UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN

DESARROLLO DE LA LÓGICA Y BASE DE DATOS PARA SISTEMA WEB Y APLICACIÓN MÓVIL STRESSMAPP

POR

David Alejandro Torres Gallardo

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título de Ingeniero(a) Civil Informático

Profesor Patrocinante
Gonzalo Rojas Durán

Profesoras Co-patrocinantes
Pamela Guevara Álvez
Kristin Schmidt

Julio 2023
Concepción (Chile)

ÍNDICE

1. INTRODUCCIÓN

El estrés (en contextos maladaptativos) está relacionado con el aumento en el riesgo, la aparición, mantención y recaída de un rango de enfermedades físicas y mentales. Sin embargo, en algunas circunstancias el estrés permite a los individuos mnejar situaciones negativas y mantener su bienestar, por lo que bajo una visión psiquiátrica transdiagnóstica de la salud y enfermedades mentales, este puede ser visto como un impulsor clave de la salud y enfermedad. De esta forma, el estudio del estrés es la base del trabajo y metodología propuesta en el presente proyecto.

Sin embargo, existe una brecha entre la investigación experimental del estrés en laboratorios y la investigación enfocada en salud aplicada (que predominantemente usa valoraciones de estrés subjetivas o evaluaciones de eventos estresantes). Luego, para abordar este problema, el presente proyecto interdisciplinario, que corresponde a uno de los 25 ganadores del concurso de financiamiento interno VRID 2022 de la Universidad de Concepción, "StressMApp: Desarrollo de una aplicación móvil para identificar marcadores cognitivos relacionados con el estrés, y sus relaciones con medidas clínicamente relevantes de autorreporte en una muestra de estudiantes chilenos" propone:

1. El desarrollo y uso de una aplicación móvil para la obtención remota de datos relacionados a medidas de estrés mediante autoinformes y tareas cognitivas.
2. Replicar los resultados de estimaciones de controlabilidad moduladas por estrés en individuos saludables, usando tareas computarizadas para evaluar el comportamiento exploratorio y realizar evaluaciones de un ambiente basado en tareas.
3. Relacionar los resultados de las tareas con las autovaloraciones de síntomas y eventos de estrés considerable.

4. Desarrollar lazos colaborativos dentro de la universidad, nacionalmente e internacionalmente para integrar esta investigación con otros proyectos científicos y clínicos en desarrollo, para la prevención de la depresión y ansiedad en jóvenes.

Ahora, la presente Memoria de Título se enfoca en el desarrollo del backend de un prototipo funcional de la aplicación móvil para la obtención de los datos para la investigación en desarrollo. Además, se implementó un sistema web para la visualización y obtención de los datos entregados por el prototipo por parte de los investigadores.

1.1 Antecedentes Generales del Problema

El equipo multidisciplinario liderado por la Dra. Kristin Schmidt, del Departamento de Psiquiatría y Salud Mental de la Facultad de Medicina de la Universidad de Concepción, requirió comenzar el desarrollo de su aplicación móvil "StressMApp" (stress map application). Este software permitirá recabar información en forma de auto-reportes y captura de comportamiento en un contexto de investigación médica sobre el estrés en estudiantes.

La aplicación móvil consiste de 3 tests en formato juego, con factores que evalúan el rendimiento cognitivo e inducen estrés al participante. Previo y posterior a los juegos, el participante debe completar una serie de formularios para conocer su estado emocional. Esto, junto al comportamiento capturado del juego, deben ser enviados a un backend que permita almacenar esta información en una base de datos remota para su posterior análisis. Los tests son los siguientes.

Test de memoria de trabajo (N-Back):

A los participantes se les pide que completen la tarea de memoria de trabajo n-back, que consiste en el seguimiento de letras (mayúsculas y minúsculas) presentadas secuencialmente con el objetivo de clasificar la letra presentada en pantalla, como la misma o diferente a la presentada hace uno, dos o tres símbolos atrás dentro de la secuencia. Se presentan un total de 16 bloques, con cada estímulo presentado durante 3 segundos, dando 2 segundos para responder, y breves períodos de

descanso entre bloques. Se registran las respuestas de los participantes ("igual" o "diferente"), y su tiempo de reacción.

Test inductor de estrés (imágenes aversivas):

Los participantes se dividen en 2 grupos, con condición controlable e incontrolable respectivamente. En cada ensayo, se presenta uno de cuatro posibles símbolos, donde cada uno de ellos tiene una respuesta correcta asociada, luego, el participante debe escoger una de las respuestas disponibles, y al escoger, se le muestra una imagen. Algunas de las imágenes son aversivas/negativas, otras neutrales. Los participantes en la condición controlable pueden evitar la presentación de una imagen desagradable aprendiendo (por ensayo y error) cuál es la respuesta correcta a cada una de las señales visuales presentadas. En la condición incontrolable, los participantes experimentan un patrón de eventos similar, pero no pueden evitar los estímulos negativos.

Test de exploración y predicción:

Esta tarea de "exploración" se desarrolla en el contexto de un viaje en avión entre tres islas, donde el participante tiene como objetivo aprender dónde volará el avión y que piloto está volando. Puede aprender esto en base a algunas reglas que se le enseñan antes y durante el juego. Los participantes saben que hay dos pilotos diferentes, con rutas de vuelo específicas dependiendo del avión, tres aviones diferentes y tres destinos posibles. Hay distintos tipos de pruebas, una de estas es de exploración, donde al participante se le muestra una isla de inicio y se le permite escoger entre 2 aviones, luego se le muestra la isla de llegada del avión. Otro tipo de prueba, es de predicción de isla y de piloto, donde se le pregunta a dónde cree que irán los aviones mostrados (predicción del estado) y también qué piloto cree que está volando (predicción de la condición) respectivamente. Además, a lo largo de la tarea existe la condición de controlabilidad e incontrolabilidad, es decir, si se le muestra información correcta o incorrecta respectivamente.

Cuestionarios:

Durante el juego, el usuario debe completar una gran cantidad de cuestionarios, con datos demográficos, sobre eventos estresantes de su vida, y diversos tests que evalúan el estado de ánimo, nivel de estrés y ansiedad. Además, debe completar un cuestionario de consentimiento informado al inicio del juego.

1.2 Objetivo General

Desarrollar el backend y base de datos de un prototipo funcional de la aplicación móvil, que permita a los investigadores obtener información relacionada con los experimentos y cuestionarios realizados por el usuario. Además, se desarrollará una plataforma web que permita visualizar y obtener los datos obtenidos por el prototipo, de modo que sea accesible para los investigadores u otros usuarios autorizados.

La información de los participantes debe ser encriptada y resguardada, ya que esta es considerada información sensible. El usuario será anónimo, de esta forma, solo se obtendrán sus datos personales e información de contacto en caso de necesitar comunicarse con el comité de ética o los investigadores de forma directa.

1.3 Objetivos Específicos

1. Gestionar los requerimientos y configuración del servidor en el que se aloja la aplicación y sistema web para asegurar su correcto funcionamiento.
2. Diseñar, implementar y testear la base de datos con la que se almacenará la información de cuestionarios y tareas cognitivas, además de mantener registro del comportamiento del usuario, tal como tiempos de respuesta y permanencia en la aplicación. Se usará una implementación de base de datos local para evitar perder los avances en los distintos cuestionarios y tareas en las que se encuentre el usuario.

3. Diseñar, implementar y testear la lógica (Backend) para la comunicación entre el servidor y la aplicación, esto implica la implementación de:
    - Rutas necesarias para la obtención de los datos requeridos por la interfaz gráfica (Frontend app móvil y sistema web).
    - Rutas para el procesado y almacenamiento de las respuestas de cuestionarios y tareas.
    - Funciones necesarias para el procesado y obtención/almacenamiento de datos.
    - Un método de encriptación para el resguardo de la información.

1.4 Metodología

Dado que este proyecto está enfocado en el desarrollo de software, específicamente una aplicación móvil y sistema web desde cero, se ha formado un equipo de tres personas, asignando las tareas del desarrollo de las interfaces para los test, cuestionarios y el sistema web a los otros dos integrantes del equipo.

Este proyecto se realizó de manera iterativa e incremental con enfoque ágil, atendiendo a su vez, los requerimientos que puedan ir surgiendo a medida que se desarrolla el proyecto. Para ello, se realizaron reuniones periódicas de manera semanal (jueves al mediodía) con el equipo multidisciplinario. Dicho esto, el proyecto se divide de la siguiente forma:

- Contratación y configuración del servicio de hosting.
- Implementación del Backend y testing de los siguientes componentes.
    - Test “N-Back”.
    - Cuestionario sociodemográfico.
    - Test inductor de estrés.
    - Cuestionario sobre eventos estresantes.
    - Test de exploración y aprendizaje.
    - Sistema web.
- Despliegue y testing con usuarios.

Las tecnologías utilizadas en este proyecto se han escogido en base al previo conocimiento por todo el equipo de desarrollo en proyectos pasados, de esta forma aseguraremos que se puedan cumplir los objetivos propuestos, estas tecnologías son:

- Control de versiones
    - Git.
    - GitHub.
- Lógica (Backend).
    - Laravel 9 (PHP 8.0 - 8.1 & Composer).
- Base de datos.
    - PostgreSQL.
    - expo-SQLite (local).

Además, se utilizaron las siguientes plataformas de interfaz gráfica por parte del equipo, comunicadas con el backend Laravel desarrollado.

- React native expo (App móvil).
- Laravel (Web).

El resto de este informe, se estructura de la siguiente forma:

- **Capítulo 2:** se describe el marco teórico de esta memoria, de esta forma contextualizando el problema a abordar, detallando el funcionamiento de los test y abordando los conceptos y tecnologías que se utilizaron.
- **Capítulo 3:** se describe de manera general la solución propuesta, mostrando la arquitectura general utilizada y la metodología con la que se aborda el desarrollo.
- **Capítulo 4:** se describe con más detalle el proceso de diseño e implementación de la solución, mostrando los distintos aspectos a considerar en el diseño, requerimientos, diagramas y detalle técnico.
- **Capítulo 5:** se describe la metodología utilizada para evaluar la solución, siendo esta en base a pruebas de rendimiento, mostrando los resultados y conclusiones obtenidas.
- **Capítulo 6:** se describe el cierre del informe, detallando las dificultades, aprendizajes, objetivos cumplidos y trabajo futuro.

2. MARCO TEÓRICO

2.1 Descripción de los test

Para comenzar, en esta sección se detalla el funcionamiento de los test y la información requerida por los investigadores para realizar sus estudios, donde cabe destacar que los test se conforman de ciertas estructuras que son relevantes a la hora de diseñar la organización de la información requerida, estas son las siguientes.

- **Run:** es una pasada completa por el test, pueden ser múltiples y pueden estar conformadas por uno o más bloques.
- **Bloque:** es un conjunto de pruebas (1 a n pruebas) dentro de un Run.
- **Prueba:** es la prueba a responder presentada al usuario participante.

Tomando esto como referencia, se puede facilitar el diseño al ya poseer una idea de las relaciones entre cada estructura general de un test, permitiendo encasillar la información en la estructura que corresponda.

2.1.1 Test de memoria de trabajo (N-Back)

El test N-Back consiste en la memorización de una secuencia de letras mayúsculas y minúsculas a medida que se presentan en pantalla secuencialmente, con el fin de poder clasificar la letra presentada en pantalla como igual o distinta a la letra presentada hace una, dos o tres letras atrás en la secuencia. Esto es con el fin de medir la capacidad de memorización del participante, que se ve reflejada tanto en sus aciertos como en el tiempo de reacción, ya que, a mejor memorización el participante debería errar menos y responder de manera más rápida. La interfaz implementada por el equipo se puede ver en los anexos 8.19, 8.20 y 8.21.

Además, este test originalmente se compone de un run de 16 bloques con 25 pruebas por cada uno, cada bloque teniendo una clasificación distinta, es decir, clasificar la letra presentada N letras atrás con un N distinto para cada bloque.

2.1.2 Test inductor de estrés (Stress Induction)

Este test consiste en el aprendizaje de la relación entre señales y botones mediante un sistema de recompensas y castigos, donde cada señal es una figura de un color. Se conforma de un solo run, sin bloques y con múltiples pruebas, donde en cada una se presenta una de tres señales y 4 botones. La interfaz implementada por el equipo se puede ver en los anexos 8.22, 8.23 y 8.24.

Cada señal está relacionada a un botón, entonces el participante debe apretar el botón que se relaciona a la señal mostrada, luego, dependiendo de si la respuesta es correcta o incorrecta, se le muestra una imagen agradable o neutral, o una imagen desagradable respectivamente, así, se espera que el participante aprenda esta relación al tratar de evitar recibir la imagen desagradable.

Además, el test se agrupa por parejas de participantes que comparten ciertas cualidades sociodemográficas, donde existe un participante con condición controlable y el otro incontrolable, que es la que decide si el participante recibe el test normal o una réplica del test de la pareja controlable asignada, provocando que el participante reciba una secuencia predeterminada de estímulos independiente de sus respuestas. De esta forma, se busca establecer si dos personas con características sociodemográficas similares poseen capacidades similares de aprendizaje.

2.1.3 Test de exploración y predicción (Explore and Predict)

Este test se realiza en un contexto de viajes en avión entre tres islas distintas, con el objetivo de que el participante aprenda el patrón que sigue el avión para poder contestar qué piloto lo vuela. Existen tres islas, tres aviones con color distinto y dos pilotos, cada piloto sigue una ruta específica, con un piloto guiándose por el color del avión para decidir a qué isla llegar y el otro siguiendo una ruta fija independiente del color. La interfaz implementada por el equipo se puede ver en los anexos 8.25, 8.26, 8.27 y 8.28.

Se conforma de múltiples run con múltiples bloques, cada bloque tiene múltiples pruebas de tres tipos distintos, estas son:

- **Prueba de exploración:** consiste en mostrarle al participante una isla de partida y dos aviones para elegir, luego, al escoger un avión se le muestra la isla de llegada, permitiendo que el participante aprenda el patrón, además, se agrega una condición de incontrolabilidad en algunas pruebas, provocando que se le muestre una isla de llegada errónea con el objetivo de dificultar el aprendizaje.
- **Prueba de predicción de isla:** consiste en mostrarle al participante un avión, la isla de partida y dos islas para elegir, con el objetivo de que el participante escoja la isla de llegada, luego, al contestar puede que se le muestre un cofre del tesoro lleno o vacío, indicando si su respuesta fue correcta o incorrecta respectivamente, también puede que no se muestre el cofre.
- **Prueba de predicción de piloto:** consiste en una única prueba en la que el objetivo del participante es contestar qué piloto está pilotando el avión, al contestar se le mostrará un cofre del tesoro lleno o vacío dependiendo de si la respuesta fue correcta o incorrecta.

2.1.4 Cuestionarios sociodemográficos

Los cuestionarios sociodemográficos cumplen el objetivo de recopilar información personal del participante relacionada a su situación demográfica y eventos estresantes de su vida, además hay cuestionarios que evalúan el estado de ánimo, niveles de estrés y ansiedad del participante antes y después de contestar un test. Todo esto con el objetivo de estudiar la relación entre su situación sociodemográfica y emocional con el desempeño en los test. Estos cuestionarios se conforman de preguntas y respuestas, que pueden ser de selección múltiple o una respuesta con texto. A diferencia de los test, los cuestionarios no poseen runs, ni bloques, ni pruebas en sí, pero se puede considerar el cuestionario como un run que se conforma de múltiples pruebas, que serían las preguntas, pero cabe destacar que las preguntas no varían, por lo que solo sirven como una relación entre las respuestas y el cuestionario.

2.2 Conceptos

En esta sección se explican los conceptos de manejo de información necesarios para diseñar, implementar y probar la solución propuesta. Además, se explica un concepto de seguridad que es relevante a la hora de diseñar e implementar la comunicación entre app móvil y servidor web.

2.2.1 Schemas

Los "*Schemas"* funcionan como un "plano" de la estructura de una base de datos relacional, que permite definir cómo está organizada la información, es decir, define los nombres de las tablas, sus campos, tipos de datos y relaciones entre ellas.

2.2.2 Migraciones

Las migraciones (o migraciones de *Schemas*) permiten definir los elementos de una base de datos relacional (tablas, campos y sus relaciones) de una manera programática, de esta forma, se puede mantener un control de versiones de la base de datos. En Laravel, estas migraciones están definidas con una interfaz orientada al objeto, permitiendo definir las funciones "*up"* y "*down*" que crean las tablas *(Schemas*) y las eliminan, respectivamente, permitiendo también realizar operaciones más complejas.

2.2.3 Modelos

Los modelos son una entidad lógica que facilita la realización de operaciones sobre una base de datos relacional (lectura y escritura) utilizando orientación al objeto, vinculando una tabla con un modelo mediante el uso de un ORM (Object Relational Mapping).

2.2.4 ORM (Object Relational Mapping)

Un ORM o Mapeo Relacional de Objetos, es un modelo de programación que permite mapear las estructuras de una base de datos relacional y vincularla a entidades lógicas (*modelos*). En otras palabras, permite realizar acciones sobre una base de datos relacional sin la necesidad de utilizar SQL directamente.

2.2.5 CSRF-Token

La vulnerabilidad CSRF (Cross-site request forgeries o falsificación de peticiones entre sitios) consiste en la realización de peticiones a un sitio desde otro con una intención maliciosa. Por ejemplo, si una aplicación web posee una ruta de modificación de email "*usuario/email"* que acepta una petición POST con un campo email solamente, desde otro sitio, se podría crear un formulario que apunte a la ruta de modificación de email y modifique el email de un usuario sin él saberlo. Para prevenir esta vulnerabilidad, se necesita inspeccionar las peticiones POST, PUT, PATCH o DELETE entrantes mediante una código de sesión único al que el sitio malicioso no tenga acceso, o sea, el CSRF-Token.

2.2.6 Pruebas de carga

Las pruebas de carga o Load testing consisten en pruebas de rendimiento para una aplicación, con el fin de medir los límites en los que esta opera eficientemente y sin errores mayores, asegurando el correcto funcionamiento en un ambiente real. Para esto, se definen distintos tipos de prueba o escenarios, donde se especifica la carga o tráfico constante que recibirá la aplicación y durante cuánto tiempo funcionará este tráfico. A continuación, se detallan los escenarios más comunes para probar una aplicación.

