizado se mencionaba que para validar el producto desarrollado se creaban casos de prueba manuales que comprobasen su correcto funcionamiento. Así pues, en este capítulo se expondrán estos casos de prueba manuales y se explicarán en profundidad.

Pruebas de funcionalidad

Las pruebas de funcionalidad se centran en verificar que la plataforma web desarrollada realiza todas las funciones y operaciones que se han detallado en los Requisitos Funcionales. Para lograr esto, se repasaron todos los Requisitos Funcionales que se habían recopilado junto al cliente y se verificó que el sistema acometiese este cumplido.

Uno de los Requisitos Funcionales que se revisó varias veces fue el RF11, 'Subir MIDI'. Para ello, se comprobó que desde el *frontend* se aceptase solo archivos de tipo MIDI y que en el momento que fuesen subidos se enviaran al *backend.* Desde el *backend,* se comprobó que estos ficheros fuesen tratados y almacenados en la carpeta asociada al usuario que los había subido.

Pruebas de facilidad de uso

Estas pruebas analizan la facilidad de uso de la plataforma web, así como la experiencia que tiene el usuario al hacer uso de esta. Aquí se verificó que la interfaz fuese intuitiva, que el usuario no necesitase de ayuda externa para guiarse, que los errores cometidos por el usuario fueran fáciles de reconocer y que las acciones fuesen fáciles de realizar.

De esta manera, se pidió ayuda a miembros del laboratorio para probar la aplicación. Se les pidió ciertas acciones a realizar y se observó si las podían llevar a cabo en un tiempo razonable y por sí solos sin necesidad de ayuda.

Pruebas de visualización

En este apartado se evalúa si la plataforma web se visualiza correctamente en distintas resoluciones de pantalla y tamaños de ventana. Básicamente, se verifica si la plataforma web es *responsive.*

Para ello, se ha desplegado la plataforma web en diversas resoluciones de pantalla y se ha comprobado que mantenga el diseño y consistencia visual, así como que siga siendo fácil de usar. Aparte de probarla en distintos tamaños de ventana, también se han utilizado distintos navegadores web.

Pruebas de integración

Las pruebas de integración consisten en comprobar que las distintas partes que forman la arquitectura de la plataforma web funcionan correctamente juntas. Por lo tanto, consisten en comprobar que las distintas capas en las que se separa la plataforma web pueden comunicarse entre sí.

Estas pruebas de integración fueron realizadas progresivamente. Primero se comprobó las conexiones entre la capa de presentación (*frontend*) y la capa lógica (*backend*). Después, entre la capa lógica y la capa de datos (API con base de datos SQL interna). Para finalizar, se comprobó la interoperabilidad entre todas las capas.

Pruebas de rendimiento

Se evalúa el rendimiento de la plataforma web, revisando el tiempo de carga de las páginas, la velocidad de respuesta y la eficiencia en el manejo de recursos.

Se ha comprobado cuánto tiempo se tarda en recibir respuestas de la API desde el *frontend,* cuánto tiempo se tarda en recuperar los archivos desde el servidor propio y cuanta ventaja nos da el utilizar CDNs para cargar ciertos archivos en vez del servidor propio.

Pruebas de aceptación

En las pruebas de aceptación se mostró al cliente el producto final para que evaluase el software y lo validase desde su perspectiva, siendo estas las últimas pruebas a realizar. Así pues, el cliente fue el encargado de revisar si se cumplían todos los Requisitos Funcionales que fueron discutidos, si la facilidad de uso del sistema era la suficiente y si se adecuó la plataforma correctamente a las necesidades del usuario.

El cliente, debido a que recibía actualizaciones cada poco de cómo iba mejorando el *software,* sabía como iba a ser el producto final. Por lo tanto, como estuvo en recurrente contacto con el desarrollador, la validación de las pruebas de aceptación fue inmediata.

8. Producto final

En este capítulo mostraremos la navegabilidad del producto final de forma estática mediante capturas de pantalla. Uno de los aspectos más característicos de esta plataforma web es su dinamismo, pero no ha sido posible plasmarlo en la memoria.

