UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

INCA-COMBOARD:
CAPTURA Y ANÁLISIS DE COMUNICACIONES INTER ESPECIES

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

CAMILA ESTER LABARCA ROSENBLUTH

PROFESOR GUÍA:
JÉRÉMY BARBAY

PROFESOR CO-GUÍA:
SERGIO OCHOA DELORENZI

MIEMBROS DE LA COMISIÓN:
JOSÉ URZÚA REINOSO
MATÍAS TORO IPINZA

SANTIAGO DE CHILE
2023

Resumen

En los últimos años, el campo de la interacción entre animales y computadoras ha experimentado un crecimiento notable. Una tendencia emergente dentro de este dominio es la utilización de dispositivos de *Augmentative and Alternative Communication* (AAC), empleados con el fin de enseñar a animales no humanos (ANH), a complementar su comunicación con palabras humanas. El diseño de estos dispositivos representan un desafío, debido a la manera en que estudiamos y entendemos la comunicación entre especies.

Si bien se han observado resultados alentadores con animales que han aprendido a utilizar dispositivos AAC, uno de los desafíos predominantes en el estudio de comunicación con ANH, es el registro preciso de los eventos de interacción. Las técnicas de registro actuales son tanto costosas como poco prácticas, lo que limita la capacidad de los investigadores para realizar análisis profundos, pertinentes y obtener conclusiones significativas sobre el comportamiento y las preferencias comunicativas de los animales.

La automatización del proceso de registro de interacciones mediante *logs* podría ofrecer una alternativa de solución eficiente al problema anterior. Haciendo uso de registros automáticos, los investigadores podrían tener un acceso más fácil y detallado a los datos, permitiendo una mejor comprensión de la comunicación animal, y apoyando de mejor manera sus actividades científicas.

Para abordar este desafío, se diseñó e implementó una aplicación web que no solo replica las funcionalidades de dispositivos AAC, sino que también guarda automáticamente los registros de interacción de varios sujetos. Esta herramienta, al automatizar el proceso de registro de interacciones (lo que anteriormente se hacía manualmente), busca ser una solución integral para apoyar las labores de investigadores y profesionales en este ámbito.

La aplicación desarrollada se deberá someter a pruebas con expertos en *Animal Computer Interaction* (ACI) y en psicología animal. Además, para asegurar su efectividad y utilidad en un entorno real, se probará el *Communication Board* con dos sujetos animales, registrando sus interacciones y luego presentándolas a los investigadores a través de la interfaz diseñada.

La solución propuesta promete ser una herramienta esencial de apoyo a la investigación de la comunicación de animales no humanos. La automatización del proceso de registro simplifica la tarea de los investigadores y abre nuevas posibilidades en el estudio del comportamiento comunicacional de distintas especies. Aunque la evaluación inicial ha sido positiva, a futuro se deben realizar más pruebas con una muestra más grande de animales, y de esa manera mejorar esta herramienta de apoyo.

Dedicado a todos los animales que se han pasado por mi vida.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Problema abordado

Augmentative and Alternative Communication (AAC) abarca todas las formas de comunicación aparte del lenguaje oral [2]. En individuos humanos con dificultades en el habla, los dispositivos electrónicos de AAC son ampliamente utilizados. Desde 2018, estos dispositivos también han ganado popularidad entre los guardianes de animales no humanos [11]. Los dispositivos empleados para animales no humanos incluyen aplicaciones móviles o web para animales pequeños, como aves [6] (ver Figura 1.1), y botones de gran tamaño colocados en el suelo para animales más grandes [11] (ver Figura 1.2), aunque también se ha observado que algunos perros pueden utilizar pantallas táctiles [18].

Figura 1.1: Ave interactuando con una aplicación móvil de AAC, específicamente *CommBoards App*. Ejemplifica el uso de dispositivos electrónicos de AAC en animales pequeños.

Figura 1.2: Perro frente a un conjunto de botones de gran tamaño, representando una variante de AAC diseñada para animales más grandes. Esta imagen ilustra cómo animales de mayor tamaño pueden comunicarse utilizando herramientas táctiles en lugar de aplicaciones electrónicas.

A pesar de la creciente adopción de estos dispositivos por varios animales no humanos, aún no se ha encontrado una forma práctica, fácil, precisa y eficiente de generar registros de estas interacciones; particularmente, sin afectar el proceso de comunicación. El registro detallado y sistemático de las interacciones de los animales con los dispositivos AAC es fundamental para comprender plenamente su capacidad comunicativa, y su proceso de aprendizaje.

Estos registros no solo ayudan a los guardianes y entrenadores a monitorear el progreso de los animales, y adaptar sus métodos de enseñanza, sino que también proporcionan datos valiosos para los investigadores que buscan entender la cognición animal, y su habilidad para utilizar herramientas comunicativas artificiales.

Generalmente, los guardianes humanos graban a los sujetos cuando interactúan con los dispositivos para obtener algún tipo de registro debido a esta necesidad. Sin embargo, este proceso no es automático, y demanda más tiempo por parte del guardián o de los investigadores para generar las grabaciones, y luego analizarlas y extraer la información de ellas.

Se han hecho intentos para abordar esta problemática, por ejemplo el de la empresa Fluent Pet [14], que en 2023 lanzó nuevos botones conectados a wifi para generar registros. Sin embargo, esta solución resultó costosa y presenta desafíos de escalabilidad.

Cuando nos referimos a la falta de escalabilidad de los dispositivos de Fluent Pet, apuntamos a la complejidad y coste que supone expandir el sistema para adaptarse a animales que requieran o utilicen una mayor cantidad de botones. Un sistema escalable debería permitir la adición de nuevos botones o funcionalidades sin grandes costes adicionales o complicaciones técnicas. En el caso de perros que emplean hasta 80 botones, por ejemplo, los costes y la logística asociados a la expansión pueden resultar prohibitivos para muchos usuarios.

1.2. Objetivos de la memoria

El propósito de esta memoria fue abordar el registro automático de interacciones entre un ANH y una interfaz física que podría tener diferentes formatos. Específicamente, se desarrolló un *communication board* configurable, que se puede utilizar con animales pequeños o animales de mayor tamaño, y que permite registrar las interacciones con esta aplicación en un *backend*.

El sistema también implementa una interfaz web para el *backend*, que facilita el análisis de la comunicación de animales no humanos. Se espera que el desarrollo de este trabajo beneficie tanto a los animales no humanos, sus guardianes y a los científicos que estudian la interacción en el ámbito de animal computing.

Si bien los sistemas de comunicación de este tipo ya existen, el generar un registro automatizado de las interacciones ayuda en la investigación del alcance de la comunicación entre animales no humanos y humanos. A través de los diversos *logs* se espera identificar diferentes patrones de comportamiento y de aprendizaje, para así comenzar a sacar conclusiones más concretas.

Este objetivo general antes definido se dividió en objetivos específicos, que son presentados a continuación.

1. Mejorar la aplicación web de *communication board*, desarrollada anteriormente por el prof. Jérémy Barbay (ver Figura 1.3), haciendo de esta un tablero altamente configurable que se adapte tanto a animales no humanos pequeños, como cotorras, como a animales de mayor tamaño, como perros conectándose a un teclado.
2. Incorporar una conexión entre la aplicación web de *communication board* y el *backend*. Esta conexión tiene como finalidad registrar las interacciones que, en el futuro, podrán ser utilizadas por diversas aplicaciones de monitoreo.
3. Facilitar la recopilación y análisis de registros de comunicación de los animales no humanos, a través de una interfaz web. Esta última debe permitir a los investigadores seleccionar y filtrar los *logs* según diversos criterios.
4. Facilitar la recopilación y análisis de registros de comunicación de los animales no humanos, utilizando una interfaz web que permite a los investigadores descargar todos los *logs* para poder hacer análisis más complejos con herramientas externas.
5. Proporcionar una interfaz intuitiva para visualizar las frecuencias de uso de palabras y otros patrones de comunicación, para eventualmente cada aplicación que se utilice para monitorear.

La Figura 1.3 presenta con un diseño básico y estático, con palabras predefinidas que no pueden ser modificadas, añadidas, eliminadas o movidas. Cada tarjeta tiene asociada una grabación única y constante, mostrando la necesidad de evolución hacia un diseño más flexible y configurable.

Figura 1.3: *Communication Board* original desarrollado por el prof. Jérémy Barbay

1.3. Resumen de la solución

Una aplicación web que registre los *logs* ayudaría a los usuarios (dueños de mascotas, entrenadores, y a los investigadores de *Animal Computer Interaction* (ACI) y de psicología animal). Particularmente facilitaría el análisis de los resultados, y les ayudaría a obtener conclusiones sobre el nivel de comunicación alcanzado por los animales no humanos. Para tratar de abordar ese desafío, se propuso desarrollar una aplicación web que registre los *logs* de varios sujetos, de manera que permita a estas personas seleccionar los *logs* según el sujeto observado, la especie, el guardián, rangos de fecha y otros criterios relevantes, y visualizar las frecuencias de uso de palabras.

Considerando eso, hemos diseñado, implementado y evaluado una solución que busca mitigar el problema planteado, desarrollando la aplicación "InCA Comboard", que registra los *logs*, además de un *frontend* para los investigadores. Los desafíos principales de su desarrollo corresponden a que esta aplicación debía ser altamente configurable, para cumplir con las diferentes necesidades de los posibles animales no humanos que la utilizarían, además del hecho de que ésta debía guardar *logs* de manera remota, en vez de solo localmente, conectándose a un *backend*.

Para evaluar esta solución, se probó la aplicación "InCA Comboard" con dos cotorras argentinas, permitiendo que tanto los guardianes como algunos investigadores puedan utilizar la aplicación para analizar los *logs* de sus animales no humanos. Esto nuevamente representó un desafío computacional y operativo importante, para poder probar aplicaciones web con animales no humanos.

1.4. Estructura del documento

A continuación (Capítulo 2) se describe la situación actual, donde se explica en más detalle el problema y su contexto. Además, se presentan ejemplos de soluciones que se acercan a la propuesta desarrollada en esta memoria, y una selección de los aspectos de interés de tales soluciones. Luego describiremos el diseño de nuestra solución (Capítulo 3), su implementación (Capítulo 4), y su validación (Capítulo 5). Finalmente, concluimos en el Capítulo 6 con un resultado del trabajo logrado, una discusión de sus limitaciones y de sus extensiones de cara al futuro.

Capítulo 2

Situación actual

Antes de explorar las soluciones diseñadas e implementadas, resulta esencial comprender el concepto de *Augmentative and Alternative Communication* (AAC) (Sección 2.1). A su vez, es crucial analizar los avances y resultados anteriores en cuanto al uso de estas técnicas con animales no humanos (Sección 2.2). En este contexto, se destacarán los estudios realizados por Christina Hunger (Sección 2.2.1) y Jennifer Cunha (Sección 2.2.2), quienes han investigado y documentado aplicaciones y resultados de estas técnicas.

Finalmente, se proporcionará un panorama de las tecnologías existentes que han sentado las bases para el desarrollo de este semestre, culminando con una síntesis de la soluciones existentes (Sección 2.3).

2.1. Augmentative Alternative Communication (AAC)

La comunicación desempeña un papel vital en la vida humana, permitiéndonos expresar nuestras necesidades, sentimientos, deseos e intercambiar información con otros. Sin embargo, hay individuos que enfrentan desafíos en el desarrollo del lenguaje hablado o que, debido a condiciones médicas o discapacidades, tienen limitaciones para comunicarse verbalmente. Para abordar estas limitaciones, se originó el campo de la *Augmentative and Alternative Communication* (AAC).

AAC engloba estrategias, técnicas y herramientas diseñadas para complementar o sustituir la comunicación verbal en aquellos con dificultades para hablar o comprender el lenguaje [2]. Esta área es diversa e incluye desde sistemas basados en símbolos y gestos manuales hasta soluciones avanzadas que se apoyan en dispositivos electrónicos. El propósito subyacente es brindar a quienes tienen problemas de comunicación un medio eficaz para expresarse, conectarse con otros y tener una participación activa en la sociedad.

Generalmente, AAC se clasifica en dos categorías principales: sin ayuda y con ayuda [2]. El AAC sin ayuda abarca las estrategias que no requieren de ningún instrumento o herramienta externa, como el lenguaje corporal y las expresiones faciales. Por otro lado, el AAC con ayuda comprende las estrategias que se basan en herramientas externas, desde objetos e imágenes

simples hasta tecnologías avanzadas. En la actualidad, la tecnología ha desempeñado un papel crucial en este ámbito, con dispositivos como computadoras, tabletas electrónicas y celulares, para los cuales se han desarrollado tableros de comunicación para representar el lenguaje, además de aplicaciones para hablar a través de mensajes de texto o para dibujar.

2.2. AAC en animales no-humanos

A partir del marco de comunicación alternativa y aumentativa (AAC) establecido para los seres humanos, se ha evidenciado en años recientes que varios animales no humanos también han demostrado la capacidad de adaptarse a estas formas de comunicación. Este fenómeno cobró notoriedad en el 2018 con la creación de la página *How They Can Talk* [10], una comunidad vinculada a *FluentPet* [14]. En este espacio, las personas comparten experiencias y evidencias de cómo diferentes animales han adoptado estrategias de AAC.

Desde su inicio, la popularidad de la página ha experimentado un crecimiento significativo. Para junio de 2023, *FluentPet* acumula más de 70,000 seguidores en *Instagram* [9] y supera los 8,000 seguidores en *Facebook* [8]. A continuación, se expondrán dos ejemplos destacados en este ámbito. Primero, se abordará el trabajo de Christina Hunger (Sección 2.2.1) con la perra Stella [11], y luego se presentará el estudio de Jennifer Cunha (Sección 2.2.2) con la cacatúa Ellie [6].

2.2.1. Animales de tamaño mediano

En su libro *How Stella Learned to Talk* [11], Christina Hunger dio a conocer como ella, usando técnicas de AAC, logró un nivel más alto de comunicación con la perra Stella. Gracias a este libro se popularizó esta forma de comunicarse con perros, sin embargo, Hunger no fue pionera en este ámbito.

En 2007, Rossi et al [15] describen una de las primeras propuestas de investigación de este tema. En su artículo explican cómo lograron enseñar a la perra Sofía a usar un teclado de lexigramas, es decir, un teclado de símbolos donde cada uno tenía un significado. Después de un entrenamiento básico, el perro aprendió a pedir diferentes objetos o actividades utilizando los lexigramas. Los resultados de este experimento los registraron manualmente para su análisis, con los cuales observaron que los lexigramas se utilizaban de una manera apropiada e intencional de acuerdo con el contexto motivacional por el sujeto. Un ejemplo de un episodio de uso que demuestra lo anterior corresponde al siguiente; se deja un oso de peluche en una mesa. Sofía mira al oso y camina hacia la mesa. Se para en dos patas para apoyarse en la mesa mientras observa el oso para luego caminar hacia el teclado y presionar el símbolo de *toy*. Regresa a la mesa y se turna mirando al guardián y al juguete hasta que el guardián se lo pasa. Rossi et al [15] observaron otras situaciones como esta, y con base en estos resultados sugirieron que los perros son capaces de aprender un sistema de signos asociados a diferentes objetos o actividades.

