PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito parcial para obtener el Título de INGENIERO DE SISTEMAS

INTEGRACIÓN DE LA INTERFAZ DE PROGRAMA DE APLICACIÓN PARA EL BRAINCEMISID EN UNA ARQUITECTURA DE SOFTWARE MULTINIVEL

Por

Br. Kristo López

Tutor: Dr. Gerard Páez

Enero 2020

INTEGRACIÓN DE LA INTERFAZ DE PROGRAMA DE APLICACIÓN PARA EL BRAINCEMISID EN UNA ARQUITECTURA DE SOFTWARE MULTINIVEL

Br. Kristo Rafael López Rojas

Proyecto de Grado -Sistemas Computacionales-52 páginas

Resumen: El BrainCEMISID es un proyecto creado en el Centro de Estudios en Micro-Computación y Sistemas Distribuidos, el cual ha desarrollado un cerebro artificial basado en redes neuronales logrando así emular comportamientos del cerebro humano tales como el reconocimiento de patrones, ejecutar operaciones de adición, comprensión del concepto de cantidad, lectura de palabras, silabas y estado mental de la intención. Sin embargo, este proyecto estaba integrado a una interfaz que ofrecía una interacción rustica con el usuario, haciendo su uso complicado y que llevara mucho tiempo para generar un estímulo. Es por esto que se llevó a cabo una migración del BrainCEMISID a una arquitectura multinivel y se desarrolló un conjunto de interfaces de programa de aplicación para el soporte de varios usuarios con la posibilidad de tener varios proyectos almacenados y que se lograra establecer un protocolo de comunicación con un cliente web.

Palabras claves: Cerebro Artificial, Interfaz de programa de aplicación, Estimulo, Neurona artificial, Protocolo.

Índice

Índice de figuras

Capítulo 1

Introducción

El desarrollo del software ha ido evolucionando de manera acelerada en estos últimos años, esto ha dado lugar a distintas vertientes en distintas áreas de trabajo, tales como la inteligencia artificial, análisis estadístico, desarrollo web, software para celulares y televisores inteligentes entre otros, sin embargo, debido a que los campos para el desarrollo son muy amplios, se han venido abriendo brechas en distintas áreas especializándose y consolidándose en una tecnología en específico. A Pesar que existen cierta homogeneidad con respecto al desarrollo del software en un área en específico siempre se es indispensable la necesidad de comunicarse con otro tipo de sistema con una arquitectura diferente de manera eficiente, inclusive, se ha dado pie a arquitecturas de múltiples niveles que hacen necesaria la idea de tener algo entre ambos límites de estos niveles para que se puedan comunicar. Es por esto que se plantea la idea de crear una interfaz entre ellas que facilite esta comunicación y que brinde todas las opciones necesarias para poder manipular datos que son solicitados de un nivel a otro, esta interfaz es denominada entonces "Interfaz de programa de aplicación" (Application Program Interface en inglés) que establece los protocolos necesarios para la comunicación y cumple con lo antes ya establecido.

1.1 Antecedentes

El termino cerebro artificial es utilizado habitualmente para describir la investigación que pretende desarrollar hardware y software con habilidades cognitivas similares a la del cerebro humano. Uno de los enfoques más comunes para desarrollar un cerebro artificial es usando redes neuronales artificiales sobre una computadora.

Las redes neuronales artificiales son un modelo abstracto de su homologo biológico mucho más simplificado y menos complejo. Estas redes consisten en un conjunto de neuronas artificiales que son conectadas entre ellas y se transmiten señales las cuales son procesadas dentro de sí mismas, dando lugar a una salida como una respuesta ante el estímulo de esta red. Estos sistemas computacionales sirven para resolver problemas tales como el reconocimiento de patrones para que luego estos sean almacenados, etiquetados y sus conceptos sean comprendidos como lo sería una letra, una palabra, un color o inclusive, el reconocimiento de un objeto o un ser vivo.

El BrainCEMISID es un cerebro artificial desarrollado por el Centro de Estudios en Microelectrónica y Sistemas Distribuidos (CEMISID) en la facultad de ingeniería de la Universidad de los Andes de Venezuela. Este proyecto fue iniciado en el año 2013 por el Dr. Gerard Páez, junto a un equipo de estudiantes de Pregrado de Ingeniería de Sistemas. Este proyecto tiene la finalidad de construir un cerebro artificial basado en redes neuronales mediante el dialogo socrático entre los estudiantes y el profesor, los cuales presenta el desafió de "atacar la complejidad" mediante el uso de la creatividad. Uno de los principios fundamentales de estos diálogos establecidos por el Dr. Páez es usando La Navaja de Ockham, este término se puede definir brevemente como, de varias teorías que expliquen un suceso, se debe tomar la cual presente menos complejidad.

En el año 2012 se daría a realizar la tesis "0" la cual sería la primera versión del BrainCEMISID basada en una red neuronal artificial de base radial en tecnología VLSIC (Very Large Scale Integrated Circuits) hecha por Andrade (2012), y Rangel (2012) quien desarrollo el mismo tipo de neurona en programación paralela en el ambiente CUDA, siendo esta las bases para los siguientes proyectos.

Luego de esto, se conocería la versión de este cerebro, pero en un ambiente virtualizado hecha por Monsalve (2014) con una estructura de 4 esferas, siendo estas la esfera sensorial, la esfera perceptora, la esfera vectorizadora y la esfera para los bloques neuro-sensoriales, todo esto llevado a cabo en la plataforma de desarrollo de Nvidia, CUDA.

Un año más tarde Muchacho (2015) desarrolla las esferas analítica, relacional y cultural, para resolver problemas de ambigüedades a la hora que el cerebro recibiese un estímulo, más aun, también se logra relacionar los sentidos de la vista con él oído. Luego de esto, Graterol (2015) le añade la característica al cerebro de la comprensión del concepto de cantidad y seguidamente Bruzual (2015) le da la capacidad para la adición de cantidades por medio de la memoria. Por otra parte, también se añade los protocolos necesarios para reconocimiento de letras y palabras.

En el año 2016, Sosa (2016) crea la capacidad del cerebro de hacer procesamiento en paralelo, tomando esta vez las ventajas que puede ofrecer un procesador multinúcleos para realizar las tareas de forma eficiente, en esta etapa se desarrolla la capacidad de pensamiento que se presenta como el flujo de las diferentes esferas que constituyen el cerebro artificial. Seguidamente Fernández (2016) hace una reingeniería de software para mejorar la integración de todas las versiones anteriores. Este trabajo deja un marco de desarrollo para las siguientes versiones y se desarrolla el estado mental de la intención.

Al siguiente año EL Halabi (2017) añade la característica de un "soporte intermedio" (middleware en inglés) para la mejora del rendimiento del procesamiento en paralelo del cerebro.

Finalmente, el trabajo más actualizado de este proyecto es el de su desarrollo de Araujo (2019) por el gusto de manera autónoma basado en la predicción de eventos sonoros que en su primera instancia son caracterizados para luego ser almacenados dentro del cerebro artificial en forma de neuronas y crear una red de ellas con impulsos pseudoaleatorios que permiten soportar la teoría del gusto por la música.

1.2 Planteamiento del problema

Hasta ahora, el cerebro tiene la capacidad de reconocer patrones gráficos (Graterol, 2015), conceptualizar cantidades (Monsalve, 2014), tener estado mental de la intención (Fernández, 2016), poder hacer reconocimiento dental para aplicaciones odontológicas (García, 2019) y tener gusto autónomo en la música (Araujo, 2019).

Sin embargo, el cerebro presenta limitantes de capacidad de procesamiento y almacenamiento, por esto, se plantea convertirlo a una arquitectura multinivel para poder interactuar de manera remota con el cerebro. Se necesita que se almacene los datos y realice el procesamiento de forma independiente a la aplicación del usuario. Esta capacidad se adquiere cuando Vílchez (2019) hace el levantamiento del BrainCEMISID a un dispositivo remoto, es entonces que se plantea el problema de comunicación entre el usuario y el cerebro. A pesar que el cerebro está aislado, existe una comunicación muy rustica y poco eficiente para que el usuario pueda manejarlo, más aun, existe la capa que se comunica con el usuario que es la encargada de recibir sus solicitudes y mostrárselas desde una maquina externa de donde se encuentra el cerebro artificial. Es por esto que surge la necesidad de poder crear un interfaz que sirva como una conexión entre ellas para que así el usuario pueda manejar el cerebro.

1.3 Objetivos

1.3.1 Objetivo general

El objetivo general de este proyecto es desarrollar una interfaz de programa de aplicación para establecer un protocolo de comunicación con el cerebro en un entorno remoto y la siguiente capa de comunicación con el usuario.

1.3.2 Objetivo específicos

1. Estudiar acerca del funcionamiento de las entradas y salidas de datos del BrainCEMISID.
2. Diseñar el protocolo de comunicación entre el cerebro y la siguiente capa de comunicación con el usuario.

3. Implementar las funciones con los protocolos ya establecidos para que se hagan las transformaciones necesarias para la comunicación entre el cerebro y la siguiente capa de comunicación con el usuario.

1.4 Justificación

Se define como Interfaz de programa de aplicación por el diccionario de la universidad de Oxford (2019) como un set de funciones y procedimientos que permiten la creación de aplicaciones que acceden a las características o datos de un sistema operativo, aplicación u otro servicio.

El BrainCEMISID se describe como un proyecto de un cerebro artificial que pretende llevar a cabo tareas complejas y de múltiples disciplinas que tiene desarrolladas y que se irán desarrollando acorde a que se vayan implementando nuevos módulos en este proyecto. Este Cerebro parte de la premisa de el uso de la creatividad para resolver problemas complejos, surgiendo así, ideas intuitivas que son la solución a los problemas con respecto a su desarrollo. La idea de migrar este cerebro a una arquitectura de software multinivel, emerge de la problemática de su capacidad de procesamiento y almacenamiento, Sin embargo, se plantea que el cerebro debe ser movido a un entorno remoto, por esto el usuario está alejado y con una comunicación rustica entre el cerebro y el, por ello, se planteo en primera instancia hacer una interfaz de programa de aplicación haciendo que esta capa sea la responsable de una comunicación más eficiente entre el cerebro y la próxima capa que será la responsable de comunicarse finalmente con el usuario.

1.5 Alcance

El alcance de este proyecto se limitó a establecer una serie de funciones y procedimientos necesarios y suficientes, que se rijan por un protocolo que definió los patrones para el envío y el recibimiento de los datos requeridos por el usuario hacia el cerebro artificial, esto se hizo por medio de la construcción de una interfaz de programa de aplicación, para establecer una comunicación eficiente entre la capa superior que interactúa con el usuario y el BrainCEMISID. Además, se contó con una aislación de los datos del cerebro en una base de datos y se establecio una comunicación igualmente entre el cerebro artificial y ella.

1.6 Metodología

1.6.1 Descripción de la metodología modelo espiral

El modelo espiral es una combinación de un modelo en cascada y un modelo iterativo. Cada fase en el modelo en espiral comienza con un objetivo de diseño y termina con el cliente revisando el progreso. El modelo en espiral fue mencionado por primera vez por Barry Boehm en su artículo de 1986.

El equipo de desarrollo en el modelo Spiral-SDLC comienza con un pequeño conjunto de requisitos y pasa por cada fase de desarrollo para ese conjunto de requisitos. El equipo de ingeniería de software agrega funcionalidad para el requisito adicional en espirales cada vez mayores hasta que la aplicación esté lista para la fase de producción (Guru,2020).

1.6.2 Fases de la metodología

Cada ciclo en la espiral representa una fase del proceso de desarrollo del software. Así, el ciclo más interno podría referirse a la viabilidad del sistema, el siguiente ciclo a la definición de requerimientos, el siguiente ciclo al diseño del sistema, y así sucesivamente.

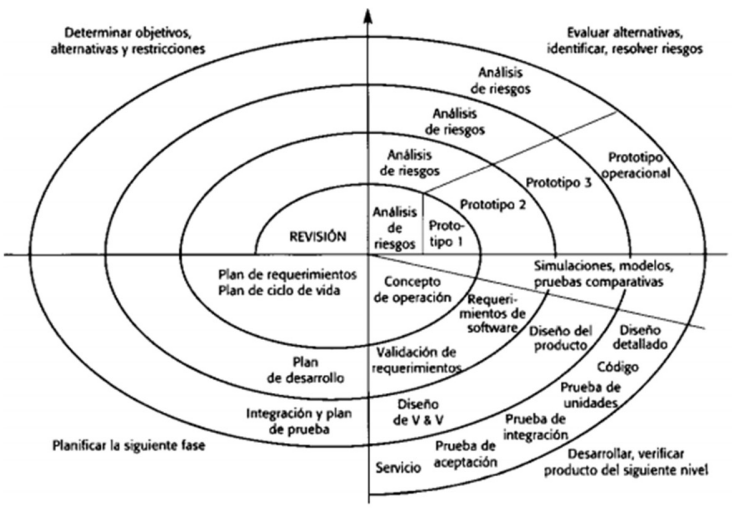

Figura 1.1 Modelo en espiral. *Tomada de* (*Sommerville, 2005*).

1.6.3 Descripción de las fases de la metodología

Cada ciclo de espiral se divide en cuatro sectores:

1. **Definición de objetivos:** para esta fase del proyecto se definen los objetivos específicos. Se identifican las restricciones del proceso y el producto. Dependiendo de estos riesgos, se planean estrategias alternativas.

2. **Evaluación y reducción de riesgos:** Se lleva a cabo un análisis detallado para cada uno de los riesgos del proyecto identificados. Se definen los pasos para reducir dichos riesgos. Por ejemplo, si existe el riesgo de tener requerimientos inapropiados, se puede desarrollar un prototipo del sistema.

3. **Desarrollo y validación:** Después de la evaluación de riesgos, se elige un modelo para el desarrollo del sistema. Por ejemplo, si los riesgos en la interfaz de usuario son dominantes, un modelo de desarrollo apropiado podría ser la construcción de prototipos evolutivos. Si los riesgos de seguridad son la principal consideración, un desarrollo basado en transformaciones formales podría ser el más apropiado, y así sucesivamente. El modelo en cascada puede ser el más apropiado para el desarrollo si el mayor riesgo identificado es la integración de los subsistemas.

4. **Planificación:** el proyecto se revisa y se toma la decisión de si se debe continuar con un ciclo posterior de la espiral. Si se decide continuar, se desarrollan los planes para la siguiente fase del proyecto.

