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

PLATAFORMA WEB PARA EL ANÁLISIS Y CLASIFICACIÓN DE ORGANISMOS ACIDÓFILOS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN COMPUTACIÓN

JOAQUÍN ALEXIS JIL BASOALTO

PROFESOR GUÍA:
MAURICIO CERDA VILLABLANCA

MIEMBROS DE LA COMISIÓN:
DAVID S. HOLMES
NANCY HITSCHFELD KAHLER
ANDRÉS MUÑOZ ORDENES

Este trabajo ha sido parcialmente financiado por:
FONDECYT 1181717,
Programa de Apoyo a Centros con Financiamiento Basal AFB170004
Fundación Ciencia & Vida.

SANTIAGO DE CHILE
2020

Resumen

La investigación de microorganismos ha sido de vital importancia durante la última década debido a las diferentes aplicaciones industriales de estos. Por lo mismo múltiples centros de investigación realizan diversos trabajos de investigación relacionadas a ciertos subgrupos de microorganismos. Los resultados de dichos trabajos muchas veces son publicados en forma de bases de datos públicas o utilizando diferentes herramientas web.

El centro de bioinformática y biología genómica de la Fundación Ciencia & Vida se enfoca en el estudio de la evolución de organismos acidófilos (pH < 5) utilizando herramientas de genómica integrativa. Dentro de sus principales áreas de estudio encontramos, identificación de los límites físico-químicos en el crecimiento de estos organismos, reconstrucciones filogenéticas, identificación de genes asociados a la resistencia a acidez y mecanismos de adquisición.

Este trabajo de título consiste en el diseño e implementación de una herramienta web interactiva que utiliza la información recopilada y almacenada por el centro de bioinformática, esta herramienta considera funciones específicas que se ajustan a las necesidades dentro del área de investigación del laboratorio. El uso de este tipo de herramientas es cada vez mas común dentro del área de la bioinformática, ya que es una forma sencilla de dar a conocer los resultados de un centro de investigación de manera complementaria a las publicaciones. Lamentablemente, estas herramientas por lo general se enfocan, en la entrega de información dejando de lado la usabilidad y funcionalidad de las interfaces.

Durante este trabajó de título, se analizó la factibilidad de implementación de las diferentes funcionalidades requeridas por el centro de bioinformática, se estudiaron las diferentes opciones para la arquitectura de la aplicación, modelo de datos y diseño de interfaces para finalmente implementar la aplicación web con las diferentes funcionalidades. Dentro de estas se encuentran la visualización de organismos agrupados por taxonomía, búsqueda y filtros de organismos utilizando diferentes atributos (pH, temperatura óptima de crecimiento, entre otros) y diferentes gráficos dependiendo de los atributos de cada organismo.

Por último, cabe destacar que se realizaron validaciones de funcionalidades e interfaces en conjunto con los miembros del centro de bioinformática, enfocándose principalmente en la usabilidad de la plataforma, este proceso se realizó de manera transversal durante toda la etapa de implementación. Con esto se logró implementar exitosamente la herramienta web, con cada una de las funcionalidades identificadas a lo largo de este proyecto junto con las interfaces asociadas a estas.

A man sometimes devotes his life to a desire which he is not sure will ever be fulfilled. Those who laugh at this folly are, after all, no more than mere spectators of life.

Akutagawa Ryūnosuke

Agradecimientos

Primero que todo, quiero agradecer a mis profesores guías por la confianza y preocupación durante todo este proceso. Agradecer al profesor David Holmes por darme la oportunidad de trabajar en un proyecto en conjunto con el centro de bioinformática además del interés y participación durante toda mi estancia en el laboratorio.

Por otra parte agradecerle al profesor Mauricio Cerda por aceptar ser mi profesor guía, por la disposición y el apoyo durante todo mi trabajo lo cual permitió realizar esta memoria de título.

También quisiera a agradecer a las personas del laboratorio, por la disposición, el apoyo y la buena conversación.

Finalmente, agradecer a mi familia, por soportar y esperar todos los años que estuve en la facultad.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Antecedentes

Los organismos acidófilos (pH óptimo de crecimiento menor a 5) tienen un amplio campo de utilización industrial siendo uno de los más importantes en Chile los procesos de biolixiviación de minerales. Una de los métodos para la recolección de datos y posterior análisis de estos, es la utilización de herramientas web especializadas pertenecientes a diferentes laboratorios y centros de investigación en el mundo. Estas herramientas funcionan como bases de datos públicas y contienen información como pH óptimo de crecimiento, clasificación taxonómica e información genética. Por lo general, una plataforma de este tipo contiene información asociada a un grupo específico de microorganismos por lo que es necesario recopilar los datos utilizando múltiples fuentes.

Otro punto relevante a considerar es el aumento de información pública asociada a microorganismos disponibles en la última década. Desde el 2010 en adelante la cantidad de información almacenada en librerías digitales creció considerablemente. El principal exponente que ilustra lo mencionado anteriormente es la biblioteca nacional de medicina de Estados Unidos, específicamente el *National Center for Biotechnology Information* (NCBI) el cual forma parte de esta.

NCBI ofrece diferentes servicios y herramientas para el análisis de secuencias de ADN, ARN y proteínas. Por otra parte también tiene a disposición diferentes bases de datos y publicaciones enfocadas en el área de la bioinformática, biomedicina, bioquímica y genómica.

En la figura 1.1 podemos notar el crecimiento de secuencias almacenadas en GenBank (base de datos de secuencias de NCBI) y la cantidad de usuarios que acceden a los servicios de NCBI.

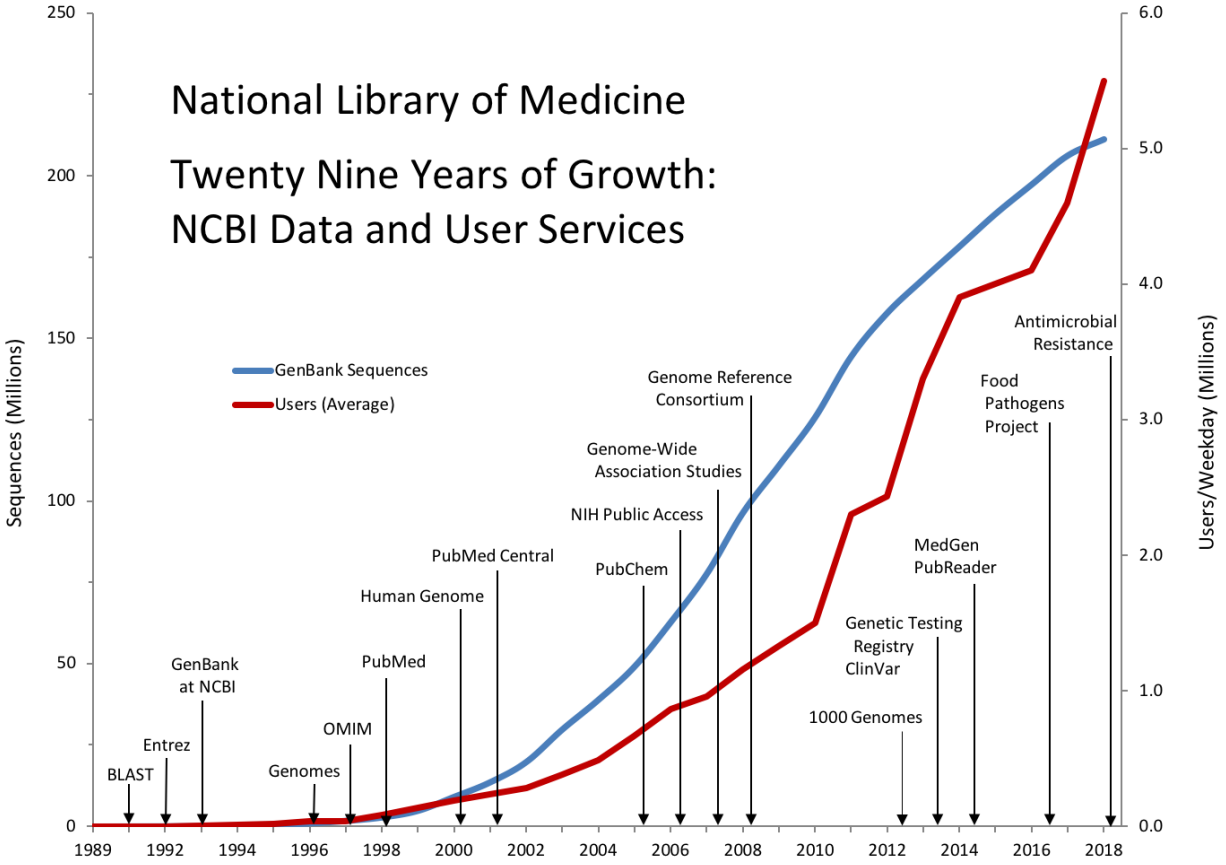

Figura 1.1: Crecimiento de NCBI 1989-2018.[6]

Considerando lo anterior cada vez son más las plataformas web que ofrecen herramientas de análisis en conjunto con bases de datos públicas dentro de áreas mucho más acotadas, especialmente en el área de la bioinformática.

Actualmente existen diferentes plataformas que proveen información de ciertos organismos, cada una desarrollada por los propios centros de investigación, lo cual se debe principalmente al nivel de personalización requerido por cada centro en particular dependiendo de su área de estudio. Por ejemplo, dentro de estas plataformas encontramos HaloWeb [7] del laboratorio DasSarma en la universidad de Maryland (Figura 1.2) la cual agrupa información genética de haloarqueas (arqueas que se encuentran en agua saturada de sal) o SinEx [9] una plataforma desarrollada dentro del mismo laboratorio de la Fundación Ciencia & Vida enfocada al estudio de secuencias genéticas de mamíferos la cual se puede observar en la Figura 1.3.

Dentro de este contexto el centro de bioinformática de la Fundación Ciencia y Vida ha recopilado y analizado información de más de 1000 organismos acidófilos diferentes utilizando bases de datos públicas, publicaciones e información del mismo laboratorio, todo esto con la finalidad de implementar una herramienta web que facilite el acceso a la información especializada dentro y fuera del centro de bioinformática.

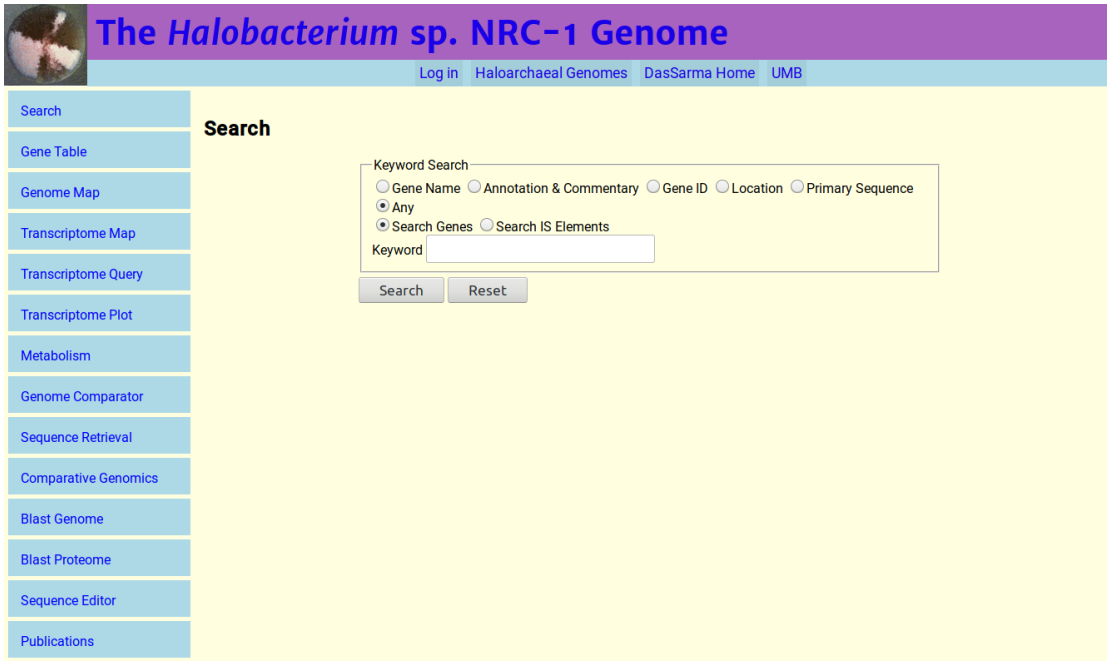

Figura 1.2: HaloWeb.

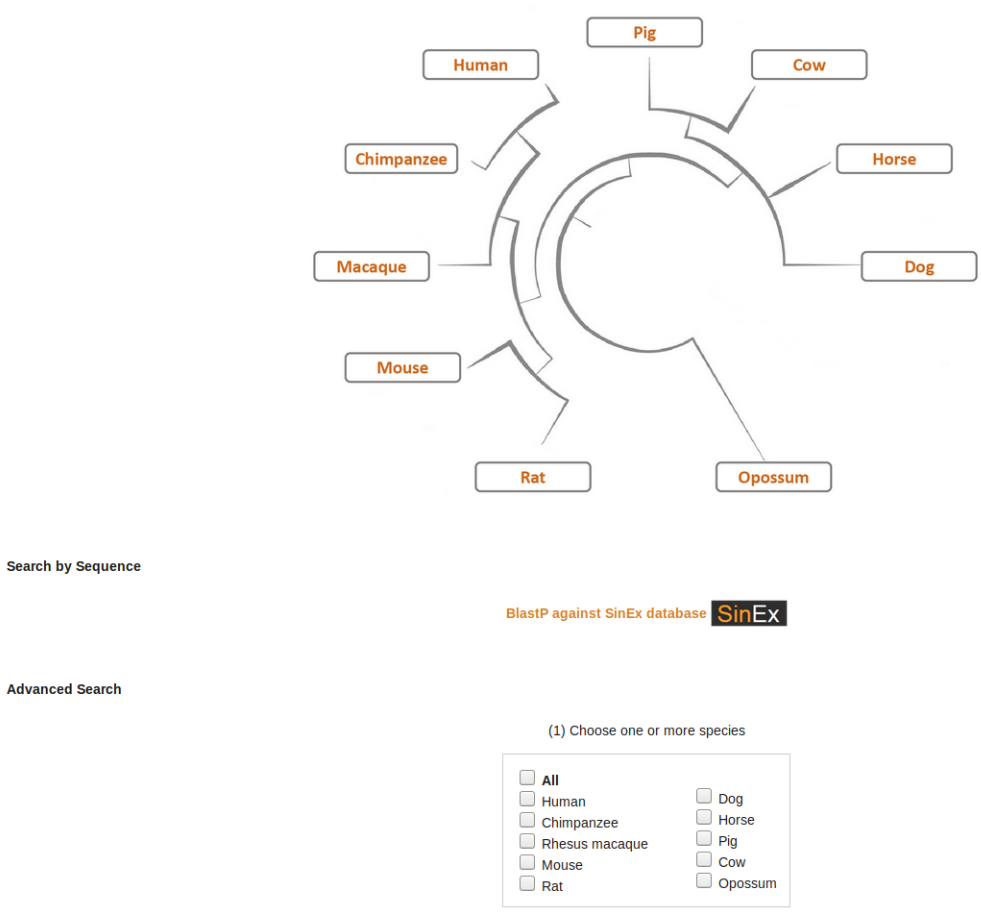

Figura 1.3: SinEx Database.

1.2. Motivación y Objetivos

El centro de bioinformática ha recopilado, estudiado y analizado la información de más de 1000 organismos, cada uno con 100 atributos[1], utilizando diversas fuentes las que incluyen bases de datos públicas como NCBI (National Center for Biotechnology Information) y IMG-JGI (Integrated Microbial Genomes & Microbiomes), además de información propia del laboratorio con la finalidad de analizar y estudiar organismos acidófilos. Por otra parte, el laboratorio además utiliza herramientas externas para realizar el análisis genético y clasificación de estos, dentro de las herramientas más comunes encontramos servicios provistos por NCBI para realizar comparación de secuencias de proteínas, principalmente BLAST[2] en conjunto con herramientas de anotación funcional, que permiten describir un gen.

Todo lo anterior sumado a que gran parte de las investigaciones del centro de bioinformática de la Fundación Ciencia & Vida se enfocan en el estudio de dichos organismos, hacen de vital importancia la implementación de una base de datos que permita unificar la información existente, en conjunto con una plataforma especializada que cumpla con los requerimientos del laboratorio la cual podrá será utilizada por miembros internos e investigadores externos.

Las principales contribuciones de este trabajo están enfocadas en dar a conocer parte de las investigaciones realizadas por el laboratorio de bioinformática, junto con unificar la información de organismos acidófilos de diferentes fuentes con la finalidad de facilitar las investigaciones dentro de este campo.

1.2.1. Objetivo general

Desarrollo de una base de datos y una plataforma web con arquitectura frontend-backend de organismos acidófilos utilizando los datos recopilados por el centro de bioinformática.

1.2.2. Objetivos específicos

1. Diseño e implementación de base de datos utilizando la información recopilada por el laboratorio (la cual se encuentra almacenada en archivos de texto plano).
2. Validación de la base de datos.
3. Implementación de frontend y backend.
4. Diseño e implementación de interfaces intuitivas para los investigadores.
5. Validación de la plataforma web (funcionalidades y usabilidad).

1.3. Metodología

Este trabajo fue realizado en el centro de bioinformática de la Fundación Ciencia & Vida, con el apoyo de sus investigadores. Tomando esto en cuenta se utilizaron las siguientes

[1] Información obtenida hasta junio del 2019
[2] Basic Local Alignment Search Tool, National Center for Biotechnology Information

metodologías durante los procesos de diseño y desarrollo de la herramienta web.