Años más tarde, Cristina Hunger publicaría su libro [11]. En este, como se mencionó

anteriormente, ella explica cómo, usando técnicas de AAC logró comunicarse con la perra Stella. Cristina Hunger es una terapeuta del habla y lenguaje. En su trabajo como terapeuta, Cristina utilizaba tableros de comunicación para ayudar a sus pacientes a comunicarse [12]. Al adoptar a la perra Stella, a comienzos del año 2018, Cristina Hunger empezó a notar en la perra joven habilidades de comunicación similares a las de los niños cuando estos están cerca de comenzar a hablar. Al ser una terapeuta del habla, naturalmente, se preguntó si la forma de comunicarse no verbalmente de Stella se podía llevar más lejos. Fue de esta forma que decidió probar hasta dónde podía llegar. Para ello, compró botones de distintos colores con un diámetro de 10 centímetros, los cuales pueden tener grabaciones personalizadas.

Este tipo de dispositivos corresponden a una estrategia de AAC ya desarrollada que, hasta el momento, se usaba con humanos. Sin embargo, Cristina decidió probar con Stella, dejando los botones en el suelo. De a poco, Stella comenzó a ser capaz de reconocer las diferentes palabras y a utilizar los botones ella misma. Con el tiempo su guardiana fue agregando más y más palabras que ella fue aprendiendo de a poco.

Al día de hoy Stella es capaz de entender y utilizar 45 palabras en botones que se mantienen fijos en una tabla, combinando hasta 5 palabras juntas para expresar mejor lo que quiere decir [12].

Gracias a su libro y sus redes sociales, las capacidades de Stella se dieron a conocer mundialmente. De esta manera, diversas personas comenzaron a replicar esto con sus propios animales no-humanos. La compañía *Fluent Pet* [14], en colaboración con los investigadores del proyecto *How They Can Talk* [10] creó una réplica de los botones utilizados por Stella, pero un poco más pequeños para que se necesite menos fuerza para presionarlos.

Para obtener datos de los botones presionados, los guardianes suelen grabar a los animales no-humanos. A veces, incluso se deja una cámara fija hacia la tabla de botones. Se pueden ver varios ejemplos de esto en el foro de *How They can Talk* donde la gente comparte como se comunican sus animales no humanos [10]. Esto no es muy eficiente para el análisis, ya que implica tener que ver muchos vídeos y hacer un registro manual de lo sucedido. A la fecha de junio de 2023, la misma compañía *Fluent Pet* [14] ya diseñó nuevos botones que automatizan este proceso de *logs*. Estos botones se conectarán vía wifi a un celular para recibir los registros. Sin embargo, estos botones son una solución demasiado costosa, por lo que no es escalable, ya que existen animales no-humanos que usan hasta 80 botones [7].

2.2.2. Animales de tamaño pequeño

Aun cuando las cotorras son conocidas por su habilidad de aprender y repetir palabras humanas habladas, su habilidad para comunicarse realmente se continúa estudiando. Dentro de estos estudios se encuentra el de Cunha y Roads [6]. En este estudio, los guardianes lograron enseñar exitosamente a algunas aves a comunicarse vía una interfaz en una pantalla táctil. Describen en su artículo como se le enseñó una cacatúa Goffin a usar una aplicación comercial de un tablero de comunicación para *Android*. Dicen que a través de condicionamiento asociativo el sujeto logró aprender a presionar imágenes representando elementos en categorías para comidas, brebajes, actividades, objetos e interacciones.

La aplicación utilizada en este caso en particular corresponde a la aplicación *CommBoards App* [17], la cual fue originalmente diseñada para humanos con dificultades de comunicación. En esta se ayuda al usuario a comunicar vía tarjetas con imágenes que al ser presionadas emiten una palabra. Es una aplicación altamente configurable, en la cual se pueden agregar todas las palabras necesarias y dividir estas mismas en subcategorías. Las tarjetas se pueden crear, mover y eliminar de la aplicación por el usuario, pero estas acciones están protegidas por un *safe mode* para evitar que el sujeto que usa la aplicación a modo de comunicación la pueda configurar. En la Figura 2.1 se puede observar una imagen de esta aplicación.

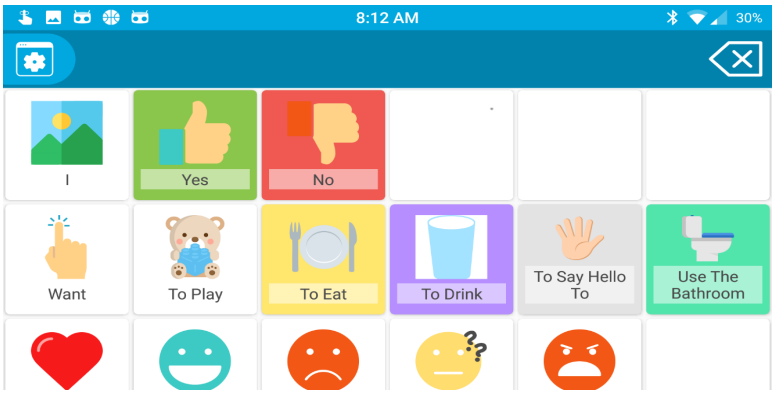


Figura 2.1: Interfaz de la aplicación *CommBoards App*, diseñada originalmente para humanos con dificultades de comunicación.

La cacatúa Goffin a la cual se le enseñó a utilizar esta aplicación corresponde a Ellie. Ellie, en ese entonces una cacatúa hembra de ocho años, ya tenía historial participando en otras investigaciones y en esta participó voluntariamente. Sus sesiones de entrenamiento eran de entre 3 a 5 por semana y duraban de 20 a 25 minutos. Luego del entrenamiento se pudo observar que Ellie logró aprender a usar la aplicación correctamente, además de que sus oportunidades de comunicación por lo general tenían sentido.

Aun cuando esta aplicación es altamente configurable y, como se vio en el estudio de Cunha y Roads [6], es posible usarla con una cacatúa, esta no tiene ninguna forma de guardar registro de lo presionado por el sujeto. De esta forma, al igual que en el caso de los botones, el trabajo de los *logs* queda como un trabajo manual.

2.3. Características destacables y deseables de soluciones existentes

Como se mencionó anteriormente, ya se ha visto el uso de técnicas de AAC en animales no humanos. Sin embargo, esto solo ha podido ser comprobado con base en vídeos e historias hechas por los guardianes de tales animales. Esto no es escalable a muchos sujetos. De esta forma surge la necesidad de generar *logs* basándose en estos dispositivos, para así tener una forma correcta de validar y analizar la comunicación.

Aunque varias soluciones en el mercado ofrecen características útiles, ninguna parece

tener todas las características ideales en un solo paquete. A continuación, se detallan las características más notables y deseables identificadas en soluciones existentes.

1. **Palabras configurables:** Tanto en los botones como en la aplicación, mencionados anteriormente, las palabras emitidas al presionar el dispositivo son configurables. En los botones se puede grabar las veces que sea necesario lo que se quiere que un botón en particular diga y en la aplicación, *CommBoards App*, es posible agregar, eliminar y mover las palabras que se deseen en el tablero, también permitiendo grabar el contenido.
2. **Categorías:** En la aplicación, *CommBoards App*, existe la posibilidad de tener las palabras organizadas por categoría. De esta forma, el usuario debe dirigirse a la categoría deseada para buscar la palabra que quiere decir. Esto con los botones no es posible, pero algunos guardianes sí lo simulan de cierta forma distribuyendo los botones agrupados en diferentes categorías.
3. **Mover botones de posición:** Los botones físicos se suelen dejar en una posición fija dentro de un tablero. Sin embargo, en caso de que fuera necesario, estos se pueden cambiar de posición dentro del mismo tablero sin problema. Esto mismo también ocurre en la aplicación *CommBoards App*, en la cual se puede modificar la disposición de las tarjetas sobre el dispositivo electrónico.
4. **Imágenes en botones:** En la aplicación *CommBoards App* cada tarjeta tiene una imagen que debería hacer referencia a la palabra asociada a esta misma. Los botones generalmente no vienen con imágenes, y simplemente varían entre ellos por el color, aunque no sería muy difícil para el guardián pegar imágenes sobre los botones.
5. ***Safe mode*:** En la misma aplicación mencionada anteriormente también se tiene un modo seguro. Este funciona bloqueando algunas opciones de configuración con algún tipo de prueba que dificulte a un usuario que no es el guardián, sea un niño o un animal no humano, configurar la aplicación. La prueba puede ser alguna ecuación matemática o algo similar, evitando que la aplicación sea des configurada por accidente.
6. ***Logs*:** Los nuevos botones de *Fluent Pet* registran *logs* de la interacción conectándose vía wifi.
7. **Escalable:** Tanto los botones normales como la aplicación son soluciones escalables. En estas se puede comenzar con poco vocabulario y luego incrementarlo muy fácilmente.

La Tabla 2.1 presenta un arreglo resumido de las soluciones descritas en la Sección 2. Se puede observar que ninguna solución cuenta con una interfaz web que ayude al análisis de los *logs*, y solo una de las soluciones implementa los *logs*. En las siguientes secciones se presentará la solución desarrollada que propone llenar los vacíos descritos previamente.

	ComBoards App	Botones normales	*Fluent pet* botones wiFi	Solución Propuesta
1. Palabras configurables	✓	✓	✓	✓
2. Categorías	✓			
3. Mover botones	✓	✓	✓	✓
4. Imágenes en botones	✓			
5. *Safe mode*	✓			✓
6. *Logs*			✓	✓
7. Escalable	✓	✓		✓

Tabla 2.1: Comparación de características relevantes entre distintas plataformas.

Capítulo 3

Diseño de Solución

En este capítulo, se presenta el diseño integral de la solución propuesta, la cual busca facilitar el análisis de la comunicación de animales no humanos. Antes de abordar el diseño, fue esencial definir claramente los requisitos de la solución. Estos se detallan, junto con sus justificaciones, en la Sección 3.1.

Con base en estos requisitos, se estructuró el diseño en tres componentes principales. En primer lugar, se diseñó y desarrolló un *frontend* para los animales no humanos (Sección 3.3). Este *frontend* consistió en un *Communication Board* básico, con diferentes funciones para hacerlo modificable al usuario, que en este caso corresponde a animales no humanos.

La segunda parte del diseño corresponde al *backend* (Sección 3.4). Este *backend* es el encargado de registrar las interacciones con el *frontend* mencionado anteriormente y así conectar con la última parte desarrollada; el *frontend* para los investigadores (Sección 3.5).

Esta última parte, más específicamente, corresponde a una aplicación donde los investigadores pueden ver un registro de las interacciones e interactuar con estas de diferentes maneras. Además, la arquitectura completa de la solución se describe detalladamente en la Sección 3.2.

3.1. Requisitos de la solución

El diseño propuesto en este trabajo de título tiene como fundamento una serie de requisitos claves, derivados de las necesidades identificadas en el contexto de uso. Estos requisitos, que establecen las bases y objetivos del sistema, guían las decisiones de diseño y desarrollo de las tres principales componentes de la solución.

El objetivo general es crear un sistema que facilite la comunicación de animales no humanos y permita a los investigadores analizar estas interacciones. Con este propósito, se establecieron los siguientes requisitos orientadores:

1. *Frontend* animales no humanos:

- Agregar o eliminar tarjetas según las necesidades comunicativas del animal usuario, considerando que distintas especies podrían tener vocabularios variados.
- Ajustar manualmente por el guardián las propiedades del sonido de la tarjeta, incluyendo tono, velocidad e idioma, adaptándose a las respuestas de diferentes animales.
- Implementar medidas de seguridad para evitar que los animales modifiquen configuraciones, garantizando su enfoque solo en la comunicación.

2. *Backend logs*
   - Los *logs* registrados en esta parte de la aplicación deben incluir la fecha, el guardián, el sujeto (animal no humano utilizando la aplicación), la palabra presionada y la cuenta, en caso de que el usuario se haya registrado. Todos estos datos son necesarios para poder hacer un análisis posterior completo, ya que se podrán ver patrones de palabras para un sujeto específico en un día en particular. Dentro de la fecha también se debe incluir la hora, porque esto puede ayudar a ver como se distribuyen las palabras utilizadas por un sujeto a lo largo del día.
3. *Frontend* investigadores
   - Acceso visual, en forma de tabla, a todos los registros recopilados en el *backend*.
   - Filtrado de *logs* por diferentes campos, facilitando la identificación de patrones según fechas, sujetos o palabras específicas.
   - Exportación de *logs* a archivos CSV para análisis más exhaustivos.

3.2. Arquitectura de la solución

Antes de explicar cada componente de la solución en detalle, es esencial comprender la estructura global de la misma. Un diagrama de arquitectura ilustra claramente los componentes principales y sus interacciones (ver Figura 3.1). Este diagrama subraya la función específica de cada componente y cómo se interrelacionan entre sí y con los usuarios.

Tal como se mencionó anteriormente, el sistema consta de los siguientes tres componentes principales:

1. *Frontend* animales no-humanos: Este *frontend* permite a los animales interactuar con el *communication board*, enfatizando la adaptabilidad y la flexibilidad del mismo. Esta parte de la aplicación se basa en el *communication board* desarrollado anteriormente durante el semestre de primavera del año 2022, pero con mejoras en la capacidad de modificación. Más detalles en la Sección 3.3.
2. *Backend logs*: Este *backend* se encarga de registrar de manera eficiente todas las interacciones generadas por los animales en el *frontend*. Este componente garantiza una gestión de datos robusta y escalable, y su diseño en la nube permite un acceso y actualización en tiempo real. El diseño detallado se encuentra en la Sección 3.4.

3. *Frontend* investigadores: Este *frontend* ofrece a los investigadores una interfaz intuitiva para acceder y analizar los registros almacenados en Firebase. Los detalles de esta implementación se exploran en la Sección 3.5.

En la Figura 3.1 se destaca claramente cómo cada componente se comunica con los demás y cómo los usuarios interactúan con las diferentes partes del sistema. Se observa que el *frontend* para animales no humanos y el *frontend* para investigadores se conectan al *backend*, que actúa como intermediario para gestionar los registros y la comunicación entre estos dos extremos.

