

Informe Descriptivo



LA INNOVACIÓN DE LA AGROINDUSTRIA

Abril Contreras - Martin Alejandro Cabrera - Lara Sofia Diaz Steinbrecher
Santino Ferrante - Mateo Kearney - Juan Ignacio Torres

HORUS

ÍNDICE

INTRODUCCIÓN	OBJETIVO	00
	ALCANCE	
	BENEFICIOS	
FRONTEND	INICIO	01
	ROBOTS	
	DATA	
	GALERÍA	
	AJUSTES	
BACKEND	CONEXIÓN USB	02
	BASE DE DATOS	
	ESQUEMAS	
	API	
ELECTRON	SETUP	03
		04
		01

INTEGRANTES

CABRERA, Martin Alejandro Cabrera
DNI 47.260.03 - Tel.: 11 6141 8080

DIAZ, Lara Sofia Steinbrecher
DNI 46.909.041 - Tel.: 11 3700 7432

KEARNEY, Mateo
DNI 47.307.247 - Tel.: 11 3786 4168

TORRES, Juan Ignacio
DNI 47.144.499 - Tel.: 11 6596 9351



ESFUERZO DEL PROYECTO

12 horas semanales
(312 horas de trabajo total)

FECHA DE INICIO

17 de Marzo de 2024

DURACIÓN

26 Semanas

DOCENTES INVOLUCRADOS

BIANCO, Carlos
CARLASSARA, Fabrizio
MEDINA, Sergio
PALMIERI, Diego

PÁGINA WEB

<https://horus-pagina-web.vercel.app>

INSTAGRAM

https://www.instagram.com/agro_horus/

LINKEDIN

<https://www.linkedin.com/company/agro-horus/about/>

INTRODUCCIÓN

OBJETIVO

El objetivo del proyecto general es brindar una **herramienta** que permita a los productores de campo en la Argentina **combatir** de forma **más efectiva** las **plagas** que afectan las cosechas, centrandonos inicialmente en la soja. Para esto, buscamos como proyecto diseñar un sistema de reconocimiento artificial e identificación a través del análisis de la cosecha. Esta herramienta permitirá **reconocer** las **4 plagas artrópodos principales** que pueden estar afectando a la planta: las orugas defoliadoras, las chinches, las arañuelas y los trips; y dar un informe detallado al usuario sobre el tipo y donde se encuentra utilizando una aplicación conectada por radiofrecuencia. La implementación de una plataforma móvil robótica permite una eficacia y velocidad mayor en una menor cantidad de tiempo en la que un ser humano podría realizar la misma tarea.

El sistema de **software** propuesta busca permitir al usuario interactuar con el sistema para otorgarle el mayor manejo posible de uno o más robots y de la manera más eficaz, de modo que pueda tanto **visualizar** e interactuar sobre la **información** que los mismos recopilan para su **almacenamiento** y/o consulta así como controlar las **áreas a reconocer** que quiere priorizar.

ALCANCE

Nuestro proyecto está **destinado** a los **productores del campo en la Argentina**, principalmente los **productores de soja**. Pero en un futuro, nuestro proyecto podrá utilizarse en cualquier país, ya que la IA se adapta a cualquier plaga artrópoda que afecte al cultivo del terreno. La IA cuenta con la posibilidad de aprender, entonces solo requeriría una etapa de aprendizaje corta para trabajar en cualquier terreno, ya sea un campo en el que se cultive maíz, soja, trigo, tomates, etc.

Por más que nuestro proyecto esté **orientado** a la **agroindustria**, Horus también se puede utilizar para monitoreo rutinario de cualquier tipo.