1. Análisis y comparación de arquitecturas y tecnologías a utilizar.
2. Metodología de desarrollo incremental e iterativa en conjunto con miembros del laboratorio con la finalidad de validar los requerimientos de la herramienta y el estado de la solución durante todo el proceso.
3. Validación de funcionalidades, interfaces de usuario y usabilidad.

1.4. Contenido de la memoria

En los capítulos siguientes se describe todo el proceso de concepción y ejecución de esta memoria. En el capítulo 2 se explica el estado actual de otras soluciones existentes y se describe la solución, propuesta. En el capítulo 3 se muestra el análisis y diseño de la solución lo cual incluye identificación de historias de usuarios y casos de usos, los que se utilizan posteriormente para la implementación de la plataforma propiamente tal.

Por otra parte el capitulo 4 se enfoca en la implementación de la plataforma describiendo como se desarrollaron cada una de las partes dentro de esta (modelo de datos, *frontend* y *backend*). Finalmente en el capitulo 5 se presentan los diferentes *test* que se realizaron sobre la plataforma en conjunto con las validaciones que se realizaron con la colaboración de los miembros del laboratorio.

Capítulo 2

Identificación del problema

2.1. Situación actual

El centro de bioinformática y genómica de la Fundación Ciencia & Vida se especializa en el estudio de la evolución de organismos acidófilos (pH < 5) utilizando herramientas de genómica integrativa. Dentro de sus principales áreas de estudio encontramos:

- Identificación de los límites físico químicos en el crecimiento de organismos acidófilos.
- Reconstrucciones filogenéticas de filos en Archaea y Bacteria con organismos acidófilos.
- Distribución de metabolismos y fuentes de energía.
- Identificación de genes asociados a la resistencia a acidez y mecanismos de adquisición.

Debido a esto una tarea importante dentro del laboratorio consiste en la recopilación y almacenamiento de información para cada uno de los organismos estudiados. Dicha información se almacena en archivos de texto plano y planillas de cálculo lo cual lleva a los siguientes problemas.

1. Problemas de integridad y consistencia en un mismo archivo.
2. Múltiples plantillas de cálculo con información que no es necesariamente consistente.
3. Lentitud al momento de ordenar o extraer información básica, por ejemplo contar organismos, seleccionar organismo de una misma familia, buscar organismos en un rango de pH entre otros.
4. Dificultad al realizar cálculos y análisis más complejos.

Al igual que el centro de informática, muchos centros de investigación chilenos e internacionales almacenan datos, ya sea en archivos de texto o en bases de datos públicas o privadas, muchas veces implementando sus propias herramientas web. Dentro de estas, podemos encontrar una gran variedad de herramientas las que van desde implementaciones muy básicas con interfaces de baja calidad hasta implementaciones mucho mas complejas que utilizan *frameworks* y arquitecturas modernas tal como se puede apreciar en las imágenes a continuación.

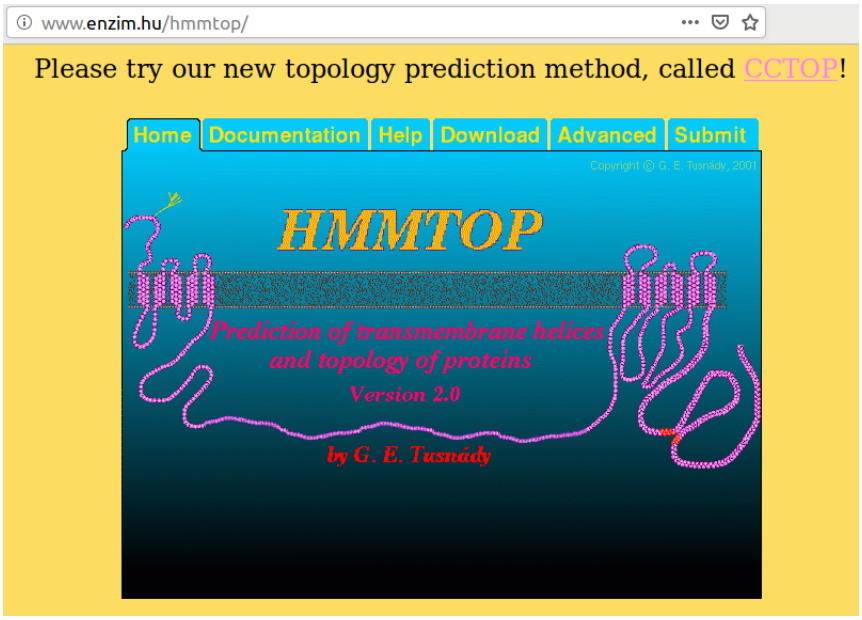

Figura 2.1: hmmtop.

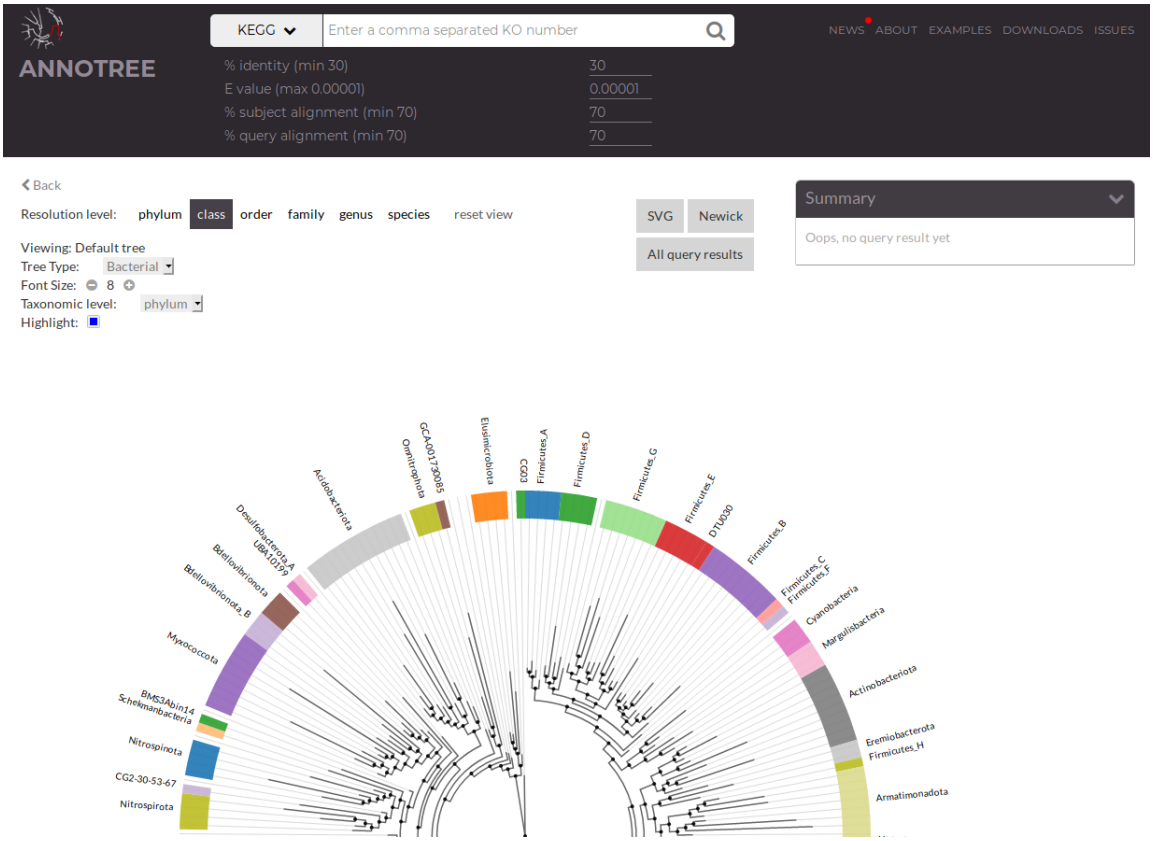

Figura 2.2: AnnoTree.

En la figura 2.1 es posible observar la página web hmmtop[5], una herramienta de predicción de proteínas transmembrana donde se da prioridad a la funcionalidad por sobre la usabilidad y calidad de la interfaz, mientras que en la figura 2.2 se puede observar AnnoTree[10],

una herramienta de visualización de árboles taxonómicos mucho más moderna, con una calidad de interfaces mucho más alta. Dicho esto es importante notar que ambas herramientas ofrecen funciones ampliamente utilizadas por los investigadores dentro del área de la bioinformática a pesar de las diferencias en diseño de interfaces y usabilidad.

Para este trabajo de título uno de los principales focos fue desarrollar e implementar una herramienta que cumpla con las funcionalidades requeridas y al mismo tiempo tenga una alta usabilidad con interfaces intuitivas y agradables para el usuario.

2.2. Solución propuesta

Para solucionar los problemas mencionados se propuso la implementación de una base de datos relacional con la finalidad de estructurar y almacenar los datos recopilados por el laboratorio. Además, junto con la base de datos también se encuentra la implementación de una plataforma web que permita visualizar, navegar comparar y generar gráficos utilizando la información existente para cada uno de los organismos.

En los siguientes capítulos se describe el diseño e implementación de la plataforma web de visualización de organismos, la cual busca solucionar los problemas mencionados con anterioridad junto con dar conocer los resultados obtenidos y los datos recopilados por el centro de bioinformática.

Capítulo 3

Análisis y diseño de la solución

3.1. Requisitos y funcionalidades

Durante los meses de mayo y junio 2019 se realizaron diversas reuniones con los integrantes del centro de bioinformática, donde se definieron los requisitos y funcionalidades principales de la plataforma web, además se aprovecharon dichas oportunidades para familiarizarse con el área de trabajo, el vocabulario, las herramientas utilizadas y los datos recolectados por los integrantes del centro. Todo lo anterior es de suma importancia considerando que este trabajo de título consiste en un proyecto multidisciplinario donde la comunicación clara entre las diferentes partes es fundamental.

Durante este periodo se utilizaron dos técnicas para identificar los requisitos de la plataforma a desarrollar, historias de usuario y casos de usuario. Por una parte las historias de usuario, son muy efectivas en casos donde los usuarios y los desarrolladores están en constante contacto. Utilizando estas es posible escribir casos de usos más generales que sirven como pilares básicos para la implementación de las diferentes funcionalidades. Por último, utilizando ambos métodos se pueden extraer los diferentes requisitos de software los cuales son validados con los integrantes del centro de bioinformática.

3.1.1. Historias de usuario

Las historias de usuario permiten interactuar de manera sencilla con clientes o usuarios que no tienen conocimientos en el área de desarrollo de software, por lo general son pequeñas descripciones de lo que un usuario esperaría de la plataforma, esto incluye las funcionalidades que esperarían ver y como las utilizarían. Estas descripciones pueden ser básicas o técnicas dependiendo del conocimiento del usuario, pueden ser escrita por una o más personas y pueden ser descompuestas si es que una historia contiene demasiada información, esta flexibilidad es de gran utilidad en contextos multidisciplinarios y permiten la participación activa de los usuarios durante todo el proceso de desarrollo.

Por lo general las historias de usuario siguen la siguiente estructura: Como *<usuario>*, quisiera *<lograr meta/hacer algo>* para/porque *<alguna razón>*.

A continuación se presentan algunos ejemplos de historias de usuario asociadas a investigadores y administradores de la plataforma.

- Como investigador, quiero buscar organismos acidófilos para ver sus características.
- Como investigador, quiero ver los organismos por dominio.
- Como investigador, quiero comparar aliadófilos por temperatura o pH.
- Como administrador, quiero agregar nuevos organismos fácilmente.
- Como administrador, quiero editar organismos existentes fácilmente.

3.1.2. Casos de uso

Durante esta etapa se revisaron y estructuraron los requisitos no transables de la plataforma web utilizando las historias de usuarios recopiladas con los investigadores. De la misma forma se definieron los casos de usos, los cuales son esenciales para contextualizar lo que se desea desarrollar ya que son mucho más técnicas y específicas que las historias de usuarios con esto es posible tener una visión clara de la aplicación y las funcionalidades que se desean desarrollar.

Además se creó un diagrama con los principales casos de uso como se observa en la figura 3.1.

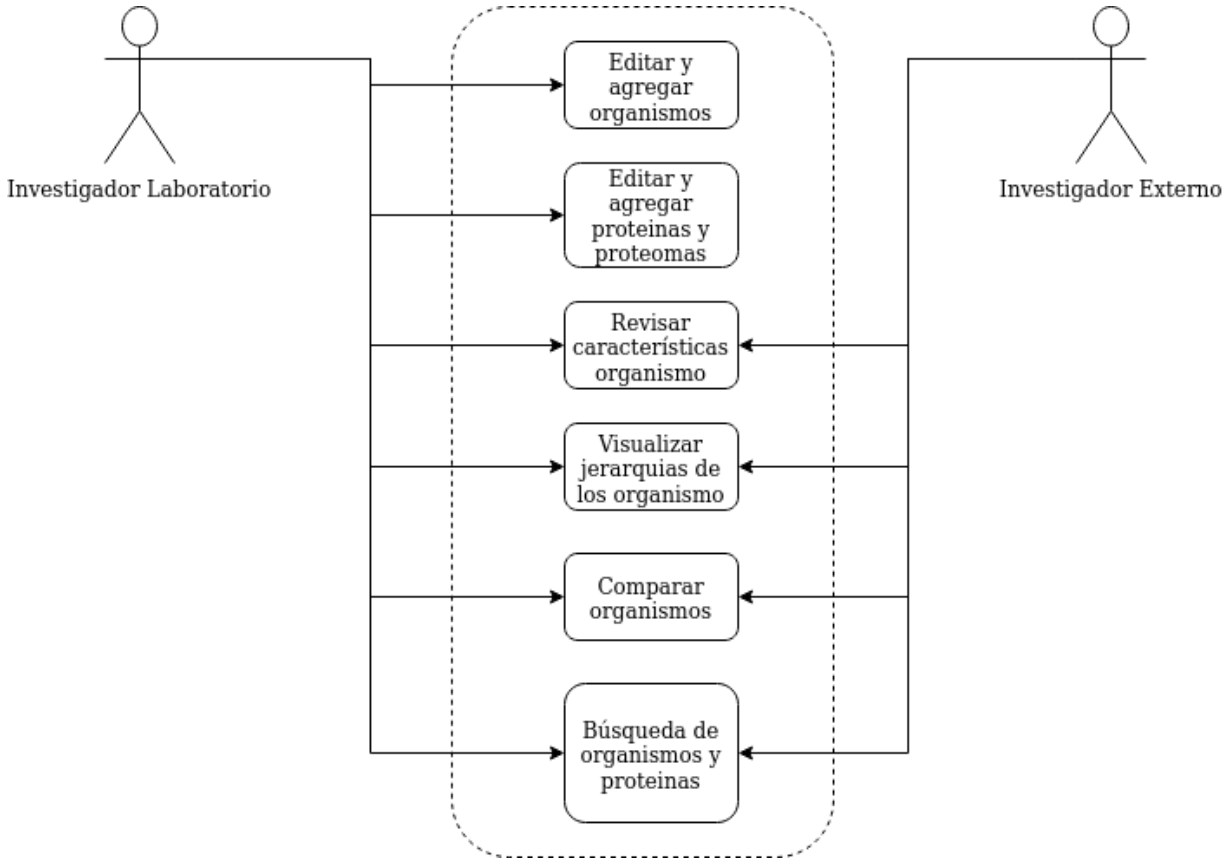

Figura 3.1: Diagrama general de casos de uso.

3.1.3. Funcionalidades

Utilizando las metodologías mencionadas anteriormente se identificaron las siguientes funcionalidades:

1. Implementación de la interfaz básica de navegación.
2. Búsqueda básica y avanzada de organismos.
3. Visualización de los atributos para los diferentes organismos tales como, lugar desde donde se obtuvo la muestra, pH óptimo de crecimiento (o rango de crecimiento), temperatura, requerimiento de O_2, entre otros.
4. Filtros sobre los atributos de los organismos.
5. Visualización de las referencias asociadas al organismo.
6. Visualización de un árbol taxonómico y clasificación taxonómica (desde dominio hasta cepa según exista información disponible) para cada organismo.
7. Visualización del volumen de organismos en las diferentes categorías utilizando gráfico de barras.
8. Visualización mediante gráficos de puntos con ejes personalizables, para los diferentes atributos de los organismos (pH, temperatura, tamaño genoma entre otros).
9. Filtros sobre los elementos en los gráficos.
10. Búsqueda y visualización de proteínas asociadas a los organismo.
11. Filtros sobre proteínas.
12. Posibilidad de descarga de datos, para cada una de las funcionalidades anteriores.

3.2. Diseño de la solución

3.2.1. Arquitectura de la solución

Para el desarrollo de esta herramienta se utilizó un arquitectura de *backend* y *frontend* como servidores independientes. Dentro de las principales ventajas de esta arquitectura encontramos:

1. Flexibilidad al momento de escoger tecnologías ya sea para el *backend* o *frontend*.
2. Desacoplar interacción entre interfaces y modelo de datos.
3. Escalabilidad y portabilidad, agregar nuevas consultas, modificar consultas o migrar por completo el motor de base de datos es mucho más sencillo cuando el *backend* se encuentra desacoplado.
4. Flexibilidad en la implementación de interfaces, la API es independiente de las plataformas o lenguajes.
5. Facilidad al momento de realizar test y validar interfaces.