- **Smoke test (Prueba de humo):** Consiste en un test de baja carga para verificar que, tanto el sistema como la prueba de rendimiento, funcionan correctamente.

- **Load test (Prueba de carga):** Consiste en un test de carga media para medir el rendimiento de la aplicación en un escenario de uso normal. Este se subdivide en 3 etapas, que son las siguientes.
    - **Ramp-up o subida del tráfico:** consiste en incrementar el tráfico gradualmente durante un tiempo determinado.
    - **Mantención de carga promedio:** consiste en mantener un tráfico constante durante un cierto tiempo.
    - **Ramp-down o bajada del tráfico:** gradualmente reduce el tráfico durante cierta cantidad de tiempo.
- **Stress test (Prueba de estrés):** Basicamente es un Load Test incremental, es decir, se aplican los mismos conceptos del Load Test múltiples veces, incrementando el tráfico a cada iteración con el fin de medir el rendimiento bajo una carga mayor y los límites de tráfico de la aplicación.
    - Se pueden realizar un número arbitrario de etapas de Ramp-up y mantención, finalizando con una etapa de Ramp-down.

2.3 Tecnologías

En esta sección se detallan las tecnologías utilizadas para la implementación de la solución acorde a los conceptos detallados anteriormente y los requerimientos de información, con el fin de satisfacer todos los requerimientos funcionales por parte de los investigadores y proveer una funcionalidad estable. En base a esto, se ha escogido el framework Laravel 9 para implementar la solución de manera rápida y segura, además, se explica Docker, Node Exporter, cAdvisor y k6, que se escogen con el fin de implementar la solución en un ambiente similar al de producción y hacer las pruebas de carga especificadas, para asegurar el correcto funcionamiento de la solución.

2.3.1 Laravel 9

Es un framework basado en el lenguaje PHP para el desarrollo de aplicaciones web y APIs, con sintaxis “elegante”, que provee de las funcionalidades necesarias para facilitar el desarrollo, evitando enfocarse en detalles que desvíen del foco principal de la aplicación. Sus características más relevantes son las siguientes.

- **Eloquent ORM:** Facilita las interacciones con la base de datos al utilizar un ORM donde asigna una tabla a un modelo.
- **Arquitectura MVC (Modelo Vista Controlador):** estilo de diseño que separa la interfaz de usuario, los datos y la lógica de negocio en tres componentes distintos. De esta forma, el usuario interactúa desde la vista con los controladores, que permiten el funcionamiento de la aplicación, y estos interactúan con los modelos, gestionando la información de la base de datos para completar las operaciones del negocio.
- **Seguridad:** Posee soluciones de seguridad para una variedad de vulnerabilidades, entre las más conocidas, la inyección SQL y CSRF (falsificación de peticiones entre sitios).

2.3.2 Laravel Sanctum

Funciona como un sistema de autenticación para SPAs (Single Page Applications), aplicaciones móviles, o APIs basadas en token. Permite que cada usuario genere múltiples API Tokens con distintas habilidades y alcances. Sanctum existe para resolver dos problemas:

- **API Token:** permite generar tokens de acceso personal para los usuarios o aplicaciones, permitiendo autenticar y administrar las operaciones que ese usuario puede realizar. Esto se logra mediante el almacenamiento de estos tokens en una tabla en una base de datos, validando los registros con las cabeceras de la petición HTTP.
- **Autenticación de SPAs:** permite autenticar aplicaciones de terceros que intentan interactuar con la aplicación, usando el sistema *“Authentication guard”* de Laravel. Este provee de protección CSRF, autenticación de sesión, entre otras, permitiendo verificar si la petición posee una cookie de autenticación o una API Token válida.

2.3.3 Docker

Docker es una plataforma para el desarrollo, despliegue y ejecución de aplicaciones en ambientes controlados llamados *containers* (contenedores). Estos se asemejan a lo que es una máquina virtual, utilizando las capacidades de virtualización del kernel de Linux que permiten desplegar aplicaciones en estas simulaciones de ambientes de producción, asegurando el buen funcionamiento en un ambiente real. Estos contenedores son creados a base de imágenes, que son instrucciones que permiten la construcción de estos contenedores, definidos en archivos *Dockerfile*, permitiendo al usuario definir todo lo que la aplicación necesita para su correcto funcionamiento (CPU, memoria, bibliotecas, etc.).

2.3.4 Grafana

Grafana es un software libre desarrollado por *Grafana Labs,* que permite el análisis de datos de rendimiento de una aplicación, mediante la obtención de métricas desde múltiples fuentes, unificándolas en uno o múltiples tableros (dashboard) con gráficos para su fácil visualización. Posee integraciones con distintas herramientas de obtención de datos, tales como las presentadas a continuación.

2.3.5 Prometheus

Prometheus es un sistema de monitoreo open-source desarrollado por Soundcloud en 2012 y apoyado por Grafana Labs, permitiendo una integración directa con Grafana. Este sistema incluye un modelo de datos multidimensional con un poderoso lenguaje de consultas llamado PromQ. Además, posee una base de datos de series de tiempo eficiente, permitiendo la recopilación de datos variados y de distintas fuentes. Una de las herramientas utilizadas por Prometheus para recopilación de información es *Node Exporter*.

2.3.6 Node Exporter

Es un sistema open-source de monitoreo y alerta de series de tiempo para ambientes nativos en la nube, escrito en Go. Puede recolectar métricas a nivel de nodo, es decir, por cada máquina física o virtual perteneciente a un grupo o red, y almacenarlas como series de tiempo con marcas de tiempo.

2.3.7 K6

Es una herramienta de tests de carga open-source desarrollado por *Grafana Labs* y la comunidad, que facilita y hace más productivo el testing de rendimiento para aplicaciones web. De esta forma, permite analizar si la aplicación desarrollada está preparada para ser lanzada a un ambiente de producción, corrigiendo errores y problemas de recursos o rendimiento que se puedan presentar.

2.3.8 cAdvisor

"cAdvisor" o Container Advisor es una herramienta de monitoreo de recursos y rendimiento de contenedores open-source. Es un daemon (proceso en segundo plano) que recolecta, agrega, procesa y exporta información sobre los contenedores en ejecución. Tiene soporte nativo para contenedores Docker por lo que la solución Prometheus puede recopilar esta información exportada y ser presentada en algún tablero de monitoreo como Grafana.

3. DESCRIPCIÓN DE LA PROPUESTA

Como se explicó anteriormente, el propósito del proyecto StressMApp es la recopilación de datos mediante una aplicación móvil para su posterior análisis, para estudiar si el estrés en estudiantes universitarios influye en su capacidad cognitiva. De esta forma, el objetivo general de esta memoria de título es la implementación de la obtención y almacenamiento de estos datos para su posterior visualización y análisis.

Para lograr los objetivos especificados, se ha desarrollado una API web (Application Programming Interface) con el framework web Laravel en su versión 9, en conjunto con una base de datos PostgreSQL para el almacenamiento y consulta de datos, y una base de datos SQLite para el almacenamiento local de las respuestas y datos de autenticación en la aplicación móvil. Se siguió una metodología iterativa e incremental con un enfoque ágil, interactuando constantemente con los clientes para la obtención, consolidación y validación de los requerimientos especificados para cada test. De esta forma, se abordó el problema descomponiéndolo en tareas más pequeñas, implementando un test a la vez y validando los prototipos con los clientes.

3.1 Arquitectura

En cuanto a la arquitectura del sistema, esta se enfoca principalmente en la interacción del usuario participante desde la aplicación móvil StressMApp con la API web para su registro, autenticación y envío de respuestas de test y cuestionarios que se almacenan en la base de datos. Luego, la API se comunica directamente con el Frontend del sistema web para entregar los datos a visualizar por los investigadores del proyecto. En la figura 3.1 se puede observar el diagrama de la arquitectura del sistema.

Como se define en los objetivos específicos de la memoria de título, el objetivo en general es la implementación del servidor web y el diseño e implementación de las bases de datos, lo que involucra también el desarrollo de aspectos funcionales en la persistencia de datos de la app móvil tales como el almacenamiento y obtención de la información desde la base de datos para ser enviada a la API posteriormente.

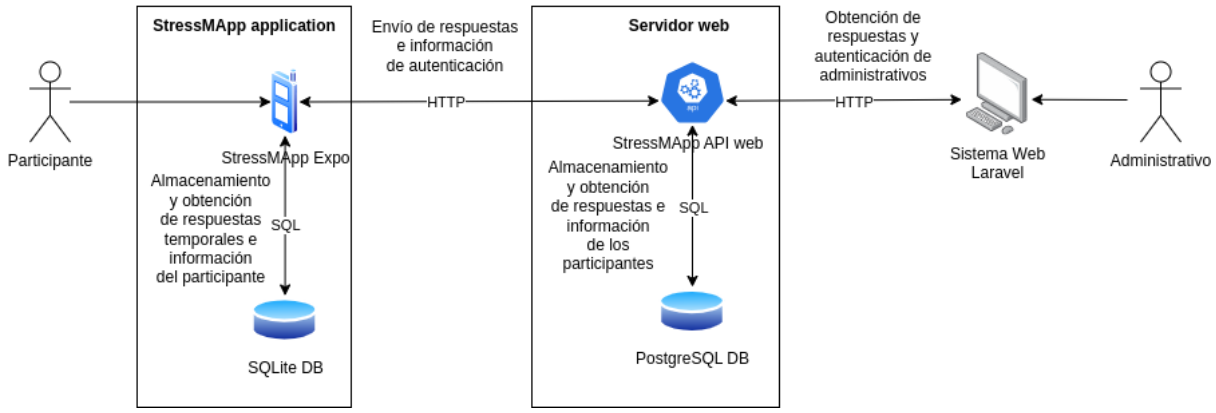


Figura 3.1: Arquitectura del sistema StressMApp.

La Arquitectura se divide en tres grandes secciones, que son las siguientes.

- **StressMApp application**: esta sección representa los componentes utilizados para el funcionamiento de la app móvil. Estos componentes son.
    - **StressMApp Expo:** Es la aplicación móvil en sí, implementada con Expo, que es una plataforma de desarrollo de aplicaciones móviles con React Native y Javascript. Su funcionamiento radica en la recepción de la información del participante que interactúa directamente con la interfaz, e información enviada desde el servidor para complementar el funcionamiento de la app, además realiza el envío de las respuestas e información del participante hacia el servidor para su procesado y almacenamiento.

    - **SQLite DB:** Es el almacenamiento local de la app móvil, implementado con una base de datos SQLite provista por la biblioteca expo-sqlite, con el fin de resolver la problemática de cómo obtener las respuestas del participante ante la pérdida de conexión a internet en su dispositivo móvil, lo que provocaría que se pierda esa valiosa información para los investigadores. De esta forma, cada respuesta se almacena localmente y al retomar la conexión se envían a la API web.
- **Servidor Web:** Esta sección representa los componentes que conforman la API web en el servidor. Los componentes son.
    - **StressMApp API web:** Es la API web en sí, implementada con el framework Laravel en su versión 9 en conjunto a otras bibliotecas de Laravel para proveer seguridad y una implementación de una interfaz web de gestión. Su funcionamiento radica en la recepción y envío de peticiones desde y hacia la aplicación móvil y el sistema web, además de comunicarse con la persistencia de datos implementada.
    - **PostgreSQL DB:** Es la persistencia de datos, implementada con la tecnología PostgreSQL que es una base de datos relacional. Su implementación es similar a la base de datos de la app móvil, con la diferencia de que al recibir respuestas de múltiples usuarios participantes es necesario establecer las relaciones entre estos y sus respuestas a los test, siguiendo un patrón similar a "*participante contesta run, run tiene bloques, bloques tienen respuestas"*. Además, se incluyen tablas que almacenan información sobre los elementos utilizados en los test.
- **Sistema web Laravel:** Este sistema es la interfaz web de visualización y gestión de la información almacenada en la API. Su funcionamiento radica en recibir información desde la API para ser presentada a los investigadores o usuarios autorizados.

3.2 Metodología de desarrollo

La metodología utilizada para estas implementaciones se basó en agilidad, teniendo reuniones semanales con los clientes para la toma de requerimientos, validando prototipos funcionales del almacenamiento y obtención de la información. Además, junto con el equipo se tomó la decisión de priorizar la implementación de los test por sobre los cuestionarios y otras funcionalidades, para lograr recopilar datos lo antes posible desde la app móvil, siguiendo un orden según la estimación de la complejidad de los test. Entonces, el orden escogido fue el siguiente.

1. Test N-Back.
2. Test Explore and Predict (Exploración y predicción).
3. Test Stress Induction (Inducción de estrés).
4. Registro y autenticación de participantes y administrativos.
5. Cuestionarios Sociodemográficos.

En cuanto a las implementaciones, se basó en una metodología iterativa e incremental de la siguiente manera:

- **Fase 1: Toma de requerimientos**
    - En esta fase se obtiene información sobre los datos que se deben almacenar, sus tipos de datos, y funcionalidades que deben haber.
- **Fase 2: Diseño de base de datos (servidor y local)**
    - En esta fase se realiza el diagrama E-R para modelar la base de datos del servidor, describiendo sus entidades, relaciones y atributos basándose en la información obtenida de la fase 1. Luego, esto es validado con el cliente y se agregan, corrigen o quitan elementos.
    - Se implementan las migraciones, en las que se definieron los *schemas* que crean las tablas correspondientes para el correcto funcionamiento de los test y sistema web. Además, se especifican los modelos, los cuales contienen en su nombre qué tabla representan, de esta forma es más intuitivo el saber con qué tabla estamos interactuando al hacer las operaciones proporcionadas por estos.
    - En cuanto a la base de datos local, no se requirió de un modelado específico, ya que la complejidad estructural es menor al solo trabajar con la información de un

participante. De este modo, se crea una tabla que almacena la información de cada estructura relevante usada por un test (Run, Bloque, respuestas e info de pruebas).

- **Fase 3: JSON de prueba**
    - Esta fase se enfoca en la estructura del cuerpo de la petición HTTP que recibirá la API con las respuestas de un test o información de un participante. Dentro de esta fase ocurre el diálogo con los clientes y el desarrollador de la app móvil, para validar los datos y consensuar la estructura en que se enviarán.
- **Fase 4: Implementación y testing**
    - Al validar la fase anterior, se realiza la implementación de las funcionalidades que provee la API mediante controladores (principalmente el almacenamiento y obtención de la información desde la app móvil) basándose en la estructura del JSON definido.
    - Cada controlador implementado está asociado a un test, cuestionarios o usuarios, conteniendo las funciones necesarias para procesar las peticiones enviadas por el usuario y obtenerlas desde la base de datos para enviarlas a un Frontend.
    - El testing se realizó con la herramienta Postman (software que permite realizar peticiones HTTP). Además, se implementaron vistas blade.php para la visualización de las respuestas almacenadas en la base de datos, para verificar que se almacena la información correctamente y las consultas realizadas poseen la estructura deseada y los atributos relevantes.
- **Fase 5: Testing con app móvil**
    - Al validar que la API funciona correctamente, se realiza el testing correspondiente con la app móvil, que implica responder el test a probar y enviar los datos hacia la API (haciendo uso de ngrok, software que permite transformar el entorno local a un entorno web, permitiendo la comunicación desde apps móviles sin necesidad de un servidor web). De esta forma, se procede a enviar las respuestas de forma tradicional (al terminar un test), y de la forma “asíncrona”, que implica enviar los datos de manera posterior, haciendo uso de la base de datos local para obtener los datos y estructurarlos.

Al completar la fase 5, el prototipo se encuentra funcional y está listo para presentarse a los clientes de forma final y se procede a la implementación del siguiente test.

4. DISEÑO E IMPLEMENTACIÓN

En esta sección se detalla el diseño e implementación de la base de datos de la app móvil y del servidor para cada test, junto a la respectiva implementación de la lógica de la API web en base a los diseños de las bases de datos. Además, se detalla la configuración inicial del servidor en el que está alojada la API de StressMApp.

4.1 Configuración del servidor

Para el despliegue de la API web a un ambiente de producción, se utilizó un servidor provisto por la Facultad de Ingeniería y la DTI de la Universidad de Concepción, siendo nombrado *stressmapp.inf.udec.cl* accesible a través del puerto 8005, con sistema operativo Ubuntu 20.04 (Linux) y servidor HTTP Apache 2. De esta forma, se transfirió la API a través de SFTP (SSH File Transfer Protocol) a los directorios internos de apache2, en específico, la ruta */var/www/html/stressmappi_backend.* Luego, se creó una configuración de sitio para la API con nombre "*stressmapp.conf*", la que contiene las directivas para escuchar en un puerto en específico (8005 en este caso), el nombre de servidor y la ruta donde se encuentra el archivo index, que permite ejecutar la aplicación. A continuación, se presenta el archivo de configuración del sitio en la figura 4.1.

```
<VirtualHost *:8005> //port 8005

    ServerAdmin admin@stressmap.inf.udec.cl //admin mail
    ServerName stressmapp.inf.udec.cl //name of the server
    DocumentRoot /var/www/html/stressmapp_api/public/ //route of the index.php file
    <Directory /var/www/html/stressmapp_api>
        Options Indexes FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined

</VirtualHost>
```

Figura 4.1: Archivo de configuración del sitio "stressmapp.conf".

Posterior a esta configuración inicial, se conceden permisos de lectura y escritura para los directorios storage y bootstrap de la aplicación. Finalmente, se habilitó el sitio con comandos de apache y se generó memoria caché para las vistas, rutas y configuración de la API, mejorando su rendimiento.

4.2 Persistencia de datos en aplicación móvil

Además de la implementación de la base de datos en el servidor, también se trabajó en la implementación de la persistencia local en la app móvil, en la que la estructura de las entidades de cada test y los campos que deben existir resultan ser una versión simplificada de lo que es la base de datos del servidor. A continuación se detalla el diseño e implementación.

