Universidad de Pamplona
Departamento de Mecánica, Mecatrónica e Industrial
Ingeniería Mecatrónica

AJUSTAR TECNOLÓGICAMENTE LA APP BI GO! PARA INCORPORAR BACKEND Y REDISEÑO DEL FRONTEND

Johnathan Gianluca Sánchez Gamboa

Director de grado
CRISTHIAN IVAN RIAÑO JAIMES
Ph.D En Sistemas Mecatrónicos

Trabajo de grado para optar
Titulo de Ingeniero Mecatrónico

UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
2022

Índice

Índice de figuras

Índice de cuadros

1. Introducción

1.1. Resumen Orientado de la propuesta

Este proyecto de prácticas se realiza con la finalidad de implementar funciones de desarrollo de software y de hardware si es requerida en la empresa Cinnov S.A.S la cual es una compañía colombiana que investiga, desarrolla y fabrica tecnología para salvar vidas. Mi función corresponde principalmente en el desarrollo *Backend* y parte del *Frontend* de la aplicación Bi Go! la cual requiere la renovación de su apariencia y la creación de un *Backend* apropiado de acuerdo a la lógica de negocio que se implementará en la aplicación para así tener una arquitectura de software capaz de de manejar gran cantidad de datos, para esto se precisa usarla arquitectura MVC (Modelo-Vista-Controlador) en cuanto a la parte del *Backend* y una *Clean Architecture* en el *Frontend*.

1.2. Contexto y condiciones del entorno de desarrollo

La empresa Cinnov S.A.S requería ingenieros mecatrónicos con capacidades de adaptabilidad, investigación y de desarrollo de software principalmente para el progreso de su aplicación móvil llamada en un principio BiSafe la cual obtuvo un crecimiento considerable con la integración de una arquitectura de software y posteriormente paso a ser llamada Bi Go!.

Esta progresión se logró gracias a las capacitaciones en temas de *Flutter*, *Clean Architecture*, Azure DevOps, Git, Nodejs, Metodología Scrum y Arquitectura MVC dadas por la empresa mediante personal capacitado en cada una de las diferentes áreas del conocimientos de programación y gran capacidad para impartir conocimiento mediante un lenguaje estructurado que permitía la asimilación de información de manera efectiva ampliando mis capacidad de desenvolvimiento en el campo asignado por la empresa.

Ahora bien, estos cursos virtuales en la plataforma de Udemy y consultas en foros de problemas puntuales que se presentaban durante el desarrollo, son parte de la estructura de enseñanza definida anteriormente, dicho esto, se me permitió acceder a los conocimientos esenciales necesarias y necesarios que funcionarían como principal pilar en el desarrollo de la Aplicación móvil.

Anudado a lo anterior, en lo que respecta a mi entorno de trabajo mis primeros pasos en la empresa Cinnov S.A.S datan sus inicios como ingeniero Junior ad-Honorem junto con otros compañeros quienes nos encargamos del área de investigación y desarrollo de la empresa. Su servidor se encarga principalmente de la creación del *Backend* que utilizara la App Bi Go! con participación en el desarrollo del *Frontend*, me he tenido que enfrentar a diversas situaciones donde se han puesto a prueba los diversos conocimientos adquiridos durante mi paso por la universidad de Pamplona y dentro de las capacitaciones recibidas por la misma empresa, con esto se quiere decir, que se ha obtenido la sapiencia básica para resolverlos de una forma productiva y eficaz.

1.3. Definición del problema

La empresa Cinnov es la responsable de la creación de los dispositivos denominados Bigo y Navi los cuales están diseñados para salvaguardar las vidas de los usuarios evitando siniestros por esta razón se requiere la implementación de una nueva interfaz que sea moderna e intuitiva que pueda dar la mejor experiencia al usuario y la creación de un *Backend* mas robusto para la sedimentación de las bases de todas las funcionalidades de la aplicación Bi Go!.

1.4. Contexto y condiciones del problema

En sus inicios la aplicación para la cual estoy prestando mis servicios profesionales,no contaba con su propio *Backend*, solía utilizar servicios de un proveedor externo o de terceros tales como los servicios de Amazon, esto les permitía manejas las rutas y la información de los usuarios pero esta información no podía utilizarse de manera libre según la necesidad de la situación, dicho esto, al permitirnos desarrolla el *Backend* no solo se nos brinda la libertad del manejo de la información de los usuarios que planteen el uso de Bi Go!, sino que también le otorga cierta independencia a la aplicación, lo cual acredita con mayor veras su calidad, al tener un equipo trabajando mancomunadamente en todo momento para ella, igualmente se permite gracias al desarrollo del *Backend* la implementación del comunicación bilateral entre clientes, haciendo posible la comunicación entre usuarios conformándose poco a poco una comunidad funcional y útil, siendo este uno de los principales objetivos con los que cuenta la empresa y en los cuales nos encontramos trabajando, es decir, se cuenta con una problemática de comunicación independiente como una necesidad y al mismo tiempo es la temática a solucionar por nuestra parte y que se resolverá a lo largo de este escrito.

1.5. Objetivos

1.5.1. OBJETIVO GENERAL

Ajustar tecnológicamente la App Bi Go! Para incorporar *Backend* y rediseño del *Frontend*.

1.5.2. OBJETIVOS ESPECÍFICOS

Implementación de una arquitectura limpia que se refleja en el momento de ejecución del *Frontend* y el *Backend* de la app Bi Go!

Implementar la metodología SCRUM consistente en trabajo de equipo compartido dentro de un marco de tiempo asignado de productividad profesional.

Asentar los conocimientos concomitantes a bases de datos tales como *MongoDB*, entornos en tiempo de ejecución multiplataforma como *Node.js*, *Framework* como *Flutter* y la utilización de librerías de *JavaScript* como Socket.io.

2. Estructura del documento

En el Capitulo 1 Introducción se da una breve descripción al documento dando al lector un resumen de lo que sera desarrollado en las practicas empresariales.

En el Capitulo 2 Se da a conocer la forma de la estructuración de este documento.

En el Capitulo 3 Marco teórico y Estado del arte se presenta un estudio de varios documentos que permiten entender con mayor facilidad la creación de un *Backend*.

En el Capitulo 4 Metodología y plan trabajo se proporciona un esquema metodológico y una descripción de este mismo.

En el Capitulo 5 Modelado Funcional se tomaran en cuenta los análisis de requerimientos y casos de usos a tener para el desarrollo del *Backend* principalmente.

En el Capitulo 6 Implementación Se da a conocer el procedimiento que se llevo a cabo con respecto al Análisis que se hizo en el Modelo Funcional de la creación de este proyecto.

En el Capitulo 7 Resultados Se presenta cada uno de los resultados obtenido en las pruebas realizadas en el *Backend* y *Frontend*.

3. Estado de Arte y Marco teórico

Ante el constante aumento en el crecimiento del desarrollo referente a las aplicaciones móviles y de sitios web, se presenta la imperiosa necesidad de implementar un servidor que otorgue una base de datos que facilite la transferencia y el respectivo almacenamiento de datos de forma eficaz, oportuna y en excelentes condiciones de seguridad. En el presente existen diversos proveedores de bases de datos en la nube (Google, AmazonAws, Mongolab) con el fin de facilitar a desarrolladores *Frontend* el despliegue de sus aplicaciones sin que exista la necesidad de desgastar demasiada atención en la creación de un *Backend*, sin embargo, esta metodología no proporciona la suficiente eficacia ni la vehemencia para que los desarrolladores de pequeña escala la utilicen, dicho lo anterior, se entiende una constante en la existencia de la acuciosa necesidad de proteger la privacidad de los datos siendo que los mismos no serán almacenados en un servidor propio si no en el de un tercero.

En relación con esto los datos pueden ser muy grandes y por lo general se requiere acceso en tiempo real, en la realización de una aplicación es muy apropiado tener un servicio *Backend* apto el cual pueda interactuar con sus aplicaciones en diversas plataformas, sin embargo, pese a lo referenciado anteriormente, se presenta el principal obstáculo para los desarrolladores al momento en que deben escoger entre seleccionar y construir un *Backend* que les permita flexibilidad y una tasa alta de simultaneidad para su aplicación.

Ahora bien, una de las características más importantes en un servidor es su capacidad para manejar múltiples usuarios con una alta concurrencia de una manera eficiente y esto se logra con la selección de un lenguaje de secuencias de comandos dinámicos como lo es *JavaScript* el cual suele implementarse ampliamente en el desarrollo web.

Es por lo anterior que en el trabajo denominado "Performance Comparison and Evaluation of Web Development Technologiesin PHP, Python and *Node.js*"[1] exponen el impacto de rendimiento que tiene *Node.js*, *PHP* y *Python-Web* y para esto desarrollaron un ambiente experimental donde dividieron en primera instancia en el entorno de hardware en el cual conectaron dos maquinas una hace el papel de servidor y la otra funge en calidad de cliente como se muestra en la Figura 1, en segunda instancia se presenta la configuración del servidor, para lograr todo lo anteriormente descrito, los autores eligieron los siguientes módulos para la construcción de los entornos: *Node.js*, *PHP* y *Python*.

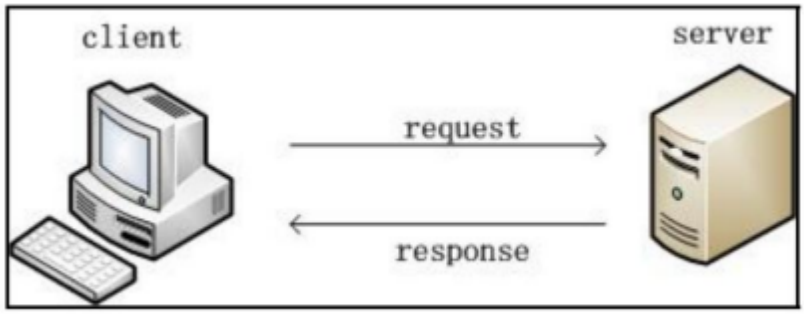

Figura 1: Configuración ambiente experimental
Fuente: Performance Comparison and Evaluation of Web Development Technologiesin PHP, Python and *Node.js* [1]

En última instancia los autores decidieron realizar la división de las pruebas en dos grandes grupos, el primer gran grupo comprenden todo lo referente a las pruebas comparativas y pruebas de escenario, para lo cual se ve la necesidad de implementar ApacheBench.

Ahora bien, las pruebas referenciadas anteriormente arrojaron resultados que permitieron inferir lo siguiente: en primer lugar se concluye que *Node.js* presenta un mayor rendimiento respecto de la técnica tradicional de *PHP* cuando se enfrenta a situaciones de alta concurrencia como se ilustra en la figura 2, evidenciando que *PHP* se desenvuelve de manera efectiva claramente satisfactoria frente al manejo de solicitudes pequeñas, sin embargo, no impide el hecho de presentar inconvenientes de resolución cuando se plantea el desenvolvimiento frente a solicitudes de gran envergadura lo que significa y nos lleva al segundo lugar en las conclusiones, se infiere que *Node.js* es una tecnología emergente que presenta grandes ventajas al confrontarlo con situaciones de E/S intensivas en la cual posee un rendimiento muy superior como se ilustra en la Figura 3, presentando obstáculos con cierta complejidad para aquellos desarrolladores que no están directamente familiarizados con la programación asíncrona y para concluir en tercer lugar, se pretende recalcar que en lo que a *PHP* incumbe este se adapta con mayor facilidad a proyectos de pequeña y mediana escala.

Conforme todo lo expuesto al interior de este trabajo investigativo, se presenta una alternativa con un manejo mas sencillo denominada Representational State Transfer reconocido por sus siglas REST, que se implementa con los proveedores de servicio Web como Google,

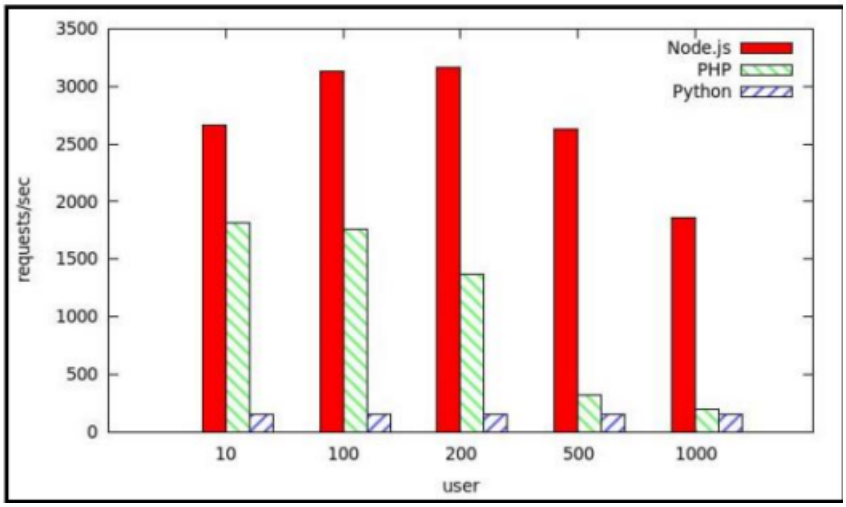

Figura 2: Solicitudes medias por segundo de operaciones con alta concurrencia
Fuente: Performance Comparison and Evaluation of Web Development Technologiesin PHP, Python and *Node.js* [1]

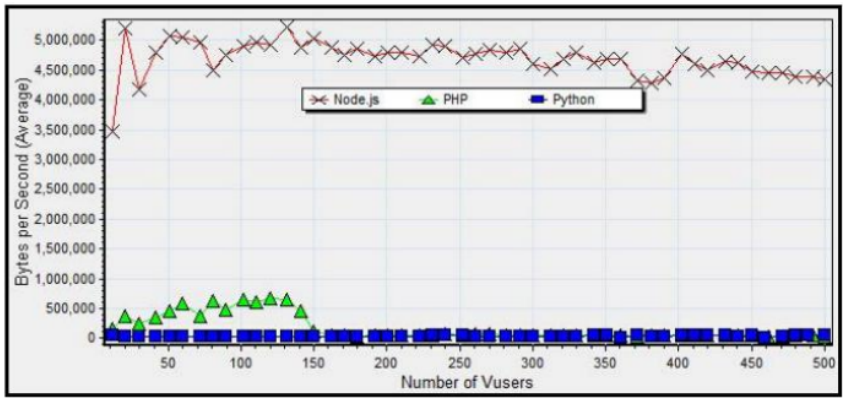

Figura 3: Rendimiento frente a situaciones de alta concurrencia
Fuente: Performance Comparison and Evaluation of Web Development Technologiesin PHP, Python and *Node.js* [1]

Yahoo y Facebook que se caracterizan por ser de mayor magnitud. Dicha alternativa permite una transmisión de datos entre el cliente y el servidos de envío y de recepción entre ambos siendo que su estructura es mas sencilla para aprender conforme los métodos implementados por HTTP. Pues bien, con el animo de generar mayor claridad en estos conceptos se pretende hacer una comparativa entre los *Framework*s *Node.js* y Sprint Web Services[2], las cuales pueden entenderse como dos tecnologías que permiten y facilitan el desarrollo óptimo de servicios Resful, lo anterior se logra por medio de pruebas sobre el funcionamiento o comportamiento de los principales métodos de http como lo son GET, POST, DELETE, PUT. esto desarrollando

servicios web que de cada una de las categorías de tecnología las cuales se comparan por sus características propias tales como robustez, integración y familiaridad con otras tecnologías, parámetros de seguridad, soporte sobre las mismas, mantenimiento óptimo, estabilidad cuando está ejecutándose y su forma de ejecución propia. La idea principal del auto, es que se generara un reporte en el que fuera posible la escogencia entre el *Framework* que mejor se adapte a las necesidades del usuario que decida implementarlo para el desarrollo de sus funciones.

Habiéndose brindado un abre bocas en lo explicado anteriormente, se da continuidad a una inducción mas profunda en el tema, con la intención de dar a comprender al lector el alcance del tema aplicado, dicho así, cuando nos enfrentamos al desarrollo de los servicios web nos encontramos con que estos se convierten en un tipo de estándar respecto la programación, al momento en que se debe compartir la información entre diferentes aplicaciones del software siendo que para realizar la distribución de funcionalidades se entienden como independientes del hardware, el sistema operativo, el lenguaje de operación son totalmente independientes respecto la función de proveer las funcionalidades de carácter independiente al momento de ser ejecutadas y enfrentarse a su normal desarrollo cuando se requiere que se pongan al servicio del usuario.