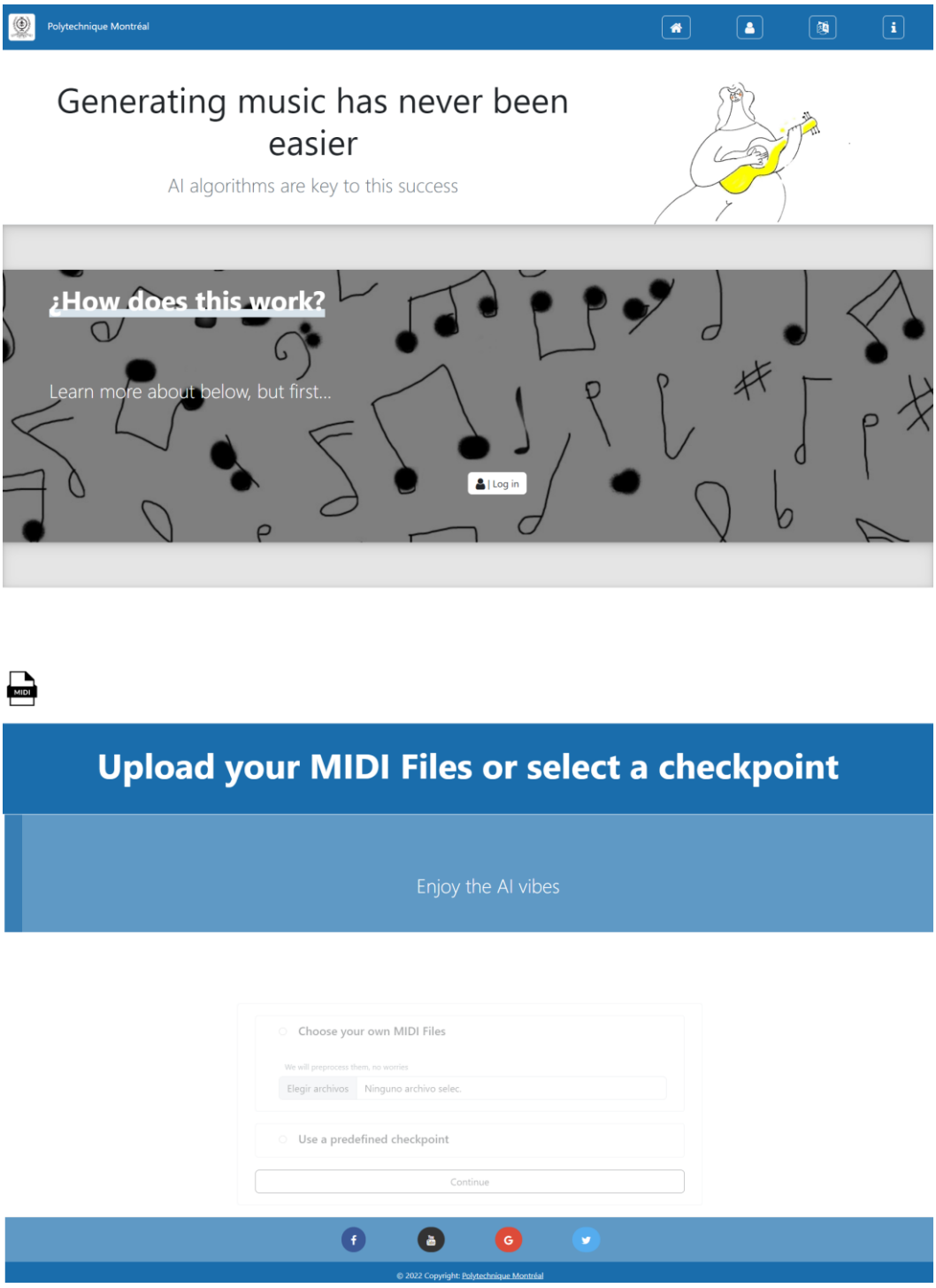

Figura 47. Página principal sin inicio de sesión. Fuente: elaboración propia.

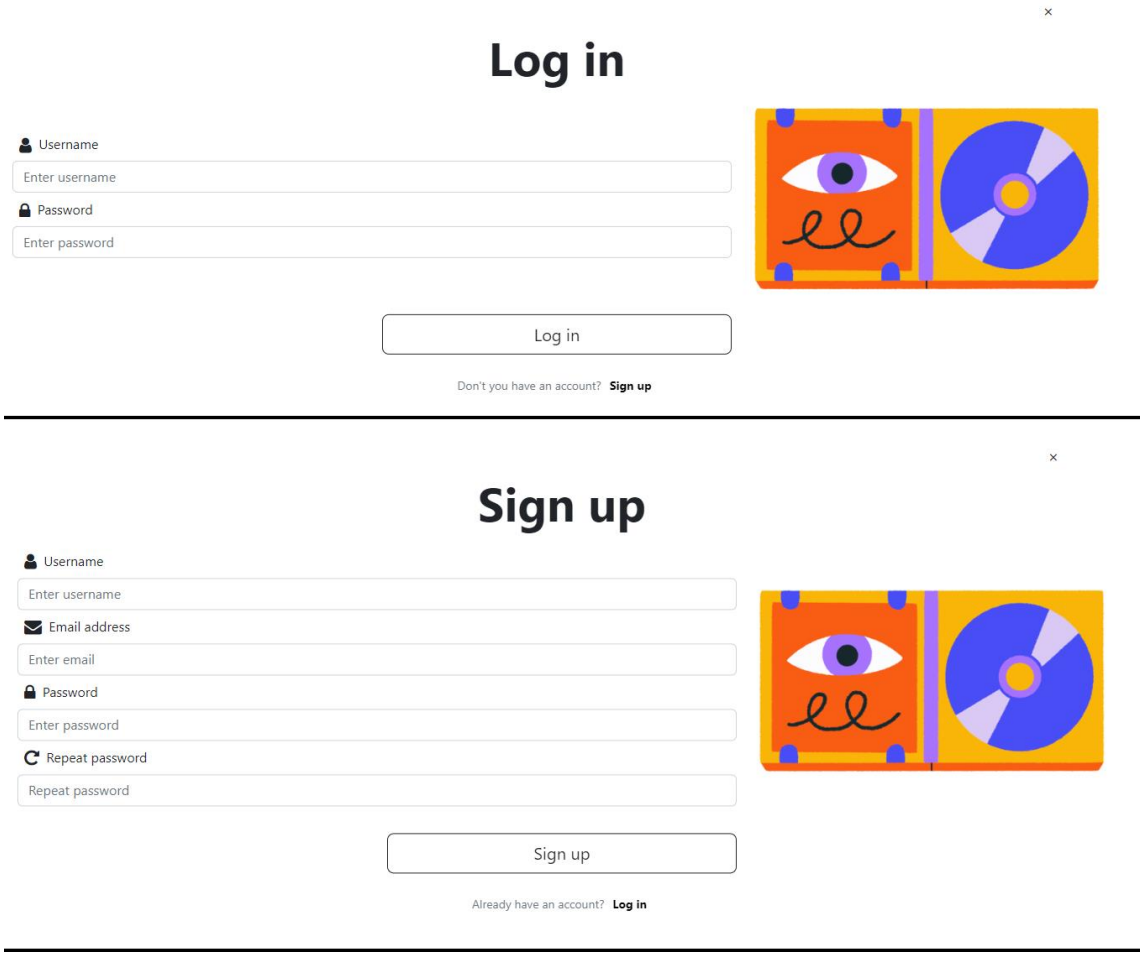

Figura 48. Formularios de inicio de sesión y creación de usuario. Fuente: elaboración propia.

