UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Graduado en Ingeniería de la Salud

Aplicación para gestión de datos de flora
App for managing flora data

Realizado por

Juan Sánchez Rodríguez

Tutorizado por

Ismael Navas Delgado

Cristobal Barba González

Departamento

Lenguajes y ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Junio 2023

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

GRADUADO EN INGENIERÍA DE LA SALUD

Aplicación para gestión de gatos de flora

App for managing flora data

Realizado por
Juan Sánchez Rodríguez

Tutorizado por
Ismael Navas Delgado
Cristóbal Barba González

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2023

Fecha defensa: Junio de 2023

Abstract

Data collection is a crucial task. However, manually gathering data can lead to spelling errors that affect the accuracy of the results. To address this issue, NLP technologies have been utilized in Python for preprocessing species names. Subsequently, a web application for data collection has been developed with offline functionality, incorporating autocomplete tools to prevent spelling errors and ensure error-free data collection. Additionally, the application includes an automatic geolocation system that enables precise annotation of the location of plant species. This web application provides an efficient, accurate, and accessible solution for data collection in biological research and other similar fields.

Keywords: **Aplication, Flora, Offline, Unification, NLP**

Resumen

La recolección de datos es una tarea crucial. Sin embargo, la recopilación manual de los datos puede resultar en errores de escritura que afectan la precisión de los resultados. Para abordar este problema, se han utilizado tecnologías NLP en Python para el preprocesado de los nombres de las especies, y posteriormente se ha desarrollado una aplicación web de recogida de datos con función *offline* que utiliza herramientas de autocompletado para prevenir errores de escritura y garantizar una recolección sin fallos. Además, la aplicación incluye un sistema de geolocalización automática que permite la anotación precisa de la ubicación de las especies vegetales. Esta aplicación web es una solución eficiente, precisa y accesible para la recolección de datos biológicos en la investigación y otros campos similares.

Palabras clave: **Aplicación, Flora, Offline, Unificación, PLN**

Índice

1 Introducción

1.1. Contexto y problema

Este proyecto de fin de grado se desarrolla en el marco del proyecto *LifeWatch ERIC Environmental and Biodiversity Climate Change Lab (EnBiC2-Lab)*, cuyo objetivo principal es investigar la biodiversidad y promover la gestión sostenible de los ecosistemas. El propósito de este proyecto es destacar la investigación llevada a cabo por la *Universidad de Málaga (UMA)* dentro de la línea *LifeWatch ERIC*, y compartir sus resultados a través de laboratorios virtuales que faciliten la colaboración con otros grupos de investigación y desarrollo en toda Europa.

El resultado esperado consiste en la digitalización y automatización de las técnicas empleadas por los grupos científicos, lo que contribuirá a su transformación digital, así como a la difusión y acceso de los resultados de investigación a la comunidad científica.

La gestión de datos [1] es un aspecto crítico en cualquier proyecto, especialmente en proyectos de investigación. En concreto, el departamento de *Botánica y Fisiología Vegetal de la Universidad de Málaga* realiza catálogos de especies vegetales [2] [3] [4] que se obtenían mediante fichas en papel, lo que conllevaban numerosos errores ortográficos. Más tarde, estas fichas fueron actualizadas a hojas en *Excel* pero los errores humanos en la transcripción persistieron. Esto ha convertido la gestión, unificación y optimización de los datos sea una tarea pendiente y esencial para el futuro y éxito del proyecto.

1.2. Motivación

El gran problema que hay con los datos de *Flora* del proyecto *EnBiC2Lab* presenta una importante oportunidad para desarrollar soluciones innovadoras y eficientes para la gestión de datos. Por lo tanto, este trabajo de fin de grado busca abordar este desafío y proponer una solución efectiva para la gestión de datos en el proyecto *EnBiC2Lab* utilizando las herramientas

y la tecnología que está a la orden del día.

Para abordar este problema, primero hay que actuar desde la raíz arreglando los datos ya existentes para finalmente desarrollar una herramienta que facilite la recogida de los datos futura y ayude a los integrantes del proyecto a no cometer los mismos errores. De esta manera el problema se habría solucionado, y se facilitaría que no volviera a ocurrir.

1.3. Objetivos

El objetivo principal del proyecto es mejorar la gestión de datos [5] del proyecto, lo que facilitará la difusión y el posterior estudio de los mismos. Para conseguirlo, el trabajo se ha dividido en dos partes:

1. **Procesado de datos:** En los datos de *Flora* hay una gran cantidad de datos que no están unificados, esto quiere decir que un mismo nombre está escrito de muchas maneras diferentes por lo que para utilizar estos datos y realizar estudios estadísticos los resultados no serían reales. Entonces, el primer objetivo es unificar esos nombres que están mal escritos eliminando el problema. El código del script de unificación se puede encontrar en *GitHub* [6].

2. **Desarrollo de la aplicación:** Después del procesado de los datos antiguos, hay que pensar en los datos futuros, ya que no sería eficiente seguir cometiendo los mismos errores. Para facilitar la recogida de datos futura se va a desarrollar una aplicación web permitirá una gestión correcta de los datos, evitando errores en la recolección de datos y facilitando la geolocalización. Además, la aplicación estará diseñada para ser accesible tanto en línea, como sin conexión (algo esencial porque la recolección de muestras se hace mayoritariamente en lugares con escasa o ninguna conexión de red por lo que la creación de una aplicación que requiera de internet de forma ininterrumpida no solucionaría el problema). La aplicación contará con una interfaz intuitiva y fácil de usar permitiendo visualización de los datos registrados por los usuarios para una mejor comprensión y análisis de los datos recopilados. El código de la aplicación se puede encontrar en *GitHub* [7]. En el desarrollo de la aplicación se tienen en cuenta los siguientes aspectos:

- Mejorar la eficiencia en la gestión de datos.
- Evitar errores en la recolección de datos.
- Facilitar la geolocalización.
- Accesibilidad en línea y sin conexión.
- Interfaz intuitiva y fácil de usar.
- Visualización de datos registrados por los usuarios.

Finalmente, con el objetivo cumplido, el proyecto tendrá una base de datos unificada y una herramienta que facilitará que se mantenga unificada en el futuro.

1.4. Estado del arte

En el ámbito de la tecnología y aplicaciones relacionadas con la biodiversidad y las plantas, se han desarrollado varias herramientas que han demostrado ser útiles para la identificación, recopilación de datos y promoción de la conservación. A continuación, se presenta un resumen del estado del arte basado en las herramientas y aplicaciones mencionadas.

- **iNaturalist** es una aplicación móvil diseñada para conectar a las personas con la naturaleza y generar datos científicamente valiosos sobre la biodiversidad [8]. La aplicación permite a los usuarios compartir información sobre las especies que encuentran, realizar identificaciones y contribuir a proyectos de ciencia ciudadana. La investigación académica ha examinado cómo esta aplicación puede despertar el interés por el medio ambiente natural, al tiempo que muestran una visión particular de la naturaleza basada en las ciencias naturales.

- **Flora Incognita** es una aplicación que ofrece la capacidad de identificar especies de plantas de manera interactiva mediante el uso de algoritmos de aprendizaje profundo [9]. Los usuarios pueden capturar observaciones y contribuir a la comprensión de la biodiversidad de las plantas. La aplicación proporciona una interfaz intuitiva y materiales educativos complementarios para facilitar la identificación precisa, independientemente del nivel de conocimiento botánico del usuario. *Flora Incognita* presenta cómo las aplicaciones móviles pueden fomentar la ciencia ciudadana, la conservación y la educación.

- **Flora Capture** es una aplicación móvil que permite a los usuarios recolectar y subir imágenes de plantas con meta-datos [10]. Es útil para entrenar algoritmos de identificación de plantas y proporciona información valiosa sobre la morfología. Con miles de especies y cientos de miles de imágenes recolectadas, es una herramienta muy potente y útil..
- **Flora - Green Focus** es una aplicación que combina la gestión del tiempo con la promoción de la plantación de árboles [11]. Los usuarios se comprometen a permanecer en la aplicación durante un tiempo establecido para ver crecer un árbol virtual. Si abandonan la aplicación, el árbol muere. La aplicación utiliza un enfoque de gamificación para incentivar la concentración y la productividad, al tiempo que ayuda a plantar árboles reales mediante donaciones monetarias. *Flora - Green Focus* muestra cómo las aplicaciones pueden aprovechar la tecnología para fomentar comportamientos sostenibles y la conciencia ambiental.
- **El proyecto World Flora Online (WFO)** es una base de datos de acceso abierto que tiene como objetivo publicar información en línea de todas las plantas conocidas [12]. WFO aborda los desafíos en la recopilación y unificación de datos de plantas, proporcionando una fuente centralizada y confiable de información sobre la diversidad de plantas en todo el mundo. WFO pretende impulsar la conservación de las especies de plantas y proporcionar una plataforma para la investigación y el conocimiento.

1.5. Estructura del documento

La estructura del trabajo será la siguiente:

- El Capítulo 1 introduce el contexto de este Trabajo Fin de Grado.
- En el Capítulo 2 se describen las tecnologías usadas en el desarrollo de la aplicación para el cumplimiento de las distintas funcionalidades.
- El Capítulo 3 se centra en el procesado de los datos, su almacenamiento y en el desarrollo de la aplicación con sus diferentes páginas y características de forma precisa.
- Finalmente, en el Capítulo 4 se van a exponer las conclusiones del desarrollo de la aplicación y de las lineas futuras a seguir para avanzar en este ámbito.

2

Tecnologías usadas

2.1. *Prisma*

Prisma es una herramienta de código abierto conocida como *ORM (Object-Relational Mapping)* utilizada para la gestión de bases de datos relacionales. Con *Prisma*, los desarrolladores pueden interactuar con la base de datos escribiendo código en diferentes lenguajes, sin necesidad de utilizar *SQL* directamente, lo que agiliza y simplifica el proceso de desarrollo.

Prisma ofrece una *API* de alto nivel que facilita las operaciones *CRUD* (Crear, Leer, Actualizar, Eliminar) y otras tareas relacionadas con la base de datos. Además, genera automáticamente el esquema de la base de datos a partir del modelo de datos definido, eliminando la necesidad de escribir código *SQL* para crear tablas y columnas.

Esta herramienta es compatible con varias bases de datos relacionales como *PostgreSQL, MySQL y SQLite*, y admite diferentes modelos de datos, tanto los tradicionales como los orientados a objetos.

Uno de los aspectos destacados de *Prisma* es su capacidad para generar consultas de base de datos optimizadas y eficientes. Al utilizar *Prisma*, las consultas se optimizan automáticamente para obtener los datos de manera más eficiente, lo que maximiza el rendimiento y disminuye el tiempo de respuesta de las consultas.