4.2.1 Diseño

El diseño de la base de datos de la app móvil se desprende de lo descrito en el funcionamiento de los test y de lo conversado con los investigadores. Dicho esto, en el diseño no fue necesario establecer un esquema relacional tan fuerte como el del servidor, esto debido a que en esta

persistencia de datos solo existe un participante con sus propias respuestas a los test, de modo que se pueden simplificar las relaciones.

Algo importante que agregar, es la implementación de una tabla llamada “participante”, que contiene la información necesaria para autenticar al participante en la API web. En la tabla 4.1 se detalla el diseño del participante.

Tabla	**Campo**	**Descripción**
participante	userID	Número identificador único del participante.
	token	Token único de autenticación, permite identificar al usuario participante y permitirle acceder a la información requerida por los test, además de permitirle enviar sus respuestas.

Tabla 4.1: Tabla participante y sus campos de la base de datos de la app móvil.

4.2.2 Implementación

La implementación se realizó con la biblioteca expo-sqlite, que entrega persistencia de datos con SQLite en el dispositivo móvil. Dentro de SQLite existen las funciones “transaction” que permiten realizar una transacción asíncrona con la base de datos, y dentro de estas funciones de transacción, se definen funciones “executeSql” que permiten ejecutar una consulta sql en la base de datos de manera asíncrona. De esta forma se implementan las funcionalidades para crear las tablas con sus respectivos campos, y la obtención y almacenamiento de datos. De esta forma se implementan 3 tipos de funciones con consultas sql para cada test, que son las siguientes.

- **Creación de tablas:** consiste en ejecutar una consulta sql “CREATE TABLE IF NOT EXISTS” para crear una tabla en la base de datos. Para ello se debe especificar el nombre de la tabla, sus campos y su tipo de dato respectivamente.
- **Almacenamiento de datos:** consiste en ejecutar una consulta sql “INSERT INTO” de inserción de datos en una tabla. Para ello, se debe especificar el nombre de la tabla, los campos a insertar y los valores a insertar.

- **Obtención de datos:** consiste en ejecutar una consulta sql de selección de datos. Para ello se debe usar una consulta SELECT con los campos a obtener, FROM el nombre de la tabla y una condición WHERE si es necesario.

De esta forma, se encuentran implementadas las funcionalidades de manejo de información de la base de datos de la app móvil.

4.3 Test N-Back

Este test fue el primero en implementarse dado por su baja complejidad en cuanto a la cantidad de estructuras, relaciones y datos a almacenar. Esto también simplifica la complejidad de la comunicación entre la API y la app móvil. Por ello, fue el primer paso para identificar la mejor manera de implementar los demás test, dado que la estructura general se repite.

4.3.1 Diseño

Las bases de datos de la app móvil y del servidor se diseñaron teniendo en consideración el cumplir con las necesidades de almacenamiento de información relevante requerida, esto es almacenar las respuestas para cada prueba presentada al participante y cierta información de las pruebas y otras estructuras, las que se detallan en la tabla 4.2.

Estructura	**Dato**	**Descripción**
Run	fecha_contestado	Fecha en la que se terminó el Run.
Bloque	num_bloque	Número de bloque dentro del run.
	n_type	Tipo de N del bloque, es decir, se debe clasificar la letra presentada N letras atrás como igual o distinta a la presentada en pantalla.
Prueba	num_bloque	Número de bloque en el que se presenta esta prueba.
	num_prueba	Número de prueba dentro del bloque.

	tipo_prueba	Si la prueba muestra una letra igual o distinta a la letra mostrada N letras atrás.
	score	Puntaje al responder correctamente, valor 1 si acierta o 0 si se equivoca o no contesta.
	match	Indica si el participante contesta que las letras son iguales.
	no_match	Indica si el participante contesta que las letras son distintas.
	miss	Indica si el participante no contesta.
	letra_mostrada	Indica qué letra se mostró en la prueba.
	tiempo_respuesta	Indica el tiempo en milisegundos que el participante demoró en contestar.
	fecha_inicio	Indica la fecha en que se habilita el botón para contestar, con precisión de milisegundos.
	fecha_respuesta	Indica la fecha en que el participante contesta la prueba, con precisión de milisegundos. Esta fecha junto a la fecha de inicio se utilizan para calcular el tiempo de respuesta, contrastando con tiempo_respuesta para verificar errores en la medición.

Tabla 4.2: Descripción de los datos requeridos en el test N-Back.

De esta forma, la base de datos de la app móvil se enfoca en almacenar la información definida, corresponde a los datos de cada prueba al momento de contestar. Para ello, se han implementado las siguientes tablas, detalladas en la tabla 4.3.

Tabla	**Descripción**
nback	Contiene las respuestas del test agrupadas por bloque.
nback_bloque	Almacena los bloques del test, sirve para agrupar las respuestas y construir la petición hacia la API.

Tabla 4.3: Tablas modeladas para el test N-Back en la base de datos de la app móvil .

La base de datos del servidor se modeló usando un diagrama entidad-relación, para validar con los clientes si es que los datos y relaciones descritas eran las correctas. En el anexo 8.1 se puede visualizar el resultado del modelado final. Se usó como guía para generar las migraciones y modelos

del test, donde se puede visualizar la estructura básica de estos. Entonces, el diagrama E-R del servidor se puede observar en la figura 4.2 y las entidades generadas se detallan en la tabla 4.4.

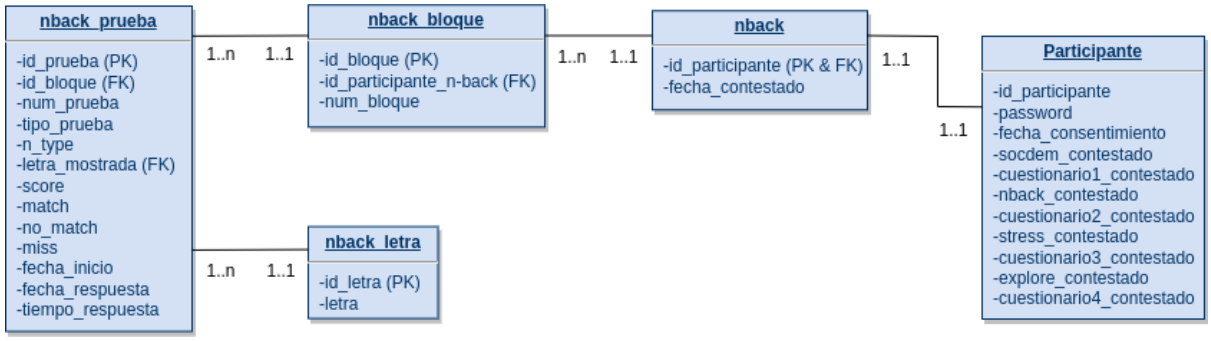


Figura 4.2: Diagrama E-R del test N-Back.

Tabla	Descripción
nback	Representa que el participante ha contestado el test, sirve para agrupar los bloques de respuestas por participante.
nback_bloque	Contiene información sobre los bloques del test por participante, agrupando conjuntos de respuestas.
nback_prueba	Contiene todas las respuestas del test, agrupadas por bloque.
nback_letra	Contiene las letras usadas en el test.

Tabla 4.4: Descripción de tablas implementadas en la base de datos para el test N-Back.

4.3.2 Implementación

Para lograr implementar lo modelado con el diagrama E-R, se hizo uso de las migraciones para definir y crear las tablas en la base de datos. En la figura 4.3 se muestra el código resultante de una migración, donde se hace uso de la clase *Schema* que permite definir una tabla en la base de datos y sus campos mediante una función.

```php
public function up(){
    Schema::create('nback_letra', function (Blueprint $table){
        $table->id('id_letra'); //1-26
        $table->char('letra'); //A-Z
    });
    Schema::create('nback', function (Blueprint $table){
        $table->foreignId('id_participante')->unique()->constrained('participante','id_participante');
        $table->string('fecha_contestado');
    });
    Schema::create('nback_bloque', function (Blueprint $table){
        $table->id('id_bloque'); //identificador
        $table->foreignId('id_participante_nback')->constrained('nback','id_participante');
        $table->unsignedTinyInteger('n_type');
        $table->unsignedTinyInteger('num_bloque'); //numero de bloque (1, 2, 3,...)
    });
    Schema::create('nback_prueba', function (Blueprint $table){
        $table->id('id_prueba'); //identificador
        $table->foreignId('id_bloque')->constrained('nback_bloque','id_bloque');
        $table->unsignedTinyInteger('num_prueba'); //numero de prueba dentro del bloque [1;20]
        $table->boolean('tipo_prueba');
        $table->foreignId('letra_mostrada')->constrained('nback_letra','id_letra');
        $table->integer('score');
        $table->boolean('match');
        $table->boolean('no_match');
        $table->boolean('miss');
        $table->string('fecha_inicio');
        $table->string('fecha_respuesta');
        $table->integer('tiempo_respuesta');
    });
}
```

Figura 4.3: Migración test N-Back.

Los modelos definidos se asignan a cada una de las tablas generadas en la migración, para facilitar las consultas a la base de datos. En la figura 4.4 se puede observar el código del modelo *NBack* y los atributos relevantes a especificar.

- ***protected $table*:** especifica el nombre de la tabla que estará asociada a este modelo.
- ***protected $primaryKey*:** la clave primaria por defecto es un campo *id* de nombre *id*, por lo que si tiene otro nombre, debe especificarse con este atributo.
- ***public $timestamps*:** especifica si se incluyen marcas de tiempo en la tabla, tales como *"creado en"* o *"actualizado en"*.
- ***public $incrementing*:** especifica si es que la clave primaria es auto incremental, en este caso es una clave foránea que apunta a un id de participante, entonces, no es auto incremental.
- ***protected $fillable:*** especifica cuales son los campos que son asignables en masa, es decir, los campos que se pueden almacenar con el método *create* del modelo.

```php
class NBack extends Model
{
    use HasFactory;
    protected $table = 'nback';
    protected $primaryKey = 'id_participante';
    public $timestamps = false;
    public $incrementing = false;
    protected $fillable = [
        'id_participante',
        'fecha_contestado'
    ];
}
```

Figura 4.4: Modelo NBack asociado a la tabla nback.

Las rutas definidas se basan en asignar una función a una ruta, implementada directamente en esta o llamando a una función externa. En este caso, se les asignaron las funciones definidas en el controlador de N-Back.

- **admin/nback/results:** se relaciona con la función *getResults* para obtener las respuestas del test N-Back.
- **api/nback/store:** se relaciona con la función *storeResults* para almacenar las respuestas del test N-Back.

El controlador implementado se llama *NBackController*, que contiene las funciones asociadas a las rutas definidas con anterioridad, las cuales son:

- **getResults**: función que obtiene las respuestas del test N-Back, agrupadas por bloque haciendo uso de las funcionalidades de *Eloquent* para generar la consulta SQL. Luego, estos datos son retornados a la vista *nback.blade.php* para visualizar si la consulta fue realizada correctamente.
- **storeResults**: función que almacena las respuestas del test N-Back recibidas en el cuerpo de una petición HTTP desde la aplicación móvil. En el anexo 8.2 se puede observar un fragmento del código, que corresponde al primer paso de la obtención de la información de la petición mediante el método *input* de la clase *Request*. Este permite obtener un valor dada la clave, así, se obtiene el id del participante y los bloques con respuestas. Luego, se hace uso del método *create* de los modelos con el atributo *$fillable* definido. De esta forma, se puede crear un elemento en la base de datos y almacenarlo en una sola llamada, esto dentro de un bloque try-catch para manejar errores SQL mediante la clase *QueryException*.

Al almacenar correctamente los datos recibidos desde la aplicación móvil, se usan los métodos *where* y *update* del modelo *Participante*, que obtiene los elementos de la tabla participante dado una condición (en este caso, que el campo *id_participante* sea igual a la variable *$id_participante*) y actualizar si es que este ha contestado nback satisfactoriamente.

4.4 Test Explore and Predict

El test de exploración y predicción fue el segundo en implementarse, y al ya tener nociones del patrón que siguen estos test, su complejidad disminuyó notoriamente, ya que, si bien posee más entidades que el test N-Back, estas siguen el mismo patrón de *"run tiene bloques y bloques tienen pruebas"*, además de añadir elementos complementarios que contienen información más descriptiva para los investigadores sobre los elementos presentados en el test.

4.4.1 Diseño

La base de datos fue pensada de manera similar a N-Back y en base a lo definido en la tabla 4.5, con la diferencia de que este test posee bastantes campos y relaciones un poco más complejas al existir tres tipos de prueba por bloque.

Estructura	**Dato**	**Descripción**
Run	num_run	Número de run.
	frec_reversion_cond	Frecuencia de reversión de condición controlable a incontrolable y viceversa, esta frecuencia se mide en número de pruebas.
	noise	Parámetro usado para calcular qué tan probable es que la isla de llegada correcta se modifique en las pruebas de exploración.
Bloque	num_bloque	Número de bloque dentro del run.
	reversiones	Indica la cantidad de reversiones de condición que han ocurrido hasta el bloque actual.

Prueba de exploración	bloque_prueba_explore	Número de bloque en el que se presenta esta prueba.
	num_prueba_condicion	Número de prueba con condición incontrolable dentro del bloque.
	num_prueba_explore	Número de prueba de exploración dentro del bloque.
	isla_llegada_afectada_noise	Indica si la isla de llegada se modificó dado por el cálculo con el parámetro noise.
	isla_mostrada	Indica la isla mostrada al participante.
	isla_llegada	Indica la isla de llegada al seleccionar uno de los aviones.
	avion_1	Indica uno de los aviones seleccionables mostrados.
	avion_2	Indica uno de los aviones seleccionables mostrados.
	avion_elegido	Indica el avión seleccionado por el participante.
	tiempo_respuesta	Indica el tiempo en milisegundos que el participante demoró en contestar.
	fecha_inicio	Indica la fecha en que se inícia la prueba, con precisión de milisegundos.
	fecha_respuesta	Indica la fecha en que el participante contesta la prueba, con precisión de milisegundos. Esta fecha junto a la fecha de inicio se utilizan para calcular el tiempo de respuesta, contrastando con tiempo_respuesta para verificar errores en la medición.
Prueba de predicción de isla	bloque_prueba_predict_island	Número de bloque en el que se presenta esta prueba.
	num_prueba_predict	Número de prueba de predicción de isla dentro del bloque.
	isla_mostrada	Indica la isla mostrada al participante.
	isla_respuesta	Indica la isla seleccionada por el participante.
	avion	Indica el avión mostrado al participante.
	tipo_respuesta	Indica si el participante contestó correctamente.
	recompensa	Indica si se le muestra el cofre al participante.
	tiempo_respuesta	Indica el tiempo en milisegundos que el participante demoró en contestar.

	fecha_inicio	Indica la fecha en que se inicia la prueba, con precisión de milisegundos.
	fecha_respuesta	Indica la fecha en que el participante contesta la prueba, con precisión de milisegundos. Esta fecha junto a la fecha de inicio se utilizan para calcular el tiempo de respuesta, contrastando con tiempo_respuesta para verificar errores en la medición.
Prueba de predicción de piloto	bloque_prueba_ predict_pilot	Número de bloque en el que se presenta esta prueba.
	piloto_respuesta	Indica el piloto que seleccionó el participante.
	tipo_respuesta	Indica si el participante contestó correctamente.
	tiempo_respuesta	Indica el tiempo en milisegundos que el participante demoró en contestar.
	fecha_inicio	Indica la fecha en que se inicia la prueba, con precisión de milisegundos.
	fecha_respuesta	Indica la fecha en que el participante contesta la prueba, con precisión de milisegundos. Esta fecha junto a la fecha de inicio se utilizan para calcular el tiempo de respuesta, contrastando con tiempo_respuesta para verificar errores en la medición.

Tabla 4.5: Descripción de los datos requeridos en el test Explore and Predict.

De esta forma, las entidades de la app móvil son detalladas en la tabla 4.6, que corresponden al almacenamiento de las respuestas para cada uno de los tipos de prueba presentados, e información sobre el run.

Tabla	**Descripción**
exp_pred_run	Contiene información sobre los run del test, sirve para agrupar los bloques y construir la petición hacia la API.
expr_pred_bloque	Contiene información sobre los bloques del run, que se componen de pruebas de exploración, predicción de isla y predicción de piloto.
exp_pred_exploratory	Almacena las pruebas de exploración, agrupadas por número de bloque y run.

exp_pred_predict	Contiene las pruebas de predicción de isla, agrupadas por número de bloque y run.
exp_pred_pilot_predict	Contiene las pruebas de predicción de piloto, agrupadas por número de bloque y run.

Tabla 4.6: Tablas modeladas para el test Explore and Predict en la base de datos de la app móvil .

Las entidades del servidor, toman una estructura similar a las definidas en la app, profundizando en las relaciones y en los elementos utlizados en las pruebas (aviones, pilotos e islas). En la figura 4.5 se puede observar el diagrama E-R del servidor y en la tabla 4.7 se detallan las entidades que existen en este test.