BENEFICIOS

Con HORUS buscamos que el sector agrícola pueda reducir la necesidad de labor humano, así como aumentar la tasa de monitoreo tanto en cantidad de días como en horas. Un pequeño grupo de personas podrá trabajar con las imágenes recopiladas que la IA del robot les envíe, evitándose así el trabajo físico a intemperie, como también reduciendo la necesidad de un mayor número de personal capacitado. Además,

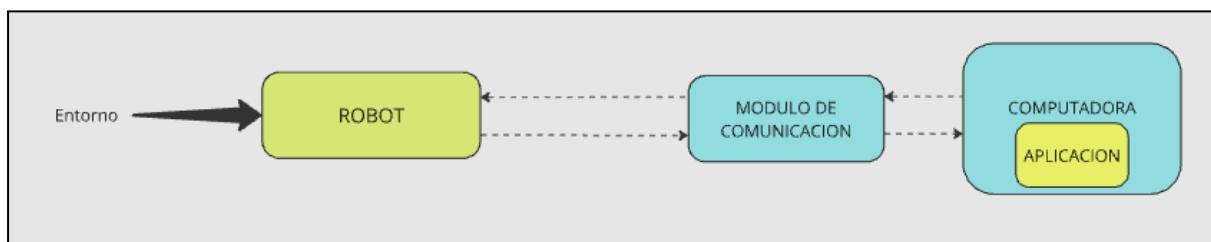
1. El proyecto reduce la carga laboral necesaria para el reconocimiento de plagas en el campo.

2. La reducción de pérdidas en la cosecha incrementará la cantidad producida y por lo tanto mayor oferta de la cosecha en el mercado.
3. El uso más eficiente y más bajo de insecticidas tendrá un impacto beneficioso en la salud alimentaria de los consumidores.
4. El personal puede pasar menos tiempo bajo el sol, y monitorear a través del robot, reduciendo la exposición y minimizando la probabilidad de enfermedades.

GENERALIDADES

DESCRIPCIÓN

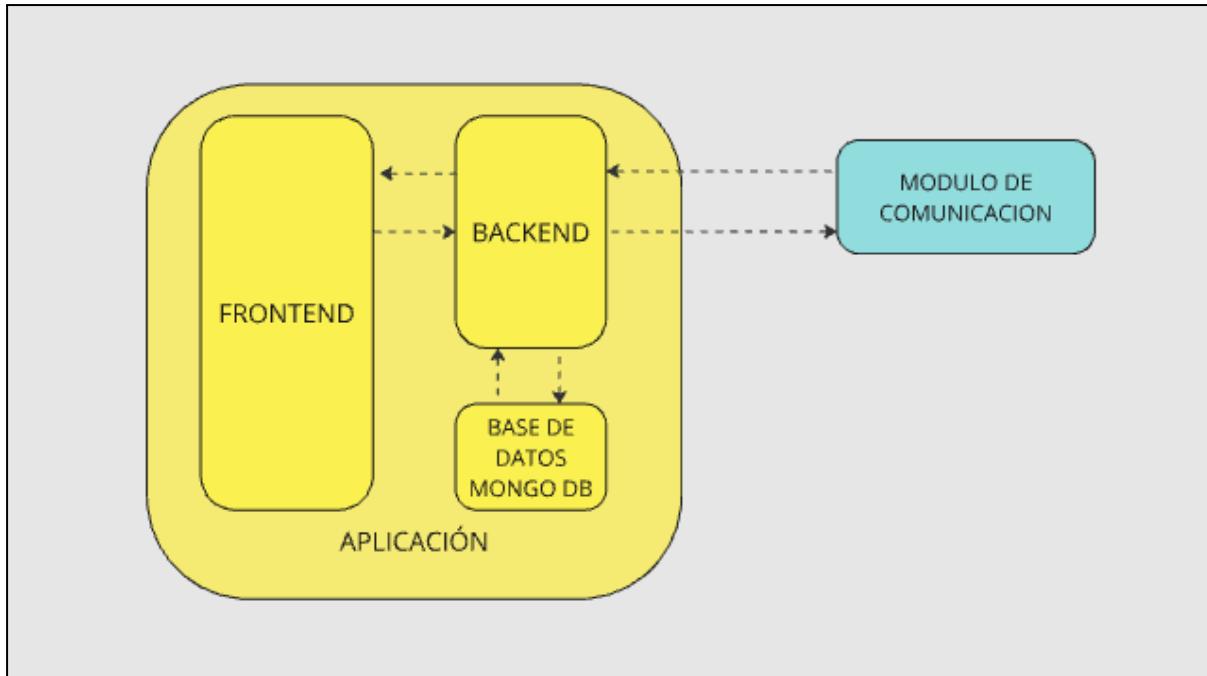
El sistema del que forma parte el software de HORUS está formado por uno o más robots que se trasladan por el campo detectando plagas mediante inteligencia artificial. La aplicación se encarga de permitir al usuario interactuar con el sistema del robot, permitiéndole visualizar y almacenar la información que este detecta y modificar su trayectoria.