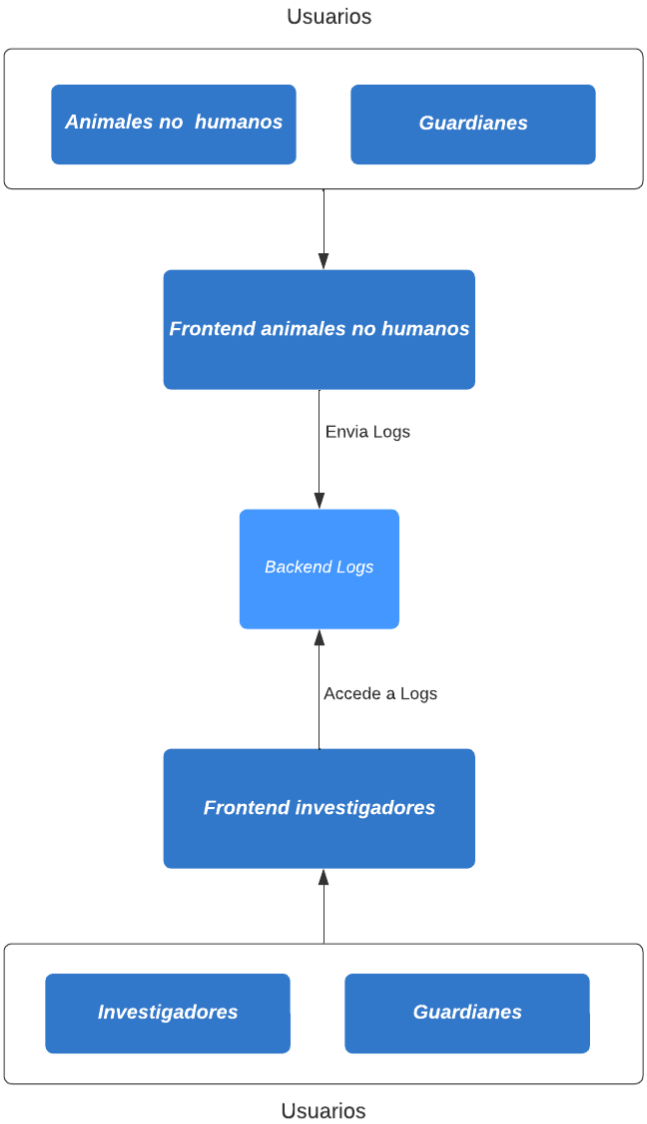


Figura 3.1: Diagrama de arquitectura de la solución propuesta.

Este diseño asegura una separación clara de responsabilidades entre las diferentes partes del sistema y permite un flujo eficiente de datos y acciones entre los usuarios y los registros de interacciones.

3.3. *Frontend* animales no-humanos

El *frontend* para animales no-humanos correspondió a la parte más extensa del trabajo. Es por esto que el diseño de este mismo se dividió en tres partes principales. En primer lugar, se tiene el diseño del *communication board* en sí. Esta es la esencia del *frontend*, ofreciendo la interfaz principal a través de la cual los animales interactúan con la aplicación. Los pormenores y características específicas de este diseño se abordan en la Sección 3.3.1.

En segundo lugar, se encuentran los *logs* locales, mostrando solo las 10 últimas interacciones en aras de brindar retroalimentación inmediata y un seguimiento reciente de la actividad (Sección 3.3.2). Finalmente, se tiene el diseño de la sección de configuración de la aplicación. Conscientes de que los animales no humanos requieren de una supervisión y guía, se diseñó una sección especial para que los cuidadores o guardianes humanos puedan ajustar y personalizar la experiencia de sus protegidos en la aplicación (Sección 3.3.3).

3.3.1. *Communication board*

La idea principal detrás del diseño de este *frontend* se basó en la aplicación altamente configurable *CommBoards App* [17], descrita en la sección anterior. En dicha aplicación, se presenta una serie de tarjetas con una imagen y un nombre, dispuestas sobre un fondo blanco.

A diferencia de la aplicación original, se decidió no usar imágenes, considerando la eficiencia y el rendimiento. Guardar imágenes en la base de datos no solo añade complejidad al desarrollo, sino que también puede afectar la velocidad de carga para el usuario. Además, como se observó en experimentos anteriores con animales, la posición de las tarjetas es un indicador suficiente para su reconocimiento. Vale la pena mencionar que, basados en experiencias previas, se ha demostrado que animales como las aves pueden aprender a usar el tablero basándose únicamente en la ubicación de las tarjetas.

El *Communication Board* desarrollado en este trabajo corresponde en una serie de tarjetas que contienen una palabra y una tecla. Estas tarjetas son todas de color gris con un borde negro y texto verde y se posicionan como una grilla sobre un fondo blanco, alineadas a la izquierda (ver Figura 3.2). En estas tarjetas se optó por no usar imágenes, como se mencionó anteriormente, considerando la eficiencia y el rendimiento.

Conforme se agregan tarjetas, éstas se organizan horizontalmente hasta llegar al límite y luego continúan en la siguiente fila. La elección de estos colores se basó en la aplicación ya existente de *What Is More* [16], la cual ya se había probado con cacatúas y cotorras argentinas, por lo que se sabía que con estos colores eran capaces de diferenciar las tarjetas.

Figura 3.2: Tarjetas *InCA Comboard*.

Cada una de estas tarjetas emite un sonido al ser presionadas o al presionar la tecla asociada. El sonido emitido debería ser el del nombre asociado a la tarjeta que se emite con una voz sintetizada o con una grabación hecha por el usuario.

3.3.2. *Logs* locales

Además de emitir el sonido cuando se presionan las tarjetas, también se registra un *log* cuando ocurre esta interacción. Estos registros, denominados *logs* locales, son cruciales para entender el contexto y la frecuencia de las interacciones en un periodo corto, capturando las últimas 10 acciones. Se encuentran en una tabla que se abre manteniendo presionado por 3 segundos un botón que dice "Local Logs".

La tabla resultante muestra el nombre de la tarjeta presionada junto con la fecha y hora exacta de la interacción. Además, estos *logs* están asociados a una lista desplegable donde se puede cambiar la intención de la interacción entre las siguientes 4 categorías: "Modeling by guardian", "Non intentional", "Intentional but unclear meaning", "Intentional and clear meaning" (ver Figura 3.3).

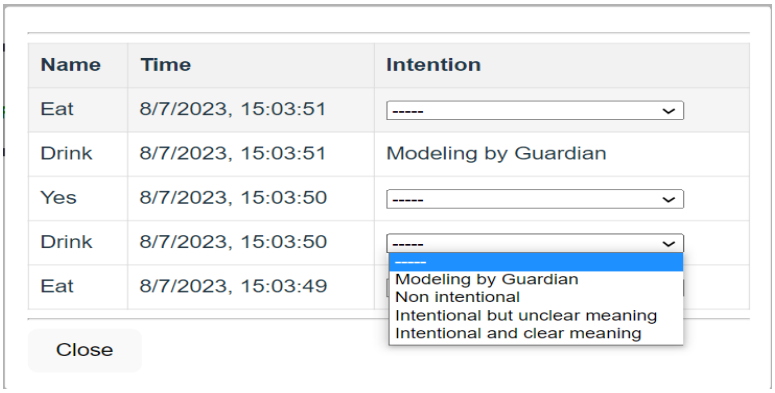


Figura 3.3: *Logs* locales.

Para facilitar el proceso de etiquetado, se incorporaron atajos. Por ejemplo, si se quiere registrar la interacción como "Modeling by guardian" automáticamente sin tener que seleccionar la opción en la lista desplegable, se puede simplemente presionar la tecla "SHIFT" junto con la tarjeta deseada, o su tecla asociada. De esta forma quedará automáticamente registrado como "Modeling by guardian".

Para el caso de los usuarios que no tienen acceso a la tecla “SHIFT”, también se puede recrear esto usando el botón llamado “Shift” en la esquina superior izquierda (ver Figura 3.4). Al presionar este botón por tres segundos y luego presionar una de las tarjetas se registrará de manera automática como “Modeling by guardian”. Si la siguiente tarjeta que se presione se quiere que también se registre como “Modeling by guardian” se debe volver a presionar el botón por tres segundos.

Para bloquear este efecto y que todas las tarjetas que se presionen a continuación se registren como “Modeling by guardian”, al lado del botón anterior también se encuentra el botón “Caps Lock” (ver Figura 3.4). Este botón se activa presionándolo tres segundos, y al estar activado todas las tarjetas que se presionen se registran con la intencionalidad mencionada anteriormente. Para desactivar este botón se debe presionar nuevamente por tres segundos.

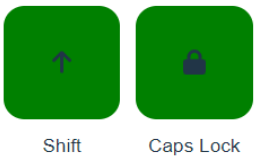


Figura 3.4: Botones “Shift” y “Caps Lock”.

3.3.3. Configuración

Para maximizar la adaptabilidad y personalización de la aplicación, se ha incluido una sección de configuración que permite a los usuarios adaptar la interfaz y funcionalidades a sus necesidades y preferencias específicas.

En la esquina inferior derecha de la aplicación se pueden observar dos botones; el botón “Local Logs”, que ya se explicó anteriormente, más a la derecha, y el botón de “Settings” (ver Figura 3.5). Este botón se activa manteniéndolo presionado por tres segundos, de la misma forma que el anterior, con lo que se abrirá un modal. Este requisito se incluye como un método de seguridad para evitar que los animales no-humanos puedan modificar la aplicación.

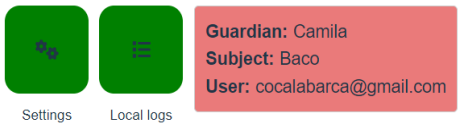


Figura 3.5: Botones “Local Logs” y “Settings”.

Dentro del modal se encuentran las diferentes opciones de configuración de la aplicación, donde se pueden personalizar sus funcionalidades (ver Figura 3.7). A continuación se listan y describen estas mismas funcionalidades.

1. Las primeras dos entradas de texto sirven para modificar el nombre del guardián y del sujeto utilizando la aplicación. Al modificar esto se visualizan estos datos al lado derecho del botón "Local Logs" (ver Figura 3.5) y se podrá observar el cambio también en los *logs*, es decir, el *log* se guardará asociado a ese guardián y sujeto.

2. Las dos siguientes partes del modal sirven para modificar la disposición del tablero. En la entrada de texto "Cards per row" se puede aumentar o disminuir la cantidad de tarjetas que se despliegan en cada fila. Luego se encuentra un interruptor con la etiqueta "Move cards". Al activar este interruptor, las tarjetas se pueden mover por el tablero simplemente seleccionando una tarjeta y moviéndola de posición, y al desactivarlo mantienen esta posición quedando estáticas.

3. Botones:

   - "Login with Google":
     Como dice el nombre del botón, este sirve para iniciar sesión con una cuenta *Google*. Al presionar este botón se abre una ventana emergente donde se puede ingresar con una cuenta existente de *Google* normalmente. Una vez se ingresó a la cuenta, el botón pasa a ser un botón que dice "Logout", para que el usuario pueda cerrar sesión. Cuando se está registrado en una cuenta, todo el estado de la aplicación quedará guardado para la próxima vez y además se puede ver el email de la cuenta en la misma esquina que el guardián y el sujeto, por lo que, de la misma forma, también se verá el cambio en los *logs*.

   - "Add New Card":
     Este botón sirve para agregar nuevas tarjetas al tablero. Al presionar este botón se abre un modal con un formulario donde se debe ingresar el nombre del sonido y la tecla asociada. Luego se presiona "Add" y la nueva tarjeta se agregará al tablero. Se vacía el formulario automáticamente, y se puede seguir agregando tarjetas hasta que se quiera salir del modal.

   - "Audio Settings":
     Al presionar este botón se abre un nuevo modal en el cual se pueden ver diferentes opciones de audio (ver Figura 3.6). En primer lugar, se puede ver un interruptor bajo la pregunta "Use audio recordings?". Al mover este interruptor se activa la opción de grabar tus propios sonidos. Si se activa, el formulario de "Add new card" cambia un poco, ya que ahora también te pide grabar el sonido. Estas grabaciones no se guardarán en la cuenta porque, al igual que las imágenes, guardar las grabaciones en la base de datos dificultaba demasiado el desarrollo y se decidió priorizar la implementación de otras funcionalidades. Sin embargo, se decidió que de todas formas era importante tener la opción de grabar el sonido en caso de que el sujeto utilizando la aplicación haya aprendido las palabras con una voz en particular y no logre comunicarse con la voz sintetizada. El resto de las configuraciones están asociadas a la voz sintetizada. A esta se le puede cambiar el idioma (inglés o español), el tono de voz y la velocidad de la voz. Al final del modal se encuentra un botón para probar como queda la voz con estos cambios.

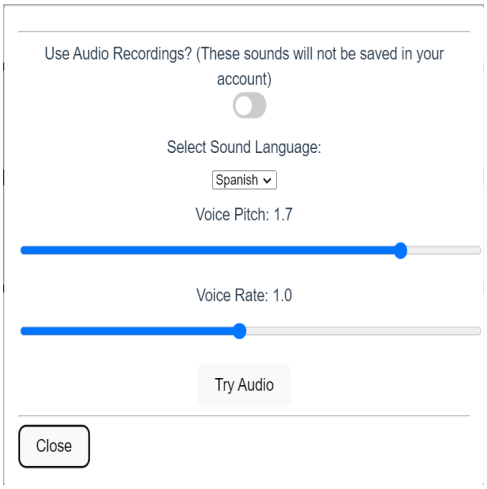


Figura 3.6: Modal de configuración de audio.

- "Save":
  Este botón guardará todos los cambios y cerrará el modal.

La idea de tener todas estas funcionalidades dentro de "Settings" y no en la aplicación general se justifica con que la mayor parte del tiempo no se harán estas acciones. La idea principal de la aplicación es funcionar como un *Communication Board*, por lo que la mayor parte del tiempo se usará solamente la parte de presionar tarjetas con sonidos. Es por esto que se quiere que las tarjetas utilicen la mayor parte del espacio, mientras que el resto de las funcionalidades no deberían ocupar demasiado espacio de la pantalla.

Dentro de este mismo modal se puede encontrar también la información de contacto. Se puede ver el nombre del creador junto con el repositorio de *Github* y un email de contacto. Además, se encuentra una dirección web a una encuesta para poder evaluar la aplicación.