La diferencia principal entre el modelo en espiral y los otros modelos pertenecientes a la ingeniería de software es la consideración explicita del riesgo en el modelo en espiral. Informalmente, el riesgo significa sencillamente algo que puede ir mal. Por ejemplo, si la intención es utilizar un nuevo lenguaje de programación, un riesgo es que los compiladores disponibles sean poco fiables o que no produzcan código objeto suficientemente eficiente. Los riesgos originan problemas en el proyecto, como los de confección de agendas y excesos en los costos; por lo tanto, la disminución de riesgos es una actividad muy importante en la gestión del proyecto.

Debido a la naturaleza del proyecto, usar esta metodología del proyecto no solo asegura un manejo eficiente de "riesgos' sino que es en cada iteración se tendrá una versión parcial cada vez mas cercana a los objetivos planteados haciendo esta aproximación incremental sin comprometer la finalización del proyecto.

1.7 Estructura del documento

La Estructura del documento se organiza de la siguiente manera:

El **capítulo 1:** Se da una breve introducción acerca del proyecto, luego se describe conceptos esenciales para la comprensión del problema, tales como antecedentes que respaldan información acerca de lo que se ha hecho hasta ahora en relación al proyecto de grado, se exponen los objetivos generales y específicos. Luego de esto se presenta la justificación, el alcance y finaliza con la descripción de la metodología seguida metodología en el proyecto.

El **capítulo 2:** Contiene el marco teórico, en esta sección se muestran las bases teóricas que sirven para levantar el proyecto. Se describe que es un cerebro artificial, las redes neuronales artificiales, neuronas artificiales. Luego se hablará de la arquitectura de software a utilizar, el tipo de interfaz de programa de aplicación necesario para la comunicación del cerebro y el usuario y de las posibles tecnologías que se pueden usar para construir toda esta estructura.

El **capítulo 3:** Contiene la arquitectura de software, en esta sección se habla de como esta estructurado el proyecto para poder sentar las bases de un modelo detallado de sus componentes.

El **capítulo 4:** Contiene la implementación y pruebas, en esta sección se habla a fondo de las tecnologías escogidas y se justifica el uso de ellas en el proyecto, luego de esto se procede a describir la implementación de cada una de ellas en los componentes por separado, finalmente se habla sobre las pruebas que se hicieron para el correcto funcionamiento del api, así como también algunas otras funcionalidades adicionales que el proyecto requería.

El **capítulo 5:** Finalmente en esta sección se habla de las conclusiones a las que se llegaron del proyecto y se habla generalmente de posteriores trabajos usando el proyecto final.

Capítulo 2

Marco Teórico

2.1 El cerebro artificial

Se define como cerebro artificial como la combinación de software y hardware con habilidades cognitivas similares a los animales o al cerebro humano. De Garis, Shuo, Goertzel, & Ruiting, (2010) plantean que, debido a los rápidos avances en hardware de computadora, neurociencia y ciencias de la computación, las investigaciones con cerebros artificiales y su desarrollo están floreciendo. Una de las formas de desarrollo de un cerebro artificial es basándose en redes neuronales artificiales, las cuales permiten procesar varias entradas y generar una abstracción de forma similar a su homónimo biológico.

2.2 Redes neuronales artificiales

Las redes neuronales artificiales son sistemas inspirados en los cerebros biológicos cuya intención es replicar la forma en la que los humanos u animales aprenden. Los Autores Russel & Norvig, (2004) definen en su libro una neurona como una célula del cerebro cuya función principal es la recogida, procesamiento y emisión de señales eléctricas. Se piensa que la capacidad de procesamiento de información del cerebro proviene principalmente de redes de este tipo de neuronas. Russel et al (2004) describen que, por esta razón, los primeros trabajos de Inteligencia artificial pretendían crear redes neuronales artificiales.

Por lo general, se presentan como una organización de "neuronas" interconectadas que pueden calcular valores a partir de entradas proporcionando información a través de la red. Las redes neuronales están característicamente estructuradas en capas. Las capas están formadas por una serie de 'nodos' interrelacionados que contienen una 'función de activación'. Los patrones están disponibles para la red a través de la 'capa de entrada', que se comunica con una o más 'capas ocultas' donde el procesamiento concreto se realiza mediante un sistema de 'conexiones' subjetivas. Las capas ocultas se unen a una 'capa de salida' donde la respuesta es la salida final del sistema (Sthapak, Khopade, Kashid, 2013).

Cada neurona recibe como entrada un conjunto de señales discretas o continuas, las pondera e integra, y transmite el resultado a las neuronas conectadas a ella. Cada conexión entre dos neuronas tiene

una importancia determinada asociada, denominada peso. En los pesos se suele guardar la mayor parte del conocimiento que la red neuronal tiene sobre la tarea en cuestión. El proceso mediante el cual se ajustan estos pesos para lograr un determinado objetivo se denomina aprendizaje o entrenamiento (Araujo, 2019).

Matich (2001) nos plantea que las Redes neuronales artificiales son redes interconectadas masivamente en paralelo de elementos simples (usualmente adaptativos) y con organización jerárquica, las cuales intentan interactuar con los objetos del mundo real del mismo modo que lo hace el sistema nervioso biológico.

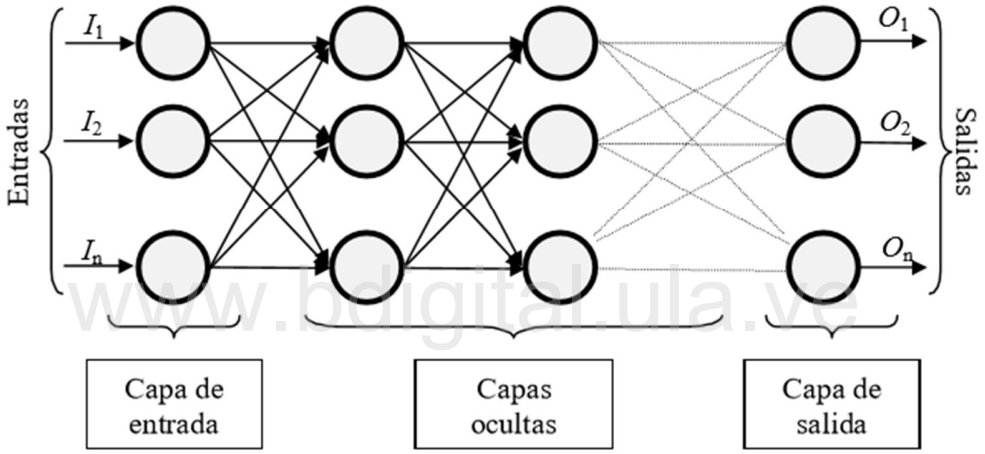

Figura 2.1 Capas de una red neuronal artificial. *Tomada de (Matich, 2001).*

La misma está constituida por neuronas interconectadas y arregladas en tres capas. Los datos ingresan por medio de la "capa de entrada", pasan a través de la "capa oculta" y salen por la "capa de salida". Cabe mencionar que la capa oculta puede estar constituida por varias capas (Matich, 2001).

2.2.1 Neuronas artificiales

Kriesel (2005) define también, que una red neuronal consiste en simple unidades de procesamiento llamadas neuronas las cuales tienen conexiones con dirección y peso entre ellas. Como su contraparte biológica podemos hacer una abstracción y hacer una aproximación técnica.

- **Entrada Vectorial:** Las entradas de las neuronas artificiales consisten en muchos componentes es por esto que puede ser representado como un vector.
- **Salida Escalar:** La salida de una neurona es un escalar, esto significa que la salida de la neurona solo consiste en un componente. Varias salidas escalares conforman la entrada vectorial de otra neurona.
- **Cambio de entrada por sinapsis:** En las redes neuronales artificiales las entradas son pre procesadas también. Estas son multiplicadas por un número para obtener el peso. Los conjuntos de estos pesos representan la información de almacenamiento de una red neuronal tanto en la biológica como en la artificial.
- **Acumulamiento de entradas:** En biología, las entradas son resumidas a un pulso acorde a cambios químicos, por ejemplo, si son acumuladas, sin embargo, por el lado artificial esto es realizado a menudo por la suma de pesos. Esto significa que después de las acumulaciones nosotros solo continuamos con un valor, un escalar en lugar de vectores.
- **Características no lineares:** La entrada de la neurona artificial es también no proporcional a su salida.
- **Pesos Ajustables:** Los pesos pesando las entradas son variables, similares a los procesos químicos en las hendiduras sinápticas. Esto añade una gran dinámica a las redes porque una gran parte del "conocimiento" de una red neuronal es guardada en los pesos y en la forma y poder de procesos químicos en una hendidura sináptica.

2.3 Programación por capas

La programación por capas o arquitectura multinivel es un modelo de desarrollo de software en donde la presentación, el procesamiento de la aplicación y el manejo de la data están físicamente separados. Las aplicaciones por capas proveen un modelo por el cual los desarrolladores pueden crear una aplicación flexible y reusable. A través de la segregación de una aplicación en varias capas, los desarrolladores adquieren la opción de modificar o añadir una capa en específico, en lugar de hacer una reelaboración completa de la aplicación.

La clave para la aplicación en capas es la gestión de dependencias. Los componentes en una capa pueden interactuar solo con compañeros en el mismo nivel o componentes de niveles más bajos. Esto ayuda a reducir las dependencias entre componentes en diferentes niveles. Hay dos enfoques generales para la colocación de capas: estrictamente estratificado y relajado (Microsoft, 2014).

Tal como define Microsoft (2014), es necesario definir una interfaz abstracta para cada componente en una capa que es llamada por componentes en un nivel superior. Las capas superiores acceden a los componentes de nivel inferior a través de las interfaces abstractas en lugar de llamar directamente a los componentes. Esto permite que la implementación de los componentes de nivel inferior cambie sin afectar los componentes de nivel superior.

2.3.1 Estructura cliente-servidor

Ortiz (2000) define esta estructura como una de dos niveles en principio y más tarde, una más compleja de tres niveles. En esta arquitectura de tres niveles, cada nivel se ocupa en una tarea en específico.

Figura 2.2 Estructura cliente-servidor. *Tomada de (Ortiz, 2000).*

- Nivel 1: En este nivel se encuentra el cliente, este contiene la presentación lógica incluyendo un control simple y una validación de entrada del usuario. Esta aplicación es también conocida como "thin client".
- Nivel 2: El nivel medio es conocido como la aplicación del servidor, el cual provee el procesamiento lógico de los negocios y el acceso a la data.
- Nivel 3: Este último nivel solo provee la data de negocios.

2.3.2 Nube

La nube es una estructura que ofrece servicios remotos a través de una red que usualmente es internet. Knorr (2018) Define a la nube como una piscina de recursos virtualizado que va desde poder en bruto de computo a funcionalidad de aplicación, que está abierto a demanda, más aun, la nube permite a sus usuarios ganar nuevas capacidades sin necesidad de invertir en hardware o en software.

Knorr (2018) plantea que los servicios de computo de la nube disponible son vastos, pero la mayoría cae en las siguientes categorías:

- **SaaS (Software como servicio):** Este tipo de computo de nube entrega aplicaciones sobre la internet a través del navegador, Las aplicaciones **SaaS** ofrecen una extensa configuración de opciones, así como también ambientes de desarrollo que permiten que los clientes codifiquen sus propias modificaciones o adiciones.
- **IaaS (Infraestructura como servicio):** A un nivel básico, las nubes proveedoras de IaaS ofrecen almacenamiento y computo de servicios sobre una base de pago-por usuario. Pero la completa colección de servicios ofrecidos por la mayoría de los proveedores es asombrosa. Base de datos altamente escalables, redes virtuales privadas, análisis de big data, herramientas de desarrollados, machine learning, monitoreo de aplicación y así muchas más.
- **PaaS (Plataforma como servicio):** PaaS provee un conjunto de servicios y flujo de trabajos que específicamente apuntan a desarrolladores quienes pueden usar herramientas compartidas, procesos y API's para acelerar el desarrollo, testeo y despliegue de las aplicaciones.
- **FaaS (Funciones como servicio):** FaaS, la versión de nube de un cómputo sin servidor que añade otra capa de abstracción a PaaS para que los desarrolladores estén completamente aislados de todo en la pila debajo de sus códigos. En lugar de usar servidores virtuales, contenedores y aplicaciones de tiempo de ejecución, ellos suben poco a poco bloques funcionales de código y los ponen de forma que sean disparados por un cierto evento.

2.4 Interfaz de programa de aplicación

La interfaz programa de aplicación es un código que permite la comunicación entre componentes de software. Se trata de un protocolo establecido conformado por un conjunto de subrutinas, funciones y procedimientos, las cuales son llamadas que se hacen de ambos lados de donde se encuentra el api para luego así, hacer las transformaciones necesarias para que llegue a la siguiente capa. Estas llamadas representan un método para extraer datos entre capas.

Cuando se hace un aproximamiento al enlazar lenguajes resulta que en realidad hay dos problemas relacionados al describir los enlaces de idiomas a las interfaces API estándar. Uno de los problemas se relaciona con la asignación de características y capacidades de lenguaje específicas a la interfaz, y el otro problema se relaciona con la forma en que se documenta el enlace del idioma como estándar. La mayoría de las API se describen en términos de un enlace de lenguaje específico. Esto se debe al hecho histórico de que la mayoría de los estándares de API se derivan de la práctica existente que evolucionó en un lenguaje de programación específico. Algunas API están escritas para ser 'independientes del lenguaje de programación', y utilizan un metalenguaje o un lenguaje de descripción formal para especificar la interfaz (Emery, 1996).

Existen varios tipos de estilo de arquitectura de API's, sin embargo, en este trabajo se enfocará en el estilo **REST** el cual proporciona interoperabilidad entre los sistemas informáticos en internet. Los servicios web denominados **RESTful** permiten a los sistemas solicitantes acceder y manipular representaciones textuales de recursos web mediante el uso de un conjunto uniforme y predefinido de operaciones sin estado (stateless en inglés) (fielding, 2000).

2.4.1 REST API

REST define un conjunto de principios arquitectónicos mediante los cuales puede diseñar servicios web que se centran en los recursos de un sistema, incluida la forma en que los estados de los recursos se dirigen y transfieren a través de HTTP a través de una amplia gama de clientes escritos en diferentes idiomas.

Si se mide por la cantidad de servicios web que lo utilizan, REST ha surgido solo en los últimos años como un modelo de diseño de servicios web predominante (Rodríguez, 2018).

El Estado de transferencia representacional (REST) es un estilo de abstracción de unos elementos de arquitectura dentro de un sistema de hipermedia distribuido. El REST ignora los detalles de implementación de componentes y sintaxis de protocolo por el motivo de enfocarse en los roles de componentes, las restricciones sobre su interacción con otros componentes y la interpretación de significado de elementos de datos (fielding, 2000).