Prisma es una herramienta valiosa para los desarrolladores que trabajan con bases de datos relacionales. Proporciona una *API* sencilla de usar, automatiza tareas de base de datos y ofrece un rendimiento optimizado. Su compatibilidad con diversas bases de datos y modelos de datos lo convierte en una opción sólida para aplicaciones empresariales críticas, donde la seguridad y la eficiencia son prioritarias.

2.2. *MongoDB*

MongoDB [13] es un sistema de gestión de bases de datos *NoSQL* de código abierto que se destaca por su capacidad para guardar y utilizar datos de manera rápida y eficiente. A diferencia de los sistemas de bases de datos relacionales, *MongoDB* utiliza una estructura de datos basada en documentos en lugar de tablas, lo que proporciona mayor flexibilidad y escalabilidad. Los datos se guardan en documentos *BSON (Binary JSON)* y se pueden anidar para crear estructuras de datos complejas.

MongoDB se caracteriza por su capacidad para tratar grandes cantidades de datos y escalar horizontalmente. También es conocido por su capacidad para realizar consultas complejas de manera eficiente. Utiliza *MongoDB Query Language (MQL)*, un lenguaje de consulta similar a *SQL* diseñado específicamente para trabajar con estructuras de datos de documentos. *MQL* es fácil de aprender y utilizar, lo que permite realizar consultas avanzadas sin necesidad de escribir código complejo.

Además, *MongoDB* es compatible con muchos lenguajes de programación, como *Java, Python y JavaScript*, y se puede utilizar en diversas plataformas, lo que lo hace versátil en diferentes contextos de desarrollo.

MongoDB es una herramienta popular para desarrolladores que trabajan con grandes volúmenes de datos y requieren una base de datos flexible y escalable. Su enfoque no relacional y la utilización de una estructura de datos basada en documentos brindan ventajas en términos de flexibilidad y escalabilidad. Además, su compatibilidad con múltiples lenguajes de programación y plataformas lo convierte en una opción útil en diversos escenarios de desarrollo.

2.3. *MinIO*

MinIO es una solución de almacenamiento en la nube de código abierto que ofrece una alternativa a los servicios de almacenamiento en la nube comercial como *Amazon S3*. *MinIO* está diseñado para ser compatible con los estándares de *Amazon S3*, lo que significa que las aplicaciones que se han desarrollado para S3 también pueden usar *MinIO* sin tener que hacer cambios significativos en el código.

Una de las principales características de *MinIO* es su alta disponibilidad. *MinIO* utiliza un enfoque de almacenamiento distribuido para garantizar que los datos estén disponibles en todo momento. Los datos se distribuyen en múltiples nodos, lo que significa que si uno de los nodos falla, los datos aún estarán disponibles en los otros nodos. Además, *MinIO* utiliza la replicación de datos para garantizar que los datos se copien en múltiples ubicaciones.

Otra característica importante de *MinIO* es su escalabilidad sencilla. *MinIO* está diseñado para ser escalable horizontalmente, lo que significa que se pueden agregar más servidores para aumentar la capacidad de almacenamiento. *MinIO* también es compatible con la arquitectura de contenedores, lo que facilita su implementación y escalabilidad en entornos de contenedores.

MinIO también ofrece una gran cantidad de características de seguridad para garantizar que los datos almacenados estén protegidos. Ofrece encriptación de extremo a extremo, autenticación de acceso basada en políticas y compatibilidad con el protocolo *HTTPS* para transacciones seguras.

2.4. *Python*

Python es un lenguaje de programación muy utilizado en diversas áreas de la informática, desde el desarrollo web hasta la inteligencia artificial. En este trabajo, se ha utilizado *Python* para procesar información mediante el uso de algoritmos y estructuras de datos *NPL*.

En particular, se ha implementado un programa en *Python* para la unificación un archivo *JSON* que contiene información sobre plantas. Este programa utiliza la biblioteca estándar de *Python* y algunas otras bibliotecas para realizar operaciones como leer y escribir archivos, procesar texto y aplicar algoritmos de comparación de cadenas de caracteres.

El programa realiza una serie de operaciones para unificar los nombres de las comunidades y subcomunidades de las plantas, lo que permite una mayor consistencia y facilidad de uso en la base de datos. Para esto, se utiliza un algoritmo de comparación de cadenas de caracteres conocido como distancia de *Levenshtein*, que mide la diferencia entre dos cadenas.

Python es una herramienta muy útil para el procesamiento de información y ofrece una gran variedad de bibliotecas para facilitar la implementación estructuras de datos y algoritmos. En este trabajo, se ha utilizado *Python* para procesar información sobre plantas y se ha implementado un programa que utiliza algoritmos de comparación de cadenas de caracteres para unificar los nombres de las comunidades y subcomunidades de los datos de *Flora*.

2.5. *PLN*

El procesamiento del lenguaje natural (*NLP*) [14] es una parte de la inteligencia artificial encargada de analizar, comprender y generar lenguaje humano de forma automática. En este campo se trabajan diferentes tareas, como la clasificación de texto, la traducción de textos automática, la generación de texto, el análisis de sentimientos y la unificación de nombres, entre otras.

La unificación de nombres es una tarea importante en *NLP*, especialmente cuando se trata de datos científicos y de investigación. Los nombres pueden estar escritos de forma distinta o contener errores ortográficos, lo que puede dificultar su búsqueda y análisis. Por esta razón, es importante tener herramientas que nos permitan unificar los nombres y eliminar los errores ortográficos.

Una de las técnicas más utilizadas en *NLP* [15] para la unificación de nombres es la distancia

de *Levenshtein*, que mide menor número de operaciones necesarias para convertir una cadena de caracteres en otra. Esta técnica es especialmente útil para detectar nombres que difieren en una letra o que tienen errores ortográficos simples.

Otra técnica comúnmente utilizada es el uso de algoritmos de agrupamiento para identificar nombres similares y agruparlos juntos. Los algoritmos de agrupamiento son capaces de encontrar patrones en los datos y agruparlos en función de su similitud, lo que puede ayudar a identificar nombres que difieren en varias letras.

En cuanto a la detección de errores ortográficos, existen diferentes herramientas que se pueden utilizar, como diccionarios de corrección ortográfica y algoritmos de detección de errores ortográficos. Estos últimos suelen basarse en modelos de lenguaje que utilizan la frecuencia de las palabras y las combinaciones de letras para detectar posibles errores ortográficos.

La unificación de nombres y la detección de errores ortográficos son tareas importantes en *NLP* que nos permiten trabajar con datos de forma más eficiente y precisa. La distancia de *Levenshtein*, los algoritmos de agrupamiento y las herramientas de detección de errores ortográficos son algunas técnicas de las más utilizadas para conseguir el objetivo.

2.6. *JavaScript*

JavaScript es un lenguaje de programación de alto nivel, dinámico e interpretado ampliamente utilizado en el desarrollo de aplicaciones web y la adición de interactividad a las páginas existentes.

Una de las principales características de *JavaScript* es su capacidad para manipular el *Document Object Model (DOM)* de una página web. El *DOM* representa la estructura de una página en la memoria y permite a los desarrolladores manipular su contenido y presentación de forma dinámica.

JavaScript es también un lenguaje orientado a objetos que admite la creación de objetos y clases, herencia y encapsulamiento de datos y métodos.

Otro aspecto importante de *JavaScript* es su naturaleza interpretada, es decir, se puede ejecutar sin anteriormente ser compilado. Esto facilita a los desarrolladores probar y modificar su código de manera ágil.

Además, *JavaScript* es un lenguaje multiplataforma que se puede utilizar en diversos entornos, desde navegadores web hasta servidores. También es compatible con diferentes para-

digmas de programación, como la programación funcional y asincrónica.

JavaScript es un lenguaje de programación versátil y poderoso muy utilizado en el desarrollo de aplicaciones web interactivas. Sus capacidades para manipular el *DOM*, su orientación a objetos, su interpretación directa y su compatibilidad multiplataforma siendo ideal para los desarrolladores web.

2.7. *Next.js*

Next.js es un *framework* de React muy utilizado para crear aplicaciones con gran rendimiento y velocidad. Se destaca por su capacidad de renderizar las páginas web en el lado del servidor (*SSR*), lo que mejora significativamente el tiempo de carga de la página al enviar versiones renderizadas desde el servidor al navegador del usuario.

Además de su enfoque en el *SSR*, *Next.js* ofrece diversas funcionalidades que simplifican y aceleran el desarrollo de aplicaciones web. Por ejemplo, incluye soporte integrado para enrutamiento, renderizado de CSS en el servidor y optimización automática de imágenes. También se destaca por su arquitectura modular, lo que permite personalizar la aplicación utilizando *plugins* y librerías según las necesidades del proyecto.

Una ventaja importante de *Next.js* es su capacidad para crear tanto aplicaciones de una sola página (*SPA*) como aplicaciones de varias páginas (MPA), e incluso combinar ambas formas de navegación en aplicaciones híbridas. También ofrece una función de precarga de páginas (*preloading*), que permite cargar la siguiente página antes de que el usuario haga clic en el enlace correspondiente, mejorando aún más la experiencia de carga.

Además, *Next.js* es compatible con tecnologías de vanguardia como *React Native y GraphQL*. Esto permite a los desarrolladores crear aplicaciones web y móviles con una experiencia de usuario coherente utilizando la misma tecnología y las mismas herramientas.

Next.js es un framework de React centrado en el rendimiento y la velocidad, gracias a su enfoque en el *SSR*. Proporciona funcionalidades integradas que facilitan el desarrollo de aplicaciones web, es compatible con tecnologías modernas y permite crear aplicaciones web y móviles con una experiencia de usuario consistente.

2.8. *Visual Studio Code*

Visual Studio Code (VS Code) es un editor de código fuente desarrollado por Microsoft, que se ha convertido en uno de los más populares en su categoría. Es ampliamente utilizado para programar en una amplia variedad de lenguajes, como *JavaScript, Python, Java, C++* , y muchos otros.

Una de las principales características de *VS Code* es su capacidad para personalizar el entorno de desarrollo a través de extensiones. Existe una amplia gama de extensiones disponibles para instalar en *VS Code*, las cuales agregan nuevas funcionalidades al editor y lo hacen más eficiente y fácil de usar. Algunas de las extensiones más populares incluyen:

- **Prettier:** Esta extensión formatea automáticamente el código en un estilo consistente, lo cual ahorra tiempo y ayuda a evitar errores de estilo.

- **ESLint:** Esta extensión verifica automáticamente el código en busca de errores y problemas de estilo, y proporciona sugerencias para corregirlos.