Entonces, REST debido a su arquitectura se orienta principalmente a los recursos encaminados al desarrollo de servicios web. Siendo que actualmente se ha presentado un gran aumento en el desarrollo de aplicaciones, las compañías presentan una gran complicación al tener que desarrollar la misma aplicación para diferentes plataformas, dicho de esta manera, el desarrollo de los servicios de Web REST se facilita esta función, demostrando una vez más su fácil adaptabilidad a las situaciones requeridas por los usuarios por su accesible arquitectura, debido a esto no se hace necesario que se realice una programación exclusiva para cada una de las plataformas implementadas, como tampoco se presenta una limitación por el lenguaje de programación implementado en cada una de ellas y el hardware de ejecución, lo que hace que sea únicamente suficiente exponer el servicio web para que este pueda ser consumido por parte de los clientes o de los usuarios.

El principal objetivo del autor es la creación de servicios web por medio de REST, el cual es implementado por empresas como Node. js, quienes a su vez han desarrollado *Framework*s que facilitan en gran escala el desarrollo para que sean debidamente expuestos y puedan desempeñar correctamente su función cuando requieran ser utilizados, siendo que *Node.js* se entiende como una plataforma de ejecución en la que se pueden movilizar programas como *JavaScript* por parte del servidor que al ser mezclado con Express.js definido como un *Framework* que puede funcionar sobre *Node.js* permite que se origine un enrutamiento para trabajar sobre el protocolo de HTTP, teniendo lo anteriormente expuesto la investigación realizada pretende que se logre la identificación de cual de los dos *Framework* mas competitivos puede funcionar de una manera óptima y adaptarse acorde a las diversas necesidades del usuario frente al uso de aplicaciones web.

Habiéndose puesto de presente las características definiciones además de las funcionalidades del *Node.js*, se hace menester precisar para el objeto de esta investigación el desempeño del mismo frente ala competitividad con otro *Framework* que comprende la misma envergadura en

aras de lograr una definición sobre su ejecución en actuaciones reales, todo frente al desarrollo de *Backend* frente a una aplicación web, mediante la puesta en práctica de herramientas que busquen probar como Postman que se entiende como un cliente para servicios web, con el objeto de publicar servicios se va a hacer uso del *Node.js*.

Procederemos a profundizar sobre las funcionalidades y amplitud de lo que implica Node. js, entonces, su definición abarca el entorno de ejecución encaminado a eventos asíncronos de *JavaScript*, como bien se ha mencionado anteriormente, que además se diseñó específicamente para construir aplicaciones de red escalables, en el evento en que no haya un trabajo por hace *Node.js* permanece suspendido hasta que exista una conexión para que se active una llamada al servicio, gracias a que el mismo implementa *JavaScript* se hace posible esta tarea del lado del servidor.

Para que sea posible la implementación de Node js, en la funcionalidad de una aplicación, requiere que se combine un IDE de desarrollo como *Visual Studio Code*, como el lenguaje de programación implementado por *JavaScript* dentro de lo que abarca Node js. En conclusión, Tanto *Node.js* como Spring tiene excelencia frente al desarrollo de servicios REST y los dos son fáciles de utilizar sin embargo Spring tiene madurez ya que su *Framework*s se baso en Java y esto lo hace mas robusto en canto al tema de Seguridad.

Se ha mencionado anteriormente sobre la arquitectura interna de las aplicaciones y de la comunicación entre el servidor y el cliente, con la intención de generar mayor calidad me permito traer el escrito investigativo el cual comprende como un flujo definido de diversos procesos y su correspondiente proceso como parte de un engranaje, comprendido como un despliegue óptimo en el que el cliente inicia diversas solicitudes a estos servidores quienes posteriormente se encargaran de enviar estas solicitudes y devolver las respuestas congruente con la solicitud inicial. Es decir que el cliente da inicio al proceso enviando una petición de trabajo cuando está preparado para hacer transición a un nuevo estado, el cliente en todo momento se va considerar en estado de transición, este estado en arquitecturas REST no existe, lo cual se explicará en acápites más adelante, en cada estado la aplicación contiene enlaces diversos que pueden ser implementados en la próxima ocasión en que el cliente decida iniciar nuevamente la transición por medio de otra solicitud y de esta manera obtener una respuesta acorde a lo peticionado, garantizando el éxito de la comunicación.

Conforme lo expuesto en el texto anterior, las mencionadas solicitudes y las respuestas enviadas a estas se constituyen alrededor de la transferencia de las representaciones de los recursos. Dicho esto, surge la incógnita sobre el concepto propio de recurso, pues bien, un recurso puede entender como cualquier concepto que conserve congruencia y sea significativo que pueda utilizarse dentro de la arquitectura, dicho así la representación de un recursos usualmente es un documento que abarca el estado actual o anteriormente previsto del mismo, mediante la implementación de ciertos protocolos de comunicación entre el servidor y el cliente.

Referente a los protocolos que deben persistir entre el servidor y el cliente se tiene que la informática y las telecomunicaciones deben comunicarse por medio de uno como un conjunto de reglas y normas que faciliten que entre dos o mas entidades pertenecientes a un sistema de

comunicación se relaciones efectivamente entre ellos y de esta manera se permita la transmisión de información por medio de cualquier tipo de variación de magnitud física, esto se refiere a reglas como posibles métodos de recuperación de errores, los protocolos mencionados puede ser utilizados por hardware, software o cualquier combinación entre ambos, entonces, suelen usarse en entornos web que generalmente han sido desarrollados para este fin a medida ideal para permitir la comunicación y la manipulación de recursos de internet.

Ahora bien, frente a la arquitectura de software podemos abarcarla desde el punto de vista como una esencia dentro del proceso de definición de una solución estructurada que satisface todos los requisitos y operativos, que se encargan de optimizar estos procesos comunes de mejor calidad frente al rendimiento, seguridad y capacidad de administración de datos de cada uno, por lo que los autores deciden definirla de la siguiente manera:

"La arquitectura de software abarca el conjunto de decisiones significativas acerca de la organización de un sistema de software, incluyendo la selección de los elementos estructurales y sus interfaces con las cuales el sistema se compone; el comportamiento especifica la colaboración entre los elementos; la composición de estos elementos estructurales y de comportamiento en subsistemas más grandes y un estilo arquitectónico que guía esta organización. La Arquitectura de Software también incluye la funcionalidad, facilidad de uso, flexibilidad, rendimiento, reutilización, comprensibilidad, las limitaciones económicas, la tecnología, ventajas, desventajas y las preocupaciones estéticas"[3].

Dicho de otra manera, los sistema deben ser creados teniendo en cuenta la necesidad del usuario, el sistema como la infraestructura de las Tecnologías de la Información, como los objetivos que se plantean dentro del problema propuesto para que se genere una lógica de negocio estructurada de manera satisfactoria, sin embargo, para cada una de estas áreas se deben tener en cuenta los escenarios claves y tener presentes los atributos de calidad de mayor relevancia como lo son la fiabilidad y la escalabilidad. Dicho así expondré ahora los implementados dentro del desarrollo de mis funciones.

Frente al modelo vista controlador MVC, se hace posible la generación de múltiples listas de estos mismos datos representados como un gráfico de barras para la gestión y la vista en tablas para los contadores, entonces se requiere de un controlador que hará las veces de interprete de entradas al mencionado sistema traduciendo las intenciones del usuario y a s vez convirtiéndolas en comandos para que pueda ser interpretado como una vista.

En segundo lugar, tenemos el servidor web implementado por medio de HTTP entendido como un programa informático que permite el procesamiento de la aplicación del lado del servidor realizando conexión bidireccional o unidireccional y al mismo tiempo asíncronas con el cliente o el usuario tendiente de esta forma a dar una respuesta a la solicitud radicada por este en cualquier tipo de lenguaje en favor de la comprensión del cliente. Entonces, el código que se recibe por parte del cliente usualmente es compilado y posteriormente ejecutado por un navegador web, frente a la transmisión de datos se utilizan los protocolos HTTP que pertenecen a la capa de aplicación del estándar OSI conocido como OPEN SYSTEM INTERCONECTION, el termino mencionado también se implementa para referirse al ordenador que se encuentra en

ejecución de un programa.

Al encontrarnos bajo el patrón de Modelo- Vista- Controlador conocido por su sigla MVC , se entiende que su meta es reducir el esfuerzo generando durante el proceso de programación, el cual es implícitamente necesario en la implementación de sistemas múltiples que deben estar sincronizados y ligados por los mismos datos, a partir de la estandarización del diseño de aplicaciones. El MVC se entiende como un gran paradigma encargado de dividir las partes que conformar una aplicación en el modelo, las vistas y los controladores, esto permite que se pueda trabajar a la sección por separado y permitiendo de igual manera la actualización y mantenimiento del software de una manera más llana reduciendo los intervalos de tiempo de trabajo, pero manteniendo la eficacia del mismo. Por medio del uso de *Framework*s basados en el patrón MVC se logra un mayor nivel de organización y productividad de trabajo especialmente para los desarrolladores y los programadores.

Su necesidad se hace notable en el día a día para aquellos que pertenecen al mundo de los que crean y desarrollan aplicaciones informáticas, estos mismos se permiten enfocar su atención en dos principales aspectos de trabajo, el primero es lograr una mejor construcción de aplicaciones en tiempos realmente reducidos manteniendo los mismos estándares de calidad de funcionamiento o mejorándolos incluso y el segundo es mejorar el diseño de las aplicaciones que permitiría de igual forma la reutilización del código fuente permitiendo óptimos mantenimientos en los sistemas que se encuentran ya desarrollados. Ahora bien, actualmente la programación que se orienta especialmente a la concepción del uso de aplicación ha venido aumentando, por lo que en la actualidad se tiende a utilizar en mayor medida los patrones internacionalmente aceptados.

Con el objeto de ilustrar el nacimiento de este modelo de patrón vista controlador, los autores expresan que su origen data desde el año 1979 por TrygveReenskaug el cual fue posteriormente introducido por la versión Smalltalk-80 por el lenguaje de programación Smalltalk, su objeto es la disminución del esfuerzo requerido en plataformas de sistemas múltiples y sincronizados a una misma base de datos. Se caracterizaba principalmente porque trata como entidades separadas el modelo -vista- y controlador, lo que permite que se refleje de manera automática cada una de las citas. Este modelo es un tipo de arquitectura de software que permite emplear sistemas de representación gráfica de datos donde se incluye el mismo diseño con diferentes escalas de aumento, pero en ventanas separadas[4].

Acto seguido me dispondré a expresar las diversas ventajas que se presentan en este modelo de arquitectura, la primera es que se permite la separación entre los componentes que crean un programa, facilitando su trabajo de manera separada, la segunda, es que su interfaz de programación de aplicaciones API conocido como Aplication Programming Interface permite separar el modelo, la vista o el controlador sin que medie mayor obstáculo o dificultar para ejecutar cada acción de manera individual y la tercera, permite la conexión entre el modelo y sus vistas dinamias, en el mismo tiempo de ejecución pero no en un tiempo de compilación.

Cuando el programador se plantea la incorporación del modelo de arquitectura MVC a un diseño, aprecia que las piezas de un programa puedan construirse de forma individual entre

ellas y luego que puedan unificarse como parte de un todo en tiempo de ejecución, Sin embargo, si uno de los componentes, posterior contiene algún error podrá tratarse la problemática de manera separada a los demás componentes de la aplicación sin que esto afecte directamente en las demás piezas que no se encuentren afectadas. Esto refleja directamente la aproximación monolítica de la mayoría de los programas de pequeña o mediana complejidad, siendo que todos presentan un frame que abarca todos los elementos, un controlador de eventos, un banco de cálculos y al final la presentación del producto de la unión de todo lo anterior en conjunto.

Cabe también la oportunidad de dar a conocer la definición brindada por los autores frente a este modelo de arquitectura, el primero es modelo, se entiende como el objeto que da representación a los datos que se encuentran en el programa, es el encargado de manejar estos datos y controlar al mismo tiempo todas sus transformaciones, posee un conocimiento especifico frente a los controladores o las vistas, de igual manera, es propio del sistema que tiene encomendada la responsabilidad de mantener estables los enlaces entre el modelos y sus vistas notificando a las segundas cuando y como cambia el modelo. Frente a la vista hacen referencia como el objeto que se manera en la presentación visual de los datos que contiene la aplicación por el modelo, creando una representación visual del modelo y reflejando los datos que ha incorporado el usuario, de igual forma interactúa preferentemente con el controlador sin que sea imposible que se relación de manera directa con el modelo a a través de la referencia de su propio modelo y para terminar, el tercer elemento es el controlador, que se entiende como el encargado de proporcionar el significado a las ordenes que imparte el usuario o cliente, este actúa en los datos que se encarga de representar el modelo, concentrando toda la interacción entre la vista y el modelo cuando se va a realizar un cambio, el ejecutar esta acción sean cambios de información en el modelo o modificaciones en la vista, el modelo interactúa por medio de la referencia directamente al modelo.

Puntualmente las funciones de cada uno de los elementos del MVC es, el modelo accede a la capa del banco de datos, es decir, donde se almacena toda la información y se encarga de establecer las reglas de negocio, es decir, lo relacionado con la funcionalidad del sistema, también se encarga de notificar a la vista cuando los cambios se han generado de los datos generados por un agente externo solo si encuentre frente a un modelo activo como puede ser un fichero Bath.

Respecto el controlador, principalmente se encarga de la recepción de eventos de entrada como pueden ser acciones externas a modo de clic, cambio de un campo de texto, entre otros, establece las reglas de gestión de eventos, es decir permite suponer acciones o peticiones al modelo o a las vistas, una de las peticiones a las vistas puede entenderse como una llamada al método para que este actualice o una nueva orden.

Para finalizar, las vistas, su función principal es la recepción de los datos que anteriormente ya han sido procesados por el controlador o el modelo y posteriormente generar una representación para la apreciación del usuario, manteniendo un registro de su controlador, puede dar el servicio de actualización, para que se comunicado por el controlador o por el modelo cuando el mismo se encuentra activo e informa los cambios de los datos que se han producido por parte de otros agentes.

La forma en que este sistema opera radica en primera medida en que el usuario o cliente debe interactuar con la interfaz y esta misma debe reaccionar a la intención del usuario, posteriormente el controlador recibe la información que viene de la interacción entre la interfaz y la vista, es decir, que la solicitud enviada por el usuario debe ser gestionada por el controlador en el momento en que llega, por medio de un gestor de eventos o callback, en este caso el controlador accede al modelo y lo actualiza posiblemente de la forma más adecuada a la acción que ha solicitado el usuario que refleja los cambios de este modelo. Pues bien, el modelo no puede contener conocimiento directo sobre lo que ocurre en la vista, pero el patrón de observador que ostenta el controlador puede implementarse como proveedor de cierta información entre el modelo y la vista, permitiendo a este mismo esperar los cambios para procesarlos y aun así el modelo por si mismo consigue realizar la acción sin contener información sobre la vista.

Posteriormente el controlador no pasa objeto de dominio, es decir, del modelo a la vista aunque si puede dar la orden para la actualización de la vista, sin embargo, en algunas implementaciones de la vista no se cuenta con el acceso directo al modelo dejando únicamente en cabeza del controlador la función de envío de datos del modelo a la vista. Acto seguido la interfaz del usuario reposa esperando información de las actualizaciones en las interacciones del usuario dando inicio a un nuevo ciclo como se ha descrito en párrafos anteriores.

Conforme todo lo que se ha expuesto anteriormente todos los patrones de tipo MVC cumplen a cabalidad con el objetivo particular de cualquier otro tipo de *Framework*, el cual consta de proveer una estructura de excelente definición y soporte para un proyecto web que brinde estabilidad al proyecto para que el mismo sea consistente, organizado y con un excelente desarrollo, pues bien, este modelo presenta diversas ventajas tales como la posibilidad de separación lógica y física de los diversos componentes que estructuran la aplicación, si existen diversos modelos , vistas y controladores, lo anterior nos permite que los desarrolladores de una aplicación pueda centrarse en la parte que le es asignada sea en el campo del diseño o en el campo de las vistas o por otra parte como programadores de modelos de negocio.

Los *Framework*s MVC tienen también cuentan con desventajas frente al manejo de flujos de tareas que tienen que hacerse a mano codificando flujos de manera directa. Por otro lado, el diseño del modelo con la vista y el controlador es usualmente implementado para realizar el diseño de aplicaciones que contengan o requieran interfaces complejas. La lógica de una interfaz de usuario es cambiante a comparación de los almacenes de datos que la lógica del negocio, es que quiere decir que se trata de la creación de un diseño que desacople la vista frente al modelo con el único objeto de mejorar la reusabilidad de cada una de las partes que la componen.