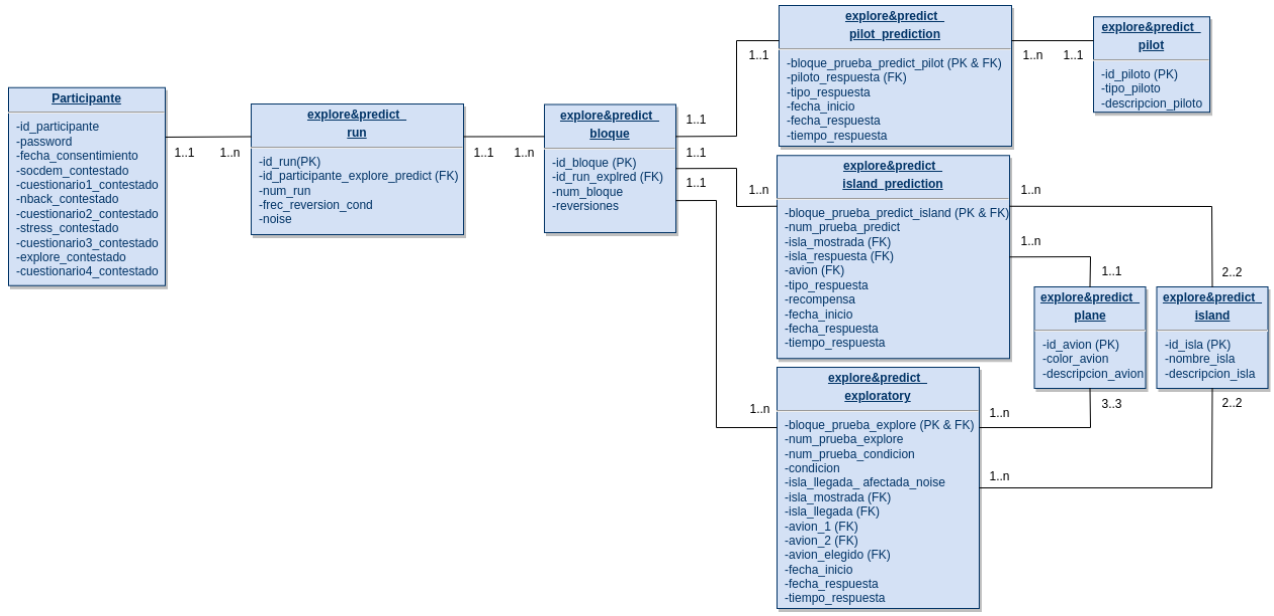


Figura 4.5: Diagrama E-R del test “Explore and Predict”.

Tabla	**Descripción**
explpred_run	Contiene información sobre un Run respondido por el participante, pueden ser múltiples y agrupan un conjunto de bloques.
explpred_bloque	Contiene información sobre las pruebas contestadas para un Run en específico, agrupando un conjunto de pruebas exploratory, island_prediction y pilot_prediction.

explpred_exploratory	Contiene información sobre las pruebas de exploración contestadas, son múltiples y pertenecen a un bloque.
explpred_island_prediction	Contiene información sobre las pruebas de predicción de isla contestadas, son múltiples y pertenecen a un bloque.
explpred_pilot_prediction	Contiene información sobre las pruebas de predicción del piloto contestadas, son únicas por bloque.
explpred_island	Contiene información sobre las islas utilizadas en las pruebas *exploratory* y *island_prediction*.
explpred_plane	Contiene información sobre los aviones utilizados en las pruebas *exploratory* y *island_prediction*.
explpred_pilot	Contiene información sobre los pilotos utilizados en las pruebas *pilot_prediction*.

Tabla 4.7: Descripción de tablas implementadas en la base de datos para el test Explore and Predict.

4.4.2 Implementación

La migración generada para este test consiste de las entidades vistas en el diagrama E-R, haciendo uso de la clase *Schema* para la generación de cada tabla. Al analizar la creación de la migración surge la problemática de que los run deben identificarse únicamente, esto es porque distintos run poseen el mismo id de participante, evitando que este sea clave primaria, por otro lado, el número de run se repite para varios usuarios (ya que todos contestan run número 1, 2, .., n). Entonces, la solución es generar una clave primaria compuesta, pero, algo a notar es que Laravel no maneja claves primarias compuestas para los modelos, por lo que la obtención de información se dificulta. Para resolver esto, se generó un identificador único llamado *id_run* que funciona como clave primaria permitiendo que otras entidades puedan referenciarla, y se ha hecho una restricción "*unique"* compuesta por *id_participante* y *num_run* indicando que no puede existir un run repetido para un usuario. En la figura 4.6 se puede observar el *Schema* resultante.

```php
Schema::create('explpred_run',function (Blueprint $table){
    $table->id('id_run');//identificador
    $table->foreignId('id_participante_explore_predict')->constrained('participante','id_participante');
    $table->unsignedBigInteger('num_run'); //numero de run del test
    $table->tinyText('frec_reversion_cond');
    $table->unsignedFloat('noise');
    $table->unique(['id_participante_explore_predict','num_run']);
});
```

Figura 4.6: Schema generador de tabla “explore_predict_run” con restricción unique compuesta.

Otro elemento importante que se incluyó en este test es el uso de *Seeders*, que su función principal es poblar una base de datos, entonces, como existen las entidades de isla, avión y pilotos que muy rara vez cambian al ser elementos intrínsecos del test, estos se pueden definir de manera previa en un *seeder*, dando información relevante de lo que se le presentó en el test al participante. Por ejemplo, los aviones tienen su número identificador, un color y una descripción, lo que en una prueba de predicción de isla implica que hayan 9 campos que dan información de los 3 aviones que se almacenan en una respuesta, en cambio, apuntar con una clave foránea a un avión evita que la tabla crezca demasiado en campos y que se ingresen aviones inexistentes.

Para la implementación de un seeder, se declara un arreglo con los elementos a insertar en la base de datos, luego, con la estructura de control *foreach* se itera por cada elemento del arreglo y se inserta en la base de datos usando la función *create* del modelo, generando una nueva instancia y guardándola.

Los modelos definidos se corresponden con cada tabla generada por la migración, y sus atributos para su correcto funcionamiento son: *$table*, *$primaryKey*, *$timestamps* y *$fillable*. En la figura 4.7 se puede ver la definición del modelo ExplorePredictRun.

```php
class ExplorePredictRun extends Model
{
    use HasFactory;
    protected $table = 'explpred_run';
    protected $primaryKey = 'id_run';
    public $timestamps = false;
    protected $fillable = [
        'id_participante_explore_predict',
        'num_run',
        'frec_reversion_cond',
        'noise',
    ];
}
```

Figura 4.7: Modelo "ExplorePredictRun" del test "Explore and Predict".

Las rutas definidas para este test se correponden con respecto a las funcionalidades requeridas, que son el almacenamiento y obtención de respuestas.

- **admin/explorepredict/results:** se relaciona con la función *getResults* para la obtención de las respuestas.
- **api/explorepredict/store:** se relaciona con la función *storeResults* para almacenar las respuestas del test.

El controlador se llama *ExplorePredictController* y tiene implementadas las funciones especificadas en las rutas, que son las siguientes.

- **storeResults:** almacena las respuestas del test exploración y predicción en la base de datos. Esta función sigue el mismo procedimiento que la implementada para el test N-Back, es decir, obtención de información de la petición recibida desde la aplicación móvil, creación de un nuevo elemento mediante el método *create* del modelo, manejo de errores, repetir para cada bloque y cada prueba en el bloque, finalmente, actualizar que el participante ha contestado el test.
- **getResults:** obtiene todas las respuestas del test exploración y predicción almacenadas, divididas en 4 conjuntos agrupados por usuario participante y run. En el anexo 8.3 se puede observar el código para obtener los datos desde la base de datos usando *Eloquent* entregandolos a la vista *explore.blade.php* para visualizar si es que la consulta se ha hecho correctamente.

- **Runs:** contiene la información almacenada de todos los run.
- **ExploreResults:** contiene la información almacenada de todas las pruebas de exploración.
- **IslandPredResults:** contiene la información almacenada de todas las pruebas de predicción de isla.
- **PilotPredResults:** contiene la información almacenada de todas las pruebas de predicción de piloto.

4.5 Test Stress Induction

Este fue el último de los test en implementarse, ya que su complejidad radica más en la lógica de la app móvil, eso sí, surge una diferencia con los otros test, que es el requerimiento de emparejar participantes dependiendo de su género y año de estudio (1° año, 2° y 3° año, 4° y 5°+), para estudiar si es que su desempeño está relacionado a su situación de vida o es otro factor, asignando una condicion controlable a un participante, que indica que el test tendrá un mapeo entre señales y botones, mostrando las imágenes en base a si sus respuestas fueron correctas o no, en cambio, al participante con condición incontrolable se le presentará el mismo test que su pareja controlable, es decir, mismo mapeo de señales y botones, y la secuencia en que se le mostraron las imágenes, independiente de si contesta correctamente o no, permitiendo al participante controlable aprender en base a ensayo y error, y al otro no.

Además, este test muestra imágenes agradables o chocantes a los participantes dependiendo de sus condiciones o respuestas, entonces, se tomó la decisión de almacenar estas imágenes en el servidor y entregarlas a la app móvil cuando se requieran y eliminarlas al terminar su uso, de esta forma evitamos que los usuarios participantes puedan obtenerlas y hacer mal uso de ellas.

4.5.1 Diseño

Nuevamente, las bases de datos fueron pensadas de manera similar a los test anteriores y en base a lo definido en la tabla 4.8 de requerimientos, con la diferencia de que ahora no existen los bloques, en cambio es un run con múltiples pruebas, además, se debe almacenar la secuencia de estimulos y las asignaciones entre una señal mostrada y su botón de respuesta correcta.

Estructura	**Dato**	**Descripción**
Run	asignacion_senal_boton	Relaciones entre las señales y botones asignadas al participante con condición controlable.
	secuencia_estimulos	La secuencia de estimulos presentada al participante con condición controlable.
Prueba	num_prueba	Número de prueba dentro del run.
	senal_mostrada	Señal mostrada en la prueba.
	imagen_mostrada	Imagen presentada al participante luego de contestar.
	boton_respuesta	Botón seleccionado por el participante.
	tipo_respuesta	Indica si el participante contesta correctamente.
	tiempo_respuesta	Indica el tiempo en milisegundos que el participante demoró en contestar.
	fecha_inicio	Indica la fecha en que se habilita el botón para contestar, con precisión de milisegundos.
	fecha_respuesta	Indica la fecha en que el participante contesta la prueba, con precisión de milisegundos. Esta fecha junto a la fecha de inicio se utilizan para calcular el tiempo de respuesta, contrastando con tiempo_respuesta para verificar errores en la medición.

Tabla 4.8: Descripción de los datos requeridos en el test Stress Induction.

De esta forma, se crean las tablas necesarias, detalladas en la tabla 4.9.

Tabla	Descripción
stress_ind_prueba	Contiene las pruebas del test.
stress_ind_senal_boton	Contiene las asignaciones entre señales y botones.
stress_ind_secuencia_ estimulos	Contiene la secuencia en que se le mostraron imágenes al aprticipante con condición controlable.

Tabla 4.9: Tablas modeladas para el test Stress Induction en la base de datos de la app móvil .

La base de datos del servidor posee más entidades que la base de datos de la app, esto es debido a que se requiere emparejar participantes por ciertas caracteristicas sociodemográficas, mostrando los mismos estímulos de un participante a la pareja asignada, además, se describen como entidades los botones, las señales, las asignaciones entre señales y botones, las secuencias de estimulos y las imágenes presentadas. En la figura 4.8 se puede observar el diagrama E-R con las entidades generadas en este test y el detalle de cada una en la tabla 4.10.

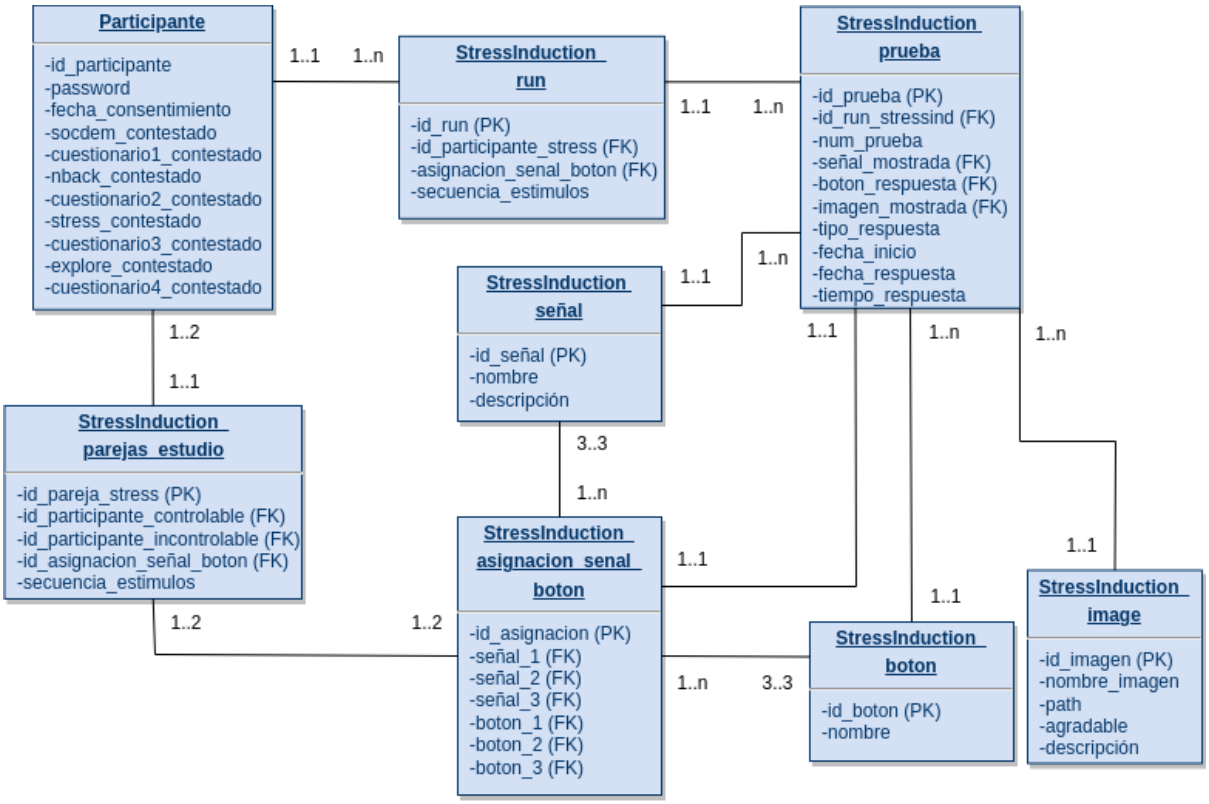


Figura 4.8: Diagrama E-R del test “Stress Induction”.

Tabla	Descripción
stressind_run	Contiene los runs del test por participante, sirve para agrupar las pruebas.
stressind_prueba	Contiene información sobre las pruebas contestadas por los participantes.
stressind_asignacion_señal_boton	Contiene información sobre las asignaciones de señales con los respectivos botones en el test.
stressind_señal	Contiene información sobre las señales utilizadas en las pruebas del test.
stressind_boton	Contiene información sobre los botones utilizados en las pruebas del test.
stressind_image	Contiene información sobre las imágenes almacenadas en el servidor que serán usadas en las pruebas del test.
stressind_parejas_estudio	Contiene las parejas asignadas para el test, con un participante controlable y el otro incontrolable.

Tabla 4.10: Descripción de tablas implementadas en la base de datos para el test Stress Induction.

4.5.2 Implementación

Las migraciones se definieron usando el diagrama E-R como referencia, haciendo uso de la clase *Schema* para la creación de las distintas tablas que se asemejan a la estructura del test de exploración y predicción, ya que las señales y botones se almacenan en tablas aparte que son pobladas con un *seeder*. Por otro lado, los runs poseen un identificador único y una restricción *unique* compuesta.

Algo relevante de notar es la tabla "*stressind_image"*, ya que esta hace referencia a las imágenes del test mediante el campo *path* que es utilizado para obtener las imágenes desde el almacenamiento interno del servidor. Con respecto a los modelos, se corresponden con cada una de las tablas generadas por la migración, especificando sus atributos relevantes.

Las rutas de este test son más en comparación a los test anteriores, esto es por los nuevos requerimientos.

- **admin/stressind/results:** se relaciona con la función *getResults* para obtener las respuestas.

- **api/stressind/store:** se relaciona con la función *storeResults* para almacenar las respuestas.
- **api/stressind/getCondition/{userId}:** cumple la función de asignar y retornar la cualidad de controlable o incontrolable utilizada en el test.
- **admin/stressind/uploadImage:** se relaciona con la función *uploadImage* que carga una imagen en el almacenamiento interno del servidor y crea su referencia en la base de datos.
- **admin/stressind/imageUpload:** se relaciona con la función *imageUpload* que retorna una vista para poder cargar imágenes al servidor.
- **api/stressind/images:** se relaciona con la función *getImages* que retorna los nombres de las imágenes cargadas al servidor que se utilizarán en el test.
- **api/stressind/image/{name}:** ruta que cumple la función de retornar una imagen a la aplicación móvil dado su **"*name*",** mediante el uso de la función response que genera una respuesta HTTP, asignandole las cabeceras *"Content-Type: image/jpeg"* que indica que el elemento que viene en el cuerpo de la respuesta es una imagen de extensión jpeg/jpg.

El controlador se llama *StressInductionController* y es responsable de manejar casi todas estas rutas definidas. Las funciones se detallan a continuación:

- **storeResults:** almacena las respuestas del test inducción de estrés en la base de datos siguiendo el mismo método que los test anteriores, obteniendo la información y respuestas en la petición HTTP y almacenandolas con el método *create* correspondiente, manejo de errores SQL, y finalmente, actualizando el progreso del participante.
- **getResults:** obtiene todas las respuestas del test inducción de estrés almacenadas, divididas en 4 conjuntos.
    - **resultsC:** Contiene la información de las respuestas de los participantes con condicion controlable.
    - **resultsU:** Contiene la información de las respuestas de los participantes con condicion incontrolable.
    - **señales:** Contiene información sobre las señales utilizadas en el test.
    - **parejas_estudio:** Contiene información sobre las parejas existentes, sus secuencias de imágenes y sus asignaciones señal-botón.

- **getCondition:** asigna la condición de controlable o incontrolable al participante indicado según lo siguiente:
    - Si existen participantes con condición controlable sin pareja incontrolable, con mismas cualidades (género y año de estudio) y que hayan contestado el test, se asigna incontrolable y se emparejan.
    - Si existen participantes con calidad controlable sin pareja, pero que no haya contestado el test, se asigna controlable y queda sin emparejar.
    - Si no existen participantes controlables sin pareja o existen pero no tienen las mismas cualidades, se asigna controlable y queda sin emparejar.
- **imageUpload:** retorna una vista para cargar las imágenes usadas en el test al servidor.
- **uploadImage:** almacena la imagen cargada en el almacenamiento del servidor haciendo uso del método *storeAs* de la clase *Request*, que hace uso de los métodos de la clase *Storage* permitiendo almacenar archivos en los distintos "*discos*", en este caso, se usó el disco local que la almacena en el directorio *storage/app/images* con el subdirectorio *agradable* o *desagradable* segun su cualidad, luego, se crea la referencia a esta imágen en la base de datos.
- **getImages:** entrega los nombres de las imágenes almacenadas en el servidor para descargarlas desde la aplicación móvil utilizando la ruta *api/stressind/image/{name}*.

