Informe de Práctica Profesional

CC5901 - Desarrollo de Aplicaciones Web Financieras y de Open Banking

Estudiante:	Josefina Bustos
Correo:	josefina.bustos.12@gmail.com
Teléfono:	+56975479183
Periodo:	03/01/2024 - 29/02/2024
Empresa:	SYNAPTIC SERVICIOS INFORMATICOS SPA
Supervisor:	Sebastián Sánchez
Teléfono del supervisor:	+56976095844
Correo del supervisor:	sebastian.sanchez@synaptic.cl

Fecha de entrega: 12 de abril de 2024
Santiago, Chile

Resumen

Durante la Práctica Profesional II, se llevaron a cabo diversas tareas en el área de desarrollo de software, centrándose especialmente en el trabajo de backend y frontend.
En un principio, se dedicó tiempo a la creación de componentes para una aplicación. Posteriormente, se realizaron tareas de desarrollo en un programa de administración de caché, que incluyeron la incorporación de nuevas funcionalidades y la refactorización del código existente. Finalmente, se culminó con la implementación de una interfaz destinada a realizar operaciones CRUD en diferentes tablas de administrador, junto con la adición de la funcionalidad de visualizador de historial.
Además de las actividades mencionadas, se establecen objetivos generales y específicos para el período de práctica, se abordan las dificultades encontradas y se presentan las soluciones implementadas para superarlas.
Para concluir, se realiza una breve reflexión sobre la experiencia adquirida durante el tiempo en Synaptic, seguida de una conclusión donde se resume el trabajo realizado a lo largo del período de prácticas.

Índice de Contenidos

1. Introducción

1.1. Descripción de la empresa

Synaptic es una empresa Fintech centrada en desarrollar productos digitales para el sector financiero y proporcionar soluciones basadas en Open Data para agilizar los procesos que requiera el cliente.
Actualmente, en Synaptic están concentrando los esfuerzos en el desarrollo de Boufin, una plataforma tecnológica que busca simplificar trámites financieros y operaciones bancarias. Esto permitiría acceder a ofertas crediticias y realizar operaciones desde una sola aplicación, eliminando la necesidad de acudir a instituciones físicas o interactuar con un ejecutivo. En 2023, la propuesta de Boufin fue ganadora del *Premio Innovación Fintech*.
Con el fin de garantizar la seguridad de la empresa, antes de iniciar el trabajo de desarrollo, los practicantes deben completar un curso sobre la norma ISO 27001, seguido de un examen final. Asimismo, se revisan documentos que contienen diversas normativas de la empresa, como por ejemplo, listados de las aplicaciones permitidas en los ordenadores de la empresa.

1.2. Descripción del trabajo realizado

En el contexto de Boufin, que aún se encuentra en desarrollo, hay áreas que aún necesitan trabajo, tanto en frontend como en backend. En Synaptic, se presentó la oportunidad de trabajar en el desarrollo de frontend al principio, para luego tener la libertad de elegir entre trabajar en frontend o backend, dependiendo de las necesidades o intereses de cada practicante. Por lo tanto, habiendo terminado el trabajo en frontend, se decidió trabajar en el desarrollo de backend, debido a la poca experiencia práctica en esa área, y posteriormente se optó por volver a trabajar en frontend.

2. Descripción del problema

Como se mencionó, Boufin es un producto en desarrollo. La problemática a abordar incluye actividades como el desarrollo de componentes, verificación y validación del código desarrollado, identificación y corrección de errores, así como la colaboración con miembros del equipo de desarrollo.
Luego de realizar una presentación sobre la empresa y su equipo, se expuso durante la primera jornada en Synaptic que todas las actividades girarían en torno al desarrollo de Boufin.
Para comenzar, se realizó una presentación sobre qué es Boufin, su objetivo y a quién está dirigido, junto con una demostración de un widget de la misma aplicación. A partir de esto, surge la primera necesidad, que corresponde al desarrollo de componentes que serían utilizados dentro de la aplicación, como *checkbox, textfield* y *search bar*. Dicho trabajo, a pesar de su simplicidad, constituyó su propio nivel de dificultad, teniendo en cuenta que las herramientas y librerías con las que se llevaría a cabo eran prácticamente nuevas o poco utilizadas en el contexto universitario.
Al finalizar el desarrollo de los componentes, se decidió trabajar en el backend, donde se presentó una nueva problemática relacionada con el sistema de almacenamiento en caché, el cual podía presentar fallas eventualmente, por ejemplo, si la conexión cliente-servidor fallaba. Por esta razón, se solicita reemplazar este sistema de caché por uno que se vea menos afectado en caso de una falla externa.
Habiendo completado el reemplazo del caché, se optó por trabajar nuevamente en frontend, por lo cual Synaptic asignó una tarea consistente en la creación de un sistema de administración, donde se trabajó con el CRUD de distintas tablas necesarias para la empresa al momento de crear entidades, usuarios o servicios para Boufin de manera accesible para cualquier miembro autorizado de la empresa.
Cabe mencionar que al enfrentar cada una de las problemáticas surgieron dificultades, las cuales afortunadamente se lograron resolver gracias al trabajo colaborativo con otras estudiantes en práctica y el resto de miembros del equipo de Synaptic, así como al esfuerzo propio.
Para finalizar, es importante destacar que, aunque las tareas mencionadas no eran urgentes, era necesario realizarlas para poder pulir el trabajo que ya se encontraba hecho y así permitirle al resto del equipo seguir avanzando en nuevas tareas.

3. Objetivos

En esta sección, se presentarán los objetivos que guían el presente informe.

3.1. Objetivo general

El objetivo general de este trabajo se centra en el desarrollo de software, específicamente, adquirir experiencia trabajando en backend y frontend, con el fin de impactar positivamente la experiencia del usuario y aliviar la carga del equipo de trabajo de la empresa. Asimismo, se busca crear un ambiente de confianza con los demás practicantes y el equipo de Synaptic, con el objetivo de fomentar la colaboración entre pares y lograr una rápida resolución de conflictos asociados al desarrollo. Por último, se espera adquirir conocimientos sobre nuevas herramientas y obtener experiencia práctica en la implementación de metodologías ágiles dentro de una empresa.

3.2. Objetivos específicos

A continuación, se detallan objetivos específicos que, al alcanzarse, contribuirán a la realización del objetivo general:

- Realizar tareas tanto en el frontend como en el backend según lo necesite el equipo, para avanzar en el desarrollo fullstack.
- Cumplir meticulosamente con las solicitudes del equipo de diseño, enfocadas en la experiencia del usuario y prestando especial atención a los detalles.
- Cumplir con los objetivos dentro de plazos establecidos internamente, priorizando la eficiencia y la efectividad de las soluciones para aligerar la carga del equipo.
- Examinar a fondo los conflictos encontrados antes de buscar ayuda de un tercero, con el propósito de mejorar en la resolución de problemas.
- Comunicar rápidamente situaciones que podrían resultar incómodas o molestas para las practicantes, con el fin de mantener un entorno seguro y de confianza.

4. Metodología

En esta sección se detallan los pasos seguidos para alcanzar los objetivos propuestos, así como las tareas realizadas para abordar los problemas descritos en la sección 2.
En primer lugar, todas las practicantes completaron una certificación de la norma ISO 27001, además de realizar una lectura detallada de la documentación de seguridad de la empresa. Una vez completadas estas tareas, se da inicio al trabajo de desarrollo, el cual será descrito en las secciones siguientes.
Teniendo en cuenta las expectativas de las practicantes, Synaptic estructura el trabajo a realizar durante enero y febrero de 2024 en dos áreas: frontend y backend. Durante el primer mes, el enfoque se encuentra en el frontend, y el mes siguiente se ofrece la flexibilidad de elección entre ambas áreas. Además, con excepción del viernes, se llevan a cabo reuniones diarias (*stand-up*) con todo el equipo para revisar el progreso diario, abordar las dificultades encontradas y planificar el trabajo futuro.
Cada tarea comienza con un proceso de adaptación y comprensión del código existente, con el fin de entender la estructura del proyecto, su propósito y las herramientas utilizadas. Al principio, Synaptic explica detalladamente el proyecto, pero a medida que se asignan nuevas tareas, son las practicantes quienes deben esforzarse por comprender por sí mismas, consultando solo aspectos puntuales.

4.1. Construcción de Componentes

Durante el primer mes del proceso de práctica, el trabajo se centra en la adaptación a la empresa y los pasos a seguir para llegar a una solución. Por esta razón, la primera tarea propuesta implica el desarrollo de diversos componentes para la aplicación. Esta labor facilita la adaptación, ya que, aunque cada practicante trabaja en un componente distinto y enfrenta diferentes dificultades, todos comparten una base común y códigos similares.
Los componentes a desarrollar, junto con instrucciones sobre cómo hacerlo y las bibliotecas útiles para ello, están organizados de manera clara en una tabla en *GitHub* dentro del proyecto de Boufin. Esta estructura simplifica el proceso de selección del componente en el que cada practicante trabajará. Cada practicante es responsable de desarrollar tres componentes diferentes, siguiendo una metodología que consiste en elegir un nuevo componente una vez finalizado el anterior.
Cada componente cuenta con un modelo elaborado en Figma, lo que facilita la obtención de medidas, colores y cambios estéticos que ocurren al interactuar con ellos.
Al finalizar un componente, el trabajo es revisado por el encargado del área correspondiente y se compara minuciosamente con el modelo diseñado en Figma. Si se encuentran detalles en el código o en la apariencia del componente, se solicita al practicante que realizó dicho componente que realice las correcciones necesarias. Si no se encuentran fallas, entonces el trabajo realizado se fusiona con la rama principal, sirviendo como base para el resto del equipo.

4.2. Reemplazo de Almacenamiento Caché

A comienzos de febrero se elige trabajar en backend, con el objetivo de adquirir experiencia en dicha área. Para ello, el equipo de Synaptic se organiza de manera que la tarea asignada cuente con un tutor especializado en backend.
La problemática a resolver está relacionada con la gestión de caché, donde se plantea reemplazar el método actual de almacenamiento (*Redis*) por uno más estable ante posibles fallas externas. Para abordar este desafío, el tutor recomienda el uso de una librería específica (*memory-cache*), lo que requiere una investigación sobre su funcionamiento y aplicación.
Al comienzo del trabajo, se establece como objetivo que, dependiendo del valor de una *feature flag*, el programa decida entre utilizar *Redis* o la librería mencionada. Esto implica la modificación del código existente para integrar la nueva funcionalidad. Además, se solicita refactorizar el código previamente creado para que cumpla con algún patrón de diseño, por lo que es necesario realizar una investigación para determinar cuál es el más adecuado.
Una vez finalizada la implementación, se establece la necesidad de someterla a pruebas de carga, para lo cual se introduce una nueva herramienta: *Google Cloud Run*. En esta etapa final, se asigna tiempo para familiarizarse con la aplicación, lo que implica leer documentación, experimentar con la herramienta y realizar consultas al tutor correspondiente.
Posteriormente, se inicia el proceso de prueba del nuevo caché, donde se recopilan ciertos datos solicitados por el tutor, como la detección de errores y el tiempo de respuesta promedio. Una vez completadas las pruebas y recopilada la información necesaria, se da por finalizado el proyecto asignado y se da comienzo al siguiente.