Es de esta manera que las modificaciones que sean realizadas en las vistas provoquen un impacto en la menor medida en la lógica de negocios o en la de datos. Entonces, se entiende que el patrón de MVC se frecuenta en aplicaciones de uso web, en la cual la vista es la pagina de HTML y el código que provee de datos dinámicos a la página, dicho de otra forma, el modelo es el sistema de gestión de bases de datos y la lógica de negocio, como el controlador representa la recepción de eventos de entrada desde la vista, por lo que se permite inferir que una de las mayores dificultades a las que se enfrentan los desarrolladores que lo implementen es tener que lidiar con la utilización del patrón de eventos con el cual el controlador podrá quedar semi-oculto

dentro de la vista.

Para concluir frente a esta estructura de la cual ha tratado este acápite, se entiende que a medida que se van incrementado las necesidades de cualquier aplicación, la modificación del código que ya existe se hace inevitable por lo que dado el caso de que no existiera una clara división de división de uso, el código se tornaría indescifrable, sino que también seria totalmente impredecible lo que se derivaría de la fusión de funciones que surgen.

En la estructura del MVC se presenta un paradigma usualmente implementado en el desarrollo de varios de los software existentes, por medio de un patrón que hace posible la división de las diversas estructuras que configuran una aplicación en su existencia, esto nos permite actualizar y realizar mantenimiento del software de una manera sencilla y en un tiempo razonablemente corto.

Entendemos que utilizar *Framework*s cuya base se el patrón MVC da una mayor facilidad en la separación de la lógica y la física de los componentes de la aplicación que permiten de igual manera un incremento en la especialización de los desarrolladores y diseñadores de aplicaciones que además de permitir la contribución en una elevada organización de trabajo facilita el mismo.

Frente a la base de datos tenemos la definición encontrada en Análisis de rendimiento entre la base de datos relacional: MySQL y una base de datos no relacional: *MongoDB* [5], en su proyecto investigativo determina que la manera mas óptima de organizar y lograr el acceso a los datos de almacenamiento en grandes bases de datos, puede lograr sin la necesidad de tener conocimiento sobre la estructura de la organización de la información a utilizar como tampoco en donde reside la base de datos, se consideró entonces como una idea revolucionaria que permitió la apertura de puertas a un mundo de independencia en los datos que a su vez originaron el lenguaje de consulta estructurado conocido como SQL conforme el autor, quien publicó 12 reglas de determinan la base de datos ideal, las cuáles vinieron implementando como mapa para el diseño de los sistemas de bases de datos relacionales.

El concepto mencionado anteriormente como base de datos relacionales hacen referencia a un conglomerado de datos y herramientas conceptuales para la descripción de los mismos, es decir, que puedan relacionarse o lograr comunicación entre ellos, se emplea con mayor frecuencia al momento de lograr una representación de los datos en nuestro mundo, de una forma intuitiva para el fácil manejo del usuario manteniendo las características propias de la información como metadatos lo cual al mismo tiempo nos facilitan las modificaciones dirimiendo las problemáticas que surgen al interior de las aplicaciones que ya han sido desarrolladas.

También se entiende que contiene instrumentos de consulta de mayor rango, los cuales funcionan independientes del RDBMS y de la organización física de los datos, entiéndase que el RDBMS es quien optimiza la información en formato estándar a las características tendientes al almacenamiento.

En complemento con lo anterior tenemos el esquema que representa a la base de datos comprendida como el diseño lógico que la conforma y se encarga de identificar y determinar la información que pretende guardarse, este hecho resalta la importancia de asignar nomenclatura a los esquemas de las relaciones de datos, lo cual permite su definir su funcionalidad y atributos

conforme sea su dominio al interior de la lista de dominio que le corresponda.

Ya se ha hecho referencia anteriormente de las bases de datos que permiten la comunicación entre si, es decir, que se pueden relacionar, también existen las bases de datos que son de carácter no relacional, comprendidas dentro de las siglas NoSQL cuyo significado es no solo SQL. Son otro tipo de almacenamiento de datos usualmente implementados cuando existen grandes cantidades de información o datos que requieren apilar, esto ocurre debido a las grandes cantidades de información provenientes de aplicaciones que funcionan bajo el uso de Web 2.0. La mencionada base de datos permite una interacción entre los sistemas operativos que funcionen con UNIX y que además ostenten un óptimo nivel de rendimiento del sistema de forma tal que escalar el horizonte sea mas sencillo, lo anterior en razón a que las bases de datos con esta característica no pretenden organizar los datos en tablas relacionales, por lo que organizar su información de forma desnormalizada.

Otra característica importante de las bases de datos mencionadas anteriormente es que son de tipo open source lo que significa que cualquier usuario tiene acceso de revisión a su código y realizar sobre el mismo todas las modificaciones que estime pertinentes conforme las intenciones de cada usuario al momento de implementarla o interactuar con ella.

Últimamente se han venido organizando diversos eventos en varias partes del globo con el objeto de informar sobre el avance de NoSQL los cuales se encuentran intrínsecamente vinculados con IoT (Internet of Things), Big Data y hasta Cloud.

Sin embargo, este modelo de base de datos contiene grandes desventajas que según el autor deben tenerse en cuenta, conforme la finalidad para la cual requiera ejercerla, la primera de estas es que ya existen demasiados sistemas de administración de datos, por lo que uno nuevo en el mercado no va a marcar diferencia alguna, el segundo es que el modelo mas utilizado es el relacional que he descrito anteriormente, por las facilidades organizacionales que este mismo promete y que a su vez las hace mas frecuentes entre los usuarios, implemente bases de datos tales como Oracle, SQL y MySQL Server Microsoft que como ya hemos descrito se traducen a sus siglas RDBMS.

En tercer lugar se entiende que estos modelos terminan proyectando déficit y problemáticas en los mismos modelos de datos respecto las limitaciones que contiene como la escalabilidad horizontal frente a múltiples servidores y monumentales cantidades de información o datos, por lo que se describen los siguientes problemas derivadas de esta situación, una es que el aumento importante presentado en el volumen de datos que son continuamente generados por los usuarios, sistemas y sensores, incrementados por la concentración de gran parte del volumen de sistemas como Amazon, Google y demás servidores comprendidos por la nube.

También debe enfrentarse al complejo crecimiento de interdependencia entre los datos que son impulsados por Web 2.0 además de redes sociales, el acceso abierto y estandarizado derivados de una amplia gama de números de sistemas.

Conforme lo anterior, se entiende que el Big Data mayormente hace referencia a la tecnología y a los negocios que implican el uso de datos de diferentes áreas que a su vez se encuentran en constante metamorfosis masivas que deben enfrentarse a las tecnologías tradicionales.

Dando continuidad a los términos mencionados anteriormente, haré mención sobre el sistema de gestión de datos SQL Open Source el cual se reconoce popularmente por ser el que se encuentra mayormente desarrollado, distribuido y soportado por Oracle Corporation, se hace pertinente mencionar sus características principales una de esas es que lo referente a SQL se considera un lenguaje estandarizado más común utilizado para el correcto ingreso a las bases de datos, que igualmente muestran dependencia a su entorno de programación, permite la inclusión del SQL de manera directa generalmente para crear informes, igualmente permite las sentencias de SQL conformadas por un código en un idioma diferente al nativo, entre otras cosas también permite la implementación de una API calificada conformada por SQL.

Lo anterior quiere decir que existe la posibilidad de que el usuario utilice y a su vez modifique el software, permitiendo que cualquiera pueda obtenerse por medio de MySQL desde internet y usarlo sin costo, se permite el acceso al código fuente para poder realizar sus propias modificaciones según los requerimientos que necesite para su propio servicio.

MySQL permite una mayor facilidad de uso agradando la experiencia del usuario puesto que permite usarse en casi cualquier tipo de ordenador y puede implementarse con otras aplicaciones web que requieran menor cuidado que otras, esto quiere decir que el referido sistema permite una configuración que ahorra memoria, potencia de CPU y las capacidades de un equipo sea cual sea, sin mayor complejidad, igualmente se puede interrelacionar con diferentes agrupaciones de maquinas unidas entre si, esto quiere decir que permite su funcionalidad tanto en sistemas embedidos como en clientes versus servidor, puesto que el software es multihilo SQL, facilitando el soporte en diferentes *Backend*'s, diversos programas, librerías o herramientas de uso administrativo que pueden ser utilizadas en una amplia lista de interfaces de programación.

Ahora bien, frente a la composición del sistema de MySQL se encuentra una gran variante diferenciadora con los demás servidores de datos pues que que funciona mediante una arquitectura de motores que permiten el almacenamiento de información y posterior copia de seguridad para futuras recuperaciones, esta información puede cargarse por plugins en tiempo de ejecución. En estos motores a los cuales se hace referencia se presentan conectores que permiten el acceso a los módulos que tenga el servidor y estos se encargan de analizar las consultas que eleve el usuario, lo anterior permite el acceso al sistema.

Se le llama pool de conexión, el cual hace posible el enlace proporcionando verdadero control de amenazas, memorias y cachés. Los motores encargados del almacenamiento de datos, componen en sí mismos el servidor de bases de datos que permiten la ejecución de acciones en datos subyacentes encargados de mantener el nivel del servidor en el mundo físico.

Pues bien, se entiende que la base de datos relacionada a lo largo de este escrito, también cuenta con ciertos motores de almacenamiento en su interior, siendo *InnoDB* su motor de almacenamiento por default, Oracle recomienda la implementación del mismo. Ahora frente a los motores de almacenamientos de datos que se encuentran en el exterior tienen la funcionalidad de mejorar el rendimiento de los productos y determinadas situaciones, suministrados por desarrolladores de software independientes o también por quienes conforman la comunidad del MySQL.

cuando se presenta la novedad de contar con variados motores de almacenamiento se permite aprovechar la disponibilidad de diversas bases de datos como puede ser los implementados en el archivo de información que no se entienden transaccionales por naturaleza, pero permiten la inserción y la lectura de información de forma óptima. Además, otros motores de almacenamientos con estas cualidades permiten operaciones de transacción cuando cuentan con cierto nivel informático y también, para los mas avanzados se permite mayor disponibilidad mediante técnicas de Clustering.

En lineas anteriores se menciona a InnoDB, el cual procedo a definir para mayor entendimiento del presente trabajo, pues bien, es un motor encargado del almacenamiento transaccional por default igualmente es de los mas populares, la base de datos cuenta con un diseño enfocado en el procesamiento de variantes transaccionales de poca duración, suelen ser completadas en lugar de revertidas, esto quiere decir que realiza sus funciones de manera automática y eficiente permitiendo la satisfacción de las necesidades de almacenamiento no transaccional.

Contiene una historia de liberación compleja, pero su utilidad y facilidad de funcionamiento permite la efectividad de las actuaciones requeridas, su origen se remonta al año 2008 cuando se lanzó por medio para el uso de MySQL 5.1. como un plugin, creado por Oracle el cual posteriormente dejándolo como un plugin de defecto propiedad de MySQL en su versión 5.5. Actualmente este plugin agrega nuevas características comprendidas como la construcción de índices de selección, capacidad para agregar o eliminar índices sin que esto implique reconstruir la tabla en un nuevo formato de almacenamiento, esto permite un mayor entendimiento al momento de almacenar grandes cantidades de información.

Su funcionamiento principal es acarrear la información en uno o en varios archivos de almacenamiento de datos conocidos popularmente como tablespace, el cual consiste principalmente en la caja negra de InnoDB que se gestiona por su propia cuenta, igualmente puede realizar particiones de disco en bruta que le permite la creación de su propio espacio en la tabla pero que es actualmente innecesario por los nuevos sistemas encargados del archivo.

En igual medida este sistema permite mediante la implementación de MVC para conseguir mayor concurrencia de datos, contiene un mayor nivel de aislamiento Repeteable Read igualmente contiene un mayor sistema de bloqueo a nivel de registro el cual se encarga de corregir las lecturas fantasma en el primer nivel de almacenamiento, es decir que se encarga del bloqueo de filas tocadas en consulta y es cuando InnoDB bloquea los vacíos que se presentan en la estructura del índice.

Cuando la plataforma cuenta con motores de almacenamiento transaccional, le permite el soporte de diversas copias de seguridad en linea por medio de varios mecanismos que incluyen MySQL Enterprise de Oracle. Se diferencia porque otros motores de almacenamiento con los que cuenta MySQL no cuentan con la capacidad de soportar copias de seguridad en linea que permita una copia coherente, se tiene que detener las escrituras de tabla que en una cara de trabajo de lectura - escritura.

Una base de datos que se permite contraponerse con *InnoDB* se conoce como *MongoDB*, lanzada para el año 2009 y su lugar se encuentra entre las bases de datos documentales, esta

base de datos nace como una novedad en lo referente al desarrollo de bases de datos que no cuentan con una estructura fija, usualmente cuenta con seguridad en seguridad en transacciones en menor nivel pero compensado con mayor rapidez en el acceso de datos y que escalan mejor que las bases de datos relaciones les tratados anteriormente en capítulos anteriores.

Entonces, *MongoDB* se comprende como una agrupación de información contenida en una base, cada una contiene diversas colecciones y cada una de las colecciones contiene diversos objetos, igualmente cada objeto representa una estructura de JSON que es comprendido como un listado de pares de clave y valor, siendo que esta base de datos principalmente trabaja con esquemas dinámicos.

MongoDB contiene diversas características que le permiten, como ya mencioné, estar a la par con InnoDB, dicho esto proceso a dar una explicación de cada una de ellas, una de las principales características de esta base de datos es que permite que los documentos y los arreglos embebidos, entendidos bajo el concepto de documento y no de fila, como ocurría anteriormente, pues bien, *MongoDB* permite que en un solo registro se reporten relaciones jerárquicas de alta complejidad , esto le permite liarse adecuadamente con la manera en que los usuarios expertos en desarrollo y lenguajes enfocados a objetos procesen mejor su información.

Se entiende igualmente, como una plataforma de libre esquema, esto quiere decir que un documento que no contiene las claves predeterminadas no contendrán un esquema que permita modificarse, dicho lo anterior, como no existe esquema alguno para modificar se presenta el movimiento de información masiva de forma innecesaria. Las claves nuevas o que resulten faltantes se pueden utilizar a modo de aplicación, en cambio a forzar a que toda la información o los datos opten por la misma forma, lo anterior quiere decir que se le presenta a los desarrolladores que la implementan una amplia facilidad en su trabajo frente a la evolución de los diversos modelos de datos[6].

Referente a los escalados de *MongoDB*, los conjuntos de datos que aumenta a un ritmo acelerado, los cambios tecnológicos actuales, también aumentan el ancho de banda de los que dispone y lo popular que se está volviendo este sistema en su implementación en equipos de mesa o portátiles que cuenten con una adecuada vinculación a la nube o a internet los cuales contaran con una inmersión segura en la que las aplicaciones de pequeña escala cuenten con la necesidad de almacenamiento de mayor cantidad de datos de los que inicialmente estaban diseñadas para soportar. Entonces, tenemos que un Terabyte de datos, en tiempos pasados se comprendía como una cantidad de información absurda por su gran tamaño y ahora, existe mayores cantidades siendo la anterior una cantidad irrisoria.

Conforme van aumentando los datos que requieren almacenamiento por parte de sus trabajadores, mas se enfrentan a la complicada situación de o ampliar implementando un equipo físico con mayor capacidad o por el contrario escalar que significa fragmentar la información en varios dispositivos. Frente a estos dos caminos se refleja que la ampliación representa menor resistencia pero un equipo de mayor calidad o capacidad tienen un muy alto precio en el mercado casi inaccesible para el desarrollador que lo requiere pero que no cuenta con suficientes recursos.

MongoDB se presentó como una solución ante una necesidad de escalar debido a que su modelo de datos se enfocado a documentos que pudieran dividirse de manera automática de los datos por medio de varios servidores. Puede implementarse un cluster el cual se encarga de la redistribución de documentos de forma autónoma lo cual le facilita al desarrollador enfocarse directamente en su función de programar aplicaciones sin necesidad de escalarlas, sin embargo, cuando se encuentran ante la necesidad de aumentar la capacidad de la maquina, únicamente añaden un cluster dejando que la base de datos organice los datos o información existente.

Igualmente como se ha hecho en InnoDB, me permitiré describir algunas de las características principales de este sistema de base datos, que permiten diferenciarse uno de l otro y que igualmente son objeto principal de este documento, pues bien, como se menciona anteriormente, ya no se toma en cuenta la información incluida a la plataforma como dato sino por el contrario como documento, eliminando el concepto de fila del diccionario, entonces, los documentos se entienden como una representación en código binario conocido como Binary JSON, dicho así, la codificación BSON se extiende a Java Script Objetc Notation permite incluir int, long, y float conocidos como documentos BSON en varios campos determinados los cuales individualmente contarán con un valor especifico, en el cual reposan los arreglos, los datos en binario y de igual forma los sub documentos[7].