4.6 Participante y administrativos

Los participantes y administrativos fueron implementados al finalizar los test, donde surge la problemática de cómo registrar a un participante de manera anónima y cómo autenticar su identidad de manera segura. Por otro lado, los administrativos utilizan un registro y autenticación común, es decir, mediante email y contraseña.

4.6.1 Diseño

La base de datos fue modelada con un diagrama E-R visible en la figura 4.9. Las entidades se describen en la tabla 4.11.

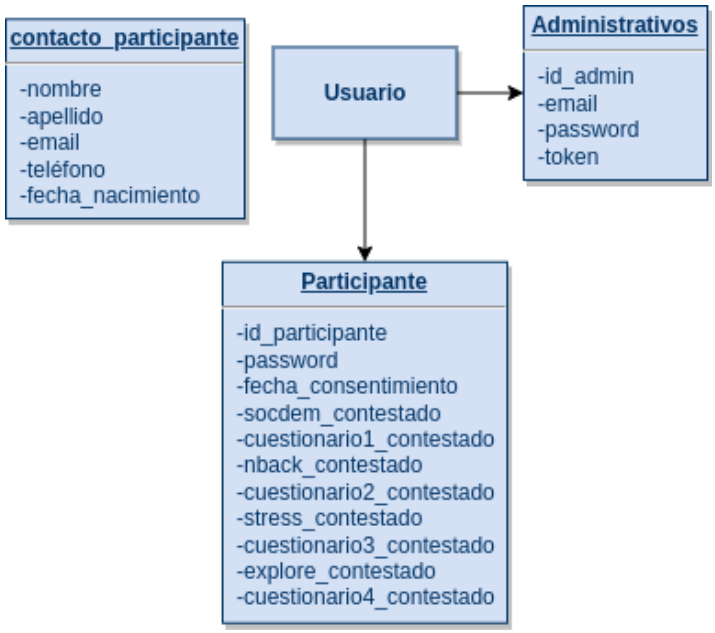


Figura 4.9: Diagrama E-R de “Participante”, “Administrativos” y “Contacto_participante”.

Tablas	**Descripción**
Administrativos	Contiene la información de autenticación de los administrativos registrados.
Participantes	Contiene la información de autenticación y avance de los participantes registrados.
Contacto_participante	Contiene información de los participantes que deseen ser contactados por el equipo de investigación o el comité de ética, se encuentra aislada para evitar relacionar información de contacto con las respuestas sociodemográficas, que se puede catalogar como información sensible.

Tabla 4.11: Descripción de tablas implementadas en la base de datos para los participantes y administrativos.

4.6.2 Implementación

En cuanto a las funcionalidades de los participantes, nace la problemática de cómo registrarlos y autenticarlos de manera anónima para proteger la confidencialidad e integridad de sus respuestas e información sensible entregada en los cuestionarios sociodemográficos.

Para solucionar esto, en el registro se ha utilizado un método que consiste en usar una contraseña ingresada por el participante en la app móvil, la que es encriptada y enviada al servidor. Luego, es encriptada nuevamente en el servidor utilizando el algoritmo bcrypt. Así, una vez registrado, se le asigna un número identificador y un token de autenticación al participante, los que son manejados internamente por el servidor y la aplicación móvil.

Para la autenticación se utiliza Laravel Sanctum como middleware para las rutas de login, almacenamiento de respuestas y obtención de información de los test y cuestionarios. Así, se hace uso del token para verificar que el participante está registrado, pudiendo acceder a la información usada en tests y cuestionarios. Además, para el almacenamiento de respuestas, se verifica que el token recibido corresponde al número identificador de participante enviado en las peticiones, evitando que terceros puedan responder para cualquier participante con un token válido. También, se han definido nuevas clases de request (peticiones HTTP), que permiten definir las reglas y validaciones que debe cumplir una petición HTTP del tipo especificado que se recibe en una función. Por ejemplo, que la petición para registrarse tenga una contraseña y que esta tenga un mínimo de 8 caracteres.

En cuanto a los administrativos, se ha implementado su registro y autenticación utilizando Laravel Breeze, que entrega una implementación de autenticación y manejo de sesiones simple y segura, utilizando interfaces web con Tailwind CSS. Así, se han generado las rutas de registro, autenticación, obtención de resultados. Además, se ha implementado la funcionalidad de exportar los resultados en formato XLSX (Excel) o CSV. Para ello, se ha utilizado la biblioteca Laravel Excel, que permite transformar las colecciones obtenidas de la base de datos a un archivo del formato especificado.

La migración fue generada en base a lo modelado en el diagrama E-R, donde se implementa el almacenamiento del progreso en la app del participante. En la figura 4.10 se puede observar la migración del participante que contiene los *Schemas participante* y *contacto_participante,* donde el progreso se ve reflejado en los campos boolean.

```php
public function up(){
    Schema::create('participante', function(Blueprint $table){
        $table->id('id_participante');
        $table->string('password');
        $table->string('fecha_consentimiento');
        $table->boolean('socdem_contestado')->default(false);
        $table->boolean('cuestionario1_contestado')->default(false);
        $table->boolean('nback_contestado')->default(false);
        $table->boolean('cuestionario2_contestado')->default(false);
        $table->boolean('stress_contestado')->default(false);
        $table->boolean('cuestionario3_contestado')->default(false);
        $table->boolean('explore_contestado')->default(false);
        $table->boolean('cuestionario4_contestado')->default(false);
    });
    Schema::create('contacto_participante', function (Blueprint $table){
        $table->tinyText('nombre');
        $table->tinyText('apellido');
        $table->string('fecha_nacimiento');
        $table->tinyText('telefono')->unique();
        $table->string('email')->unique();
    });
}
```

Figura 4.10: Migración “Participante”.

Los modelos generados corresponden a cada una de las tablas generadas en las migraciones *Participante* y *Administrativos*. Ahora, a diferencia de los demás modelos implementados en los test, estos poseen la propiedad *protected $hidden*, que corresponde a los campos a ocultar al serializar (transformar a JSON en este caso) los datos obtenidos desde la base de datos. En la figura 4.11 se puede observar esta implementación para el modelo Participante, que se extiende a las propiedades del usuario por defecto de Laravel, que se ha renombrado como *Authenticatable*, el cual puede acceder a métodos de autenticación internos.

```php
class Participante extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable;
    protected $table = 'participante';
    protected $primaryKey = 'id_participante';
    public $timestamps = false;

    protected $fillable = [
        'password',
        'fecha_consentimiento',
        'socdem_contestado',
        'nback_contestado',
        'stress_contestado',
        'explore_contestado',
    ];

    protected $hidden = [
        'password',
    ];
}
```

Figura 4.11: Modelo "Participante".

Las rutas definidas para los participantes son las siguientes.

- **api/register:** se relaciona con la funcion *register* para el registro de participantes.
- **api/login**: se relaciona con la funcion *login* para la autenticación de participantes.
- **api/reset:** se relaciona con la función *resetPassword*, permitiendo modificar la contraseña de un participante.
- **api/contacto:** se relaciona con la función *contacto*, permitiendo almacenar los datos de contacto de un participante sin relacionarlo con sus respuestas.

El controlador implementado para el participante se llama *AuthController*, que a diferencia de los demás controladores, hace uso de peticiones especiales, estas son:

- **RegisterParticipanteRequest:** petición para registrarse, contiene reglas de validación para la contraseña y fecha de consentimiento.
- **LoginParticipanteRequest:** petición de autenticación, contiene reglas de validación para el número identificador, contraseña y token único del usuario.
- **ResetPasswordRequest:** petición de recuperación de contraseña, requiere de la nueva contraseña, el id de participante y su respectivo token.
- **ContactoParticipanteRequest:** petición de registro de contacto con participante, contiene reglas de validación para los distintos campos de contacto.

En los anexos 8.4 y 8.5 se puede observar la implementación de la función *rules* de las peticiones de registro y autenticación del participante respectivamente, estas reglas son usadas para validar los datos recibidos por las peticiones enviadas desde la aplicación móvil.

Ahora, para las funcionalidades requeridas se utilizaron estas peticiones y reglas para poder registrar y autenticar al participante, a continuación se detalla el proceso de validación.

- **login:** En esta función se recibe la petición del usuario para autenticarse, primero validando que el token recibido se corresponda con el id de participante recibido, luego se validan los datos llamando al método *validated* de la petición, se utiliza el método *guard* de la clase *Auth* que corrobora que los datos recibidos se correspondan con lo almacenado en la base de datos, si los datos son correctos, se retorna un "ok" a la aplicación móvil. En el anexo 8.6 se muestra la implementación.
- **register:** En esta función se recibe la petición y se valida de la misma forma que en la función login, si estas cumplen con las reglas, se crea un nuevo participante mediante el método *create*, se hace hash de la contraseña, se crea su token único, se almacena y se retorna esta información a la aplicación para ser manejada internamente en la autenticación. En el anexo 8.7 se muestra la implementación.
- **resetPassword:** En esta función se recibe la petición de cambio de contraseña de un participante, se validan los datos requeridos, si estos son correctos, se modifica la contraseña.
- **contacto:** En esta función se reciben los datos de contacto especificados y se almacenan.

En la tabla 4.12 se puede visualizar un resumen de la asociación de las rutas, las peticiones especiales y sus respectivas funciones implementadas.

Ruta	**Petición**	**Función**
api/register	RegisterParticipanteRequest	Register
api/login	LoginParticipanteRequest	Login
api/reset	ResetPasswordRequest	ResetPassword
api/contacto	ContactoParticipanteRequest	Contacto
api/progress/{*userID*}	*HTTP Request común*	GetProgress

Tabla 4.12: Rutas, peticiones especiales y funciones asociadas implementadas.

Para los Administrativos se utilizó Laravel Breeze para satisfacer las necesidades de registro y autenticación, siendo este una implementación simple de estas funcionalidades, con manejo seguro de sesiones y una implementación gráfica con TailwindCSS, permitiendo implementar rápidamente un sistema de gestión para Administrativos. Las rutas definidas para los administrativos son las siguientes:

- **admin/register:** se relaciona con la función store de *RegisteredUserController*, para crear y almacenar un administrativo.
- **admin/login:** se relaciona con la función store de *AuthenticatedSessionController*, para autenticar un administrativo.
- **admin/logout:** se relacion con la función destroy de *AuthenticatedSessionController* para destruir la sesión activa del administrativo.

De la misma forma que los participantes, se hace uso de una petición especial en la función store para el login, que es la siguiente:

- **LoginRequest:** de la misma forma que las definidas para el participante, esta petición especial tiene las reglas de validación para los datos requeridos, además, posee funciones adicionales, la más notable es la función authenticate.
    - **authenticate:** función que verifica si es que se ha excedido el número máximo de intentos de inicio de sesión. Luego, verifica en la base de datos si es que existe el administrativo ingresado, si existe, termina la función, en el caso de no existir arroja un error.

En la tabla 4.13 se detallan las funciones requeridas para el registro y autenticación.

Controlador	**Ruta**	**Función**	**Descripción**
RegisteredUser Controller	admin/register	store	Función que valida los campos recibidos desde la petición de registro, una vez validados, se crea un nuevo administrativo, se hace inicio de sesión y se redirecciona al dashboard.
AuthenticatedSession Controller	admin/login	store	Función que llama a la función authenticate de la petición LoginRequest, si esta termina satisfactoriamente, se crea una nueva sesión para el administrativo y se redirecciona a la ruta protegida que se intentó acceder.

	admin/logout	destroy	Función que cierra la sesión del administrativo e invalida la sesión activa, luego redirecciona al login.

Tabla 4.13: Descripción de funciones de registro y autenticación de administrativos.

4.6.3 Sistema web de Administrativos

Para los administrativos se ha implementado un sistema web de gestión básico, permitiendo observar los resultados de los distintos test, poder exportar los resultados en archivos en formato xlsx o csv y poder cargar imágenes utilizadas en el test de inducción de estrés. La interfaz se puede observar en los anexos 8.8 y 8.9.

A continuación, en la tabla 4.14 se puede observar un resumen de las rutas disponibles para navegación en el sistema web de administrativos y las respectivas funciones asociadas.

Controlador	**Ruta**	**Función**
SocdemController	admin/socdem/results	getAnswers
NBackController	admin/nback/results	getResults
StressInduction Controller	admin/stressind/results	getResults
ExplorePredict Controller	admin/explorepredict/results	getResults
ExportsController	admin/export/	exportView
	admin/export/socdem/{extension}	exportSocdem
	admin/export/nback/{extension}	exportNBack
	admin/export/stressind/{extension}	exportStressind
	admin/export/explorepredict/{extension}	exportExplorePredict

Tabla 4.14: Controladores, rutas y funciones asociadas al sistema web de administrativos.

4.7 Cuestionarios Sociodemográficos

Los cuestionarios sociodemográficos fueron la última implementación dado por su baja complejidad al ser formularios con preguntas y respuestas.

4.7.1 Diseño

El almacenamiento de los cuestionarios sociodemográficos se diseñó en base a lo definido en la tabla 4.15, lo que indica que es necesario implementar una tabla para almacenar los cuestionarios contestados y las respuestas a cada pregunta, y las correspondientes preguntas.

Estructura	**Dato**	**Descripción**
Cuestionario	cuestionario	El cuestionario que se ha contestado.
	fecha_contestado	Fecha en que se contestó el cuestionario.
Pregunta	tipo	Tipo de pregunta, se refiere a que cuestionario pertenece.
	enunciado	El enunciado de la pregunta.
Respuesta	id_pregunta	Número identificador único de la pregunta presentada.
	respuesta	Respuesta a la pregunta presentada.

Tabla 4.15: Descripción de los datos requeridos para los cuestionarios sociodemográficos.

De esta forma, se diseña la persistencia de la app móvil, sin ser necesario almacenar las preguntas ya que se acordó almacenarlas en el servidor. En la tabla 4.16 se describen las tablas para los cuestionarios sociodemográficos.

Tabla	**Descripción**

socdem_cuestionarios	Contiene el nombre del cuestionario y la fecha en la que se contestó
socdem_respuestas	Contiene las respuestas a las preguntas de los cuestionarios sociodemográficos.

Tabla 4.16: Tablas modeladas para los cuestionarios sociodemográficos en la base de datos de la app móvil .

La base de datos se modeló con un diagrama E-R visible en la figura 4.13. Las entidades se detallan en la tabla 4.17.

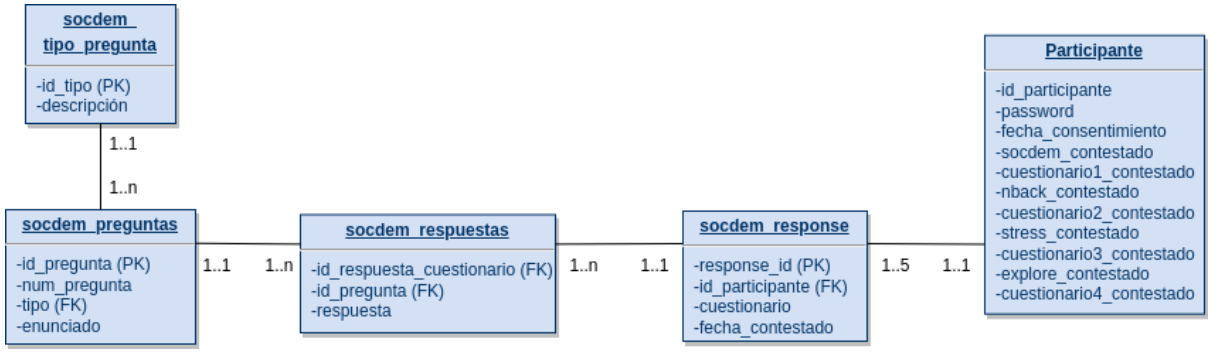


Figura 4.13: Diagrama E-R "Cuestionario Sociodemográfico".

Tabla	**Descripción**
socdem_response	Contiene los cuestionarios contestados por los participantes, sirve para agrupar las respuestas.
socdem_respuestas	Contiene las respuestas a las preguntas de los distintos cuestionarios sociodemográficos.
socdem_preguntas	Contiene las preguntas de los cuestionarios, se usa para obtener información de estas como su enunciado y tipo de pregunta.
socdem_tipo_pregunta	Contiene los tipos de preguntas, es decir, si corresponde al cuestionario Sociodemográfico, PANAS, STAIT, GAD, entre otros.

Tabla 4.17: Descripción de tablas implementadas en la base de datos para los cuestionarios sociodemográficos.

4.7.2 Implementación

La migración fue generada en base a lo modelado por el diagrama E-R, y los modelos corresponden a cada una de estas tablas. En la tabla 4.18 se puede observar los modelos y su tabla asociada.

Modelo	**Tabla**
SocDemResponse	socdem_response
SocDemPreguntas	socdem_preguntas
SocDemRespuestas	socdem_respuestas
SocDemTipoPregunta	socdem_tipo_pregunta

Tabla 4.18: Modelos implementados para los cuestionarios sociodemográficos.

Las rutas fueron definidas pensando en las funcionalidades requeridas, que son las siguientes.

- **admin/socdem/results:** se relaciona con la función *getAnswers* para obtener las respuestas de los cuestionarios.
- **api/socdem/store:** se relaciona con la función *storeAnswers* para almacenar las respuestas de los cuestionarios.
- **api/socdem/preguntas:** obtiene las preguntas de los cuestionarios sociodemográficos, con una función declarada internamente en la ruta.

El controlador *SocDemController* tiene las funciones especificadas anteriormente, que son las siguientes.

- **getAnswers:** obtiene las respuestas de los cuestionarios sociodemográficos de todos los participantes.