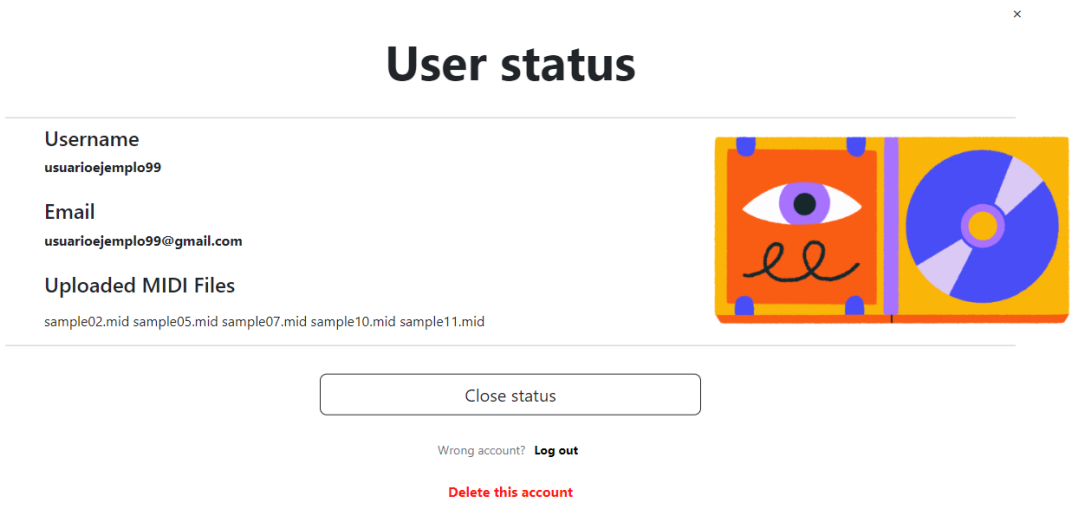

Figura 49. Formulario del estado del usuario. Fuente: elaboración propia.

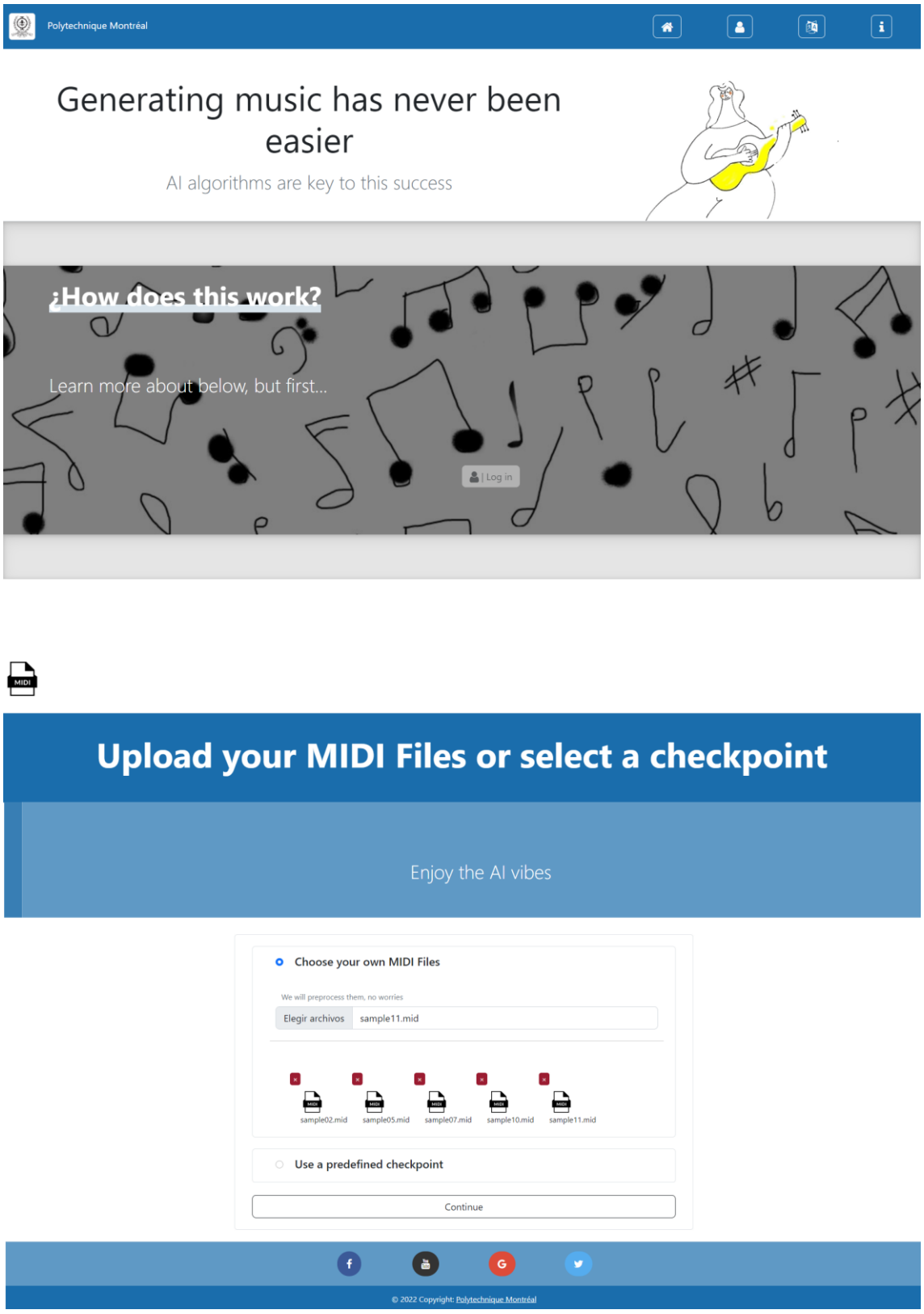

Figura 50. Página principal con inicio de sesión y entrenar IA seleccionado. Fuente: elaboración propia.

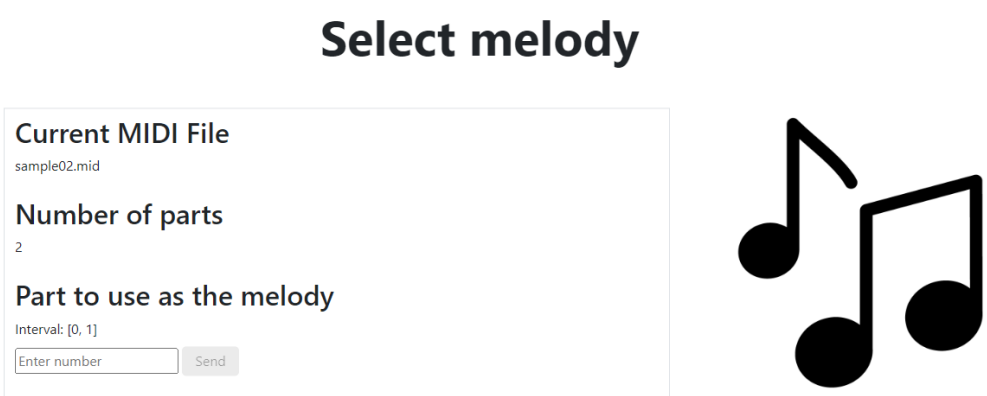

Figura 51. Formulario seleccionar melodía MIDI. Fuente: elaboración propia.