Su principal ventaja es que en su implementación se pueden modelar los datos a modo de colecciones, por una parte se tiene la presentada ante el usuario y por otra parte la implementada para los artículos, cada uno se entiende como un documentos de Blog que permite diversos comentarios, etiquetas y categorías como interacciones de un conjunto integrado en la que puede participar el usuario.

Concluyendo con esta base de datos se entiende que *MongoDB* contiene los datos en un registro soportado en un solo documento, y cuando la información o datos reposan en una base de datos de carácter relacional para su registro suele representarse en varios espacios de tabla.

Teniendo un modelo documental en este sistema de datos, se permite un mejor seguimiento sobre la información que ha sido almacenado en él, facilitando en gran medida su búsqueda y encuentro, esto disminuye la necesidad de implementar JOINS en gran medida cuando se trata de tablas separadas.

Al concluir su funcionamiento se aprecia que representa un mayor rendimiento y escalabilidad al permitir con una única lectura en la base de datos puede permitir la recuperación del documento que contiene los datos relacionados, igualmente lo de carácter BSON implementada por *MongoDB* se entienden vinculados estrechamente con la arquitectura de los objetos del lenguaje de programación implementado, lo anterior facilita el trabajo del desarrollador cuando este tiene la tarea de modelas la forma en que los datos de la aplicación se asignan a los datos que se encuentran almacenados en una base de datos.

Legados hasta este punto, se hace necesario conforme se explica en "*MongoDB*: the definitive guide: powerful and scalable data storage"[7] la diferencia entre MySQOL y *MongoDB*, principalmente iniciaremos con los conceptos básicos propios implementados en el lenguaje de cada una de las plataforma, en la primera se entiende por el concepto de tabla lo que para

la segunda es definida como una colección, en la primera se implementa el concepto de fila el cual se modifica o desaparece con la segunda al entenderlo como documentos, en la primera se aprecia la definición de columnas en el almacenamiento de su información, en la segunda se determina como campo y para la primera tenemos los joins que para la segunda se manejan bajo el concepto de documentos embedidos o como su nombre inglés indica, linking.

Frente a los lenguajes de consulta que implementa cada uno se tiene que manejan un lenguaje rico en consulta, el SQL que maneja MySQL implementa un lenguaje en cadena que permite la consulta que será analizado posteriormente con la base de datos lo cual permite ejecutar la acción de SQL.

Ahora bien, se trataran los modelos de consulta implementados por *MongoDB*, contiene controladores nativos diseñados para todos los lenguajes de programación que pueda implementar en su trabajo el desarrollador, igualmente cuenta con controladores para el funcionamiento de *Framework*s, lo anterior permite un desarrollo adecuado de la estructura de la plataforma, también se tienen los drivers compatibles con Java, NET, *PHP*, PERL, RUDY, *JavaScript* incluso Node,Js implementado en mi empresa para el desarrollo que me asignan en la misma. Sin embargo, *MongoDB* se sirve de prescindir de todos los anteriores siendo que sus drivers están completamente diseñados para ejecutar cualquier lenguaje que se pretenda manejar en su espacio.

MongoDB por el contrario debe implementar objetos de consulta, esto quiere decir que pasa de un documento para dar explicación de lo que pretende consultarse, lo que quiere decir que no existe ningún lenguaje para analizar.

En lo que tiene que referirse a las relaciones se explica que MySQL cuenta con su mejor función conocida como JOIN frente a las bases de datos relacionales, popular por su forma de facilitar la consulta por medio de la implementación de diversas tablas.

Por otra parte, *MongoDB* no cuenta con la función de JOIN por se incompatible con la misma, sin embargo, si cuenta con variados tipos de datos multidimensionales como son los arreglos y los documentos que se manejan en este. El uso de un documento dentro de otro documento es conocido como una incrustación, es decir, si la necesidad fuese la creación de un blog en la plataforma de MySQL debería utilizarse obligatoriamente la tabla de publicaciones y a la par una de comentarios, pero en la plataforma de *MongoDB* si se permite la colección de manera unificada entre publicaciones y un array de comentarios en el interior de la publicación.

Cuando se requiere el almacenamiento de datos en MySQL, se depende de las tablas y las columnas que estructuran la misma, hecho que no ocurre en *MongoDB* puesto que no cuenta con la definición del esquema, es decir, que únicamente nos pide o exige el ingreso de los datos o la información en los documentos para que posteriormente el gestor de bases de dados ejecute la adecuación de esta información de la estructura o arquitectura conforme lo que requiere el desarrollador que la está implementando.

3.1. Conceptos Claves

A continuación se presentan algunos conceptos que son clave para el entendimiento del tema.

Node.js: Es un entorno de tiempo de ejecución de *JavaScript* de código abierto y multiplataforma el cual ejecuta el motor *JavaScript* V8 y el cual permite la fácil creación de aplicaciones de red, rápidas y escalables. *Node.js* utiliza un modelo de E/S sin bloqueo y controlado por eventos que lo hace liviano y eficiente, perfecto para aplicaciones en tiempo real con uso intensivo de datos que se ejecutan en dispositivos distribuidos. [8]

Python: Se entiende como un lenguaje de programación cuyo objeto principal es habilitar de forma eficiente y óptima la integración de sistemas de manera efectiva, lo que le permite ser ideal en la implementación de tareas de administración de sistemas de procesamiento y de diseño web, desde el momento en que datan sus orígenes esto es para la década de 1990. A medida que ha avanzado en el tiempo y *Python* evoluciona a cambiado su lenguaje a uno más preciso y claro permitiendo su propio entorno de desarrollo efectivo.[9]

PHP: Es un lenguaje de secuencias de comandos de propósito general el cual se entiende ampliamente especializo en el desarrollo web, incluso al punto de estimarlo en el 79.2 % de los sitios web actualmente dependientes en un amplio grado de lenguaje, con el constante avance progresivo del tiempo se han desarrollado varios *Framework*s para *PHP* como lo son Laravel, Symfony, Codelgniter entre otros.[10]

ApacheBench: Herramienta en Apache que permite y facilita la realización de solicitudes en un servidor Web Local para avalar únicamente el tiempo de procesamiento, desestimando el extremo temporal de inicio y final que lleva normalmente la transmisión de datos.[1]

Framework: Marco de trabajo que agrega funcionalidad extendida a un lenguaje de programación, automatiza muchos de los patrones de programación para orientarlos a un determinado propósito, proporcionando una estructura al código, mejorándolo y haciéndolo más entendible y sostenible, y permite separar en capas la aplicación.[11]

Modelo MVC: se entiende como un patrón de arquitectura de software que permite la separación entre la recolección de información de la interacción con el usuario, el modelo comprende los elementos conocidos como datos de aplicación, reglas de negocio, lógica del mismo y las funciones que se derivan de estos, entonces, una vista puede identificare como cualquier representación de salida de datos comprendidos como gráficos o diagramas.[3]

MongoDB: Base de datos no relacional que guarda la estructura de los datos en documentos tipo JSON con un esquema dinámico llamado BSON, no existe un esquema predefinido. Los elementos de los datos son llamados documentos y se guardan en colecciones, las cuales pueden tener un número indeterminado de documentos.[12]

JavaScript: Se presenta como un lenguaje de desarrollo de aplicaciones cliente/servidor a través de internet.[13]

4. Metodología y plan de trabajo

4.1. Esquema metodológico

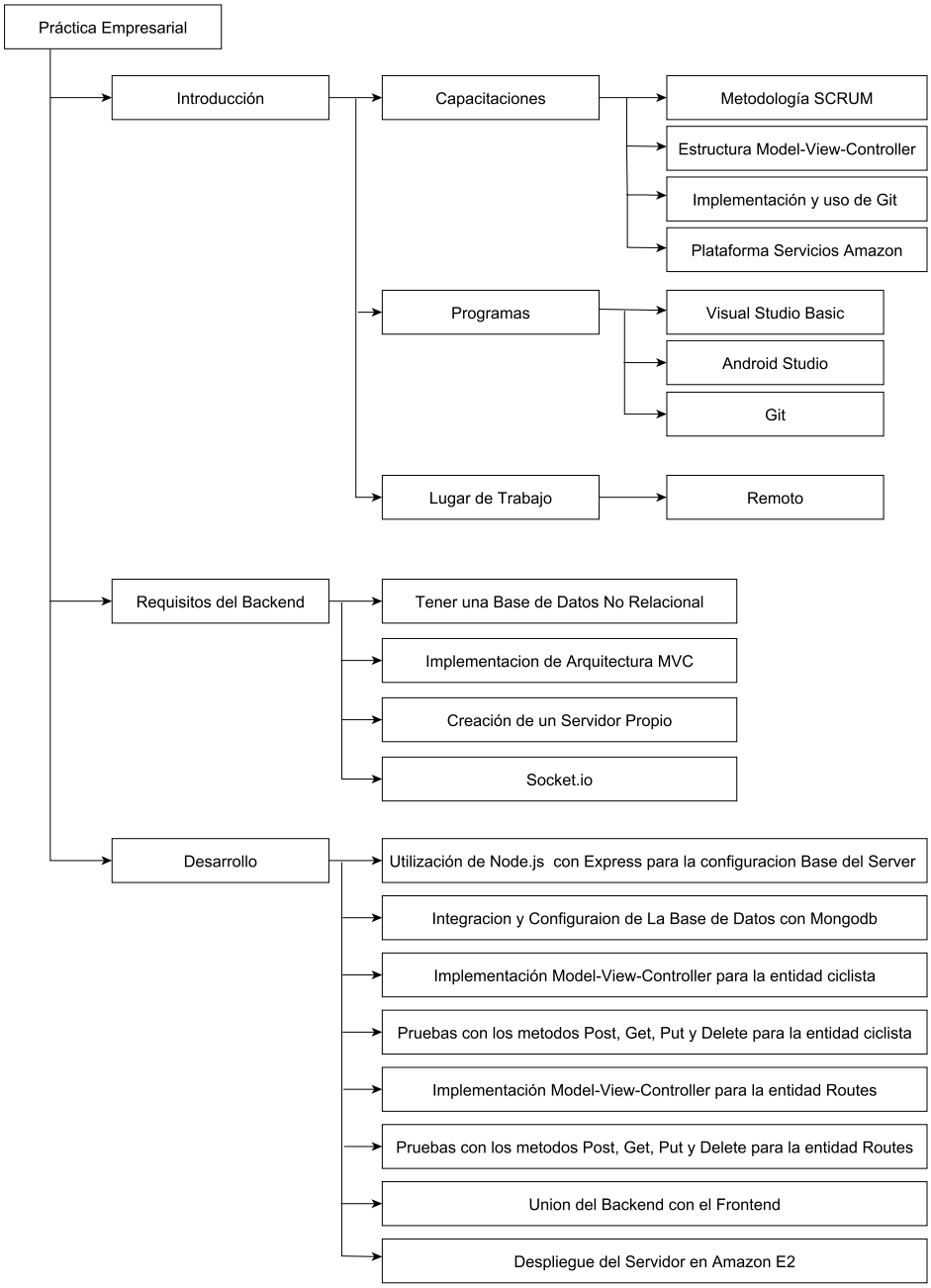

Figura 4: Esquema metodológico Practica empresarial
Fuente: Propia del Autor

4.2. Descripción de esquema metodológico

Este esquema metodológico representa las etapas y actividades necesarias para el desarrollo de este proyecto en la empresa Cinnov S.A.S.

La Primera Etapa consistió en una introducción general a todo nuestro entorno de desarrollo el cual esta compuesto por tres programas fundamentales:

- *Visual Studio Code*: El cual es un editor de código fuente que incorpora soporte para JavaScript, *Node.js* y posee un amplio ecosistema de extensiones. [14]
- *Android Studio*: Entorno de desarrollo integrado que proporciona herramientas para la creación de aplicaciones móviles y posee una característica fundamental llamada Fast Emulator que permite emular un dispositivo físico.[15]
- *GitHub*: Sistema de control de versiones gratuito que esta diseñado para manejar proyectos con rapidez y eficacia. [16]

Posteriormente se realizaron capacitaciones en temas relacionados con:

- Metodología *SCRUM*: la cual permitió obtener la capacidad óptima de flujo de trabajo en el desarrollo de software, estableciendo una serie de tareas en un determinado tiempo.
- Estructura MVC: Estilo de arquitectura la cual es ampliamente empleada en el desarrollo *Backend* ya que permite la separación de la interfaz de usuario, la lógica de control y los datos de aplicación.
- Manejo de Git y Plataforma de Servicios de Amazon: La cual proporciono el conocimiento necesario para el manejo de versiones en el código base del *Backend* y la implementación de este código en un servicio de Amazon para tener un Server en la Nube.

La segunda Etapa consiste en los Requisitos necesarios para la creación de un *Backend* con arquitectura MVC, una base de datos no relacional para el almacenamiento de usuarios y coordenadas de rutas en tiempo real y la implementación de Socket.io para la comunicación bidireccional de eventos entre la aplicación y el servidor, teniendo en cuenta que se opto por crear el *Backend* desde cero ya que en un principio la parte de almacenamiento de usuarios lo realizaba un servicio de tercero el cual era Amazon Cognito y el cual no permitía un tratamiento de datos adecuado a la lógica de negocio de la empresa.

En la Tercera etapa (Desarrollo) se opto por la utilización de *Node.js* como lenguaje base y su *Framework* Express para la fácil creación del servidor local, seguidamente de la integración de la base de persistencia utilizando *MongoDB*, y la implementación de las entidades con arquitectura MVC de ciclista que almacena los datos relevantes para el registro y la entidad Routes que guarda pared de coordenadas en tiempo real y de las cuales se realizaron sus respectivas pruebas con los métodos de petición HTTP, cabe aclarar que también participé en una parte del diseño del *Frontend* para su unión con el *Backend* y posteriormente subir el código base del *Backend* a la nube.

4.3. Recursos necesarios

1. Computador Personal con Licencia de Windows 11
2. Conexión a internet
3. Energía Eléctrica
4. Dispositivo físico Bigo
5. Sensor dispositivo Físico Bigo
6. Teléfono Móvil

5. Modelado Funcional

5.1. Análisis de Requerimientos

La administración de un proyecto es un método que permite organizar y planificar el trabajo y así alcanzar un objetivo y para ello se es necesario tener presente cuales son los requerimientos necesarios para el funcionamiento de nuestro proyecto, en el desarrollo de software lo podemos dividir estos requerimientos en dos grupos:

- Requerimientos Funcionales: Son las funciones de los servicios que entregara nuestro *Backend* dependiendo de los datos obtenidos desde el *Frontend*.
- Requerimientos no Funcionales: Corresponden a todo objeto que no intervienen directamente en los procesos de los servicios del *Backend*.

5.1.1. Requerimientos Funcionales

En la siguiente tabla se plasma los requerimientos funcionales presentes en el desarrollo del *Backend* los cuales hacen referencia a cada una de las funciones desarrolladas con una breve descripción.

Cuadro 1: Requerimientos Funcionales.

Entidad	Funcionalidad	Descripción
Cyclist	Create	Permite la creación de un usuario
	findAll	Lista a todos los usuarios en la base de datos
	update	Encuentra a un usuario por su ID y lo actualiza
	findOne	Muestra a un usuario en especifico, su parámetro de entrada es el ID
	filterEmail	Comprueba si un correo existe, de no existir lo crea automáticamente y de existir manda un reporte de novedad
Course	Create	Crea una ruta con los pares de coordenadas
	findAll	Lista toda las rutas almacenadas
	find	Encuentra una ruta por su ID
	update	Encuentra una ruta por su ID y lo actualiza
	delete	Elimina una ruta
Socket.io	Room	Crea una sala y permite el envío de coordenadas en tiempo real a todos aquellos unidos a la sala

Fuente: Propia del Autor

5.1.2. Requerimientos no Funcionales

A continuación se presenta un pequeño esquema que refleja todo lo referente a los requerimiento no funcionales con una breve descripción sobre lo que atañe a cada uno.

Cuadro 2: Requerimientos no Funcionales.