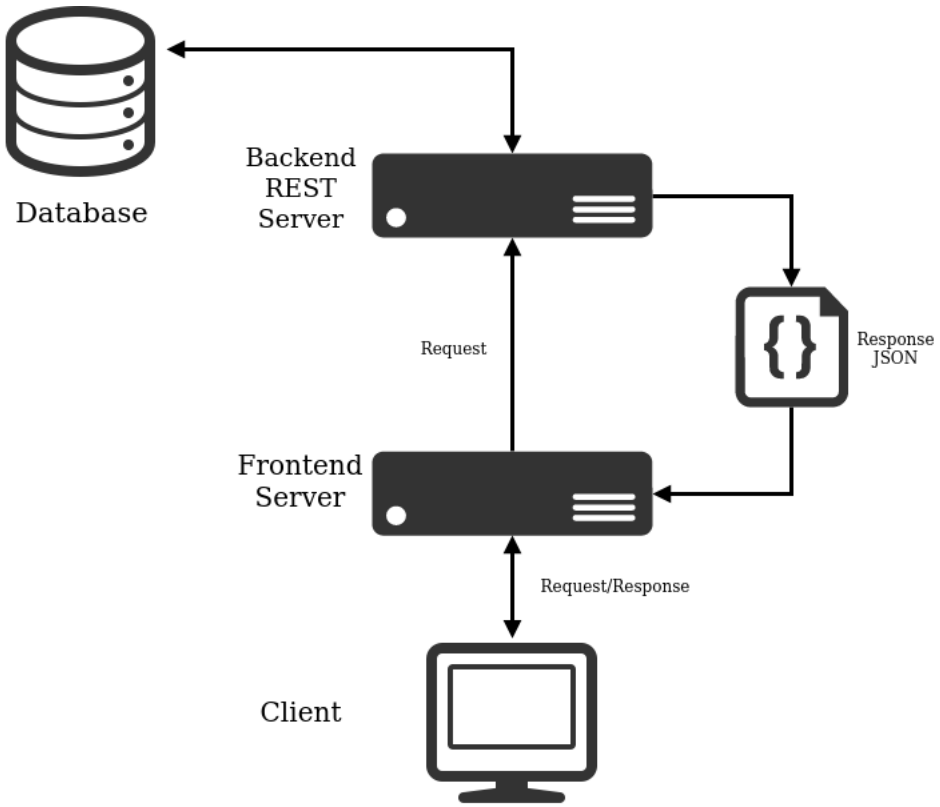

Figura 3.2: Arquitectura simplificada de la solución.

3.2.2. Evaluación de tecnologías a utilizar

Una de las tareas más importantes para el desarrollo de toda aplicación web es la selección de herramientas y tecnologías a utilizar, tomando en cuenta la arquitectura mencionada anteriormente se decidió utilizar un servidor *backend* utilizando Django REST, un framework para creación de web APIs basado en python. Cuando un cliente realiza una *request* al servidor REST, este resuelve la url, identifica qué información se solicita, realiza las consultas pertinentes a la base de datos, serializa dichos datos y finalmente retorna un objeto de tipo JSON el cual es utilizado por el servidor *frontend*. Todo el proceso descrito anterior se puede observar en la figura 3.3.

Dentro de las principales ventajas de un servidor Django REST encontramos:

- Incluye interfaz personalizable para administradores (Django Admin).
- Integración con ORM (*Object-Relational mapping*) para interactuar con la base de datos utilizando modelos.
- Integración con herramientas para serialización y generación de *endpoints* REST.
- Administración flexible de permisos de acceso para los diferentes *endpoints*.
- Generación automática de urls dada una vista. De esta forma cada vez que se agrega un nuevo *endpoint* es posible agregar de manera trivial una nueva url al *router* de Django.

- Soporte de una gran cantidad de librerías y amplia documentación lo cual acelera el desarrollo.

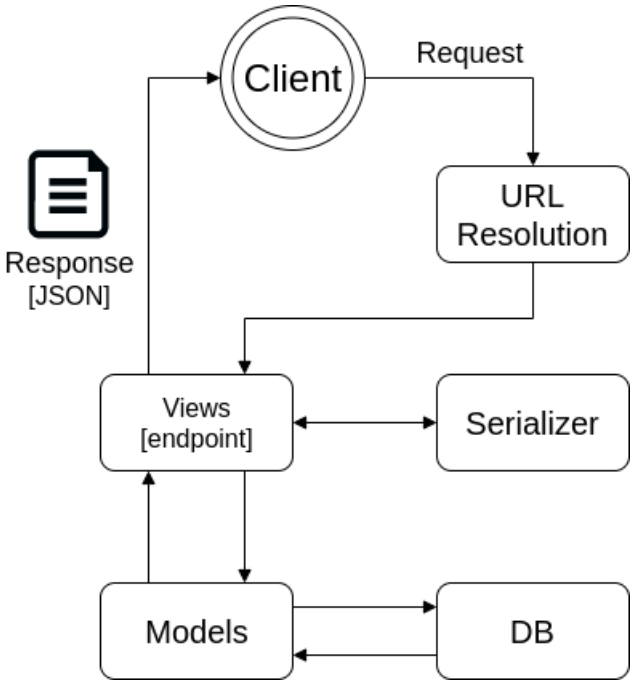

Figura 3.3: Diagrama de flujo para una request en Django REST.

En el caso del servidor *frontend*, se utilizó React v16.8 con componentes funcionales y *hooks* [12] para creación de la plataforma. Además se utilizó Material UI como framework para el diseño de interfaces de usuario en conjunto con otras librerías de componentes para la creación de tablas y gráficos.

React tiene como unidad básica componentes que representan piezas dentro de la interfaz, un conjunto de componentes forma una página la cual está asociada a un componente raíz que representa la aplicación web. Dichos componentes pueden ser estáticos o dinámicos (componentes con estado), en cuyo caso es posible almacenar variables de manera asíncrona las que se actualizan dependiendo de las necesidades del componte, este comportamiento se ve reflejado en actualizaciones dentro de la interfaz visual, por ejemplo filtrar un tabla con datos, agregar filas o borrar columnas. Por otra parte componentes estáticos se utilizan en elementos que no cambian dentro de la interfaz, tales como barras al pie de página, párrafos con texto o tablas estáticas.

Dentro de las principales ventajas de este tipo de estructura encontramos:

- Componentes reutilizables, por lo tanto menor redundancia de código.
- Carga de interfaces en demanda, lo cual implica que no se necesita recargar toda la página cuando uno navega a través de esta, solo se actualizan los componentes pertinentes.

- Desacoplamiento de componentes, la aplicación se mantiene responsiva a pesar de que un componente falle o se quede esperando una actualización, debido a su naturaleza asíncrona.

Todo lo anterior aumenta el rendimiento general de una aplicación que utiliza React.

En el caso de la base de datos escogió el motor PostgreSQL dado los requerimientos de la herramienta a desarrollar junto con la robustez que ofrece este. Además se integró con la herramienta de administración pgAdmin, la cual permite gestionar la base de datos con sus respectivas tablas. Lo anterior se implementó utilizando un contenedor Docker para facilitar el *deployment* de la base de datos, independiente de la máquina utilizada durante el desarrollo de la memoria.

Por último destacar que se utilizó un repositorio GitLab como herramienta de control de versiones durante todo el proceso de desarrollo.

3.2.3. Diseño del modelo de datos

Luego de analizar las características y requisitos de la herramienta, dado que esta herramienta solo lee datos para luego visualizarlos y nunca escribe o actualiza datos a través de la interfaz web (sólo es posible actualizar datos desde una interfaz de administrador o accediendo directamente a la base de datos), se diseñó un modelo similar a los utilizados generalmente en herramientas tipo $Data\ Warehouse$. Estos modelos no se encuentran completamente normalizados y poseen redundancia de información, pero debido a esta estructura, permiten realizar consultas más rápidas y con pocas operaciones $join$.

Dentro de las tablas no normalizadas encontramos taxonomía y proteoma, en el caso de taxonomía, es fácil notar que un dominio posee, n filos, los cuales a su vez poseen m clases y así sucesivamente para cada atributo. Utilizar una versión normalizada para este tipo de estructura conlleva dos principales problemas o desventajas, por una parte se deben implementar consultas más complejas sobre el modelo, ya que para obtener una taxonomía completa se debe realizar una operación de $join$ por cada categoría taxonómica, el segundo problema que se presenta es el aumento del tiempo total de la consulta cuando se quiere obtener una lista completa de la taxonomía para cada organismo.

Por el contrario utilizando la versión no normalizada de esta tabla ambos problemas mencionados anteriormente se solucionan, ya que las consultas más pesadas solo se realizan sobre una tabla lo cual además, permite escribir consultas mucho más cortas y concisas que entregan los mismos resultados.

A continuación se muestra en la figura 3.4, el diseño final para el modelo de datos.

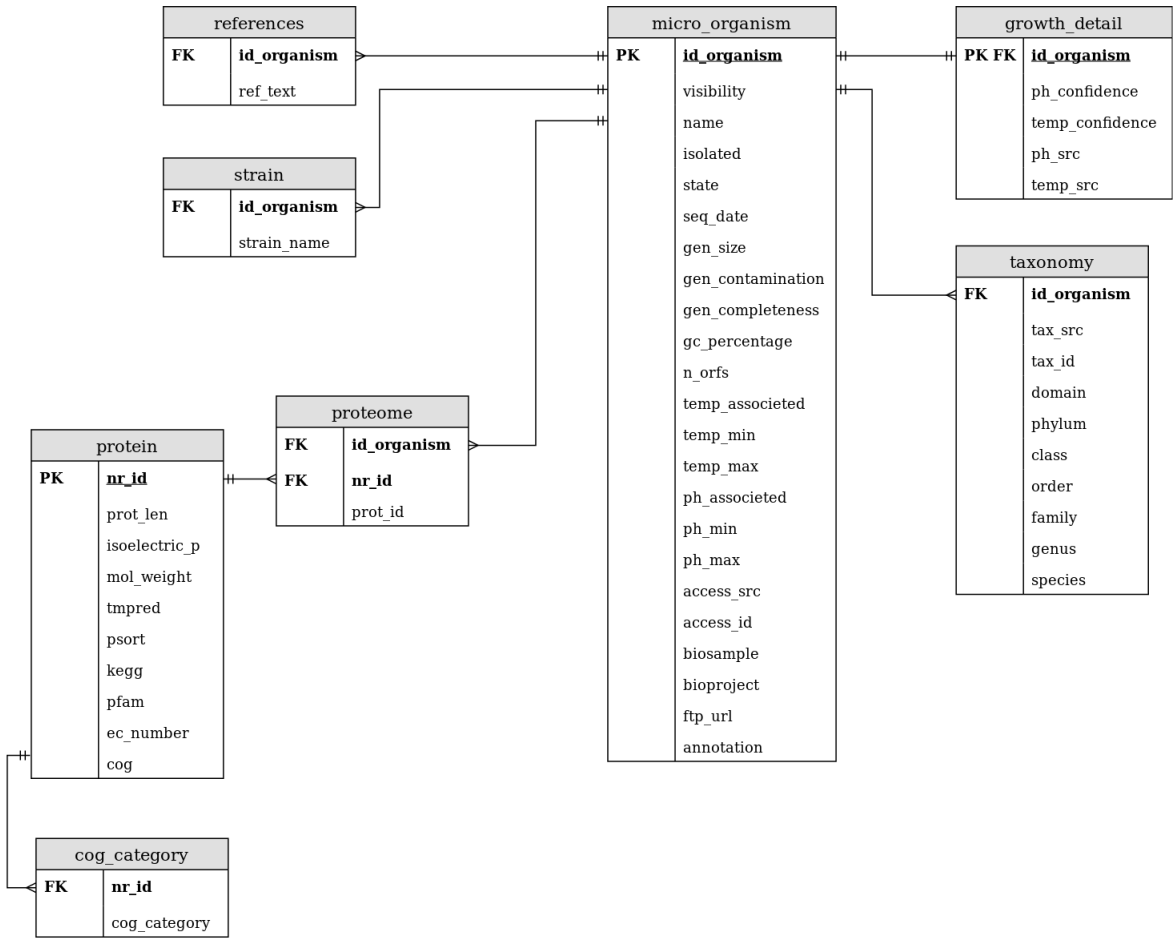

Figura 3.4: Modelo base de datos.

Es importante notar que la tabla *growth_detail* que contiene la "confianza" y el origen de las mediciones realizadas para cada organismo (en este caso de pH y temperatura), se encuentran en una tabla independiente, a pesar de estar en una relación uno es a uno con la tabla principal de organismos. Esta decisión se tomó considerando como prioridad la legibilidad y modularidad del modelo. La confianza no es un atributo intrínseco para cada organismo, si no un valor arbitrario dado por los investigadores, el cual se encuentra asociado a mediciones tales como pH de crecimiento, temperatura óptima, temperatura mínima, temperatura máxima y al nivel de "confianza" de las fuentes de información de donde se recopilaron los datos.

Otra característica importante que se puede observar, es la utilización de un identificador no redundante como llave primaria para las proteínas (*nr_id*), esto se debe, a que dos proteínas iguales pueden tener diferente identificador dependiendo de donde se extrae la información asociada a esta.

3.2.4. Diseño de interfaces y herramientas de visualización

Como se mencionó anteriormente se decidió utilizar un servidor *frontend* con React completamente independiente el cual consume datos de una API externa, esto implica que el diseño de la plataforma web debe ser, en su mayoría, del tipo *single page application* (SPA),

manteniendo los principios de diseño de React.

Esto implica la implementación de componentes como estructura básica al momento de crear la aplicación los cuales pueden cambiar o modificarse dependiendo de la estructura de la aplicación. Componentes como la barra de navegación y pie de página se mantienen constantes en las diferentes vistas mientras que componentes como tablas y gráficos son dinámicos y se actualizan dependiendo de la información que reciben desde el *backend*, en capítulos posteriores se describirán las ventajas que posee este tipo de implementación.

El principal enfoque de estos diseños es la usabilidad, esto se debe, a que a pesar de que existen gran variedad de herramientas web desarrolladas por diferentes laboratorios y centros de investigación, muy pocas cumplen con requerimientos básicos de funcionalidad ya que por lo general, estas se enfocan en entregar sólo las funcionalidades básicas necesarias para operar la herramienta ya sea web o local. Tomando esto en cuenta se diseñaron diferentes interfaces básicas en conjunto con miembros del centro de bioinformática.

Para el diseño inicial de interfaces se utilizó la herramienta Balsamiq Cloud[2], que permite generar bocetos semi funcionales de las interfaces los que posteriormente se validaron y modificaron dependiendo de los problemas y necesidades identificados. En las figuras 3.5 y 3.6 se pueden observar las maquetas creadas para las funcionalidades principales, además tomando en cuenta que durante todo el desarrollo de este trabajo de título se contó con el apoyo de los integrantes del laboratorio, muchas de las modificaciones se realizaron posteriormente, directamente en las interfaces luego de ser testeadas y validadas personalmente con otros miembros del centro.

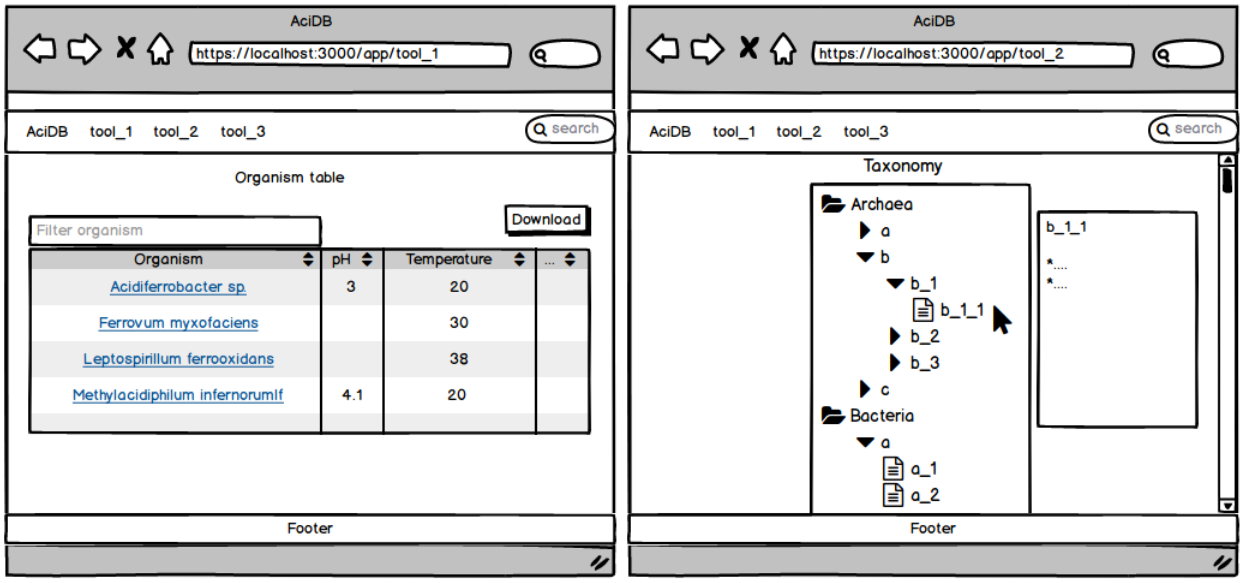

Figura 3.5: Tabla organismo - Navegación taxonómica.