4.3. Admin CRUD

Al finalizar el trabajo relacionado con el caché, alrededor de mediados de febrero, se solicita trabajar nuevamente en el área de frontend. En esta ocasión, Synaptic asigna un proyecto asociado al administrador de bases de datos de Boufin. Específicamente, se solicita la creación de una interfaz que simplifique la gestión de las tablas utilizadas en la aplicación.
Para iniciar, se presentan las herramientas que se utilizarán durante esta fase del proyecto, seguidas de las directrices para la creación de la interfaz. El administrador actualmente cuenta con acceso a cuatro tablas distintas, sin embargo, en dos de ellas aún no tiene permisos de gestión. Por lo tanto, el trabajo se limita a dos tablas, donde se espera que la interfaz permita realizar operaciones de creación, lectura, actualización y eliminación (o deshabilitación) de valores, conforme a lo que implican las siglas CRUD.
Al concluir el trabajo, se emplea *Postman* para verificar la efectividad de la solución, asegurándose de que la función implementada en la interfaz haya logrado el efecto deseado. Además, se solicita al tutor responsable la revisión del trabajo realizado.

5. Descripción de la solución

A continuación se presenta una descripción más detallada del trabajo expuesto en la sección 4. En la sección 8 es posible ver más detalles sobre la manera en que se organiza el código de cada proyecto y breves secciones de código de los mismos.

5.1. Construcción de Componentes

Para iniciar, se proporciona un listado de diez componentes para trabajar. Primero se selecciona el componente *Checkbox*, seguido por *Search Bar* y posteriormente *TextField*.
Este proyecto cuenta con cinco herramientas principales a utilizar: *StoryBook*, que permite probar componentes de manera aislada, *Chakra UI*, biblioteca de componentes de interfaz de usuarios, *Figma*, que es una herramienta de diseño, *Typescript* y *React*. Sin embargo, hasta el momento, solo se había tenido un leve acercamiento a *Typescript*.
De manera general, los pasos a seguir para el desarrollo de cada componente son los mismos:

- Revisar en el tablero de *GitHub* los componentes asignados para trabajar.
- Revisar el diseño realizado en *Figma* y utilizarlo como guía para el desarrollo.
- Buscar documentación en *Chakra UI* sobre un componente adecuado para realizar la tarea.
- Crear archivos tanto para el manejo de la estética y funcionalidad del componente como para que los cambios se vean reflejados en *StoryBook*.
- Crear pruebas unitarias para el componente.

No obstante, la creación de cada componente trajo consigo sus propias dificultades, las cuales serán descritas a continuación.

5.1.1. Checkbox

Utilizando la misma definición brindada en Synaptic, éste es un elemento interactivo que permite a los usuarios seleccionar o deseleccionar una o varias opciones. En la Figura 1 se observa la propuesta de diseño que guía la creación del componente.

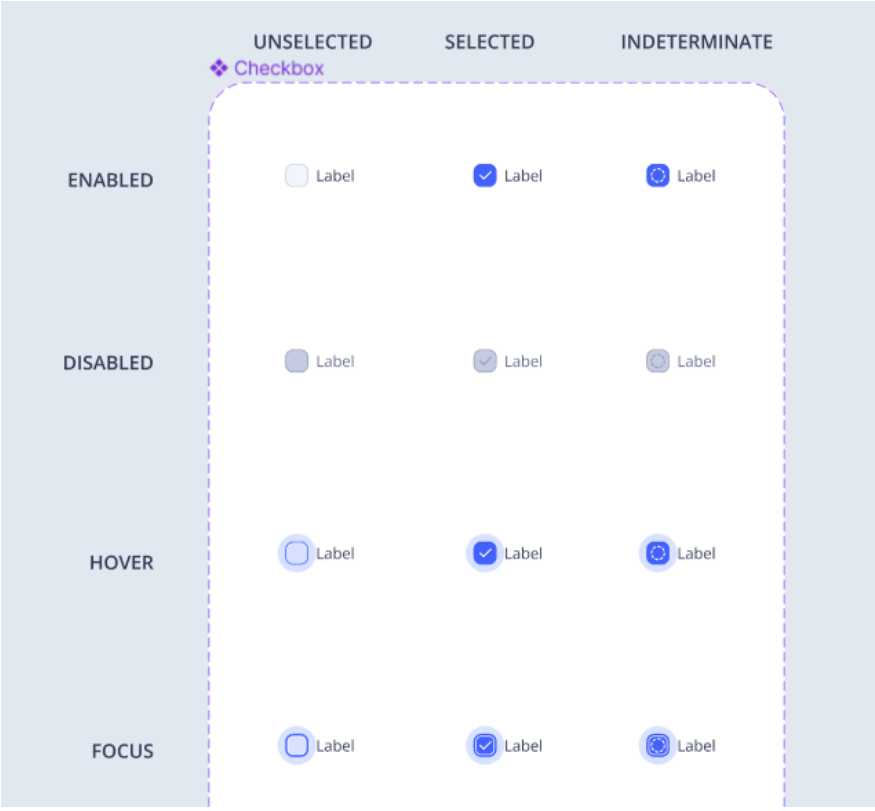

Figura 1: Diseño Checkbox Figma

Durante la creación de este componente, se enfrentaron varias dificultades relacionadas con el manejo de las herramientas requeridas y la comprensión de la estructura del código. Esto llevó a realizar varias consultas al tutor encargado. Además de lo anterior, surgieron dificultades al intentar implementar ciertos aspectos estéticos en el componente. Por ejemplo, hubo problemas al intentar hacer que la forma del *hover* fuera redonda, así como al manejar el cambio de ícono cuando el componente se encontraba en estado indeterminado.
Para abordar la primera dificultad, se propuso una solución alternativa donde, en lugar de utilizar una forma completamente redonda, se opta por un enfoque con bordes redondeados en una forma cuadrada. Esta solución fue aceptada por el resto del equipo.
En cuanto a la segunda problemática, se trabajó en colaboración con el tutor para lograr cambiar el ícono del componente y garantizar que se viera tal como estaba especificado en el diseño original.

5.1.2. Search Bar

El segundo componente con el que se decide trabajar es la barra de búsqueda, que permite a los usuarios ingresar términos de búsqueda y debe mostrar recomendaciones a medida que el usuario tipea. En la Figura 2 se muestra el diseño del componente en *Figma*.

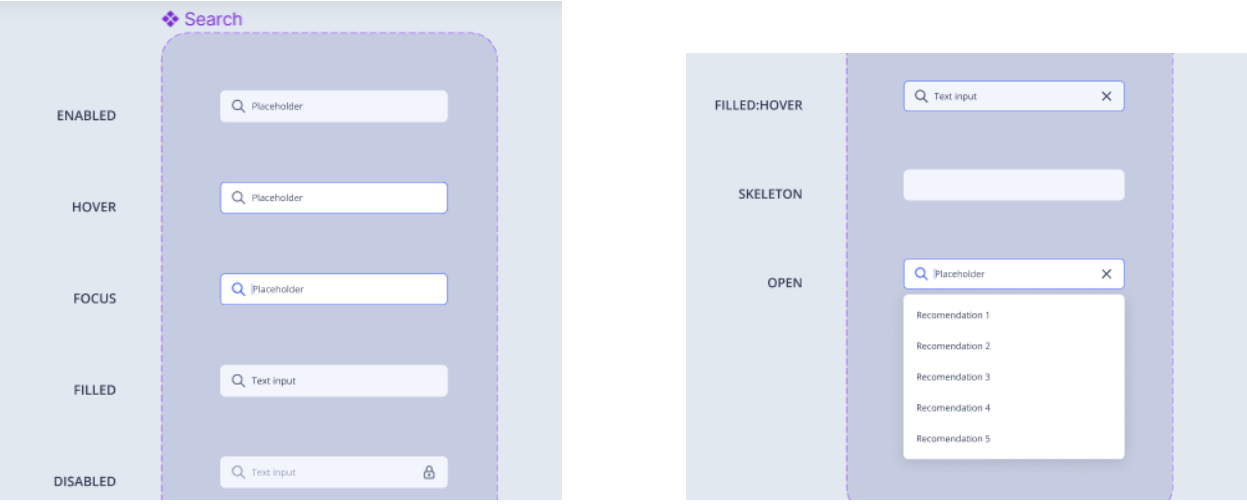

Figura 2: Diseño Search Figma

Para este componente no existe una base definida en *Chakra UI* que permita el desarrollo de manera directa, por lo cual se tuvo que revisar la documentación de cada uno de los componentes un busca de aquellos que pudieran ser útiles. Así, se llega a la elección de dos componentes distintos: *Input* para la barra donde se ingresan los términos y *List*, para el manejo de las recomendaciones.
A continuación se presentan como lista las dificultades enfrentadas y cómo fueron resueltas.

- Hubo propiedades de CSS que a pesar de ser cambiadas de manera correcta en el código, no se veían reflejadas en el *StoryBook*, por lo cual se tuvo que buscar en todo el código dónde sería posible realizar los cambios y que realmente fueran aplicados. Finalmente se encontró ese lugar y se logró resolver el conflicto.
- Manejar la programación basada en eventos para el manejo del filtro de las recomendaciones a medida que se tipea, junto con el manejo de la selección de una recomendación.

Finalmente, se logran superar dichas dificultades y se crea el componente tal cual se solicita en el diseño.

5.1.3. TextField

El *TextField* o campo de texto es un componente que permite a los usuarios ingresar datos de forma intuitiva y eficiente. En la Figura 3 se presenta el diseño realizado en Figma.