Tipo	Requerimiento	Descripción
Software	Persistencia de datos	Base de datos no relacional que permite almacenar los datos en un formato ligero
	Formato de respuesta de las funciones	Cada función que se realiza en el *Backend* devuelve una respuesta en formato JSON
	Versión	En cuanto a versiones se uso la versión v16.14.0 de *Node.js* y la versión 5.0.6 de *MongoDB*
Desempeño	Numero de usuarios	Se estima que el numero de usuarios activos sea elevado
	Numero de operaciones concurrentes	Al tener un numero elevado de usuarios se espera una alta tasa de solicitudes
	Tiempo de Respuesta	Se espera que el tiempo de espera entre solicitudes sea lo más mínimo posible
Interfaz	Mensajes de errores	Cada solicitud hecha de forma incorrecta al servidor devolverá un mensaje de error 400 ó 500 dependiendo del caso
Calidad de Software	Flexibilidad y Estructura de directorios	Con la separación de capas en Modelo, Vista y Controlador se logra una flexibilidad para la incorporación de nuevas funcionalidades y se logra un código con estructura y flexible

Fuente: Propia del Autor

5.2. Diseño de Implementación

Para el diseño de la implementación se decidió utilizar un lenguaje unificado de modelado o UML el cual permite de una forma fácil y practica la visualización del diseño de un proceso conociendo así su estructura y comportamiento, esto es una ventaja ya que cualquier persona puede comprender al instante los diferentes procesos mediante los diagramas de casos de uso.

5.2.1. Diagramas de Casos de uso

En este apartado se exhiben los diagramas de casos de uso y su comportamiento en relación a la interacción de un evento previo. Estos diagramas tienen la finalidad de ilustrar los requerimientos funcionales.

Para lograr tener un mayor entendimiento de lo expresado en las lineas anteriores me permitiré ilustrarlo en los siguientes diagramas:

En el primer diagrama se manifiesta la representación de un esquema UML de la entidad Cyclist y los proceso específicos que se realizan dependiendo del tipo de petición que se realiza, cabe aclarar que cada proceso o función dentro del *Backend* tiene una dirección especifica, para posteriormente devolver una respuesta en formato JSON.

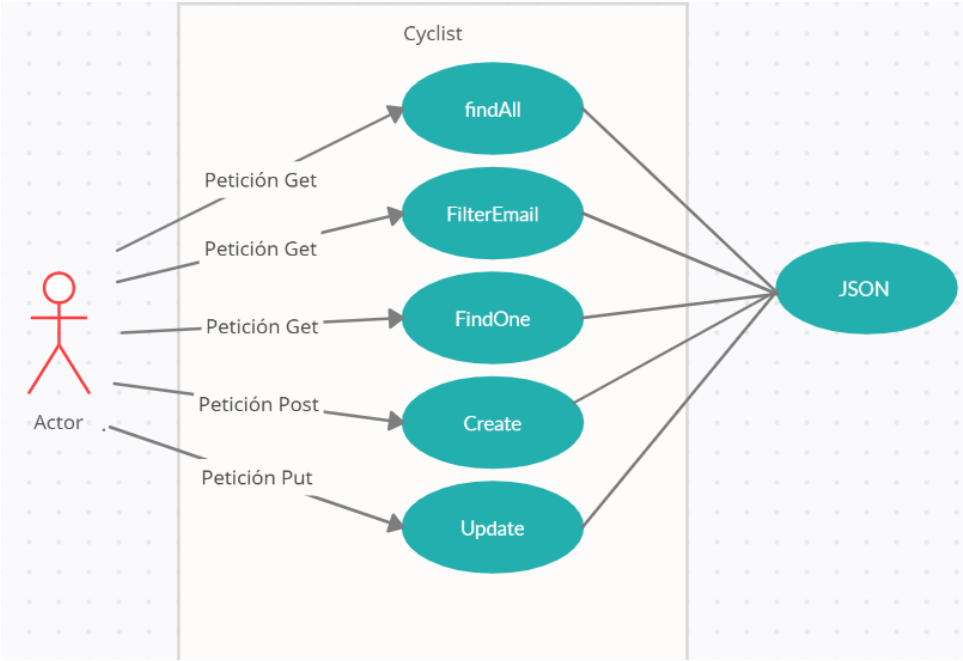

Figura 5: Diagrama UML para la entidad Cyclist
Fuente: Propia del Autor

Ahora, refiriendo al segundo diagrama se representan los procesos específicos para la entidad Course y de igual manera su respuesta en formato JSON.

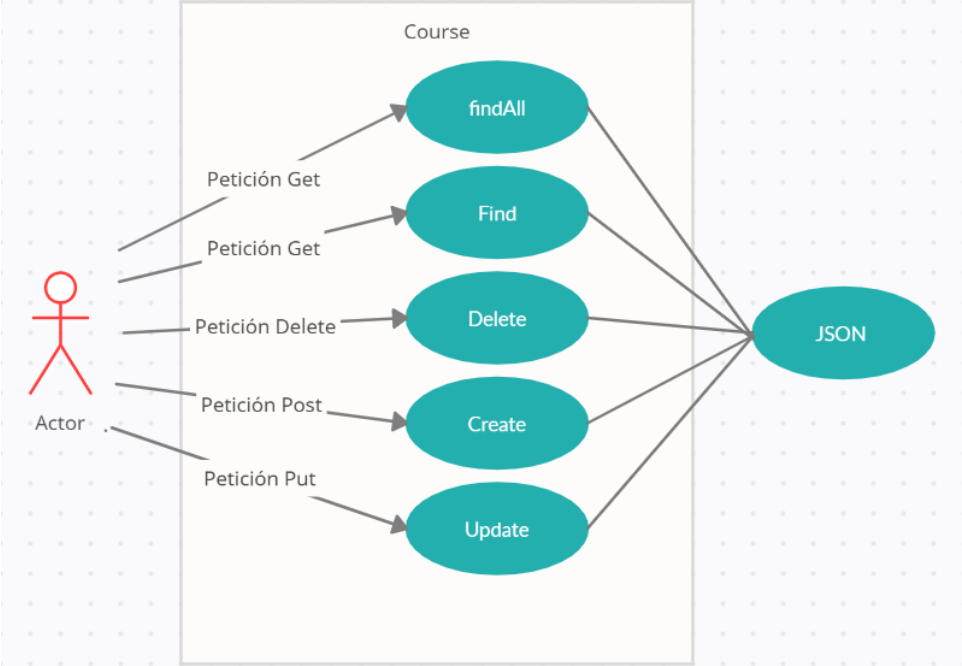

Figura 6: Diagrama UML para la entidad Course
Fuente: Propia del Autor

El asunto tratado en el tercer diagrama representa la funcionalidad de la implementación en socket para ejecutar una comunicación bilateral entre el cliente y el servidor, teniendo un evento llamado connection el cual creara una sala y procederá a ejecutar la función savegps la cual como resultado enviara en formato JSON las coordenadas gps a todos los que se encuentran unidos a la sala

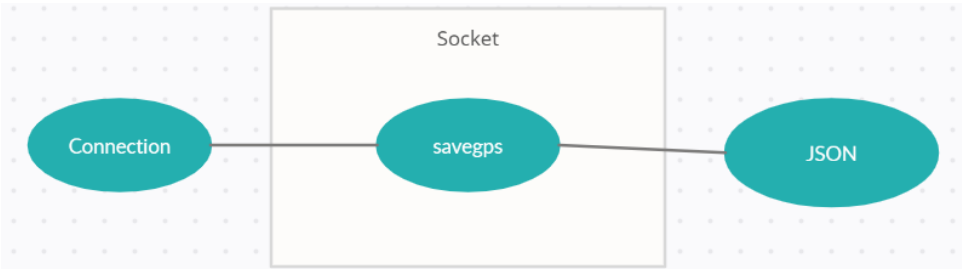

Figura 7: Diagrama UML para Socket
Fuente: Propia del Autor

5.2.2. Arquitectura

Para la creación del *Backend* se utilizó la arquitectura MVC ya que esta me permitió tener un código flexible y separarlo en tres capas fundamentales, Modelo, Vista y Controlador y la cual represento en el siguiente diagrama.

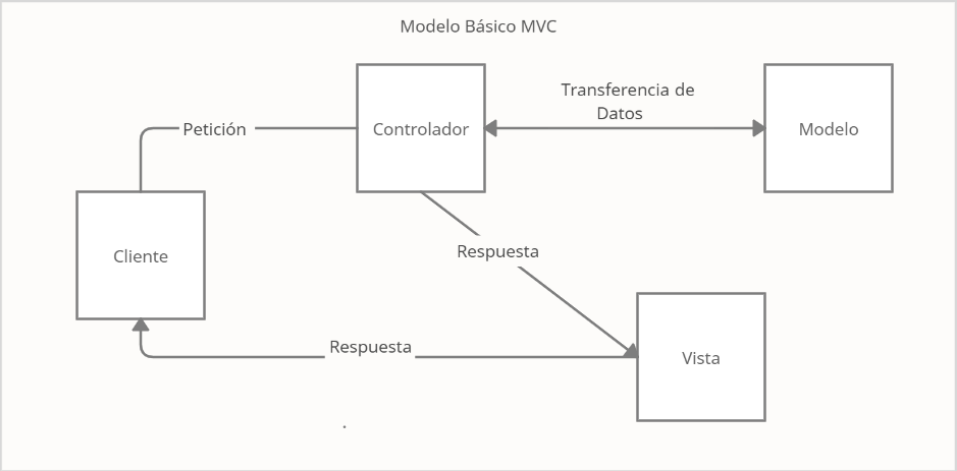

Figura 8: Diagrama Base de la arquitectura MVC
Fuente: Propia del Autor

- Modelo: Responsable de interactuar con la base de datos y dar respuesta a nuestro controlador de cualquier cambio en esta.
- Controlador: Responsable de recibir las peticiones por parte del cliente y realizar las acciones que gestionan los eventos y transmitirlo a la vista.
- Vista: Responsable de recibir datos procesados por el controlador y mostrarlos al cliente.

El tener esta arquitectura en todo el código permite integrar nuevas funcionalidades a un futuro.

El siguiente diagrama representa los parámetros del modelo utilizados en la entidad cyclist y los métodos que contiene el controlador.

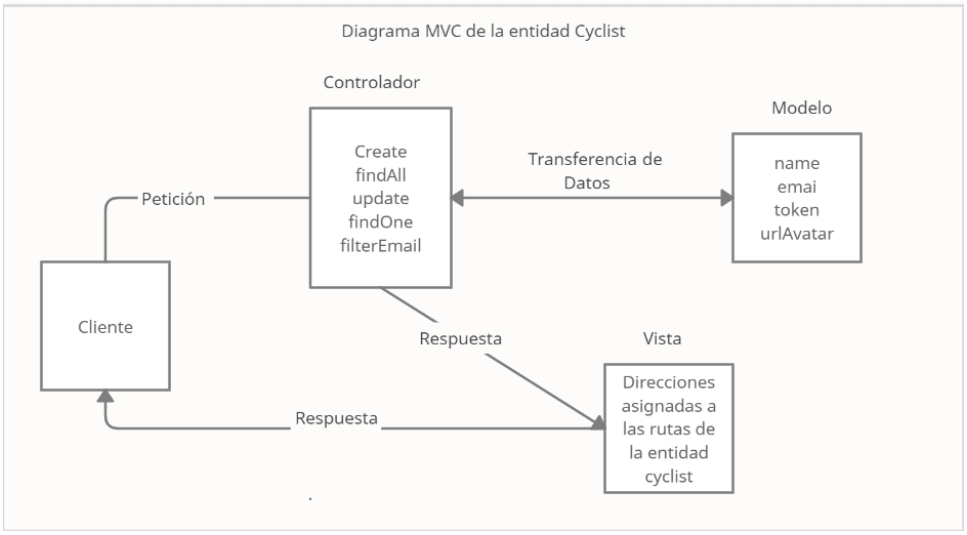

Figura 9: Diagrama MVC entidad ciclista
Fuente: Propia del Autor

Para el siguiente diagrama MVC que corresponde a la entidad Course podemos notar que la estructura es la misma a la de cyclist con la diferencia que cambian los parámetros del modelo y algunos métodos del controlador

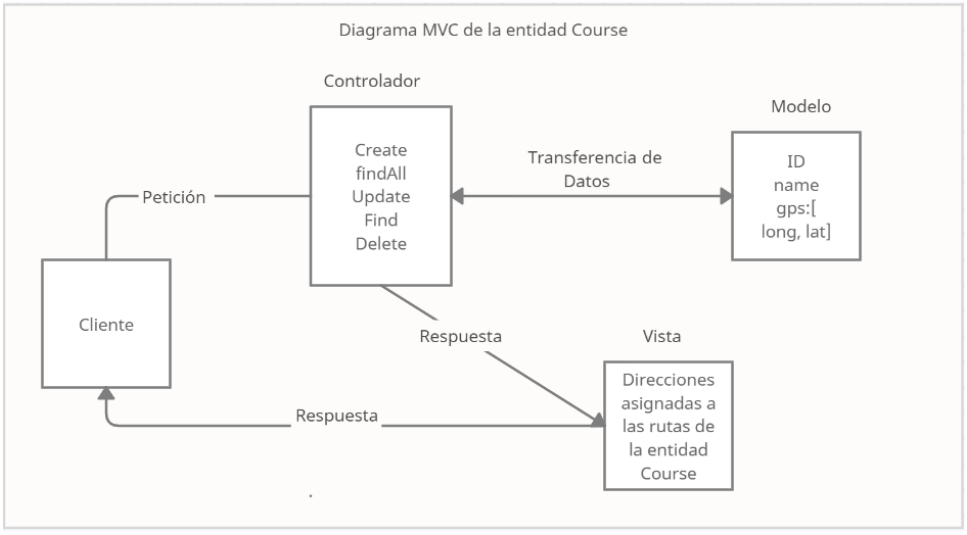

Figura 10: Diagrama MVC entidad course
Fuente: Propia del Autor

Para el caso de Socket en el solo se implemento la comunicación bilateral entre el cliente y el servidor como se ilustra en el siguiente diagrama

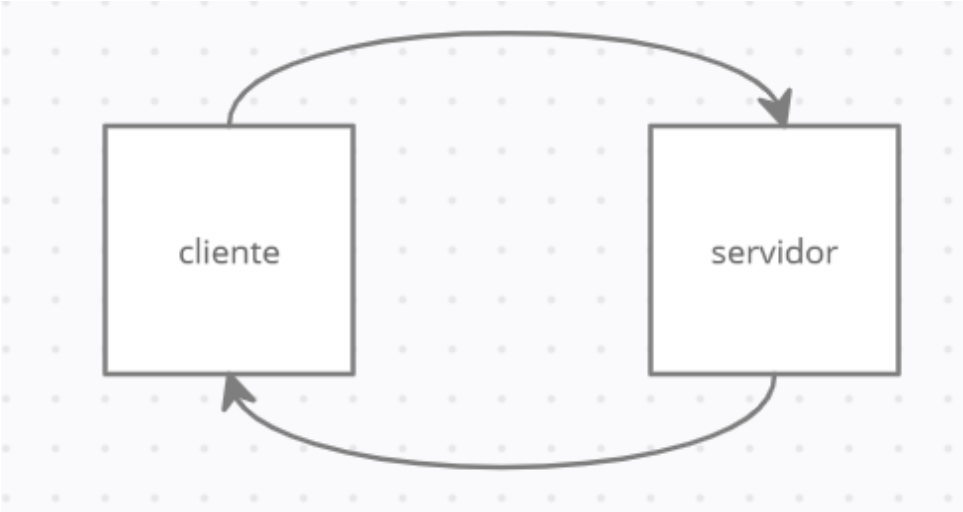

Figura 11: Comunicación bilateral de Socket
Fuente: Propia del Autor

6. Implementación

Teniendo en cuenta lo requerimientos y diseño de la arquitectura que se propuso anteriormente se puede proceder a la etapa de implementación, en las siguiente sección se presentará lo mas relevante de la configuración del entorno, estructura del código y el despliegue del servidor, ya que por políticas de derecho de autor no esta permitido divulgar el código fuente porque es propiedad exclusivamente de la empresa.