- **GitLens:** Esta extensión mejora la integración de Git en *VS Code*, proporcionando información y herramientas útiles para trabajar con repositorios de Git.

- **Debugger for Chrome:** Esta extensión permite depurar código *JavaScript* directamente desde *VS Code*, lo cual facilita la identificación y corrección de errores.

- Otra característica destacada de *VS Code* es su integración con herramientas de desarrollo web, como *Node.js y TypeScript.* Además, cuenta con una amplia variedad de funciones integradas que facilitan la edición de código, como la búsqueda y reemplazo de texto, la navegación de archivos y la gestión de proyectos.

Visual Studio Code es un editor de código fuente altamente personalizable y flexible, desarrollado por Microsoft. Su capacidad para integrarse con diferentes herramientas y lenguajes de programación lo convierte en una herramienta valiosa para los desarrolladores de software en todo el mundo.

2.9. *PWA*

Una *Progressive Web App (PWA)* es una aplicación web que tiene la capacidad de funcionar tanto en línea como sin conexión. En el caso de la gestión de datos de *Flora*, una *PWA*, sería una opción ideal para mejorar la eficiencia en la gestión de datos, evitar errores en la recolección de datos y facilitar la geolocalización.

Una *PWA* para la gestión de datos de *Flora* permitiría a los usuarios recopilar y registrar datos directamente en la aplicación, en lugar de tener que usar formularios de papel o de hojas de cálculo. La aplicación también podría ofrecer funciones de geolocalización para facilitar la recopilación de datos relacionados con la ubicación. Además, al ser una *PWA*, los usuarios podrían acceder a la aplicación tanto en línea como sin conexión, lo que sería particularmente útil en zonas remotas donde la conexión a Internet puede ser limitada o inestable.

La interfaz de usuario de la *PWA* debería ser intuitiva y fácil de usar, lo que ayudaría a minimizar errores en la recolección de datos y a fomentar la participación de los usuarios. La visualización de los datos registrados por los usuarios también sería un aspecto clave de la aplicación, permitiendo una mejor comprensión y análisis de los datos recopilados.

Una *PWA* para la gestión de datos de *Flora* sería una herramienta valiosa para mejorar la eficiencia y precisión en la recolección y gestión de datos, facilitar la geolocalización y ofrecer una experiencia de usuario accesible en línea y sin conexión.

2.10. *Dexie*

Dexie es una biblioteca de indexación de bases de datos en *JavaScript* que permite almacenar y recuperar datos en el navegador sin necesidad de conexión a internet. Utiliza *IndexedDB* como motor de almacenamiento subyacente y aprovecha las características de *JavaScript* para mejorar el rendimiento.

A continuación se destacan los aspectos clave de *Dexie* relacionados con su capacidad de uso sin conexión:

- ***IndexedDB***: *Dexie* utiliza *IndexedDB* como motor de almacenamiento para guardar los datos en el navegador sin conexión. *IndexedDB* es una base de datos NoSQL integrada en los navegadores modernos.

- **Promesas y async/await**: *Dexie* utiliza promesas y *async/await* de *JavaScript* para realizar operaciones de almacenamiento y recuperación de datos de manera asíncrona, evitando bloqueos en la interfaz de usuario y mejorando el rendimiento.

- **Índices**: *Dexie* permite la creación de índices en las bases de datos almacenadas en *IndexedDB*, mejorando la eficiencia de las consultas y el acceso a los datos.

- **Transacciones y versionado de bases de datos**: *Dexie* facilita el uso de transacciones para garantizar la consistencia de los datos durante las operaciones de almacenamiento y recuperación. Además, ofrece funciones para actualizar y gestionar el versionado de las bases de datos, lo que permite realizar cambios en el esquema de datos sin perder información existente.

- **Eventos y notificaciones**: *Dexie* utiliza eventos y notificaciones para manejar cambios en los datos almacenados, lo que facilita la implementación de lógica personalizada en respuesta a dichos cambios. Esto ayuda a mantener la aplicación actualizada con los cambios en la base de datos.

Dexie es una biblioteca que permite almacenar y recuperar datos en el navegador sin conexión. Utiliza *IndexedDB* como motor de almacenamiento y aprovecha características como

promesas, índices, transacciones y eventos para ofrecer una solución eficiente y confiable. *Dexie* facilita el desarrollo de aplicaciones web con capacidad *offline* al permitir el acceso a los datos incluso en ausencia de conexión a internet.

3
Desarrollo

Durante la fase de desarrollo del proyecto se explicará la forma en la que se han aplicado las tecnologías explicadas anteriormente. En esta sección se va a tener especialmente en cuenta que para el uso *online* y *offline* de la aplicación se van a utilizar herramientas distintas para cumplir todas funcionalidades y objetivos.

3.1. Bases de datos

Para guardar y consultar los datos, se van a utilizar dos gestores de bases de datos que son *MongoDB* y *Dexie*. Cada uno se utilizará en unas circunstancias distintas y es una de las principales características que se han tenido en cuenta para garantizar la doble función *online*/*offline* de forma satisfactoria como se explicará a continuación.

- **Dexie:** *Dexie* es una biblioteca de base de datos en línea basada en *IndexedDB* que permite el almacenamiento y recuperación de datos en el navegador del usuario. Su principal característica es que permite guardar los datos de forma local sin necesidad de conexión a internet, por lo que mientras la aplicación está funcionando de forma *offline* es la base de datos que garantizan la funcionalidad total de la aplicación. En esta base de datos se ha guardado la información necesaria para el autorrellenado de los datos y las listas desplegables del formulario. *Dexie* también proporciona una gran flexibilidad a la hora de la inserción de datos, que como *MongoDB*, también es *NoSQL* y se utilizan objetos de tipo *JSON*. En la base de datos local se han creado diferentes colecciones (Figura 1) que se consultan dependiendo el dato al que se quiere acceder, las diferentes colecciones son:

    - **Authors:** Esta colección guarda los datos de los usuarios registrados en el sistema, esta colección permite al usuario añadir autores a las muestras sin necesidad de

consultar la base de datos remota, garantizando la función *offline* en la inserción de los autores.

- **Community, subcommunity y species:** Estas colecciones guardan los datos de los parques naturales con las comunidades/subcomunidades y especies que hay en ellos. Gracias a estas colecciones se pueden ofrecer al usuario un desplegable con la lista de los datos que ya hay en la base de datos, lo que garantiza que si alguno de los datos ya fue guardado anteriormente, se puede seleccionar y de esta manera se evitan errores en la escritura y posterior duplicidad de los datos.
- **Flora:** Esta colección guarda los datos de las muestras que están en la base de datos local después de enviar el formulario. Gracias a esta colección se puede mostrar una lista de las muestras locales que todavía no se han sincronizado con la base de datos remota permitiendo su edición, visualización y en caso de que el usuario lo desee la eliminación de la muestra. La colección *flora* es esencial en la función *offline* de la aplicación porque es el paso intermedio que garantiza que toda muestra pueda pasar de la base de datos local a la base de datos remota mediante la sincronización directa de los datos (Figura 2).
- **Time:** Esta colección guarda los datos de la última actualización de la base de datos local para que en caso que haya conexión a internet, si han pasado 10 minutos desde la última actualización, se hace de forma automática. Gracias a esta colección el usuario puede ver cuando fue la última actualización y si lo desea, volver a hacerlo.

- ***MongoDB*:** *MongoDB* es el gestor de base de datos utilizado para guardar las muestras que ya se han sincronizado de la aplicación. En *MongoDB* está la colección donde se guardan todas las muestras que se han ido tomando y las vistas de la colección para acceder a los datos que permiten el autorrellenado en el formulario. La colección de *MongoDB*, se basa en una lista de diccionarios y es perfectamente compatible con los diccionarios que se van insertando en la base de datos local por lo que la actualización entre base de datos local y remota es directa. Otra característica es que es muy fácil trabajar por la flexibilidad que hay con los datos al ser *NoSQL* como se ha comentado anteriormente y la posibilidad de exportar e importar los datos de una forma sencilla y rápida (Figura 3). Una característica interesante de *MongoDB* es que proporciona un

análisis de los datos dependiendo el tipo de dato, por ejemplo se puede ver un diagrama de barras con la distribución de los años de las muestras y del porcentaje de coverage de las muestras (Figura 4).

La utilización de ambas bases de datos ha sido una de las tareas principales sobre las que mas se ha tenido que trabajar para poder desarrollar una aplicación que pueda funcionar sin conexión y la correcta transición entre ellas esencial para garantizar todas las funciones de autorrellenado evitando errores de escritura que posteriormente pudieran provocar datos erróneos y un análisis poco eficiente.

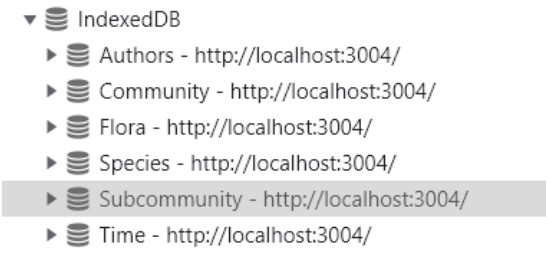

Figura 1: Colecciones guardadas en la base de datos local

Figura 2: Ejemplo colección base de datos local *Flora*

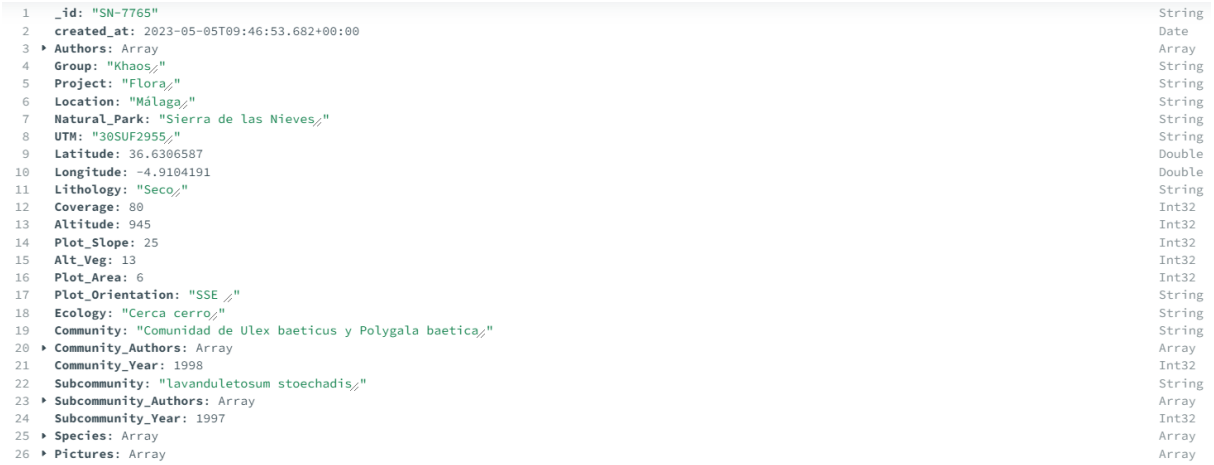

Figura 3: Muestra de la colección de datos de *MongoDB*.