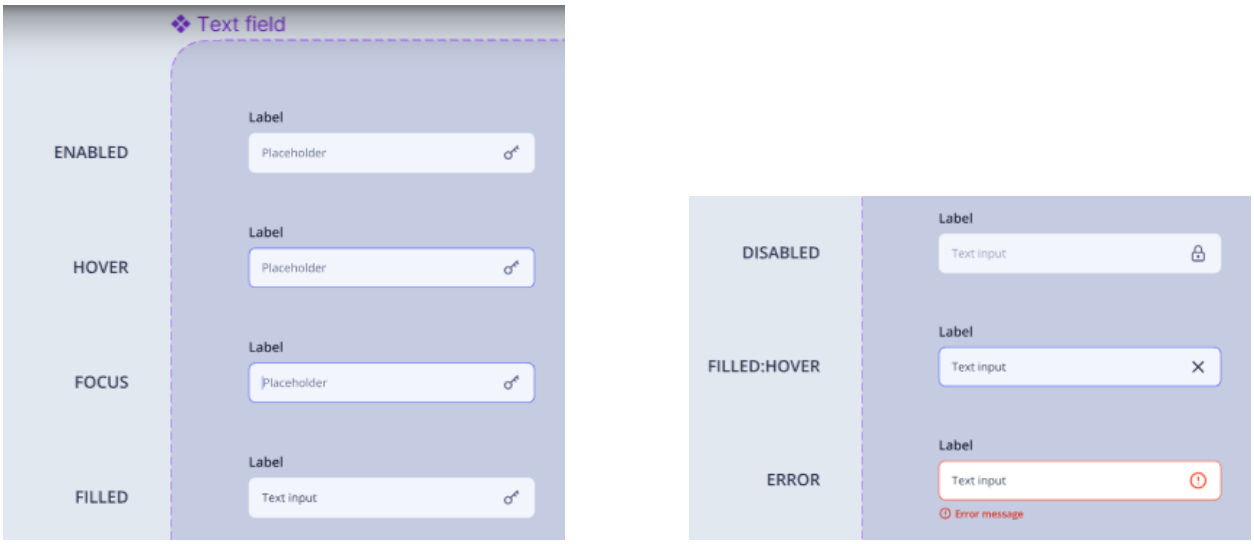

Figura 3: Diseño TextField Figma

Para este componente, era necesario volver a utilizar el elemento *Input* de *Chakra UI*. Sin embargo, esto planteaba dificultades, ya que la estructura básica de dicho componente había sido modificada para adaptarse a la barra de búsqueda. Por lo tanto, fue necesario refactorizar esa sección del código para que el componente *Input* se ajustara tanto a las necesidades del campo de texto como a las de la barra de búsqueda. Además, al realizar este cambio, fue necesario crear dos variantes distintas: una opción para generar la barra de búsqueda y otra para generar el campo de texto.
A pesar de tener que realizar un cambio relativamente significativo, considerando el trabajo realizado hasta el momento, al final resultó en una simplificación de la tarea de creación del campo de texto. Esto se debió a que no fue necesario crear un nuevo componente desde cero, y además, el componente anterior compartía muchas similitudes con el nuevo, por lo que las dificultades previas ya no representaban un problema en la actualidad.

5.2. Reemplazo de Almacenamiento Caché

Antes de comenzar en el proyecto, se solicita practicar y conocer sobre el uso de caché mediante la creación de un sistema de caché para solicitudes a una API determinada y así resolver todas las dudas referentes al uso de librerías y el manejo de solicitudes del tipo *get*. Una vez finalizada dicha actividad, se da inicio al proyecto real.
Al iniciar este nuevo proyecto, se señala que actualmente se está utilizando *Redis* para el manejo del caché. Sin embargo, se han experimentado situaciones en las que se pierde la conexión cliente-servidor, lo que afecta el sistema de caché. Debido a esto, surge la necesidad de crear un reemplazo que pueda ser activado mediante una *feature flag*, una herramienta que permite seleccionar qué parte del código se utilizará en función de un valor determinado. Para implementar este reemplazo, se sugiere utilizar la librería *memory-cache*. Además, se solicita refactorizar una sección del código de manera que todo lo asociado a la

inicialización de *Redis* quede en una sección aparte y dependiente del valor de la *feature flag*.
Una vez recibidas las instruciones, se comienza con la comprensión del código entregado, para luego seguir con la implementación de la librería sugerida. La idea inicial consistía en crear algo funcional pero mejorable y luego ir mejorándolo a medida que se avanzaba en el proyecto y se recibían sugerencias y correcciones por parte del tutor. Así, se logra un programa eficiente y bien modularizado, siguiendo el patrón *strategy*.
Una vez finalizado el proceso de desarrollo, se establece la necesidad de realizar pruebas de carga en *Cloud Run* para evaluar el tiempo de respuesta de las consultas, la cantidad de instancias que se crean y la presencia de errores. Esto plantea un nuevo desafío debido al desconocimiento de esta herramienta. Por lo tanto, se destinan dos días para familiarizarse con ella y estudiar cómo llevar a cabo las tareas requeridas.
Finalmente, se logra aprender cómo crear una imagen del programa, subirla a *Cloud Run* y realizar las pruebas de carga. Durante este proceso, se analizan gráficas y los números obtenidos para evaluar el rendimiento del programa. Como resultado, el programa creado supera con éxito las pruebas de carga.

5.3. Admin CRUD

Este proyecto implica crear una interfaz de administración de tablas utilizando *React*, *Material UI* y *Refine*, una herramienta para desarrollar interfaces interactivas en aplicaciones web.
En el primer día, se asigna la tarea de leer y seguir el tutorial en la página web de *Refine* para familiarizarse con la herramienta. Al día siguiente, se reciben las instrucciones sobre las tablas a manejar y se inicializa la base de datos para probar los cambios implementados. A lo largo de febrero, se trabaja exclusivamente en el CRUD de dos de las cuatro tablas disponibles, debido a que el backend de las otras dos tablas se encontraba aún en desarrollo.
Con los objetivos y herramientas claras, se comienza con el desarrollo del proyecto, que ya contaba con código relacionado con la conexión a la API y el funcionamiento básico de la aplicación. El progreso es rápido gracias a la utilidad de *Refine*, aunque surgen dificultades, especialmente en la implementación de cambios estéticos solicitados. No obstante, dichas dificultades resultaron relativamente fáciles de resolver, ya que muchas contaban una solución detallada en *Google*.
Después de completar el CRUD de las tablas, el tutor solicita agregar un historial de cambios, aplicable a todas las tablas, por lo cual debía ser modularizado. Se especifica que este historial debe mostrar una versión a la izquierda y otra a la derecha, permitiendo comparar distintas versiones para cada entidad en la tabla.
Teniendo en cuenta la solicitud, se realiza una búsqueda de librerías que permitieran implementar esta funcionalidad, encontrando una que parecía adecuada, sin embargo, resultó más compleja de lo necesario. Se opta entonces por *react-diff-viewer*, que cuenta con documentación y ejemplos claros. Sin embargo, surgen dificultades al implementar la selección de versiones para comparar, las cuales fueron solucionadas consultando con el resto de las practicantes sobre cuál sería la manera más intuitiva de seleccionar versiones e implementando sus sugerencias. Asimismo, resulta más difícil de lo esperado hacer la conexión con la API para obtener el historial de la base de datos, por lo cual, luego de buscar soluciones sin éxito, se consulta al tutor, quien al final del día proporciona una respuesta satisfactoria. Con estos problemas resueltos, se avanza sin contratiempos en la implementación, logrando completar el trabajo una semana antes de lo previsto.

6. Reflexión

Durante el periodo de prácticas, se enfrentaron diversos obstáculos tanto en el ámbito laboral como en las interacciones humanas. En el ámbito laboral, la mayoría de las dificultades estuvieron relacionadas con el manejo de herramientas y el intento de minimizar las consultas para mejorar la habilidad de encontrar soluciones de manera individual. Sin embargo, un conflicto destacable surgió al recibir el primer sueldo: Se esperaba un pago de cuatrocientos mil pesos chilenos, pero se solicitó hacer la boleta por un sueldo menor, considerando ciertos descuentos. Esto no fue aceptado por ninguna de las practicantes, ya que no coincidía con lo pactado inicialmente. Ante esta situación, se decidió abordar el problema enviando un correo a recursos humanos, expresando la preocupación por los descuentos y la sensación de injusticia, asumiendo toda la responsabilidad del mensaje para no afectar negativamente al resto de las practicantes. Tras conversaciones entre el encargado de recursos humanos y el supervisor, se aclaró que efectivamente se había cometido un error, y al final todo se resolvió como se esperaba.

A pesar de los obstáculos relacionados con el manejo de herramientas, que se anticipaban antes de comenzar la práctica, la empresa y el trabajo realizado cumplieron con las expectativas. Especialmente significativo es que, según la descripción de la postulación, no se requería un conocimiento previo de todas las herramientas necesarias, sino la disposición de aprender, para lo cual resultaron bastante útiles las habilidades adquiridas durante ciertos cursos de la carrera, como Ingeniería de Software (I y II) y Metodologías de Diseño y Programación.

La inserción dentro de la empresa resulta amena y acogedora, ya que desde el primer día se experimenta un cálido recibimiento por parte del equipo. Se percibe un ambiente colaborativo y respetuoso, donde cada integrante, sin importar su posición, es valorado de igual manera, fomentando una relación horizontal entre los miembros del equipo y facilitando la comunicación. Además, trabajar con más practicantes del género femenino, resultó motivador debido a la diversidad que aportaron al equipo. Este cambio en el entorno permitió tener conversaciones más variadas y una dinámica de trabajo más inclusiva. La presencia de más mujeres en el equipo ofreció una perspectiva diferente respecto de experiencias previas en el entorno universitario, donde los grupos de trabajo se componen de hombres en su mayoría, promoviendo el sentido de pertenencia. Desde el comienzo de la práctica hacia el final de la misma, la relación con el resto de las practicantes pasó de ser netamente laboral a una amistad.

Respecto de la puntualidad, no hubo ningún día en que se llegara tarde, incluso se llegó varios días antes que todo el resto, principalmente porque el vivir en la periferia lleva a ser más precavida con los horarios.

En el contexto de resolución de conflictos interpersonales se destaca la capacidad y "valentía" de comentar situaciones potencialmente incómodas para las practicantes, como el asunto de la boleta, ya mencionado, o una situación ocurrida, donde durante las primeras reuniones de equipo (*stand-up* con cámara) se reían visiblemente, debido al envío de mensajes entre ellos mediante un chat interno, a lo cual se reaccionó enviando un mensaje a uno de los tutores en busca de aclarar si los motivos de su risa eran de burla hacia las practicantes o si era alguna situación no relacionada, logrando solucionar la confusión y provocar que se dejara de hacer.