- **storeAnswers:** almacena las respuestas de los cuestionarios sociodemográficos en la base de datos siguiendo el mismo método de los test, es decir, se obtienen los campos desde la petición HTTP, se usa el método *create* del modelo respectivo, y al finalizar se actualiza el progreso.

4.8 Testing

Para el testing de los prototipos incrementales, se ha poblado la base de datos del servidor con los elementos necesarios para el almacenamiento de las respuestas (preguntas, tipos, alternativas, islas, aviones, pilotos, señales e imágenes) usando seeders. Luego, se ha utilizado la herramienta Postman para enviar las peticiones de almacenamiento y visualización de datos, verificando que funcionan correctamente, con la salida correspondiente dependiendo de las validaciones o errores que podrían ocurrir. Al funcionar todo de manera correcta, se realizan las pruebas de integración con la aplicación móvil, que consisten en contestar un test y enviar las respuestas a la API, verificando que se almacenen correctamente y que no ocurran fallas en la comunicación. Además, se ha hecho testing de la funcionalidad de enviar respuestas almacenadas localmente en la aplicación móvil, en caso de una pérdida de conexión a internet, primero verificando que las respuestas se guarden en el formato especificado y persista ante cierres o reinicios de la aplicación. Luego, se verifica que se puedan obtener estos datos y poder ordenarlos en el formato de envío correcto.

5. EVALUACIÓN DE LA PROPUESTA

La evaluación de la implementación desarrollada se ha realizado en base al análisis del rendimiento de la API, es decir, el análisis de métricas como: tiempos de respuesta, escalabilidad, consumo de recursos CPU y memoria. Para esto, se han utilizado las siguientes herramientas de monitoreo.

- Grafana
- Prometheus
- Node Exporter
- cAdvisor

Estas se han utilizado en conjunto con la herramienta Docker, permitiendo crear un contenedor para la API simulando un servidor linux, y distintos contenedores para Grafana, Prometheus, Node Exporter y cAdvisor, con una red interna que permite obtener las métricas sin interferir en el rendimiento de la API. Por otro lado, para medir de manera realista estas métricas, se ha utilizado la herramienta *k6* para hacer pruebas de carga simulando usuarios participantes haciendo uso de la aplicación.

Las pruebas realizadas se han dividido en escenarios, que permiten establecer la cantidad de usuarios virtuales concurrentes (VUs) y cuanto tiempo se estará ejecutando la prueba. Los escenarios definidos se pensaron según la cantidad de usuarios para el pilotaje (alrededor de 200 participantes) y según la carga concurrente, la cual es baja, ya que la mayor parte del tiempo se están contestando tests o cuestionarios que demoran alrededor de 20 a 40 minutos, habiendo un tiempo similar entre peticiones. Los test son los siguientes.

- **Smoke test (Prueba de humo):** Consiste en un test de baja carga para verificar que, tanto el sistema como la prueba de rendimiento, funcionan correctamente.
    - 5 VUs con 30s de duración total de la prueba.
- **Load test (Prueba de carga):** Consiste en un test de carga media para medir el rendimiento de la API en un escenario de uso normal. Este se subdivide en 3 etapas, que son las siguientes.
    - **Ramp-up o subida del tráfico:** realiza la creación gradual de VUs en un tiempo determinado.
        - 100 VUs en 5 minutos.
    - **Mantención de carga promedio:** consiste en mantener una carga constante durante un cierto tiempo.
        - 100 VUs durante 30 minutos.
    - **Ramp-down o bajada del tráfico:** gradualmente reduce la carga durante cierta cantidad de tiempo.
        - 0 VUs en 5 minutos.
- **Stress test (Prueba de estrés):** Consiste de un test sobre el promedio, para medir el rendimiento bajo una carga mayor y los límites de tráfico de la aplicación.
    - Siguiendo un patrón similar a Load test, se realizan 3 etapas de Ramp-up y mantención, seguidas de una etapa final de Ramp-down
        - 100 VUs en 2 minutos.
        - 100 VUs durante 5 minutos.
        - 200 VUs en 2 minutos.
        - 200 VUs durante 5 minutos.
        - 300 VUs en 2 minutos.
        - 300 VUs durante 5 minutos.
        - 0 VUs en 10 minutos.

Los resultados obtenidos se pueden observar en la tabla 5.1 que los resume, para una visualización gráfica, referirse a los anexos 8.10, 8.11, 8.12, 8.13, 8.14, 8.15, 8.16, 8.17, 8.18.

Escenario	**Etapa**	**CPU (%)**			**Memoria (MiB)**			**Tiempo de respuesta promedio (s)**
		Min	**Avg**	**Max**	**Min**	**Avg**	**Max**	
Smoke test	*Única*	0.05	1.91	13	73.25	85.54	94.38	0.045
Load test	Ramp-up	0.03	14.3	45.4	138.15	155.12	162.6	7.76
	Mantención	0.07	37.1	63.4	156.18	158.1	166.21	
	Ramp-down	0.018	14.9	36.9	159.22	161.42	167.18	
Stress test	Ramp-up	0.027	24.5	53	95.4	95.4	95.6	19.72
	Mantención	24.5	39.7	49.4	95.5	95.9	100.7	
	Ramp-up	27.3	36.7	58.6	95.7	96	99.4	
	Mantención	15.3	41.7	56.3	95.8	95.9	96	
	Ramp-up	23.5	44.2	58.9	96	96	96.2	
	Mantención	33.8	41.9	51.4	96.1	97.3	107.8	
	Ramp-down	0.01	27.7	61.4	97.2	97.9	108.9	

Tabla 5.1: Resultados de uso de recursos para las distintas pruebas de carga realizadas.

A continuación, en las figuras 5.1 y 5.2 se pueden observar gráficos comparativos que representan de mejor manera los datos expuestos en la tabla anterior, principalmente entre los valores promedio de CPU por etapa y de memoria utilizada en MiB por etapa, donde el color azul corresponde al Smoke test, el amarillo al Load test y el rojo al Stress Test.

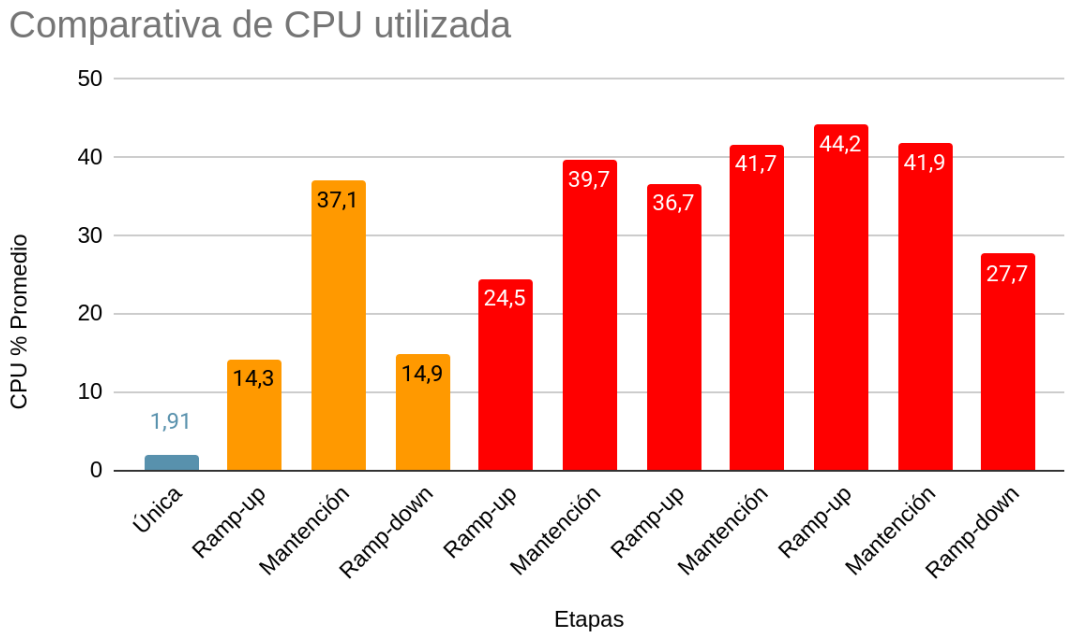


Figura 5.1: Gráfico comparativo de valores promedio de porcentaje de CPU utilizado por etapa.

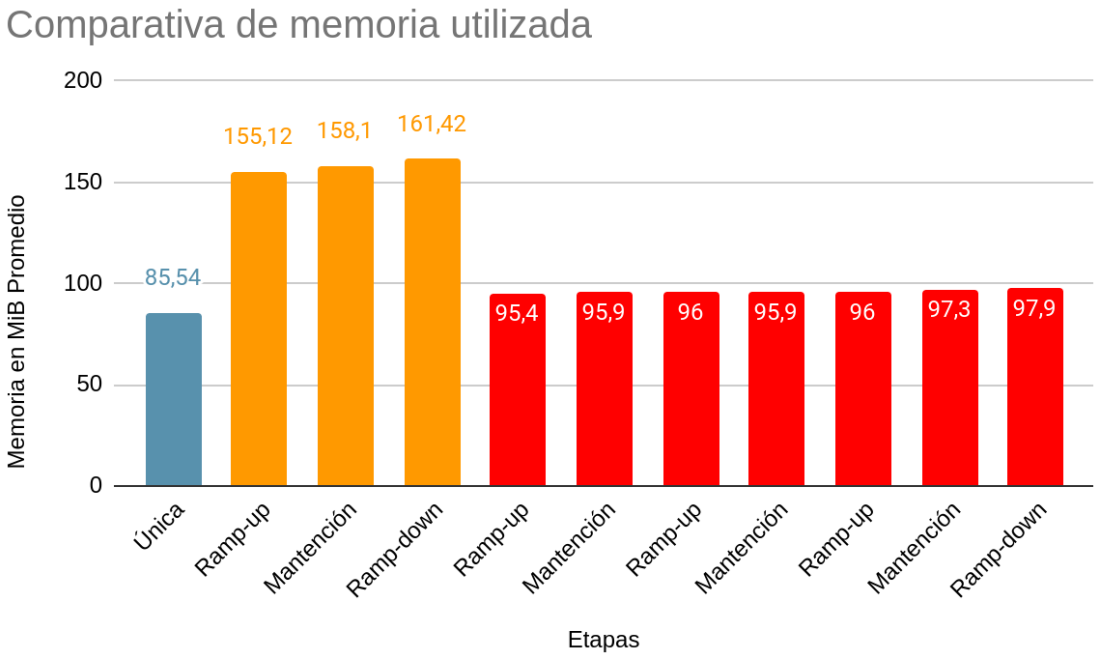


Figura 5.2: Gráfico comparativo de valores promedio de memoria utilizada en MiB por etapa.

Analizando los valores obtenidos, y en base a lo observado durante la realización de las pruebas, se puede decir que la API se comporta de la manera esperada, permitiendo realizar un pilotaje de la aplicación móvil con la cantidad especificada de usuarios.

Esto es válido a pesar de que su rendimiento empeoró en el Stress test, debido a que las peticiones comenzaron a rechazarse por parte de la API (Anexo 8.18), lo que indica una sobrecarga de tráfico manejado internamente por el limitador de peticiones por minuto por parte de una misma IP. Sin embargo, es necesario considerar que 300 usuarios concurrentes es una cifra no menor, que según las pruebas, es un tráfico constante por usuario.

En condiciones normales de operación de la app móvil, esto es muy difícil que ocurra, debido a que la realización de los test comprende entre 20 a 40 minutos enviando solo una petición HTTP al finalizar. Además, estos se pueden realizar a cualquier hora del día, por lo que el tráfico real tendrá ventanas de tiempo bastante amplias. Por otro lado, considerando que el tiempo de respuesta promedio de la API es de 7 segundos para una carga regular de 100 usuarios concurrentes, si llegara a ocurrir que 100 personas contesten tests o cuestionarios simultáneamente, la API es capaz de manejarlo, además, considerando que si bien 100 Mib de consumo de memoria es una cifra alta, esta permanece constante a través del tiempo. En conclusión, la API se encuentra funcional para realizar un pilotaje en un ambiente de producción.

6. CONCLUSIONES

Al finalizar esta memoria de título, se han cumplido los objetivos específicos de:

- Configuración del servidor en el que se aloja la aplicación y sistema web, permitiendo tener un ambiente de pruebas real, además de permitir al equipo de desarrollo poder lanzar la app a producción y obtener datos para el equipo de investigación.
- Diseñar e implementar las bases de datos del servidor y local en la app móvil, evitando la pérdida de información por parte de la app en caso de no poder enviar las respuestas de manera automática al finalizar un test, y también permitiéndoles tener la información almacenada en una base de datos en un servidor, dándoles la opción de visualizar y obtener los datos para su posterior análisis y entrega de resultados.
- Diseñar e implementar la lógica de la API (Backend), permitiendo establecer las funcionalidades principales que son vitales para el funcionamiento correcto de la obtención de datos desde la aplicación móvil y para poder mostrarlos mediante un sistema web a los investigadores.

Y se ha cumplido el objetivo general de resguardar el anonimato de los usuarios participantes mediante el registro solo utilizando una contraseña, a la que se le relacionan los identificadores únicos que permiten autenticarse y no vincularlo con alguna forma de contacto directo.

La solución propuesta presenta bastantes ventajas con respecto al funcionamiento de la API, agilidad y facilidad de desarrollo, y legibilidad del código ya que es bastante simple en sintaxis y estructuración. Además, provee un rápido funcionamiento de la base de datos y un amplio abanico de funcionalidades que facilitan mucho la implementación. Aun con estas tecnologías, se presentaron dificultades, principalmente en las primeras etapas, dada la poca experiencia en

desarrollo y comunicación web, además de la necesidad de establecer una buena comunicación entre el equipo y los investigadores para esclarecer los requerimientos.

Se trató de priorizar los requisitos funcionales frente a los no funcionales, y también se estableció una buena comunicación dentro del mismo equipo de desarrollo para acordar las estructuras de comunicación. Otra dificultad más técnica fue la implementación de la autenticación anónima, ya que comúnmente el registro de usuarios necesita una identificación única que no sea fácil de falsificar. También, otra dificultad fue la implementación del envío de imágenes hacia la app móvil y la recepción de estas desde la app, ya que la mayoría del tiempo se trabaja con JSON, y una imagen es un elemento complejo de transmitir. Finalizando, en base a lo desarrollado y a la experiencia adquirida al desarrollar el proyecto, el trabajo futuro que se estima es.

- Contratación y configuración de un servicio de hosting web para la API y sistema web, ya que este fue suplementado mediante el uso de un servidor provisto por la Facultad de Ingeniería y la DTI de la Universidad de Concepción.
- Configuración de certificados SSL para utilizar protocolo HTTPS y proveer otra capa de seguridad a la comunicación.
- Optimización de código haciendo mejor uso de las funcionalidades de los modelos, migraciones y controladores.

Dicho esto, el desarrollo del sistema StressMApp ha sido satisfactorio. Si bien queda por mejorar y perfeccionar, se logró la implementación de un prototipo funcional que comprende las funcionalidades principales requeridas para la obtención de información, todo esto dentro de las capacidades del equipo de desarrollo. Por esto, a pesar de las dificultades del proceso, el equipo se esforzó para lograr cumplir con lo pedido y aportar al proyecto StressMApp con una app móvil y una aplicación web que facilitarán la obtención y utilización de información de manera fácil y eficiente, lo que nos hace sentir orgullosos de haber podido apoyar en un proyecto que busca mejorar la calidad de vida estudiantil.

7. BIBLIOGRAFÍA

[1] Cadvisor. https://github.com/google/cadvisor

[2] CSRF-Token. https://laravel.com/docs/9.x/csrf#main-content

[3] Eloquent ORM. https://laravel.com/docs/9.x/eloquent

[4] expo-SQLite. https://docs.expo.dev/versions/latest/sdk/sqlite/

[5] Git. https://git-scm.com/

[6] Github. https://github.com/

[7] Grafana. https://grafana.com/

[8] IBM. (s.f.).*What is Docker?*. https://www.ibm.com/topics/docker

[9] IBM. (s.f.). *What is a database schema?*. https://www.ibm.com/topics/database-schema#:~:text=the%20next%20step-,What%20is%20a%20database%20schema%3F,the%20relationships%20between%20these%20entities.

[10] Keep Coding Team. (5 de junio de 2023). ¿Qué es cAdvisor o Container Advisor?. *keepcoding.io.* https://keepcoding.io/blog/que-es-cadvisor/

[11] k6. https://k6.io/

[12] Laravel Sanctum. https://laravel.com/docs/9.x/sanctum

[13] Laravel 9. https://laravel.com/docs/9.x/releases

[14] Modelo MVC. https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html

[15] Node exporter. https://github.com/prometheus/node_exporter

[16] PostgreSQL. https://www.postgresql.org/

[17] Prisma.io. (s.f.). *What are database migrations?*. https://www.prisma.io/dataguide/types/relational/what-are-database-migrations

[18] Prometheus. https://prometheus.io/

[19] React.js. https://es.reactjs.org/

[20] React bootstrap. https://react-bootstrap.github.io/

[21] React Native expo. https://expo.dev/

[22] VRID Universidad de Concepción. (27 de Abril de 2022). UdeC financiará 25 nuevos proyectos multidisciplinarios. *Noticias UdeC*. https://noticias.udec.cl/udec-financiara-25-nuevos-proyectos-multidisciplinarios/

8. ANEXOS

8.1 Diagrama Entidad-Relación

1. Diagrama E-R y arquitectura del servidor.
   https://viewer.diagrams.net/?tags=%7B%7D&highlight=0000ff&edit=_blank&layers=1&nav=1&title=StressMApp%20arquitectura.drawio#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D1oEdFuczVLEKuvdONnrBqvZdoEmFfm_GS%26export%3Ddownload

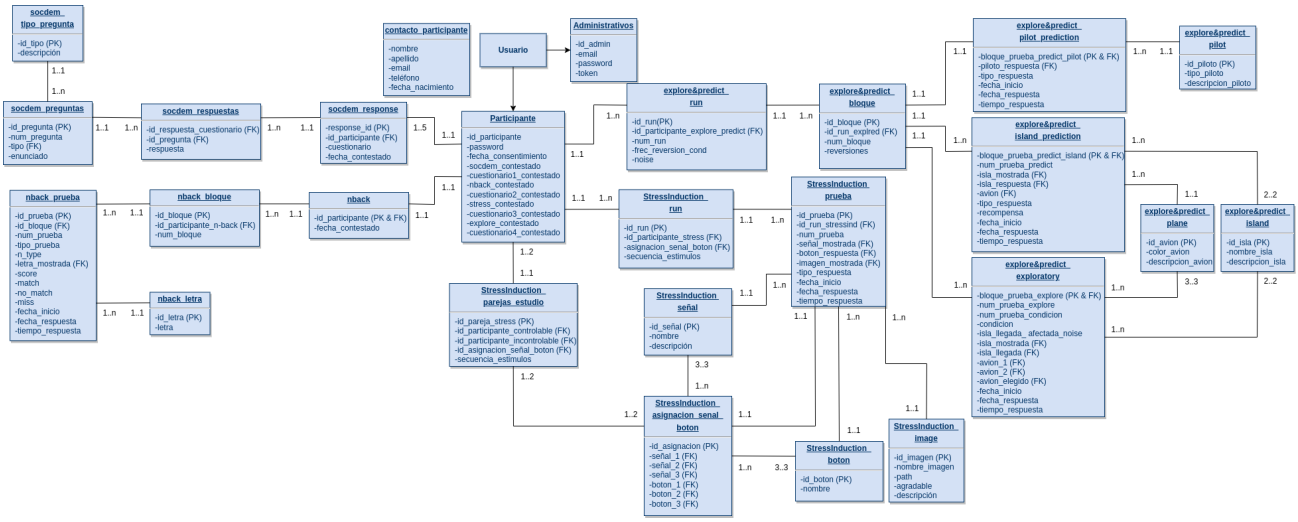


Figura 8.1: Diagrama E-R de la base de datos del servidor.