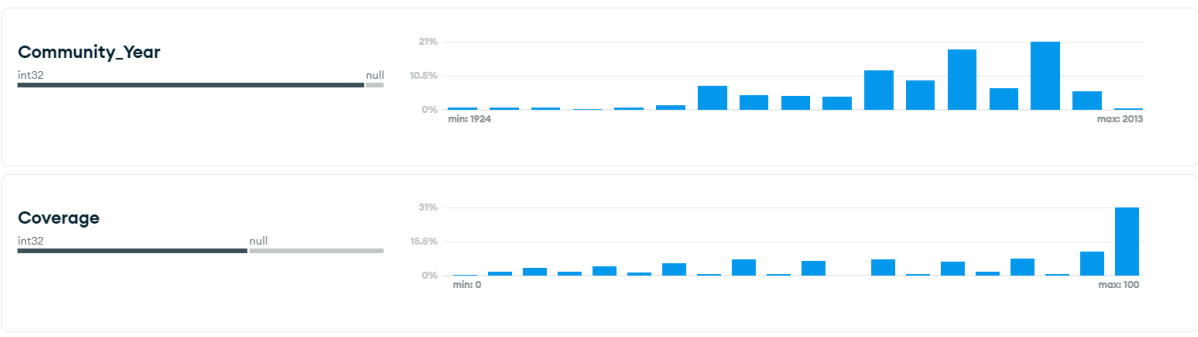

Figura 4: Ejemplo análisis de datos en *MongoDB*.

3.2. *Online* y *Offline*

Una de las características principales de la aplicación es que se puede utilizar sin conexión de red. El único momento en el que se requiere internet es cuando un usuario quiere iniciar sesión o registrarse por lo que el usuario debe tener en cuenta que para usar la aplicación primero debe iniciar sesión con una conexión estable.

Cuando un usuario ya a iniciado sesión, tener conexión de red pasa a un segundo plano porque ya puede acceder a todas las páginas y funciones de la aplicación sin preocuparse por la conexión de red. Para conseguir esta funcionalidad se ha desarrollado la aplicación teniendo en cuenta que en cualquier momento el usuario puede perder la conexión, entonces, en todas las páginas encontramos características que pertenecen a una función con conexión, sin conexión y ambas, haciendo continuamente comprobaciones del estado de la red.

En la aplicación podemos encontrar las siguientes características que permiten que la aplicación se pueda usar sin conexión manteniendo todas las funcionalidades:

- **Unificación de datos:** Para poder tener disponible una lista con los datos de autores, comunidades, subcomunidades, especies y parques naturales se ha utilizado una base de datos local en la que se pueden almacenar estos datos que están alojados en la base de datos remota sin necesidad de acceso a internet. Gracias al uso de una base de datos local se puede disponer de forma *offline* a los datos necesarios para garantizar la función de las listas desplegables que evitan errores en la inserción de datos.

- **Sincronización:** Cada vez que un usuario inserta una nueva muestra, esta muestra es guardada en la base de datos local para poder sincronizarla posteriormente en caso de

tener conexión a internet, gracias a esta característica no tendremos ningún problema en la inserción de muestras haya o no conexión a internet teniendo dos funciones bien diferenciadas y con unas funciones distintas.

- **Actualización:** Cada usuario puede actualizar su base de datos local para tener todas las especies y parques naturales nuevos. Lo que permite al usuario que si hay varias personas trabajando con la aplicación al mismo tiempo, mantener su base de datos local actualizada. Cuando el usuario tiene conexión a internet habrá una actualización automática si desde la última han pasado más de 10 minutos, gracias a esto, si al usuario se le olvida actualizar, no tendrá datos desfasados más de 10 minutos.

3.3. *Prisma*

A continuación se va a describir el esquema de *Prisma*, los modelos de datos y las relaciones que hay entre ellos (Figura 5).

```
generator client {
  provider        = "prisma-client-js"
  output          = "../node_modules/@~internal/prisma"
  previewFeatures = ["referentialIntegrity"]
}
datasource db {
  provider             = "mysql"
  url                  = env("DATABASE_URL")
  referentialIntegrity = "prisma"
}
model User {
  id         Int                    @id @default(autoincrement())
  createdAt  DateTime               @default(now())
  email      String                 @unique
  username   String                 @unique
  fullname   String?
  password   String
  role       Role                   @default(USER)
  belongsTo  UsersOnResearchGroup[]
  isDisabled Boolean                @default(false)
}
```

```
model UsersOnResearchGroup {
  user            User          @relation(fields: [userId], references: [id])
  userId          Int
  researchGroup   ResearchGroup @relation(fields: [researchGroupId], references: [id])
  researchGroupId Int
  assignedAt      DateTime      @default(now())
  assignedBy      String
  status          Status        @default(GUEST)
  @@id([userId, researchGroupId])
}
model ResearchGroup {
  id    Int                  @id @default(autoincrement())
  name  String               @unique
  users UsersOnResearchGroup[]
}
enum Role {
  USER
  ADMIN
}
enum Status {
  RESEARCHER
  PRINCIPAL
  GUEST
}
```

Figura 5: Capturas schema.Prisma

- **Generador y cliente:**
  - El generador y cliente se configuran para generar el cliente de Prisma en *JavaScript*.
  - El cliente de Prisma se utilizará para realizar consultas y manipulaciones de datos en la base de datos.

- **Fuente de datos:**
  - La fuente de datos está configurada para conectarse a una base de datos *MySQL*.
  - La URL de la base de datos se obtiene de la variable de entorno *DATABASEURL*.
  - La integridad referencial se habilita para la base de datos utilizando Prisma.
- **Modelo User:**
  - El modelo ***User:*** representa una entidad en la base de datos llamada *User*.
  - Tiene varios campos como *id*, *createdAt*, *email*, *username*, *fullname*, *password*, *role*, *belongsTo* y *isDisabled*.
  - ***id:*** es un campo de tipo entero que actúa como identificador único y tiene la opción *@id* para indicar que es una clave primaria.
  - ***createdAt:*** es un campo de tipo DateTime que registra la fecha y hora de creación del usuario y tiene el valor predeterminado *now()* para establecerlo automáticamente al momento de la creación.
  - ***email:*** es un campo de tipo String que debe ser único.
  - ***username:*** es un campo de tipo String que también debe ser único.
  - ***fullname:*** es un campo de tipo String que es opcional, lo que significa que puede estar vacío.
  - ***password:*** es un campo de tipo String que representa la contraseña del usuario.
  - ***role:*** es un campo de tipo enumerado (enum) llamado *Role* que puede tener los valores *USER* o *ADMIN*. El valor predeterminado es *USER*.
  - ***belongsTo:*** es una relación con el modelo *UsersOnResearchGroup*. Indica que un usuario puede pertenecer a varios grupos de investigación.
  - ***isDisabled:*** es un campo de tipo Booleano que indica si el usuario está deshabilitado. El valor predeterminado es *false*.
- **Modelo UsersOnResearchGroup:**
  - El modelo ***UsersOnResearchGroup:*** representa una tabla de unión entre los modelos *User* y *ResearchGroup*.

- Tiene los campos *user*, *userId*, *researchGroup*, *researchGroupId*, *assignedAt*, *assignedBy* y *status*.
- ***user:*** es una relación con el modelo *User* y se especifica mediante la directiva *@relation*.
- ***userId:*** es un campo de tipo entero que se refiere al campo *id* en el modelo *User*.
- ***researchGroup:*** es una relación con el modelo *ResearchGroup*.
- ***researchGroupId:*** es un campo de tipo entero que se refiere al campo *id* en el modelo *ResearchGroup*.
- ***assignedAt:*** es un campo de tipo DateTime que registra la fecha y hora de asignación y tiene el valor predeterminado *now()*.
- ***assignedBy:*** es un campo de tipo String que indica quién realizó la asignación.
- *status* es un campo de tipo enumerado (enum) llamado *Status* que puede tener los valores *RESEARCHER*, *PRINCIPAL* o *GUEST*. El valor predeterminado es *GUEST*.
- ***@@id([userId, researchGroupId]):*** especifica que la combinación de *userId* y *researchGroupId* forma una clave primaria compuesta para esta tabla de unión.

- **Modelo ResearchGroup:**
  - El modelo ***ResearchGroup:*** representa una entidad en la base de datos llamada *ResearchGroup*.
  - Tiene los campos *id* y *name*.
  - ***id:*** es un campo de tipo entero que actúa como identificador único y tiene la opción *@id* para indicar que es una clave primaria.
  - ***name:*** es un campo de tipo String que debe ser único.

- **Enumeraciones:**
  - Se definen dos enumeraciones (enums) llamadas *Role* y *Status*.
  - ***Role*** puede tener los valores *USER* o *ADMIN*.
  - ***Status*** puede tener los valores *RESEARCHER*, *PRINCIPAL* o *GUEST*.

- **Relaciones entre los datos:**

    - Un usuario (*User*) puede pertenecer a varios grupos de investigación (*ResearchGroup*) a través de la relación *belongsTo* en el modelo *User*.
    - Un usuario puede tener varios registros en la tabla de unión *UsersOnResearchGroup* a través de la relación *belongsTo*.
    - Un grupo de investigación puede tener varios usuarios a través de la relación *users* en el modelo *ResearchGroup*.
    - La tabla de unión *UsersOnResearchGroup* establece la relación entre un usuario y un grupo de investigación mediante los campos *userId* y *researchGroupId*.
    - La tabla de unión también registra información adicional como la fecha de asignación (*assignedAt*), el responsable de la asignación (*assignedBy*) y el estado (*status*) del usuario en el grupo de investigación.

3.4. Preprocesado

En un proyecto de gestión de datos, es importante tener una base de datos unificada y coherente para poder realizar análisis estadísticos precisos. Sin embargo, los datos iniciales pueden estar sujetos a errores de escritura o transcripción, lo que resulta que un mismo dato esté escrito de diferentes maneras. Esto dificulta el acceso a los datos reales y falsea cualquier análisis que se realice.

En este caso, los datos de *Flora* presentaban el gran problema de que los nombres de las comunidades y subcomunidades no siempre estaban escritos de la misma manera. En el caso de las comunidades, se encuentran 351 comunidades únicas en un principio cuando realmente, después del preprocesado se reducen a 322 por lo que se unifican 29 comunidades y en el caso de las subcomunidades se reducen de 97 a 93.