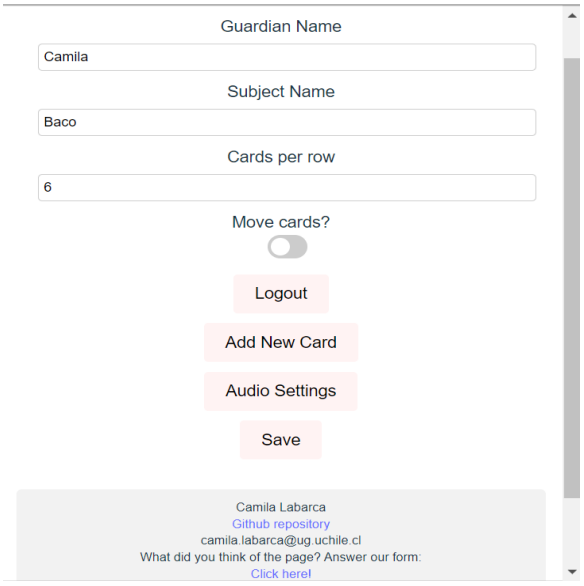


Figura 3.7: Modal de configuración.

Finalmente, se tiene otra funcionalidad de configuración de la aplicación que no se encuentra en el modal. Esta funcionalidad corresponde a la posibilidad de hacer *click* derecho sobre una de las tarjetas. Al hacer esto se abrirá otro modal en el cual se muestran dos opciones: "Change key" y "Delete Card".

Al presionar la primera opción, se observa un formulario con solo una entrada de texto donde se puede cambiar la tecla asociada a la tarjeta. Si se presiona la segunda opción, el modal pregunta si estás seguro de que deseas eliminar la tarjeta, y si se presiona que sí, esta tarjeta desaparece del tablero.

3.4. *Backend logs*

El diseño del *backend* de la aplicación busca capturar y almacenar información relevante sobre las interacciones del usuario con el *frontend* descrito anteriormente. Se definió un modelo de datos que captura distintos tipos de interacciones y configuraciones del usuario.

La estructura principal para capturar las interacciones con el *frontend* para animales no humanos consta en la siguiente estructura:

1. *Interactions*: Corresponde a los *logs* de interacción con las tarjetas del tablero.
    - ID (Identificador único de la interacción).
        - *date*: Fecha de la interacción con el tablero, con formato día/mes/año, hora: minutos: segundos.
        - *sound*: Nombre del sonido que fue presionado, lo cual en general corresponde a lo que dice la grabación.
        - *guardian*: Nombre del guardián que se registró en el *frontend*. En caso de que no se haya registrado ningún guardián, este se guardará como "Anonymous" en la base de datos.
        - *subject*: Funciona de la misma forma que *guardian* y también corresponde a un campo que se definió en el *frontend*. Guarda el nombre del sujeto.
        - *user*: Correo electrónico del usuario registrado en el *frontend*. Nuevamente, en caso de no estar registrado, se guardará como "Anonymous".
2. *RandomClick*: Corresponde a los *logs* de las interacciones (*clicks*) con el resto de la aplicación. No se incluyen las interacciones con las tarjetas del tablero ni con los botones de la aplicación cuando estos son activados. Tampoco se consideran las interacciones cuando se tiene abierto un modal. Las interacciones que no se incluyen corresponden a las que se supone fueron hechas por el guardián de manera intencional.
    - ID (Identificador único de la interacción).
        - *date*: Fecha de la interacción con el tablero, con formato día/mes/año, hora: minutos: segundos.

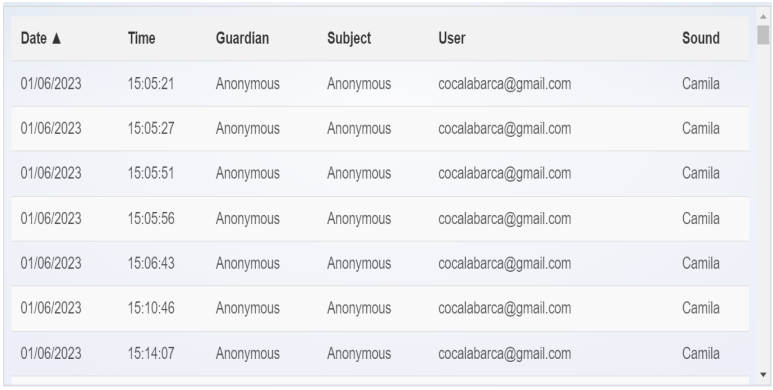
  - *guardian*: Nombre del guardián que se registró en el *frontend*. En caso de que no se haya registrado ningún guardián, éste se guardará como "Anonymous" en la base de datos.
  - *subject*: Funciona de la misma forma que *guardian* y también corresponde a un campo que se definió en el *frontend*. Guarda el nombre del sujeto.
  - *user*: Correo electrónico del usuario registrado del *frontend*. Nuevamente, en caso de no estar registrado, se guardará como "Anonymous".

El modelo de datos también incluye una estructura para almacenar a los usuarios registrados, además de configuraciones específicas y el estado actual de la aplicación descrita anteriormente para un usuario registrado:

1. *Users*: Usuario registrado con "Google" en el *frontend* anterior.
   - UserID (Identificador único del usuario).
     - Email: Correo electrónico del usuario registrado.
     - *name*: Nombre asociado al correo electrónico.
     - *role*: Rol del usuario.
2. *CommBoardState*: Estado de la aplicación para un usuario en particular.
   - UserID (Identificador único del usuario).
     - *guardian*: Nombre del guardián que se registró en el *frontend*.
     - *subject*: Nombre del sujeto que se registró en el *frontend*.
     - *keyboard*: Teclas asociadas a las tarjetas del tablero del *communication board*
     - *sections*: Diccionario que incluye el sonido asociado a cada tecla.
     - *CardsPerRow*: Cantidad de tarjetas por fila.
     - *lang*: Idioma de voz sintetizada.
     - *pitch*: Tono de voz sintetizada.
     - *rate*: Velocidad de voz sintetizada.

3.5. *Frontend* para los investigadores

Este *frontend* para los investigadores corresponde a una página simple que muestra los *logs* guardados en *Firebase*. La página en sí corresponde a una tabla simple con 6 columnas; "Date", "Time", "Guardian", "Subject", "User" y "Sound" (ver Figura 3.8). Estas columnas corresponden a las mismas que se guardaron en *Firebase*, por lo que simplemente muestran toda la información que está registrada ahí. Los títulos de cada columna son cliclable. Cuando se presionan la tabla se ordenará por esa columna, ascendentemente en primer lugar y descendentemente si se presiona nuevamente.



Date ▲	Time	Guardian	Subject	User	Sound
01/06/2023	15:05:21	Anonymous	Anonymous	cocalabarca@gmail.com	Camila
01/06/2023	15:05:27	Anonymous	Anonymous	cocalabarca@gmail.com	Camila
01/06/2023	15:05:51	Anonymous	Anonymous	cocalabarca@gmail.com	Camila
01/06/2023	15:05:56	Anonymous	Anonymous	cocalabarca@gmail.com	Camila
01/06/2023	15:06:43	Anonymous	Anonymous	cocalabarca@gmail.com	Camila
01/06/2023	15:10:46	Anonymous	Anonymous	cocalabarca@gmail.com	Camila
01/06/2023	15:14:07	Anonymous	Anonymous	cocalabarca@gmail.com	Camila

Figura 3.8: Tabla de *logs*.

Sobre la tabla se puede ver una serie de campos de entrada, uno para cada columna (ver Figura 3.9). Estos campos de entrada sirven para poder filtrar la información que se muestra en las tablas. Todos funcionan de la misma forma, escribiendo un texto, a excepción de la fecha en la cual se abre un calendario donde se puede escoger una fecha exacta o un intervalo de tiempo. Se puede filtrar por múltiples columnas a la vez y si se quisiera eliminar el filtro simplemente se debe presionar la "x" a la derecha de cada campo de entrada.

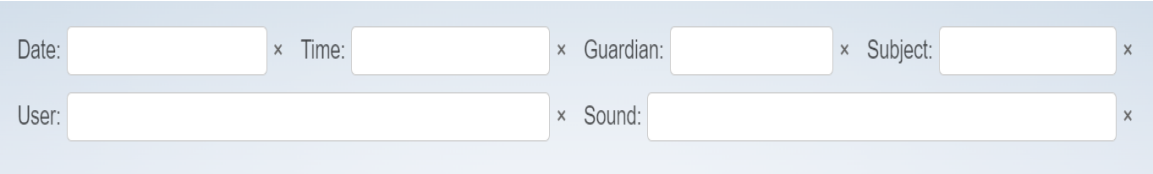


Figura 3.9: Campos de entrada para filtrar.

Finalmente, en la esquina inferior izquierda se tiene un botón con el nombre "Download". Este botón sirve para descargar los datos, por lo que al presionarlo se descargará en el computador del usuario un archivo CSV con todos los datos mostrados en la tabla. En caso de que se tenga un filtro sobre los datos, los datos descargados también tendrán ese filtro.

Capítulo 4

Implementación

La implementación de este trabajo de título, al igual que en el capítulo anterior, se dividió en tres partes; el *frontend* para los animales no-humanos (Sección 4.1), el *backend* (Sección 4.2) y el *frontend* para los investigadores (Sección 4.3). Para cada una de estas secciones se explicará en detalle las tecnologías utilizadas, además de la justificación de la elección de cada una. También, se explicará cuál fue el plan de trabajo que se siguió durante la implementación, junto con las dificultades enfrentadas en el camino.

4.1. *Frontend* para animales no-humanos

Este *frontend* fue la primera parte que se desarrolló en este trabajo de título. Durante el semestre primavera 2022 se llevó a cabo un trabajo dirigido, durante el cual se comenzó la implementación de esta parte del trabajo.

Para desarrollar esto se usó como base una aplicación ya existente. Esta aplicación corresponde al *Communication Board* desarrollado por el prof. Jérémy Barbay [3]. La figura 4.1 muestra una vista completa del *frontend* final, basado en la aplicación mencionada. Esta aplicación ya desarrollada correspondía a un *Communication Board* muy básico implementado en *Svelte*. Para la implementación del proyecto se decidió continuar con *Svelte*, por diversas razones que describiremos a continuación.

En primer lugar, al ya tener un prototipo básico desarrollado en este programa, comenzar con el desarrollo utilizando este mismo facilitaba mucho el proyecto. De esta forma se podía tener más tiempo en implementar las cosas nuevas, tales como el *backend* y el *frontend* para investigadores. En segundo lugar, considerando que la aplicación no presentaba una complejidad técnica demasiado grande, *Svelte* era suficiente para su implementación.

Finalmente, se cree que, dada la simplicidad de *Svelte*, si un usuario que no tiene tanta familiaridad con el desarrollo de software quisiera modificar la aplicación para su uso personal, lo podría hacer sin mayor dificultad. Esta última razón es importante, ya que el trabajo es de código abierto y los investigadores podrían necesitar modificar la aplicación según sus propias necesidades.

La interfaz mostrada en la Figura 4.1 presenta las mejoras y adaptaciones realizadas sobre la versión básica inicial, reflejando las capacidades y características ampliadas de la aplicación.

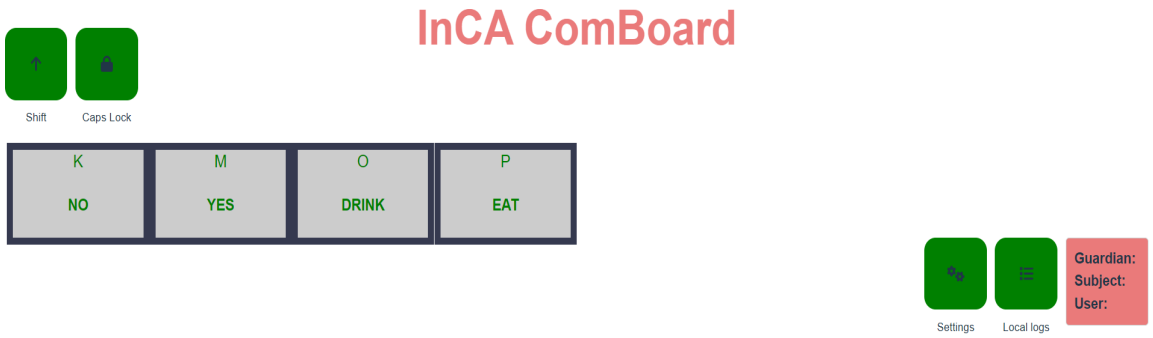


Figura 4.1: Vista del *frontend* final del *Communication Board* basado en el prototipo original desarrollado por el prof. Jérémy Barbay

Luego de tomar esta decisión, se tomó la base del prototipo ya existente y se comenzó a implementar la aplicación final. El desarrollo de la aplicación se dividió en cuatro archivos principales; "Modal.Svelte" (Sección 4.1.1), "Commboard.Svelte" (Sección 4.1.2), "Headear.Svelte" (Sección 4.1.3) y "App.Svelte" (Sección 4.1.4). A continuación se explicará cada uno de estos archivos en detalle, incluyendo bloques de código para entenderlos mejor.

4.1.1. "Modal.Svelte"

El archivo "Modal.Svelte" es responsable de crear modales para la aplicación. Estos modales son ventanas emergentes que se superponen a la interfaz principal, proporcionando información adicional o permitiendo al usuario realizar acciones específicas sin abandonar la vista actual.

Funcionamiento y lógica

La lógica principal detrás de esta componente se encuentra en el bloque "<script>", el cual contiene varias funciones y lógica para manejar la interacción del usuario con el modal.

1. **Manejo de eventos**: Al principio, la componente importa "createEventDispatcher" y "onDestroy" de Svelte. La función "createEventDispatcher" permite que el componente envíe eventos personalizados, mientras que "onDestroy" se ejecuta cuando el componente está a punto de ser destruido.
2. **Cerrar el modal**: Utilizando el "createEventDispatcher" definido anteriormente, se crea una función "close" para despachar un evento de cierre. Esto permite que otras

partes de la aplicación sepan cuándo el modal ha sido cerrado y tomen medidas en consecuencia.

3. **Manejo de teclas**: La función "handle_keydown" maneja eventos de teclado mientras el modal está abierto. Esta función es especialmente útil para mejorar la accesibilidad y la experiencia del usuario. Por ejemplo, permite cerrar el modal con la tecla "Escape" y manejar la navegación con tabulación, restringiendo el enfoque dentro del modal, es decir asegura que el usuario no pueda mover el foco fuera del modal (por ejemplo, con la tecla tabulador) mientras este está activo. (Ver Anexo 6.1)

Estructura

Dentro del cuerpo de la componente, se tienen varios elementos que definen la estructura y presentación del modal.