6.1. Configuración del entorno de Desarrollo

El primer paso para la creación del *Backend* es instalar en nuestras computadoras los programas necesarios, entre los cuales podemos mencionar, Visual Studio Code, *Node.js*, *MongoDB* y Insomnia los cuales son el entorno de Desarrollo, entorno de ejecución para JavaScript, Base de Datos no relacional y Aplicación de solicitudes HTTP, respectivamente, una vez instalados los prerrequisitos procederemos a crear un proyecto en nuestro entorno de Desarrollo y ejecutar el comando "npm init"que aparece en la siguiente imagen y el cual nos arrojara una serie de opciones como nombre del proyecto, versión, descripción entre otros y así proceder a crear un archivo llamado package.json el cual es el encargado de contener todas las dependencias del código

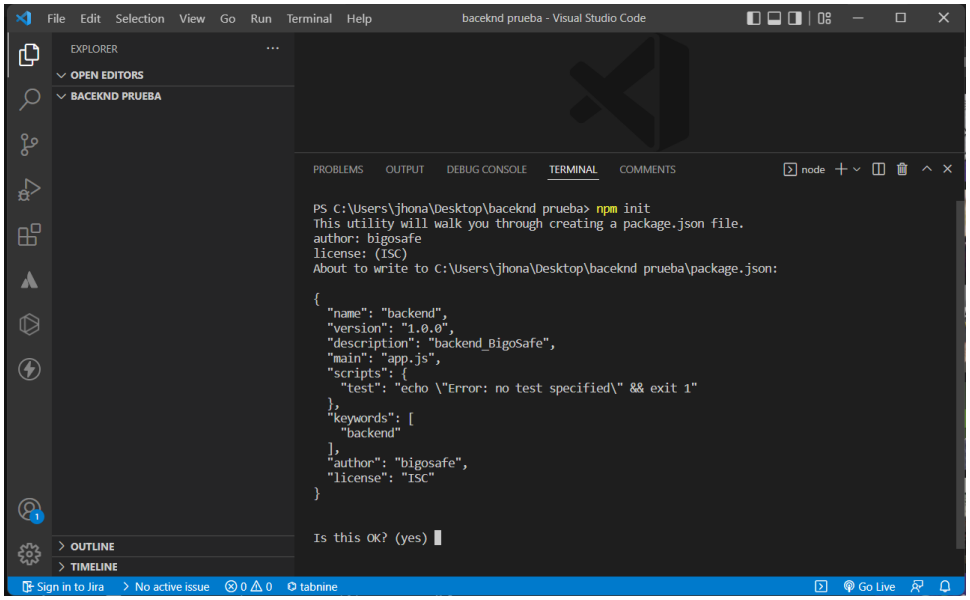

Figura 12: Iniciación de un Proyecto con *Node.js*
Fuente: Propia del Autor

Seguido de esto se procede a instalar las demás dependencia al código fuente con el comando npm install (Nombre del paquete) en la Figura 10 se representa las dependencias incluidas en el archivo package.json.

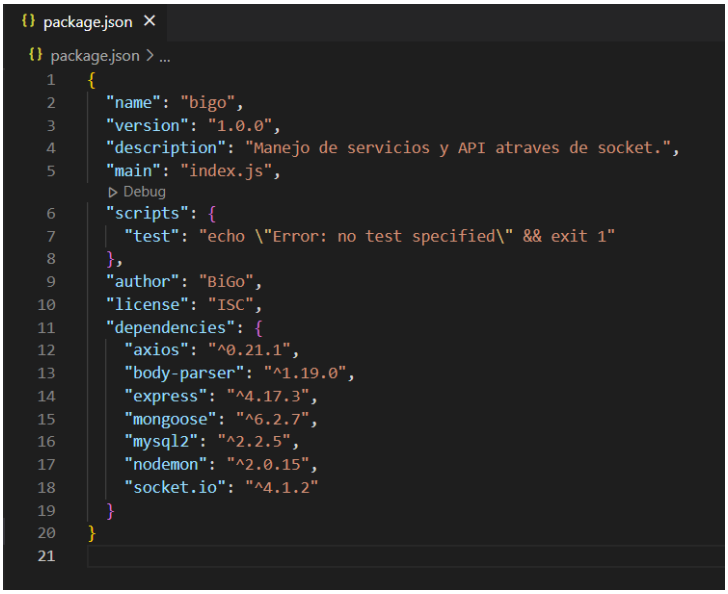

Figura 13: Archivo Package.json con dependencias incluidas
Fuente: Propia del Autor

6.1.1. Implementación Arquitectura MVC

En la parte de arquitectura de nuestro *Backend* se logra notar la organización y nomenclatura usada en los nombres de los archivo para así facilitar a cualquier programador el entendimiento del código, también se siguió lo planeado en la etapa de diseño.

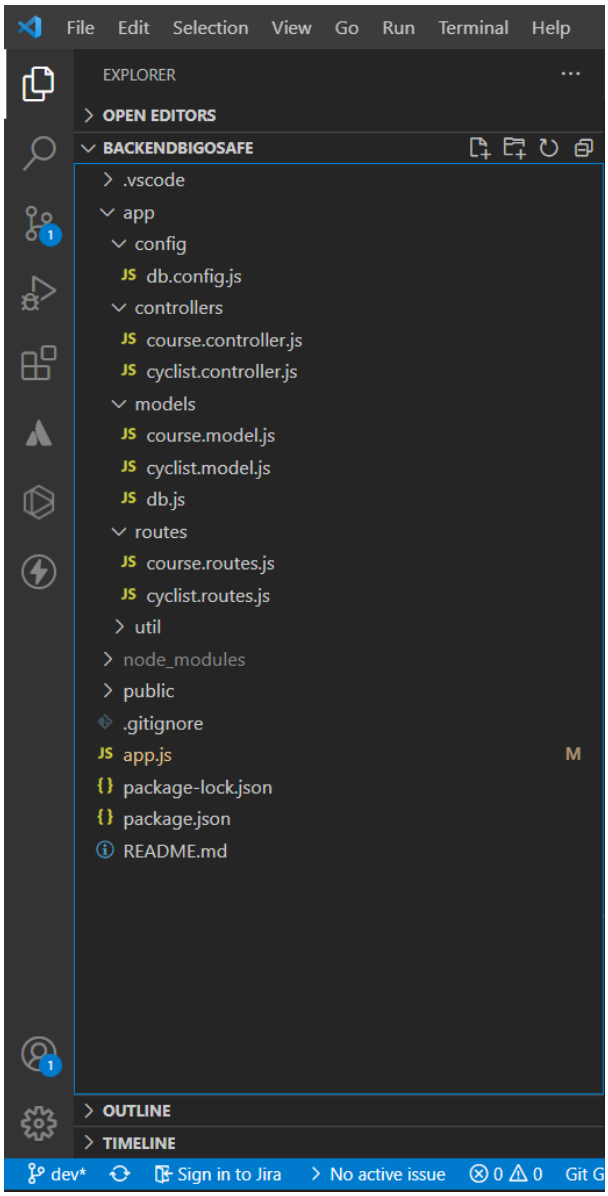

Figura 14: Arquitectura Implementada MVC
Fuente: Propia del Autor

6.1.2. Despliegue del servidor en la nube

Ahora bien, después de tener nuestro servidor funcional y testeado en local se procedió a su despliegue a la nube a través de los servicios de AWS Amazon para ello se creo una instancia en el Servicio E2 el cual permite crear una maquina virtual con base en sistema linux para a la cual denominaremos instancia.

En la figura 12 se muestra la opción seleccionada para la creación de de dicha instancia

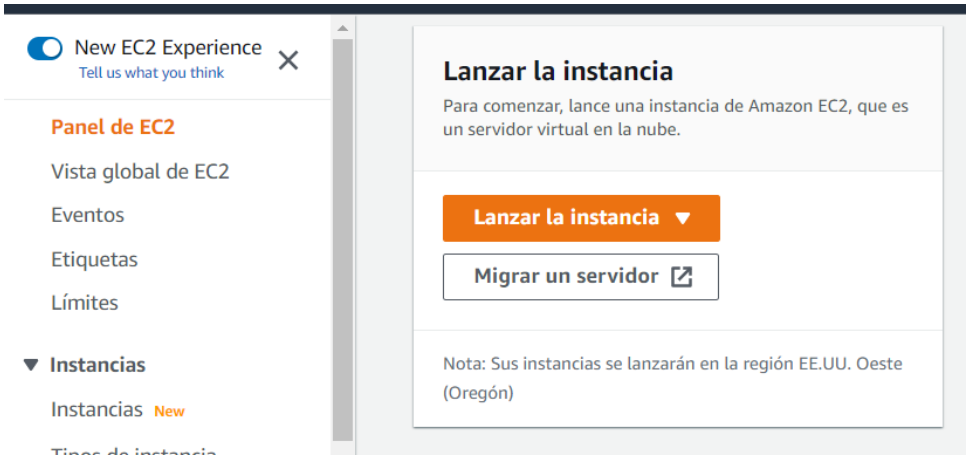

Figura 15: Panel Servicio E2 de AWS Amazon
Fuente: Propia del Autor

Al momento de lanzar nuestra instancia se despliega un menú para la configuración de esta misma, el cual contiene Nombre y etiqueta como lo representa la figura 13.

Figura 16: Panel Servicio E2 de AWS Amazon
Fuente: Propia del Autor

En la Figura 14 muestra el tipo de sistema operativo que se pueden seleccionar para la maquina virtual, entre los cuales esta una variedad de distribuciones de linux, windows y mac OS.

Otra opción importante a seleccionar es el

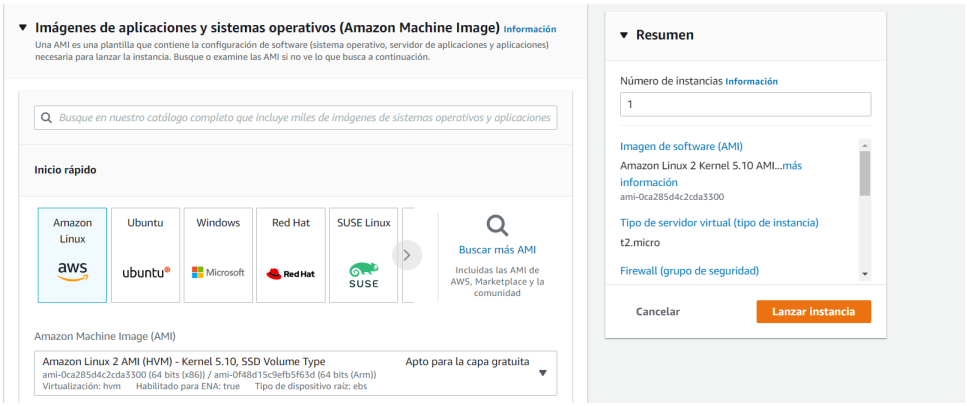

Figura 17: Opciones de Sistemas Operativos de la Instancia
Fuente: Propia del Autor

Seguidamente se configuran las opciones de Tipo de instancia en la cual seleccionaremos el microprocesador que mejor se adapte a nuestras necesidades, y generaremos un par de claves que usaremos posteriormente para ingresar a la maquina virtual por consola.

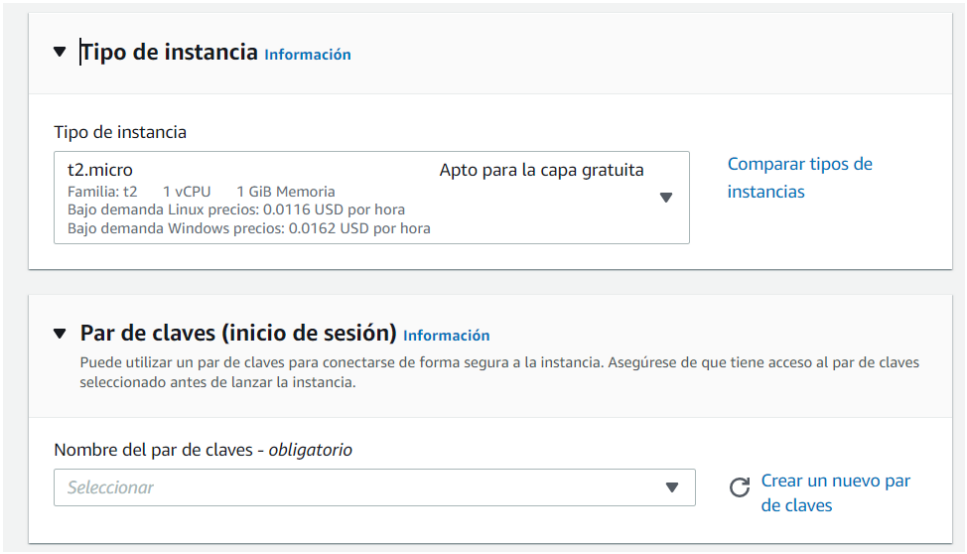

Figura 18: Opciones de configuración de tipo de instancia y par de claves
Fuente: Propia del Autor

A continuación procederemos a seleccionar las configuraciones de red necesarias, en este caso dejaremos las opciones por defecto.

Figura 19: Opciones de configuración de red
Fuente: Propia del Autor

Por ultimo en la Figura 17 se muestra las opciones de almacenamiento disponibles y en la parte derecha un resumen de todo lo seleccionado y la opción para lanzar la instancia ya configurada

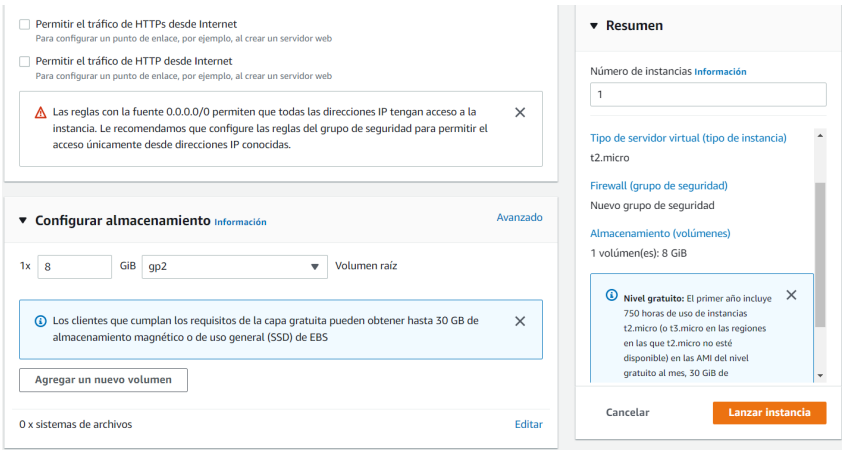

Figura 20: Opciones de configuración de almacenamiento y Resumen
Fuente: Propia del Autor

Una vez desplegada nuestra instancia se debera mostrar como se ilustra en la Figura 18

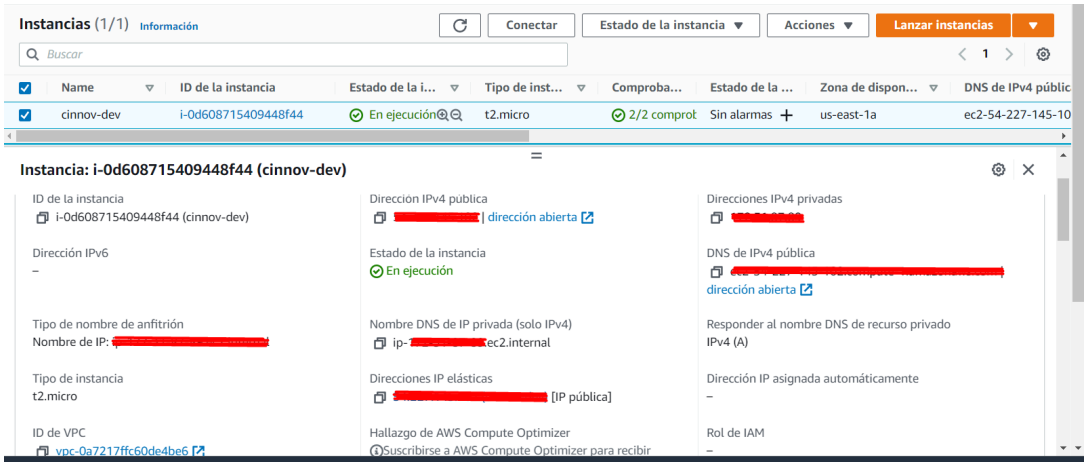

Figura 21: Instancia desplegada
Fuente: Propia del Autor

Como paso final ingresamos a la instancia, mediante putty que es un cliente SSH que permite la conexión a través de Windows como se ilustra en la figura 19 y posteriormente usaremos el comando git clone y la dirección del repositorio del Código.