Para resolver este problema, se desarrolló un componente en *Python* utilizando *NLP* (Procesamiento de Lenguaje Natural) para unificar los nombres de las comunidades y subcomunidades. En primer lugar, se cargó el *JSON* de la base de datos original de *MongoDB* mediante el paquete *JSON* de *Python*, después se creó una lista de diccionarios con todas las comunidades y subcomunidades que había.

Después de crear la lista de diccionarios, se tenía la información del número original de comunidades distintas y la aparición de cada una de ellas. Entre esas comunidades había algunas que se referían a la misma pero que estaban mal escritas lo que era el principal problema de la base de datos por lo que había que solucionar ese problema.

Para abordar el problema, primero hay que localizar cuales son los nombres similares y que, por tanto, se referirían a la misma entidad. Para ello se ha utilizado la función *lev()* del paquete *Levenshtein*. Esta función es muy utilizada en las tareas de NPL porque permiten calcular la distancia entre palabras, es decir, el numero de letras que deben cambiarse para que dos palabras sean iguales.

Utilizando la distancia de *Levenshtein* se hicieron diferentes pruebas estableciendo el índice óptimo (comunidades:5 subcomunidades:3) en el que dos palabras podrían considerarse iguales para que el filtrado fuera lo suficientemente preciso no confundiendo distintas entidades como similares.

Después, se recorre toda la lista de diccionarios comparando cada entidad con todas las demás, en el caso que se consideren similares con la distancia de *Levenshtein*, se compara el número de apariciones. El nombre que tiene mayor número de apariciones es el que se ha considerado escrito correctamente, entonces el otro nombre quedará sustituido automáticamente.

Este mismo procedimiento se aplica a comunidades y subcomunidades y de esta manera, se logró reducir a un nombre de comunidad/subcomunidad por entidad. A continuación, se muestran las tablas con los cambios que ha habido en la base de datos con los nombres originales, sus repeticiones y los nombres a los que se han unificado y la distancia *Levenshtein* que había entre ellos (Tabla 1) (Tabla 2).

Subcomunidad original	Reps.	Subcomunidad unificada	Reps.	Lev
pinguiculatosum lusitanicae	1	*pinguiculetosum lusitanicae*	2	1
Quercetosum canariensis	3	*quercetosum canariensis*	7	1
quersetosum suberis	2	*quercetosum suberis*	19	1
hypochaeridetosum platylepidis	3	*hypochoeridetosum platylepidis*	29	1

Cuadro 1: Tabla de subcomunidades, unificación, apariciones y distancia Levenshtein

Comunidad original	Reps.	Comunidad unificada	Reps.	Lev
Clematidi cirrhosae-Ceratonietum siliquae	3	*Clematido cirrhosae-Ceratonietum siliquae*	5	1
Velezio rigidae-Asteriscetum aquatici	7	*Velezio rigidae-Astericetum aquaticae*	9	3
Calamintho baeticae-Gallietum scabri	4	*Calamintho baeticae-Galietum scabri*	8	1
Fumario sepium-Geranietum purpurei	1	*Fumario sepii-Geranietum purpurei*	4	2
Cytiso baetici-Arbutetum unedi	5	*Cytiso baetici-Arbutetum unedonis*	9	3
Davallio canariensis-Sedetum baetici	1	*Davallio canariensis-Sedetum baetici*	16	1
Pulicario paludosae-Agrostietum pourretii	1	*Pulicario paludosae-Agrostietum porretii*	2	1
Trifolio cherleri-Plantaginetum bellardi	6	*Trifolio cherleri-Plantaginetum bellardii*	21	1
Sideritido osteoxyllae-Teucrietum charidemi	7	*Siderito osteoxyllae - Teucrietum charidemi*	8	4
Siderito osteoxyllae-Teucrietum charidemi	8	*Siderito osteoxyllae-Teucrietum charidemi*	12	2
Phlomidi almeriensis-Ulicetum canescentis	7	*Phlomido almeriensis-Ulicetum canescentis*	8	1
Rubo ulmifoli-Coriarietum myrtifoliae	2	*Rubo ulmifolii-Coriarietum myrtifoliae*	4	1
Trachelio caeruleae-Adiantetum capilli-veneris	1	*Trachelio coerulei-Adiantetum capilli-veneris*	2	3
Phlomido lychnitidis-Brachypodietum ramosi	3	*Phlomido lychnitidis-Brachypodietum retusi*	4	3
Arenario capillipes-Iberidetum fontquerii	3	*Arenario capillipedis-Iberidetum fontqueri*	4	3
Sarcocapno baetici-Centaureetum clementei	3	*Sarcocapno baeticae-Centaureetum clementei*	5	2
Paeonio broteroi-Abietetum pinsapo	10	*Paeonio broteroi-Abietetum pinsaponis*	18	3
Rhamno pumilae-Saxifragetum granatensis	22	*Rhamno pumili-Saxifragetum granatensis*	41	2
Seseli granatensis-Festucetum hystricis	5	*Seselido granatensis-Festucetum hystricis*	10	2
Abieti pinsapo-Juniperetum sabinae	8	*Abieto pinsapo-Juniperetum sabinae*	22	1
Violo demetriae-Ionopsidietum prolongoi	3	*Violo demetriae-Jonopsidietum prolongoi*	16	1
Centaureo baticae-Carlinetum corymbosae	8	*Centaureo baeticae-Carlinetum corymbosae*	11	1
Asparago albi-Rhamnetum oleidis	7	*Asparago albi-Rhamnetum oleoidis*	42	1
Heliotropio supini-Paspaletum paspaloidis	1	*Heliotropio supini-Paspaletum paspalodis*	2	1
Hedysaro coronarii-Phalarideum caerulescentis	4	*Hedysaro coronarii-Phalaridetum caerulescentis*	5	1
Hedysaro coronarii-Phalaridetum caerulescentis	5	*Hedysaro coronarii-Phalaridetum coerulescentis*	14	1
Phlomidi almeriensis - Ulicetum canescentis	2	*Phlomidi almeriensis-Ulicetum canescentis*	7	2
Mayteno europaei - Periplocetum angustifoliae	2	*Mayteno europaei-Periplocetum angustifoliae*	13	2
Artemisio barrelieri - Launaeetum arborescentis	1	*Artemisio barrelieri-Launeetum arborescentis*	2	3

Cuadro 2: Tabla de comunidades, unificación, apariciones y distancia Levenshtein

Con esta base de datos unificada y coherente, es posible acceder a los datos reales de cada comunidad y subcomunidad para realizar búsquedas y análisis estadísticos precisos. Además, al tener una base de datos unificada, se pueden comparar los datos entre diferentes parques naturales y hacer análisis más complejos sobre la presencia de comunidades y subcomunidades en diferentes áreas geográficas.

3.5. Aplicación

En el desarrollo de la aplicación se tienen en cuenta principalmente los siguientes aspectos:

- **Intuitiva y fácil de usar:** La aplicación debe tener una interfaz amigable en la que el usuario puede usarla cómodamente, desplazarse entre las paginas y funcionalidades sin problemas.
- **Cómoda y eficiente:** La aplicación debe estar diseñada para que los usuarios puedan realizar sus tareas de manera cómoda y eficiente, sin sentirse abrumados o confundidos.
- **Funciona en todas las circunstancias:** La aplicación debe ser capaz de funcionar correctamente en todas las situaciones, independientemente del dispositivo, sistema operativo o conexión a internet del usuario.
- **Seguridad de los datos:** La aplicación debe tener medidas de seguridad adecuadas para proteger los datos del usuario de posibles amenazas
- **Funcionalidad *offline*:** La aplicación debe garantizar su correcta utilización cuando el usuario no tiene conexión a internet.

A continuación se van a explicar las diferentes páginas de la aplicación y sus funciones.

3.5.1. *Login*

La primera página que se encuentra en la aplicación es el formulario de inicio de sesión que proporciona una interfaz para que los usuarios ingresen sus credenciales y autenticarse en la aplicación. Está implementado en *React* utilizando *Next.js* y *NextAuth* para gestionar la autenticación.

El formulario de inicio de sesión consta de los siguientes elementos principales:

- ***Campos de Usuario y Contraseña***: El formulario incluye dos campos de entrada, uno para el nombre de usuario y otro para la contraseña. Los usuarios deben ingresar sus credenciales en estos campos (Figura 6).

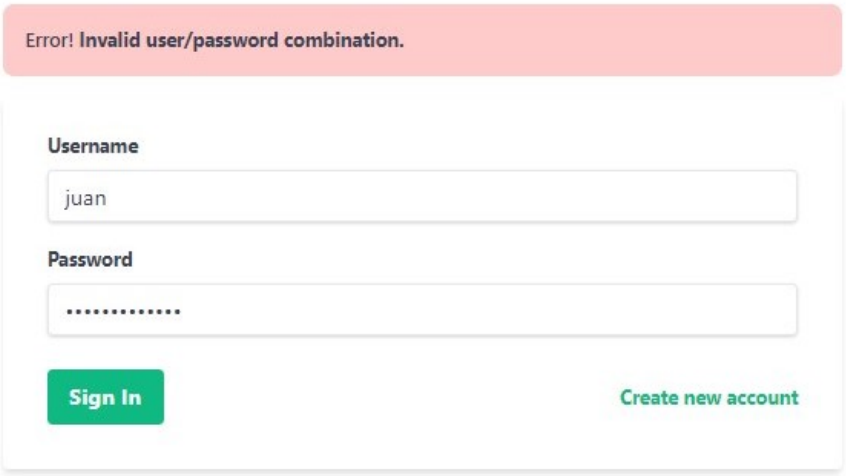

Figura 6: Captura de pantalla página inicio sesión erróneo.

- ***Validación de Credenciales***: Cuando el usuario envía el formulario, se llama a la función ***validateCredentials***. Esta función utiliza la función ***signIn*** de ***next-auth/react*** para autenticar al usuario mediante el método de autenticación ***credentials***. Los valores ingresados en los campos de usuario y contraseña se utilizan para realizar la autenticación. Si las credenciales son válidas, el usuario es redirigido a la página de inicio de la aplicación. En caso de que las credenciales sean inválidas, se muestra un mensaje de error correspondiente en la interfaz.

- ***Mensajes de Error***: Si ocurre algún error durante el proceso de inicio de sesión, se muestra un mensaje de error en la interfaz de usuario. Estos mensajes se basan en el estado local ***errorMessage*** y se muestran en la sección de errores del formulario.

- ***Redirección***: Si el usuario ya ha iniciado sesión y el estado de autenticación (***status***) es *authenticated*, se redirige automáticamente al usuario a la página de inicio de la aplicación sin mostrar el formulario de inicio de sesión.