ESTRUCTURA

El sistema puede dividirse en tres secciones principales: la interfaz o frontend, el servidor o backend y la base de datos.

- FRONTEND: es la parte del sistema con la que el usuario interactúa
- BACKEND: es la parte del sistema que comunica el frontend con la base de datos y el robot y organiza la información de la que deba ser intermediario.
- BASE DE DATOS: es la parte del sistema que almacena las instancias de reconocimiento que detecta el robot y las configuraciones de la interfaz.



BACKEND

El servidor de la aplicación se encarga de intermediar entre la base de datos, el sistema del robot y la interfaz de usuario. El lenguaje elegido para la programación del backend fue javascript. Utiliza la biblioteca y entorno Node.js para que sea posible usar javascript fuera del buscador y por lo tanto usarlo para crear un servidor.

CONEXIÓN USB

El robot y la aplicación se comunican a través de módulos radiofrecuencia, y debido a que la computadora no puede acceder a la información electrónica de los mismos, utilizamos un microcontrolador Raspberry Pi Pico como intermediario, el cual se conecta a la computadora por medio de cable USB. La aplicación entonces utiliza la librería de serialport para poder acceder al puerto USB. La comunicación por puerto serie se realiza con protocolo UART.}

La información que transmite el robot a la aplicación incluye:

- Porcentaje de batería
- Tiempo restante de batería
- Instancias de detección (dia, hora, ID del robot, tipo de plaga, ID de imagen relacionada, probabilidad, coordenadas)
- Coordenadas GPS

La información que transmite la aplicación al robot incluye:

- ID (al conectarse)
- Las coordenadas del punto de partida que debe seguir
- Las coordenadas del área que debe recorrer el robot

BASE DE DATOS

La base de datos se encarga de guardar información del robot, como las instancias de detección, e información del frontend, sobre todo marcadores y configuraciones del mapa. Esto permite mantener un historial de las plagas detectadas, y es donde el servidor busca información cada vez que el frontend le solicita información. En el caso del mapa, permite que los datos que el usuario ingresa a la aplicación, como ubicación de trampas de feromonas o el área a recorrer por el robot, puedan conservarse entre sesiones, incluso cuando haya cerrado la aplicación.

Debido a que los datos que transmite el robot son instancias de detección, no tienen relación entre sí, y tampoco la tienen con los datos guardados por el mapa. Es así que la base de datos elegida para este sistema ha sido MongoDB, una base de datos no relacional que guarda los datos a partir de objetos en formato .json. Ya que la aplicación es de uso offline, el sistema requiere que el usuario instale dicha base de datos para que el servidor pueda conectarse y guardar datos correctamente.

Para la gestión en el servidor de la base de datos utilizamos la librería de mongoose, que facilita el uso de la base de datos, permitiendo, entre otras cosas, utilizar objetos de javascript en vez de JSON para crear las entradas de la colección, el uso de esquemas y una gran personalización para queries.

La base de datos está formada por dos colecciones con sus respectivos esquemas, “robotdata” y “mapdata”. La primera guarda las instancias de detección y la segunda las configuraciones del mapa.