La naturaleza y el estado de los elementos de datos de una arquitectura es un aspecto clave de REST. La razón de este diseño se puede ver en la naturaleza de hipermedia distribuida. Cuando se selecciona un enlace, la información debe moverse desde la ubicación donde se almacena a la ubicación donde será utilizado, en la mayoría de los casos, por un lector humano. Esto es diferente a muchos otros paradigmas de procesamiento distribuido, donde es posible, y generalmente más eficiente, mover el "agente de procesamiento" (por ejemplo, código móvil, procedimiento almacenado, expresión de búsqueda, etc.) a los datos en lugar de que mueva los datos al procesador (fielding, 2000).

2.5 Tecnologías

En Esta sección se hablará sobre las posibles tecnologías candidatas que abarcaran el desarrollo de la REST API y la presentación lógica para la arquitectura de software multinivel que se empleara en el BrainCEMISID, analizando brevemente sus características para luego hacer una elección correcta que ayude y facilite no solo la construcción del software sino el desempeño en su ejecución.

2.5.1 Python

Python es un lenguaje de programación creado por Guido van Rossum (2009) en el año 1991, el desarrollo de Python ocurre en el tiempo cuando muchos lenguajes de programación dinámicos como Tcl, Perl y ruby fueron también siendo activamente desarrollados y ganando popularidad. este lenguaje es fuertemente tipado y presenta múltiples paradigmas, soportando programación orientada a objetos, programación imperativa y programación funcional. Otra de las características de Python es que puede incluirse en aplicaciones que necesitan una interfaz programable. Python presenta también la ventaja de ser un código abierto, esto quiere decir que su modelo de desarrollo es en una comunidad abierta que se dedica a colaborar con el mejoramiento del código y la implementación de nuevos módulos.

Según la documentación, Python (2019) es un lenguaje interpretado, a diferencia de uno compilado, aunque la distinción puede ser borrosa debido a la presencia del compilador de bytecode. Esto significa que los archivos de origen se pueden ejecutar directamente sin crear explícitamente un ejecutable que luego se ejecuta. Los lenguajes interpretados generalmente tienen un ciclo de desarrollo y depuración más corta que los compilados, aunque sus programas generalmente también se ejecutan más lentamente.

El código fuente de Python se compila en el código de bytes, la representación interna de un programa de Python en el intérprete de CPython. El bytecode también se almacena en caché en archivos ".pyc", por lo que la ejecución del mismo archivo es más rápida la segunda vez (se puede evitar la recompilación desde el código fuente hasta el bytecode). Se dice que este "lenguaje intermedio" se ejecuta en una máquina virtual que ejecuta el código de la máquina correspondiente a cada código de bytes. Tenga en cuenta que no se espera que los códigos de bytes funcionen entre diferentes máquinas virtuales de Python, ni que sean estables entre las versiones de Python.

2.5.2 Flask

Flask (2019) es un microframework escrito en el lenguaje de programación Python creado por Armin Ronacher en el año 2010. Es una herramienta versátil que permite el desarrollo de varias características de una aplicación como si esta estuviese basada en el por completo, una de estas características que se hará hincapié es en "RESTful Request dispatching", que es una extensión que permite la implementación de soporte para rápidamente construir REST API's, esta, será la encargada para la comunicación entre las capas. Algunos de los componentes más resaltantes de este marco de trabajo son:

- **Werkzeug (2019):** es una completa biblioteca de aplicaciones web WSGI. Comenzó como una simple colección de varias utilidades para aplicaciones WSGI y se ha convertido en una de las bibliotecas de utilidades WSGI más avanzadas. Flask envuelve a Werkzeug, usándolo para manejar los detalles de WSGI mientras proporciona más estructura y patrones para definir aplicaciones poderosas.
- **Jinja2 (2019):** es un lenguaje de plantillas moderno y fácil de usar para Python, inspirado en las plantillas de Django. Es rápido, ampliamente utilizado y seguro con el entorno de ejecución de la plantilla de espacio aislado opcional.

2.5.3 PostgreSQL

PostgreSQL es un potente sistema de base de datos relacional de objetos de código abierto que usa y amplía el lenguaje SQL combinado con muchas características que almacenan y escalan de manera segura las cargas de trabajo de datos más complicadas. Los orígenes de PostgreSQL se remontan a 1986 como parte del proyecto POSTGRES en la Universidad de California en Berkeley y tiene más de 30 años de desarrollo activo en la plataforma central. PostgreSQL se ha ganado una sólida reputación por su arquitectura comprobada, confiabilidad, integridad de datos, conjunto de características robustas, extensibilidad y la dedicación de la comunidad de código abierto detrás del software para ofrecer soluciones innovadoras y de alto rendimiento. PostgreSQL se ejecuta en todos los principales sistemas operativos, cumple con ACID desde 2001 y tiene complementos potentes como el extensor de base de datos geoespaciales PostGIS (PostgreSQL, 2020).

2.5.4 Mongo

MongoDB es un sistema de gestión de bases de datos de código abierto (SGBD) que utiliza un modelo de base de datos orientado a documentos que admite diversas formas de datos. Es una de las numerosas tecnologías de bases de datos no relacionales que surgieron a mediados de la década de 2000 bajo el lema NoSQL para su uso en aplicaciones de big data y otros trabajos de procesamiento que involucran datos que no encajan bien en un modelo relacional rígido. En lugar de usar tablas y filas como en bases de datos relacionales, la arquitectura MongoDB está compuesta de colecciones y documentos (Margaret, 2019).

2.5.5 Django

Django es un marco web de Python de alto nivel que fomenta el desarrollo rápido y el diseño limpio y pragmático. Django se desarrolló originalmente en World Online, el departamento web de un periódico en Lawrence, Kansas, EE. UU. Django usa una arquitectura de "nada compartido", lo que significa que puede agregar hardware a cualquier nivel: servidores de bases de datos, servidores de almacenamiento en caché o servidores de aplicaciones Web. El marco de trabajo separa de manera clara los componentes, como la capa de base de datos y la capa de aplicación. Y se entrega con un marco de caché simple pero potente (Django, 2019).

2.5.4 PHP

PHP (acrónimo recursivo de PHP: Hypertext Preprocessor) es un lenguaje de código abierto muy popular especialmente adecuado para el desarrollo web y que puede ser incrustado en HTML.

PHP es un lenguaje de «scripting» que puede ser embebido en HTML. Gran parte de su sintaxis se toma prestada de C, Java y Perl con un par de características específicas propias de PHP. El objetivo del lenguaje es permitir a los desarrolladores web escribir con rapidez páginas generadas dinámicamente (PHP, 2019).

2.5.5 Laravel

Laravel es un marco web basado en PHP creado en el 2011, el cual se basa en gran medida en la arquitectura MVC. Este marco de trabajo muestra ser accesible, pero potente, y proporciona las herramientas poderosas necesarias para aplicaciones grandes y robustas. Una magnífica inversión del contenedor de control, el sistema de migración expresivo y el soporte de prueba de unidad estrechamente integrado le brindan las herramientas que necesita para construir cualquier aplicación que tenga a su cargo laravel (2019).

Laravel presenta una arquitectura MVC que es un acrónimo de 'Model View Controller', esto representa la arquitectura que los desarrolladores adoptan cuando construyen aplicaciones. Con la arquitectura MVC, observamos la estructura de la aplicación con respecto a cómo funciona el flujo de datos de nuestra aplicación (ighodaro, 2018).

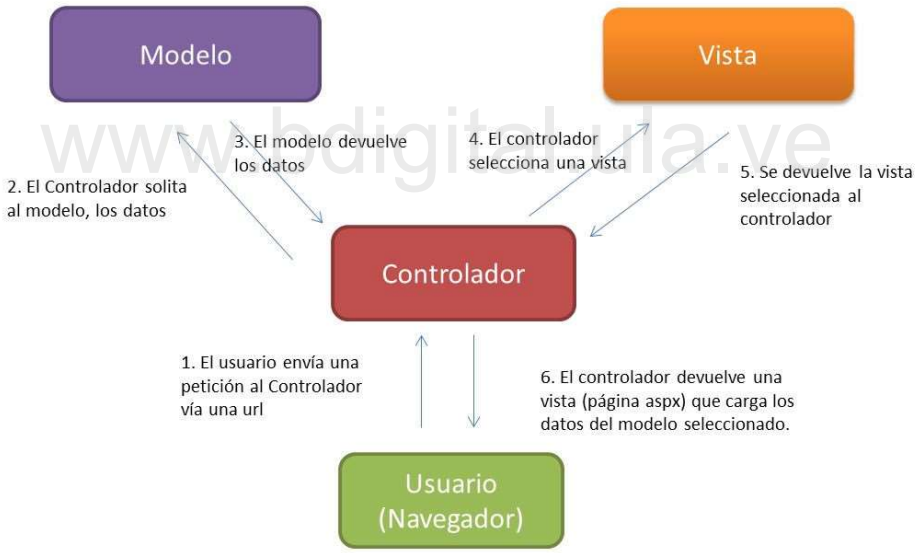

Figura 2.3 Arquitectura MVC de laravel. *Tomada de (Medium, 2016).*

Un **modelo** es una representación de una instancia u objeto de la vida real en nuestra base de código. La **Vista** representa la interfaz a través de la cual el usuario interactúa con nuestra aplicación. Cuando un usuario realiza una acción, el **Controlador** maneja la acción y actualiza el Modelo si es necesario (ighodaro, 2018).

2.5.7 Ruby

Ruby es un lenguaje de programación dinámico de código abierto que se centra en la simplicidad y la productividad. El enfoque orientado a objetos puro de Ruby se demuestra más comúnmente con un poco de código que aplica una acción a un número. En Ruby, todo es un objeto. Cada bit de información y código puede tener sus propias propiedades y acciones. La programación orientada a objetos llama a las propiedades por las variables y acciones de la instancia de nombre que se conocen como métodos (Ruby, 2019).

2.5.8 Ruby on rails

Rails es un marco de desarrollo de aplicaciones web escrito en el lenguaje de programación Ruby. Está diseñado para facilitar la programación de aplicaciones web teniendo como base el concepto de arquitectura MVC que es un acrónimo de "Modelo Vista Controlador" (Model View Controller en inglés), con algunos detalles diferentes (Rails, 2019). Sin embargo, como describe Fabela y Salas (2017):

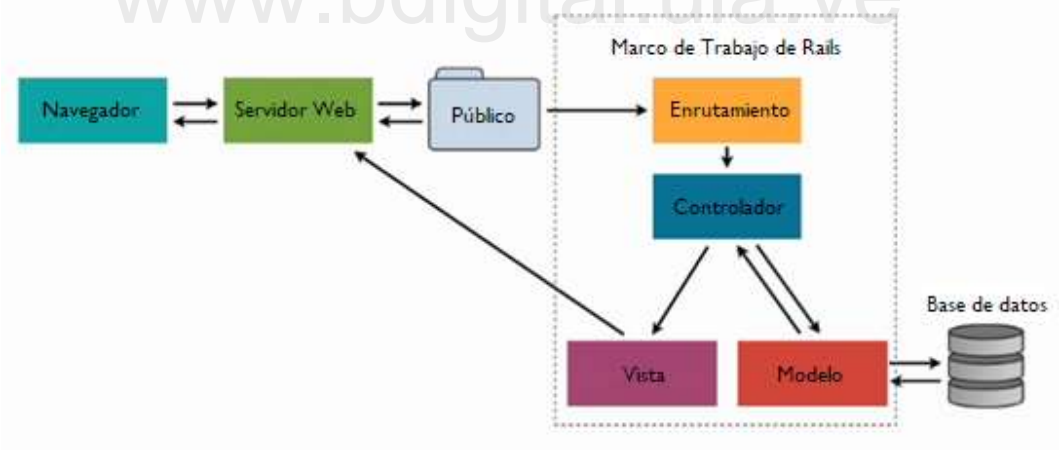

Figura 2.4 Arquitectura MVCR de ruby on rails. *Tomada de (Medium, 2017).*

- **Modelo:** Define las relaciones del dominio que luego serán migradas a la base de datos.
- **Vista:** Es la capa que será visible para el usuario final.
- **Controlador:** Define las acciones que se tomaran para generar cambios en las vistas y pide datos del dominio al modelo.
- **Enrutamiento:** Asigna la ruta URL al controlador correcto. En el controlador se puede obtener, guardar, editar, eliminar datos, y brindar una vista al usuario.

Capítulo 3

Arquitectura de software

En este capítulo se describe el diseño de la arquitectura de software que se usó para el BrainCEMISID, se expondrán las diferentes partes que compone la estructura y su funcionalidad a fondo de cada una de ellas por separado.

3.1 Vista general de la estructura

La arquitectura que se adopto para el desarrollo del BrainCEMISID es la de una arquitectura por capas, más específicamente una estructura cliente servidor, en donde podemos apreciar que el lado del cliente se encuentra en la parte del frontend del proyecto y el servidor es todo lo que comprende el backend.

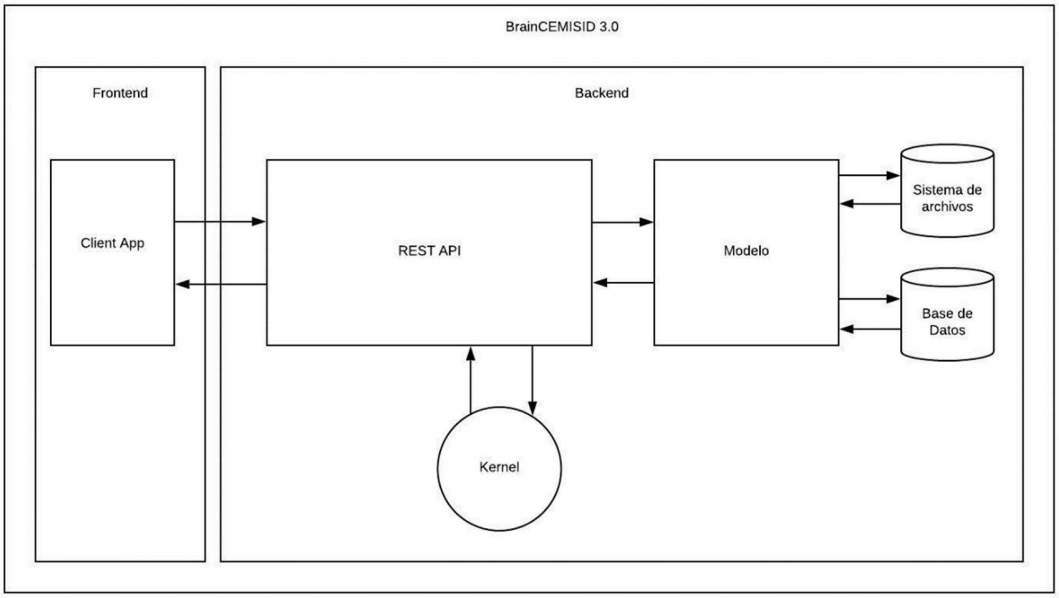

Figura 3.1 Vista general del proyecto BrainCEMISID 3.0.