Es importante reconocer ciertas áreas de mejora identificadas durante el período de prácticas. Por ejemplo,

se detectó una tendencia a reaccionar de manera rápida frente a situaciones que parecían injustas, lo que llevó a recibir el apodo de “reclamona”, por lo cual se debe trabajar en la reactividad frente a distintas situaciones, con tal de mantener un ambiente ameno. Asimismo, se reconoce que podría haberse invertido más tiempo en la investigación de las herramientas utilizadas y en comprender la utilidad real de los proyectos asignados, siempre manteniendo en mente las necesidades del usuario final. Es por esto que a futuro se busca tener más en cuenta dichas falencias, que afortunadamente en esta ocasión lograron ser compensadas gracias al apoyo continuo de tutores y compañeras.

7. Conclusiones

Analizando las secciones anteriores, se aprecia que se ha logrado alcanzar el objetivo general establecido para el período de prácticas. Sin embargo, es esencial reflexionar sobre la perspectiva presentada por Synaptic, donde se enfatiza que el verdadero propósito de aceptar practicantes no radica en aliviar la carga laboral del equipo, sino en brindarles una experiencia que les permita desarrollarse como ingenieros competentes en el futuro, capaces de resolver problemas de forma autónoma y eficaz.

En este contexto, se plantea la posibilidad de que la sección donde se menciona el objetivo de "aliviar la carga del equipo de trabajo" no se haya cumplido completamente. Esto se debe a que, además de realizar sus propias tareas, cada tutor debía estar disponible para atender las consultas y dudas planteadas por las estudiantes en práctica, lo que implica una responsabilidad adicional.

A pesar de este aspecto, se destaca el logro de cumplir eficientemente con todas las tareas asignadas, aunque no de manera impecable. Este proceso implicó aprender a trabajar con nuevas herramientas en un tiempo limitado y en áreas donde la experiencia previa no era necesariamente un punto fuerte, como el frontend y backend.

Además, es importante destacar el trabajo y apoyo brindado por Synaptic durante todo el período de práctica. Desde el inicio, la empresa se esfuerza por hacer que las estudiantes se sientan parte integral del equipo, promoviendo los almuerzos en conjunto y facilitando conversaciones que van más allá de lo estrictamente profesional. Además, Synaptic proporciona herramientas de primera calidad y está siempre disponible para resolver cualquier problema que pueda surgir durante el desarrollo de las tareas asignadas. Este respaldo constante y el ambiente inclusivo creado por la empresa contribuyen a la satisfacción de las practicantes durante su experiencia en la organización.

Finalmente, se reconoce un sentido marcado de justicia que no había sido identificado anteriormente. Este descubrimiento resulta ser una grata sorpresa, especialmente considerando la importancia fundamental de la ética en el campo de la ingeniería.

8. Anexos

En esta sección se incluyen segmentos de código que podrían aclarar ciertas dudas o brindar cierto contexto de las soluciones a quien lo requiriera.
Por cada problemática se mostrará una imagen donde se puede observar la estructura del código y a continuación secciones de código y su respectiva ubicación dentro de esta estructura.

8.1. Construcción de Componentes

Los resultados de esta sección se pueden observar en el *Strorybook de Boufin*.

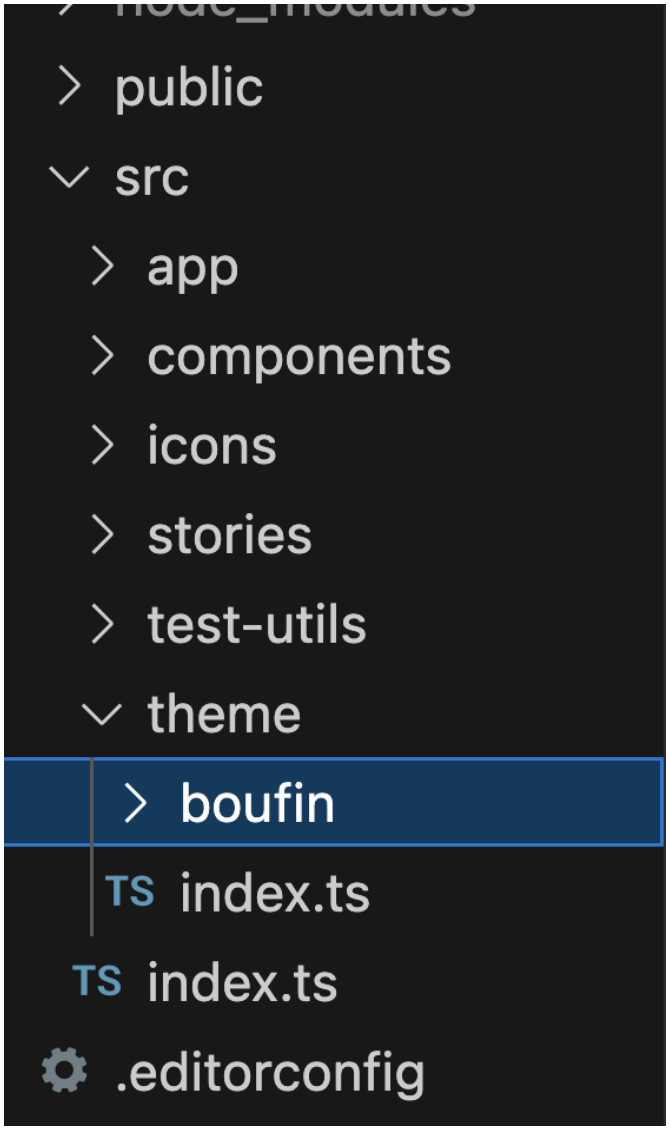

Figura 4: Estructura Componentes

Código 1: src/components/Checkbox/Checkbox.tsx

```tsx
export enum variants {
  GENERIC = 'generic'
}

export enum icons {
  CHECKED = 'checked',
  DASHED = 'dashed'
}

export type CheckboxProps = {
  label?: string;
  isDisabled?: boolean;
  icon?: icons;
};

const CustomIcons = ({ isChecked, icon }: { isChecked?: boolean; icon: string
    ↪ }) => {
  const iconMaps: { [key: string]: JSX.Element } = {
    checked: <Icon as={IconCheck} boxSize="18px" />,
    dashed: <Icon as={IconCircleDashed} boxSize="18px" />
  };
  return <>{isChecked && !!iconMaps[icon] ? iconMaps[icon] : null}</>;
};

export const Checkbox = forwardRef<HTMLInputElement, CheckboxProps>(
  ({ label, isDisabled = false, icon = 'checked', ...rest }, ref) => {
    const styles = useStyleConfig('Checkbox');
    return (
      <ChakraCheckbox
        data-testid="checkbox-testid"
        icon={<CustomIcons icon={icon} />}
        isDisabled={isDisabled}
        ref={ref}
        sx={styles}
        variant={variants.GENERIC}
        {...rest}
      >
        <span id="checkbox-test-label">{label}</span>
      </ChakraCheckbox>
    );
  }
);
```

A continuación se presenta el código de testeo del componente, donde se le “saca una foto” al componente actual y se compara con “fotos” anteriores, para ver los cambios aplicados.

Código 2: src/components/Checkbox/Checkbox.spec.tsx

```tsx
import renderer from 'react-test-renderer';
import { createSerializer } from '@emotion/jest';
import TestProvider from '../../test-utils/testProvider';
import { Checkbox, CheckboxIcons } from './index';

expect.addSnapshotSerializer(createSerializer());

it('renders Checkbox component with boufinTheme', () => {
  const component = renderer
    .create(
      <TestProvider>
        <Checkbox label="testing" isDisabled icon={CheckboxIcons.CHECKED} />
      </TestProvider>
    )
    .toJSON();
  expect(component).toMatchSnapshot();
});
```

El código 3 muestra el trabajo realizado para ver reflejados los cambios en *StoryBook*.

Código 3: src/components/Checkbox/Checkbox.stories.tsx

```tsx
import type { Meta, StoryObj } from '@storybook/react';
import { Checkbox } from './Checkbox';

const meta = {
  title: 'Boufin/Checkbox',
  component: Checkbox,
  parameters: {
    layout: 'centered'
  },
  tags: ['autodocs'],
  argTypes: {
    label: {
      description: 'Checkbox label',
      control: 'text'
    },
    isDisabled: {
      description: 'Checkbox disabled',
      control: 'boolean'
```

```
    }
  }
} satisfies Meta<typeof Checkbox>;

export default meta;
type Story = StoryObj<typeof Checkbox>;

export const Primary: Story = {
  args: {
    label: 'checkbox',
    isDisabled: false
  }
};
```

El siguiente código (Código 4) es el utilizado para los componentes de *search bar* y *textField*. Luego, en el Código 5 se observan las distintas variantes utilizadas.
Los cambios realizados al componente *Input* como tal, es decir, los cambios útiles tanto para *search bar* como para *textField*, se observan el el Código 6.