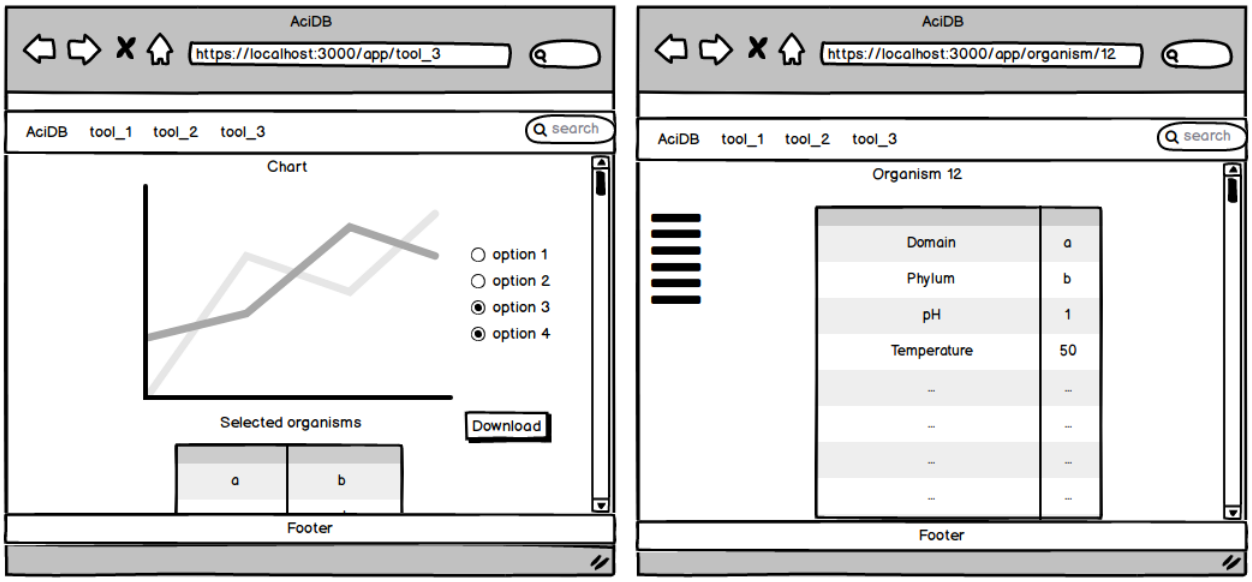

Figura 3.6: Visualización de gráficos - Detalle de organismos.

Capítulo 4

Implementación

En esta sección se describe el proceso de implementación de la herramienta web, el cual incluye la creación de la base de datos, implementación de *backend* e implementación de *frontend* con las respectivas interfaces.

4.1. Metodología de desarrollo

Como se mencionó anteriormente, este trabajo de título fue realizado en conjunto con integrantes del laboratorio de bioinformática por lo cual durante el diseño y desarrollo se trabajó con una estrategia de desarrollo iterativa e incremental utilizando un tablero Kanban [1] en conjunto con diagramas UML [13] (cuando se consideró pertinente) lo cual facilita la comunicación y validación de los diferentes requisitos.

Las etapas de desarrollo utilizadas se puede dividir en tres etapas principales:

1. Creación modelo de datos, extracción, transformación y carga de datos.
2. Generación API REST (*backend*).
3. Diseño e implementación interfaces (*frontend*).

Para la creación de APIs e implementación de interfaces nuevamente destaca la utilización de una metodología iterativa con lo cual es posible volver a diseñar, corregir detalles y validar el trabajo realizado en conjunto con los integrantes del laboratorio tal como se observa en la figura 4.1.

Cabe destacar que al final de cada uno de estas etapas se realizó un proceso de *testing* de funcionalidades e interfaces.

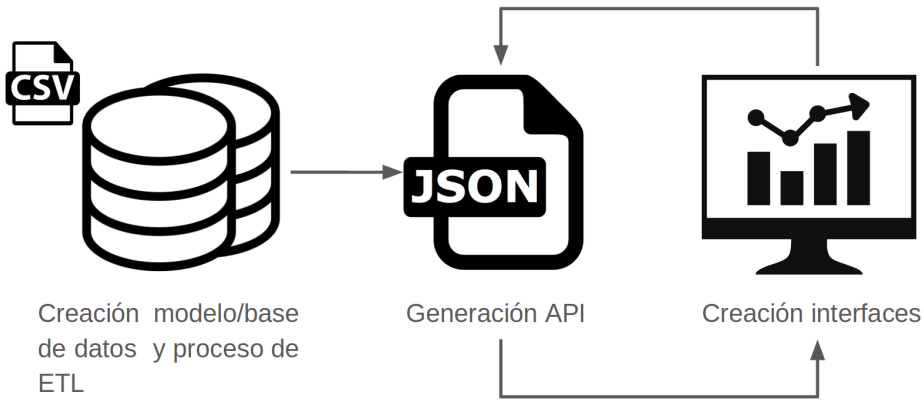

Figura 4.1: Metodología de desarrollo.

4.2. Modelo de datos

Como se mencionó en secciones anteriores, se escogió PostgreSQL como motor de base de datos en conjunto con un servidor Django para la creación y edición del modelo. Adicionalmente, lo anterior se integró con pgAdmin, una herramienta para gestionar y administrar directamente las tablas, estructura y credenciales de la base de datos, esta herramienta también fue utilizada posteriormente para realizar pruebas de estrés sobre la base.

4.2.1. Extracción de datos

Este proceso fue realizado por integrantes del laboratorio, los cuales recopilaron, seleccionaron y descargaron información de organismos acidófilos de diversas fuentes las que incluyen NCBI (National Center for Biotechnology Information), JGI (Joint Genome Institute), GenBank, publicaciones, entre otros. Estos datos se almacenaron en plantillas de cálculo y archivos de texto plano.

Todo el proceso de recopilación y extracción de datos duró aproximadamente un año desde Septiembre 2018 a Agosto 2019.

4.2.2. Limpieza de datos

En conjunto con investigadores del laboratorio se ordenaron y revisaron los diferentes archivos que almacenaban los datos obtenidos con la finalidad de realizar un proceso de limpieza y estructuración previo a la carga de datos. Se eliminaron 62 atributos que no eran necesarios dentro de la base de datos de microorganismos, de la misma forma se eliminaron diferentes organismos irrelevantes para la plataforma según el criterio de los integrantes del laboratorio. Además, se implementaron diferentes *scripts* en Python que permiten limpiar archivos de texto eliminando caracteres especiales y espacios en blancos, corregir el formato de las fechas y eliminar filas, eliminar columnas o modificar la estructura de los datos.

Por otra parte, se normalizaron los atributos que poseen una relación uno es a varios con

los organismos. Dentro de estos están las referencias (publicaciones que hacen referencia a un organismo), que se agregaban como columnas nuevas dentro de los archivos de texto y las cepas, las cuales utilizaban una misma columna para almacenar diferentes nombres de cepas (una cepa puede tener uno o más nombres), tal como se observa en las figuras 4.2 y 4.3 respectivamente.

id_organism	ref_1	ref_2	ref_3
0	paper_01	paper_02	
1	paper_11		
2	paper_21	paper_22	paper_23

(a) Estructura original

id_organism	ref
0	paper_01
0	paper_02
1	paper_11
2	paper_21
2	paper_22
2	paper_23

(b) Estructura propuesta

Figura 4.2: Normalización referencias.

id_organism	name_strain
0	name_01;name_02
1	name_11

(a) Estructura original

id_organism	name_strain
0	name_01
0	name_02
1	name_11

(b) Estructura propuesta

Figura 4.3: Normalización cepas

Además de la normalización muchos datos asociados al proteoma debieron ser preprocesados, como por ejemplo la creación de una lista de proteínas no redundantes utilizando la lista completa de proteínas. Estos procesos fueron realizados por integrantes del laboratorio de bioinformática.

Finalmente se almacenaron los datos limpios y normalizados en diferentes archivos CSV los cuales representan cada una de las tablas que fueron cargadas posteriormente en la base de datos.

4.2.3. Creación del modelo y carga de datos

Durante esta etapa se implementó el modelo diseñado previamente utilizando modelos de Django, además se indexaron las tablas asociadas a proteínas y proteomas tomando en cuenta su gran tamaño, por último se cargaron los datos normalizados utilizando *scripts* de carga masiva. Esta etapa está altamente relacionada con la implementación del servidor *backend* lo cual se discute en la siguiente sección. Los *scripts* de carga masiva se pueden revisar en el anexo A.1.

4.3. Backend

Para el servidor backend se implementó una API REST con las características que se describen a continuación.

4.3.1. Funcionalidades y características principales

El desarrollo del servidor *backend* se enfocó principalmente en la creación del modelo de datos e implementación de la API REST. Una de las principales ventajas que ofrece Django es la implementación de modelos que permiten crear o modificar estructuras dentro de la base de datos además sirve como una interfaz de comunicación que no depende de consultas SQL simplificando la forma en que se accede a los datos, esto a su vez añade una nueva capa de seguridad a la aplicación, ya que Django se encarga de validar y sanitizar entradas y consultas.

En el fragmento de código 4.1 se puede observar la estructura de los modelos para organismos y referencias, cada clase representa una tablas con su estructura dentro de la base de datos, es importante notar que dentro de los modelos también se definen las relaciones entre tablas tal como se puede ver en el modelo *Reference* el cual se encuentra en una relación uno es a muchos con el modelo *Organism* (un organismo puede tener asociada a una o más referencias).

Una de las características más importantes dentro del modelo de datos es la utilización de eliminación de datos en cascada utilizando llaves foráneas. La eliminación en cascada se uso dentro de cada tabla relacionada a un organismo, de esta forma si eliminamos un organismo de la base de datos, también eliminamos sus referencias, cepas taxonomía y detalle de crecimiento, dicho procedimiento es análogo para las proteínas y sus tablas relacionadas.

Código 4.1: Fragmento modelos Django.

```
from django.db import models
# Modelo organismos
class Organism(models.Model):
    id_organism = models.AutoField(primary_key=True)
    name = models.CharField(max_length=50, default=None, null=True)
    isolated = models.BooleanField(default=None, null=True, blank=True)
    ...
    ...
    ...
# One organism can have one or more references
class Reference(models.Model):
    organism = models.ForeignKey(
        Organism, on_delete=models.CASCADE, related_name='references')
    ref_text = models.CharField(
        max_length=1000, default=None, null=True, blank=True)
```

Además se agregó en las llaves foráneas el atributo *related_name* por un tema de legibilidad, este atributo además es usado cuando uno desea ejecutar operaciones del tipo JOIN entre dos tablas.

Otra de las ventajas que ofrece Django es el sitio de administración que viene ya implementado dentro de este *framework* el cual permite acceder al modelo de datos, editando agregando o borrando elementos dentro de las tablas. Utilizando esto como punto de partida se implementó una nueva interfaz de administración que soporta cargar nueva información a la base de datos utilizando archivos del tipo CSV o TSV sin tener que usar *scripts* de carga masiva o conectarse directamente a PostgreSQL, esta funcionalidad es relevante ya que, en su mayoría, los investigadores necesitan actualizar, borrar o ingresar nueva información a la plataforma web pero, por lo general no poseen conocimientos necesarios para realizarlo sin la ayuda de una interfaz web.

Finalmente se implementaron los *endpoints* para la API REST, dicho proceso se describe detalladamente en la sección siguiente.

4.3.2. API

La plataforma web solo lee y entrega datos procesador por el servidor, debido a esto la API REST implementada solo soporta operaciones del tipo GET (obtención de datos), sin ediciones, creación o eliminación de datos.

El primer paso al momento de crear la API, es la definición de la estructura de las urls, luego se crean las vistas o *viewsets* las que contienen la lógica de las consultas realizadas al modelo de datos y finalmente se serializan los datos obtenidos para ser entregados en formato JSON a través de la API web.

Resolución urls

Dentro del *backend* podemos encontrar cuatro tipos estructuras de url para los *endpoints*:

1. Básica : <dominio>/api/<endpoint>
2. Básica con objeto: <dominio>/api/<endpoint>/<:id_objeto>
3. Filtro: <dominio>/api/<endpoint>/?value=query&value_2=query_2&...
4. Árbol: <dominio>/api/<endpoint>/<value>/<value_2>/<value_3>/...

Implementación endpoints

Toda la lógica de las consultas se implementan como vistas en Django, específicamente como *viewsets* los cuales realizan las consultas respectivas, serializan datos y retornan las respuestas a través de la API. Estos *viewsets* también permiten administrar la estructura de las respuestas (listas de objetos u objetos individuales) y los permisos de lectura/escritura de cada *endpoint*.

Código 4.2: Fragmento viewset Django REST.

```
1 class OrganismViewSet(viewsets.ReadOnlyModelViewSet):
2     queryset = Organism.objects.filter(
3         visibility =1).prefetch_related('strains').order_by('name')
```

```
    serializer_class = SummaryOrganismSerializer

    # Return everything
    @method_decorator(cache_page(60*60))
    def list(self, request):
        serializer = SummaryOrganismSerializer(self.queryset, many=True)
        return Response(serializer.data)

    # Return only one instance
    @method_decorator(cache_page(60*60))
    def retrieve(self, request, *args, **kwargs):
        instance = self.get_object()
        serializer = self.get_serializer(instance)
        return Response(serializer.data)
```

En el fragmento de código 4.2 podemos ver un *viewset* que realiza una consulta sobre el modelo de organismos, filtra los organismos visibles (organismos con atributo *visibility* igual a 1), realiza un JOIN con la tabla de cepas y finalmente los ordena por nombre. Además existen dos métodos que responden a solicitudes *list* que retorna una lista de objetos tal como indica su nombre y *retrieve* que retorna un objeto específico dado un identificador.

Es importante notar que también se implementaron filtros para ciertos *endpoints*, estos filtros actúan sobre las consultas ya existentes y permiten, por ejemplo, buscar organismo que contengan uno o más caracteres, filtrar por rango de pH o filtrar por temperatura.

Serialización datos

Uno de los procesos principales durante la implementación de una API REST es la serialización de datos, este proceso se encarga de transformar objetos a una estructura con un formato claro que pueda ser almacenada o transmitida, en este caso se transforman los resultados de las consultas (objetos de Django) en estructuras JSON las cuales se retornan como resultado a una solicitud a la API.

Django REST ofrece un serializador de modelos ya implementado el cual permite agregar nuevos campos dependiendo de los atributos que se necesiten, modificar el tipo de los atributos modificar atributos existentes entre otras funciones. En el fragmento de código 4.3, es posible observar un serializador de organismos el cual entrega todos los atributos de un organismo con sus respectivos tipos exceptuando el campo booleano *isolated*, el cual se transforma en una *string*, con los valores *yes* o *no*.

Código 4.3: Fragmento serializer Django REST.

```
class OrganismSerializer(serializers.HyperlinkedModelSerializer):
    strains = StrainSerializer(many=True)

    isolated = serializers.SerializerMethodField(read_only=True)

```

```
6     def get_isolated(self, obj):
7         return 'yes' if obj.isolated else 'no'
8
9     class Meta:
10        model = Organism
11        fields = ['id_organism', 'name', 'isolated', 'state', 'seq_date', 'gen_size',
12                  'gen_completeness', 'gen_contamination', 'gc_percentage', 'n_orfs',
13                  ...
14                  ...
```

Además estos serializadores pueden ser utilizados dentro de otros serializadores, por ejemplo, el serializador de cepas (línea 2, fragmento código 4.3) se puede utilizar dentro del serializador de organismos, ya que todo organismo contienen el atributo cepa, permitiendo reutilizar gran parte del código. Con esto es posible generar estructuras complejas con datos anindados dentro de las respuestas REST.

Con los *viewset* y sus respectivos serializadores implementados se crearon los siguientes *endpoints*:

Endpoints para búsqueda

- search_list: Estructura básica, retorna toda la lista de objetos.
- search: Estructura con filtros, retorna lista de uno o más objetos. Los parámetros se entregan directamente en la url, por ejemplo:

```
1 /api/search/?organism_or_strain=a&domain=bacteria&temp_associated=10
```

- protein_search: Estructura con filtros, retorna lista de uno o más objetos.

Endpoints de lectura datos

- organism/ : Estructura básica, retorna lista.
- organism/:id_objeto: Estructura básica, retorna objeto especificado.

Código 4.4: Estructura respuesta API organismo.

```
1     # Respuesta api/organism, lista de objetos
2     [
3         {object 1},
4         {object 2},
5          ...
6     ]
7     # Respuesta api/organism/:id, objeto único
8     {
9         object
10    }
11
```

- organism_detail/:id_objeto: Estructura básica, retorna un objeto.

- taxonomy: Estructura tipo árbol, retorna lista de objetos.

Código 4.5: Respuesta API taxonomía.

```
{
"tree": [
        {
        "category": "domain",
        "current_url": "/api/taxonomy/",
        "total": 376,
        "type": "tree",
        "node": {
            "id_organism": "",
            "name": "Bacteria"
            },
            "next_url": "/api/taxonomy/Bacteria"
        },
        {
         ...
            "name": "Eukarya"
         ...
        }
    ]
}

```

Notar que un cliente puede acceder recursivamente los nodos hijos mientras estos sigan siendo árboles hasta encontrar una hoja la cual hace referencia a un organismo. Esta estructura se acomoda perfectamente a la taxonomía además de entregar respuestas rápidas y livianas ya que la API solo retorna los elementos necesarios y no un bloque con todos los organismos y sus respectivas categorías taxonómicas.

Otra ventaja que encontramos es la legibilidad de la url, un usuario que conozca la taxonomía completa o incompleta de un organismo puede acceder directamente a este *endopint* construyendo su respectiva url en caso de necesitarlo.

Endpoints para generación gráficos

- simple_plot_data: Estructura básica, retorna lista de objetos.

Cabe destacar que además se implementaron variaciones de los *endpoints* anteriores los cuales a pesar de ser públicos no son usados directamente por la plataforma, si no que fueron diseñados para ser utilizados de manera interna por integrantes del laboratorio.

4.3.3. Optimización y caché

Finalmente se realizaron optimizaciones asociadas a las consultas y al caché, esto motivado por dos razones principales, la estructura de la API la cual solo lee o entrega datos y el tamaño de las respuestas, las cuales pueden retornar archivos JSON de gran tamaño.

Se agregó un sistema de caché utilizando *decorators* sobre las funciones que definen un *endpoint*, además se le asigno un tiempo de vida al caché dependiendo del tipo de consulta, esto se puede observar en las líneas 7 y 13 del fragmento de código 4.2. El caché almacena los resultados de las solicitudes basándose en los argumentos de la función (*viewset*) esto implica que se usa caché cuando un cliente solicita dos o más consultas con la misma estructura y argumentos en una url.