1. **Fondo del modal**: La capa que oscurece el contenido detrás del modal y que, al hacer clic sobre ella, cierra el modal. Esta contenida dentro de un " " con una clase "modal-background" para manejar el estilo. Además, se incluyen los eventos "on:click", que llama a la función "close" y "on:keydown" que llama a la función "handle_keydown".
2. **Contenido del modal**: Aquí es donde se muestra el contenido principal del modal. Se utiliza el mecanismo de "<slot>" para permitir la inserción de contenido personalizado en el modal para que este se pueda reutilizar desde diferentes partes de la aplicación. Además, el botón "Close" proporciona una forma intuitiva para que los usuarios cierren el modal en caso de que no quieran continuar con la acción (ver Figura 4.2)

```
<div class="modal" role="dialog" aria-modal="true" bind:this={modal}>
    <slot name="header"></slot>
    <hr>
    <slot></slot>
    <hr>
    <button autofocus on:click={close}>Close</button>
</div>
```

Figura 4.2: Contenido del modal de la componente "Modal.Svelte"

4.1.2. "Commboard.Svelte"

La componente "Commboard.Svelte" corresponde a la parte principal de la aplicación. Esta componente es fundamentalmente un teclado de comunicación en el que cada tecla representa un sonido. Además de este tablero, la componente también proporciona una serie de botones de control y varios modales que facilitan operaciones específicas.

Funcionamiento y lógica

Al igual que en la componente anterior, el bloque "<script>" contiene la lógica principal de "Commboard.Svelte". Acá se encuentran las funciones de la componente que le permiten manejar las funcionalidades de la componente. A continuación se detallan las funciones principales.

1. **Reproducción de sonido**: La función "playSound" gestiona la reproducción de sonidos de las tarjetas. Se verifica primero si la tarjeta fue presionada durante una operación de arrastrar y soltar o con un *click* derecho, en cuyo caso no se reproduce sonido. Si se decide reproducir un sonido, existen dos opciones: utilizar voz sintetizada o utilizar una grabación preexistente. En el siguiente fragmento de código se puede ver como se maneja esto (ver Figura 4.3). Finalmente, la función agrega el sonido a la lista "previousEntries" que corresponde a las 10 últimas interacciones que se muestran en los *logs* locales.

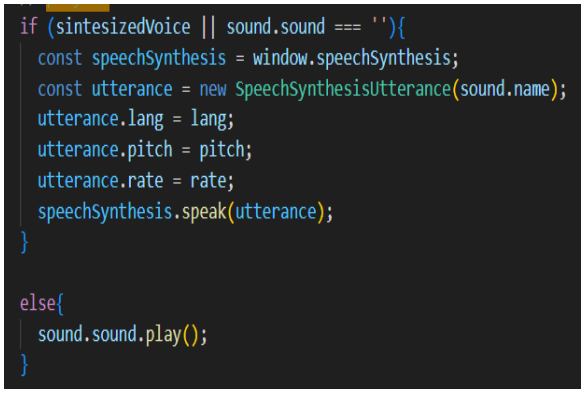


Figura 4.3: Fragmento de la función "playSound". Si la variable "sintesizedVoicce" es verdadera o no existe un sonido asociado a la tarjeta se utiliza la voz sintetizada con sus configuraciones correspondientes; en los otros casos se utiliza la grabación.

2. **Interacción con el teclado**: Para las interacciones con el teclado se tienen dos funciones; "documentKeyDown" y "documentKeyUp". La primera función llama a "playSound" en el caso de que la tecla presionada esté asociada a una de las tarjetas y que ninguno de los modales esté abierto. Además, en esta función se cambia el valor de "currentSound" al sonido que fue presionado. La segunda función simplemente cambia el valor de "currentSound" a "null".

3. **Manejo de tarjetas del tablero**: El manejo de las tarjetas en el tablero incluye tres funciones; "deleteCard", "changeKey" y "addSound". La primera se encarga de borrar una tarjeta, la segunda cambia la tecla asociada a la tarjeta y la tercera agrega una nueva tarjeta con una palabra y una tecla asociadas.

4. **Grabación**: Existen funciones para iniciar y detener la grabación de un sonido, que luego se asocia a una tarjeta específica. Esta funcionalidad permite a los usuarios grabar su propia voz o sonidos específicos en lugar de utilizar una voz sintetizada. El código detallado de esta funcionalidad se encuentra en el anexo (Ver Anexo 6.2).

5. ***Drag and Drop***: Cuando la función de mover tarjetas está activada, es decir, la variable "moveCards" es verdadera, se agregan "EventListeners" a todas las tarjetas en el tablero. Esto permite cambiar las tarjetas de posición en el tablero de comunicación utilizando los eventos "dragstart", "dragend", "dragover" y "drop", como se muestra a continuación. En el evento "dragstart" se cambia el valor de la variable"isDragging" a verdadero, el cual se vuelve a cambiar a falso en el evento "dragend". En el evento "dragover" se llama al método "event.preventDefault()" para prevenir el comportamiento predeterminado del navegador para el evento "event", que en este caso podría ser no permitir que un elemento sea soltado en ciertos lugares. Finalmente, en el evento "drop" se cambian las posiciones de la tarjeta que está siendo arrastrada y la tarjeta de destino como se muestra en el código a continuación (ver Figura 4.4).

```
if (targetIndex !== -1 && sourceIndex !== -1) {
  const updatedSections = [...sections];
  const targetCard = updatedSections[targetIndex];
  const sourceCard = updatedSections[sourceIndex];

  // Swap positions of the dragged card and the target card
  updatedSections.splice(sourceIndex, 1, targetCard);
  updatedSections.splice(targetIndex, 1, sourceCard);

  // Update the sections array in one reactivity update
  sections = updatedSections;
}
```

Figura 4.4: Fragmento de código de la funcionalidad de mover las tarjetas en el tablero. Se cambian las posiciones de las tarjetas "target" y "source" y luego se actualiza la lista de tarjetas.

6. **Interacción con Firebase**: Se tienen dos funciones que manejan esta interacción. En primer lugar, "sendData" envía a Firebase un *log* de una interacción con alguna de las tarjetas del tablero. En segundo lugar se tiene "saveCommboardState" que guarda el estado de la aplicación cuando existe un usuario registrado.

Estructura

En el cuerpo de esta componente se tienen diversos elementos que permiten al usuario interactuar con diferentes botones y modales. A continuación se describe la estructura principal de estos elementos.

1. **Botones "Shift" y "Caps Lock"**: Estos dos botones corresponden a botones de la librería *InCA Utils* [13]. Esta librería contiene una colección de utilidades para aplicaciones de *InCA lab* [5], los cuales se optó por utilizar para asegurarse de un buen funcionamiento y estilo de los botones. Ambos botones se activan al mantenerse apretados tres segundos, cambiando de valor las variables "shiftPressed" y "capsLock" respectivamente de las cuales depende el color del botón. A continuación se muestra como se utiliza uno de estos botones en el HTML (ver Figura 4.5).

```
<TrainerButton
label="Shift"
longpressTime={3000}
on:click={() => shiftPressed = true}
--background-color={shiftPressed ? "rgb(234 122 122)" : "green"}
--font-size="0.8em"
--width="3px"
><Fa icon={faArrowUp} />
</TrainerButton>
```

Figura 4.5: Botón "Shift" utilizando "Trainer Button"

2. ***Communication board***: Se visualizan todos los sonidos disponibles en tarjetas. Cada tarjeta tiene una tecla asociada y el nombre del sonido. Puedes hacer *click* en ellas para reproducir el sonido y abrir un modal para modificar las tarjetas haciendo *click* derecho. Para esto se usa un bloque "each" que crea un bloque " " para cada sonido como se muestra en la Figura 4.6.

```
{#each sections as sound, index}
  <div
    class='sound-button' on:mousedown={() => playSound(sound)}
    id = {`sound-${index}`}
    on:contextmenu={(event) => handleContextMenu(event, sound.name)}
    class:sound-button-active={currentSound === sound.name}
    >
    <small class='key'>{sound.key}</small>
    <h4>{sound.name}</h4>

  </div>
{/each}
```

Figura 4.6: Tarjetas del tablero de comunicación

3. **Modal de *local logs***: Si la variable "logsModal" es verdadera, se muestra un modal que contiene una tabla con *logs* de las últimas diez interacciones con el tablero de comunicación, donde cada *log* tiene un nombre, un tiempo y una intención. Para esto se utiliza la componente de modal creada y las tablas de HTML. Para la intención si, esta no es nula, se utiliza una etiqueta "select" como se muestra en la Figura 4.7.

```
{#if entry.intention === null}
  <select id={index.toString()} on:change={() => logSelected(index)}>
    <option value='' selected>-----</option>
    <option value='Modeling by Guardian'>Modeling by Guardian</option>
    <option value='Non intentional'>Non intentional</option>
    <option value='Intentional but unclear meaning'>Intentional but unclear meaning</option>
    <option value='Intentional and clear meaning'>Intentional and clear meaning</option>
  </select>
{:else}
  {entry.intention}
{/if}
```

Figura 4.7: Menu desplegable para seleccionar intención de interacción

4. **Modal de adición de tarjeta**: Si la variable "addCardModal" es verdadera, se abre el modal principal para añadir una nueva tarjeta, el cual utiliza la componente creada para modales y corresponde a un formulario con campos para el nombre del sonido, la tecla y opciones de grabación. Dentro de este modal se incluye una parte para comenzar y detener la grabación de un sonido solo si la variable "sintesizedVoice" es falsa, la cual se observa en el Anexo 6.3.

5. **Modal de modificación o eliminación de tarjeta**: Este modal, también creado con la componente "Modal.Svelte", da al usuario opciones para modificar o eliminar una tarjeta. Se activa cuando la variable "showModifyModal" es verdadera. Si se elige modificar, se presenta un nuevo modal con un formulario para cambiar la tecla asociada a la tarjeta, el cual depende de la variable "changeKeyModal". Si se elige eliminar, se presenta un modal de confirmación, asociado a "showDeleteModal".

4.1.3. "Header.Svelte"

La componente "Header.Svelte" está diseñada para representar una cabecera o encabezado en la aplicación. Su función principal es mostrar un título, proporcionando un diseño visual coherente y centrado para las cabeceras de las páginas o secciones dentro de la aplicación.

Funcionamiento y lógica

Dentro del bloque "<script>", hay una declaración que exporta una variable llamada "title". Esta variable actúa como una propiedad de la componente, lo que significa que cuando se utilice la componente en otra parte de la aplicación, se podrá pasarle un título específico como atributo. Por ejemplo, al usar el componente en otro archivo ".Svelte", se podrá hacer lo siguiente:

```
<Header title="Main page" />
```

Estructura

La estructura principal de la componente es bastante simple y se explicará en detalle a continuación.

1. **Elemento "<header>"**: Este elemento es una etiqueta semántica en HTML que indica que el contenido dentro de ella es un encabezado. Es una buena práctica usar esta etiqueta para cabeceras, ya que mejora la accesibilidad y la visibilidad en línea.
2. **Elemento "<h1>"**: Dentro de "<header>" se tiene un elemento " ", dentro del cual hay un elemento "<h1>" que muestra el título. El uso de "<h1>" indica que es el encabezado principal o más importante de la página o sección. El valor de "title" se interpola directamente dentro de este elemento, lo que significa que el valor de la propiedad "title" que se pasa a la componente se mostrará aquí.

4.1.4. "App.Svelte"

El archivo "App.Svelte" es la componente principal que reúne las funcionalidades y representaciones visuales de la aplicación. Esta corresponde a la principal interfaz de usuario que gestiona la autenticación, modales, configuración y presentación de datos.

Funcionamiento y lógica

En el bloque "<script>" de la aplicación se encuentran las importaciones, variables y funciones necesarias para el funcionamiento y lógica de la aplicación. A continuación se describirá este bloque en detalle.

1. **Importaciones y variables iniciales**: El código inicia con la importación de módulos y las componentes necesarias, incluyendo todo lo relacionado con Firebase. Además, se inicializan diversas variables para el manejo del estado y la configuración de la aplicación.
2. **Autenticación de Usuario**: El sistema establece un observador ("onAuthStateChanged") para el estado de autenticación del usuario. En caso de cambio, se ajusta el valor de "firebaseUser" acorde al estado actual. También se definen funciones para la autenticación ("login()") y cierre de sesión ("logout()"). En particular, el inicio de sesión utiliza el proveedor de Google y, en caso de identificar a un nuevo usuario, registra sus datos en Firestore (Anexo 6.4)
3. **Manejo de modales y configuración**: Diversas funciones han sido establecidas para el manejo de modales. Estas permiten abrir y cerrar ventanas emergentes, que podrían contener configuraciones, ajustes de audio, adición de tarjetas, entre otros, cambiando los valores de las variables asociadas a dichos modales a "false" para cerrarlos y a "true" para abrirlos
4. **Manejo de audio**: La función "tryAudio()" permite probar un texto en específico con las configuraciones de voz que se han seleccionado para la voz sintetizada.
5. **Configuración del CommBoard**: La función "saveCommBoardState" se encarga de guardar la configuración y el estado actual del tablero de comunicación en la base de datos Firestore de Firebase. Antes de guardar cualquier dato, la función verifica si el usuario está autenticado a través de la variable "firebaseUser". Una vez confirmado el usuario, la función prepara la información que se va a guardar, estructurándola y organizándola de una forma específica, la cual se puede ver en la Figura 4.8. Finalmente, la función guarda la estructura previamente preparada en la colección "commBoardStates" de Firestore bajo el identificador único del usuario.

```
const serializedState = {
    keyboard: commBoardState['keyboard'],
    keys: Object.keys(commBoardState['keys']).map((key) => ({
    [key]: {
        sound: '',
        name: commBoardState['keys'][key].name,
        key: commBoardState['keys'][key].key
    }
    })),
    sections: commBoardState['sections'].map((section) => ({
    sound: '',
    name: section.name,
    key: section.key
    })),
    guardian: guardian,
    subject: subject,
    lang: lang,
    pitch: pitch,
    rate: rate,
    cardsPerRow: cardsPerRow
};
```

Figura 4.8: Estructuración y organización de estado del tablero de comunicación que se guardara en Firebase.

Estructura

La estructura en esta componente es similar a las componentes anteriores. En esta, además de las etiquetas de siempre, se utilizan todas las componentes anteriormente descritas.