Código 4: src/components/Input/Input.tsx

```
import React, { forwardRef, useState } from 'react';
import {
  Input as ChakraInput,
  InputGroup as ChakraInputGroup,
  InputLeftElement as ChakraInputLeftElement,
  InputRightElement as ChakraInputRightElement,
  List as ChakraList,
  ListItem as ChakraListItem,
  FormControl as ChakraFormControl,
  FormLabel as ChakraFormLabel,
  FormErrorMessage as ChakraFormErrorMessage,
  useStyleConfig
} from '@chakra-ui/react';
import { IconSearch, IconLock, IconX, IconKey, IconAlertCircle } from '
    ↪ @tabler/icons-react';

export enum variants {
  SEARCH = 'SearchComponent',
  TEXTFIELD = 'TextFieldComponent'
}

export type InputProps = {
  label?: string;
```

```
  isDisabled?: boolean;
  variant?: variants;
  data?: string[];
  keyIconVisible?: boolean;
  isError?: boolean;
  errorMessage?: string;
};

export const Input = forwardRef<HTMLInputElement, InputProps>(
  (
    {
      label,
      isDisabled,
      variant,
      data = [],
      keyIconVisible = true,
      isError,
      errorMessage,
      ...rest
    },
    ref
  ) => {
    const styles = useStyleConfig('Input', { variant });
    const [inputValue, setInputValue] = useState('');
    const [isHovered, setIsHovered] = useState(false);
    const [filteredData, setFilteredData] = useState<string[]>([]);
    const [isVisible, setVisible] = useState(true);

    const handleListItemClick = () => {
      setVisible(false);
    };

    const handleInputChange = (e: React.ChangeEvent<HTMLInputElement>) => {
      setInputValue(e.target.value);
      const searchWord = e.target.value;
      const newFilter = data.filter((value: string) => {
        return value.toLowerCase().includes(searchWord.toLowerCase());
      });
      if (searchWord == '') {
        setFilteredData([]);
      } else {
        setFilteredData(newFilter);
      }
    };
```

```
67
68    const handleRecommendationSelect = (selectedValue: string) => {
69      const newInput = selectedValue;
70      setInputValue(newInput);
71    };
72
73    const clearInput = () => {
74      setInputValue('');
75      setFilteredData([]);
76    };
77    const handleMouseEnter = () => setIsHovered(true);
78    const handleMouseLeave = () => setIsHovered(false);
79
80    if (variant === variants.TEXTFIELD) {
81      return (
82        <ChakraFormControl isInvalid={isError}>
83          <ChakraFormLabel color="var(--chakra-colors-neutral-800)"
    ↪ pointerEvents="none">
84            {label}
85          </ChakraFormLabel>
86          <ChakraInputGroup onMouseEnter={handleMouseEnter} onMouseLeave={
    ↪ handleMouseLeave}>
87            <ChakraInput
88              value={inputValue}
89              sx={styles}
90              variant="secondary"
91              placeholder="Placeholder"
92              isDisabled={isDisabled}
93              onChange={handleInputChange}
94            />
95            <ChakraInputRightElement>
96              {isDisabled ? (
97                <IconLock color="var(--chakra-colors-neutral-500)" />
98              ) : (
99                !isDisabled &&
100               keyIconVisible &&
101               !inputValue && <IconKey color="var(--chakra-colors-neutral
    ↪ -500)" />
102             )}
103             {inputValue && isHovered && !isDisabled && (
104               <IconX onClick={clearInput} color="var(--chakra-colors-
    ↪ neutral-700" />
105             )}
106             {isError && <IconAlertCircle color="var(--chakra-colors-danger
```

```
↪ -500)" />}
107             </ChakraInputRightElement>
108           </ChakraInputGroup>
109
110           <ChakraFormErrorMessage>
111             <IconAlertCircle color="var(--chakra-colors-danger-500)" width="1
↪ rem" height="1rem" />
112             <span style={{ marginLeft: '0.25rem' }}>{errorMessage}</span>
113           </ChakraFormErrorMessage>
114         </ChakraFormControl>
115       );
116     }
117
118     return (
119       //TODO:when clicked outside the chakraInputGroup component (onBlur),
↪ set the chakraList component
120       //visibility to false, making sure that handleRecommendationSelect
↪ still works.
121       <ChakraInputGroup
122         onMouseEnter={handleMouseEnter}
123         onMouseLeave={handleMouseLeave}
124         sx={styles}
125         variant="primary"
126         display="inline-block"
127         ref={ref}
128         {...rest}
129       >
130         <ChakraInputLeftElement>
131           <IconSearch />
132         </ChakraInputLeftElement>
133         <ChakraInput
134           value={inputValue}
135           placeholder="Placeholder"
136           isDisabled={isDisabled}
137           onChange={handleInputChange}
138         />
139         {inputValue && isHovered && !isDisabled && (
140           <ChakraInputRightElement>
141             <IconX onClick={clearInput} color="var(--chakra-colors-neutral
↪ -700)" />
142           </ChakraInputRightElement>
143         )}
144         {isDisabled && (
145           <ChakraInputRightElement>
```

```
            <IconLock color="var(--chakra-colors-neutral-500)" />
          </ChakraInputRightElement>
        )}
        {filteredData.length !== 0 && !isDisabled && isVisible && (
          <ChakraList variant="primary">
            {filteredData.slice(0, 6).map((value, index) => {
              return (
                <ChakraListItem
                  onClick={() => {
                    handleRecommendationSelect(value);
                    handleListItemClick();
                  }}
                  key={index}
                >
                  {value}
                </ChakraListItem>
              );
            })}
          </ChakraList>
        )}
      </ChakraInputGroup>
    );
  }
);
Input.displayName = 'Input';
export default Input;
```

Código 5: src/theme/boufin/components/Input/variants.ts

```
const search = {
  field: {
    color: 'var(--chakra-colors-neutral-700)',
    bg: 'var(--chakra-colors-neutral-100)',
    borderColor: 'var(--chakra-colors-primary-500)',
    width: '21.5rem',
    height: '3rem',
    _disabled: {
      pointerEvents: 'none',
      bg: 'var(--chakra-colors-neutral-100)',
      color: 'var(--chakra-colors-neutral-400)',
      opacity: 1,
      _placeholder: {
        color: 'var(--chakra-colors-neutral-400)'
      },
```

```
      cursor: 'not-allowed'
    },
    _focus: {
      bg: 'white',
      borderWidth: '0.0625rem',
      borderColor: 'var(--chakra-colors-primary-500)',
      caretColor: 'rgba(66, 97, 255, 1)'
    },
    _hover: {
      bg: 'white',
      borderColor: 'var(--chakra-colors-primary-500)',
      borderWidth: '0.0625rem'
    }
  },
  element: {
    color: 'var(--chakra-colors-neutral-500)'
  }
};

const textfield = {
  field: {
    color: 'var(--chakra-colors-neutral-700)',
    bg: 'var(--chakra-colors-neutral-100)',
    borderRadius: '0.5rem',
    width: '21.5rem',
    height: '3rem',
    borderColor: 'var(--chakra-colors-primary-500)',
    _placeholder: {
      color: 'var(--chakra-colors-neutral-500)'
    },
    _hover: {
      bg: 'var(--chakra-colors-neutral-100)',
      borderColor: 'var(--chakra-colors-primary-500)',
      borderWidth: '0.0625rem'
    },
    _focus: {
      bg: 'var(--chakra-colors-neutral-100)',
      borderWidth: '0.0625rem',
      borderColor: 'var(--chakra-colors-primary-500)',
      caretColor: 'rgba(66, 97, 255, 1)'
    },
    _disabled: {
      pointerEvents: 'none'
    },
```

```
    _invalid: {
      borderColor: 'var(--chakra-colors-danger-500)',
      borderWidth: '0.0625rem',
      _hover: {
        borderColor: 'var(--chakra-colors-danger-500)'
      }
    }
  }
};

const variants = {
  primary: search,
  secondary: textfield
};

export default variants;
```

Código 6: src/theme/boufin/components/Input/index.ts

```
import { ComponentStyleConfig } from '@chakra-ui/react';
import '@fontsource/open-sans';
import variants from './variants';

const InputStyle: ComponentStyleConfig = {
  baseStyle: {
    ul: {
      bg: 'white',
      marginTop: '0.5rem',
      maxHeight: '16rem',
      width: '21.5rem',
      borderRadius: '0.5rem',
      boxShadow: '0 0.25rem 1.5rem 0 rgba(0, 0, 0, 0.12)',
      padding: '0.5rem',
      overflow: 'hidden',
      overflowY: 'auto'
    },
    li: {
      padding: '0.875rem 0rem 0.875rem 0.75rem',
      height: '3rem',
      color: 'var(--chakra-colors-neutral-700)',
      _hover: {
        bg: 'var(--chakra-colors-neutral-100)',
        borderRadius: '0.5rem'
      }
```

```
    },
    fontFamily: "'Open Sans', sans-serif",
    '&:has(input:focus)': {
      '& > div': {
        color: 'var(--chakra-colors-primary-500)'
      }
    },
    '&:has(input:disabled)': {
      '& > div': {
        color: 'var(--chakra-colors-neutral-400)',
        cursor: 'not-allowed'
      }
    },
    field: {
      borderRadius: '0.5rem'
    },
    element: {
      height: '3rem'
    }
  },
  variants: {
    ...variants
  }
};

export default InputStyle;
```

8.2. Reemplazo de Almacenamiento Caché

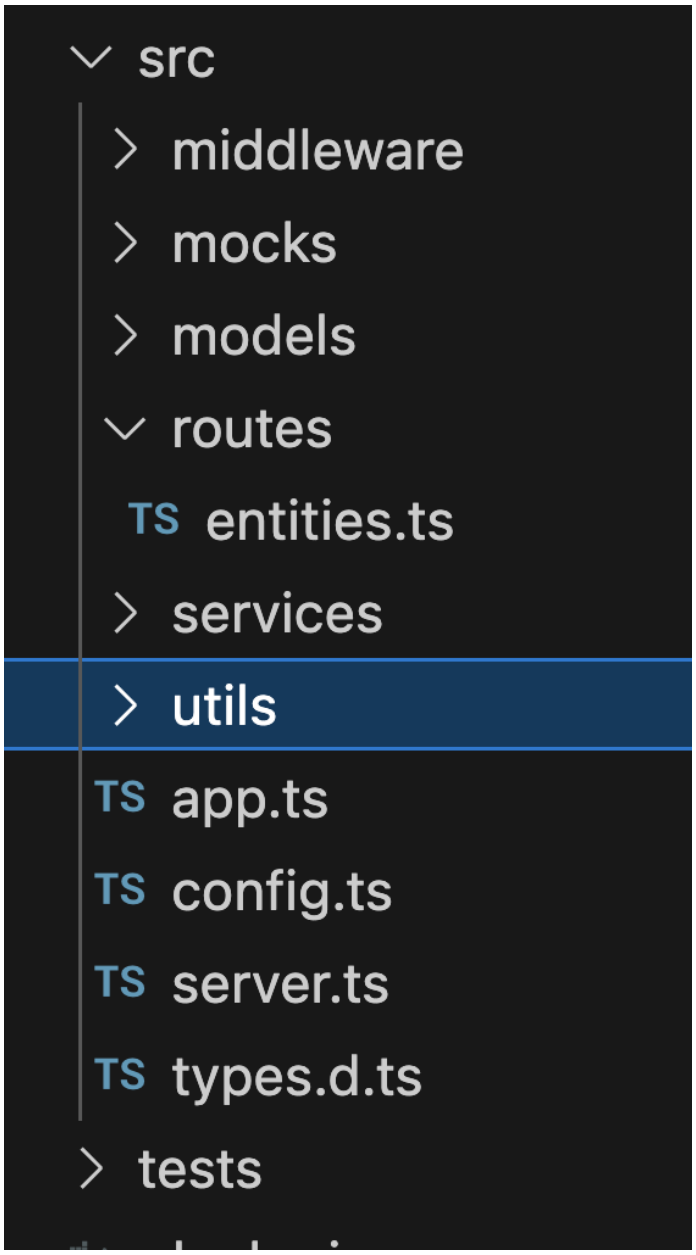

Figura 5: Estructura Caché

En el Código 7 se muestra el manejo de las solicitudes, en el 8 se aplica el patrón *strategy* y en el 9 se definen los métodos de la librería *flatCache* que se utilizarán.