La interacción entre *Prisma* y *NextAuth* es fundamental para el proceso de autenticación. A continuación se presenta un resumen de dicha interacción:

- ***Prisma***: *Prisma* es un *ORM* (*Mapeo Objeto-Relacional*) utilizado para interactuar con la base de datos en la aplicación. Proporciona una capa de abstracción que simplifica y asegura las operaciones de base de datos.

- ***Prisma Client***: Prisma genera automáticamente una biblioteca llamada *Prisma Client* que se utiliza para interactuar con la base de datos en la aplicación. El código importa *Prisma Client* como ***Prisma*** y se utiliza en la función ***authorize*** de *NextAuth* para realizar consultas y verificar las credenciales del usuario durante el inicio de sesión.

- **Función *authorize***: Durante la autenticación, la función ***authorize*** realiza consultas utilizando *Prisma Client* para buscar un usuario con el nombre de usuario proporcionado en el formulario. Además, verifica si el usuario pertenece a uno de los grupos válidos. A continuación, se realiza una verificación de la contraseña utilizando la biblioteca ***bcrypt*** para comparar la contraseña ingresada en el formulario con la contraseña almacenada en la base de datos. Si las credenciales son válidas, el usuario se autentica correctamente y se devuelve como resultado de la función ***authorize***.

- **Eventos y Callbacks de NextAuth**: *NextAuth* proporciona eventos y *callbacks* que se utilizan para realizar acciones adicionales durante el proceso de autenticación. En el código proporcionado, se utiliza el evento ***signIn*** para registrar información de inicio de sesión en un registro de registro y el evento ***signOut*** para registrar información de cierre de sesión. También se utilizan los *callbacks* ***jwt*** y ***session*** para modificar y enriquecer el token y la sesión del usuario, respectivamente.

Esto permite garantizar la seguridad y la correcta gestión de la autenticación en la aplicación, al rellenar el formulario de inicio de sesión *Prisma* y *NextAuth* interactúan para autenticar a los usuarios. La información ingresada por el usuario se valida y se verifican las credenciales utilizando *Prisma Client* y la función ***authorize*** de *NextAuth* lo que permite finalmente si se han ingresado las credenciales correctas un login seguro y correcto.

3.5.2. *Home*

Una vez un usuario ha iniciado sesión correctamente, entra en la página de inicio (Figura 7). En este punto de la aplicación se va a diferenciar entre la función *online* y *offline* de la aplicación. Como se explicó anteriormente, esta aplicación está preparada para poder ser utilizada en cualquier circunstancia de conexión, ya que, la recogida de muestras de *Flora* en muchas ocasiones se realiza en zonas de escasa o nula conexión a internet (Figura 8). Durante

el uso de la aplicación y el registro de muestras no es necesaria la conexión a internet, entonces, la información básica para el registro de muestras y las funcionalidad principal de la aplicación se guardará en una base de datos local que se actualizará a petición del usuario. Entonces habrá partes de la aplicación preparadas para funcionar de forma *online*, *offline* y ambas con el objetivo de que se cumpla completamente la función para la que la aplicación fue diseñada. En esta página se pueden encontrar los siguientes elementos:

- **Mensajes:** En la página de inicio aparecen mensajes informativos para ayudar al usuario a utilizar la aplicación correctamente.
- **Botones superiores:** En la parte superior de la página se pueden encontrar dos botones, uno para acceder al registro de una nueva muestra y otro para acceder al registro de un nuevo parque natural, ambos funcionan con y sin conexión de red.
- **Listado de muestras:** Cada usuario solo podrá ver las muestras en las que aparezca como autor. En el listado de muestras se pueden distinguir dos partes principales, una con información básica (fecha, número de registro y número de especies) de cada muestra y otra con una serie de botones que son accesibles dependiendo el carácter de la muestra y la conexión de red que son:
    - **Visualizar:** El botón de visualización está disponible tanto en las muestras locales como remotas. Una vez pulsado este botón se accede a una página en la que se puede ver una ficha con todos los datos e imágenes de la muestra.
    - **Editar:** El botón de edición solo está disponible para las muestras locales. Una vez pulsado este botón se accede al formulario de registro de las muestras con los datos ya rellenos con la finalidad de hacer algún cambio y actualizar la muestra.
    - **Eliminar:** El botón de eliminación está disponible tanto en las muestras locales como remotas. Una vez pulsado este botón se procede a la eliminación total y permanente de la muestra, por ello, para evitar eliminaciones accidentales, el usuario debe confirmar la eliminación de la muestra.
    - **Sincronizar:** El botón de sincronización solo está disponible para las muestras locales. Una vez pulsado este botón se procede a la sincronización de la muestra que está en la base de datos local con la base de datos remota. Al pulsar este botón hay

una validación del número de registro con todas las muestras *online* que aunque ya se hizo anteriormente como se va a explicar en el formulario se podría dar el caso que otro compañero haya recogido una muestra y la haya sincronizado a la base de datos mientras el usuario estaba sin internet, entonces en ese caso se mostrará un mensaje de aviso y se deberá editar el número de registro.

- **Actualización base de datos local:** El usuario es el encargado de actualizar la base de datos local y debe hacerlo mientras tenga conexión de red por eso es recomendado que lo haga antes de la recogida de muestras y periódicamente siempre y cuando tenga conexión de red. Esta parte de la aplicación es donde se encuentran los datos de cuando fue la ultima actualización y un botón que permite volver a actualizar (solo se muestra cuando haya conexión) (Figura 9).

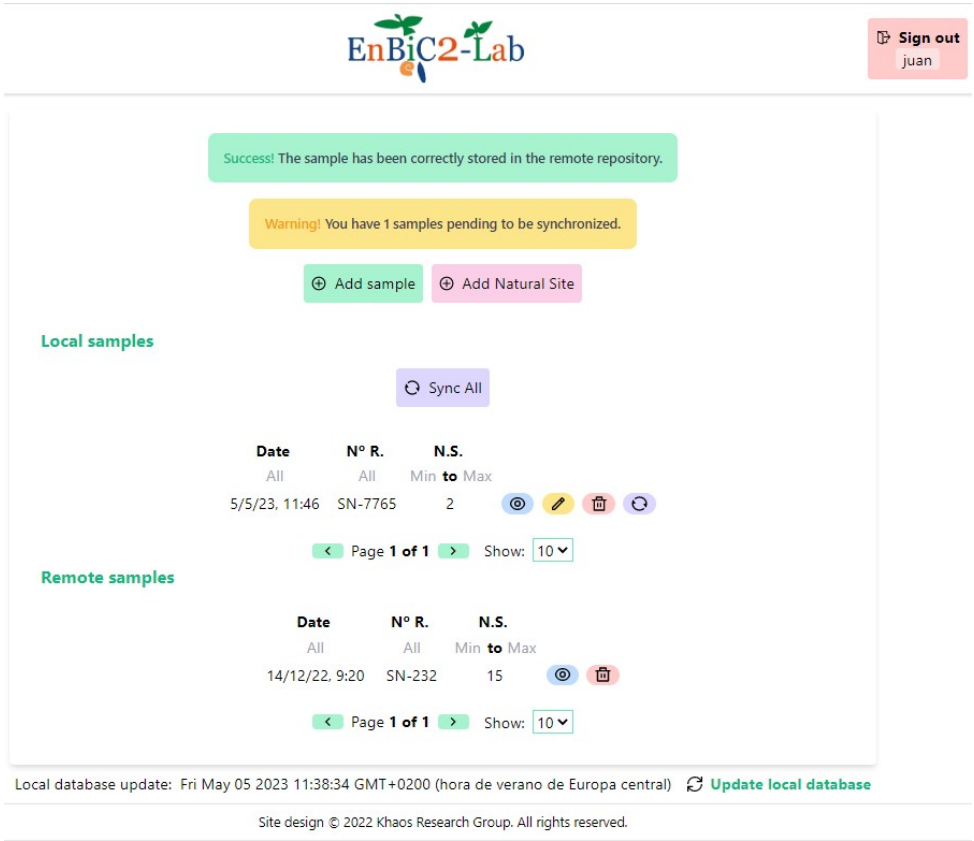

Figura 7: Página de inicio, donde se pueden ver las muestras remotas y locales creadas por el usuario

Como resumen, en esta página encontramos botones de registro de muestras y parques naturales, listado de muestras remotas y locales con sus respectivos botones y zona de actualización de la base de datos local.

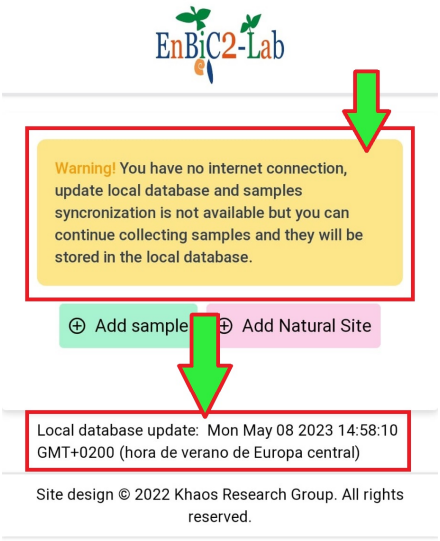

Figura 8: Mensaje modo *offline* y datos última actualización.

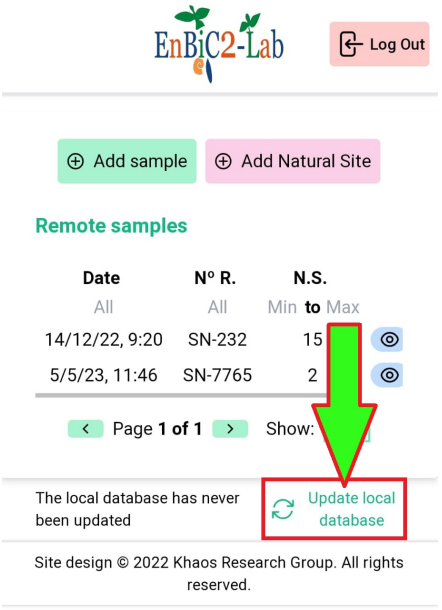

Figura 9: Actualización inicial base de datos local.

3.5.3. *Form*

En esta página se encuentra el formulario de registro de muestras (Figura 10) donde se encuentra la función principal de la aplicación para evitar errores y duplicidad de nombres. En este formulario hay distintos tipos de campos que se van a explicar a continuación.