8.2 Códigos

```php
public function storeResults(Request $request){
    //$start = microtime(true);
    $id_participante = $request->input('id_participante');
    if($request->user()->id_participante != $id_participante){
        return response('No autorizado',401);
    }
    //ids for deletion when error occurs
    $nback_bloques_ids = array();
    $nback_pruebas_ids = array();
    try{
        NBack::create([
            'id_participante' => $id_participante,
            'fecha_contestado' => $request->input('fecha_contestado'),
        ]);
    }catch(\ErrorException $err){
        return response(["No se pudo guardar nback", $err->getCode()],503);
    }catch(QueryException $ex){//Query Error handling
        switch ($ex->getCode()){
            case 23505: //Duplicate key
                return response("Error 23505: Usuario ya ha contestado N-Back", 503);
            case 23502: //Null Key
                return response("Error 23502: Falta id de usuario (id null)", 503);
            case 23503: //Foreign key
                return response("Error 23503: Usuario no existe", 503);
            default:
                return response(["No se pudo guardar nback", $ex->getCode()],503);
        }
    }
```

Figura 8.2: Obtención y almacenamiento de información, y manejo de errores SQL en función storeResults en archivo NBackController.php.

```php
public function getResults(){
    $runs = DB::table('explpred_run as run')
        ->select(['run.id_participante_explore_predict as id_participante','run.num_run as run','run.noise','run.frec_reversion_cond'])
        ->get();

    $exploreResults = DB::table('explpred_run as run')
        ->join('explpred_bloque as bl','bl.id_run_explpred','=','run.id_run')
        ->join('explpred_exploratory as exp','exp.bloque_prueba_explore','=','bl.id_bloque')
        ->join('explpred_island as is1','is1.id_isla','=','exp.isla_mostrada')
        ->join('explpred_island as is2','is2.id_isla','=','exp.isla_llegada')
        ->join('explpred_plane as p1','p1.id_avion','=','exp.avion_1')
        ->join('explpred_plane as p2','p2.id_avion','=','exp.avion_2')
        ->join('explpred_plane as p3','p3.id_avion','=','exp.avion_elegido')
        ->select(['run.id_participante_explore_predict as id_participante','run.num_run as run','bl.num_bloque as bloque',
            'exp.num_prueba_explore as prueba','exp.condicion','exp.isla_llegada_afectada_noise','is1.nombre_isla as isla_mostrada',
            'is2.nombre_isla as isla_llegada','p1.color_avion as avion_1','p2.color_avion as avion_2','p3.color_avion as avion_elegido',
            'exp.fecha_inicio','exp.fecha_respuesta','exp.tiempo_respuesta'])
        ->get();

    $islandPredResults = DB::table('explpred_run as run')
        ->join('explpred_bloque as bl','bl.id_run_explpred','=','run.id_run')
        ->join('explpred_island_prediction as islP','islP.bloque_prueba_predict_island','=','bl.id_bloque')
        ->join('explpred_island as is1','is1.id_isla','=','islP.isla_mostrada')
        ->join('explpred_island as is2','is2.id_isla','=','islP.isla_respuesta')
        ->join('explpred_plane as p1','p1.id_avion','=','islP.avion')
        ->select(['run.id_participante_explore_predict as id_participante','run.num_run as run','bl.num_bloque as bloque',
            'islP.num_prueba_predict as prueba','is1.nombre_isla as isla_mostrada','p1.color_avion as avion','is2.nombre_isla as isla_respuesta',
            'islP.tipo_respuesta','islP.recompensa','islP.fecha_inicio','islP.fecha_respuesta','islP.tiempo_respuesta'])
        ->get();

    $pilotPredResults = DB::table('explpred_run as run')
        ->join('explpred_bloque as bl','bl.id_run_explpred','=','run.id_run')
        ->join('explpred_pilot_prediction as pilP','pilP.bloque_prueba_predict_pilot','=','bl.id_bloque')
        ->join('explpred_pilot as p1','p1.id_piloto','=','pilP.piloto_respuesta')
        ->select(['run.id_participante_explore_predict as id_participante','run.num_run as run','bl.num_bloque as bloque','p1.tipo_piloto as piloto',
            'pilP.tipo_respuesta','pilP.fecha_inicio','pilP.fecha_respuesta','pilP.tiempo_respuesta'])
        ->get();

    return view('explore',compact(['runs','exploreResults','islandPredResults','pilotPredResults']));
}
```

Figura 8.3: Función “getResults” en archivo “ExplorePredictController.php”.

```php
public function rules()
{
    return [
        'password' => ['required', Password::defaults()],
        'fecha_consentimiento' => ['required','string'],
    ];
}
```

Figura 8.4: Reglas de validación para “RegisterParticipanteRequest”.

```php
public function rules()
{
    return [
        //
        'id_participante' => ['required','integer'],
        'password' => ['required','string','min:8'],
    ];
}
```

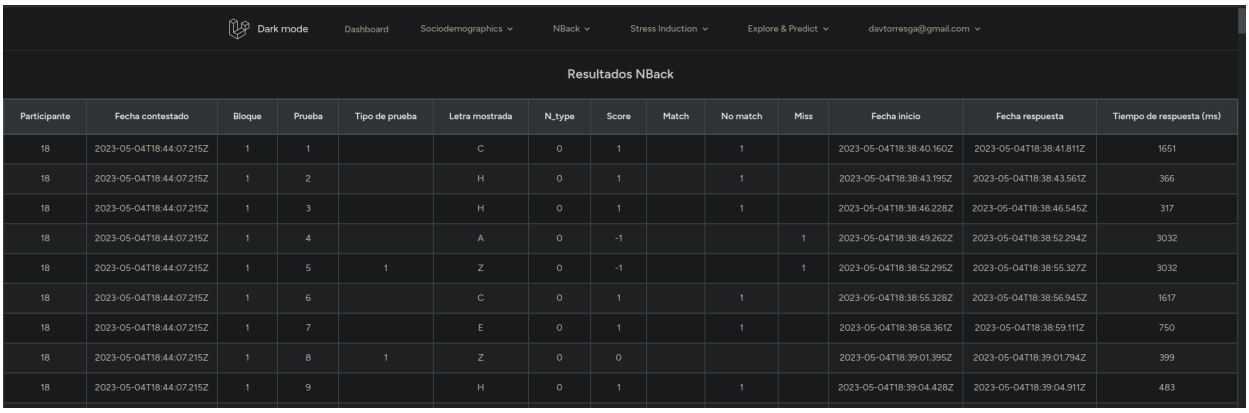
Figura 8.5: Reglas de validación para “LoginParticipanteRequest”.

```php
public function login(LoginParticipanteRequest $request){
    //verifica que el participante que esta contestando es el dueño del token
    if($request->user()->id_participante != $request->input('id_participante')){
        return $this->error('','No autorizado',401);
    }
    //Valida que las credenciales cumplan con los requisitos
    $request->validated($request->all());
    //Valida que las credenciales correspondan a lo de la DB, osea, valida que id y pass correspondan
    if (!Auth::guard('participante')->attempt($request->only(['id_participante', 'password']))) {
        return $this->error('', 'Credenciales incorrectas', 401);
    }
    return $this->success([
        "Successful login"
    ]);
}
```

Figura 8.6: Función “login” de participante en el archivo “AuthController.php”.

```php
public function register(RegisterParticipanteRequest $request){
    //valida la peticion
    $request->validated($request->all());
    //si se valida, crea el participante
    $participante = Participante::create([
        'password' => Hash::make($request->password),
        'fecha_consentimiento' => $request->input('fecha_consentimiento')
    ]);
    //crea el token
    $token = $participante->createToken('API token of participante ' . $participante->id_participante)->plainTextToken;
    //retorno los datos de success
    return $this->success([
        'id_participante' => $participante->id_participante,
        'token' => $token,
    ]);
}
```

Figura 8.7: Función “register” de participante en el archivo “AuthController.php”.



Participante	Fecha contestado	Bloque	Prueba	Tipo de prueba	Letra mostrada	N_type	Score	Match	No match	Miss	Fecha inicio	Fecha respuesta	Tiempo de respuesta (ms)
18	2023-05-04T18:44:07.215Z	1	1		C	0	1		1		2023-05-04T18:38:40.160Z	2023-05-04T18:38:41.811Z	1651
18	2023-05-04T18:44:07.215Z	1	2		H	0	1		1		2023-05-04T18:38:43.195Z	2023-05-04T18:38:43.561Z	366
18	2023-05-04T18:44:07.215Z	1	3		H	0	1		1		2023-05-04T18:38:46.228Z	2023-05-04T18:38:46.545Z	317
18	2023-05-04T18:44:07.215Z	1	4		A	0	-1			1	2023-05-04T18:38:49.262Z	2023-05-04T18:38:52.294Z	3032
18	2023-05-04T18:44:07.215Z	1	5	1	Z	0	-1			1	2023-05-04T18:38:52.295Z	2023-05-04T18:38:55.327Z	3032
18	2023-05-04T18:44:07.215Z	1	6		C	0	1		1		2023-05-04T18:38:55.328Z	2023-05-04T18:38:56.945Z	1617
18	2023-05-04T18:44:07.215Z	1	7		E	0	1		1		2023-05-04T18:38:58.361Z	2023-05-04T18:38:59.111Z	750
18	2023-05-04T18:44:07.215Z	1	8	1	Z	0	0				2023-05-04T18:39:01.395Z	2023-05-04T18:39:01.794Z	399
18	2023-05-04T18:44:07.215Z	1	9		H	0	1		1		2023-05-04T18:39:04.428Z	2023-05-04T18:39:04.911Z	483

Figura 8.8: Interfaz del sistema web de administrativos para resultados del test N-Back.

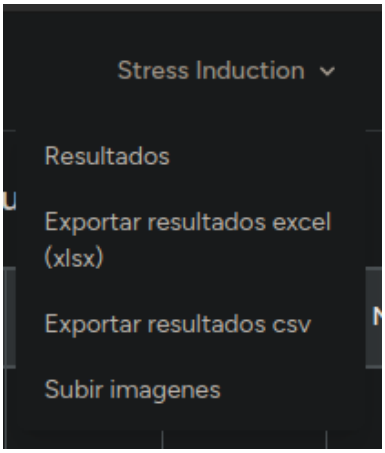


Figura 8.9: Opciones disponibles en dropdown stress induction.

8.3 Resultados del Load Testing

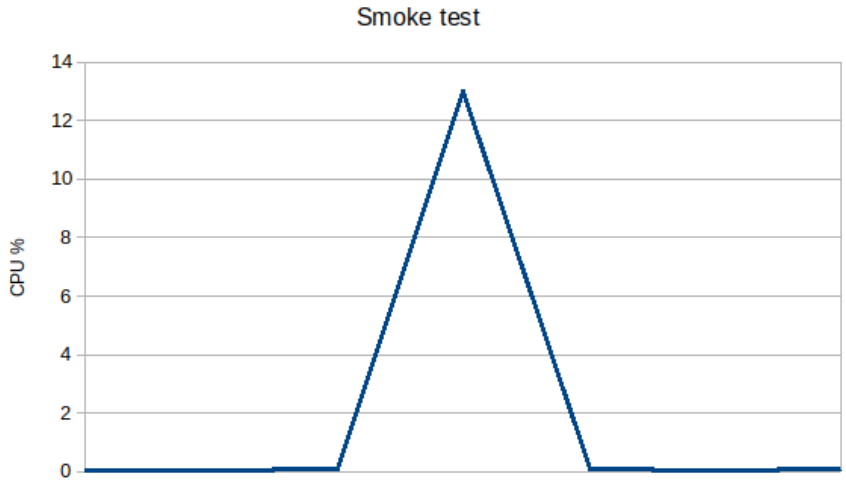


Figura 8.10: Gráfico de la correlación de porcentaje de uso de CPU con respecto al tiempo para el Smoke test.

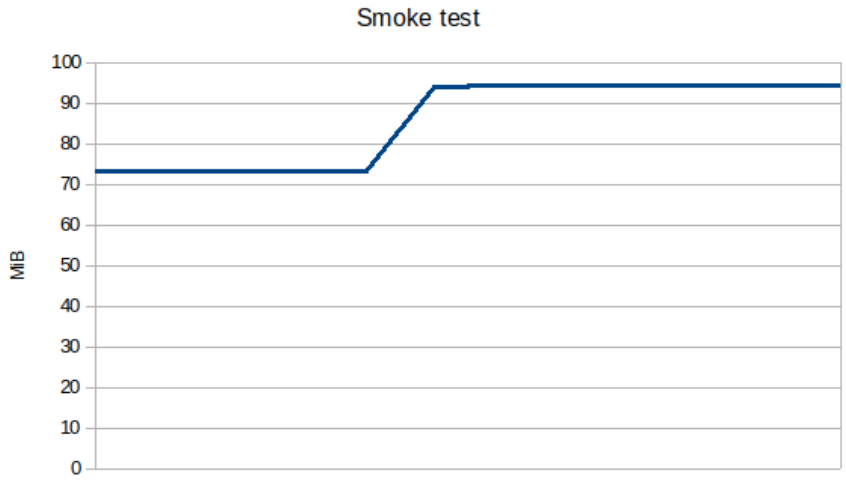


Figura 8.11: Gráfico de la correlación entre consumo de memoria en mebibytes (MiB) con respecto al tiempo para el Smoke test.

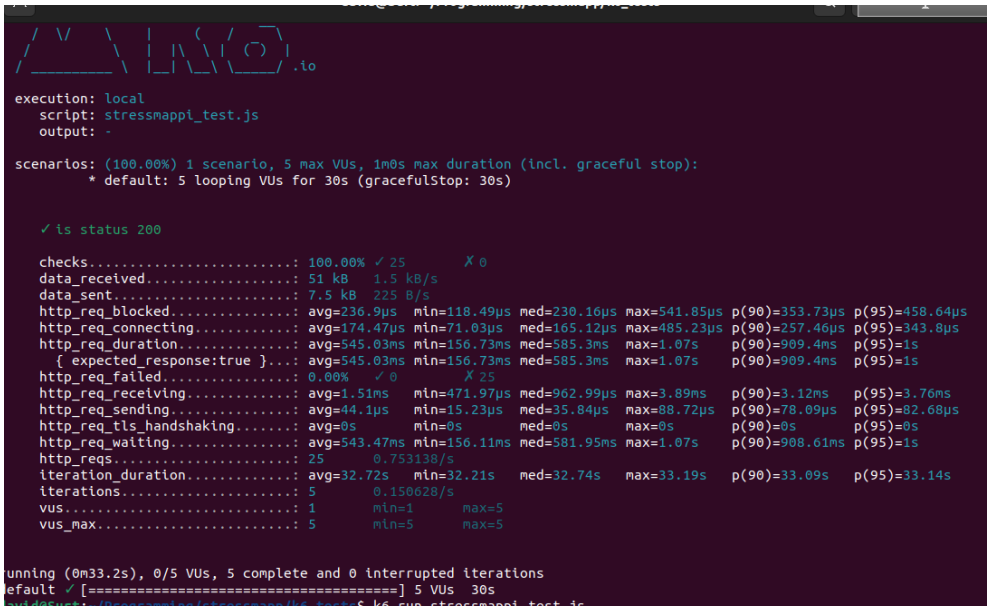


Figura 8.12: Resultados del Smoke test con k6.

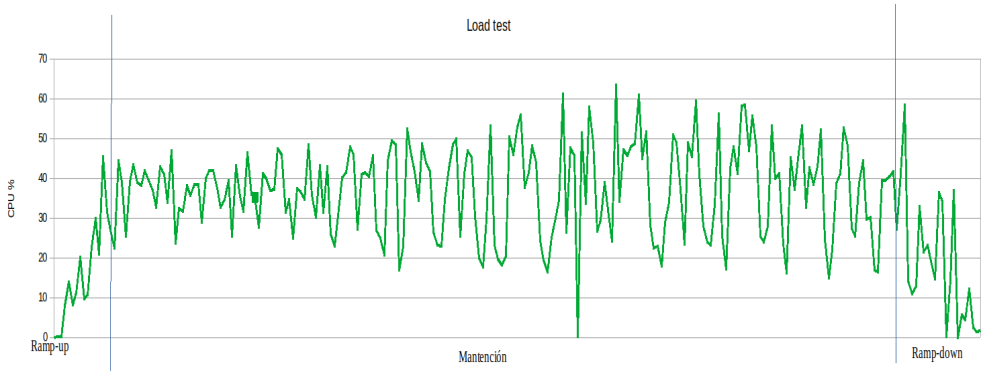


Figura 8.13: Gráfico de la correlación de porcentaje de uso de CPU con respecto al tiempo para el Load test.