Código 7: src/routes/entities.ts

```
1 import crypto from 'crypto';
```

```
import express from 'express';
import { checkApiKey } from '../middleware';
import { Entity, makePublicEntityFromEntity } from '../models';
import { EntitiesService } from '../services';
import { filterAllowedEntities } from '../utils';
import { isAllowedToSee } from '../utils/entities';
import { FEATURE_FLAG_MEMORY_CACHE } from '../config';
import CacheManager from '../utils/cache';

const router = express.Router();
router.use(checkApiKey);

const hash = (key: string) => crypto.createHash('sha256').update(key).digest(
    ↪ 'base64');

router.get('/', async (req, res) => {
  try {
    const cache = new CacheManager(FEATURE_FLAG_MEMORY_CACHE);
    const apiKeyPermissions = req.apiKey?.permissions || [];
    if (apiKeyPermissions.length === 0) return res.status(404).send();

    let entities: Entity[] = [];
    const key = 'public_entities' + hash(apiKeyPermissions.sort().join(','));

    const entitiesExist = await cache.exists(key);
    let entitiesCache: Entity[] | undefined;
    if (entitiesExist) {
      try {
        const entitiesFromCache = await cache.get(key);
        if (entitiesFromCache && typeof entitiesFromCache === 'string') {
          entitiesCache = JSON.parse(entitiesFromCache);
        }
      } catch (e) {
        console.error(e);
      }
    }

    if (entitiesCache && Array.isArray(entitiesCache) && entitiesCache.length
    ↪  > 0) {
      entities = entitiesCache;
    } else {
      console.log('get all entities from Service');
      entities = await EntitiesService.listEntities();
      if (entities.length > 0) {
```

```
        await cache.set(key, JSON.stringify(entities));
      }
    }
    return res.status(200).json(filterAllowedEntities(entities,
    ↪ apiKeyPermissions));
  } catch (e) {
    console.error(e);
    return res.status(500).end();
  }
});

router.get('/:entityId', async (req, res) => {
  try {
    const apiKeyPermissions = req.apiKey?.permissions || [];
    if (apiKeyPermissions.length === 0) return res.status(404).send();
    const entityId = req.params.entityId;

    let entity: Entity | undefined = undefined;
    console.log('get entity from Service');
    entity = await EntitiesService.retrieveEntity(entityId);

    if (!entity || !isAllowedToSee(entity, apiKeyPermissions)) {
      return res.status(404).send();
    }
    return res.status(200).json(makePublicEntityFromEntity(entity, new Set(
    ↪ apiKeyPermissions)));
  } catch (e) {
    console.error(e);
    return res.status(500).end();
  }
});

export default router;
```

Código 8: src/utils/cache.ts

```
import { Cache } from './flatCache';
import { Tedis } from 'tedis';
import { REDIS_HOST, REDIS_PORT, CACHE_EXPIRE, CACHE_PATH } from '../config';

interface CacheStrategy {
  get(key: string): Promise<string | number | null>;
  set(key: string, value: string): Promise<void>;
  exists(key: string): Promise<boolean>;
```

```
}

class RedisCache implements CacheStrategy {
  private tedis: Tedis;
  constructor(tedis: Tedis) {
    this.tedis = tedis;
  }

  async get(key: string): Promise<string | number | null> {
    console.log('get from Redis');
    const result = await this.tedis.get(key);
    return result;
  }

  async set(key: string, value: string): Promise<void> {
    console.log('set to Redis');
    await this.tedis.setex(key, CACHE_EXPIRE, value);
  }

  async exists(key: string): Promise<boolean> {
    return (await this.tedis.exists(key)) === 1;
  }
}

class FlatCache implements CacheStrategy {
  private cache: Cache;
  constructor(cache: Cache) {
    this.cache = cache;
  }

  async get(key: string): Promise<string | number | null> {
    console.log('get from flatCache');
    return this.cache?.getKey(key);
  }

  async set(key: string, value: string): Promise<void> {
    console.log('set to flatCache');
    this.cache.setKey(key, value);
    this.cache.save();
  }

  async exists(key: string): Promise<boolean> {
    const value = this.cache.getKey(key);
    return Promise.resolve(value !== undefined && value !== null);
```

```
  }
}

class CacheManager {
  private cacheStrategy: CacheStrategy;

  constructor(useMemoryCache: boolean) {
    if (!useMemoryCache) {
      const tedis = new Tedis({
        port: parseInt(REDIS_PORT),
        host: REDIS_HOST
      });
      tedis.on('connect', () => {
        console.log('connect');
      });
      tedis.on('timeout', () => {
        console.error('timeout');
      });
      tedis.on('error', (err) => {
        console.error('error', err);
      });
      tedis.on('close', (had_error) => {
        console.log('close with err: ', had_error);
        throw new Error('tedis connection closed');
      });

      this.cacheStrategy = new RedisCache(tedis);
    } else {
      const cache = new Cache('cacheJSON', CACHE_PATH, CACHE_EXPIRE);
      this.cacheStrategy = new FlatCache(cache);
    }
  }

  public async get(key: string): Promise<string | number | null> {
    return this.cacheStrategy.get(key);
  }

  public async set(key: string, value: string): Promise<void> {
    return this.cacheStrategy.set(key, value);
  }

  public async exists(key: string): Promise<boolean> {
    return this.cacheStrategy.exists(key);
  }
```

```
}

export default CacheManager;
```

Código 9: src/utils/flatCache.ts

```
import flatCache from 'flat-cache';
import { CACHE_EXPIRE } from '../config';

class Cache {
  name: string;
  path: string;
  cache: flatCache.Cache;
  expire: number | false;
  constructor(name: string, path: string, cacheTime = 0) {
    this.name = name;
    this.path = path;
    this.cache = flatCache.load(name, path);
    this.expire = cacheTime === 0 ? false : (cacheTime = CACHE_EXPIRE * 1000)
    ↪ ; //seconds to miliseconds
  }

  getKey(key: string) {
    const now = new Date().getTime();
    const value = this.cache.getKey(key);
    if (value === undefined || (value.expire !== false && value.expire < now)
    ↪ ) {
      return null;
    } else {
      return value.data;
    }
  }

  setKey(key: string, value: string) {
    const now = new Date().getTime();
    this.cache.setKey(key, {
      expire: this.expire === false ? false : now + this.expire,
      data: value
    });
  }

  removeKey(key: string) {
    this.cache.removeKey(key);
  }
```

```
37
38    save() {
39      this.cache.save(true);
40    }
41
42    remove() {
43      flatCache.clearCacheById(this.name, this.path);
44    }
45  }
46
47  export { Cache };
```

8.3. Admin CRUD

En esta sección se mostrará el trabajo realizado para una de las dos tablas, ya que, con excepción de algunas diferencias inherentes a la naturaleza de los datos de cada tabla, son bastante similares.

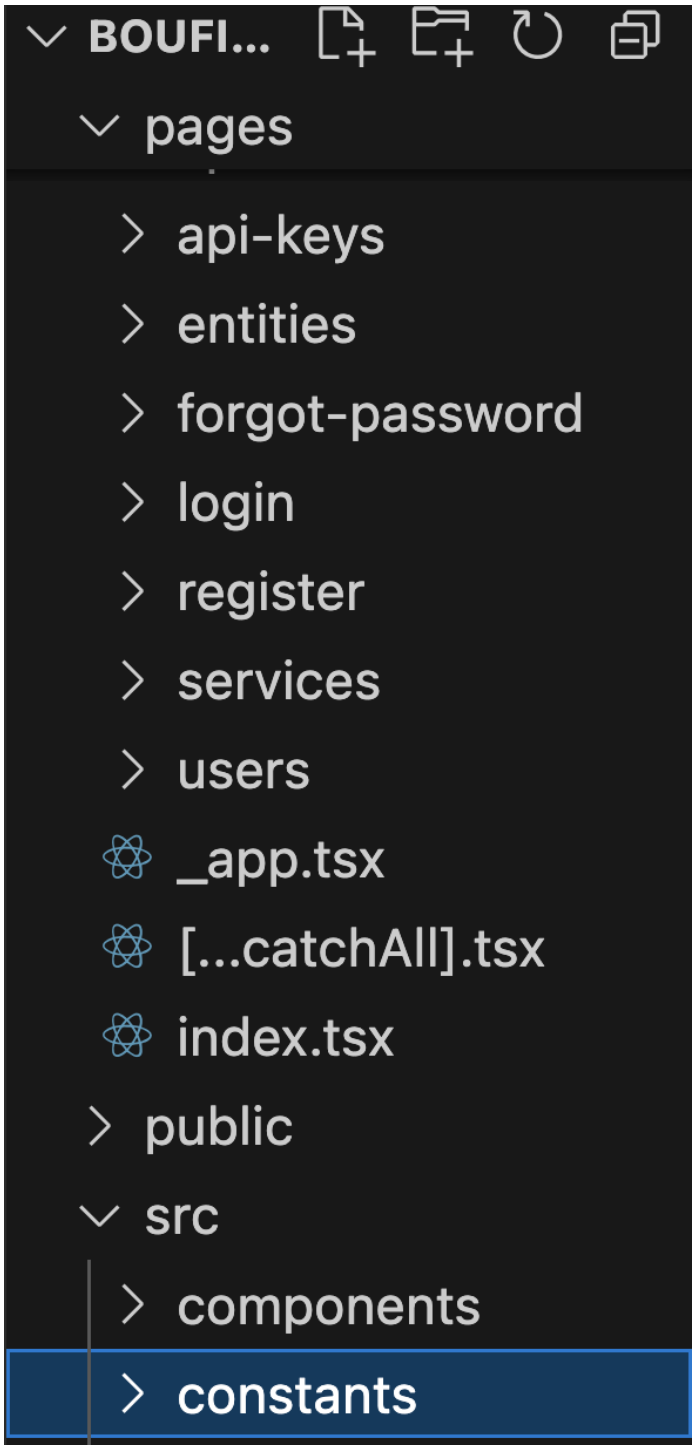

Figura 6: Estructura Admin

En el código a continuación es posible observar cómo fueron creadas las páginas de creación (Código 10), edición (Código 11), lectura (Código 12) y lectura del historial (Código 13), junto con el código donde se define la tabla y se agrega una sección (*active*) donde se puede cambiar un estado de la entidad (Código

14). En el Código 15 es posible observar cómo se define el historial que será utilizado en cada una de las tablas. Finalmente, en el Código 16 se muestra cómo se definen las páginas disponibles para cada una de las tablas.