3.2 Diseño del frontend

El diseño del frontend se lleva a cabo en la tesis de Vilchez (2020), una breve descripción a primera vista de este trabajo, es que este es el desarrollo de una interfaz gráfica para que el usuario interactúe con el BrainCEMISID. Esto se logra en primera instancia por medio de una progressive web app hecha en React que es una librería de javascript desarrollado por Facebook que permite la generación de vistas (paginas) dinámicas con soporte de otra librería de javascript llamada Redux que permite el manejo de endpoints de manera segura y eficiente, logrando así un puente entre React y el backend. La anterior interfaz hecha en el trabajo de Fernández (2016) proporciona un manejo muy rustico del cerebro, es por esto que el rediseño de esta interfaz con un fin más ergonómico esta también presente en el trabajo de Vilchez.

3.3 Diseño del backend

Dejando a un lado el diseño del frontend, se enfocó en el diseño del backend y su estructura final. El proceso de desarrollo del backend se realizó con la plataforma de Django en su versión 3.0.2 junto a su extensión llamada Django Rest Framework en su versión 3.11.0, que ofrece un kit de herramientas para el desarrollo de web APIs más ventajoso que la versión por defecto que ofrece el marco de trabajo. En la figura 3.2 se muestra de manera general, pero más detallada en el diseño del backend del BrainCEMISID 3.0.

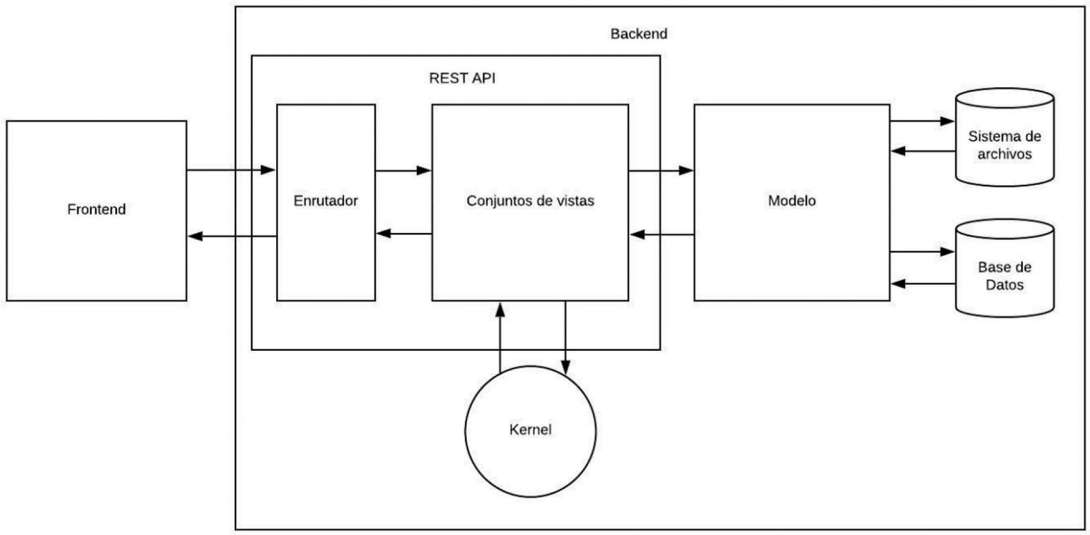

Figura 3.2 Vista detallada del diseño del backend del BrainCEMISID 3.0

3.3.1 Enrutador

El enrutador es el encargado de manejar el despacho de datos del conjunto de vistas hacia la parte del cliente y viceversa, de esta manera, el cliente (frontend) puede pedir varias solicitudes y ser atendidas a través de las terminales (endpoints) establecidas. Las diferentes rutas pueden ser agrupadas por su aplicación destinada:

Brain: Es el grupo de rutas en donde se aloja todo lo relacionado con el gestiona miento del cerebro, esta ruta permite crear, destruir, procesar conocimiento y ver las respuestas que el cerebro puede darle al usuario. En esta ruta se encontrar las siguientes rutas:

- **api/kernel:** Esta ruta es la que se utiliza para accionar las funciones del cerebro, en ella se pide la solicitud de carga del cerebro, y se le envía un protocolo de tipo BBCC que es el protocolo descrito por Fernández (2016). Además de esto, se puede pedir la solicitud de la creación y la eliminación del proyecto.
- **api/user_projects:** Esta ruta permite ver un prevista de algunos datos básicos del cerebro, entre ellos se incluye, el nombre del cerebro, el estado interno del cerebro y su estado deseado.

Images_collection: En este grupo de rutas se pueden obtener las colecciones de los usuarios, es decir, todo lo referente al banco de memoria de imágenes que un usuario y/o los usuarios puedan tener, ahora bien, estas imágenes no deben confundirse con las imágenes que el cerebro ya tiene aprendida, porque deberán ser aprendidas realizando solicitudes desde la anterior aplicación ya antes mencionada.

- **api/user_collection:** En esta ruta se transmiten datos referentes a la colección que el usuario tenga, aquí también, se puede borrar y eliminar todas y cada una de las imágenes almacenadas.
- **api/all_collections:** En esta ruta se transmiten los datos referentes a la colección de todos los usuarios, haciendo así posible que otros usuarios puedan ver imágenes de otras personas.

Accounts: En este grupo de rutas se gestiona todo lo referente a la cuenta del usuario BrainCEMISID:

- **api/auth/register:** Esta ruta es la que se utiliza para el registro de usuario.
- **api/auth/login:** Esta ruta permite iniciar sesión al usuario.
- **api/auth/user:** Esta ruta permite obtener información con respecto al usuario.
- **api/auth/logout:** Esta ruta permite al usuario salirse de su sesión.

Sight_network: Este es un grupo de rutas que abarca solo la siguiente ruta:

- **api/sight_net:** Esta ruta es para la obtención de neuronas de vista de un proyecto específico a través de la base de datos.

Hearing_network: Este es un grupo de rutas que abarca solo la siguiente ruta:

- **api/hearing_net:** Esta ruta es para la obtención de neuronas de oído de un proyecto específico a través de la base de datos.

Episodic_memory: Este es un grupo de rutas que abarca solo la siguiente ruta:

- **api/episodicmemory:** Esta ruta es para la obtención de la información acerca de su memoria episódica, en donde retorna el numero de la neurona y su estado interno en el momento de aprendizaje.

Relational_network: Este es un grupo de rutas que abarca solo la siguiente ruta:

- **api/rel_net:** Esta ruta es para la obtención de las relaciones de las neuronas del oído con la vista. Esta terminal retorna los identificadores de las neuronas pareadas, el peso de su relación, el estado de conocimiento de estas neuronas y su estado de reconocimiento.

3.3.2 Conjuntos de vistas

Un conjunto de vistas (viewset en inglés) es un controlador basado en clases que proporciona métodos basados en la arquitectura REST que permiten serializar y deserializar los datos de las solicitudes que llegan por el enrutador. Cada conjunto de vistas es una clase que tiene su propio serializador (serializer en inglés) que a su vez pueden estar o no relacionado con alguno o varios de los modelos establecidos en el proyecto, pues puede haber solicitudes que no requieran directamente datos de la base de datos como, por ejemplo, los datos de salida que genera el BrainCEMISID después de un estímulo. En estos casos se generan clases apartes para esta salida y se llaman al conjunto de vistas para poner los datos dentro de esta clase y luego ser serializada. Como en la anterior sección, los conjuntos de vistas también pueden ser agrupados por su aplicación destinada:

Brain: Este conjunto de vistas permite gestionar el cerebro en su totalidad, es, en primera instancia, donde se desarrolla mayor parte del api para las funciones principales del cerebro. Aparte de ofrecer la funcionalidad de poder crearlo y eliminarlo, tiene soporte para el protocolo BBCC que establece una serie de reglas para que el cerebro pueda interactuar con la otra interfaz del lado del cliente. Además de esto

ofrece un conjunto de vistas para ver el resumen de los otros proyectos del usuario que están disponibles y otro conjunto de vistas el estado deseado.

- **KernelViewSet:** El conjunto de vistas del kernel es una clase que ofrece la creación, eliminación y uso del cerebro.

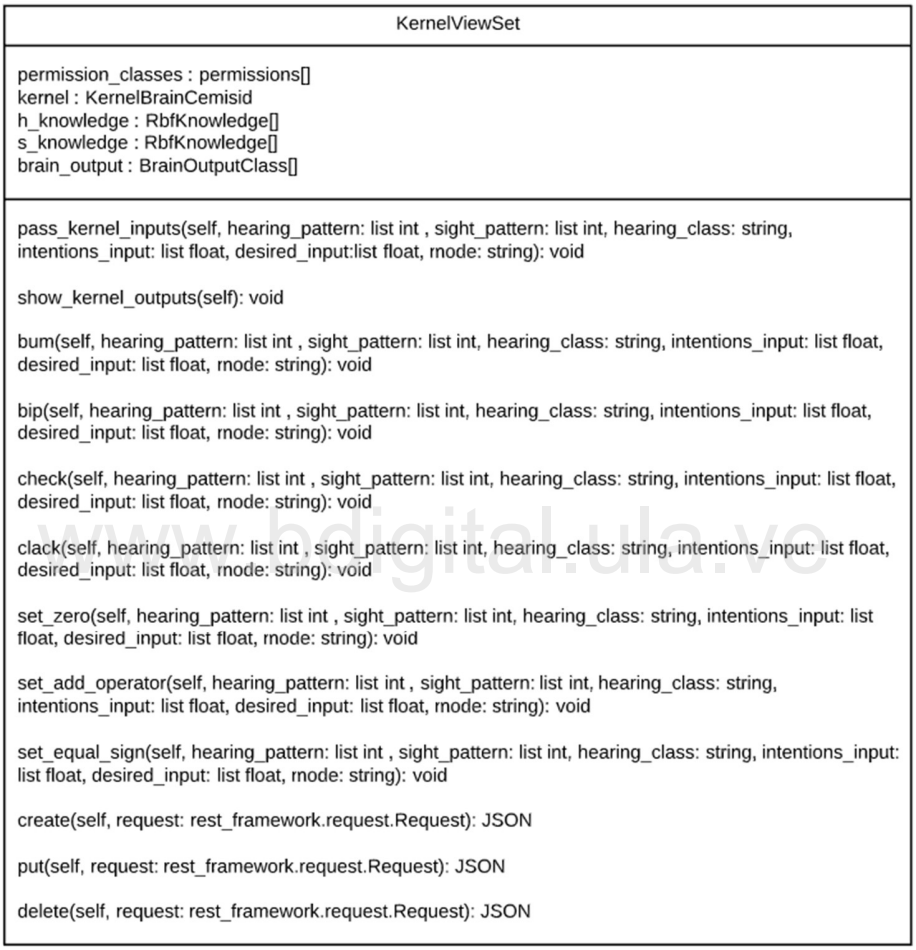

Figura 3.3 Conjunto de vistas del kernel.

- **ProjectSummaryViewSet:** Este conjunto permite acceder a la información básica de los proyectos de un usuario en específico, estos son, el nombre del proyecto, su estado interno y su estado deseado.

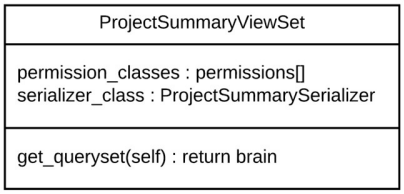

Figura 3.4 Conjunto de vistas **ProjectSummary**.

- **DesiredStateViewSet:** Este conjunto de vista permite la obtención del estado deseado del cerebro y su modificación.

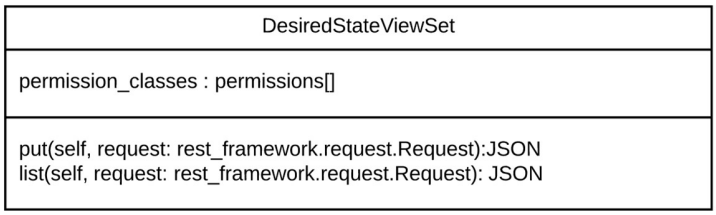

Figura 3.5 Conjunto de vistas **DesiredStateViewSet**.

Images_collections: En este grupo se definen los conjuntos de vistas para acceder a las imágenes almacenadas en el sistema de archivos.

- **UserCollectionViewSet:** Este conjunto de vista permite gestionar la colección de imágenes que el usuario puede tener, esto incluye, eliminación, agregación y hasta cierto punto modificación de los datos de la imagen.

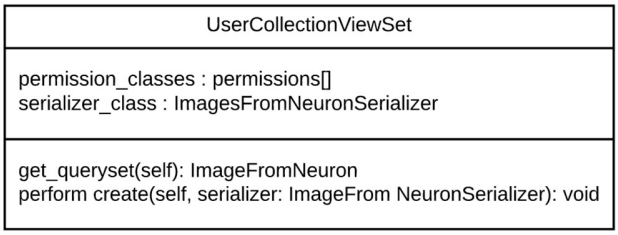

Figura 3.6 Conjunto de vistas **UserCollection**.

- **AllCollectionsViewSet:** Este conjunto de vista solo permite la vista a la colección de imágenes de todos los usuarios en formato solo lectura (Read-Only). A través de ella cualquier usuario puede ver la colección de imágenes de otros usuarios.

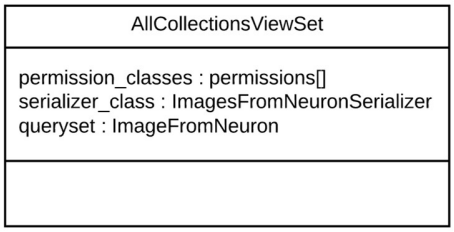

Figura 3.7 Conjunto de vistas **AllCollections**.

Accounts: Este grupo de conjuntos de vistas fueron diseñadas con el propósito de gestionar todo lo referente a las cuentas de usuarios en la plataforma, incluyendo el registro, inicio de sesión y obtención de información del usuario.

- **RegisterAPI:** Este conjunto de vistas ofrece la funcionalidad de registro del usuario en la plataforma.