El servidor *backend* solo realiza operaciones de lectura, lo cual es un ambiente óptimo para implementar un sistema con caché de larga duración ya que nunca se generan problemas de *data race* por escritura o actualizaciones en la base de datos por parte del usuario. En el caso de escrituras por parte de administradores al momento de agregar o editar nuevos elementos como organismos o proteínas, tampoco generan un problema mayor, ya que dichas actualizaciones son programadas y no debiesen ser frecuentes. Considerando todo lo anterior se utilizó un caché de larga duración para cada uno de los *endpoints*.

También se utilizaron mecanismos de *prefetch* de datos en las consultas tipo JOIN, lo cual mejora considerablemente el rendimiento en consultas que impliquen relaciones uno es a muchos. Por último recordar que todas las tablas asociadas a proteínas o proteoma fueron indexadas tomando en cuenta su gran tamaño, aproximadamente 750 organismos cada uno con al menos 2000 proteínas.

4.4. Frontend

Tal como se decidió en la etapa de análisis y diseño, el servidor *frontend* se implementó como una aplicación en React. A continuación se detalla la estructura de aplicación con sus características y funciones principales.

4.4.1. Páginas

React se utiliza para el desarrollo de aplicaciones del tipo SPA (*single page application*), en este caso la herramienta web contiene tres páginas, la página inicial (página de bienvenida), la aplicación propiamente tal y una página de redirección para errores 404, siendo la página de la aplicación la que contiene todos los elementos y funcionalidades presentes en la plataforma web. Estas páginas se componen de pequeños componentes (fragmentos de la interfaz) que se actualizan dinámicamente generando nuevas páginas, en este punto es importante recalcar que la página se carga solo una vez por lo cual estas nuevas páginas solo se construyen reutilizando, agregando o actualizando componentes ya existentes.

Otra de las características principales que encontramos dentro de estas páginas es el ruteo, ya que a diferencia de otros *frameworks*, las rutas no están asociadas a una página, sino a los componentes presentes en la página, es decir las rutas están asociadas a la combinación de componentes presentes en la página y no a la página propiamente tal.

Por último, exceptuando por la página inicial, la plataforma no posee soporte para dispositivos móviles ni dispositivos con pantallas menores a 720 píxeles de ancho. Esta decisión se tomó considerando el perfil de los usuarios de la plataforma y la dificultad de adaptar ciertas

funcionalidades, especialmente la tabla de organismo y los gráficos, a pesar de esto dada la naturaleza responsiva de React es posible utilizarla en dispositivos con resoluciones más pequeñas, pero esto no asegura que las herramientas de visualización funcionen correctamente.

4.4.2. Componentes

Los componentes son estructuras independientes y autónomas que pueden interactuar con otros elementos o componentes en una página, además dependiendo de las necesidades de la aplicación un componente puede estar formado por un conjuntos de otros componentes. Estos componentes pueden ser clases o funciones en JavaScript las cuales poseen propiedades, atributos y un estado local (objeto que representa partes del componente que pueden cambiar en el tiempo). Por otra parte existen componentes sin estado o estáticos también conocidos como *stateless components*, originalmente para crear un componente con estado era necesario crear una clase mientras que un componente sin estado se implementaba utilizando funciones, pero desde el lanzamiento de React 16.8 es posible crear componentes funcionales con estado utilizando *hooks*.

Los *hooks* son funciones que permiten utilizar estados dentro de componentes funcionales, proporcionándole a estos las mismas características y utilidades que posee un componente basado en clases, además del *hook* de estado existen *hooks* que controlan el contexto del componente, referencias, *callbacks* entre otros.

Para esta plataforma web se utilizaron componentes funcionales con *hooks* lo cual permite crear componentes más simples, con una sintaxis concisa y lógica reusable.

Finalmente una última característica a destacar dentro de los componentes es la existencia de los ciclos de vida, funciones que revisan cuando, cómo y qué elementos se deben actualizar dentro de un componente. A continuación se puede observar el ciclo de vida de componentes funcionales utilizando *hooks*, este diagrama presenta una versión modificada, del diagrama propuesto por Dan Abramov [14] uno de los principales desarrolladores dentro del equipo de React.

Mount hace referencia al momento cuando se cargan por primera vez los componentes mientras que *props* se refiere a las propiedades o argumentos iniciales con los cuales se realiza el *render* del componente. En resumen podemos notar que se produce un *render* al momento de montar o actualizar un componente, además existen funciones encargadas de limpiar los ”efectos secundarios” que se producen luego de las actualizaciones.

4.4.3. Funcionalidades y características principales

En la siguiente sección, se describen la principales funciones e interfaces implementadas en el *frontend*.

Barra de navegación y búsqueda básica de organismos

Se creó un componente barra de navegación (figura 4.5) que permite moverse entre las diferentes funcionalidades y componentes dentro de la aplicación, en este componente se

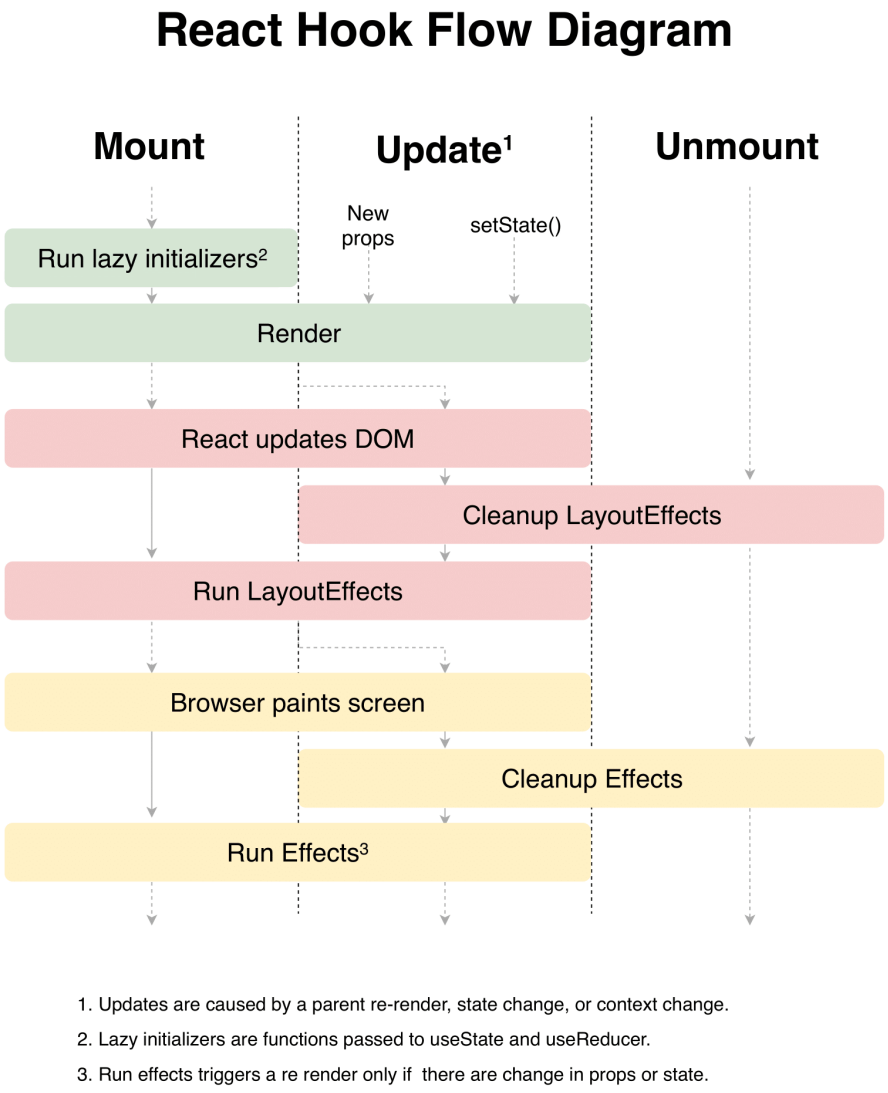

Figura 4.4: Ciclo de vida para componentes funcionales con *hooks* (react >16.8).

implementó una barra de búsqueda la cual puede filtrar organismos por nombre o cepa.

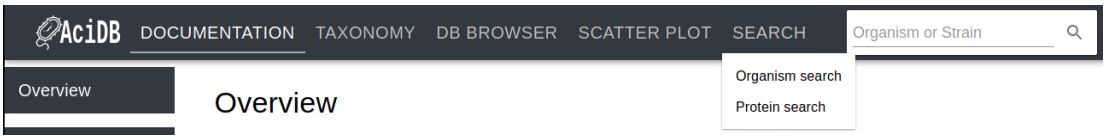

Figura 4.5: Barra de navegación

La barra de búsqueda permite buscar por nombre o cepa dentro de los organismos existentes en la base de datos y redirige los resultados al componente de búsqueda avanzada

retornando una lista de elementos que contengan la palabra completa o un fragmento de esta.

Interfaz detalle de organismo

Este componente se encarga de mostrar todos los datos asociados a un organismo particular, además posee una barra lateral que permite navegar por las diferentes secciones que se encuentran en la misma página.

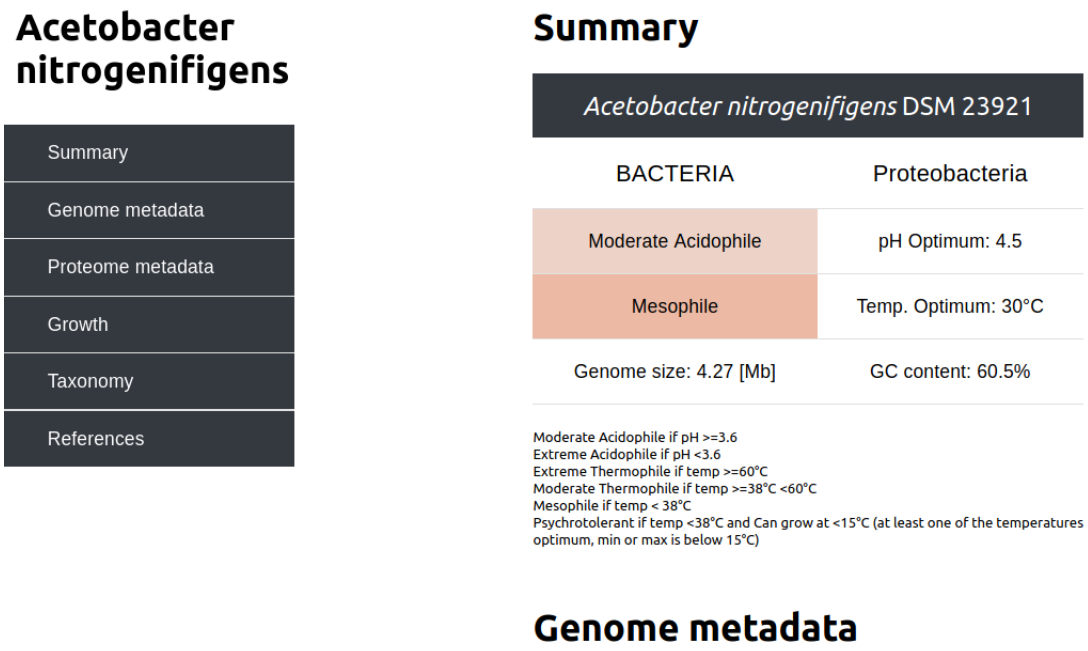

Figura 4.6: Detalle organismos.

Es posible acceder al detalle de organismos a través de la búsqueda avanzada, utilizando la tabla de organismos o a través de los resultados obtenidos en la búsqueda de proteínas.

Búsqueda avanzada de organismos y proteínas

La búsqueda avanzada se define como un formulario dinámico sobre el cual es posible agregar u ocultar filtros, por ejemplo en la figura 4.7 están visibles la búsqueda por organismo/cepa en conjunto con los filtros de taxonomía y rango de crecimiento, mientras que el filtro de meta datos asociados al genoma se encuentra oculto. Por otra parte los resultados de la búsqueda pueden variar entre 0 y el total de elementos existentes en la base de datos, debido esto, dichos resultados se muestran en un componente independiente.

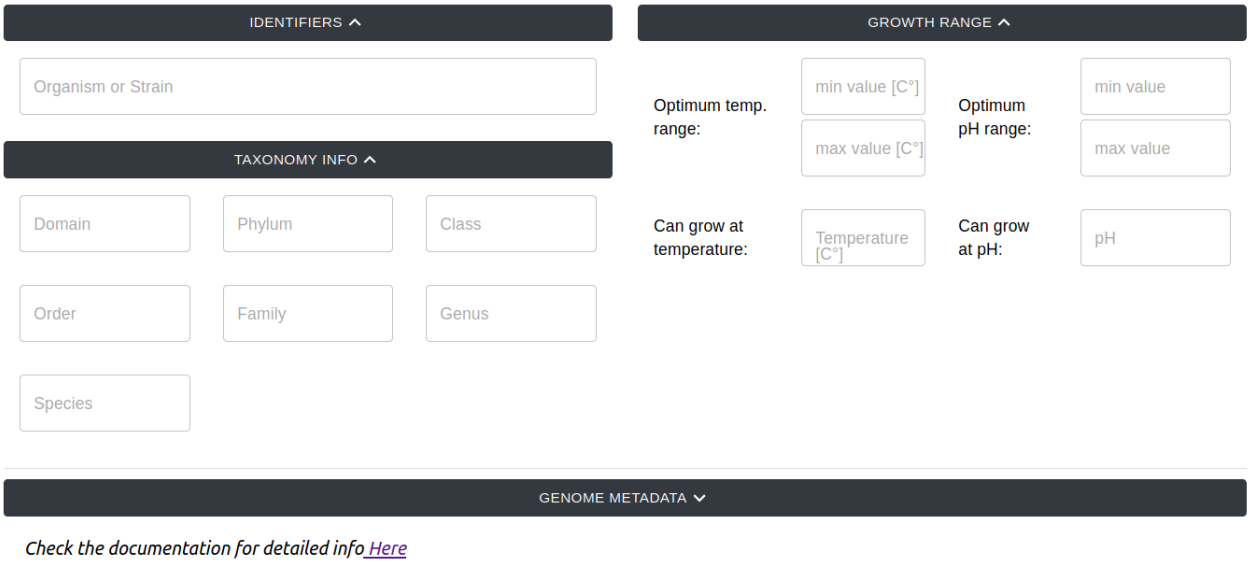

Figura 4.7: Búsqueda avanzada de organismos.

El componente de resultados entrega una tabla de organismos que coinciden con los filtros aplicados, además muestra el total de elementos encontrados e implementa un botón de descarga de datos en formato CSV al igual que todos los componentes que entregan resultados. Este componente se puede observar en la figura 4.8.

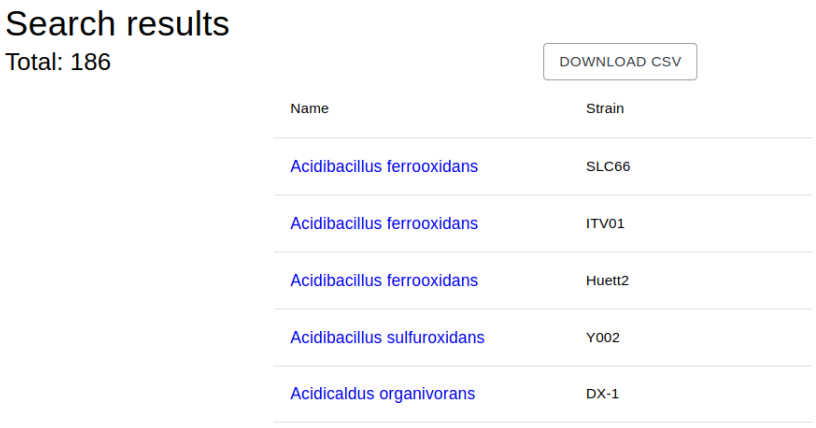

Figura 4.8: Componente resultados búsqueda organismos.

Análogo la búsqueda de organismos se creó una búsqueda avanzada de proteínas, utilizando la misma estructura ya descrita, este permite filtrar elementos utilizando atributos funcionales de las proteínas. Este componente posee la misma estructura lógica y de interfaces que la búsqueda avanzada de organismos, siendo las únicas diferencias el *endpoint* sobre el que se realiza la consulta y el componente de resultados el cual genera páginas de a los más 1000 elementos que muestran las proteínas encontradas con sus respectivos organismos. Además de lo anterior la tabla de resultados soporta componentes de diálogos para visualizar el detalle de una proteína tal como se observa en la sección derecha en la figura 4.9.

Cabe destacar, que también es posible acceder al detalle del organismo a través de los hipervínculos presentes en la tabla de resultados.