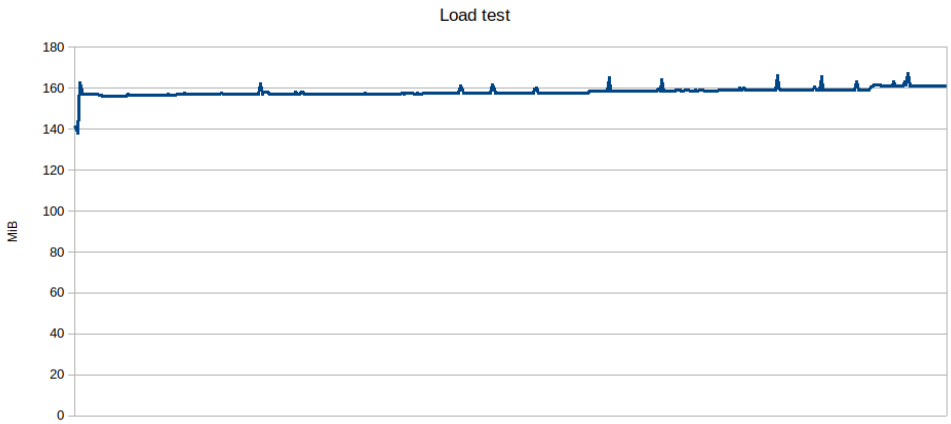


Figura 8.14: Gráfico de correlación de consumo de memoria en mebibytes (MiB) con respecto al tiempo para el Load test.

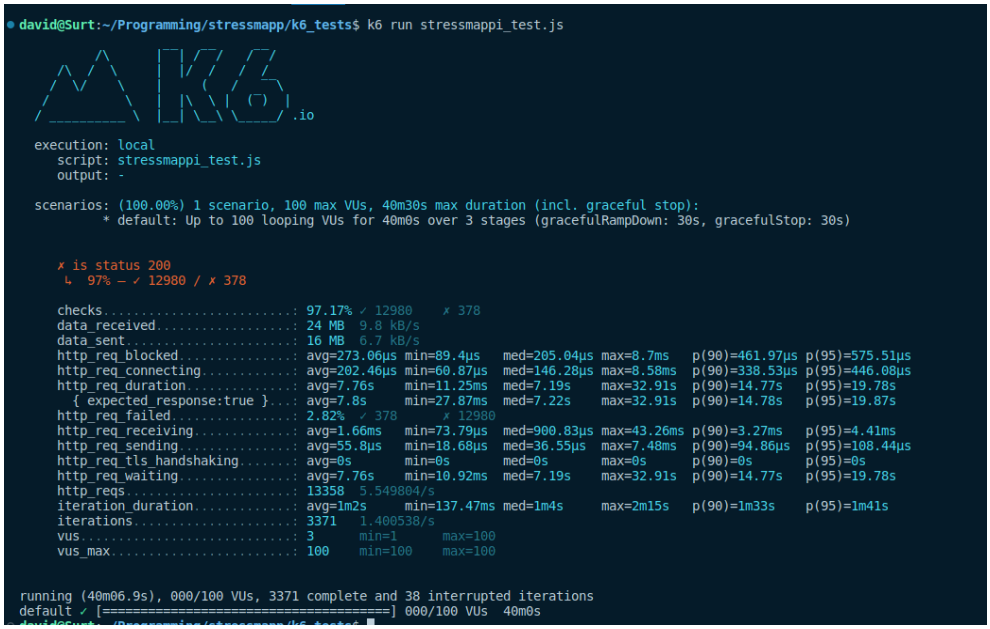


Figura 8.15: Resultados del Load test con k6.

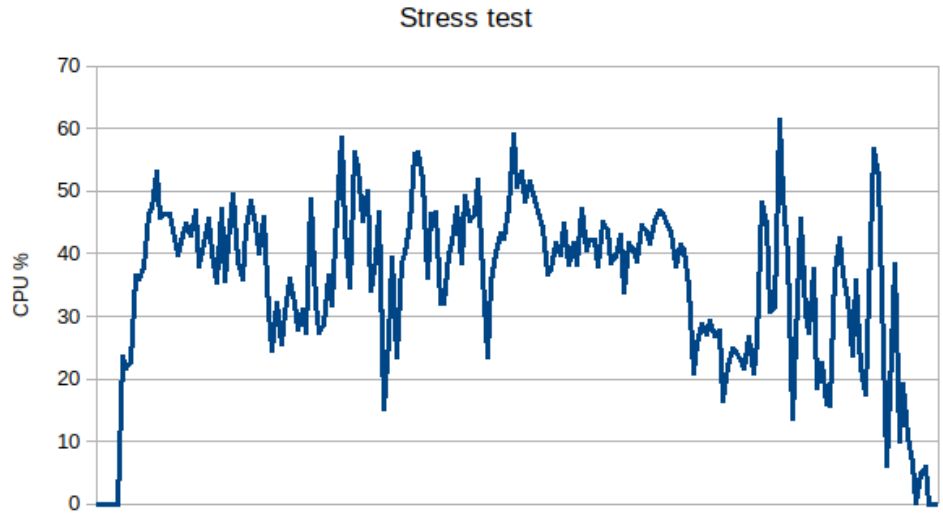


Figura 8.16: Gráfico de correlación de porcentaje de uso de CPU con respecto al tiempo para el Stress test.

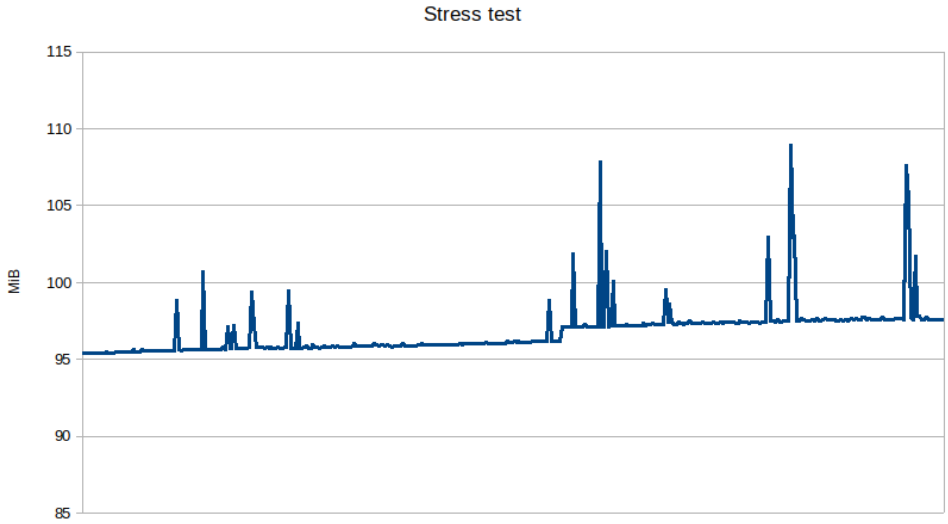


Figura 8.17: Gráfico de correlación de consumo de memoria en mebibytes (MiB) con respecto al tiempo para el Stress test.

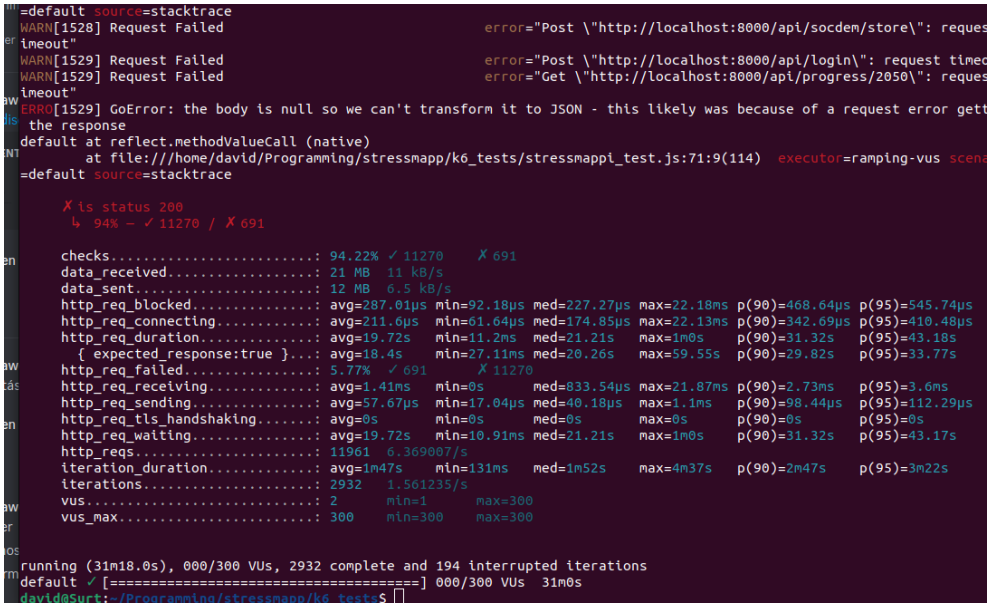


Figura 8.18: Resultados del Stress test con k6.

8.4 Interfaz del test N-Back

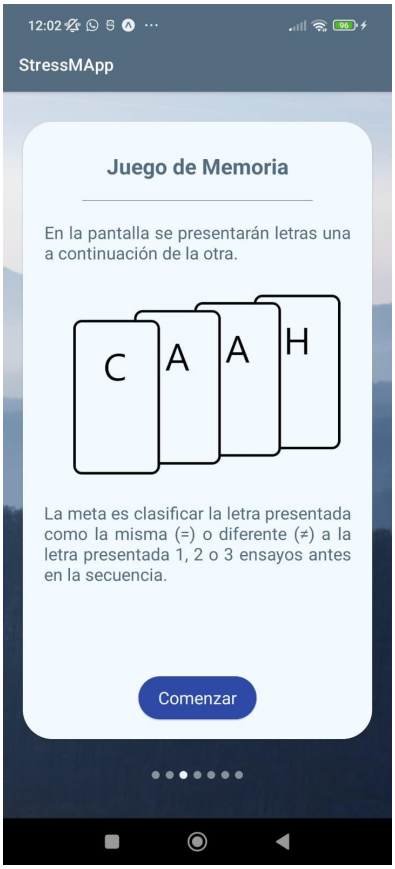


Figura 8.19: Interfaz introductoria al test N-Back.

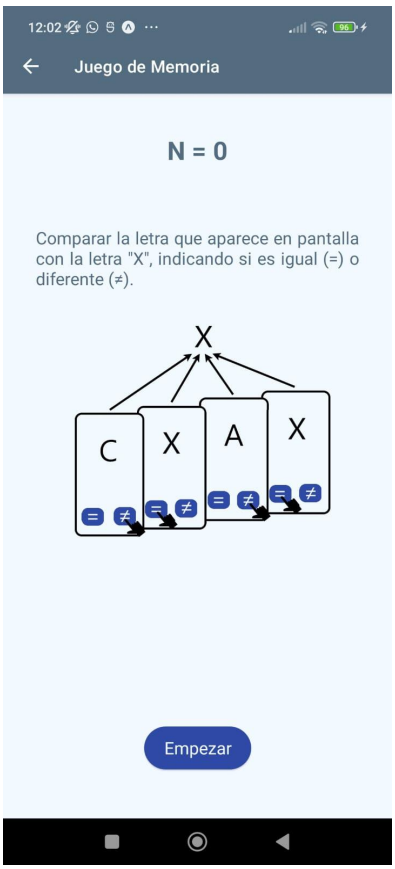


Figura 8.20: Instrucciones del test N-Back.

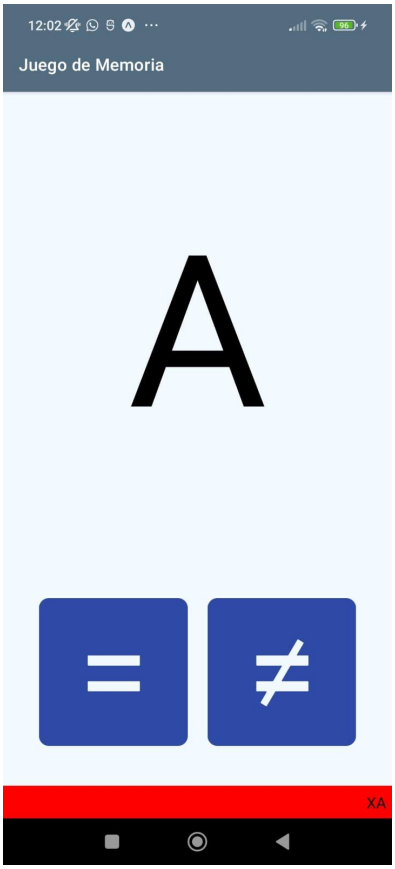


Figura 8.21: Interfaz de una prueba del test N-Back, mostrando la secuencia de letras en la parte inferior de la pantalla con fines de desarrollo.

8.5 Interfaz del test Stress Induction

Figura 8.22: Primera parte de instrucciones del test Stress Induction.

Figura 8.23: Segunda parte de las instrucciones del test Stress Induction.

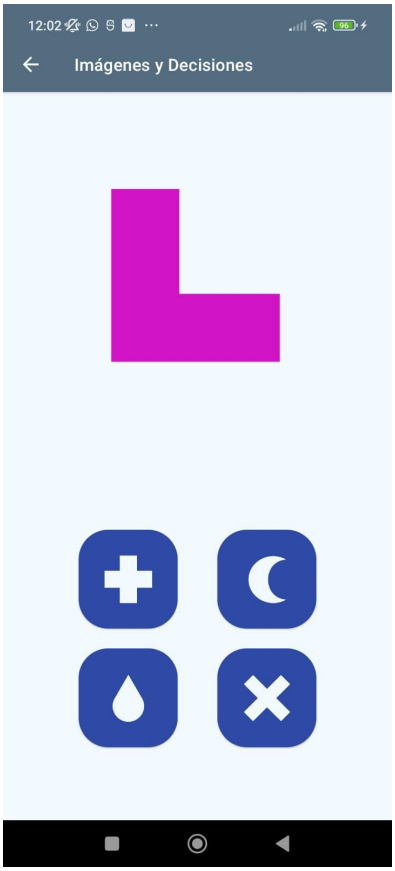


Figura 8.24: Interfaz de una prueba del test Stress Induction.

8.6 Interfaz del test Explore and Predict

Figura 8.25: Interfaz introductoria del test Explore and Predict.

Figura 8.26: Interfaz de prueba de exploración del test Explore and Predict.

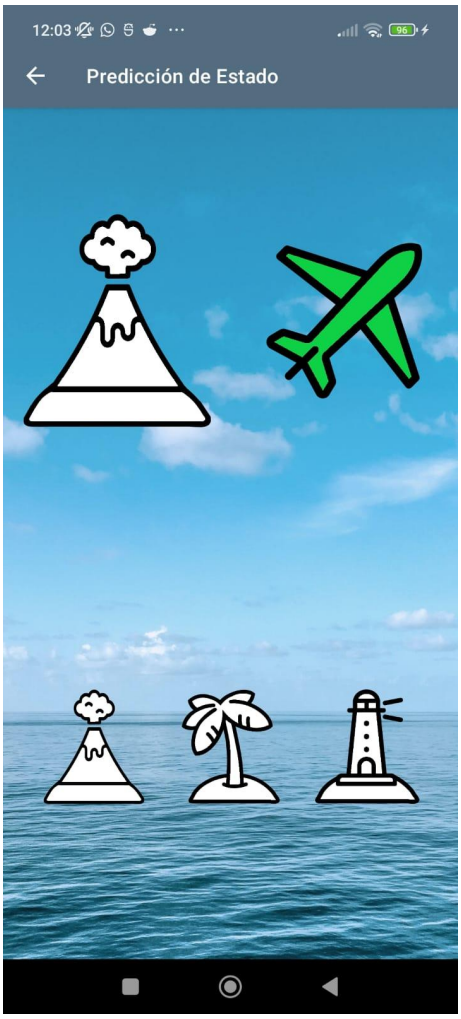


Figura 8.27: Interfaz de una prueba de predicción del test Explore and Predict.

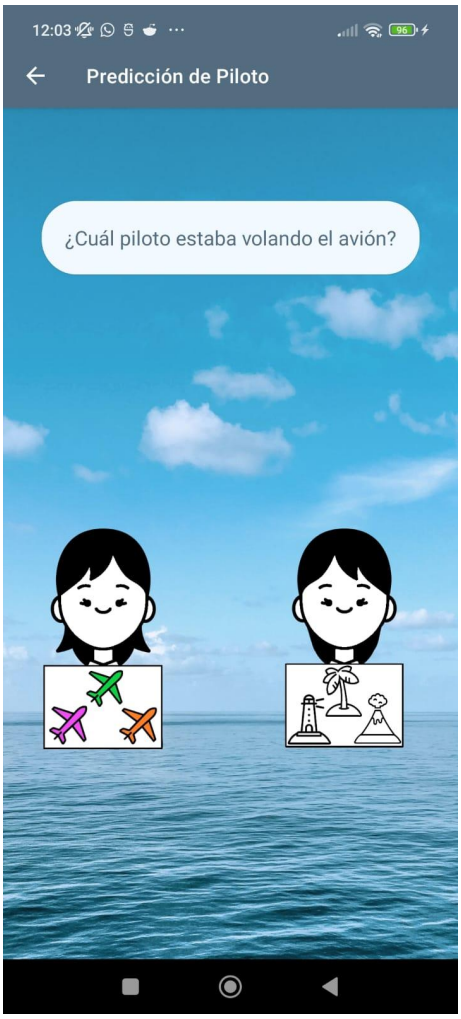


Figura 8.28: Interfaz de una prueba de predicción de piloto del test Explore and Predict.