Código 10: pages/users/create/index.tsx

```tsx
import { GetServerSideProps } from "next";
import { authProvider } from "@providers/authProvider";
import { IResourceComponentsProps } from "@refinedev/core";
import { useForm } from "@refinedev/react-hook-form";
import { Create } from "@refinedev/mui";
import { Box, Checkbox, FormControlLabel, MenuItem, TextField } from "@mui/
    ↪ material";
import { Controller } from "react-hook-form";
import { planOptions } from "@utils/dataOptions";

const UserCreate: React.FC<IResourceComponentsProps> = () => {
  const {
      saveButtonProps,
      refineCore: { formLoading },
      register,
      control,
      formState: { errors },
  } = useForm();

  return (
    <Create isLoading={formLoading} saveButtonProps={saveButtonProps}>
      <Box component="form" sx={{ display: 'flex', flexDirection: 'column' }}
    ↪  autoComplete="off">
        <TextField
          {...register('name', {
            required: 'This field is required'
          })}
          error={!!(errors as any)?.name}
          helperText={(errors as any)?.name?.message}
          margin="normal"
          fullWidth
          InputLabelProps={{ shrink: true }}
          type="text"
          label="Name"
          name="name"
        />
        <TextField
          {...register('email', {
```

```
            required: 'This field is required',
            pattern: {
              value: /^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$/i,
              message: 'Invalid email address',
            },
          })}
          error={!!(errors as any)?.email}
          helperText={(errors as any)?.email?.message}
          margin="normal"
          fullWidth
          InputLabelProps={{ shrink: true }}
          type="email"
          label="Email"
          name="email"
        />
        <Controller
          control={control}
          name="active"
          defaultValue={true}
          render={({ field }) => (
            <FormControlLabel
              label="Active"
              control={
                <Checkbox
                  {...field}
                  checked={field.value}
                  onChange={(event) => {
                    field.onChange(event.target.checked);
                  }}
                />
              }
            />
          )}
        />
        <TextField
          select
          {...register('plan', {
            required: 'This field is required'
          })}
          defaultValue="free"
          error={!!(errors as any).plan}
          helperText={(errors as any)?.plan?.message}
          margin="normal"
          fullWidth
```

```
          InputLabelProps={{ shrink: true }}
          label="Plan"
          name="plan"
        >
          {planOptions.map((option) => (
            <MenuItem value={option.toLowerCase()}>{option}</MenuItem>
          ))}
        </TextField>
      </Box>
    </Create>
  );
};


export const getServerSideProps: GetServerSideProps<{}> = async (context) =>
  ↪ {
  const { authenticated, redirectTo } = await authProvider.check(context);

  if (!authenticated) {
    return {
      props: {},
      redirect: {
        destination: `${redirectTo}?to=${encodeURIComponent("/users")}`,
        permanent: false,
      },
    };
  }
  return {
    props: {},
  };
};

export default UserCreate;
```

Código 11: pages/users/edit/[id].tsx

```
import { GetServerSideProps } from "next";
import { authProvider } from "@providers/authProvider";
import { IResourceComponentsProps } from "@refinedev/core";
import { useForm } from "@refinedev/react-hook-form";
import { Edit } from "@refinedev/mui";
import { Box, Checkbox, FormControlLabel, MenuItem, TextField } from "@mui/
  ↪ material";
import { Controller } from "react-hook-form";
```

```
import { planOptions } from "@utils/dataOptions";

const UserEdit: React.FC<IResourceComponentsProps> = () => {
  const {
      saveButtonProps,
      register,
      control,
      formState: { errors },
  } = useForm();


  return (
    <Edit saveButtonProps={saveButtonProps} canDelete={false}>
      <Box component="form" sx={{ display: 'flex', flexDirection: 'column' }}
    ↪  autoComplete="off">
        <TextField
          {...register('name', {
            required: 'This field is required'
          })}
          error={!!(errors as any)?.name}
          helperText={(errors as any)?.name?.message}
          margin="normal"
          fullWidth
          InputLabelProps={{ shrink: true }}
          type="text"
          label="Name"
          name="name"
        />
        <TextField
          {...register('email', {
            required: 'This field is required'
          })}
          error={!!(errors as any)?.email}
          helperText={(errors as any)?.email?.message}
          margin="normal"
          fullWidth
          InputLabelProps={{ shrink: true }}
          type="email"
          label="Email"
          name="email"
        />
        <Controller
          control={control}
          name="active"
```

```
          defaultValue={null}
          render={({ field }) => (
            <FormControlLabel
              label="Active"
              control={
                <Checkbox
                  {...field}
                  checked={field.value}
                  onChange={(event) => {
                    field.onChange(event.target.checked);
                  }}
                />
              }
            />
          )}
        />
        <Controller
          name="plan"
          control={control}
          defaultValue={''}
          render={({ field }) => (
            <TextField
              {...field}
              select
              {...register('plan', {
                required: 'This field is required'
              })}
              defaultChecked={field.value}
              defaultValue={field.value}
              error={!!errors.plan}
              helperText={(errors as any)?.plan?.message}
              margin="normal"
              fullWidth
              InputLabelProps={{ shrink: true }}
              label="Plan"
              name="plan"
            >
              {planOptions.map((option) => (
                <MenuItem value={option.toLowerCase()}>{option}</MenuItem>
              ))}
            </TextField>
          )}
        />
        <TextField
```

```
          {...register('creditsPerMonth', {
            required: 'This field is required',
            valueAsNumber: true
          })}
          error={!!(errors as any)?.creditsPerMonth}
          helperText={(errors as any)?.creditsPerMonth?.message}
          margin="normal"
          fullWidth
          InputLabelProps={{ shrink: true }}
          type="number"
          label="Credits Per Month"
          name="creditsPerMonth"
        />

        <TextField
          {...register('id', {
            required: 'This field is required'
          })}
          error={!!(errors as any)?.id}
          helperText={(errors as any)?.id?.message}
          margin="normal"
          fullWidth
          InputLabelProps={{ shrink: true }}
          type="text"
          label="Id"
          name="id"
          disabled
        />
      </Box>
    </Edit>
  );
};

export const getServerSideProps: GetServerSideProps<{}> = async (context) =>
    ↪ {
  const { authenticated, redirectTo } = await authProvider.check(context);

  if (!authenticated) {
    return {
      props: {},
      redirect: {
        destination: `${redirectTo}?to=${encodeURIComponent("/users")}`,
        permanent: false,
      },
```

```
    };
  }

  return {
    props: {},
  };
};

export default UserEdit;
```

Código 12: pages/users/show/[id].tsx

```
import { useShow, IResourceComponentsProps, useOne } from "@refinedev/core";
import { BooleanField, DateField, EmailField, NumberField, Show,
    ↪ TextFieldComponent as TextFieldOnlyView } from "@refinedev/mui";
import { Typography, Stack, Button } from "@mui/material";

import { GetServerSideProps } from "@interfaces/next";
import { authProvider } from "@providers";

export const UsersById: React.FC<IResourceComponentsProps> = () => {
  const { queryResult } = useShow();
  const { data, isLoading } = queryResult;

  const record = data?.data ?? {};

  return (
    <Show isLoading={isLoading} canDelete={false} canEdit={true}
    ↪ headerButtons={({ defaultButtons }) => (
      <>
        {defaultButtons}
        <Button href={`history/${record.id}`} type="primary">
          History
        </Button>
      </>
    )}>
    <Stack gap={1}>
        <Typography variant="h6" fontWeight="bold">
            ID
        </Typography>
        <TextFieldOnlyView variant="body1" value={record?._id} />

        <Typography variant="h6" fontWeight="bold">
            Name
```

```
        </Typography>
        <TextFieldOnlyView variant="body1" value={record?.name} />

        <Typography variant="h6" fontWeight="bold">
            Active
        </Typography>
        <BooleanField value={record?.active} />

        <Typography variant="h6" fontWeight="bold">
            Plan
        </Typography>
        <TextFieldOnlyView variant="body1" value={record?.plan} />

        <Typography variant="h6" fontWeight="bold">
            Email
        </Typography>
        <EmailField variant="body1" value={record?.email} />

        <Typography variant="h6" fontWeight="bold">
            Credits Per Month
        </Typography>
        <NumberField variant="body1" value={record?.creditsPerMonth ?? ""} />

        <Typography variant="h6" fontWeight="bold">
            Created At
        </Typography>
        <DateField variant="body1" value={record?.createdAt} format="LLL" />

        <Typography variant="h6" fontWeight="bold">
            Updated At
        </Typography>
        <DateField variant="body1" value={record?.updatedAt} format="LLL" />
    </Stack>
</Show>
  );
};

export const getServerSideProps: GetServerSideProps<{}> = async (context) =>
    ↪ {
  const { authenticated, redirectTo } = await authProvider.check(context);

  if (!authenticated) {
    return {
      props: {},
```

```
      redirect: {
        destination: `${redirectTo}?to=${encodeURIComponent("/users")}`,
        permanent: false,
      },
    };
  }

  return {
    props: {},
  };
};

export default UsersById
```

Código 13: pages/users/show/history/[id].tsx

```
import { GetServerSideProps } from 'next';
import { authProvider } from '@providers/authProvider';
import { useShow, IResourceComponentsProps, useParsed } from '@refinedev/core
    ↪ ';
import { Show, TextFieldComponent as TextFieldOnlyView } from '@refinedev/mui
    ↪ ';
import {
  Typography,
  Stack,
  styled,
  Paper,
  Select,
  MenuItem,
  InputLabel,
  Button
} from '@mui/material';
import { CopyToClipboard } from '@components/CopyToClipboard';
import useVersionComparison from '@components/History';
import { formatDate } from '@utils/dateFormat';

const Item = styled(Paper)(({ theme }) => ({
  backgroundColor: theme.palette.mode === 'dark' ? '#1A2027' : '#fff',
  ...theme.typography.body2,
  padding: theme.spacing(1),
  textAlign: 'center',
  color: theme.palette.text.secondary
}));