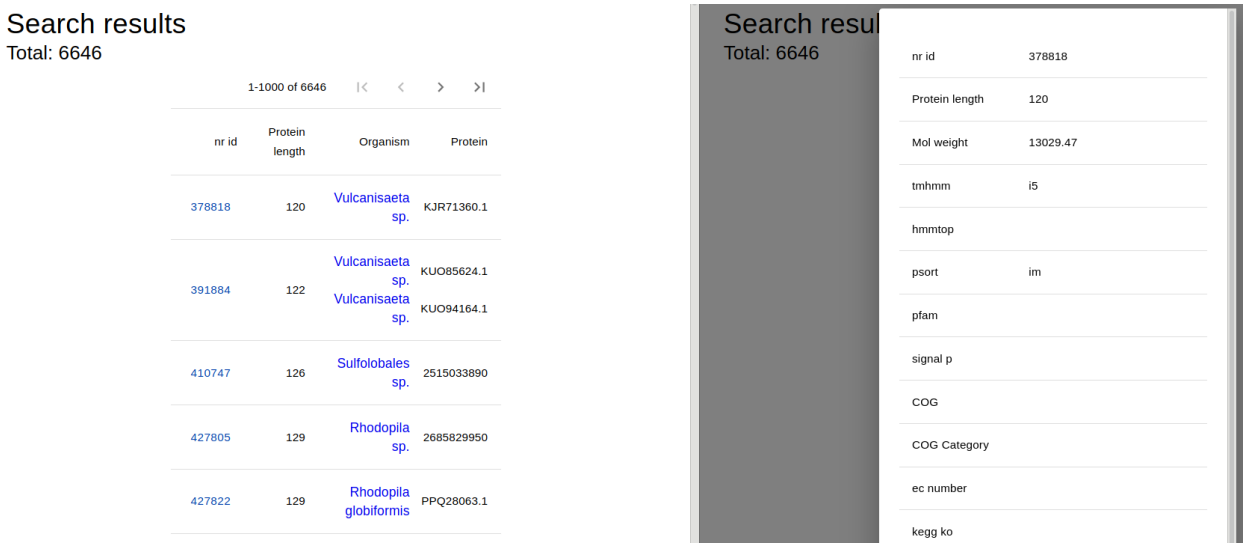

Figura 4.9: Componente resultados búsqueda proteínas y componente diálogo detalle proteína.

Visualización y navegación de datos mediante tablas

Este componente permite visualizar, ordenar y filtrar organismos, para esto se implementó una tabla con sub componentes que pueden modificar la estructura de los elementos visibles ya sean filas o columnas.

Es posible agregar o eliminar columnas usando un selector de columnas disponible en la parte superior de la tabla, también es posible ordenar las filas usando los valores de una columna ya sea en orden ascendente o descendente, para eso basta con hacer *click* en la cabecera de la columna a utilizar como referencia.

Otra de las características más relevantes de este componente son los filtros, en la figura 4.10 se pueden ver como entradas de texto ubicadas bajo los títulos de las columnas, estos filtros son funciones anónimas que se aplican sobre objetos en JavaScript, en este caso sobre la tabla de organismos, y permiten seleccionar elementos dada una o más condiciones. Para esto se implementó una versión primitiva de un evaluador de expresiones el cual recibe como entrada una expresión en forma de texto plano y retorna una versión filtrada de la tabla, todo esto de manera asíncrona lo cual implica que nunca se recarga la página, solo se actualiza el

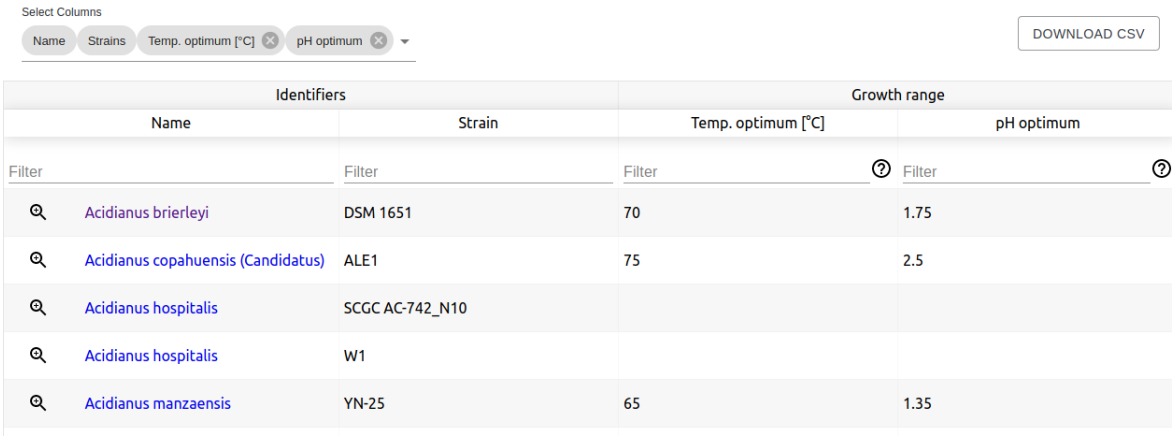

Figura 4.10: Tabla de organismos.

componente.

Existen dos tipos de filtros búsqueda por palabras y filtros con expresiones, la búsqueda por palabra filtra todos los elementos que contengan la palabra ingresada, este filtro no discrimina entre mayúsculas y minúsculas. Por otra parte el filtro de expresiones, se aplica sólo sobre columnas numéricas y permite filtrar por rangos numéricos, este filtro soporta enteros y decimales.

A continuación se observan los tipos de expresiones soportadas.

1. Rangos con intervalos cerrados: $1 - 10$
2. Expresiones: $<= 10$
3. Expresiones compuestas: $<= 6; > 1$

La expresión (1) filtra los elementos entre 1 y 10 con 1 y 10 incluidos, este tipo de expresión soporta una *wildcard* (el carácter $*$), este se puede utilizar al comienzo o al final de la expresión y representa el valor más alto o más bajo dentro del rango, por ejemplo la expresión $1 - *$ filtra todos los valores mayores iguales a 1. La expresión (2) filtra todos los valores menores o iguales a 10, utilizando este tipo de expresión se pueden componer filtros mucho más complejos como se observa en la expresión (3), donde se filtran elementos menores o iguales a 6 y mayores a 1.

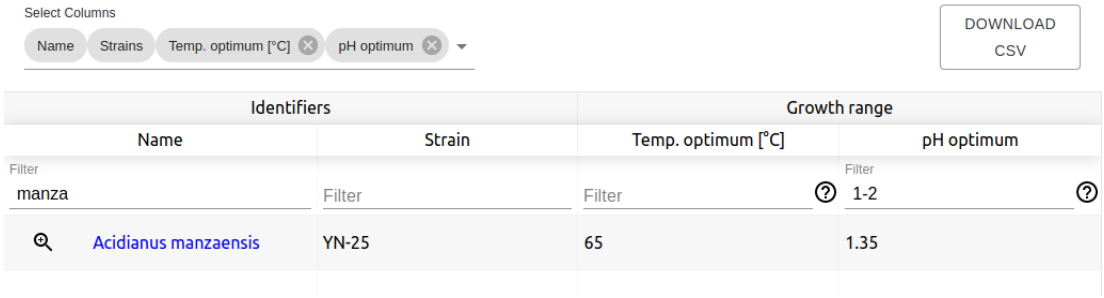

Figura 4.11: Filtros en tabla de organismos.

En la figura 4.11 se puede observar cómo se filtran todos los organismos que contienen la cadena ”manza” y al mismo tiempo tienen un pH óptimo entre 1 y 2.

Finalmente se implementó un componente de paginación el cual sirve para navegar entre las páginas de la tabla, seleccionar una página específica o cambiar el tamaño de estas.

Navegación por taxonomía

El componente de navegación por taxonomía es similar a un componente de navegación de carpetas y archivos. Internamente esta estructura se representa en forma de árbol, donde cada nodo representa un categoría taxonómica (Dominio, Filo, Clase, etc.) mientras que las hojas representan las cepas, que en este caso son organismo particulares.

La interfaz utiliza elementos plegables los cuales contienen las diferentes categorías taxonómicas mientras que las cepas son botones que activan un componente de diálogo con un resumen del organismo.

A la izquierda de la figura 4.12 se puede observar el árbol de navegación, mientras que a la derecha se observa el componente de diálogo que se abre al seleccionar una cepa, este contiene resumen de la información para dicha cepa.

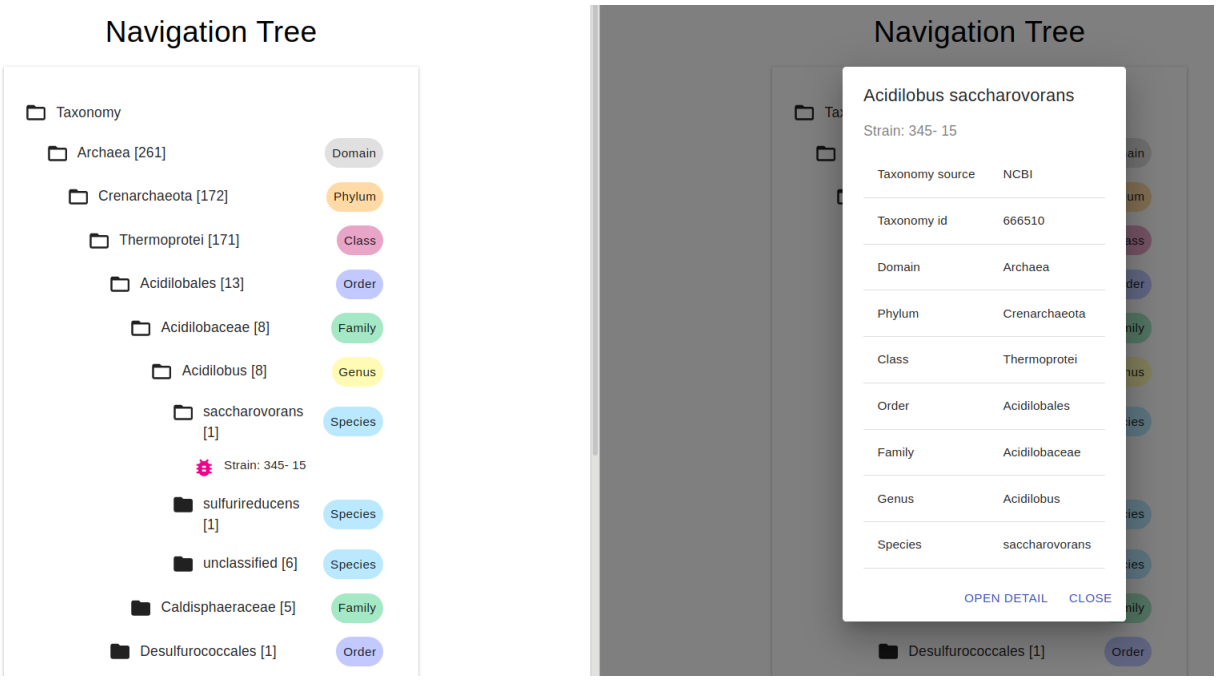

Figura 4.12: Navegación por taxonomía.

Un detalle importante de este componente es que poda ciertas ramas hasta llegar a la hoja, esto ocurre en casos particulares donde ciertos organismos no poseen una o más de una categoría taxonómica. En la tabla 4.1 se observa que el primer organismo no posee clase, orden o familia. El componente de navegación taxonómica no posee nodos vacíos por lo cual busca la primera categoría que contenga algún elemento y la toma como nodo, este proceso se realiza recursivamente para cada hijo dentro del noto hasta encontrar una hoja, de esta forma es posible encontrar nodos que se saltan de ciertas categorías llegando directamente a las cepas u hojas (siempre existe al menos una cepa).

Tabla 4.1: Ejemplo taxonomía organimos.

Phylum	Class	Order	Family	Genus	species
Euryarchaeota				Aciduliprofundum	boonei
Crenarchaeota	Thermoprotei	Sulfolobales	Sulfolobaceae	Acidianus	brierleyi

Gráficos personalizables

Se implementó un gráfico de puntos con ejes personalizables que permite visualizar los organismos como un gráfico de puntos seleccionando diferentes atributos.

Debido a artefactos de medición muchos organismos pueden ocupar las mismas coordenadas, para representar esto, se utilizaron puntos de diferente tamaño y opacidad dependiendo de la cantidad y organismos ubicados en las mismas coordenadas.

Es posible seleccionar uno o más elementos, para un organismo basta hacer *click* sobre un punto en el gráfico, en el caso de que en las mismas coordenadas exista más de un organismo se seleccionan todos los organismos en ese punto. En caso de seleccionar un conjunto de organismos, basta mantener presionado el botón izquierdo del *mouse* y arrastrarlo por la zona que se desee seleccionar, además una zona previamente seleccionada se puede arrastrar o modificar ya sea agregando o quitando organismos (*click* sobre el elemento mientras se presiona la tecla *ctrl*).

En la figura 4.13 se puede observar el gráfico de puntos, con ejes pH óptimo vs. Temperatura óptima y con un bloque de organismo seleccionados en la esquina inferior derecha.

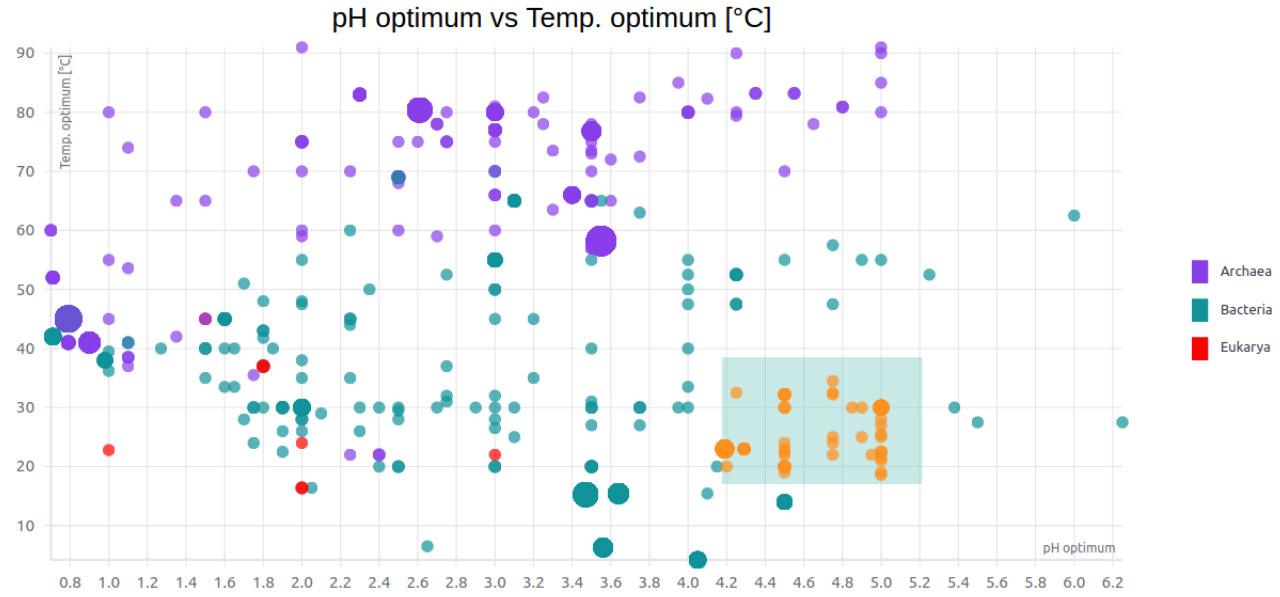

Figura 4.13: Gráfico de organismos.

Dentro de los campos que pueden ser seleccionados como ejes encontramos atributos propios de un organismo como temperatura, pH y tamaño del genoma, por otra parte se pueden filtrar los organismos por Dominio (Arquea, Bacteria o Eukarya) o por variables categóricas (*isolated* y *assembly level*). El componente de selección de atributos es un formulario donde cada entrada forma parte del estado del gráfico, lo cual permite modificar en tiempo real la estructura y elementos visibles.

El componente de selección de atributos se puede observar en la figura 4.14.

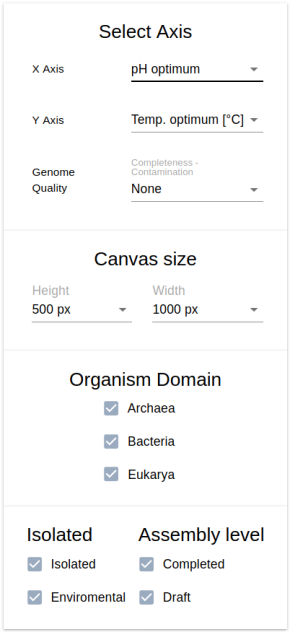

Figura 4.14: Selección atributos gráfico.

En caso de seleccionar uno o más organismos, los resultados se pueden ver en una tabla (figura 4.15) bajo el gráfico de puntos, este componente es similar a los resultados que se obtienen a través de la búsqueda avanzada con la principal diferencia en que las dos últimas columnas se generan dinámicamente dependiendo de los ejes seleccionados.

There are 38 selected points DOWNLOAD CSV

Name	Strain	pH optimum	Temp. optimum [°C]
Acidithiobacillus albertensis	DSM 14366	3.75	30
Acidithiobacillus ferrooxidans	BY0502	3.75	30
Acidithiobacillus sp.	SH	4	30
Granulicella pectinivorans	DSM 21001	4.15	20
Rhodanobacter sp.	FW501-T8	4.29	23

Figura 4.15: Resultados organismos seleccionados.

Descarga elementos para cada una de las funcionalidades

Cada componente de visualización de resultados (tabla de organismos, resultados de búsqueda y selección de organismos en los gráficos) poseen un componente de descarga de datos que le permite al usuario obtener un archivo CSV, con la información obtenida, esta información incluye atributos extras que no necesariamente se muestran en las interfaces ya que este archivo utiliza todos los datos solicitados a través de la API, por ejemplo al descargar los datos del gráfico pH vs. temperatura, el archivo generado, también incluye el tamaño del genoma, contenido GC entre otros.

Es importante volver a destacar que sólo se descargan los datos seleccionados o filtrados por el usuario (organismos seleccionados en el gráfico).

Página de documentación y resumen de funcionalidades

Considerando la gran cantidad de detalles que se muestran en esta herramienta web, se implementó una página que contiene documentación de los datos y funcionalidades (figura 4.16). Con respecto a los datos, esta documentación incluye fuentes de información, criterios de inclusión y otras herramientas utilizadas por integrantes del laboratorio para generar ciertos atributos, en el caso de las funcionalidades, se describen con detalle los casos de uso principales para cada uno de los componentes, además se agregaron pequeños tutoriales en forma de vídeo embebidos dentro de esta página.