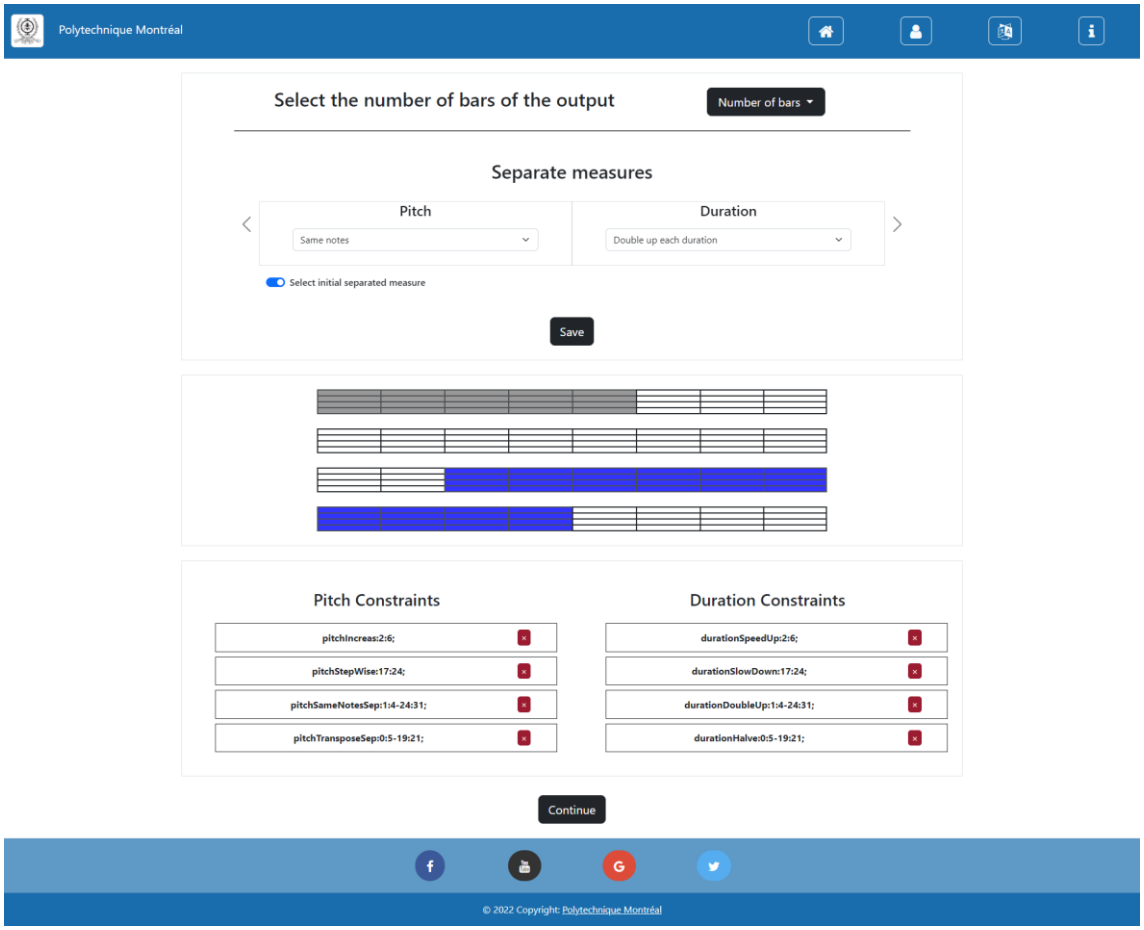

Figura 52. Seleccionar restricciones a la melodía. Fuente: elaboración propia.