1. **Encabezado**: Lo primero que se tiene en el HTML de este archivo corresponde al encabezado de la aplicación. Para esto se utiliza la componente "Header.Svelte" descrita anteriormente.
2. ***Commmunication Board***: Utilizando la componente "Commboard.Svelte" que se explicó previamente, se crea el tablero de comunicación. A esta componente se le pasan todas las variables y funciones necesarias para su funcionamiento.
3. **Modales de configuración**: Se utiliza la componente "Modal.Svelte" para ofrecer una configuración personalizada al usuario con modales. Estos modales permiten ajustar preferencias, introducir información y configurar parámetros relacionados con el audio. El modal de configuración general de la aplicación se activa cuando la variable "settingsModal" es verdadera. Proporciona campos de entrada para el nombre del guardián y el sujeto, número de tarjetas por fila, interruptor para activar la opción para mover las tarjetas y acciones relacionadas con la autenticación del usuario, como iniciar y cerrar sesión. Dentro de este mismo modal también se observa un botón para pasar al modal de configuraciones de audio. Este modal se activa cuando "audioSettingsModal" es verdadero y en este los usuarios pueden decidir si usar grabaciones de audio o una voz sintetizada con un interruptor, seleccionar el idioma del audio en un menú desplegable y ajustar el tono y velocidad de voz con un control deslizante.
4. **Botones e información de usuario**: Finalmente, se tiene un bloque " " en el cual se encuentran los botones "Local Logs" y "Settings" junto con la información de

usuario; nombre del guardián, nombre del sujeto y correo de usuario registrado. Los botones que se utilizan corresponden nuevamente a los "Trainer Buttons" de la librería *InCA Utils* [13], ya que estos están dirigidos solamente para los guardianes.

4.2. *Backend logs*

Para poder registrar los *logs* desde el *frontend* anteriormente descrito, se debió implementar el *backend* donde estos se guardarían. Para esto, se seleccionó la plataforma *Firebase*, una elección estratégica debido a su escalabilidad y facilidad de integración con diferentes *frontends*.

Firebase, una solución ofrecida por *Google*, se destaca por su conjunto robusto de herramientas y servicios para el desarrollo de aplicaciones. En el contexto de nuestro proyecto, tres servicios clave de *Firebase* fueron esenciales: *Realtime Database*, *Firestore Database*, y *Authentication*.

Realtime Database corresponde a un servicio de base de datos en tiempo real el cual almacena datos en la nube en forma de un árbol de documentos JSON. Se sincroniza de manera automática con los clientes conectados, por lo que los cambios realizados en un cliente se reflejan de manera instantánea en los otros. Fue con este servicio que se decidió guardar los *logs* desde el *frontend* para animales no-humanos, ya que esta se usaba desde dos clientes; para guardar los *logs* y para obtener los datos desde el *frontend* de investigadores. En esta base de datos se guardaron los *logs* de las interacciones con toda la aplicación en el formato que se mencionó en el Capítulo 3 (Sección 3.4). Para enviar los *logs* desde el *frontend* de animales no humanos se utilizaron dos funciones; "sendData" y "handleClickOutside". La primera función recibe los datos a enviar y los organiza en un diccionario "data", con el formato ya mencionado, para luego guardar la interacción en Firebase con

```
database.ref("interactions").push(data);
```

La segunda función funciona de la misma manera, pero ahora en vez de "interactions" se usa "randomClick" para guardar la interacción. Luego, se obtienen los datos guardados en "interactions" desde el *frontend* de investigadores de la siguiente forma

```
await firebase.database().ref('interactions').once('value');
```

Con respecto a *Firestore*, esta corresponde a un servicio de base de datos de documentos NoSQL y orientado a documentos. Esta, a diferencia de la base de datos anterior, corresponde a una base de datos más flexible en cuanto a la organización y consulta de datos. De esta forma, se logró usar este tipo de base de datos para registrar el estado de la aplicación para animales no-humanos asociados a una cuenta de *Google*. En este estado se guardan todas las tarjetas presentes en el *communication board*, junto con sus teclas asociadas, además del nombre del guardián, el sujeto y el idioma, tono y velocidad de la voz sintetizada. Todo esto se guarda asociado al ID del usuario que está registrado, y así cuando un usuario entra en su cuenta puede utilizar la aplicación tal como la había dejado. La estructura utilizada para registrar esto corresponde a la descrita en el Capítulo 3 (Sección 3.4). Para guardar el estado en *Firestore* se utiliza la función "saveCommBoardState", que se explico en la Sección

4.1. Una vez se tienen los datos en la estructura correspondiente, en un diccionario llamado "serializedState", se mandan a Firebase con

```
db.collection('commBoardStates').doc(userId)
.set({ commBoardState: serializedState })
```

donde "userId" corresponde al id del usuario identificado. Este estado se debe obtener cada vez que ingrese un usuario, por lo que usando "onAuthStateChanged", cuando existe un usuario, se obtienen los datos con

```
db.collection('commBoardStates').doc(userId).get()
.then((snapshot) => { if (snapshot && snapshot.exists) {...} })
```

Dentro del "if" se modifican los datos para que estén en el formato correcto para utilizarlos en la aplicación.

Authentication fue el servicio utilizado para implementar el inicio de sesión para el *frontend* de animales no-humanos. Con este servicio es posible habilitar diferentes formas de iniciar sesión y para este caso en particular se habilitó la posibilidad de iniciar sesión con cuentas de *Google* ya existentes. Esta decisión se tomó por simplicidad y también debido a que el número de usuarios de *Google* es bastante grande. Para implementar esto se utilizaron las funciones "onAuthStateChanged", "GoogleAuthProvider", "signInWithPopup" y "signOut" de "firebase/auth". "OnAuthStateChanged" se utiliza como se menciono anteriormente, para obtener el estado de la aplicación, y también para cambiar la variable "firebaseUser" al usuario que inicio sesión. Luego, en la función "login", se hace uso de las siguientes dos funciones como se muestra en el código de la Figura 6.4. Finalmente, se usa "signOut" para cerrar sesión en la función "logout", donde además se cambia el valor de la variable "firebaseUser" a nulo.

4.3. *Frontend* para investigadores

Tras finalizar con éxito la implementación del *frontend* destinado a animales no-humanos y consolidar un *backend* que registra eficientemente los *logs*, se pudo proceder a la creación del *frontend* para investigadores. Era crucial que este componente comenzara una vez que los elementos anteriores estuvieran en funcionamiento para asegurar su dependencia correcta de los *logs* almacenados en el *backend*.

Para este nuevo desafío, se eligió *Svelte Kit* como el marco de desarrollo. Aunque ya se había empleado *Svelte* en el desarrollo anterior y se tenía una buena familiaridad con él, *Svelte Kit* presentó ventajas clave que motivaron su uso en este proyecto. Esencialmente, *Svelte Kit* es una extensión avanzada de *Svelte*, diseñada para ofrecer características adicionales para el desarrollo de aplicaciones web más robustas y altamente escalables.

Existieron dos razones principales para optar por *Svelte Kit* en lugar de *Svelte* puro para este proyecto. En primer lugar, durante el semestre de primavera de 2022, cuando comenzó el desarrollo del *frontend* inicial, *Svelte Kit* aún no había lanzado oficialmente su versión estable. Las variantes disponibles eran versiones alfa y beta, lo que podría haber comprometido la

estabilidad del proyecto. Sin embargo, con el lanzamiento oficial en enero de 2023, *Svelte Kit* ofreció una base sólida y confiable para esta nueva implementación. En segundo lugar, aunque el alcance inicial de este trabajo de título podría no haber requerido todas las capacidades avanzadas de *Svelte Kit*, su elección anticipa y facilita futuras expansiones y escalabilidad de la aplicación. Una vista general de la aplicación final desarrollada con *Svelte Kit* se presenta en la Figura 4.9.

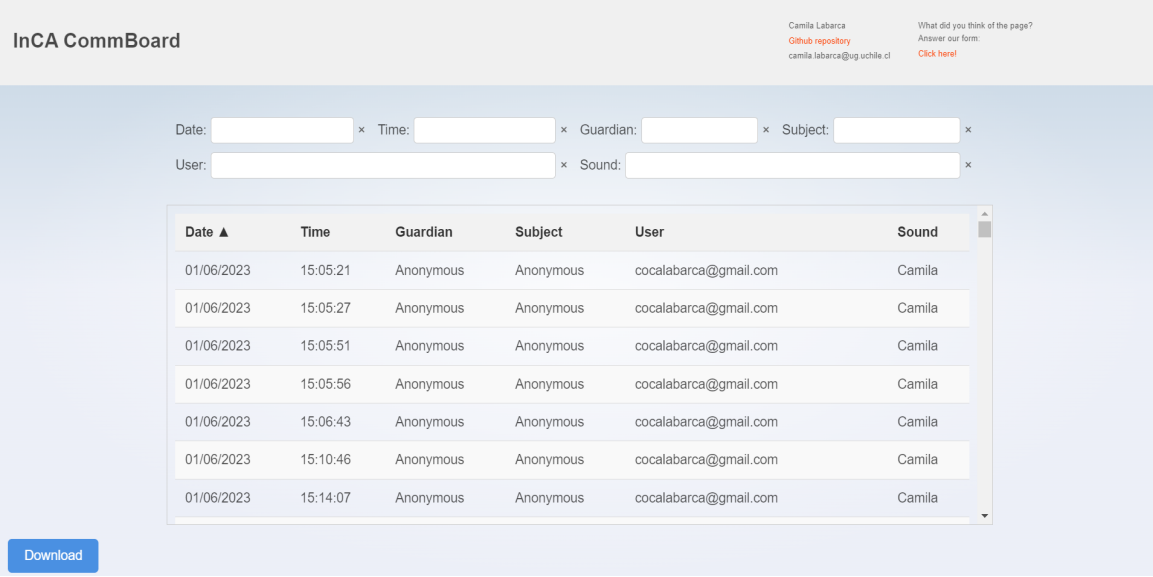


Figura 4.9: Pantalla principal de la aplicación desarrollada utilizando *Svelte Kit*, el *frontend* para investigadores.

El proceso de desarrollo comenzó desde un proyecto en blanco de *Svelte Kit*. La inicialización del proyecto se realizó a través de "npm create", proporcionando una estructura de archivos y directorios estándares. La estructura principal del proyecto se encuentra en la carpeta "src". Dentro de esta se tienen dos subdirectorios; "lib" y "routes". El primero corresponde al directorio donde se almacenan las bibliotecas y utilidades compartidas en todo el proyecto y el segundo corresponde al directorio central que aloja todas las componentes del proyecto.

Dentro de la carpeta "lib" se encuentra el archivo "firebase.js". Este archivo es esencial para la integración con Firebase, ya que en este se configuran y se inicializan los servicios de Firebase que se utilizan en el *frontend*. Dentro de la segunda carpeta, "routes", se tiene tres archivos importantes que definen las componentes principales del proyecto; "Header.svelte" (Sección 4.3.1), "+page.svelte" (Sección 4.3.2) y "+layout.svelte" (Sección 4.3.3).

4.3.1. "Header.svelte"

En esta componente se muestra información del proyecto, incluyendo el título, detalles del autor y enlaces útiles. El título se exporta dentro del bloque "<script>", por lo que cuando se utilice esta componente en alguna otra componente del proyecto se podrá pasar un título como atributo. Luego del bloque mencionado anteriormente se tiene un bloque

“ ” donde se muestra el título y el resto de la información de contacto de la creadora de la página.

4.3.2. “+page.svelte”

Este es la componente central de la aplicación. En esta se muestra la tabla de datos principal de la aplicación, junto con todas sus funcionalidades.

Funcionamiento y lógica

Las funcionalidades de las cuales se encarga esta componente están dentro del bloque “<script>”. A continuación se listan las principales.

1. **Inicialización de datos**: Se utiliza la función de Svelte “OnMount”. Al montarse la aplicación, se conecta a Firebase para recuperar los *logs* y los almacena en una variable llamada “data”.
2. **Filtrado de datos**: La página permite filtrar los *logs* por múltiples criterios, como fecha, hora, guardián, entre otros. Para esto, cuenta con una función “applyFilters()” (ver Figura 4.10). Esta función revisa cada *log* y determina si cumple con los criterios seleccionados. Los resultados se almacenan en una lista llamada “filteredData”, que es la que se muestra en la interfaz del usuario.

```
filteredData = data.filter(item => {
for (const column in filters) {
    // @ts-ignore
    const filterValue = filters[column];
    if (filterValue && !item[column].toLowerCase().includes(filterValue)) {
    return false;
    }
}
```

Figura 4.10: Aplicación de filtros en lista de *logs*.

3. **Ordenación de datos**: La tabla de *logs* puede ordenarse haciendo clic en los encabezados de las columnas. La dirección de ordenación (ascendente o descendente) se alterna con cada clic. Para manejar los *clicks* en los encabezados de las columnas se tiene la función “handleTitleClick(column)”, la cual llama a “sortTable()” para ordenar las filas. La primera función tiene como propósito determinar cómo se deben ordenar los datos de la tabla en función de la columna seleccionada. La función toma un argumento llamado “column”, que representa la columna en la que el usuario ha hecho *click*. Si el usuario ha hecho *click* en la misma columna que ya estaba seleccionada para ordenar, simplemente se invierte el orden actual. Esto significa que si los datos estaban siendo ordenados de forma ascendente, ahora se ordenarán de forma descendente y viceversa. Por otro lado, si el usuario ha hecho *click* en una columna diferente a la que estaba seleccionada previamente, la función actualiza la variable “sortColumn” para que refleje la nueva columna seleccionada. Además, establece el orden de clasificación en ascendente. La segunda función, utilizando las variables definidas anteriormente que dictan

como se ordenan las columnas, ordena la lista "filteredData" con la función "sort()" de JavaScript, como se muestra en el código a continuación (ver Figura 4.11).

```
filteredData = filteredData.sort((a, b) => {
    const aValue = a[sortColumn];
    const bValue = b[sortColumn];

    if (aValue < bValue) {
        return sortAsc ? -1 : 1;
    } else if (aValue > bValue) {
        return sortAsc ? 1 : -1;
    } else {
        return 0;
    }
});
```

Figura 4.11: Ordenación de *logs* en base a una columna en particular y un orden en particular (ascendente o descendente)

4. **Descarga de datos**: Se puede descargar la lista filtrada de *logs* en formato CSV. Para esto se crea la función "downloadCsv() ". La función se encarga de transformar los datos filtrados en este formato y facilitar la descarga al usuario (Ver Anexo 6.5).

Estructura

La estructura de esta componente se divide en tres partes; los filtros, la tabla y el botón de descarga de datos.

1. **Filtros**: Dentro de un bloque " " se encuentran todos los filtros de la tabla. Existe un filtro para cada columna de la tabla y todos se componen de un "label", un "input" y un botón para eliminar el filtro. A continuación se muestra un ejemplo de un filtro en particular (ver Figura 4.12).

```
<div class="filter-input">
    <label>Guardian:</label>
    <input type="text" bind:value={filters.guardian} on:input={() => handleFilterChange('guardian', event)} />
    <button class="clear-button" on:click={() => clearFilter('guardian')}></button>
</div>
```

Figura 4.12: HTML de filtro para filtrar por guardián.