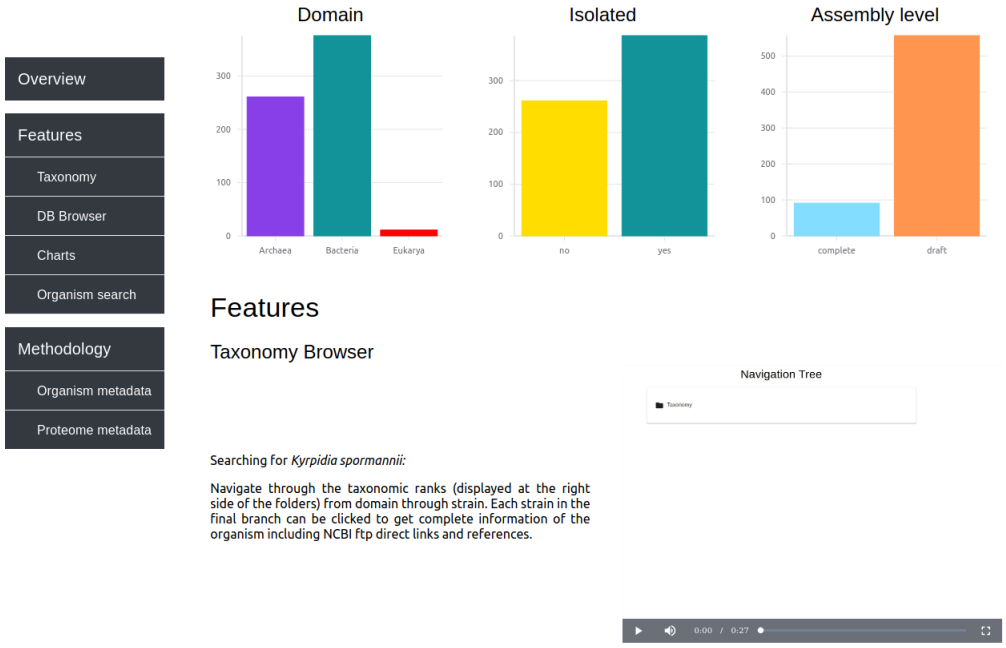

Figura 4.16: *Overview* y documentación funcionalidades.

Esta página de documentación incluye una barra lateral de navegación para moverse a través de la documentación, imágenes de las funcionalidades principales, vídeos mostrando casos de usos y gráficos de barra con estadísticas básicas de la base de datos (total de organismos, organismos por dominio, pH, etc).

Resumen de componentes

Con todos los componentes principales finalizados y conectados con cada uno de los *end-points* previamente implementados en el servidor *backend* se cumplieron cada uno de los requisitos y funcionalidades identificados en la sección 3.1.3.

A continuación, en la tabla 4.2 se muestran como cada uno de los componentes satisfacen dichas funcionalidades.

Tabla 4.2: Componentes asociados a funcionalidades.

Funcionalidad	Componente
Implementación de la interfaz básica de navegación	Barra navegación Barra lateral de navegación
Búsqueda básica y avanzada de organismos	Barra búsqueda básica Búsqueda avanzada organismos
Visualización de atributos	Detalle organismo Tabla navegación organismos Navegación taxonómica Gráficos personalizables
Filtros sobre los atributos de los organismo	Búsqueda avanzada organismos Tabla navegación organismo
Visualización referencias asociadas a organismos	Detalle organismo
Visualización árbol taxonómico y clasificación taxonómica	Navegación taxonómica
Visualización volumen de organismos	Página de documentación
Visualización mediante gráficos	Gráficos personalizables
Filtros sobre elementos en los gráficos	Gráficos personalizables
Búsqueda proteínas	Búsqueda avanzada proteínas
Filtros sobre proteínas	Búsqueda avanzada proteínas
Posibilidad de descarga de datos	Componentes de descarga

4.4.4. Mecanismos de *awareness*

Los mecanismos de *awareness* son elementos visuales en las interfaces que ayudan al usuario a interactuar con la aplicación, estos cumplen la función de dar a conocer características elementales, interacciones con la interfaz o límites de una funcionalidad. Dentro de los mecanismos más usados encontramos, *placeholders*, *tooltips*, entradas de texto con advertencias, animación de carga entre otros

En la figura 4.17, se pueden observar los distintos mecanismos en acción, por una parte tenemos diferentes *placeholders* en las entradas de texto con la finalidad de destacar estos elementos en la interfaz. En la misma figura también es posible ver que existen otros tipos de marcadores en caso de que las entradas sean erróneas (por ejemplo ingresar texto en un filtro numérico). Por última encontramos marcadores con advertencias cuando existen límites para ciertas acciones, esto se puede ver en el caso de la tabla de organismos donde el número máximo de columnas visibles no puede ser mayor a 8.

En esta misma figura también se puede observar un elemento tipo *tooltip*, estos elementos de ayuda aparecen cuando el usuario coloca el *mouse* sobre los signos de interrogación que se muestran en las diferentes interfaces y su principal función es informar al usuario el formato o estructura de las diferentes entradas, por ejemplo mostrar una breve descripción de la estructura de los filtros numéricos en la tabla de organismos.

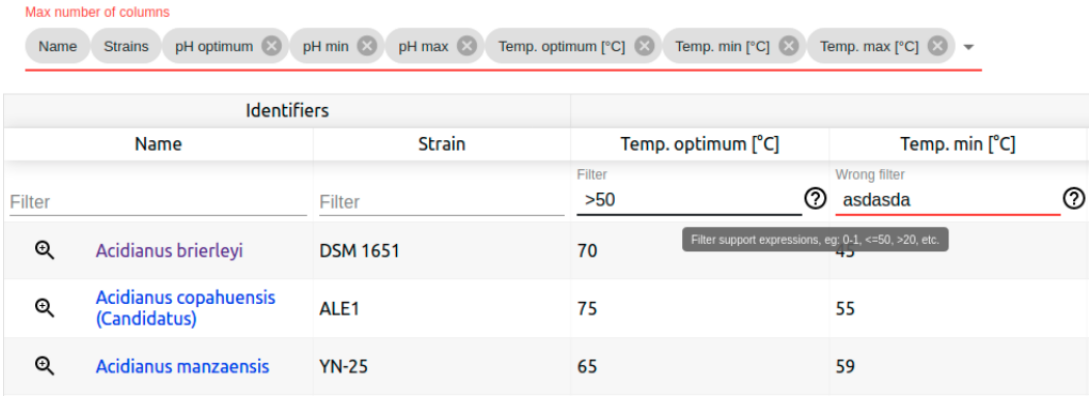

Figura 4.17: Mecanismos de *awareness*.

Por último se implementaron animaciones de carga para cada una de las funcionalidades que realizan consultas a la API, con esto se le entrega información al usuario acerca del estado de la aplicación lo cual es sumamente relevante tomando en cuenta que ciertas características como la búsqueda de proteínas pueden tomar unos segundo en entregar resultados dependiendo de la cantidad de filtros utilizados.

4.4.5. Optimización de componentes

Optimización componente resultado de búsqueda.

Como se mencionó en secciones anteriores, los resultados para búsqueda aparecen en una tabla con los elementos encontrados, particularmente en la búsqueda de proteínas podemos encontrar una tabla de resultados de gran tamaño (más de 1000 elementos), estos resultados se encuentran en páginas de tamaño 1000 por lo cual en el peor caso podemos encontrar 1000 elementos visibles en los resultados de búsqueda. Cada uno de estos elementos es una fila en la tabla de resultados y además de mostrar la información permiten abrir un elemento de diálogo con el resumen de la proteína seleccionada (figura 4.9).

Este componente de diálogo forma parte del estado del componente padre lo cual implica que si se actualiza el estado del componente padre se debe actualizar el componente hijo provocando un nuevo *render* (solo la parte visual, un nuevo *render* no implica necesariamente una nueva llamada a la API), con esto cada vez que se selecciona una proteína se abre un diálogo lo cual desencadena una actualización en el componente de resultados, que en el peor de los casos, implica una revisión o actualización de una tabla de 1000 elementos.

Esta operación es extremadamente costosa e ineficiente ya que implica dibujar cada fila de la tabla múltiples veces cada vez que se haga *click* en una proteína. Como se observa en la figura 4.18, la cual contiene una imagen de la herramienta de *profiling* dn Google Chrome, abrir el diálogo toma 8.4 segundos, mientras que cerrarlo toma una cantidad similar de tiempo. El tiempo total de ejecución de *scripts* alcanza los 18 segundos aproximadamente, esto incluye otros eventos, pero la mayor carga son los 16 segundos que toma abrir y cerrar el diálogo.

Para solucionar este problema se utilizaron técnicas de *memoization* sobre el componente de resultados. Para esto se usó React.Memo que permite revisar ciertos argumentos antes de actualizar un componente, en este caso revisa que la URL, el estado del formulario y los resultados sean iguales antes de actualizar el componente.

Para lograr esto fue necesario modificar la estructura del componente de búsqueda y del componente de resultados, además fue necesario agregar la URL al estado general e implementar una función que compara estados previos y actuales, lo cual es sumamente sencillo utilizando las funciones provistas por React, tal como se ve en el fragmento de código 4.6.

Código 4.6: Ejemplo React.Memo.

```
function areEqual(prevProps, nextProps) {
  if (prevProps.url === nextProps.url) {
    return true
  }
  //return true if passing nextProps to render would return the same result

  return false
}
const MemoizedProteinResultsTable = React.memo(ProteinResultsTable, areEqual);
```

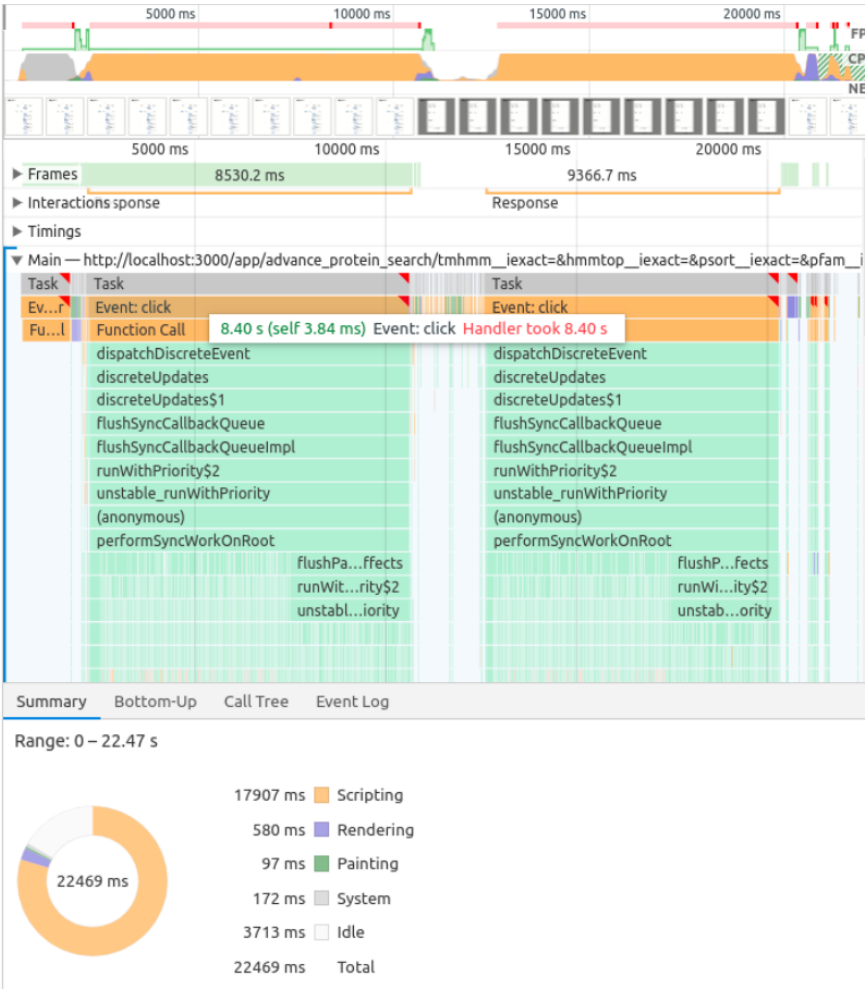

Figura 4.18: Rendimiento sin React memo.

Con todo lo anterior se mejoraron considerablemente los tiempos de carga y respuesta en este componente tal como se observa en la figura 4.19, donde los tiempos de carga del componente de diálogo alcanzan tal solo 96 milisegundos, mientras que el tiempo total para la ejecución de *scripts* es inferior a los 2 segundos.

Cabe mencionar que React.Memo, no requiere una función de comparación de estados, ya que en caso de no recibir ninguna React compara los *props* del componente previos a la actualización y decide si es necesario actualizar el componente. Debido a esto también se utilizó este tipo de técnica en otros componentes más pequeños dentro de esta aplicación.

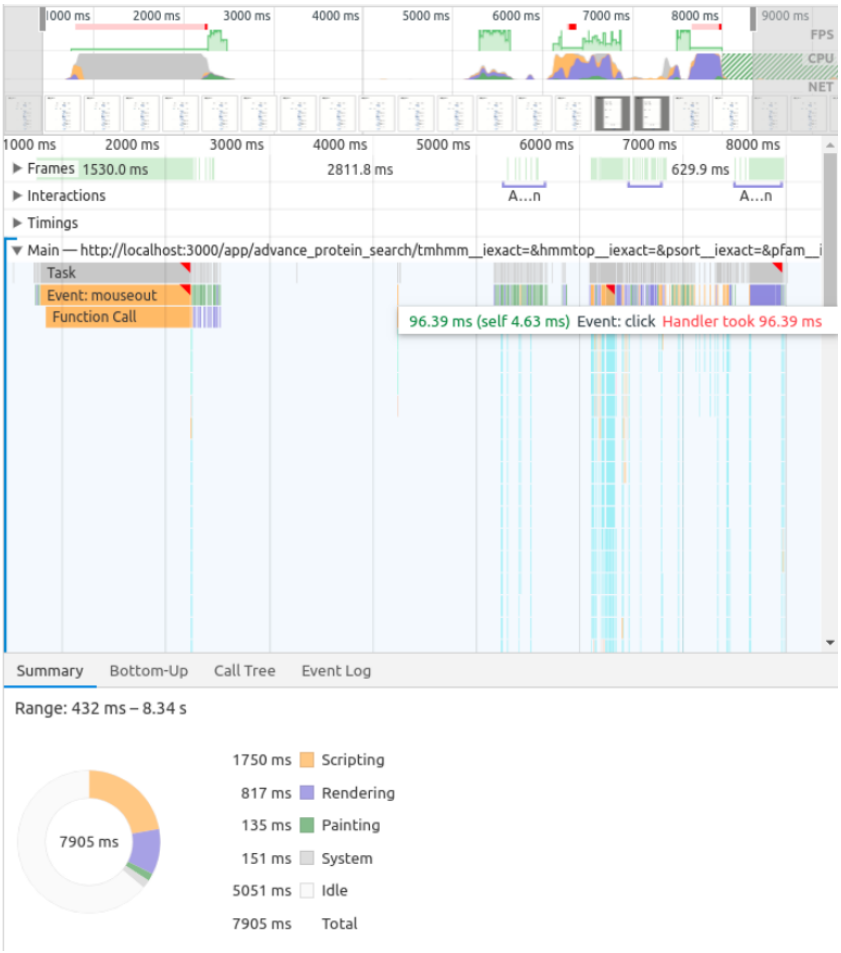

Figura 4.19: Rendimiento con React memo.

Optimización componente gráfico de puntos

Luego de implementar este componente se notaron problemas de rendimiento al momento de seleccionar múltiples organismos, esto se debe a que el rectángulo de selección destaca dinámicamente punto por punto cada vez que uno arrastra el *mouse* mientras se modifica el tamaño del área seleccionada, dentro de áreas pequeñas el retraso no es notorio, pero si se desean seleccionar bloque con 100 o más organismo se pueden apreciar problemas de respuesta en la interfaz.

Para solucionar este problema se decidió modificar el comportamiento de la herramienta de selección para que esta actuara de manera *lazy*, esto implica que cada vez que se seleccione un bloque de elementos, dichos elementos solo aparecerán destacados cuando el usuario deje de arrastrar el *mouse*, con esto un usuario no debiese sentir ningún retraso mientras esté arrastrando, moviendo o modificando el área de selección.

De todo lo anterior es posible observar la importancia de optimizar diferentes componentes y elementos dentro de una aplicación, pequeños cambios en la estructura pueden disminuir considerablemente la carga de los servidores y la velocidad de ejecución de la aplicación lo cual a su vez mejora la experiencia final del usuario.

Capítulo 5

Test y validación

Test y validación son dos aspectos transversales que se consideraron durante toda la etapa de desarrollo. En el caso del servidor *backend*, se implementaron diferentes *test* encargados de revisar la integridad y estructura de las respuestas de la API, mientras que en el caso del servidor *frontend*, cada componente en la interfaz era revisado y validado con otros miembros del laboratorio manteniendo una metodología iterativa.

5.1. Test de integración *backend*

Encontramos dos tipos de *test* en el *backend*, *test* de estructura para las respuestas y *test* que revisan el de estado de las respuestas. Para esto se utilizó Postman[11], un *software* que permite revisar y validar todo tipo de API ya sea utilizando la plataforma web o una aplicación de escritorio, otra ventaja que ofrece este *software* es la capacidad de automatizar los diferentes *tests* y tareas simplificando este tipo de procesos.

Los primeros *tests* implementados fueron *tests* de tiempos de respuesta y tipo de respuestas, estos se encargan de revisar si es posible acceder a la API y medir el tiempo que existe entre consulta y respuesta, además en caso de existir errores o problemas de conexión Postman entrega registro detallado de estos (estado de los *tests*, tiempos de conexión, resultados en la respuestas y errores).