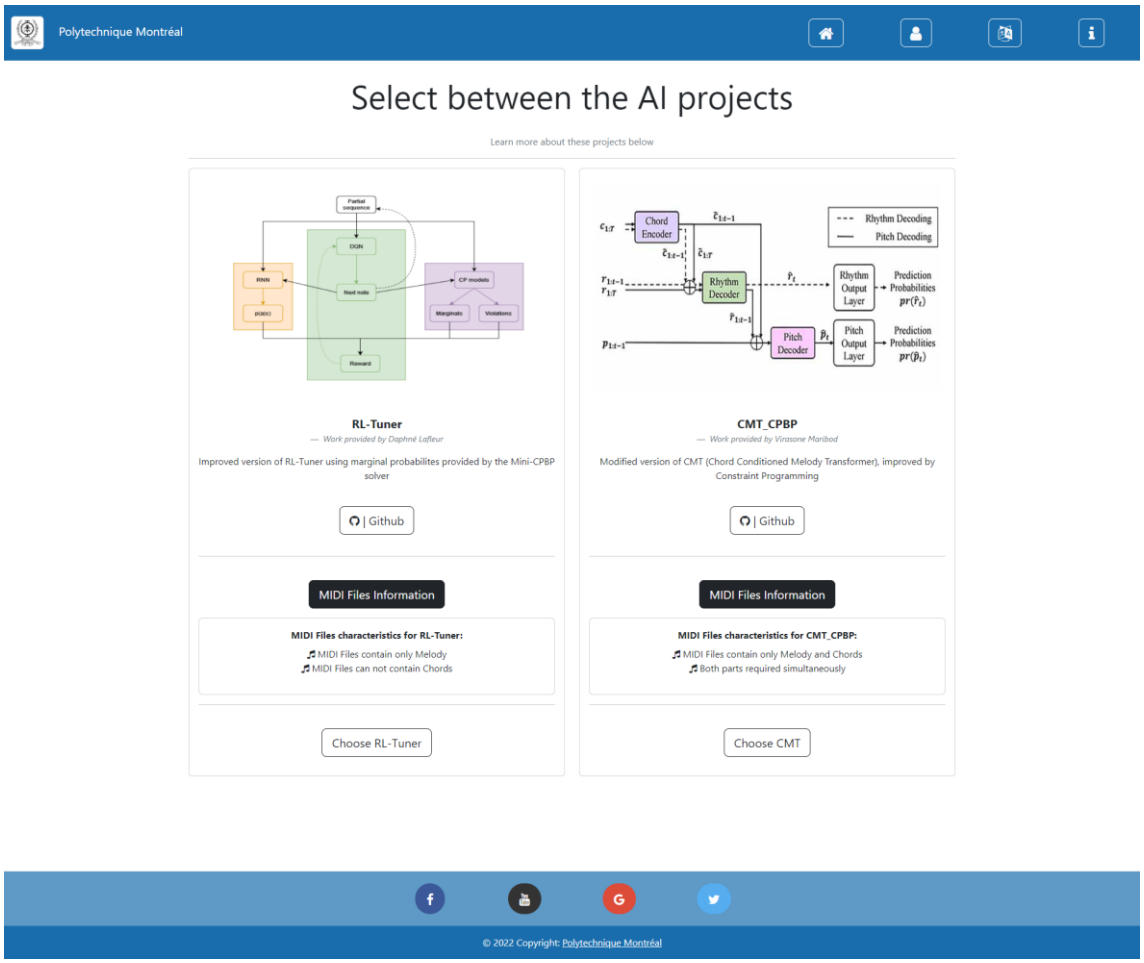

Figura 53. Página seleccionar algoritmo de IA. Fuente: elaboración propia.

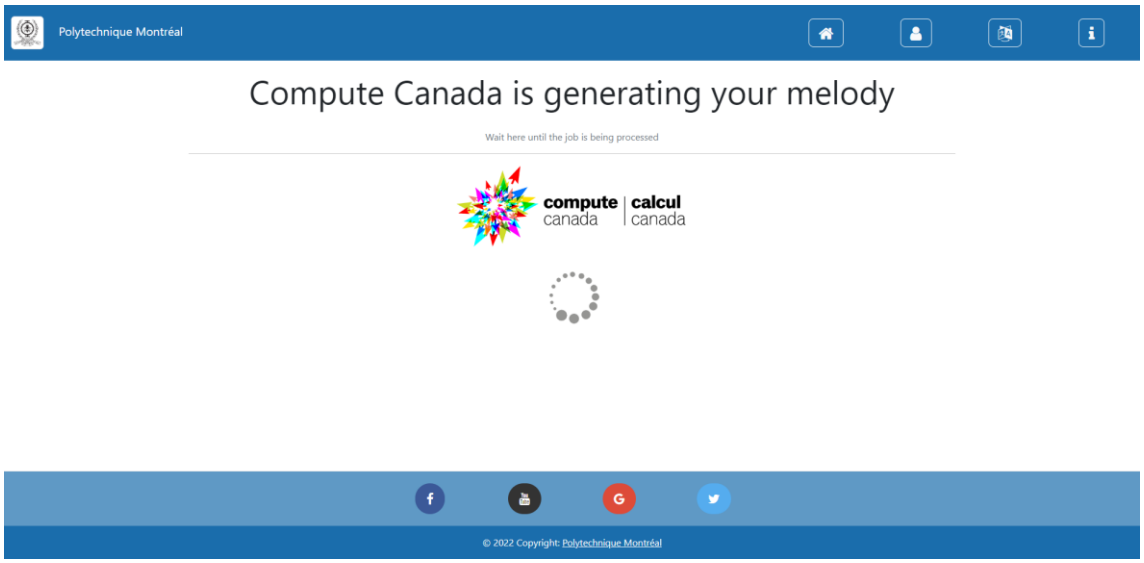

Figura 54. Página enviar trabajo al clúster y esperar respuesta. Fuente: elaboración propia.

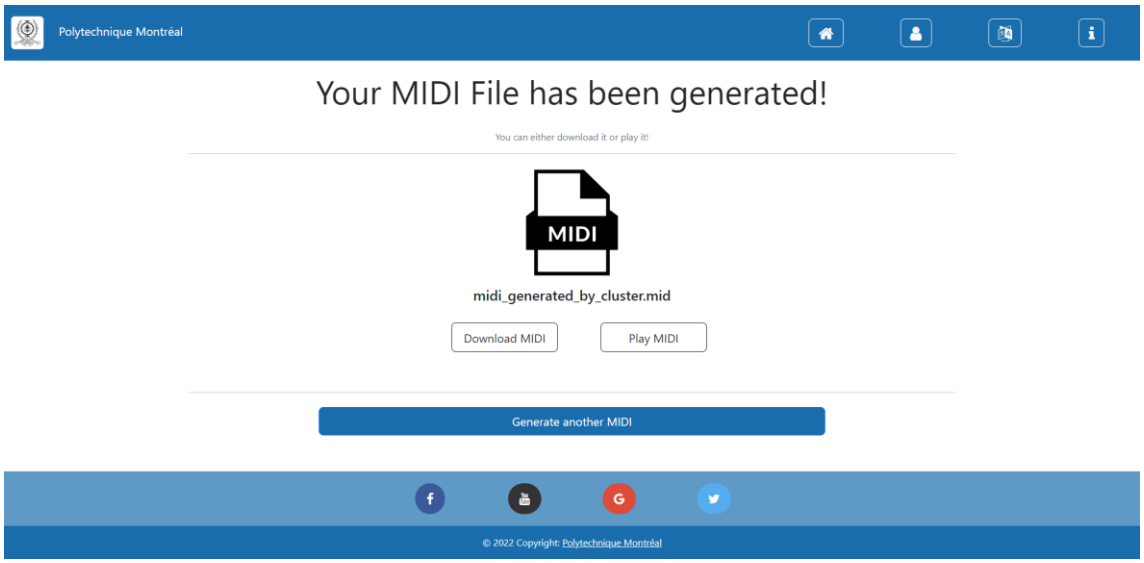

Figura 55. Página recibir MIDI del clúster. Fuente: elaboración propia.