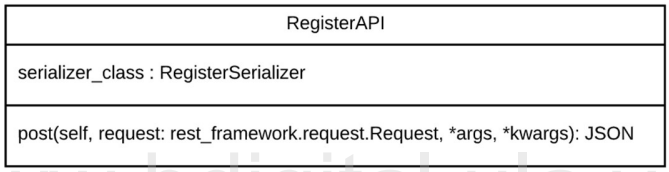

Figura 3.8 Conjunto de vistas del registro.

- **LoginAPI:** Este conjunto de vistas ofrece la funcionalidad de inicio de sesión de un usuario en la plataforma.

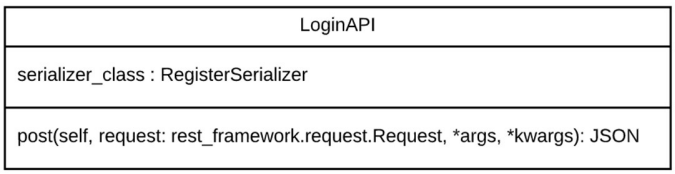

Figura 3.9 Conjunto de vistas del inicio de sesión.

- **UserAPI:** Este conjunto permite la obtención de información del usuario con inicio de sesión.

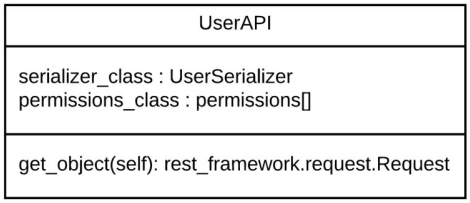

Figura 3.10 Conjunto de vistas de información del usuario.

Sight_network: Este grupo solo contiene un conjunto de vistas que se utiliza para obtener la red neuronal de la vista de un cerebro en particular. Este conjunto de vistas se llama **SightNeuronsViewSet**.

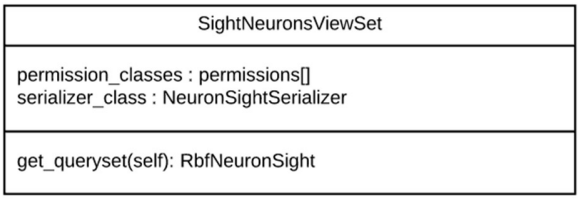

Figura 3.11 Conjunto de vistas **SightNeuronsViewSet**.

Hearing_network: Este grupo solo contiene un conjunto de vistas que se utiliza para obtener la red neuronal del oído de un cerebro en particular. Este conjunto de vistas se llama **HearingNeuronsViewSet**.

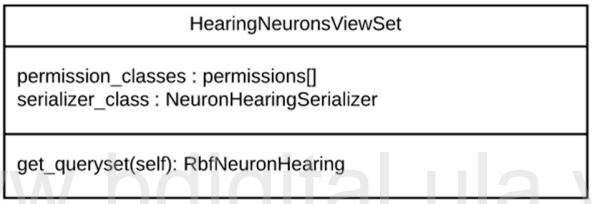

Figura 3.12 Conjunto de vistas **HearingNeuronsViewSet**.

EpisodicMemory: Este grupo solo contiene un conjunto de vistas que se utiliza para obtener la memoria episódica de un cerebro en particular. Este conjunto de vistas se llama **EpisodicMemoryViewSet**.

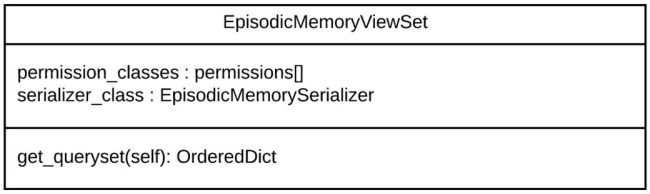

Figura 3.13 Conjunto vistas **EpisodicMemoryViewSet**.

Relational_network: Este grupo solo contiene un conjunto de vistas que se utiliza para obtener la red relacional de un cerebro en particular. Este conjunto de vistas se llama **RelNetworkViewSet**.

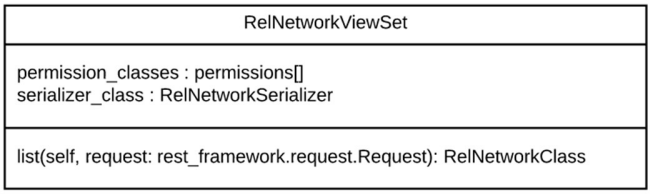

Figura 3.14 Conjunto de vistas **RelNetworkViewSet**.

3.3.3 Modelo

Como se describe en la documentación de Django (2020) un modelo es la fuente única y definitiva de información sobre los datos. Contiene los campos y comportamientos esenciales de los datos que están almacenando. En general, cada modelo se asigna a una sola tabla de base de datos.

Estos modelos ayudan a guardar los datos que necesita el kernel y los datos de los usuarios para que la plataforma pueda funcionar. Además de esto, el marco de trabajo de Django ofrece una interfaz para poder hacer consultas de manera más sencillas a la base de datos denominada mapeo objeto-relacional (Object-Relational Mapping en inglés). Esto facilita el desarrollo y ofrece seguridad en comparación a su contra parte, que sería el lenguaje de consultas estructurado (SQL en inglés) que la base de datos necesita para ser accedida desde la plataforma, más aun, ofrece también un soporte para la agregación de imágenes que se encuentran almacenadas en el sistema de archivos. Este modelo se puede ver gráficamente en la figura 0.2 del anexo B.

3.3.4 Base de Datos

La base de datos presenta un cambio con respecto al sistema de almacenamiento que se tenía en la anterior versión del BrainCEMISID, esto facilita almacenar múltiples instancias del cerebro de manera organizada para su carga en general y también para la selección de datos de una instancia en específico. El tipo de base de datos que se utilizó para el proyecto fue una base de datos relacional ya que presentaba una similitud con respecto a la estructura del componente modelo por la estructura de sus sistemas que funcionan con el paradigma de relaciones. La tecnología que se utilizó para el desarrollo de esta parte fue PostgreSQL en su versión 12 junto con PgAdmin que es su interfaz gráfica por defecto.

3.3.5 Sistema de archivos

Un sistema de archivos puede considerarse como un índice o base de datos que contiene la ubicación física de cada dato en el disco duro u otro dispositivo de almacenamiento. Los datos generalmente se organizan en carpetas llamadas directorios, que pueden contener otras carpetas y archivos (Fisher, 2019).

Django tiene la capacidad de poder interactuar con cualquier sistema de archivos que soporte el marco de trabajo, dando lugar a la capacidad de poder almacenar imágenes, videos, música y otros tipos archivos de carga pesada. El sistema de archivo se utilizó para guardar las imágenes que los usuarios almacenaban en la plataforma, siendo este una de las mejores formas de almacenamiento para este tipo de datos en específico.

3.3.6 Kernel

El kernel es el núcleo desarrollado por Fernandez (2016) en su tesis de grado, sim embargo, se realizaron unas mejoras para poder ser utilizado en el proyecto. Estas Modificaciones se pueden organizar de la siguiente forma:

3.3.6.1 Actualización

La versión anterior del kernel estaba desarrollada en Python 2.7, lo cual hacia incompatible el uso de este en el proyecto, pues la versión que se utilizó de Django solo puede ser ejecutado en versiones de Python 3 o superior. Sin embargo, a pesar de que se hubiese podido dejar la versión 2.7 del kernel y utilizar versiones más viejas de Django, el proyecto no solo hubiese corrido el riesgo en caer en la obsolescencia y falta de soporte por parte de los desarrolladores, sino que la versión que ofrece para Python 2 no tiene varias características que sus versiones recientes tienen que facilitan significativamente el desarrollo de la plataforma. Por esto se decidió hacer la actualización del código a Python 3.7 haciendo cambios en cada uno de los módulos y resolviendo conflictos que emergían por cambio de versión.

3.3.6.2 Cambio de gestión de almacenamiento de datos

El kernel utilizaba un método de almacenamiento por medio del sistema de archivos guardando los datos serializados en una carpeta en el disco duro sin soporte para almacenar varios proyectos. Esto se decidió pasar a la base de datos debido a la ventaja de poder organizar los datos de una manera más eficaz y poder dar un soporte para el almacenamiento de varios proyectos con un mayor manejo de los datos.

3.3.6.3 Reestructuración parcial del kernel

Al pasar los datos a la base de datos se obtuvo una ventaja significativa para su gestión, sin embargo, el proyecto requirió la carga externa de algunos módulos del cerebro para su observación del lado del cliente, esto hizo que algunos módulos se hayan tenido que reestructurar en su almacenamiento para luego ser cargados al lado del cliente de manera óptima. Los módulos que fueron reestructurados fueron el ***sensory_neural_block*** que es el encargado de procesar las redes neuronales de la vista y el oído, y el ***internal_state*** que es el módulo encargado de procesar el estado interno y el estado deseado.

Capítulo 4

Pruebas

En este capítulo se presentan las pruebas realizadas para los conjuntos de vistas del BrainCEMISID 3.0, cada conjunto de vista de cada módulo que conforma la plataforma será puesto a prueba con el fin de seguir el flujo de eventos y verificar que las funcionalidades que poseen las vistas estén operando correctamente.

4.1 Planificación de las pruebas

Las pruebas del sistema son realizadas de manera manual sobre cada conjunto de vista en cada módulo de la plataforma. Vistas como pruebas funcionales, las pruebas del sistema implican la observación del conjunto de vista como una caja negra en donde la salida y entrada de un método de un conjunto de vistas, se revisa para obtener una retroalimentación de sus funcionalidades.

Para la Ejecución de las pruebas del sistema se utilizó una versión prototipo del trabajo de Vilchez (2020), la cual provee las vistas (que es como él llama a la parte visual de su aplicación) necesarias para probar los protocolos principales del API, esto no solo garantiza que si las pruebas son exitosas el api está funcionando, sino que también garantiza una integración con respecto al trabajo hecho para el lado del cliente.

4.2 Criterio de las pruebas

Las pruebas se hicieron por medio de la aplicación cliente de modo que, se hizo una captura de los datos de entrada serializados que se pasan al api y luego se hace otra captura de la respuesta serializada en crudo de lo que retorna. De esta manera la prueba consistió en que, si la funcionalidad retornaba la respuesta esperada, se tomó la evaluación como exitosa. Sin embargo, si la prueba no retorna una respuesta esperada se verificará el código del API para encontrar una posible falla, se corregirá y luego se

aplicará la prueba nuevamente. Las pruebas se darán concluidas cuando todas las funcionalidades que están siendo usadas en el cliente demuestren un comportamiento esperado sin ningún tipo de error.

4.3 Pruebas realizadas

4.3.1 Registro del usuario

4.3.1.1 flujo de la prueba

Se dirige a la página principal del cliente del BrainCEMISID y se selecciona la opción registro, Se llena la planilla de registro con la información pedida mostrado en la figura 4.1. Esto contiene los campos: Usuario, correo electrónico, contraseña y confirmar contraseña, luego de esto se selecciona la opción acepto términos y condiciones y se da al botón "regístrate". El enrutador debe recibir esta información y luego debe pasarlo al método **post** del **RegisterAPI** generando una respuesta con los datos del nombre de usuario, su correo electrónico y un token de autenticación.

4.3.1.2 postcondiciones

Como se muestra en la figura 4.2 la entrada de datos al api son los valores que le provisiono el cliente de hecho en react, y luego, en la figura 4.3 se puede observar la respuesta que da el **RegisterAPI** es el de los datos del usuario más un token para validar y permitir el inicio de sesión del usuario en el lado del cliente una vez que el este registrado en la plataforma. Podemos evaluar entonces esta prueba como exitosa.

Figura 4.1 Prueba de registro de usuario.

Figura 4.2 Entrada para el conjunto de vistas **RegisterAPI** provisionada por el cliente.

Figura 4.3 Salida del **RegisterAPI**.

4.3.2 Inicio de sesión del usuario

4.3.2.1 flujo de la prueba

Se dirige nuevamente a la página para inicio de sesión, se rellenan los campos de usuario y contraseña como se muestra en la figura 4.4 y se le da a la opción de "iniciar sesión". El enrutador debe recibir esta información y debe pasarlo al método **post** del conjunto de vistas **LoginAPI** y este debe enviar los datos para inicio de sesión.

4.3.2.2 postcondiciones

Como se muestra en la figura 4.5 el cliente genera la entrada para el api del inicio de sesión que se compone con el nombre del usuario y su contraseña y en la figura 4.6 se observa la respuesta del api que son el nombre el correo y el id del usuario junto al token para la autorización de inicio de sesión esto significa que la prueba ha sido exitosa.

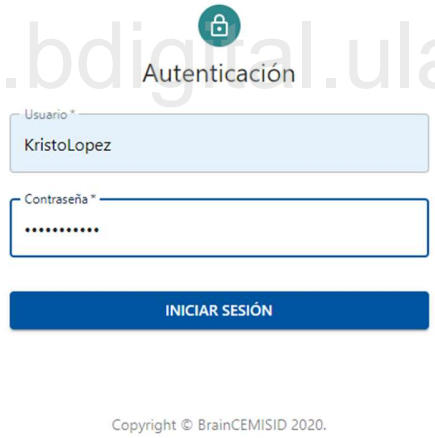

Figura 4.4 Prueba de inicio de sesión.

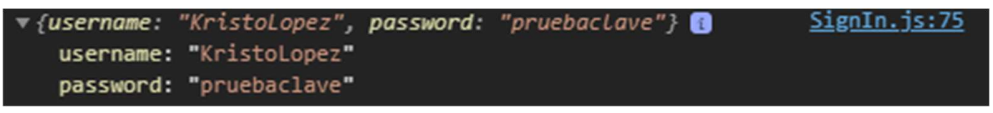

Figura 4.5 Entrada para el conjunto de vistas **LoginAPI** provisionada por el cliente.

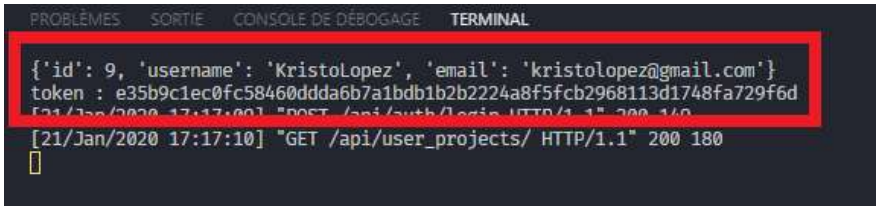

Figura 4.6 Salida del **LoginAPI**.