En el caso de la estructura, se implementaron *tests* que revisan la forma de las respuestas a la API, para esto se define un esquema con la estructura general e interna de la respuesta lo que incluye formato de los elementos, tipo de los elementos (números o palabras) y patrones (anexo B.1). Este tipo de *test* es de vital importancia, ya que con estos es posible asegurar que siempre llega una respuesta correcta al *frontend*, aún así es importante destacar que este tipo de *test* no asegura que los datos entregados sean correctos solo que poseen una estructura correcta.

Este tipo de *test* fue implementado para cada uno de los *endpoints* existentes lo cual permite revisar de manera continua los cambios que se realizan en la API y pueden ser usados tanto localmente como en producción ya que son completamente independientes del servidor. Un ejemplo concreto con los *tests* utilizados para el *endpoint* asociado a los organismos se puede revisar en el anexo B.2.

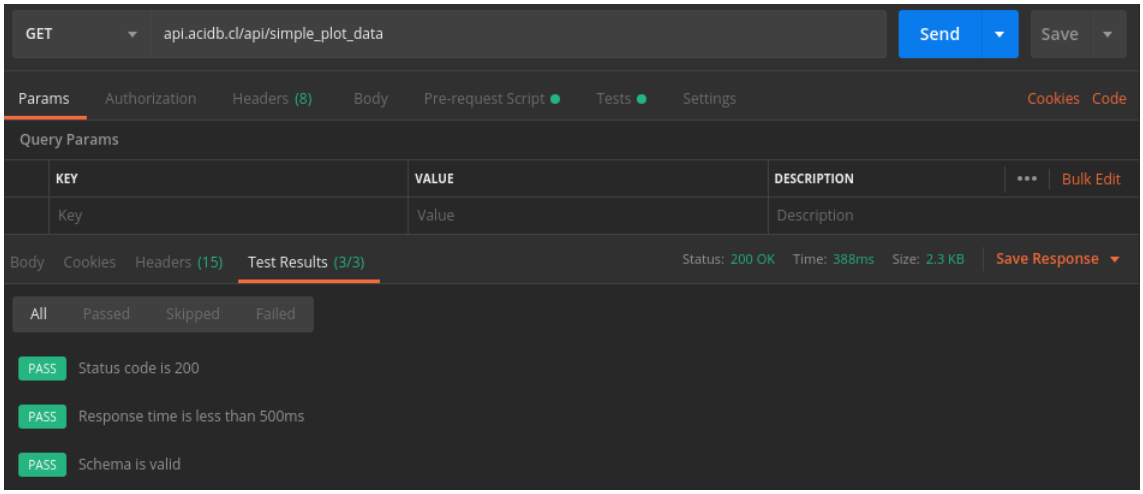

Figura 5.1: Interfaz de resultados en Postman.

5.2. Validación interfaces y test de usabilidad

Como se mencionó anteriormente se realizó validación de interfaces durante toda la etapa de desarrollo revisando cada componente al menos una vez, además al final de la etapa de desarrollo se validó la plataforma completa utilizando la escala SUS (*System Usability Scale*[3]), la cual consiste en una pequeña encuesta de usabilidad con preguntas predefinidas, donde se le asigna un puntaje entre 1 a 5 a cada una de estas.

Para interpretar los resultados, se evalúan por separado los puntajes de preguntas pares e impares. En el caso de las preguntas pares, el valor asignado es 5 menos el puntaje de la pregunta, en el caso de las impares simplemente se le resta 1 a cada pregunta, el resultado de ambas se suma y se multiplica por 2.5, este resultado final que puede ir de 0 a 100 representa la usabilidad de la plataforma o aplicación. Un resultado mayor a 68 puede ser considerado mejor a la media.

Dicha encuesta fue desarrollada en *Digital Equipment Corporation* (Reino unido) en el año 1986 por John Brook como una herramienta de validación dentro de sus propias oficinas, debido a su simplicidad y adaptabilidad esta encuesta logró un gran éxito y hoy en día es ampliamente utilizada dentro de diferentes áreas. La tabla completa de preguntas y respuestas con su respectiva escala y puntajes se puede consultar en el anexo B.1

Tabla 5.1: Resultados SUS.

Tipo integrante	Odd SUS Score	Even SUS Score	Total	Final Score
Laboratorio	16	13	29	72.5
Laboratorio	17	18	35	87.5
Laboratorio	17	19	36	90
Laboratorio	18	20	38	95
Externo	16	20	36	90
Externo	16	18	34	85

5.3. Documentación y aspectos generales

Encontramos dos tipos de documentación dentro de este proyecto, una asociada a la implementación y otra asociada a los datos y sus atributos lo que incluye el proceso de extracción y el criterio de inclusión en la base de datos.

Todo el trabajo realizado durante esta memoria de título se encuentra público en un repositorio Git, específicamente en un repositorio personal en GitLab[1], en dicho repositorio se encuentra la documentación que incluye dependencias, estructura y ejecución de los servidores *backend* y *frontend*.

Por otra parte encontramos la documentación asociada a los atributos de los microorganismos, esta se encuentra en el componente de documentación dentro de la plataforma web, el cual incluye detalles específicos de los organismos aliadófilos utilizados en la base de datos. Este componente fue implementado en conjunto con los integrantes del laboratorio ya que el contenido de este se enfoca principalmente en el aspecto biológico de los atributos y las herramientas informáticas utilizadas para identificarlos.

Además, durante todo el proceso de desarrollo se utilizaron archivos de configuración independientes para administrar credenciales, direcciones IP, llaves de seguridad entre otras cosas. Estos archivos de configuración son locales por lo cual se ignoran dentro del repositorio Git, dentro de este solo aparece un archivo tipo *template* con la estructura básica de la configuración (pero sin ningún dato), con esto se asegura que nunca se publicaran las credenciales manteniendo un archivo de configuración dentro del repositorio.

Por último destacar que durante todo el desarrollo se utilizó un servidor de pruebas provisto por la Fundación Ciencia & Vida, mientras que para las pruebas finales de validación se utilizó Heroku[8], un servicio en la nube que ofrece *hosting* de aplicaciones web, el cual se configuró con el dominio *acidb.cl*, esta versión de la plataforma solo contiene una cantidad acotada de los datos y funcionalidades a la espera de que el laboratorio publique oficialmente los datos y resultados. Este mismo dominio alojará la versión final de la plataforma con todos los datos cargados junto con todas las funcionalidades descritas en este informe a medida que se liberen las publicaciones correspondientes.

[1] https://gitlab.com/Hawkline451/acidb

Capítulo 6

Conclusiones y trabajo futuro

En el presente trabajo de título, se describe la concepción, diseño e implementación de la plataforma web AciDB, la cual permite buscar, filtrar y visualizar los atributos o características de diferentes microorganismos. Específicamente el desarrollo de esta herramienta da a conocer parte del trabajo realizado por el centro de bioinformática enfocado principalmente en el estudio de organismos acidófilos.

Con todo lo descrito a lo largo de este informe es posible notar que la principal ventaja que entrega la implementación de esta plataforma es la posibilidad de publicar datos e información recopilada por el centro de bioinformática los cuales al estar en una plataforma web, puede ser utilizado por otros investigadores externos al laboratorio. Además de lo anterior, otro punto importante a considerar son las ventajas que otorga la utilización de una base de datos relacional para almacenar datos de los organismos, la cual sumada a la interfaz web de administración implementada, permite a los investigadores y administradores de la plataforma editar, agregar y actualizar los datos, los cuales además de ser utilizados por la herramienta web, también pueden ser aprovechados para otro tipo de proyectos internos del laboratorio. De la misma forma, esta base de datos permitió ordenar y organizar muchos de los archivo Excel y CSV que eran utilizados como principal medio para almacenar información de los microorganismos previo al desarrollo de esta memoria.

Por otra parte, durante toda la etapa de desarrollo se identificaron dos dificultades importantes. La primera de estas fue la creación de componentes e interfaces intuitivos para los usuarios finales, en este caso un investigador dentro del área de la biología. Se invirtió gran parte del tiempo validando y reconstruyendo fragmentos de componentes e interfaces para que éstas se adaptaran a las necesidades de los usuarios.

La segunda dificultad que se presentó estuvo relacionada con la optimización de componentes web, principalmente asociada a disminuir el tiempo de carga y *render*, de los gráficos y tablas siempre con la finalidad de mantener la usabilidad y funcionalidad de la herramienta web. Este último problema se resolvió utilizando técnicas de *memoization* y optimización de componentes tal como se mencionó en la sección 4.4.5 dentro de este informe.

Dicho lo anterior, es posible concluir que los objetivos generales y específicos presentados al comienzo de este documento se cumplieron. Se logró transformar y almacenar datos recopilados por el laboratorio en una base relacional, se implementó una API REST para la lectura

e interacción con la base de datos y por último se desarrolló una herramienta web como medio de visualización de estos datos, todo esto con la finalidad de dar a conocer parte del trabajo realizado por el Centro de Bioinformática de la Fundación Ciencia y Vida junto con unificar la información de organismos acidófilos lo cual facilitará el trabajo de investigadores dentro y fuera del laboratorio.

Además esta memoria de título formará parte de un artículo científico que será publicado dentro de la primera mitad del año 2020, el cual se enfoca en la extracción de datos e implementación de la plataforma web desarrollada.

Finalmente dada las características y la arquitectura de esta herramienta web, es posible agregar nuevas funcionalidades, características o conexiones a servicios externos dependiendo de futuros requerimientos, debido a esto como trabajo futuro se propone la integración con servicios de NCBI tales como *protein* BLAST[4], extensión de las funcionalidades asociadas a proteomas e incorporación de nuevos organismos y atributos. Por lo mismo esta plataforma se pueden considerar como la base para futuros proyectos dentro del centro de bioinformática.

Bibliografía

[1] David Anderson. *Kanban - Successful Evolutionary Change for your Technology Business*. Blue Hole Press, 1st edition, 2010. ISBN 098-452-140-2.

[2] Balsamiq. Rapid, effective and fun wireframing software. https://balsamiq.com/. Último acceso: 08-09-2019.

[3] John Brooke. Sus: A quick and dirty usability scale, 1996.

[4] NCBI National Center for Biotechnology Information. Basic local alignment search tool. https://blast.ncbi.nlm.nih.gov. Último acceso: 2019-07-17.

[5] Istvan Simon Gabor E. Tusnady. The hmmtop transmembrane topology prediction server. http://www.enzim.hu/hmmtop/. Último acceso: 19-04-2019.

[6] GitLab. Ncbi.twenty nine years of gwoth, congressional justification fy 2020 department of health and human services national institutes of health national library of medicine (nlm). https://www.nlm.nih.gov/about/2020CJ.html.

[7] DasSarma Lab Group. The haloarchaeal genomes database. https://halo.umbc.edu. Último acceso: 19-04-2019.

[8] Heroku. Cloud application platform. https://heroku.com. Último acceso: 1-03-2020.

[9] Ortiz R. Ossandon F. Jorquera, R. Database for single exon coding sequences in mammalian genomes. http://www.sinex.cl. Último acceso: 18-04-2019.

[10] Kerrin Mendler, Han Chen, Donovan H Parks, Briallen Lobb, Laura A Hug, and Andrew C Doxey. AnnoTree: visualization and exploration of a functionally annotated microbial tree of life. *Nucleic Acids Research*, 47(9):4442–4448, 04-2019. ISSN 0305-1048. doi: 10.1093/nar/gkz246. URL https://doi.org/10.1093/nar/gkz246.

[11] Postman. Postman. the collaboration platform for api development. https://postman.com/, Último acceso: 01-10-2020.

[12] React. Introducing hooks. https://reactjs.org/docs/hooks-intro.html. Último acceso: 12-12-2019.

[13] Perdita Steeven, Rob Pooley, Marta Fernández Alarcón, and Ruben Gonzalez Crespo. *Utilización de UML en Ingeniería del Software con Objetos y Componentes*. Pearson Educación, 1st edition, 2007. ISBN 978-84-7829-086-4.

[14] Abramov D. Carlberg West R. West, D. hook-flow. https://github.com/donavon/hook-flow, 13-10-2019.

Anexo A

Scripts

A.1. Carga masiva

Código A.1: Script carga masiva usando modelos de Django.

```
1
2  import csv
3  from acidb.models import *
4
5  """
6      Script to import data from .tsv file to Model Database DJango
7      To execute this script run:
8                                  1) manage.py shell
9                                  2) exec(open('load_csv.py').read())
10                                 or copy everything in the shell
11
12 Load organism
13 """
14
15 path = '<path_files>'
16 def load_organism():
17     with open(path+'organism.tsv') as f:
18         reader = csv.reader(f, delimiter='\t')
19         # Ignore first row
20         next(reader, None)
21         for row in reader:
22             row = [None if col=='' else col for col in row]
23             _, created = Organism.objects.get_or_create(
24                 id_organism = row[0],
25                  visibility  = bool(row[1]),
26                 name = row[2],
27                 isolated = row[3],
28                 state = row[4],
29                 seq_date = row[5],
30                 gen_size = row[6],
31                 gen_contamination = row[7],
```

```
                gen_completeness = row[8],
                gc_percentage = row[9],
                n_orfs = row[10],
                temp_associated = row[11],
                temp_min = row[12],
                temp_max = row[13],
                ph_associated = row[14],
                ph_min = row[15],
                ph_max = row[16],
                access_src = row[17],
                access_id = row[18],
                biosample = row[19],
                bioproject = row[20],
                ftp_url = row[21],
                annotation = row[22]
            )

def load_taxonomy():
    with open(path+'taxonomy.tsv') as f:
        reader = csv.reader(f, delimiter='\t')
        # Ignore first row
        next(reader, None)
        for row in reader:
            row = [None if col=='' else col for col in row]
            _, created = Taxonomy.objects.get_or_create(
                organism = Organism.objects.get(pk = (row[0])),
                tax_src = row[1],
                tax_id = row[2],
                domain = row[3],
                phylum = row[4],
                tax_class = row[5],
                order = row[6],
                family = row[7],
                genus = row[8],
                species = row[9]
            )

def load_references():
    with open(path+'refs_clean.tsv', encoding='utf-8') as f:
        reader = csv.reader(f, delimiter='\t')
        # Ignore first row
        next(reader, None)
        for row in reader:
            row = [None if col=='' else col for col in row]
            _, created = Reference.objects.get_or_create(
                organism = Organism.objects.get(pk = (row[0])),
                ref_text = row[1]
            )
...
```

Anexo B

Test y validación

B.1. Postman test

Código B.1: Estructura de un esquema en Postman.

```
var organismSchema = {
  "definitions": {},
  "type": "object",
  "required": [
    "id_organism",
    "name",
    "isolated",
    "state",
    "seq_date",
    "gen_size",
    "gen_completeness",
    "gen_contamination",
    "gc_percentage",
    "n_orfs",
    "temp_associated",
    "temp_min",
    "temp_max",
    "ph_associated",
    "ph_min",
    "ph_max",
    "access_src",
    "annotation",
    "strains",
    "taxonomy",
    "access_id",
    "ftp_url"
  ],
  "properties": {
    "id_organism": {
      "$id": "#/properties/id_organism",
      "type": "integer",
```

```
      " title ": "The Id_organism Schema",
      "default": 0,
      "examples": [
        1
      ]
    },
    "name": {
      "$id": "#/properties/name",
      "type": "string",
      " title ": "The Name Schema",
      "default": "",
      "examples": [
        "Acidilobus saccharovorans"
      ],
      "pattern": "^(.*)$"
    },
    "temp_max": {
      "$id": "#/properties/temp_max",
      "type": "integer",
      " title ": "The Temp_max Schema",
      "default": 0,
      "examples": [
        90
      ]
    },
    "taxonomy": {
      "$id": "#/properties/taxonomy",
      "type": "array",
      " title ": "The Taxonomy Schema",
      "items": {
        "$id": "#/properties/taxonomy/items",
        "type": "object",
        " title ": "The Items Schema",
        "required": [
          "domain",
          "phylum"
        ],
        "properties": {
          "domain": {
            "$id": "#/properties/taxonomy/items/properties/domain",
            "type": "string",
            " title ": "The Domain Schema",
            "default": "",
            "examples": [
              "Archaea"
            ],
            "pattern": "^(.*)$"
          },
         ...
        }
```

```
      }
    },
    ...
  }
}

//Store it as environmental variable (or global).
pm.environment.set("organismSchema", organismSchema);
```

Código B.2: Test de estructura e integridad para un *endpoint*

```

//Get it from environmental variable (or global).
var expectedSchema = pm.environment.get("organismSchema");

//Check response code
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});

//Check response time
pm.test("Response time is less than 500ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(500);
});

//Check schema
pm.test('Schema is valid', function() {
    var result=tv4.validateResult(JSON.parse(responseBody), expectedSchema);

    if(!result.valid){
        console.log(result);
    }

    pm.expect(result.valid).to.be.true;
})
```

B.2. Escala usabilidad de sistemas

Tabla B.1: Tabla SUS completa, 5 representa muy de acuerdo y 1 representa completamente en desacuerdo

	Laboratorio	Laboratorio	Laboratorio	Laboratorio	Externo	Externo
I think that I would like to use this app frequently	5	4	4	5	5	4
I found the app unnecessarily complex	3	1	1	1	1	2
I thought the app was easy to use	4	4	4	4	4	4
I think that I would need the support of a technical person to be able to use this app	2	1	1	1	1	1
I found the various functions in this app were well integrated	4	5	5	5	4	5
I thought there was too much inconsistency in this app	3	3	1	1	1	1
I would imagine that most people would learn to use this app very quickly	4	5	4	4	5	4
I found the app very cumbersome to use	2	1	2	1	1	2
I felt very confident using the app	4	4	5	5	3	4
I needed to learn a lot of things before I could get going with this app	2	1	1	1	1	1