9. Conclusiones

El apartado de conclusiones se dividirá en la evaluación por parte del alumno del trabajo desarrollado y cuál ha sido la relación del trabajo con los estudios cursados en el grado.

9.1 Valoración personal

El actual trabajo que se ha presentado a lo largo de toda la memoria fue realizado durante la estancia del alumno en Montreal, Canadá. Su paso por la escuela politécnica de Montreal ha hecho de él no solo un mejor profesional, sino una mejor persona.

Este trabajo fue el primer contacto con un proyecto de cierta envergadura donde no todo estaba claro y la correcta actuación del alumno era clave para lograr desarrollar el producto. El desarrollo web, ciertos lenguajes y sus *frameworks* eran prácticamente desconocidos y el aprender estos por sí solo desde 0 supuso un gran esfuerzo.

Los principales problemas que se encontraron fueron los definidos en los objetivos iniciales. El hecho de nunca haber desarrollado una plataforma web desde 0 e implementar toda su estructura, desde la parte visual hasta la parte lógica que la dota de funcionalidad, era el principal reto a asumir.

Se requería una plataforma web intuitiva, dinámica y funcional. Para solucionar esto, el alumno tuvo que recopilar información sobre diseño web, visitar las páginas web más usadas para captar ideas e investigar sobre los lenguajes y *frameworks* más utilizados en el desarrollo web. Tras esto, concluyó qué lenguajes y *frameworks* eran los que debía de usar para llevar a cabo su cometido.

El alumno aprendió autodidácticamente a través de YouTube y documentación los lenguajes de HTML, CSS y el *framework* de Bootstrap. Además, extendió sus conocimientos de JavaScript, enfocando este lenguaje al desarrollo web. Junto con todo este aprendizaje consiguió desarrollar el *frontend* de la aplicación.

Para el *backend,* el alumno investigó sobre cuáles son los lenguajes más demandados en el mercado. Así pues, concluyó con que Python, debido a su facilidad de uso y su corta curva de aprendizaje, era el elegido. El *framework* de Flask respondía a todo lo que la plataforma web necesitaba y no era complicado de aprender, por lo que esto fue otro motivo por el que escoger Python. Además, necesitábamos tratar los ficheros MIDI que el usuario iba a subir, y una de las mejores bibliotecas que existen para el tratado de estos estaba escrita en Python. Por todos estos motivos, el alumno comenzó a aprender Python y a dominar Flask y la biblioteca '*music21*'.

Por último, se tenía que construir una API desde la cual se pudiera acceder a la base de datos que almacenaba los usuarios de la aplicación. Como no se tenía ningún conocimiento previo de cómo se estructuraba una API, el alumno decidió utilizar algún *framework* que separase de forma clara las capas para aclarar los conceptos y fuera lo más escalable posible, por si el proyecto iba a más. Así pues, se desarrolló en C# y el *framework* NET Core, el cual tuvo que aprenderse desde 0.

Tras haber superado todos estos retos, se concluye con el buen trabajo realizado por el alumno. No solo tuvo que llevar a cabo un proyecto por sí solo, sino que tuvo que adaptarse a todas las adversidades que surgían por el camino. El hecho de no tener ningún concepto previo sobre desarrollo web llevó a cometer multitud de errores, y esto era complicado de resolver si no había nadie supervisando el proyecto que entendiese de esta materia.

Por ejemplo, en un inicio no se definió bien la arquitectura, ya que en las primeras fases de implementación el *frontend* contactaba directamente con la API, sin hacer uso de Flask. Otra de las complicaciones era la multitud de lenguajes nuevos con distintos paradigmas a aprender, así como los marcos de trabajo que se utilizaban. Encontrar información hoy en día sobre algo es fácil, pero lo complicado es que esta información sea valiosa. Muchos errores tomaron días para ser resueltos, ya que o bien la documentación no era clara o el problema no se encontraba en Internet. Algunas veces se tuvo que cambiar completamente el planteamiento de la solución para llevarla a cabo.

Pasar por todos estos obstáculos y ser capaz de resolverlos hizo de mí un mejor ingeniero. Sin embargo, una de las cosas que poca gente habla y que hace a alguien un verdadero profesional no es la facilidad de resolución, sino la constancia en el trabajo durante tiempos adversos. Durante mi larga estancia en Canadá surgieron problemas que han llegado a definir mi carácter, a hacerme una persona más adulta y a centrarme en lo que quiero. Estos problemas a veces te obstruyen como persona y te envían a lo más bajo de ti mismo. El encontrar la suficiente confianza, creencia y constancia para superarlos y llevar a cabo el trabajo por el que había viajado hasta este país tan lejano fue lo que en realidad hizo que todo esto valiese la pena.

9.2 Relación con los estudios cursados

A continuación, se presentará la relación que ha tenido el desarrollo de este proyecto con las asignaturas del grado y con las competencias transversales adquiridas durante la etapa universitaria.

9.2.1 Asignaturas de grado

- **Ingeniería del *software***: en esta asignatura se aprendió sobre metodologías, arquitecturas multicapa, modelación de Casos de Uso, etc. Para desarrollar la plataforma web se tuvo que hacer uso de todas estas técnicas adquiridas, desde recopilar los Requisitos Funcionales en la fase de Análisis mediante Casos de Uso a implementar las distintas capas en las que el *software* había sido separado.

- **Interfaces Persona – Computador:** los conocimientos que se enseñaron en esta asignatura tuvieron especial hincapié en la interacción que un usuario tiene con la parte visual de un sistema. Aquí se puso de manifiesto la importancia de la facilidad de uso de un sistema, de su consistencia visual para guiar al usuario, de la importancia del tamaño y color de las cosas, etc. Para el desarrollo del *frontend* se tuvo en cuenta todo lo aprendido en esta asignatura.

- **Gestión de proyectos:** la apropiada planificación de un proyecto teniendo en cuenta el tiempo que se tiene hasta su entrega, los recursos de los que se dispone y las posibles adversidades que pueden surgir durante el desarrollo es uno de los aspectos clave para el éxito. Esta asignatura nos enseñó a planificar nuestros proyectos, así pues, se hizo uso del conocimiento proporcionado creando un Diagrama de Gantt para planificar el desarrollo de la plataforma.