4.3.3 Creación de proyecto

4.3.3.1 flujo de la prueba

Con la sesión iniciada, en la página de creación de proyecto, se introduce el nombre del proyecto que se desee como se aprecia en la figura 4.7 y luego se da al botón de "create project". El enrutador debe recibir el nombre del proyecto y el token del usuario de la sesión iniciada y pasárselo al método **create** del **KernelViewSet**.

4.3.3.2 postcondiciones

Como se muestra en la figura 4.8 el cliente genera una entrada para el api de la creación del proyecto que sería el nombre del proyecto más el token de inicio de sesión, luego en la figura 4.9 se muestra la salida del api que es el mensaje de creación exitosa con el id del proyecto, lo que permite validar que la prueba es exitosa.

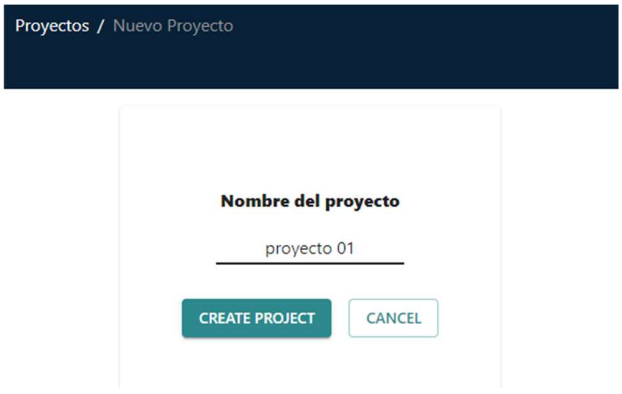

Figura 4.7 prueba de creación de proyecto.

Figura 4.8 Entrada para el método **create** del conjunto de vistas **KernelViewSet** provisionada por el cliente.

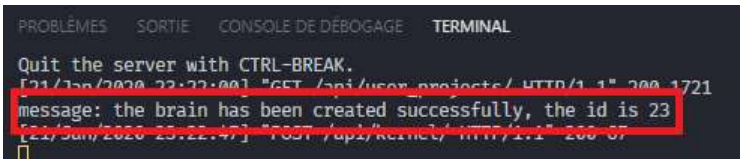

Figura 4.9 Salida del **KernelViewSet**.

4.3.4 Aprendizaje

4.3.4.1 flujo de la prueba

En el tablero del proyecto se selecciona la opción aprendizaje, posteriormente se procede a seleccionar una tarjeta de las colecciones que tenga el usuario, se introduce la categoría del oído y se ajusta la tolerancia al color como se muestra en la figura 4.10 con ajustes a los termómetros biology, cultural y feelings acorde al estado interno que se quiere generar. El enrutador debe recibir el token de autenticación del usuario más los datos del estímulo al método **put** del **KernelViewSet**.

4.3.4.2 postcondiciones

En la figura 4.11 el cliente genera una entrada para el api siendo estos el patrón de escucha, la clase del oído, el patrón de la vista, la entrada de intenciones como un vector de 3 variables biology, cultural y feelings respectivamente, el id de la imagen que se desea parear con la neurona, opción de renombramiento imagen, nombre del conjunto y modo de aprendizaje del cerebro, luego en la figura 4.12 se puede apreciar el mensaje que se pasa al cliente que la neurona ha sido pareada exitosamente junto al mensaje por defecto del protocolo http que el método **put** ha sido logrado satisfactoriamente. Este resultado se puede evaluar como exitoso.

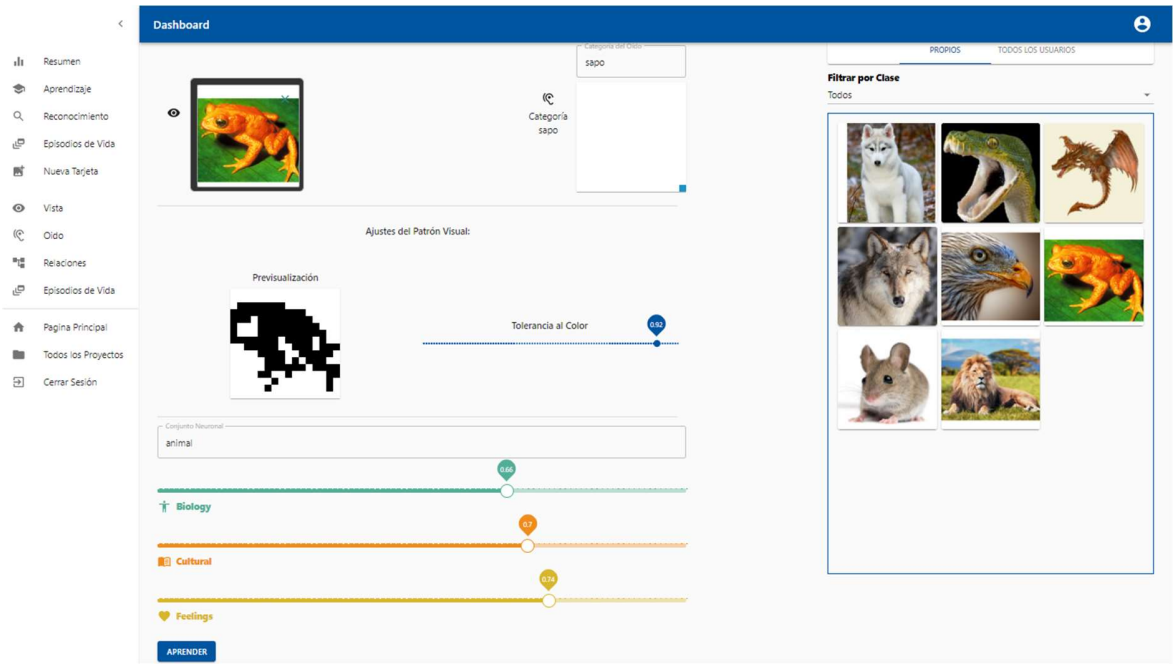

Figura 4.10 Prueba de aprendizaje.

Figura 4.11 Entrada de aprendizaje para el método **put** del conjunto de vistas **KernelViewSet** provisionada por el cliente.

Figura 4.12 Salida de aprendizaje del **KernelViewSet**.

4.3.5 Reconocimiento

4.3.5.1 flujo de la prueba

En el tablero del proyecto se selecciona la opción reconocimiento, posteriormente se procede a seleccionar una tarjeta de las colecciones que tenga el usuario, luego se ajusta el nivel de tolerancia al color como se muestra en la figura 4.13 y se da a la opción de reconocer. El enrutador debe recibir el token de autenticación del usuario más los datos del estímulo al método **put** del **KernelViewSet.**

4.3.5.2 postcondiciones

Se muestra la figura 4.14 que el cliente genera una entrada para el api siendo estos el patrón de escucha en blanco, la clase del oído en blanco, el patrón de la vista, la entrada de intenciones como un vector de 3 variables biology, cultural y feelings respectivamente y modo de reconocimiento del cerebro, luego en la figura 4.15 se puede apreciar la respuesta del kernel que es el patrón del oído, la clase del oído, el patrón de la vista, el numero neuronal, los estados internos, los estados deseados, el id de la imagen y el estado del cerebro, haciendo que este resultado es el esperado y se puede evaluar la prueba como exitosa.

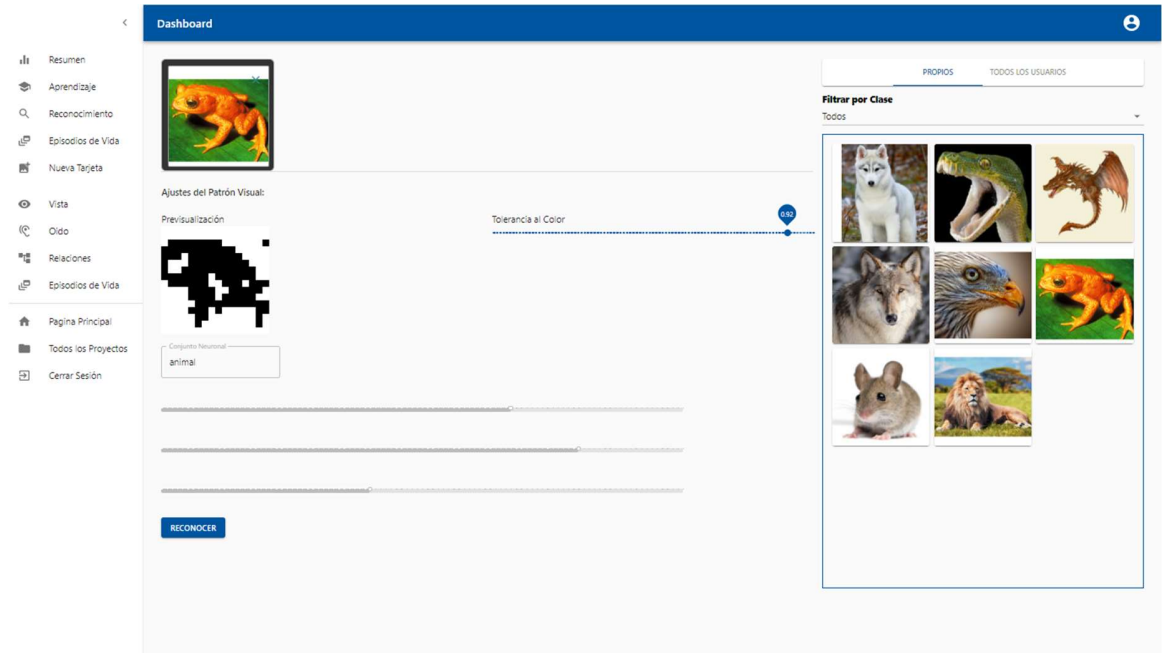

Figura 4.13 Prueba de reconocimiento.

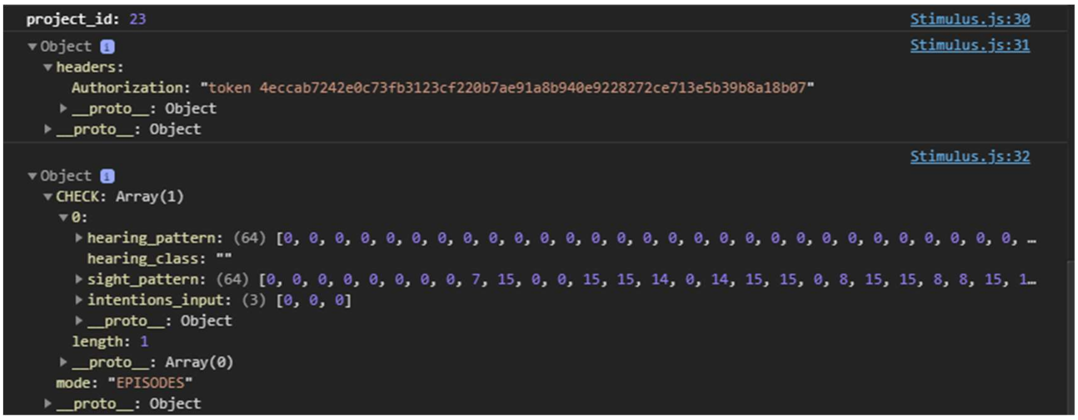

Figura 4.14 Entrada de reconocimiento para el método **put** del conjunto de vistas **KernelViewSet** provisionada por el cliente.

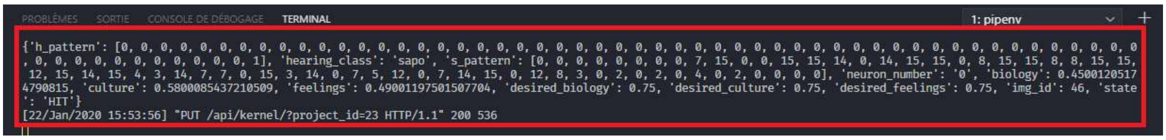

Figura 4.15 Salida de reconocimiento del **KernelViewSet**.

4.3.6 Inserción de tarjeta

4.3.6.1 flujo de la prueba

En la página de “Nueva Tarjeta” se introducen los datos solicitados “Nombre de la Nueva Tarjeta”, “Clase de la Nueva Tarjeta” y se inserta la imagen que se desea como se muestra en la figura 4.16 y luego se le da al botón “guardar”. Esto será recibido por el enrutador que deberá pasarle al método **perform_create** del **UserCollectionViewSet** el token de autenticación, el nombre de la tarjeta y el nombre de la clase de la tarjeta.

4.3.6.2 postcondiciones

Se observa que en la figura 4.17 el cliente genera una entrada para el api de colección de imágenes del usuario que serían los mismos datos que se introdujeron en la página, más el token de autenticación y la especificación del tipo de contenido que se le está pasando. Luego en la figura 4.18 se muestra nuevamente los valores que llegaron al api para remarcar la respuesta del servidor la cual es el código 201 que significa que la tarjeta ha sido creada satisfactoriamente. Este resultado indica que la prueba ha sido exitosa.

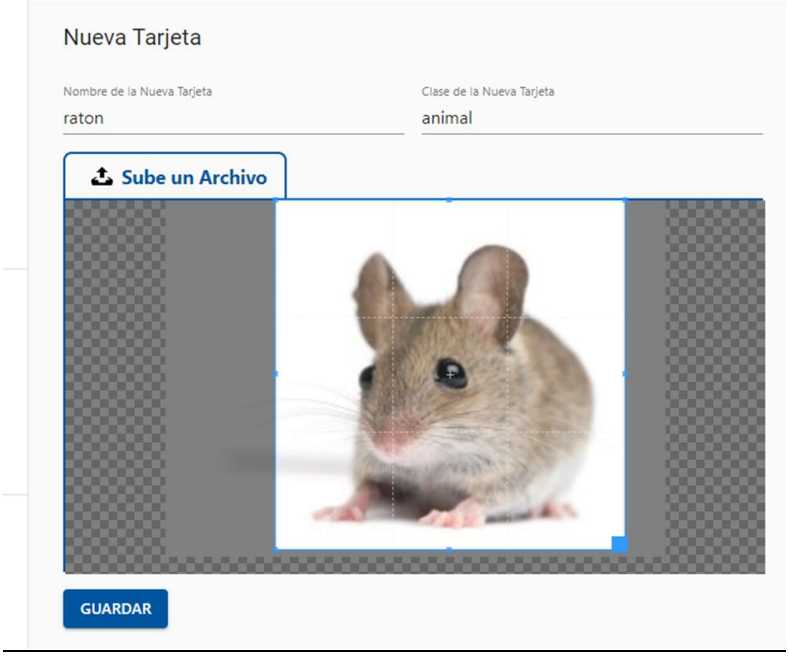

Figura 4.16 Prueba de inserción de tarjeta.