```

```
export const Component: React.FC<IResourceComponentsProps> = () => {
  const { id } = useParsed();
  const { queryResult } = useShow({
    resource: 'users',
    id,
    dataProviderName: 'history'
  });

  const { data, isLoading } = queryResult;

  const record = data?.data ?? {};
  const { selectedUpdate1, selectedUpdate2, newVersion, oldVersion, showDiff,
   ↪  filteredOptions, handleSelectChange, DiffViewer } =
   ↪ useVersionComparison(record);
  return (
    <Show isLoading={isLoading} canDelete={false} goBack={false}
   ↪ headerButtons={({ defaultButtons }) => (
      <>
        {defaultButtons}
        <Button href={`/users/show/${id}`} type="primary">
          Go back to user data
        </Button>
      </>
    )}>
      <h1>History</h1>
      <Stack gap={1}>
        <Typography variant="h6" fontWeight="bold">
          User ID {id ? <CopyToClipboard value={id.toString()} /> : null}
        </Typography>
        <TextFieldOnlyView variant="body1" value={id} />

        <Typography variant="h6" fontWeight="bold">
          Name
        </Typography>
        <TextFieldOnlyView variant="body1" value={record[0]?.data.name} />

        <Typography variant="h6" fontWeight="bold">
          Select the entries you want to compare
        </Typography>
        <InputLabel id="select1-label">Newer Version</InputLabel>
        <Select
          labelId="select1-label"
          id="select1"
          value={selectedUpdate1 || ''}
```

```
68            onChange={(e) => handleSelectChange(e, 'select1')}
69          >
70            {Object.values(record).map((record: { data: { updatedAt: string }
    ↪ }, index) => (
71              <MenuItem key={record.data.updatedAt} value={record.data.
    ↪ updatedAt}>
72                {index === 0 ? `v${index} (latest): ` : `v${index}: `}
73                {formatDate(record.data.updatedAt)}
74              </MenuItem>
75            ))}
76          </Select>
77          <InputLabel id="select2-label">Older Version</InputLabel>
78          <Select
79            id="select2"
80            labelId="select2-label"
81            value={selectedUpdate2 || ''}
82            onChange={(e) => handleSelectChange(e, 'select2')}
83            disabled={!selectedUpdate1}
84          >
85            {filteredOptions && filteredOptions.length > 0 ? (
86              filteredOptions.map((record: { data: { updatedAt: string };
    ↪ originalIndex: number }, index) => (
87                <MenuItem value={record.data.updatedAt}>
88                  {`v${record.originalIndex}: `}{formatDate(record.data.
    ↪ updatedAt)}
89                </MenuItem>
90              ))
91            ) : (
92              <MenuItem disabled>No other versions to compare</MenuItem>
93            )}
94          </Select>
95        </Stack>
96        <DiffViewer showDiff={showDiff} oldVersion={oldVersion} newVersion={
    ↪ newVersion} />
97      </Show>
98    );
99  };
100
101 export const getServerSideProps: GetServerSideProps<{}> = async (context) =>
    ↪ {
102   const { authenticated, redirectTo } = await authProvider.check(context);
103
104   if (!authenticated) {
105     return {
```

```
      props: {},
      redirect: {
        destination: `${redirectTo}?to=${encodeURIComponent('/users')}`,
        permanent: false
      }
    };
  }

  return {
    props: {}
  };
};

export default Component;
```

Código 14: pages/users/index.tsx

```
import { useMemo } from 'react';
import { useDataGrid, ShowButton, List, DateField, NumberField, EditButton }
    ↪ from '@refinedev/mui';
import { IResourceComponentsProps, useModal, useTranslate, useUpdate } from '
    ↪ @refinedev/core';
import { DataGrid, GridColDef } from '@mui/x-data-grid';

import { authProvider } from '@providers';
import { CopyToClipboard } from '@components/CopyToClipboard';
import { GetServerSideProps } from '@interfaces/next';
import { Checkbox } from '@mui/material';
import Modal from '@components/Modal';

const getFields = (t: Function): GridColDef[] => [
  {
    field: 'id',
    flex: 0,
    width: 60,
    headerName: 'ID',
    headerAlign: 'center',
    renderCell: ({ row }) => <CopyToClipboard value={row.id} />
  },
  {
    field: 'name',
    flex: 1,
    headerName: 'Name'
  },
```

```
26  {
27    field: 'active',
28    headerName: 'Active',
29    align: 'center',
30    renderCell: ({ row }) => {
31      const { show: handleModalShow, close: handleModalClose, visible:
    ↪ modalVisible } = useModal();
32      const { mutate } = useUpdate();
33
34      const handleCheckboxChange = async () => {
35        try {
36          handleModalShow();
37        } catch (error) {
38          console.error('Error changing the value of the "Active" field: ',
    ↪ error);
39        }
40      };
41
42      const handleConfirmationModal = () => {
43        try {
44          mutate({
45            resource: 'users',
46            id: row.id,
47            values: {
48              name: row.name,
49              userId: row.userId,
50              plan: row.plan,
51              active: !row.active,
52              email: row.email,
53              creditsPerMonth: row.creditsPerMonth
54            }
55          });
56          handleModalClose();
57        } catch (error) {
58          console.error('Error changing the value of the "Active" field: ',
    ↪ error);
59        }
60      };
61
62      return (
63        <>
64          <Modal
65            isOpen={modalVisible}
66            onClose={handleModalClose}
```

```
            buttonHandler={handleConfirmationModal}
            buttonText="Confirm Change"
            title={`Change Confirmation for user ${row.name}:`}
          >
            Are you sure you want to change the "Active" field value to "{row
 ↪ .active ? 'inactive' : 'active'}"?
          </Modal>
          <Checkbox value={row.active} checked={row.active} onChange={
 ↪ handleCheckboxChange} />
        </>
      );
    }
  },
  {
    field: 'plan',
    flex: 1,
    headerName: 'Plan'
  },
  {
    field: 'creditsPerMonth',
    flex: 1,
    headerName: 'Credits',
    renderCell: ({ value }) => <NumberField value={value} />
  },
  {
    field: 'email',
    flex: 1,
    headerName: 'Email'
  },
  {
    field: 'createdAt',
    flex: 1,
    headerName: 'Created At',
    renderCell: ({ value }) => <DateField value={value} format="L" />
  },
  {
    field: 'updatedAt',
    flex: 1,
    headerName: 'Updated At',
    renderCell: ({ value }) => <DateField value={value} format="L" />
  },
  {
    field: 'actions',
    flex: 1,
```

```
    headerName: 'Actions',
    sortable: false,
    align: 'center',
    headerAlign: 'center',
    renderCell: ({ row }) => (
      <div>
        <EditButton hideText recordItemId={row.id} />
        <ShowButton hideText recordItemId={row.id} />
      </div>
    )
  }
];
export const Users: React.FC<IResourceComponentsProps> = () => {
  const t = useTranslate();
  const { dataGridProps } = useDataGrid({
    initialPageSize: 10
  });

  const columns = useMemo<GridColDef[]>(() => getFields(t), []);

  return (
    <List>
      <DataGrid {...dataGridProps} columns={columns} autoHeight />
    </List>
  );
};

export const getServerSideProps: GetServerSideProps<{}> = async (context) =>
  ↪ {
  const { authenticated, redirectTo } = await authProvider.check(context);

  if (!authenticated) {
    return {
      props: {},
      redirect: {
        destination: `${redirectTo}?to=${encodeURIComponent('/users')}`,
        permanent: false
      }
    };
  }

  return {
    props: {}
  };
```

```
};

export default Users;
```

Código 15: src/components/History/index.tsx

```
import { SelectChangeEvent } from '@mui/material';
import { BaseRecord } from '@refinedev/core';
import { useState, useEffect } from 'react';
import ReactDiffViewer, { DiffMethod } from 'react-diff-viewer';

interface DiffViewerProps {
    showDiff: boolean;
    oldVersion: string;
    newVersion: string;
  }

  const DiffViewer: React.FC<DiffViewerProps> = ({ showDiff, oldVersion,
   ↪ newVersion }) => (
    <div style={{ width: '100%', overflow: 'auto', padding: '8px' }}>
      {showDiff && (
        <ReactDiffViewer
          oldValue={oldVersion}
          newValue={newVersion}
          splitView={true}
          styles={{ diffContainer: { width: '100%', overflowX: 'auto' } }}
          leftTitle={'Older Version'}
          rightTitle={'Newer Version'}
          disableWordDiff
          compareMethod={DiffMethod.WORDS}
        />
      )}
    </div>
  );
const useVersionComparison = (record: BaseRecord) => {
    const [selectedUpdate1, setSelectedUpdate1Value] = useState('');
    const [selectedUpdate2, setSelectedUpdate2Value] = useState('');
    const [newVersion, setNewVersion] = useState('');
    const [oldVersion, setOldVersion] = useState('');
    const [showDiff, setShowDiff] = useState(false);
    const [filteredOptions, setFilteredOptions] = useState([]);

    useEffect(() => {
      if (newVersion && oldVersion) {
```

```
38          setShowDiff(true);
39        } else {
40          setShowDiff(false);
41        }
42      }, [newVersion, oldVersion]);
43
44      const handleSelectChange = (event: SelectChangeEvent<string>, select:
    ↪ string) => {
45        const selectedValue = event.target.value as string;
46        if (select == 'select1') {
47          setSelectedUpdate1Value(selectedValue);
48          const selectedRecord = record.find(
49            (record: { data: { updatedAt: string } }) => record.data.updatedAt
    ↪ === selectedValue).data;
50          setNewVersion(JSON.stringify(selectedRecord, null, 2));
51          const filterSelectData = record
52          .map((record: { data: { updatedAt: string } }, index: number) => ({
    ↪ data: record.data, originalIndex: index }))
53          .filter((record: { data: { updatedAt: string } }) => record.data.
    ↪ updatedAt < selectedValue);
54          setFilteredOptions(filterSelectData)
55        } else {
56          setSelectedUpdate2Value(selectedValue);
57          const selectedRecord = record.find(
58            (record: { data: { updatedAt: string } }) => record.data.updatedAt
    ↪ === selectedValue);
59          setOldVersion(JSON.stringify(selectedRecord.data, null, 2));
60        }
61      };
62
63    return {
64      selectedUpdate1,
65      selectedUpdate2,
66      newVersion,
67      oldVersion,
68      showDiff,
69      filteredOptions,
70      handleSelectChange,
71      DiffViewer
72    };
73  };
74
75  export default useVersionComparison;
```

Código 16: src/constants/index.ts

```
1
2  export const LOCAL_API_URL = process.env.NEXT_PUBLIC_LOCAL_API_URL;
3  export const ENTITIES = [
4    {
5      name: 'users',
6      list: '/users',
7      create: '/users/create',
8      edit: '/users/edit/:id',
9      show: '/users/show/:id',
10     history: '/api-keys/show/history/:id',
11     meta: {
12       canDelete: false
13     }
14   },
15   {
16     name: 'api-keys',
17     list: '/api-keys',
18     create: '/api-keys/create',
19     edit: '/api-keys/edit/:id',
20     show: '/api-keys/show/:id',
21     history: '/api-keys/show/history/:id',
22     meta: {
23       canDelete: false
24     }
25   },
26   {
27     name: 'services',
28     list: '/services',
29     // create: '/services/create',
30     // edit: '/services/edit/:id',
31     show: '/services/show/:id',
32     meta: {
33       canDelete: true
34     }
35   },
36   {
37     name: 'entities',
38     list: '/entities',
39     // create: '/entities/create',
40     // edit: '/entities/edit/:id',
41     show: '/entities/show/:id',
42     meta: {
```

```
      canDelete: true
    }
  }
  // {
  //   name: "categories",
  //   list: "/categories",
  //   create: "/categories/create",
  //   edit: "/categories/edit/:id",
  //   show: "/categories/show/:id",
  //   meta: {
  //     canDelete: true,
  //   },
  // },
];
```