ESQUEMAS

Mongoose utiliza esquemas cada vez que tiene que crear una nueva entrada en la colección para validar el tipo de dato de cada key del objeto. Funcionan como plantillas, en las que se especifican las keys que se quiere tener en las entradas creadas con dicha plantilla. Cuando se utiliza una entrada para crear una entrada a una de las colecciones, el servidor revisa las keys que coinciden en el objeto que se le pasa a la función con las keys de la plantilla, y crea una nueva entrada en la colección en la que a cada key que coincide se le asigna el valor correspondiente del objeto.

API

Para comunicarse la información con el frontend, el sistema utiliza su propia API, creada con ayuda de la framework de Express.js.

Cuando el servidor requiere información de la base de datos, como cuando se montan los componentes de la ruta de Inicio o Data, envía una request por medio de método GET a una de las rutas del servidor y el servidor le responde con la información correspondiente a la ruta a la que accedió (en el ejemplo mencionado sería todos las configuraciones del mapa y todas las detecciones). Esto es útil cuando la información que se requiere al acceder a la ruta es la misma.

En otros casos, el usuario provee datos por medio de método POST, cómo son sus preferencias de filtros cuando el usuario selecciona el filtro en la ruta de data, y el servidor realiza una query a la base de datos con los datos seleccionados por el usuario en el filtro y el selector de orden de la interfaz y responderá con los logs que contengan los datos por los que se filtró.

En el caso de las configuraciones del mapa, el usuario accede a otra ruta por medio de método POST y cuyo cuerpo lleva la información en formato de objeto para que se pueda crear con el esquema la entrada correspondiente a la configuración.

DEPENDENCIAS

Dependencias a destacar:

- serialport: permite utilizar los puertos serie de la computadora, en este caso para comunicarse con el robot a través del módulo de comunicación.
- @serialport/parser-readline: permite leer la información del puerto USB.
- mongoose: facilita el uso de la base de datos de MongoDB y permite usar esquemas, queries y validación de tipo de dato.
- express: utilizado para manejar las requests de la api del servidor
- cors: es necesario para realizar requests al frontend porque los buscadores bloquean las requests.
- concurrently: permite realizar más de un comando de npm a la vez (para el script de backend y el de frontend).

FRONTEND

La interfaz de la aplicación es la forma que el usuario tiene para comunicarse con el sistema. Se encuentra programada en html, css y javascript mediante la framework de Vue.js.

El archivo principal contiene un header, que se mantiene en todas las rutas de la aplicación y le permite navegar entre las diferentes rutas de la misma:

- Inicio
- Robots
- Data
- Galería
- Ajustes

Y también contiene un componente llamado RouterView que proviene de Vue.js. Este componente carga las diferentes vistas o views dependiendo de la ruta designada. Las diferentes rutas se encuentran designadas por un archivo javascript (index.js en router) y en el nav del header cada item de la lista tiene vinculada una dirección de la ruta asignada por el archivo de router.

Algunas rutas, como Inicio, Robots y Data, realizan requests al servidor para obtener información de la base de datos. Esto lo hacen a partir de la api de fetch, tanto para las requests de GET como de POST.