- **Base de Datos y Sistemas de Información:** en esta asignatura se aprendió a hacer consultas en SQL a Bases de Datos relacionales así como a diseñarlas. Los conocimientos proporcionados por la asignatura nos han permitido realizar un correcto diseño de la Base de Datos y entender como la API realiza las consultas sobre esta.

9.2.2 Competencias transversales

- **Análisis y resolución de problemas**: este trabajo implicó desarrollar un proyecto desde 0 sin previo conocimiento, por lo que surgieron una gran cantidad de errores que tuvieron que ser paliados. Así pues, se analizaron previamente las soluciones para dar con la óptima.

- **Aprendizaje permanente:** el desarrollar todo desde 0 y tener que aprender tantos lenguajes y *frameworks* nuevos ha hecho que hasta el último momento se haya aprendido.

- **Planificación y gestión del tiempo:** una correcta planificación del trabajo era necesaria para llevar a cabo el proyecto en el tiempo estipulado por el contrato de prácticas que tenía en la universidad canadiense. Así pues, una buena gestión del tiempo era imprescindible.

- **Aplicación y pensamiento crítico:** se han aplicado conocimientos y técnicas aprendidas durante la carrera. Para saber elegir por qué camino ir, qué lenguaje utilizar y cómo planificarse correctamente se ha hecho uso del pensamiento crítico.

- **Diseño y proyecto:** esta competencia transversal describe la habilidad de poder diseñar soluciones y llegar a desarrollar un plan para transformar estas soluciones en algo tangible como un proyecto. Como es obvio y como se ha descrito en la metodología seguida, se ha partido de una recolección de datos que ha llevado a un diseño. Finalmente, se ha hecho tangible este diseño en un producto final.

10. Trabajos futuros

En este último apartado se describirán qué trabajos futuros son de alta importancia para mejorar la plataforma web. Algunos de estos trabajos futuros no se han podido implementar por falta de tiempo, aunque otros han sido por indisponibilidad. De cualquier forma, el listado de mejoras que se observa a continuación aporta valor añadido al producto final:

- **Desarrollar las comunicaciones con el clúster**: actualmente, esta parte está parada por la imposibilidad de acceso al clúster. Una vez se consiga el acceso, es prioridad número uno el poder comunicarse con el clúster y recibir los trabajos que este calcule.

- **Recuperar las últimas restricciones añadidas por el usuario**: sería de interés poder recuperar los últimos movimientos que el usuario ha tenido a la hora de aplicar limitaciones a la melodía a generar. Así pues, si algún conjunto de restricciones le parece que ha funcionado bien, darle la opción de recuperarlo inmediatamente.

- **Deshabilitar/habilitar proyectos IA:** dependiendo del tipo de MIDIs que el usuario haya subido a la plataforma, podrá usar un proyecto u otro. Si el usuario no hubiese subido ninguna MIDI con acordes y melodía, existe un proyecto que no puede ser usado, por lo que debe de ser deshabilitado. La implementación de esto es sencilla, simplemente basta con comprobar qué carpetas tiene creadas internamente este usuario.

- **Interfaz gráfica que muestre la melodía generada:** sería de interés que se mostrase la MIDI generada por el clúster en un formato legible e interactivo para el usuario. Así, este a parte de observarla, podrá realizar cambios en las notas de manera dinámica.

- **Crear un repositorio de MIDIs generadas para los usuarios:** sería interesante que los propios usuarios compartiesen lo que han generado, con qué restricciones lo han hecho y con qué han entrenado (o no) el algoritmo. Desarrollar un apartado parecido a este expandiría la comunidad de usuarios y la utilidad de la plataforma.

11. Bibliografía

1. Choi, K., Park, J., Heo, W., Jeon, S., & Park, J. (2021). *IEEE Xplore. Chord Conditioned Melody Generation With Transformer Based Decoders*. Recuperado de https://ieeexplore.ieee.org/abstract/document/9376975/figures#figures

2. Jackson, L. (2020). *.NET Core 3.1 MVC REST API - Full Course*. Recuperado de https://www.youtube.com/watch?v=fmvcAzHpsk8&t=4407s&ab_channel=LesJackson

3. Kaalel. (2023). *Geeks for Geeks. MVC Framework - Introduction*. Recuperado de https://www.geeksforgeeks.org/mvc-framework-introduction/

4. Kumar Pal, S. (2022). *Geeks for Geeks. Software Engineering | Classical Waterfall Model*. Recuperado de https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/

5. Lafleur, D., Chandar, S., & Pesant, G. (2022). *DROPS. Combining Reinforcement Learning and Constraint Programming for Sequence-Generation Tasks with Hard Constraints*. Recuperado de https://drops.dagstuhl.de/opus/volltexte/2022/16659/