2. **Tabla de *logs***: Dentro de otro " " se encuentra la tabla. Esta corresponde a un bloque "table" normal de HTML, pero utilizando la lista de *logs* para el cuerpo de la tabla, como se muestra a continuación (ver Figura 4.13).
3. **Botón de descarga**: El último elemento corresponde a un botón de HTML. Este tiene un atributo "on:click" donde se llama la función creada para descargar los datos.

```
<tbody>
    {#each filteredData as item}
    <tr>
        <td>{item.date}</td>
        <td>{item.time}</td>
        <td>{item.guardian}</td>
        <td>{item.subject}</td>
        <td>{item.user}</td>
        <td>{item.sound}</td>
    </tr>
    {/each}
</tbody>
```

Figura 4.13: Cuerpo de tabla de *logs*.

4.3.3. "+layout.svelte"

Finalmente, se tiene la componente "+layout.svelte". Esta componente proporciona la estructura básica de la aplicación, incluyendo el encabezado que se describió anteriormente y un espacio principal ("<main>") donde se pueden insertar otros componentes o contenido usando "<slot/>".

Capítulo 5

Validación de la Solución

Para llevar a cabo la validación del software, la aplicación para animales no humanos primero debió ser revisada por el prof. Jérémy Barbay, para asegurar que la aplicación estuviera lista para probarse con animales no humanos. Durante esta revisión se fue modificando este *frontend* para mejorar su usabilidad.

Luego de esto, se quiso validar con usuarios externos al proyecto, lo cual está en proceso actualmente. Cada aplicación se está validando con su usuario objetivo, es decir, el *frontend* para animales no humanos se valida con estos mismos y el *frontend* para investigadores se valida con sus guardianes. Para validar el primero de estos dos se hará uso de los *logs* que incluye la aplicación y se está probando con cotorras argentinas. También, para ambas aplicaciones, se complementó con un formulario para los guardianes, en el cual podrán evaluar la usabilidad de la aplicación.

A continuación se presenta en mayor detalle como se llevó a cabo la validación, junto con los resultados de esta misma.

5.1. *Frontend* para animales no-humanos

La validación del *frontend* para animales no-humanos se dividió en dos partes. Antes de llevar a cabo estas dos partes se tuvo que pasar por la revisión hecha por el prof. Jérémy Barbay (Sección 5.1.1), como se mencionó anteriormente. Luego de esto se puede proceder con las dos partes de la validación. La primera parte, la cual aún se está llevando a cabo, consiste en validar la aplicación con cotorras argentinas (Sección 5.1.2) y para la segunda parte, para la cual también está actualmente en proceso, se hizo un formulario para que los guardianes puedan evaluar la aplicación (Sección 5.1.3).

5.1.1. Revisiones con el profesor guía

Antes de probar la aplicación con las cotorras argentinas se revisó con el prof. Jérémy Barbay que esta fuera adecuada para estas mismas. El profesor guía fue el encargado de esto debido a que fue él quien propuso un protocolo de experimento de esta aplicación al comité de ética de la Universidad de Chile, el cual fue aceptado para las fechas enero 2023 hasta mayo 2024. En el protocolo aceptado se sugiere un entrenamiento con una duración de 1 a 6 meses y una evaluación y generación de datos del mismo rango de duración.

Las sesiones de ejercicio terminan en 10 minutos o cuando el sujeto decide parar voluntariamente y si el sujeto rechaza participar por 3 días consecutivos, las actividades se suspenden durante 3 días. Los sujetos son presentados a las actividades entre 1 y 5 veces al día, y son recompensados con retroalimentación oral o alimento. En caso de que el sujeto se niegue a participar por 3 meses consecutivos, el experimento se termina.

Considerando este protocolo de ética, el prof. Jérémy Barbay llevó a cabo esta revisión antes de pasar la aplicación a animales no-humanos. En ésta se notaron diferentes problemas en la aplicación, que impedían usarla con las cotorras argentinas. En primer lugar, se notó que utilizar la tecla "CTRL" para automatizar la modelación de un guardián causaba problemas, ya que combinada con algunas teclas funcionaba mal. Para solucionar esto se decidió cambiar "CTRL" por "SHIFT", lo cual no debería tener problemas con ninguna tecla, puesto que solo se activa la mayúscula combinando esta tecla con todas las letras. En segundo lugar, debido a que las cotorras argentinas utilizarían la aplicación en un celular o tableta electrónica, se notó que el uso de la tecla "SHIFT" combinada con la tarjeta no era factible para este caso. Es por esto que se agregó un botón llamado "Shift" que simula el mismo efecto que se había programado con la tecla, junto con un botón llamado "Caps Lock" que hace lo mismo que el anterior, pero bloqueando este efecto hasta que se presione nuevamente. Finalmente, se notó que los *logs* locales de la aplicación eran accesibles por las cotorras argentinas, ya que se encontraban en la parte inferior de la aplicación y no tenían ningún tipo de modo seguro, por lo que se optó por moverlos dentro de otra vista tal como se explicó en el diseño.

Durante esta misma revisión también se notaron otros detalles. Principalmente, se notó que no se guardaban algunos datos del estado de la aplicación, como el nombre del guardián y del sujeto. Además, se recomendó agregar el tono de voz y la velocidad de la voz a las configuraciones del audio, ya que estas antes no se tenían. La razón por la cual se recomendó agregar estas configuraciones es principalmente para personalizar la aplicación adecuadamente a cada usuario. Esto ayuda a que si un mismo guardián quisiera utilizar la aplicación paralelamente con más de un sujeto, podría configurar distintas voces sintetizadas para cada sujeto. Además, ayuda a que se pueda encontrar la velocidad y tono de voz que funciona mejor para cada sujeto.

Finalmente, se notaron algunos problemas básicos de estética, que influían en la utilidad de la aplicación. El primer problema con respecto a esto correspondió a los botones protegidos. Estos, aun cuando funcionaban bien, no funcionaban de manera perfecta, ya que no se mostraba por cuanto tiempo se estaba presionando el botón. Es por esto que se optó por cambiar los botones por los *Trainer Buttons* de *InCA Utils* [13], que, además de solucionar el problema, mejoraban la estética de la aplicación. El segundo problema correspondía a que el texto de la aplicación, tanto en los botones como en el encabezado, era seleccionable. Esto

era problemático, ya que los usuarios no humanos podrían seleccionar el texto de manera no intencional, evitando el funcionamiento de la aplicación. El último problema a solucionar, y luego del cual se decidió que la aplicación ya estaba lista para ser validada con sus usuarios, fue el encabezado. Dado que se quiere que el tablero de comunicación use la mayor parte del espacio, un encabezado grande no ayudaba mucho a la usabilidad. Es por esto que se cambió el encabezado original, que correspondía a una barra grande con el título de la aplicación, dos botones y la información del usuario, guardián y sujeto, por un simple título que no usara mucho espacio. El resto de los elementos, como los botones y la información, se movió a la parte inferior de la aplicación.

5.1.2. Validación con animales no-humanos

Para validar con este tipo de usuario, como se mencionó anteriormente, se hará uso de los *logs* que llegan a *Firebase*. Al tener tanto la cantidad de *clicks* sobre las tarjetas, como los *clicks* fuera de las tarjetas, se podrá hacer una comparación entre estos para ver que tan aleatorios son las presiones en la aplicación. Con esto se podrá evaluar si los colores, forma y posición de las tarjetas es el correcto para que los animales no-humanos utilicen la aplicación.

También, con los *logs* de las interacciones con las tarjetas se medirá cuantas veces se presiona cada una de las cuatro palabras que vienen por defecto; "no", "yes", "drink" y "eat", para un sujeto en particular. De esta forma se podrá medir si hay una distribución más o menos equitativa entre las palabras, lo cual podría implicar que el sujeto entiende de alguna forma el significado y no presiona simplemente siempre la misma palabra.

Dado que la validación aún esta en proceso, todavía no se tienen resultados de esta misma, pero al menos ya se observó que dos cotorras argentinas son capaces de presionar las tarjetas. A continuación se muestra una imagen donde se puede observar una de las dos cotorras argentinas utilizando la aplicación (ver Figura 5.1). Aun cuando todavía se está validando el nivel de la aplicación, ya se sabe que por lo menos es capaz de reconocer las tarjetas sobre el tablero.

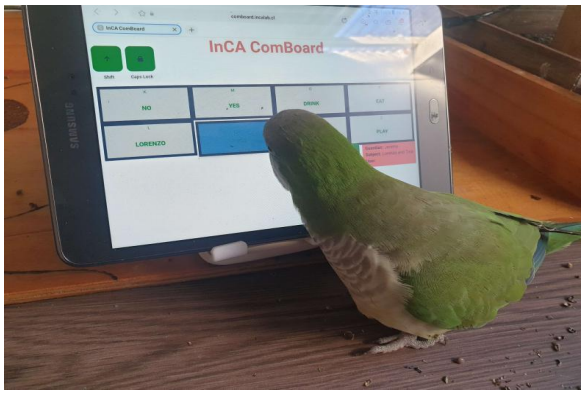


Figura 5.1: Cotorra argentina utilizando aplicación "InCA Comboard". Se puede observar que este mira hacia la pantalla y acaba de presionar una de las tarjetas sobre esta misma.

5.1.3. Validación con guardianes

Finalmente, para evaluar el resto de las funcionalidades de la aplicación, se desarrolló una encuesta para los guardianes de los animales no-humanos que utilizaron la aplicación. Esta encuesta se enfoca en evaluar las funcionalidades de la aplicación que están dirigidas específicamente a los guardianes, como todo lo que viene en las configuraciones de la aplicación. Se puede encontrar la encuesta en la aplicación junto con el resto de la información de contacto. Al igual que la sección anterior, todavía no se tienen resultados de esta encuesta, ya que la validación aún se está llevando a cabo.

1. ¿Qué tan intuitiva encontraste la interfaz de usuario de la aplicación?
   (a) Muy intuitiva
   (b) Intuitiva
   (c) Neutral
   (d) Poco intuitiva
   (e) Muy poco intuitiva
2. ¿Cómo calificarías la facilidad de agregar nuevas tarjetas con sonidos asociados?
   (a) Muy fácil
   (b) Fácil
   (c) Neutral
   (d) Difícil
   (e) Muy difícil
3. ¿Cómo calificarías la facilidad de modificar las configuraciones de audio?
   (a) Muy fácil
   (b) Fácil
   (c) Neutral
   (d) Difícil
   (e) Muy difícil
4. ¿Cómo calificarías la facilidad de mover tarjetas en el tablero?
   (a) Muy fácil
   (b) Fácil
   (c) Neutral
   (d) Difícil
   (e) Muy difícil
5. ¿Cómo calificarías la facilidad de eliminar tarjetas?
   (a) Muy fácil

(b) Fácil
(c) Neutral
(d) Difícil
(e) Muy difícil

6. ¿Cómo calificarías la facilidad de modificar la llave asociada a una tarjeta?
   (a) Muy fácil
   (b) Fácil
   (c) Neutral
   (d) Difícil
   (e) Muy difícil
7. ¿Cómo calificarías la facilidad de iniciar sesión con tu cuenta de *Google*?
   (a) Muy fácil
   (b) Fácil
   (c) Neutral
   (d) Difícil
   (e) Muy difícil
8. ¿Tienes algún comentario adicional o sugerencia para mejorar la aplicación?
9. ¿Qué aspectos te gustaron más de la aplicación?
10. ¿Qué aspectos crees que podrían mejorarse en la aplicación?

5.2. *Frontend* para investigadores

Dado que en este caso solo se tienen usuarios humanos, y no se considera ningún protocolo de ética, simplemente se llevó a cabo una encuesta para su validación. En esta encuesta, al igual que en la anterior, se evalúan las diferentes funcionalidades de la aplicación. La dirección web de esta misma se encuentra en el encabezado de la aplicación junto al resto de la información de contacto. Al igual que para la aplicación anterior, todavía se están recolectando respuestas para esta encuesta para poder hacer un análisis de los resultados.

1. ¿La tabla muestra los datos de forma clara y organizada?
   (a) Sí, de manera muy clara y organizada
   (b) Sí, en su mayoría clara y organizada
   (c) Neutral
   (d) No es muy clara y organizada
   (e) No es clara y organizada en absoluto

2. ¿Encuentras útil la función de ordenar la tabla por distintas variables (fecha, hora, nombre del sonido, usuario)?
   (a) Sí, es muy útil
   (b) Sí, es útil
   (c) Neutral
   (d) No es muy útil
   (e) No es útil en absoluto
3. ¿Encuentras útil la función de descargar la tabla como csv?
   (a) Sí, es muy útil
   (b) Sí, es útil
   (c) Neutral
   (d) No es muy útil
   (e) No es útil en absoluto
4. ¿Tienes algún comentario adicional o sugerencia para mejorar alguna de las aplicaciones?
5. ¿Qué aspectos te gustaron más de la aplicación?
6. ¿Qué aspectos crees que podrían mejorarse en la aplicación?

Capítulo 6

Conclusiones y Trabajo a Futuro

Para concluir, en este capítulo se presenta un resumen de los resultados logrados (Sección 6.1), una discusión de sus limitaciones (Sección 6.2) y de sus extensiones futuras (Sección 6.3).

6.1. Resultados Logrados

En términos generales, los resultados logrados de este trabajo de título fueron bastante positivos. Se logró llegar al software deseado, incluyendo todas las funcionalidades mínimas deseadas para una aplicación funcional. La implementación de este trabajo fue fluida, logrando superar los inconvenientes en el camino. Se logró, también, tener un protocolo de validación adecuado y se comenzó a llevar a cabo este mismo. Además, durante el desarrollo de la propuesta de este trabajo de título también se desarrolló y publicó un artículo en el área *Work in Progress* para la conferencia de ACI 2022 [4].

Los resultados del diseño, más detalladamente explicados en el Capítulo 3, fueron exitosos. Se logró diseñar tanto un *frontend* para los animales no humanos como para los investigadores, incluyendo diversas funcionalidades que ayudan con la usabilidad de las aplicaciones. Además, se logró conectar ambos *frontend* con una base de datos *Firebase* donde se registran los *logs* para luego obtener estos mismos y mostrarlos a los usuarios.

Con respecto a la implementación, también se puede hablar positivamente de los resultados. Esta se desarrolló sin mayores problemas, usando, en su mayoría, las herramientas ya seleccionadas en la propuesta del trabajo. Se logró aprender sobre estas herramientas de desarrollo, además de utilizarlas correctamente.

Definir un buen protocolo de validación fue lo más desafiante durante el trabajo, dado que esto nunca se había hecho antes y menos en un área tan específica como lo es *Animal Computer Interaction* (ACI). Aun así, se logró llevar a cabo esta parte correctamente, definiendo un protocolo claro y completo para la validación y mejorando la aplicación acorde a esto.