- **Texto:** Campo simple para introducir un texto con caracteres de cualquier tipo que serán validadas según el dato.
- **Número:** Campo simple para introducir un número con sus validaciones correspondientes.
- **Autocompletado:** Campo de texto con autocompletado.
- **Desplegable:** Campo desplegable con opciones para selección.
- **Booleano:** Botón de selección entre dos valores.
- **Dashboard:** Campo de inserción de imágenes.

Dentro del formulario hay ciertos campos que tienen características especiales y ayudan al usuario a la inserción de datos manteniendo la base de datos unificada, estos campos son:

- **Número de registro:** En este campo se hace una validación con la base de datos local para que no haya dos números de registros iguales.

- **Autores:** En este campo hay un autocompletado y se hace una validación para que solo se puedan meter autores ya registrados. También se establece que el usuario logueado es el usuario por defecto con la posibilidad de añadir mas autores a la muestra. Gracias a esta característica no habrá un autor escrito de diferentes formas y este dato estará unificado. También se garantiza que el usuario pertenece al grupo *Flora* del proyecto.

- **Sitio Natural:** En este campo hay una lista de valores de los sitios naturales que hay en la ultima actualización de la base de datos local, en el caso que el usuario esté en un nuevo sitio natural, puede añadir valores a la lista mediante el botón que se ha explicado anteriormente en la página de inicio. Gracias a esta característica no habrá un sitio natural escrito de diferentes formas y este dato estará unificado.

- **Geolocalización:** Se dispone de un botón para obtener la ubicación actual de una forma muy precisa evitando errores en la inserción de la localización.

- **Comunidad y subcomunidad:** El usuario puede seleccionar si hay o no hay mediante la selección booleana. En el caso que haya, el nombre de la comunidad/subcomunidad es un campo obligatorio de autocompletado en la que el usuario podrá ver los nombres que están vinculados al parque natural seleccionado e introducir otros nuevos. Esta función de autocompletado es posible gracias al preprocesado que se hizo anteriormente y tiene la finalidad de que la base de datos se mantenga unificada.

- **Especies:** El nombre de la especie es un autocompletado en el que los nombres posibles están vinculados al parque natural seleccionado para evitar errores de escritura.

- **Imagen:** En la subida de la imagen existe la posibilidad de cargar una imagen de la galería de fotos y de hacer una foto con la cámara del dispositivo (modo *online*).

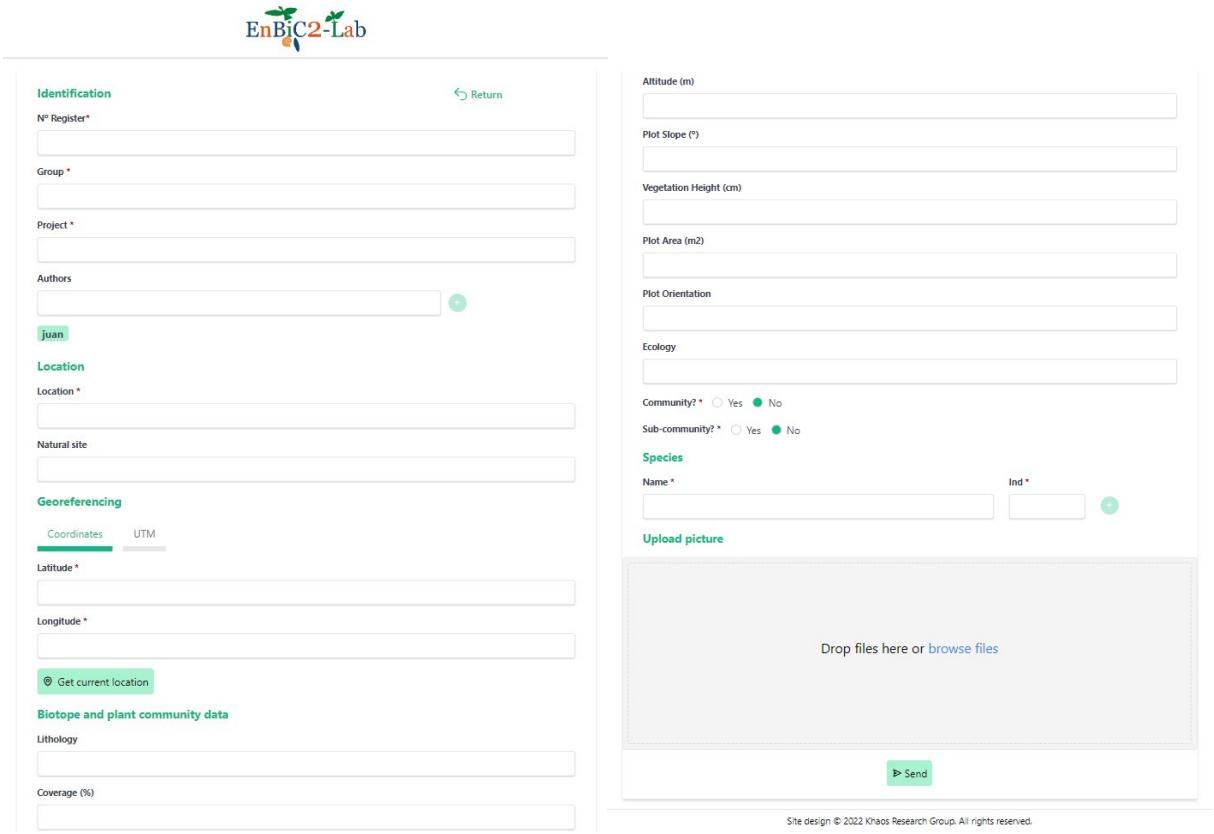

Figura 10: Captura de formulario

En esta página encontramos herramientas de autocompletado que funcionan de forma *offline* y ayudarán a mantener unificada la base de datos. En caso de error en la inserción de algún dato, se mostrará un mensaje de ayuda en los campos afectados.

4

Conclusiones y Líneas Futuras

4.1. Conclusiones

La gestión de datos en proyectos de investigación es un aspecto crítico que puede ser abordado de manera efectiva mediante la aplicación de tecnología y herramientas innovadoras.

En este proyecto se ha visto cómo la recolección de datos mediante fichas en papel y hojas en Excel puede llevar a errores humanos en la transcripción y distintas formas de denominar a las especies/comunidades lo que dificulta la gestión de los datos y su posible utilización para estadísticas, investigaciones o artículos científicos. Además, este proyecto también presentó una oportunidad para desarrollar soluciones innovadoras y eficientes para la gestión de datos.

Con el objetivo de mejorar la gestión de datos en el proyecto *Flora*, primero se ha abordado el problema de los datos que no estaban unificados, usando un algoritmo de NPL realizando labores de preprocesamiento de los datos, lo que ha permitido la implementación de funciones de autocompletado y desplegables en la aplicación, evitando problemas de unificación.

Después de la unificación de los datos se ha desarrollado una aplicación web que permite una gestión más eficiente de los datos, evitando que se vuelvan a producir errores en la recolección de datos y facilitando la geolocalización automáticamente. La aplicación también se ha diseñado para ser accesible tanto en línea como sin conexión, con una interfaz intuitiva y fácil de usar, y permitiendo una mejor comprensión y análisis de los datos recopilados mediante la visualización de los mismos.

En conclusión, este trabajo ha dado una solución efectiva, ante el problema de la gestión de datos de *Flora* proporcionando con la aplicación una herramienta para no volver a caer en los mismos errores optimizando la recogida de datos y su posterior estudio.

4.2. Líneas Futuras

Como parte de las líneas futuras de desarrollo de la aplicación, una mejora potencial sería la integración de datos provenientes de *GBIF*Global Biodiversity Information Facility [16]. El *GBIF* es una organización global que recopila y comparte datos de biodiversidad validados por la comunidad científica. *GBIF* juega un papel fundamental en la recopilación y difusión de información sobre la flora mundial. A través de su amplia base de datos, *GBIF* proporciona acceso a nombres de especies, características taxonómicas y distribución geográfica de plantas, permitiendo a los investigadores y científicos obtener información precisa y actualizada para el estudio y la conservación de la flora en todo el mundo.

La integración con *GBIF* permitiría que la aplicación acceda a una amplia base de datos de nombres de especies, características taxonómicas y distribución geográfica. Al utilizar esta información validada, los usuarios de la aplicación podrían beneficiarse de sugerencias y autocompletado más precisos durante la recolección de datos de *Flora*.

Una posible funcionalidad a implementar sería la capacidad de buscar en la base de datos de *GBIF* al introducir el nombre de una especie, y ofrecer opciones de autocompletado basadas en los nombres científicos o comunes validados por la comunidad científica. Esto ayudaría a prevenir errores de escritura y garantizaría que los datos recolectados se correspondan con las especies reales.

Además, la integración con *GBIF* podría proporcionar información adicional sobre cada especie, como su distribución geográfica, características morfológicas, estado de conservación, referencias bibliográficas, entre otros. Esta información complementaria podría mostrarse en la aplicación como referencia para los usuarios durante la recolección de datos.

La integración de datos de *GBIF* en la aplicación mejoraría la calidad y precisión de los datos recolectados, al tiempo que agilizaría el proceso de recopilación al ofrecer opciones validadas y confiables para el autocompletado. Esto facilitaría la labor de los investigadores y científicos que utilizan la aplicación, brindando una herramienta más completa y útil en el estudio de la flora y la investigación biológica.

Otras posibles líneas futuras podrían ser extrapolar este tipo de aplicaciones a diferentes ámbitos, como por ejemplo una aplicación para que los ciudadanos puedan poner sugerencias sobre diferentes aspectos de la ciudad. Una aplicación fácil de utilizar, con una interfaz

intuitiva y que pueda utilizarse sin problemas con la conexión podría ser la clave para que los ciudadanos participen y colaboren con la gestión de las ciudades. Actualmente los ayuntamientos tienen un buzón de sugerencias, ya sea físico o virtual mediante correo electrónico, pero los ciudadanos no lo utilizamos o solo lo utilizamos en escasas circunstancias porque consideramos que eso nadie lo verá y se pierde el tiempo. Nos sentimos lejos del ayuntamiento y no participamos en él. Con la creación de una aplicación para la colaboración ciudadana no se tendría este problema, los ciudadanos podrían utilizarla fácilmente, hacer posts con sus sugerencias y que todos los demás ciudadanos puedan opinar sobre ella dando un feedback. Esta característica invita a seguir participando y sumar usuarios. La visibilidad que puede dar una aplicación de este tipo es mucho mayor a la que puede dar un buzón de sugerencias al uso porque hace partícipes a todos los ciudadanos y a su vez animaría a los ayuntamientos a arreglar esos problemas dando de nuevo un feedback positivo entre el ayuntamiento y los ciudadanos gracias a la aplicación. Esta aplicación podría tener actualizaciones para incluir diferentes ámbitos en lo que se refiere la participación ciudadana en la ciudad acercando a los diferentes ciudadanos entre ellos y a su vez al ayuntamiento.