6. Scott Asato Cuthbert, M. (2023). *music21 Documentation*. Recuperado de http://web.mit.edu/music21/doc/index.html

7. TIOBE. (2023). *TIOBE Index*. Recuperado de https://www.tiobe.com/tiobe-index/

8. Virasone, M. (2022). *POLYPUBLIE Polytechnique Montréal*. Recuperado de https://publications.polymtl.ca/10495/

9. W3Techs. (2023). *W3Techs*. Recuperado de https://w3techs.com/technologies/overview/client_side_language

10. Witt, S. (2019). *OSFHome. Compute Canada*. Recuperado de https://osf.io/k89fh/wiki/Compute%20Canada/

Anexo 1

```
import music21
import os
import shutil
import stat
import sys

def MIDIConverter(path_to_file, cm_path, om_path, garbage_path):

    midifileMusicScore = music21.converter.parse(path_to_file)
    partsMIDIMusicScore =
len(midifileMusicScore.recurse().getElementsByClass(music21.stream.Part))
    return(midifileMusicScore, partsMIDIMusicScore)


def MIDISelector(path_to_file, cm_path, om_path,
garbage_path,midifileMusicScore,part_vble):
    create_folder_if_not_exists(cm_path)
    create_folder_if_not_exists(om_path)
    create_folder_if_not_exists(garbage_path)
    #the music Score we create as an output only for the Melody
    outputMusicScoreOM = music21.stream.Score()

    #the music Score we create as an output for the Melody + Chords
    outputMusicScoreMC = music21.stream.Score()

    #First we extract the Chord Part using .chordify()
    streamChords = midifileMusicScore.chordify()
    for c in streamChords.recurse().getElementsByClass('Chord'):
        c.closedPosition(forceOctave=4, inPlace=True)

    streamMelody =
midifileMusicScore.getElementsByClass(music21.stream.Part)[part_vble]

    #our MIDI File input contains Melody as well as Chords, so we create
2 MIDI Files, 1 with C&M, and the other one only with M
    outputMusicScoreMC.insert(0, streamMelody)
    outputMusicScoreMC.insert(0, streamChords)
    outputMusicScoreOM.insert(0, streamMelody)
```

```
    #for the MIDI File we created only for Melody
    shutil.move(str(outputMusicScoreOM.write(fmt="Midi")), om_path)
    #outputMusicScoreOM.show()                                          #this
is the function for displaying the music score of the MIDI File which
it's still in the 'music21' format. The bad part is that the code waits
until we are done with MusicScore. Maybe we should parallelize this
.show() section
    #for the MIDI File we created with M&C
    shutil.move(str(outputMusicScoreMC.write(fmt="Midi")), cm_path)
    #outputMusicScoreMC.show()                                           #thi
s is the function for displaying the music score of the MIDI File which
it's still in the 'music21' format. The bad part is that the code waits
until we are done with MusicScore. Maybe we should parallelize this
.show() section
    #we need to store the MIDI Files we used for create the MusicScores
(used for generating more MIDI FIles), and delete it after
    shutil.move(path_to_file, garbage_path)

def delete_folder_if_exists(path_in):
    if(os.path.exists(path_in)):
        os.chmod(path_in, stat.S_IRWXU)
        shutil.rmtree(path_in)

def create_folder_if_not_exists(path_in):
    if(os.path.exists(path_in) == False):
        os.mkdir(path_in)

def path_establish(username):
    cwd = os.getcwd().replace('\\', '/')
    directory = cwd + '/users/' + username + '/midi_files_checking'
    path_om = directory + "/only_melody_midi"
    path_cm = directory + "/chords_melody_midi"
    path_MIDI_garbage = directory + "/garbage"
    return(cwd, directory, path_om, path_cm, path_MIDI_garbage)
```

```
#-----------------------------------MAIN-----------------------------#
def process():
    #we establish our paths
    print('files processing')
    cwd = os.getcwd().replace('\\', '/')
    directory = cwd + '/midi_files_checking'
    path_om = directory + "/only_melody_midi"
    path_cm = directory + "/chords_melody_midi"
    path_MIDI_garbage = directory + "/garbage"

    #we check if the files where we have to store the MIDI files have
been created previously, in order to delete them and its content
    delete_folder_if_exists(path_om)
    delete_folder_if_exists(path_cm)
    #just if there's any problem with the garbage file where we store
Files
    delete_folder_if_exists(path_MIDI_garbage)

    for f in os.scandir(directory):
        if (f.is_file() and f.name.endswith('.mid')):
            MIDIConverter(str(f.path), path_cm, path_om,
path_MIDI_garbage)


    #we delete the MIDI FIles we could not move (the raw ones with M&C)
    delete_folder_if_exists(path_MIDI_garbage)
```

Anexo 2

ODS

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				**X**
ODS 2. Hambre cero.				**X**
ODS 3. Salud y bienestar.				**X**
ODS 4. Educación de calidad.		**X**		
ODS 5. Igualdad de género.				**X**
ODS 6. Agua limpia y saneamiento.				**X**
ODS 7. Energía asequible y no contaminante.				**X**
ODS 8. Trabajo decente y crecimiento económico.	**X**			
ODS 9. Industria, innovación e infraestructuras.		**X**		
ODS 10. Reducción de las desigualdades.	**X**			
ODS 11. Ciudades y comunidades sostenibles.				**X**
ODS 12. Producción y consumo responsables.				**X**
ODS 13. Acción por el clima.				**X**
ODS 14. Vida submarina.				**X**
ODS 15. Vida de ecosistemas terrestres.				**X**
ODS 16. Paz, justicia e instituciones sólidas.				**X**
ODS 17. Alianzas para lograr objetivos.				**X**

Relación entre los ODS y el trabajo realizado

- **ODS 4. Educación de calidad:** este objetivo promueve una educación equitativa, significante y de valor para todos. La educación es un síntoma de la prosperidad económica y social. Este objetivo tiene una relación de nivel medio con el trabajo ya que proporcionamos una herramienta interactiva de generación musical donde cualquiera puede experimentar con ella. Así pues, puede aprender con qué restricciones una melodía es subjetivamente mejor, o con qué se debe de entrenar el algoritmo para obtener lo buscado.

- **ODS 8. Trabajo decente y crecimiento económico:** el objetivo número 8 de la Agenda 2030 es uno de los más representativos en nuestro trabajo. Con nuestro trabajo se pretende visualizar proyectos desarrollados en un laboratorio que de otra forma no hubiesen salido a la luz. Dándoles visibilidad aumenta la satisfacción del desarrollador, motivándolo para seguir creando. Esto repercute en mejores proyectos los cuales pueden conducir a conseguir mejores trabajos o mayor financiación para la universidad.

- **ODS 9. Industria, innovación e infraestructuras:** la relación con este objetivo es de nivel medio, ya que no tiene nada que ver con la mejora de infraestructuras. Sin embargo, se promociona la innovación dándole visibilidad, haciendo más conocida la industria y, por ende, repercutiendo más en el mercado global.

- **ODS 10. Reducción de las desigualdades:** el hecho de acercar nuevas tecnologías a cualquier persona con una conexión a Internet y un ordenador hace que todo el mundo desde cualquier parte pueda explotar estas herramientas. Así pues, la accesibilidad que esta plataforma ofrece casa con el objetivo 10 de la Agenda 2030, ofreciendo algoritmos punteros a cualquier individuo con interés en estos de manera gratuita.