Considerando los resultados planteados anteriormente, cabe destacar que estos muestran avances notables respecto a los objetivos propuestos inicialmente. Primero, la mejora en la aplicación web de *communication board* permitió alcanzar una mayor adaptabilidad a diferentes animales, cumpliendo así con lo propuesto en el primer objetivo. Esta versión mejorada, tomando como base el trabajo previo del prof. Jérémy Barbay, ha incrementado significativamente su utilidad y flexibilidad.

En cuanto al registro automatizado de *logs*, se integró exitosamente un *backend* a la aplicación web, lo que satisface el segundo objetivo. Los *logs*, ahora almacenados en una base de datos *Firebase*, ofrecen una visión profunda de las interacciones, facilitando análisis y estudios futuros.

La interfaz desarrollada para la recopilación y análisis de *logs* cumple con los objetivos tres y cuatro, proporcionando herramientas para filtrar *logs*, así como descargar la información. Así, investigadores y usuarios interesados pueden entender mejor el alcance de la comunicación entre animales no humanos y humanos.

Finalmente, la visualización de patrones de comunicación, otro objetivo esencial, fue satisfecho mediante la creación de una interfaz intuitiva, en la cual se pueden observar las frecuencias de uso de palabras y otros patrones relevantes.

6.2. Discusión

A pesar de que el trabajo logrado durante el semestre fue positivo, a lo largo de su desarrollo se encontraron diversas limitaciones y debilidades que se tuvieron que ir superando.

En primer lugar, una de las debilidades principales fue la gestión del tiempo. El desarrollo de software implica una serie de tareas complejas y requiere una planificación cuidadosa para cumplir con los plazos establecidos. En este proyecto, hubo desafíos para establecer una programación eficiente, lo que resultó en ciertas dificultades para cumplir con los hitos y objetivos establecidos inicialmente. Además, dado que el manejo de los tiempos en este trabajo dependía más que nada de la estudiante, fue difícil encontrar estructura.

Otra debilidad importante fue el *deploy* de la aplicación en el servidor. Esta etapa no suele ser muy complicada, pero dado que nunca se había hecho con una aplicación de esta complejidad, no fue tan directo aprender a hacerlo. Esto generó retrasos en la puesta en marcha de la aplicación, lo cual retraso un poco la validación y también continuar con el desarrollo de la aplicación.

Además de esto, también se fueron encontrando limitaciones en el desarrollo de software a lo largo del trabajo. Aun cuando ya se había desarrollado antes, nunca se había usado exactamente las mismas herramientas utilizadas para este proyecto. Tampoco, nunca se había desarrollado una aplicación de esta complejidad completamente desde cero, por lo que esto igual implicó un desafío.

Finalmente, otro desafío significativo fue la validación de la aplicación con usuarios. Aunque el desarrollo del software puede ser riguroso, es esencial someterlo a pruebas y recibir

comentarios de los usuarios reales para evaluar su eficacia y usabilidad. En este caso, la falta de experiencia previa en definir un protocolo de validación para aplicaciones web implicó cierta curva de aprendizaje y dificultades adicionales, enlenteciendo el proceso de validación. Además, el hecho de que se tratase de una aplicación enfocada no solo en humanos, sino que también en animales no humanos, agregaba una dificultad a la validación.

A pesar de estas debilidades, es importante destacar que el trabajo se llevó a cabo correctamente dentro del tiempo designado. A través del esfuerzo y la dedicación, se lograron superar los desafíos mencionados anteriormente y se obtuvieron resultados satisfactorios en términos de funcionalidad y calidad del producto final.

6.3. Perspectiva sobre Trabajos Futuros

Aun cuando se cree que se logró un trabajo final positivo, todavía quedan muchas posibles funcionalidades que se podrían implementar para ayudar a los usuarios. Todas estas funcionalidades quedan propuestas como trabajo futuro en las siguientes dos listas. En primer lugar, se propone una lista para el *frontend* de los sujetos y luego una lista para el *frontend* de los investigadores.

6.3.1. *Frontend* para los sujetos

1. **Agregar imágenes a las tarjetas:**
   La aplicación *CommBoards App* [17] usa imagen en sus tarjetas. Dado que esta aplicación ya fue probada con aves [6], sabemos que con imágenes funciona correctamente, por lo que sería útil agregar imágenes a cada una de las tarjetas. No se necesita mucho tiempo ni conocimiento extra para implementar esto, ya que simplemente se debe agregar la opción de subir una imagen dentro del formulario de tarjetas. Sin embargo, una complicación podría ser tener que guardar estas imágenes en el *backend* al guardar el estado de la aplicación para una cuenta en particular. Considerando esto, la implementación podría tomar un par de semanas.

2. **Regrabar un sonido**:
   La opción de regrabar un sonido es importante, ya que puede ocurrir el caso que, aun cuando esté bien grabado el sonido, en el momento de utilizar la aplicación se observa que el sujeto nunca utiliza la tarjeta asociada, lo cual podría deberse a la grabación. Para implementar esto, con lo que ya está implementado, no se necesita un nivel muy alto, ya que se puede reutilizar código ya implementado. De esta forma, con un par de días sería suficiente para llevarlo a cabo.

3. **Agregar la opción de usar la aplicación sin enviar los *logs* a un servidor externo, o cambiar el servidor externo al cual se envían:**
   Por temas de privacidad, podría ser interesante agregar esta opción a la aplicación. Esta funcionalidad podría tener algunas complicaciones más que las otras, ya que se necesita conocer un poco de *Firebase*, o del servidor al que se quieran enviar los *logs*, para implementarlo. La opción de usar la aplicación sin enviar los *logs* se podría implementar

en un par de días. Sin embargo, la opción de cambiar el servidor al que se envían los *logs* se podría complicar si se quisiera dejar la opción de utilizar cualquier tipo de servidor, lo cual podría tomar una semana.

6.3.2. *Frontend* para los investigadores

1. **Agregar inicio de sesión:**
   Tener una cuenta, tal como en el *frontend* para animales no-humanos, ayudaría a poder lograr un análisis más personalizado para cada investigador. Esto se puede hacer de la misma forma que se hizo el inicio de sesión en la otra aplicación, por lo que se podría realizar en un día.

2. **Agregar una columna de intencionalidad de la interacción que se puede modificar:**
   Sería útil para los investigadores tener una columna donde se pueda cambiar la intencionalidad de la interacción correspondiente a cada *log*, como por ejemplo si esta fue modelada por el guardián o intencionalmente por el sujeto. Sin embargo, para esto se requiere tener un inicio de sesión, dado que un usuario no debería poder modificar la intencionalidad de cualquier *log*, sino que solo los correspondientes a su cuenta. La implementación de esto es sencilla y, si se tuviera el inicio de sesión implementado, no tomaría más que un par de días.

3. **Permitir filtro con una expresión regular:**
   Para que el usuario pueda hacer búsquedas más interesantes con la aplicación, sería interesante que en los buscadores se pueda filtrar con una expresión regular. Esto es sencillo de implementar y se podría hacer en medio día.

4. **Agregar un texto de ayuda para cada columna:**
   Dado que los usuarios no necesariamente saben a qué se refiere cada columna se debería agregar un pequeño texto de ayuda para cada una que explique qué es cada una de estas. Esto, nuevamente, es muy sencillo de implementar y se puede lograr en un día.

Bibliografía

[1] ACI. Aci conference. https://www.aciconf.org/. Last accessed on [2022-09-18 Sun].

[2] American Speech-Language-Hearing Association. Aac. https://www.asha.org/public/speech/disorders/aac/, 2023. Last accessed on [2022-09-18 Sun].

[3] Jeremy Barbay. InCA-ComBoard. https://barbay.cl/ComBoard/, 2022. Last accessed on [2022-09-18 Sun].

[4] Jérémy Barbay, Camila Labarca-Rosenbluth, and Brandon Peña Haipas. A loggable aid to speech: A research proposal. In *Proceedings of the Ninth International Conference on Animal-Computer Interaction*, ACI '22, New York, NY, USA, 2023. Association for Computing Machinery.

[5] Jérémy Barbay. InCA Lab. https://incalab.cl/, 2022. Last accessed on [2022-09-18 Sun].

[6] Jennifer Cunha and Carlie Rhoads. Use of a tablet-based communication board and subsequent choice and behavioral correspondences in a goffin's cockatoo (cacatua goffiana). In *Proceedings of the Seventh International Conference on Animal-Computer Interaction*. ACM, November 2020.

[7] Alexis Devine. What about bunny. https://www.youtube.com/@whataboutbunny, 2022. Last accessed on [2022-09-18 Sun].

[8] FluentPet. www.facebook.com/fluentpet. https://www.facebook.com/FluentPet, 2022. Last accessed on [2022-09-18 Sun].

[9] FluentPet. www.instagram.com/fluentpet/. https://www.instagram.com/fluentpet/, 2022. Last accessed on [2022-09-18 Sun].

[10] HowTheyCanTalk. how.theycantalk.org. https://how.theycantalk.org, 2022. Last accessed on [2022-09-18 Sun].

[11] Christina Hunger. *How Stella Learned to Talk: The Groundbreaking Story of the World's First Talking Dog*. William Morrow Paperbacks., 2018.

[12] Christina Hunger. hungerforwords.com. Website https://www.hungerforwords.com/, 2022. Last accessed on [2022-09-16 Fri].

[13] Fabian Jaña. InCA Utils. https://shockerqt.gitlab.io/inca-utils/, 2022. Last accessed on [2022-09-18 Sun].

[14] Fluent Pet. fluent.pet. https://fluent.pet/, 2022. Last accessed on [2022-09-16 Fri].

[15] Alexandre Pongrácz Rossi and César Ades. A dog at the keyboard: using arbitrary signs to communicate requests. *Animal Cognition*, 11(2):329–338, November 2007.

[16] Cristobal Sepulveda. What is more. https://whatismore.incalab.cl/, 2022. Last accessed on [2022-09-18 Sun].

[17] ShmoontzApps. Commboards app. https://www.shmoontzapps.com/, 2020. Last accessed on [2022-09-18 Sun].

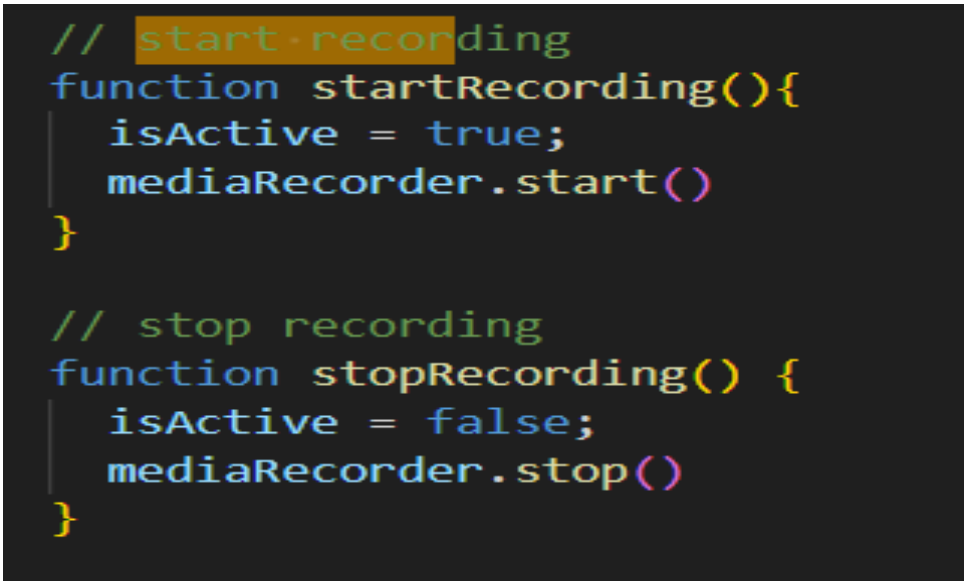
[18] Clint Zeagler, Scott Gilliland, Larry Freil, Thad Starner, and Melody Jackson. Going to the dogs: Towards an interactive touchscreen interface for working dogs. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, page 497–507, New York, NY, USA, 2014. Association for Computing Machinery.

Anexo

Fragmentos de código fuente

```
const handle_keydown = e => {

    if (e.key === 'Escape') {
        close();
        return;
    }

    if (e.key === 'Tab') {
        const nodes = modal.querySelectorAll('*');
        const tabbable = Array.from(nodes).filter(n => n.tabIndex >= 0);

        let index = tabbable.indexOf(document.activeElement);
        if (index === -1 && e.shiftKey) index = 0;

        index += tabbable.length + (e.shiftKey ? -1 : 1);
        index %= tabbable.length;

        tabbable[index].focus();
        e.preventDefault();
    }
};
```

Figura 6.1: Función para manejar eventos de presiones de teclas cuando se tiene un modal abierto.



```
// start recording
function startRecording(){
    isActive = true;
    mediaRecorder.start()
}

// stop recording
function stopRecording() {
    isActive = false;
    mediaRecorder.stop()
}
```

Figura 6.2: Función para comenzar y detener grabación de sonido.

```
{#if !sintesizedVoice}
  <section>
    <button on:click={startRecording}>Record</button>
    <button on:click={stopRecording}>Stop</button>
    <div class="player">
      <div id="info" class="info {isActive ? 'active' : ''}">
        <span class="artist">Recording</span>

      </div>
      <div id="control-panel" class="control-panel {isActive ? 'active' : ''}">
        <div class="album-art" />
      </div>
    </div>
  </section>
  <div>
    <audio controls />
  </div>
{/if}
```

Figura 6.3: Sección para grabar sonido para una tarjeta.

```
onAuthStateChanged(auth, (user) => {
    firebaseUser = user;
});

async function login() {
    const provider = new GoogleAuthProvider();
    try {
        const result = await signInWithPopup(auth, provider);
        const user = result.user;
        console.log(user);
        const userRef = doc(db, "users", user.uid);
        const document = await getDoc(userRef);
        if (!document.exists()) {
            await setDoc(userRef, {
                name: user.displayName,
                role: "usuario",
                email: user.email,
            });
        }
    } catch (error) {
        console.log("Error signing in:", error);
    }
}

function logout() {
    signOut(auth);
    firebaseUser = null;
    console.log(firebaseUser);
}
```

Figura 6.4: Funciones para autenticación de usuario.

```
function downloadCsv() {
    const rows = filteredData
        .map(
            (data) =>
                `${data.date},${data.time},${data.guardian},${data.subject},${data.sound}`
        )
        .join("\n");
    const csv = `Date,Time,Guardian,Subject,User,Sound\n${rows}`;
    const blob = new Blob([csv], { type: "text/csv" });
    const url = URL.createObjectURL(blob);
    const link = document.createElement("a");
    link.href = url;
    link.download = "interactions.csv";
    link.click();
}
```

Figura 6.5: Función para descargar *logs* en formato CSV.