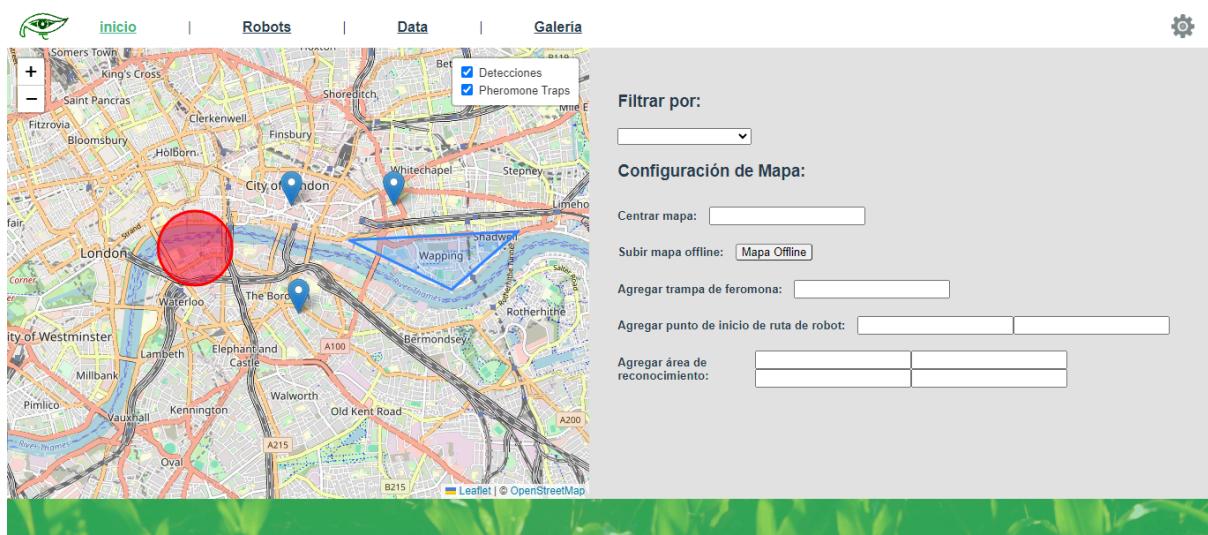
INICIO

El propósito de la ruta de inicio es proveer al usuario de la información general y más importante del sistema. Está formada por dos partes, la primera es el mapa y la segunda es la sección de control.

Para el mapa utilizamos la librería de leaflet.js que permite crear mapas a partir de los datos proveídos por un proveedor de datos geográficos (tiles), en nuestro caso OpenStreetMap. En un principio, el mapa trata de buscar los datos al proveedor de manera online, pero si esto falla, carga los datos que hayan sido guardados mediante el importe de archivos offline de la sección de control, si es que fueron cargados. El mapa se carga dentro de un div, y la librería permite que se carguen las imágenes de las tiles de manera dinámica, permitiendo zoom y movimiento en el mapa. Una vez montado el mapa, realiza una request a la base de datos para cargar los datos preexistentes de configuración del mapa. Esto incluye el eje central del mapa, marcadores de detecciones, marcadores de trampas de feromonas, marcadores de puntos de partida que tome cada robot y polígonos relacionados al área a recorrer por el robot. Cada uno de los datos mencionados se cargan en una capa diferente (el mapa se encuentra en una capa base) de modo que puedan ser desactivados en el controlador de capas en la parte superior derecha del mapa.

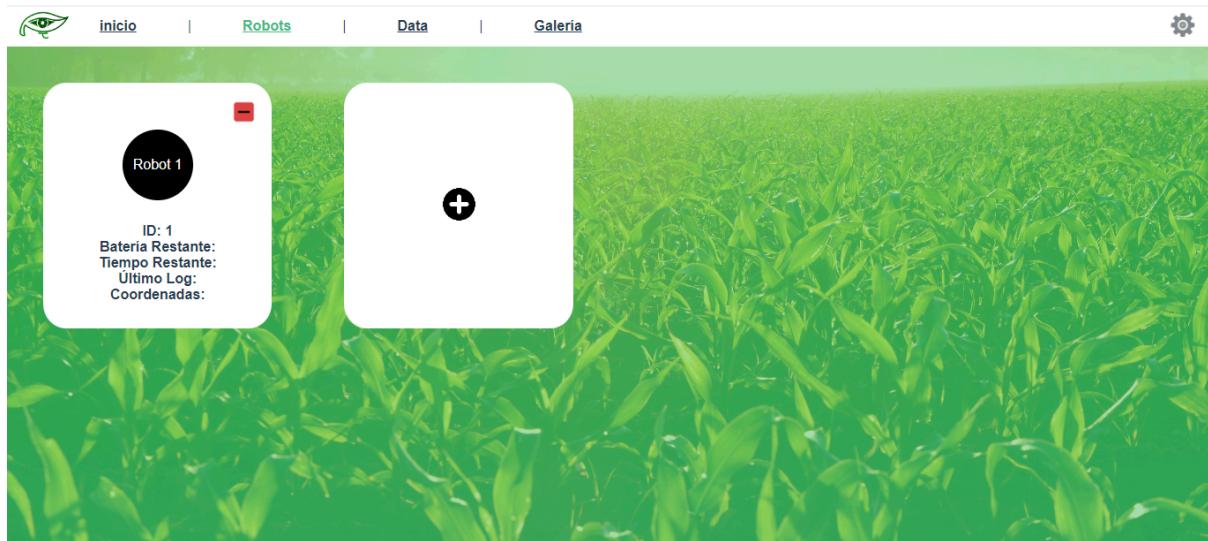
Con respecto a la sección de control, tiene diferentes controles cuya función se detalla a continuación:

- Permite centrar el eje del mapa - Esto tiene como objetivo centrar el mapa a las coordenadas del usuario.
- Permite importar un archivo offline - Una vez montado el mapa, comprueba si puede conectar con los servicios del proveedor de servicios de mapas y en su defecto, utiliza el archivo importado por este input
- Agregar trampas de feromonas al mapa
- Agregar un área de detección para un robot
- Agregar un punto de partida para un robot
- Filtro: permite filtrar las detecciones que se cargan en el mapa por cualquiera de sus keys, incluido por día, por hora, por ID de robot, por ID de imagen, por coordenadas, por tipo de plaga, por probabilidad o por trampa de feromona.



ROBOTS

La ruta de Robots tiene como objetivo dar a conocer el estado del o los robots conectados al servidor. De manera predeterminada no hay robots, y una vez que se agrega uno, se comprueba la conexión con el mismo y se carga un componente si es positiva.



DATA

La ruta de Data tiene como objetivo la visualización de los reconocimientos del robot de manera completa y organizada. Está formado por un filtro, un selector de orden y por el display de información.

- Filtro (DataFilter): la sección de filtro se comporta de la misma manera que el filtro en la ruta de inicio. Se puede elegir entre las diferentes keys (ya sea algunas o todas):
 - Dia
 - Hora
 - ID de robot
 - ID de imagen
 - Coordenadas
 - Tipo de plaga
 - Probabilidad
 - Trampa de feromonas.

Una vez se elige un tipo de key en el selector, se carga la sección correspondiente y permite elegir el dato por el que filtrar. Se pueden cargar todas las keys del selector o ninguna, y cada una puede eliminarse con el botón adjunto.

Cada vez que se elige asigna un valor a alguna de estas keys, se realiza una request POST via api fetch al servidor con los datos en el filtro.

Filtrar por:

Order by

Data Log 1:

- Date: 09-10-2024
- Time: 10:53
- Image ID: 1
- Robot ID: 1
- Plague Type: oruga
- Phermone Trap: true
- Probability: 100
- Location: [0, 1]

Data Log 2:

- Date: 10-10-2024
- Time: 10:53

Todos los filtros seleccionados:

Filtrar por:

Order by

Data Log:

- Date: 10-10-2024
- Time: 10:23
- Image ID: 1
- Robot ID: 1
- Plague Type: oruga
- Phermone Trap: true
- Probability: 100
- Location: [0, 1]

- Selector de orden (OrderFilter): el selector de orden permite elegir si la información se presenta de manera ascendente o descendente. La key que se elige en el selector es por el que se ordena. El orden es descendente de manera predeterminada. Estos valores se incluyen junto a los datos del filtro cuando se efectúa la POST request al servidor.
- Display de información (DataLogs): el display carga un componente por cada log que se encuentre en la variable de listLogs, en el que se le asigna el valor a cada key correspondiente de acuerdo a sus valores en el objeto guardado. Cuando se monta los componentes de la ruta, se realiza una request vacía al servidor, lo que devuelve todos los documentos de las instancias de detección de la base de datos (robotdata).