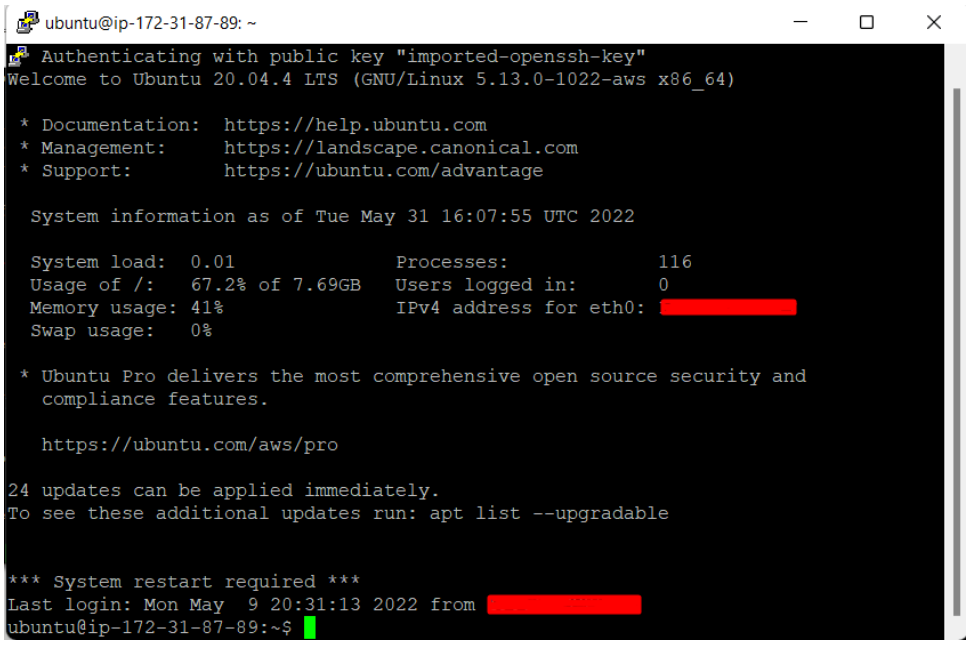

Figura 22: Instancia desplegada
Fuente: Propia del Autor

7. Resultados

En esta sección evidenciaremos los resultados que arroja nuestro *Backend* a cada una de las solicitudes que se mencionaron en sección 5.2.1 de este documento y resultados de la pequeña participación que tuve en la parte del *Frontend* ya que por cuestiones internas de organización de la empresa me designaron únicamente al *Backend*.

Primeramente comenzaré con los resultados del *Backend*.

Como se puede apreciar en la Figura 20 al momento de realizar una petición GET a nuestro servidor con la ruta especifica http://localhost:3000/cyclist que esta relacionada al método findAll del controlador cyclist nos devolverá un archivo en formato JSON con toda una lista de usuarios registrados en la base de datos

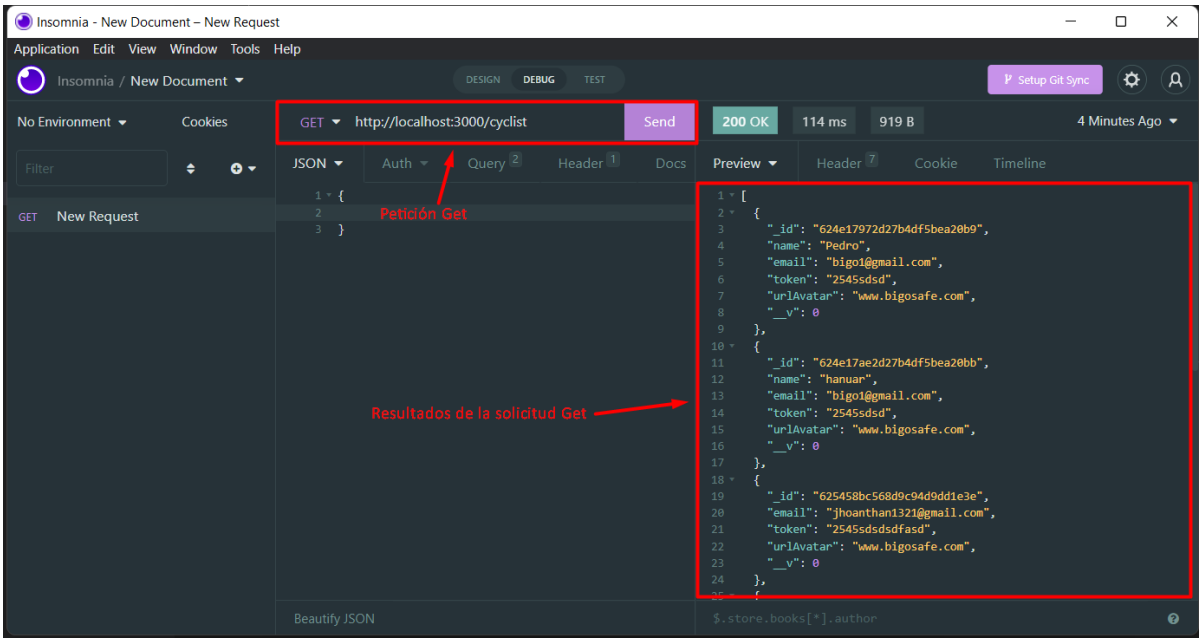

Figura 23: Resultado Petición Get del metodo findAll
Fuente: Propia del Autor

En la figura 21 correspondiente a una solicitud POST de la entidad Cyclist de nuestro *Backend* se evidencia la creación de un usuario en la base de datos ya que guarda los datos de entrada y le asigna un ID especifico.

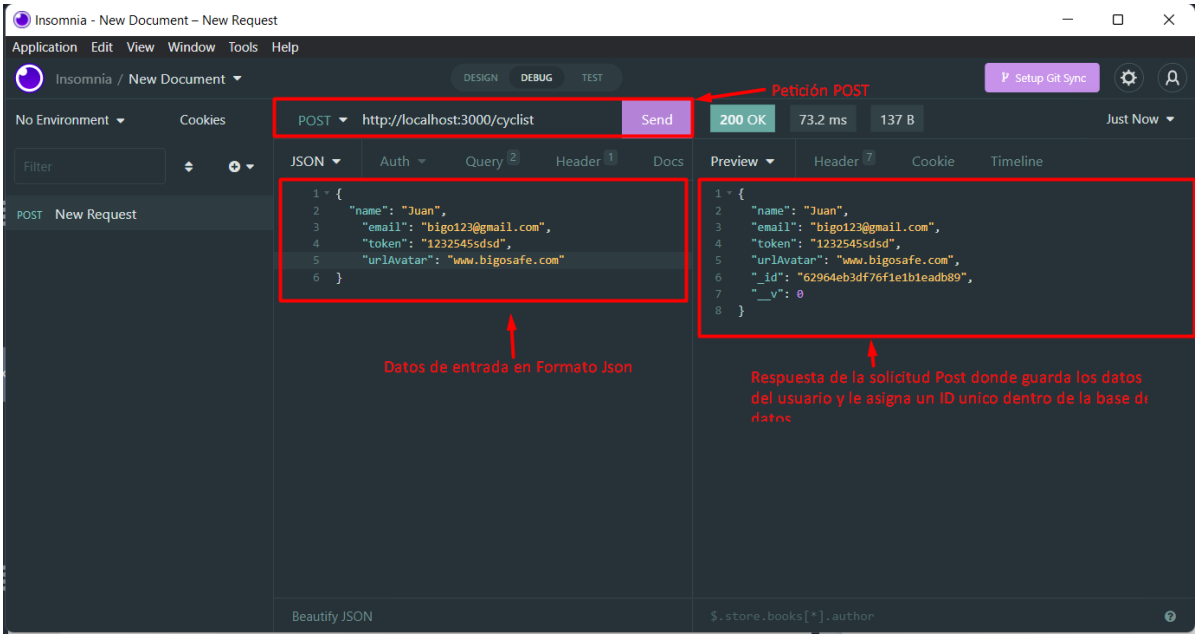

Figura 24: Resultado Petición POST del método Create
Fuente: Propia del Autor

En la figura 22, representa el resultado de una petición GET pasando por parámetro de entrada el correo electrónico y devolviendo el ID asociado a este.

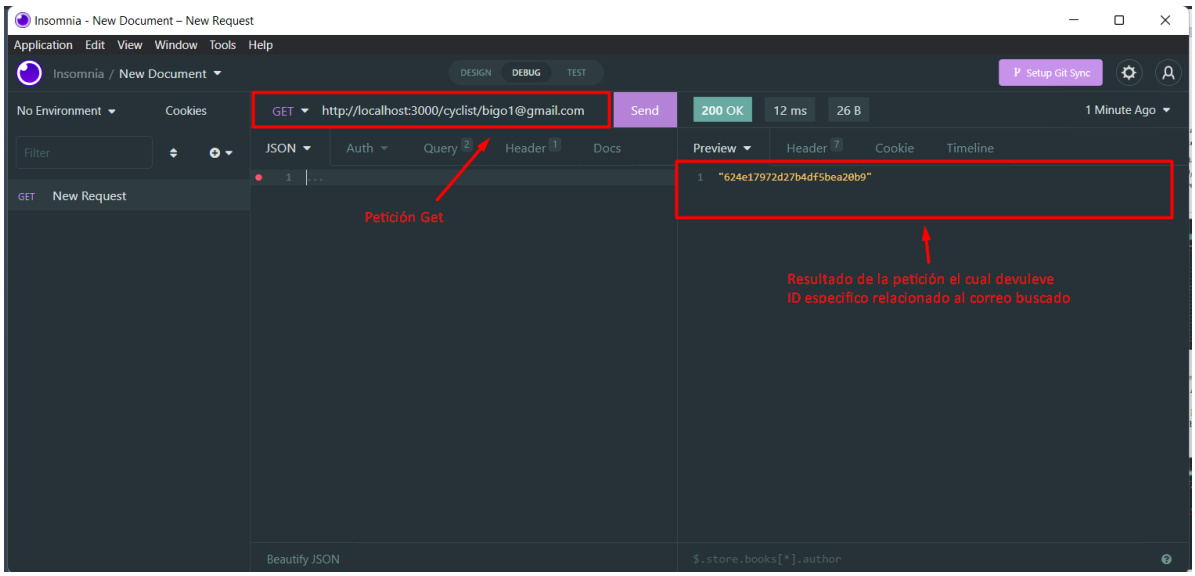

Figura 25: Resultado Petición GET del método filterEmail
Fuente: Propia del Autor

En la figura 23, representa el resultado de la petición PUT la cual tiene como requisito el ID, para la actualización de cualquier parámetro asociado al ID especifico.

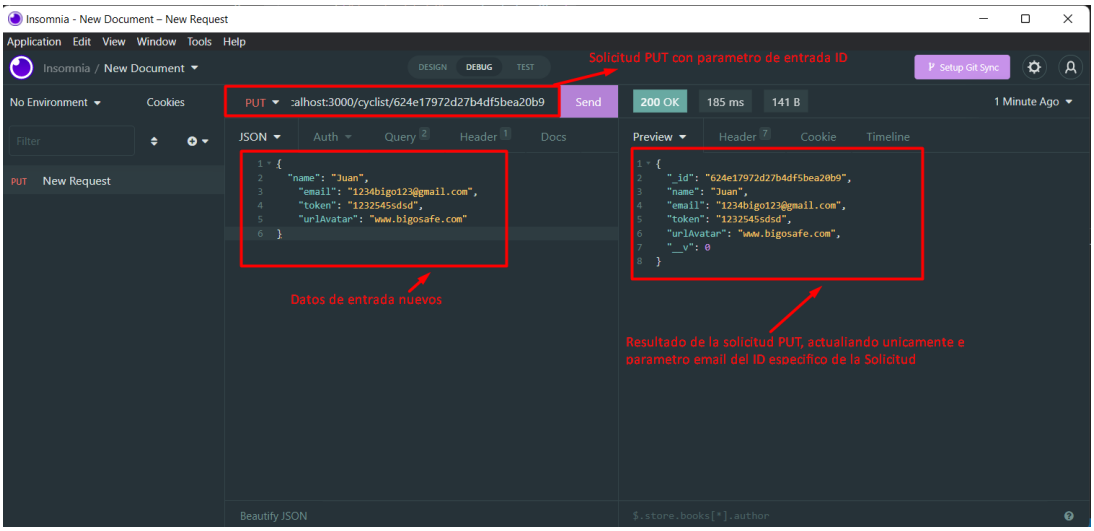

Figura 26: Resultado Petición PUT del método update
Fuente: Propia del Autor

En la figura 24, representa el resultado de la petición GET la cual tiene como requisito el ID, para la consulta de cualquier parámetro asociado al ID especifico.

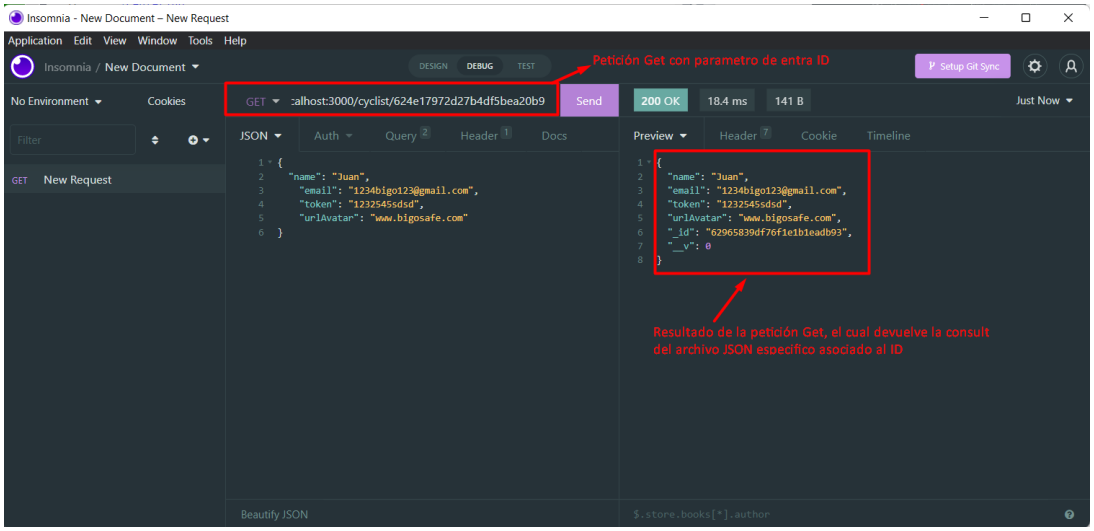

Figura 27: Resultado Petición GET del método findOne
Fuente: Propia del Autor

A continuación se representan los resultados arrojados por el *Backend* para la entidad Course la cual es la encargada de manejar la parte de rutas y coordenadas en el servidor. En la figura 25 se hace referencia a una solicitud Post que corresponde al método Create en el controlador de Course y el cual es el encargado de crear y guardar con un ID único los datos del modelo.

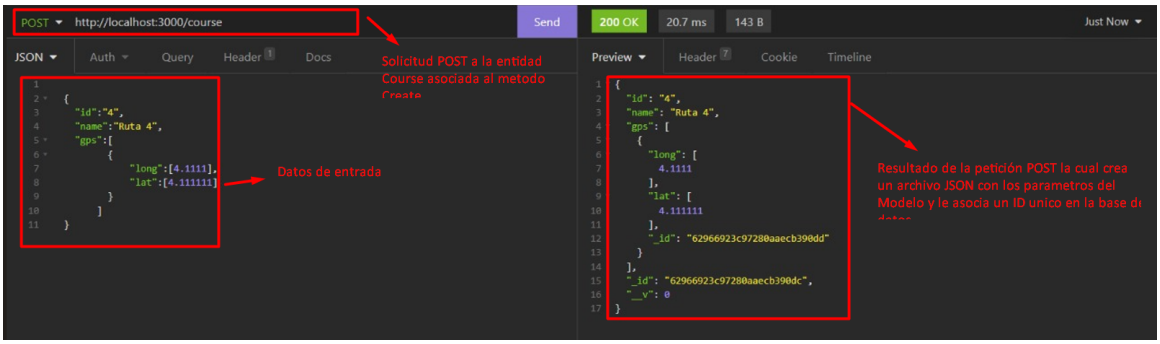

Figura 28: Resultado Petición Post del método Create de la entidad Course
Fuente: Propia del Autor

En la imagen 26 se representa una solicitud de tipo GET para la entidad Course la cual devuelve en un archivo JSON todas las Rutas guardadas en la base de datos

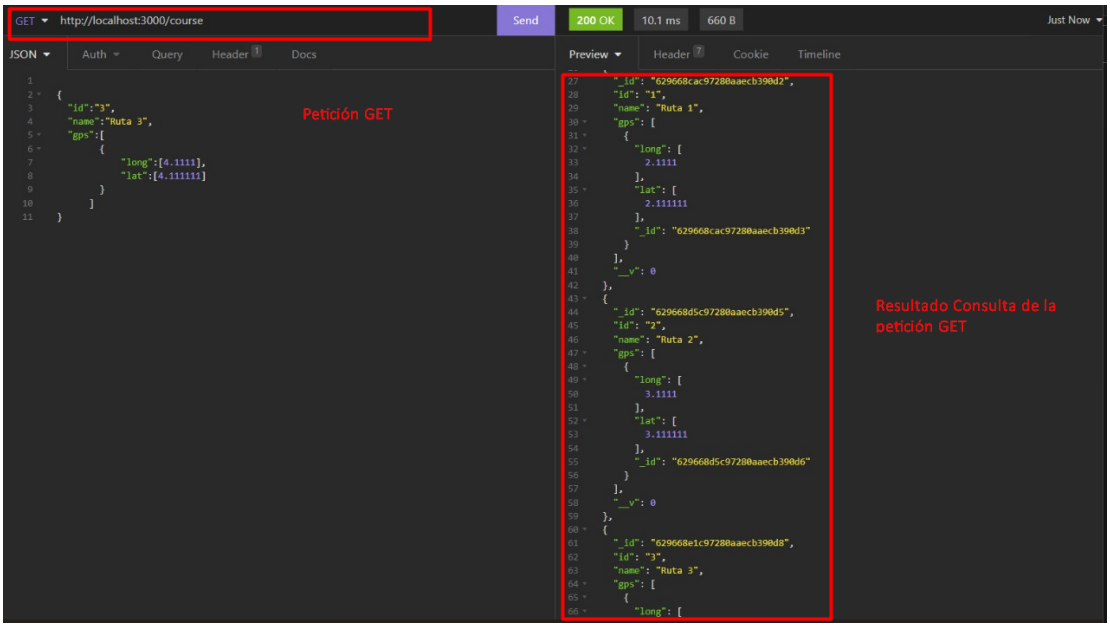

Figura 29: Resultado Petición Get del método findALL de la entidad Course
Fuente: Propia del Autor

En la imagen 27 se evidencia el resultado de una solicitud GET con parámetro de entra ID el cual proporciona la búsqueda de los datos asociados a ese ID únicamente dentro de toda la base de datos.