Figura 4.17 Entrada para el conjunto de vistas **UserCollectionViewSet** provisionada por el cliente.

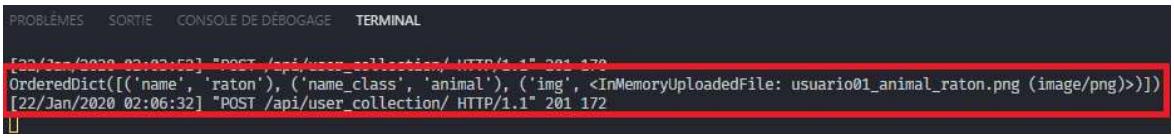

Figura 4.18 Salida del **UserCollectionViewSet** remarcada con su entrada de datos.

4.3.7 Visualización de neuronas del oído

4.3.7.1 flujo de la prueba

En el tablero del proyecto como se muestra en la figura 4.19 se selecciona a la opción oído. Esta señal genera una salida que recibe el enrutador para pasársela al método **get_queryset** del **HearingNeuronViewSet**.

4.3.7.2 postcondiciones

Se observa en la figura 4.20 que el cliente genera una entrada al api compuesta por el id del proyecto y el token de autenticación del usuario y esto a su vez genera una salida compuesta por las neuronas del oído que tienen conocimiento que son del proyecto como se ve en la figura 4.21, lo cual indica un éxito en la prueba realizada.

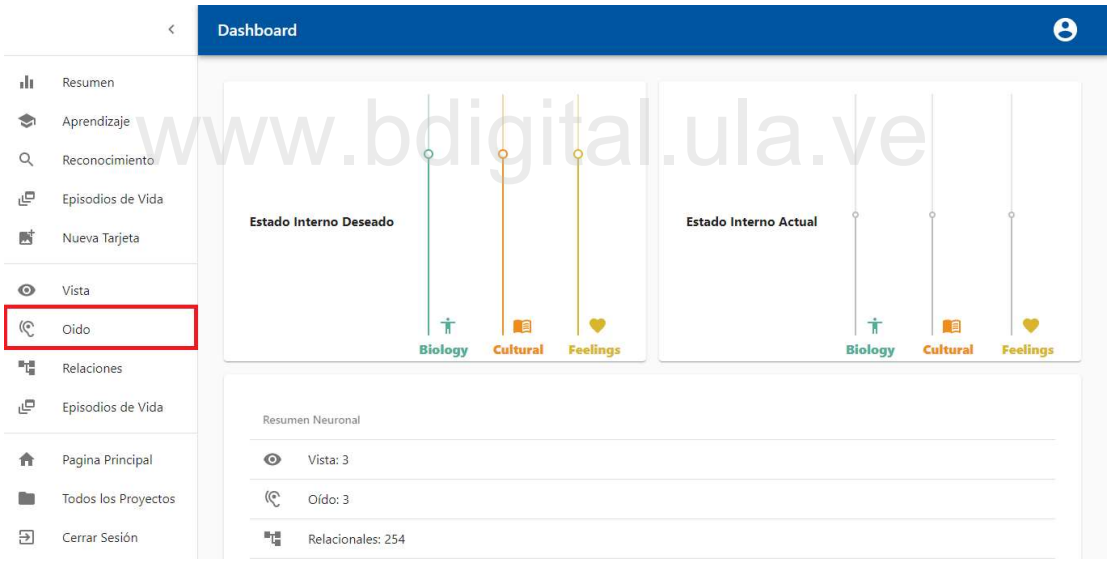

Figura 4.19 Prueba de visualización de neuronas del oído.

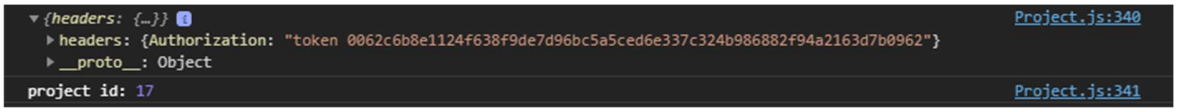

Figura 4.20 Entrada para el conjunto de vistas **HearingNeuronViewSet** provisionada por el cliente.

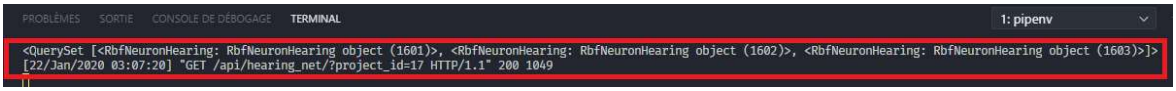

Figura 4.21 Salida del **HearingNeuronViewSet**.

4.3.8 Visualización de neuronas de la vista

4.3.8.1 flujo de la prueba

En el tablero del proyecto como se muestra en la figura 4.22 se selecciona a la opción vista. Esta señal genera una salida que recibe el enrutador para pasársela al método **get_queryset** del **SightNeuronViewSet**.

4.3.8.2 postcondiciones

Se observa en la figura 4.23 que el cliente genera una entrada al api compuesta por el id del proyecto y el token de autenticación del usuario y esto a su vez genera una salida compuesta por las neuronas de la vista que tienen conocimiento que son del proyecto como se ve en la figura 4.24, lo cual indica un éxito en la prueba realizada.

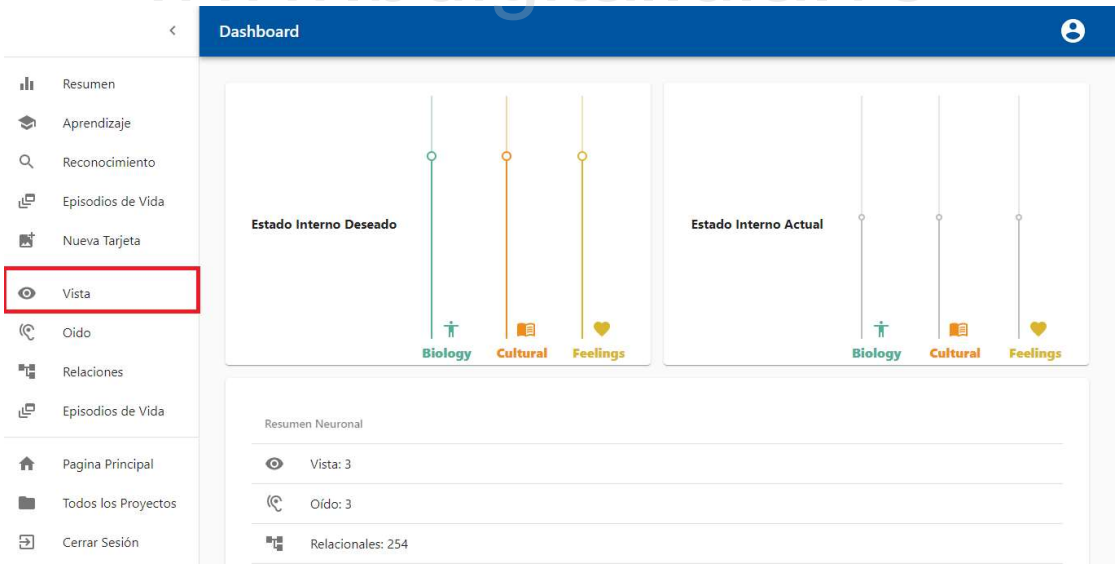

Figura 4.22 Prueba de visualización de neuronas de la vista.

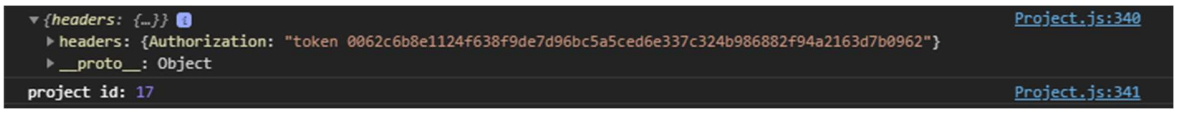

Figura 4.23 Entrada para el conjunto de vistas **SightNeuronViewSet** provisionada por el cliente.

Figura 4.24 Salida del **SightNeuronViewSet**.

Capítulo 5

Conclusiones y trabajos futuros

5.1 Conclusiones

Los proyectos que involucran al BrainCEMISID han presentado ser un desafío que presenta un paradigma que esta fuera de la zona de confort de sus contribuyentes, en este proyecto se presentaron varios desafíos, el primero de ellos era la adaptación del cerebro a un modelo de arquitectura multinivel, en donde surgieron varias posibilidades de como implementarlo a la plataforma; el segundo era el diseño de la api como tal tomando en consideración solo las posibilidades que no salieran del marco de trabajo de Django o que no tomaran mucho tiempo en implementarlas asegurando que el proyecto fuese fidedigno y por último, la reestructuración parcial de algunos datos que el cerebro almacenaba que requerían esta característica para facilitar algunas funcionalidades del API, todo esto requirió un estudio del anterior proyecto de Fernández (2016) junto a su código fuente que ofrecía con la problemática que no existía un manual concreto que explicaba el funcionamiento del cerebro claramente. Al principio cada decisión que se tomaba en el proyecto era vital porque comprometía el diseño y su consistencia, sin embargo, se tomaron decisiones en base al principio filosófico de la navaja de Ockham para poder resolver todos los problemas, esto no solo facilito el desarrollo del API, sino que inclusive hizo que en ciertos aspectos fuera más confiable su adherencia para ambas partes, tanto al cerebro como a su interfaz. Una vez tomada las decisiones más generales de diseño, las siguientes decisiones no eran más que la especificación más detallada de estas, haciendo que el problema se redujera su complejidad considerablemente y aumentando la productividad del desarrollo.

La nueva versión del kernel esta actualizada a una versión de Python con soporte de la comunidad y que brinda nuevas características que pueden requerirse en el futuro, más aun, la forma en que se implementó el kernel en la plataforma hace que sea como una especie de probeta (pipe en inglés) que se inicialice y actúe solo cuando el usuario lo requiera, haciendo que su funcionamiento solo sea el de procesar datos y enviar salidas tanto para la base de datos como para el usuario.

El diseño del api y la realización del proyecto en el marco de trabajo de Django hacen que el proyecto no solo mantenga su portabilidad, sino que también lo hace más escalable, debido a que el marco da una estructura para su desarrollo sin salirse de ciertos límites además de, poder tener apoyo por medio de su documentación y de su comunidad. Mas aún se cumple uno de los logros más importantes que han sido perseguidos en esta serie de proyectos que es poder tener el BrainCEMISID sobre la web, haciendo que varias personas puedan tener acceso al proyecto para su uso y desarrollo. No obstante, este proyecto no

ha sido llevado hasta su despliegue (del inglés deployed) en una plataforma como servicio (PaaS sus siglas en inglés) por problemas de tiempo y bloqueo nacional en servidores web que ofrecen recursos gratuitos para este tipo de proyectos, sin embargo, se acotara más adelante esta característica agrupándola con otras para proponer varios trabajos futuros sobre este proyecto para que así se tome un acercamiento inicial mucho más sencillo y se invierta este tiempo en pensar más detalladamente estos problemas.

Los objetivos que se plantearon en el proyecto fueron logrados porque se hizo un estudio analizando las salidas y entradas del BrainCEMISID junto al diseño e implementación del protocolo de comunicación entre ambas capas, esto solo fue posible en base a la creatividad y la identificación de los verdaderos problemas que necesitaban ser comprendidos, de esta forma, el proyecto demuestra que los procesos algorítmicos para la resolución de problemas no son siempre la solución porque a veces se es más difícil encontrar la pregunta que detalle el problema que su misma respuesta, viéndolo de este modo, se puede describir este como un trabajo de ingeniería pura por que denota lo más valioso e importante para esta área de trabajo, estas serían, la habilidad de comprensión de un problema y la creatividad para poder resolverlo.

5.2 Trabajos futuros

Los trabajos futuros que se describen en esta sección del proyecto comprenden desde implementar un sistema de seguridad más robusto, crear un entorno para su recreación, mejorar su percepción de ánimos, hasta ampliar y mejorar sus funcionalidades haciéndolas más precisas y refinadas.

5.2.1 Implementación de un sistema de seguridad más robusto

Aunque existe un sistema de seguridad para este proyecto ya implementado, puede haber acoples a otras interfaces customizadas del lado del cliente, esto es debido a que las credenciales CORS que se usan para pedir solicitudes a estos, tienen una leve seguridad que debe ser reforzada por medio del cambio de credenciales a tipo CSP para garantizar completa seguridad a la hora de su ejecución, esto permitirá la restricción al lado del backend solo con aplicaciones certificadas con acceso a estas credenciales.

5.2.2 Creación de un entorno contenerizado para el despliegue del BrainCEMISID

El despliegue de la aplicación es necesario para poder aprovechar un procesamiento por medio de la nube que elevaría el rendimiento y ejecución de la aplicación considerablemente, más aún puede ser una muy buena forma de poder aprovechar las tecnologías más recientes como Docker que usan el paradigma de la contenarización (containerization en inglés) de aplicaciones para así aprovechar al máximo los recursos de la nube en donde se esté ejecutando.

5.2.3 Fragmentación del BrainCEMISID dentro del conjunto de vistas para su optimización

A pesar de ser un requisito no funcional, el rendimiento y la escalabilidad son factores importantes para cualquier desarrollo de software en mayor o menor medida dependiendo de los casos, sin embargo, el BrainCEMISID puede aprovechar las ventajas que le ofrece la arquitectura multinivel en la que ya está instanciado y sufrir una fragmentación parcial o total de su kernel, dispersándolas así en el conjunto de vistas que conforman su API. Esto pudiese ser parte de un proyecto que afronte mayores retos, pero a su vez obtengan un soporte sustancialmente mejor a la hora de tener que actualizar un módulo o inclusive ayudar a mejorar su vista general y así facilitar la comprensión de su funcionamiento como un sistema.

5.2.4 Implementación del módulo de predicción musical y paralelización de funciones

El módulo creado anteriormente por Araujo (2019) proporciona una funcionalidad importante para la ampliación de la percepción que tiene el BrainCEMISID hasta ahora, este trabajo se puede añadir al kernel para que se pueda aprovechar esta característica que otorga un grado apreciación de la música e inclusive, en segundo pensamiento, extrapolar y adaptar este modelo a otros sentidos como lo explica el en su trabajo. A su vez las mejoras hechas por El Halabi (2016) podrían ser necesarias para la optimización proyecto haciendo paralelo el trabajo de algunos módulos para aprovechar un procesador con arquitectura multinúcleos.