Referencias

[1] Raghu Ramakrishnan, Johannes Gehrke y Johannes Gehrke. *Database management systems*. Volumen 3. McGraw-Hill New York, 2003.

[2] Baltasar Cabezudo, FEDERICO CASIMIRO-SORIGUER SOLANAS y ANDRÉS V PÉREZ-LATORRE. "Vascular flora of the Sierra de las Nieves National Park and its surroundings (Andalusia, Spain)". En: *Phytotaxa* 534.1 (2022), páginas 1-111.

[3] Federico Casimiro-Soriguer Solanas et al. "Nuevas citas de interés para la flora vascular terrestre del litoral ibérico del mar de Alborán (Andalucía, España)". En: *Acta Botanica Malacitana* 47 (2022).

[4] Federico Casimiro Soriguer Solanas et al. "Catálogo actualizado de la flora vascular endémica de la Cordillera Bética Occidental (Serranía de Ronda, Andalucía, España): taxones, biogeografía y conservación in situ". En: (2021).

[5] Robinson Ganchala Benítez. "Aplicación móvil recolectora de datos. RGBData-Forms". Tesis doctoral. ETSI_Sistemas_Infor, 2020.

[6] Juansanchezrodriguez1999. *Flora unification*. https://github.com/Juansanchezrodriguez1999/tfg-unification-npl. 2022. URL: https://github.com/Juansanchezrodriguez1999/tfg-unification-npl.

[7] Juansanchezrodriguez1999. *Flora App*. https://github.com/Juansanchezrodriguez1999/tfg-flora-app. 2023. URL: https://github.com/Juansanchezrodriguez1999/tfg-flora-app.

[8] Soledad Altrudi. "Connecting to nature through tech? The case of the iNaturalist app". En: *Convergence* 27.1 (2021), páginas 124-141.

[9] Patrick Mäder et al. "The Flora Incognita app–interactive plant species identification". En: *Methods in Ecology and Evolution* 12.7 (2021), páginas 1335-1342.

[10] David Boho et al. "Flora Capture: a citizen science application for collecting structured plant observations". En: *BMC bioinformatics* 21.1 (2020), páginas 1-11.

[11] AppFinca Inc. *Flora – Green Focus App*. Último acceso el 12/05/2023. 2023.

[12] Thomas Borsch et al. “World Flora Online: Placing taxonomists at the heart of a definitive and comprehensive global resource on the world’s plants”. En: *Taxon* 69.6 (2020), páginas 1311-1341.

[13] Kyle Banker et al. *MongoDB in action: covers MongoDB version 3.0*. Simon y Schuster, 2016.

[14] KR1442 Chowdhary y KR Chowdhary. “Natural language processing”. En: *Fundamentals of artificial intelligence* (2020), páginas 603-649.

[15] Thomas Dean, James Allen y Yiannis Aloimonos. *Artificial intelligence: theory and practice*. Benjamin-Cummings Publishing Co., Inc., 1995.

[16] *GBIF: The Global Biodiversity Information Facility (año) ¿Qué es GBIF?* URL: `https://www.gbif.org/what-is-gbif` (visitado 13-01-2020).

Apéndice A
Manual de Instalación

Para llevar a cabo la instalación correcta hay que seguir los siguientes pasos, es muy importante completarlos todos y seguir el orden para que la aplicación funcione correctamente de forma *offline*. Se diferencian dos partes, una en la que es necesaria la conexión de red y otra en la que no.

- ***Online*** (los primeros pasos hay que hacerlos con conexión a internet)

1. Primero se debe iniciar sesión con un usuario y contraseña correctos (Figura 11).

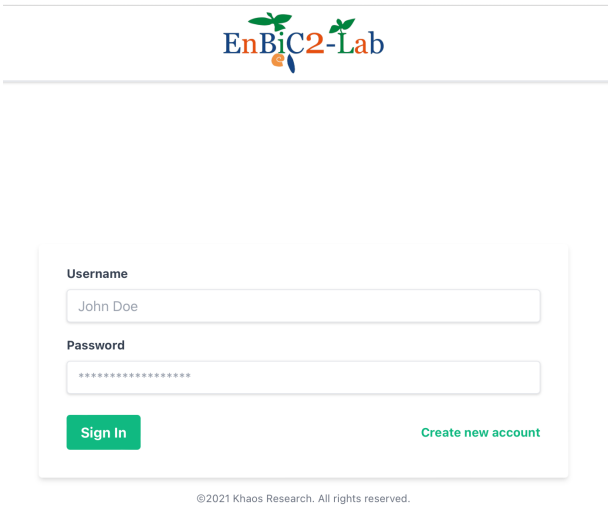

Figura 11: Formulario inicio sesión.

2. Después de debe actualizar la base de datos local para tener los datos necesarios cargados y cumplir la función *offline* de la aplicación. (Se recomienda hacer cada vez que se vaya a recoger muestras (Figura 12).

3. Una vez se ha actualizado la base de datos local se puede ver la fecha/hora en la que fue la última actualización (Figura 13).

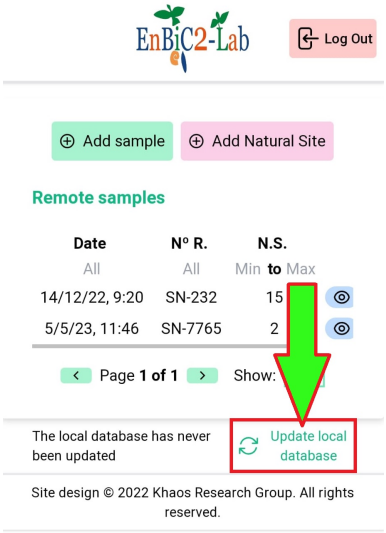

Figura 12: Actualización inicial base de datos local.

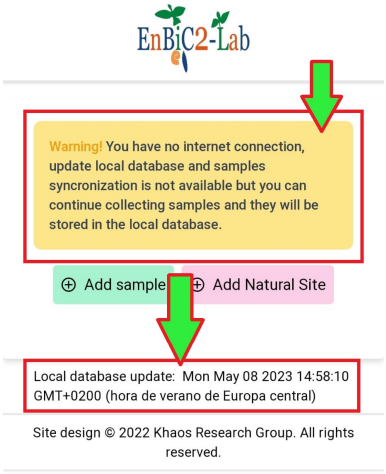

Figura 13: Mensaje modo offline y datos última actualización.

- ***Offline*** (los siguientes pasos hay que hacerlos sin conexión a internet y dependen del sistema operativo y navegador utilizados)
  - Android
    - Google Chrome: Hay que añadir la página a marcadores, primero se pulsa en los tres puntos (Figura 14) y después en la estrella (Figura 15).

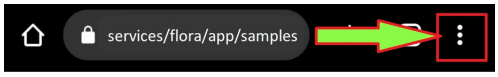

Figura 14: Puntos opciones Google Chrome Android

Figura 15: Añadir a marcadores Google Chrome Android

    - Firefox: Hay que añadir la página a marcadores, primero se pulsa en los tres puntos (Figura 16) y después en la parte de añadir marcador (Figura 17).

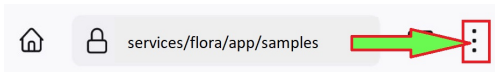

Figura 16: Puntos opciones Firefox Android

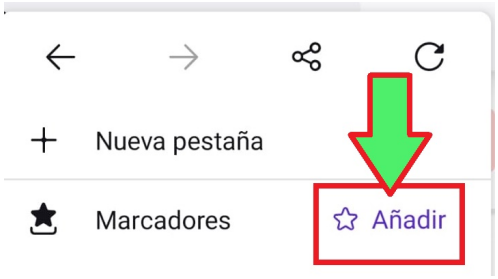

Figura 17: Añadir a marcadores Firefox Android

- IOS
    - Google Chrome: Hay que añadir la página a marcadores, primero se pulsa en los tres puntos y después en añadir a marcadores .
    - Firefox: Hay que añadir la página a marcadores, primero se pulsa en las tres lineas y después en la parte de añadir marcador .
    - Safari: Hay que añadir la página a favoritos, primero se pulsa en el símbolo de compartir, después en añadir a favoritos (Figura 18) y seleccionar guardar (Figura 19).

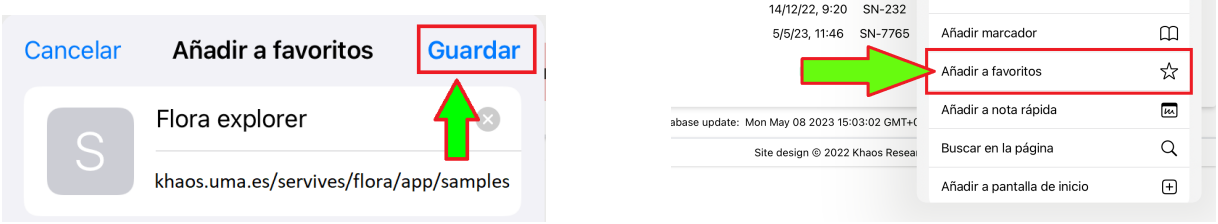

Figura 18: Puntos opciones Firefox Android

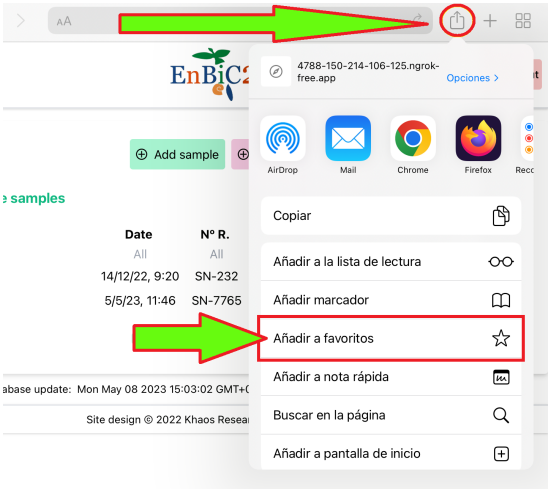

Figura 19: Añadir a marcadores Firefox Android

Cuando ya está la aplicación en favoritos o marcadores, podemos acceder desde el navegador sin volver a iniciar sesión, si en algún momento se llega a perder la sesión repetir los pasos de la instalación. Hay navegadores y dispositivos que ofrecen la posibilidad de instalar la aplicación, en ese caso, la aplicación instalada solo se podrá usar con internet, ya que para iniciar la aplicación instalada se requiere internet en la carga.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga

E.T.S. DE INGENIERÍA INFORMÁTICA