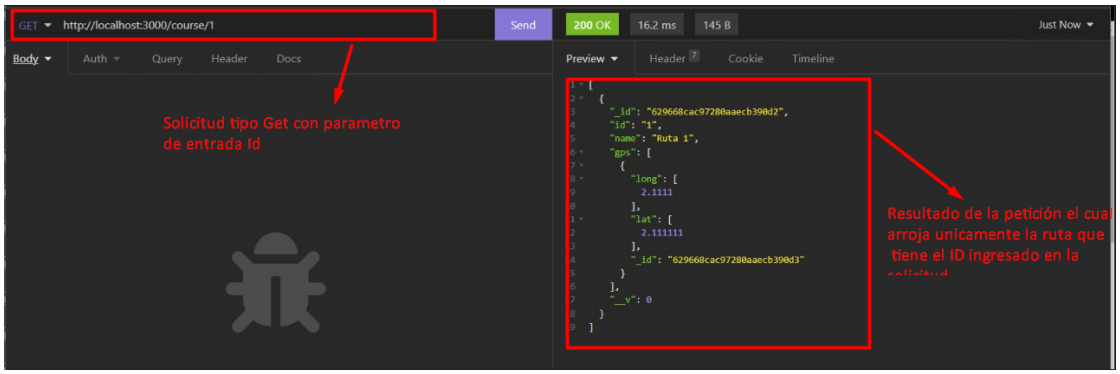

Figura 30: Resultado Petición Get del método findOne de la entidad Course
Fuente: Propia del Autor

En la imagen 28 hace referencia al método delete el cual tiene como parámetro de entrada el ID del objeto que se desea eliminar.

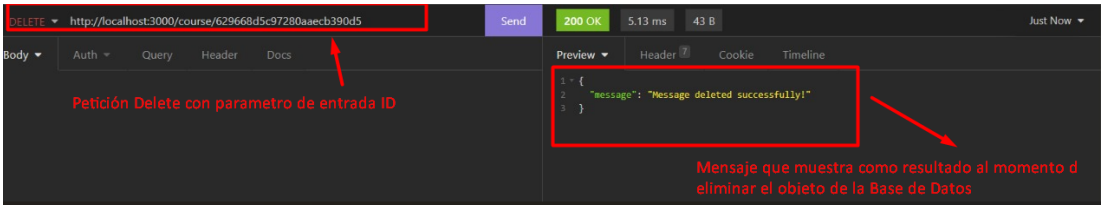

Figura 31: Resultado Petición Get del método Delete de la entidad Course
Fuente: Propia del Autor

Terminando la parte del *Backend*, presentare las vistas del diseño de la ventana de registro, login, y recuperar contraseña en las que hice parte del desarrollo *Frontend*.

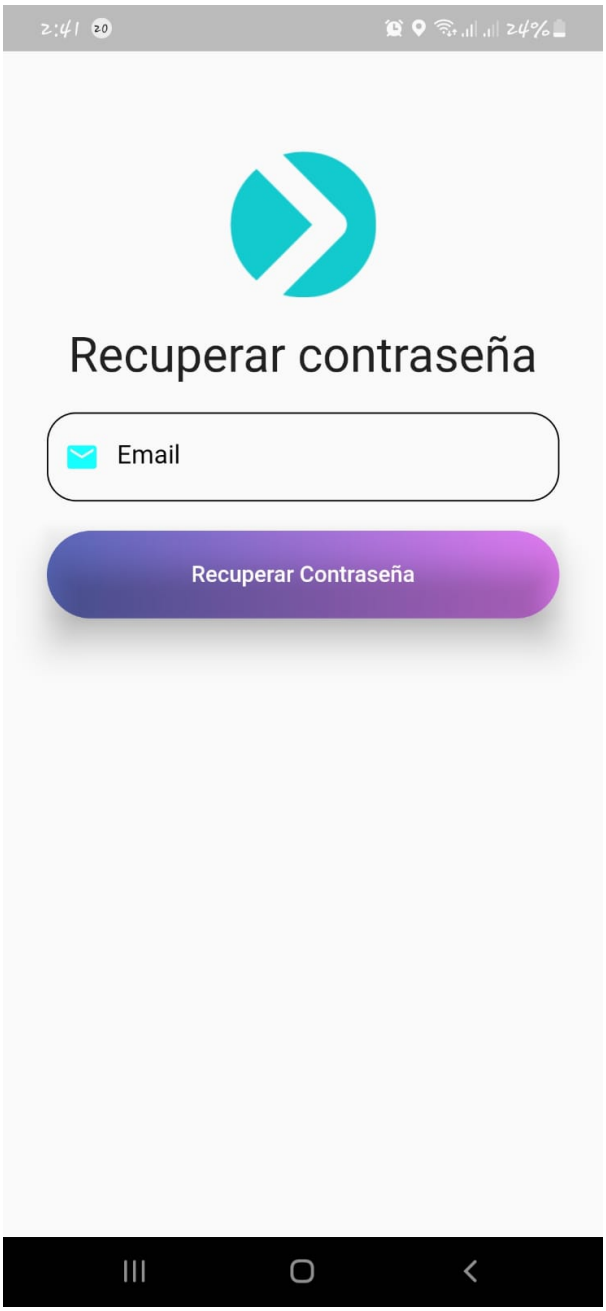

Figura 32: Vista del diseño de la Ventana Login
Fuente: Propia del Autor

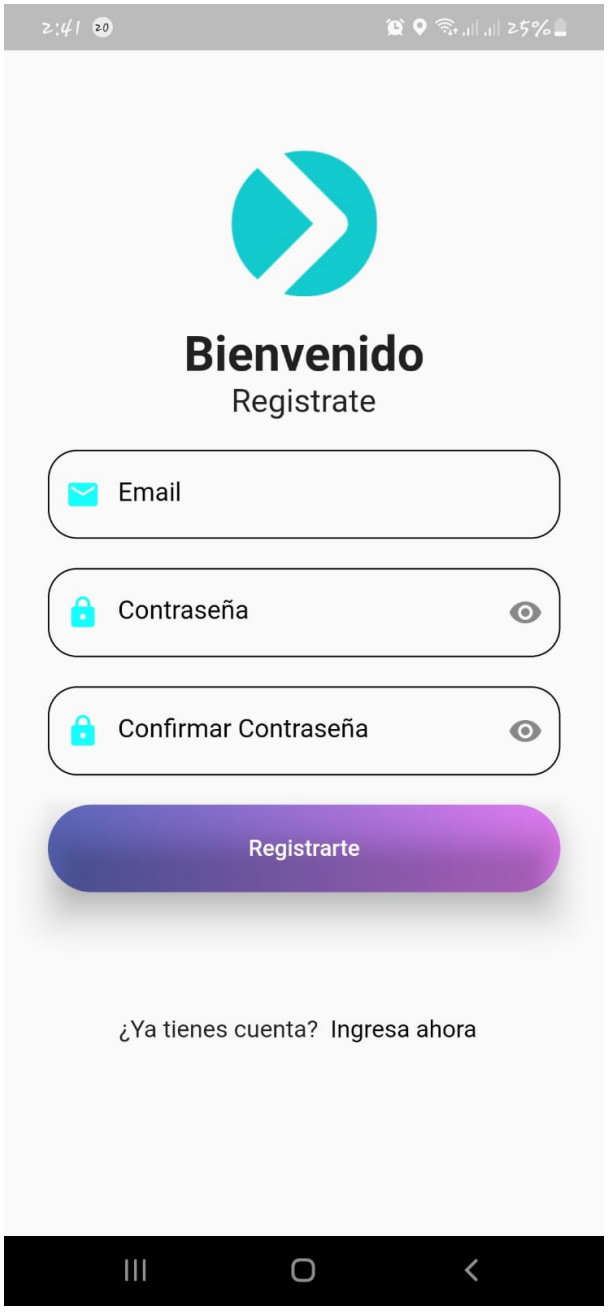

Figura 33: Vista del diseño de la Ventana Recuperar Contraseña
Fuente: Propia del Autor

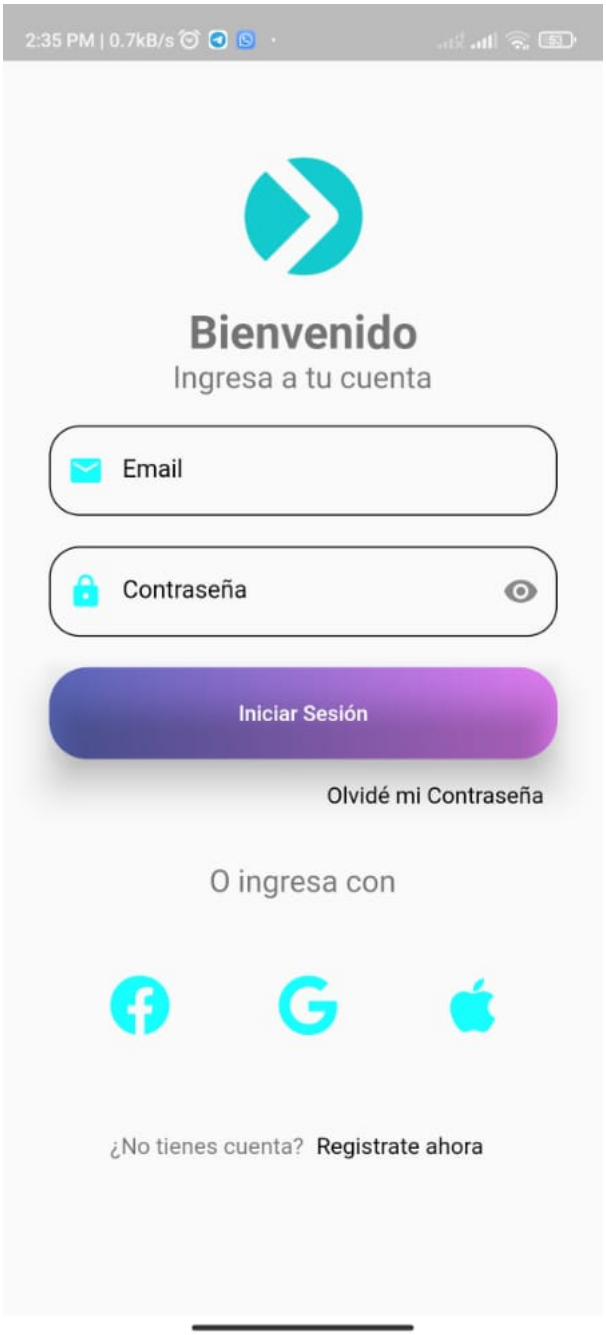

Figura 34: Vista del diseño de la Ventana Login
Fuente: Propia del Autor

8. Conclusiones

El desarrollo de este proyecto coloco varios retos que poseían un grado de complejidad ya que era un campo totalmente desconocido que requería una capacidad de adaptabilidad y de auto-aprendizaje en temas de Diseño, Arquitectura e Implementación de Software y gracias a esto se experimento un acercamiento al campo laboral de la ingeniería de Software permitiendo conocer un basto mundo de posibilidades, que a la larga pueden integrarse y mejorar mis habilidades como Ingeniero Mecatrónico.

Teniendo presente lo anteriormente dicho se manifiesta la importancia de tener preparación adecuada como desarrollador a partir de capacitaciones, Bootcamp o cursos especializados en el área para así al momento de conocer la lógica de negocio de la empresa y ser parte de un equipo de desarrolladores se logre un desarrollo estructurado y eficaz de cualquier proyecto propuesto.

En cuanto al objetivo principal de este proyecto el cual consiste en desarrollar un Backend Robusto y flexible que proporcione los servicios necesarios de la aplicación para ciclistas basada en geolocalización y permitiera al equipo de Frontend migrar sus funcionalidades proporcionadas por un servicio de terceros a un Backend estable y propio.

Entre las funcionalidades desarrolladas por el *Backend* esta la creación, modificación y búsqueda tanto para la parte usuarios como la de rutas, todo esto se logro gracias al trabajo conjunto que se llevo con el equipo de Frontend ya que el Backend tiene una relación indirecta con este mismo y viceversa.

Siendo que el *Backend* esta aún en desarrollo aun tiene un largo recorrido ya que se busca integrar todos los servicios ofrecidos por la empresa en un único lugar para así tener mejor control de los datos y esto pueda mejorar la experiencia del usuario.

El sistema de persistencia de datos no relacional tiene un mayor rendimiento frente al relacional ya que este no depende de la realización de consultas en varias tablas para entregar una respuesta lo cual proporciona una velocidad mayor de consulta cuando se presenta una gran cantidad de usuarios.

Referencias

[1] K. Lei, Y. Ma, and Z. Tan, "Performance comparison and evaluation of web development technologies in php, python, and node. js," in *2014 IEEE 17th international conference on computational science and engineering*. IEEE, 2014, pp. 661–668.

[2] E. Haro, T. Guarda, A. O. Z. Peñaherrera, and G. N. Quiña, "Desarrollo backend para aplicaciones web, servicios web restful: Node. js vs spring boot," *Revista Ibérica de Sistemas e Tecnologias de Informação*, no. E17, pp. 309–321, 2019.

[3] A. M. Florez Mendoza, D. C. Daza Díaz *et al.*, "Desarrollo de una arquitectura de software para gestión de información no estructurada."

[4] Y. D. González and Y. F. Romero, "Patrón modelo-vista-controlador." *Telemática*, vol. 11, no. 1, pp. 47–57, 2012.

[5] S. Keller, "Mongodb an introduction and performance analysis," Master's thesis, 2012.

[6] C. A. Vele Zhingri, "Análisis de rendimiento entre la base de datos relacional: Mysql y una base de datos no relacional: Mongodb," B.S. thesis, Universidad del Azuay, 2016.

[7] S. Bradshaw, E. Brazil, and K. Chodorow, *MongoDB: the definitive guide: powerful and scalable data storage*. O'Reilly Media, 2019.

[8] Jesús Lucas. (04 de Septiembre de 2019) `Qué es NodeJS y para qué sirve`.

[9] A. C. Álvarez, *Python 3. Curso Práctico*. Grupo Editorial RA-MA, 2016.

[10] M. A. Arias, *Introducción a PHP*. IT Campus Academy, 2013.

[11] G. M. Villalobos, G. D. C. Sánchez, and D. A. B. Gutiérrez, "Diseño de framework web para el desarrollo dinámico de aplicaciones," *Scientia et technica*, vol. 16, no. 44, pp. 178–183, 2010.

[12] A. E. Martín, S. B. Chávez, M. A. Murazzo, N. R. Rodríguez, and A. Valenzuela, "Mongodb en ambiente cloud híbrido con openstack," in *XVII Workshop de Investigadores en Ciencias de la Computación (Salta, 2015)*, 2015.

[13] M. Á. S. Maza, *Javascript*. Innovación Y Cualificación, 2012.

[14] J. Hugon, *C# 7: desarrolle aplicaciones Windows con Visual Studio 2017*. Ediciones Eni., 2018.

[15] B. Hohensee, *Introducción a Android Studio. Incluye proyectos reales y el código fuente*. Babelcube Inc., 2014.

[16] V. Cosentino, J. L. C. Izquierdo, and J. Cabot, "Findings from github: methods, datasets and limitations," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016, pp. 137–141.