5.2.5 Creación de un entorno de vida para el BrainCEMISID e implementación de un nuevo modelo para generar cambios de su estado interno

La autonomía de este cerebro artificial se denota como uno de los fines que busca saciar este proyecto, por esto, es necesario el planteamiento de la creación de un entorno de vida en donde el cerebro pueda descubrir lugares, aprender cosas nuevas y vivir episodios de vida. Esto a su vez, viene acompañado por un cambio necesario en el modelo actual que el BrainCEMISID emplea para hacer cambios en su estado interno. Acorde a sus necesidades y el deseo de poder satisfacerlas, se diseñaría entonces un modelo que en primera instancia se tomaría como un conjunto de histéresis las cuales, al haber un cambio, crean formas más complejas de cambios en su estado interno.

Bibliografía

Araujo A. (2019). Estructura neuronal de Predicción musical para el proyecto BrainCEMISID. Proyecto de Grado EISULA, Universidad de los Andes, Escuela de Ingenieria de Sistemas.

Andrade, L. L. (2012). Diseño en VLSI (Very Large Scale Integration) y FPGA (Field Programmable Gate Array) de una Red Neuronal. Tesis, Universidad de Los Andes, Mérida, Venezuela

Bruzual, R. A. (2015). Brain-CEMISID v1.4 - Diseño de un núcleo operacional para la construcción de un kernel-cerebro artificial implementado en programación paralela en el ambiente CUDA: Creación de la Lectura Comprensiva de Palabras. Tesis, Universidad de Los Andes, Mérida, Venezuela.

De Garis, H., Shuo, C., Goertzel, B., & Ruiting, L. (2010). A world survey of artificial brain projects, Part I: Large-scale brain simulations. Neurocomputing, 74(1-3), 3–29. En Internet, página web: https://www.sciencedirect.com/science/article/pii/S0925231210003279

El Halabi M. (2016). BRAIN-CEMISID versión 2.1 - Construcción de un Kernel Cerebro Artificial en Python: Creación de un soporte intermedio para el procesamiento en paralelo. Tesis, Universidad de Los Andes, Mérida, Venezuela.

Emery D. (diciembre 20, 1996). Standards, APIs, Interfaces and Bindings. En Internet, página web:https://web.archive.org/web/20150722003250/http://www.acm.org/tsc/apis.html

Graterol, R. X. (2015). BRAIN-CEMISID V J&J. Ver R - Diseño de un núcleo operacional para la construcción de un kernel-cerebro artificial implementado en programación paralela en el ambiente CUDA: Creación de la esfera de capacidades. Tesis, Universidad de Los Andes, Mérida, Venezuela.

Knorr E. (octubre 2, 2018). What is cloud computing? Everything you need to know now. En Internet, página web: https://www.infoworld.com/article/2683784/what-is-cloud-computing.html

Kriesel, D. (2005). A brief introduction to neural networks. En Internet, página web: http://www.dkriesel.com/en/science/neural_networks.

Fernández B. (2016). BRAIN-CEMISID versión 2.0 Construcción de un Kernel Cerebro Artificial en Python: Reingeniería de Software y Creación del Estado Mental de la Intención. Tesis, Universidad de Los Andes, Mérida, Venezuela.

Fisher T. (2019). What Exactly is a File System?. En Internet, página web: https://www.lifewire.com/what-is-a-file-system-2625880 - consultado el 15 de enero del 2020.

Fielding R. (2000). Representational State Transfer (REST). En Internet, página web: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm - consultado el 15 de enero del 2020.

Flask (2019). Flask-RESTful User's Guide. En Internet, página web: https://flask-restful.readthedocs.io/en/latest/ - consultado el 15 de junio del 2019.

Guru99 (2020). What is Spiral Model? When to Use? Advantages & Disadvantages. En internet, página web: https://www.guru99.com/what-is-spiral-model-when-to-use-advantages-disadvantages.html consultado el 7 de enero del 2020.

Ighorado N. (mayo 24, 2018). How Laravel implements MVC and how to use it effectively.En Internet, página web: https://blog.pusher.com/laravel-mvc-use/

Jinja2 (2019) Welcome to Jinja2. En Internet, página web: http://jinja.pocoo.org/docs/2.10/ - consultado el 16 de junio del 2019.

Margaret R. (2019). Guide to NoSQL databases: How they can help users meet big data needs. En internet, página web: https://searchdatamanagement.techtarget.com/definition/MongoDB - consultado el 6 de enero del 2020.

Matich D. (2001). Redes Neuronales: Conceptos Básicos y Aplicaciones. Tesis, Universidad Tecnológica Nacional, Facultad regional Rosario. Santa fe, Argentina. https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

Medium (2016). Entendiendo M de MVC y sus problemas. En internet, página web: https://medium.com/@davidenq/entendiendo-m-de-mvc-y-sus-problemas-ebc0cbf518ec - consultado el 8 de enero del 2020.

Medium (2016). Ruby on Rails: HTTP, MVC and Routes. En internet, pagina web : https://medium.com/the-renaissance-developer/ruby-on-rails-http-mvc-and-routes-f02215a46a84 - consultado el 8 de enero del 2020.

Microsoft (2014). Layered Application. En Internet, página web: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff650258(v=pandp.10) - consultado el 10 de junio del 2019.

Monsalve, J. J. (2014). BRAIN-CEMISID versión J&J - Diseño de un núcleo operacional para la construcción de un kernel básico de un cerebro artificial implementado en programación paralela en el ambiente CUDA: Creación del Estado Cerebral. Tesis, Universidad de Los Andes, Mérida, Venezuela.

Muchacho, J. C. (2015). BRAIN-CEMISID versión J&J - Diseño de un núcleo operacional para la construcción de un kernel cerebro artificial implementado en programación paralela en el ambiente CUDA: Creación del Mundo Cerebral. Tesis, Universidad de Los Andes, Mérida, Venezuela.

NetScape (2002). netscape and sun announce javascript, the open, cross-platform object scripting language for enterprise networks and the internet. En Internet, página web: https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html - consultado el 30 de junio del 2019.

PHP (2019) Manual PHP. En Internet, página web: https://www.php.net/manual/es/ - consultado el 14 de junio del 2019.

PostgreSQL (2020). About. En internet, página web: https://www.postgresql.org/about/ - consultado el 6 de enero del 2020.

Python (2019). Python 3.7.4rc2 documentation. En Internet, página web: https://docs.python.org/ - consultado el 15 de junio del 2019.

Rangel, C. R. (2012). Motor de Neuronas Programado en Paralelo Sobre el Ambiente CUDA. Tesis, Universidad de Los Andes, Mérida, Venezuela.

React (2019). Empezando. En Internet, página web: https://es.reactjs.org/docs/ - consultado el 16 de junio del 2019.

Rodríguez A. (noviembre 6, 2008). RESTful Web services: The basics. En Internet, página web: http://www.gregbulla.com/TechStuff/Docs/ws-restful-pdf.pdf

Ruby (2019). Documentation. En Internet, página web: https://www.ruby-lang.org/en/documentation/ - consultado el 20 de junio del 2019.

Russell, S. J.; Norvig, P. (2004). Inteligencia Artificial. Un enfoque moderno Segunda edición Pearson Educación, S.A., Madrid.

Salas F., Fabela C. (noviembre 23, 2017). Web App Development: Defining MVC on Ruby on Rails Framework. En Internet, página web: https://www.itexico.com/blog/web-app-development-defining-mvc-on-ruby-on-rails-framework

Shiwani S., Minal K., Chetana K. (2013). Artificial Neural Network Based Signature Recognition & Verification, IJETAE, *3*(8), 191-197. En Internet, página web: http://www.cs.sjtu.edu.cn/~shengbin/course/SE/Verification%20of%20Digital%20Signature%20Verification%20Using%20Artificial%20Neural%20Networks.pdf

Sommerville, I. (2005). Ingenieria de Software. Pearson Educación S.A, Madrid, España, 7 edition.

Sosa, J. E. (2016). BRAIN-CEMISID versión 1.5 Construcción de un Kernel Cerebro Artificial Implementado en Programación Paralela en el Ambiente CUDA: Ensamblaje de las Esferas Sensorial, Analítica, Cultural, Relacional, de Capacidades y Perceptora. Tesis, Universidad de Los Andes, Mérida, Venezuela.

Technopedia, (2020). Database (DB). En internet, página web : https://www.techopedia.com/definition/1185/database-db - consultado el 15 de enero del 2020.

Ortiz A. (julio 1, 2000). Three-Tier Architecture. En Internet, página web: https://www.linuxjournal.com/article/3508

Van Rossum G. (enero 20, 2009). A Brief Timeline of Python. En Internet, página web: http://python-history.blogspot.com/2009/01/brief-timeline-of-python.html

Vilchez P. (2019). BrainCEMISID 3.0 - Reestructuración de un kernel cerebro artificial para su adaptación a una arquitectura de software multinivel. Tesis, Sin Publicar.

Werkzeug (2019). Werkzeug. En Internet, página web: https://www.palletsprojects.com/p/werkzeug/ - consultado el 16 de junio del 2019.

Anexos

Anexos A. Repositorio del proyecto

En el siguiente enlace: https://github.com/KristoLopez/BrainCEMISID-on-Web se encuentra alojado el repositorio del proyecto: **BrainCEMISID on web** en el sitio web: https://github.com/ para próximas colaboraciones (ver Figura 0.1).

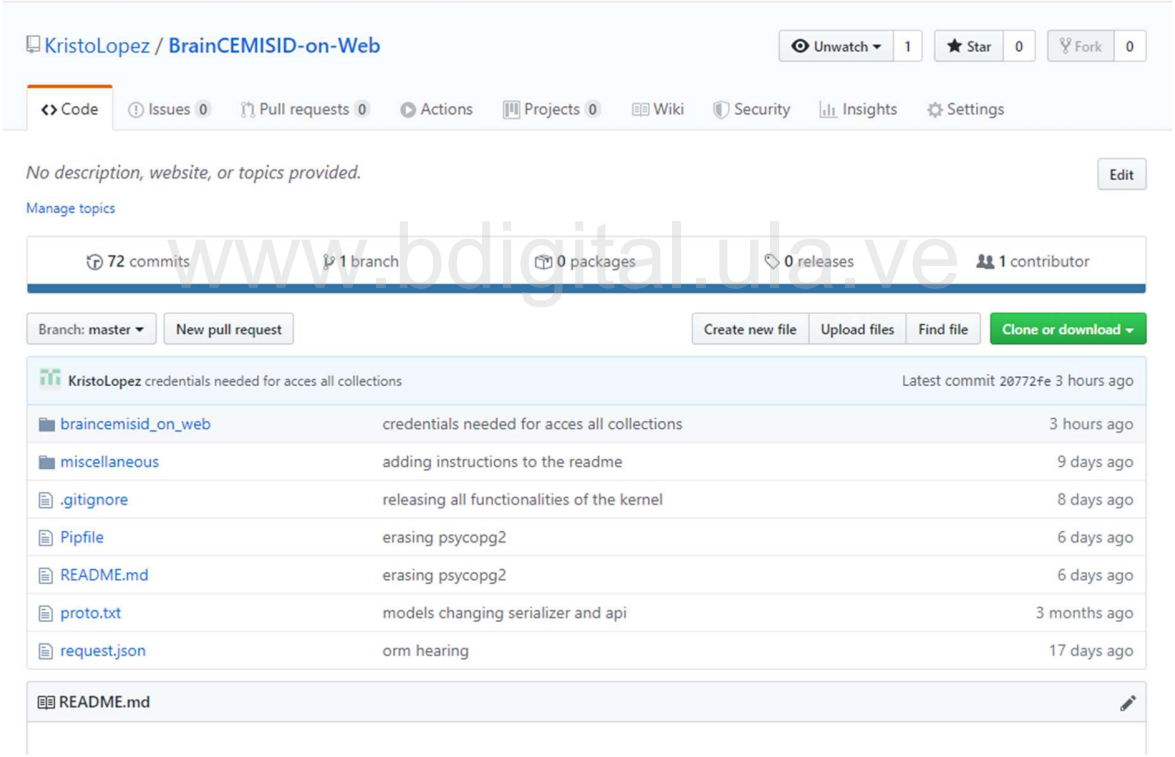

Figura 0.1 Repositorio del proyecto.

Anexos B. Modelos almacenados en la base de datos de PostrgreSQL.

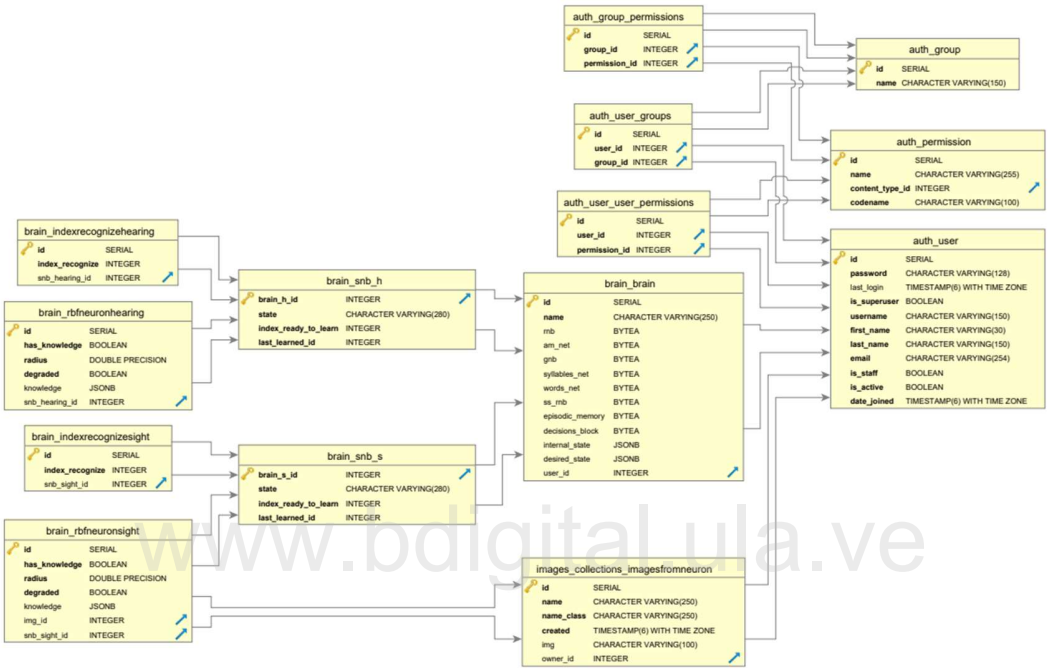

Figura 0.2 Modelos del BrainCEMISID 3.0.