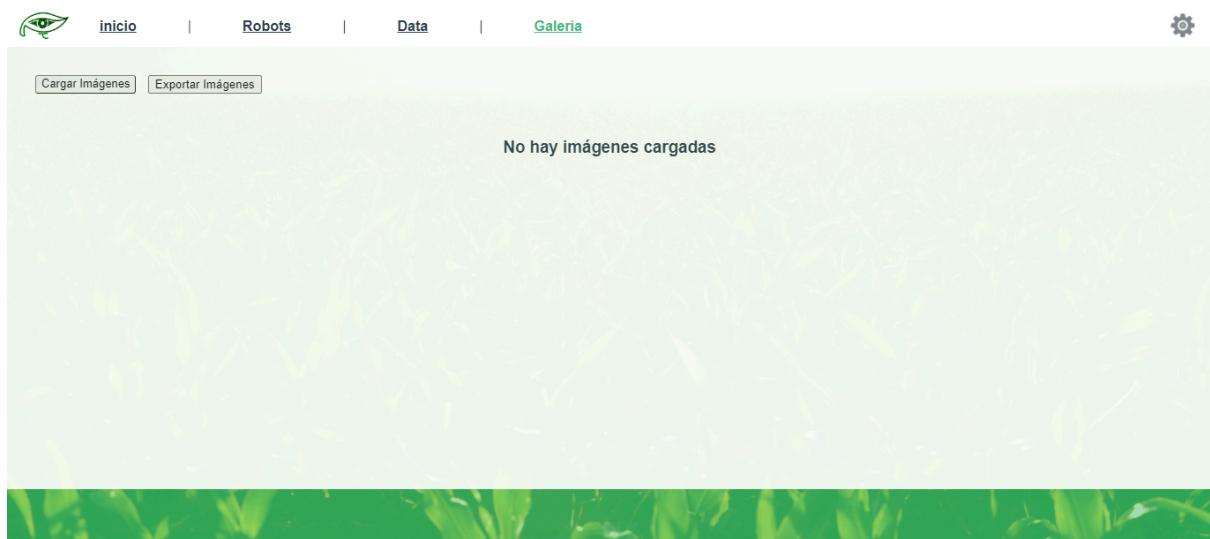
GALERÍA

El propósito de la galería es la visualización y almacenamiento de las imágenes captadas por el robot, y que el mismo almacena en su pendrive. Dicho pendrive puede ser insertado en el puerto de la computadora e importado a la galería.

Debido a que la base de datos de MongoDB tiene limitaciones en cuanto a la cantidad de almacenamiento que puede ocupar cada documento y que en general no está optimizada para almacenar archivos grandes como imágenes, se optó por la base de datos local IndexedDB, que permite almacenar incluso archivos más grandes del lado del cliente (frontend en nuestro caso). Cuando se suben las imágenes desde el pendrive, se convierten las mismas a base64 y se guardan en la colección de ‘image-store’.

Un botón para exportar las imágenes se encuentra junto al de subir imágenes. Este utiliza la librería de JSZip para comprimir todas las imágenes en un archivo zip y que el usuario lo pueda guardar.

Al hacer click en las imágenes en la galería, se sobrepone un overlay con la imagen de tamaño más grande, de modo que al usuario le sea más sencillo enfocar en diferentes aspectos de la imagen.



AJUSTES

El botón de ajustes carga un overlay con diferentes opciones.

- Ayuda: descarga el manual de usuario de la aplicación
- Contactos: contiene un link a la página web de Horus con los contactos del grupo
- Exportar logs: exporta como archivo zip las instancias de detección de toda la base de datos mediante la librería JSZip.



DEPENDENCIAS

- vue-router: permite crear el archivo para las rutas y usar el componente de RouterView para mostrar las views.
- idb: es la librería de IndexedDB que permite guardar información, en este caso imágenes, en el buscador.
- JSZip: es la librería que permite crear archivos comprimidos a partir de la información de logs o de las imágenes.
- leaflet: es la librería que permite crear mapas, capas y marcadores para la ruta de inicio de la aplicación

ELECTRON SETUP

Electron.js es una framework que permite la ejecución de la aplicación como aplicación de escritorio. Un archivo de javascript (main.js en electron) indica la creación de una ventana de buscador y controles de ventana cuando se ejecuta. Los paquetes de ejecución se encuentran en el directorio raíz del proyecto.