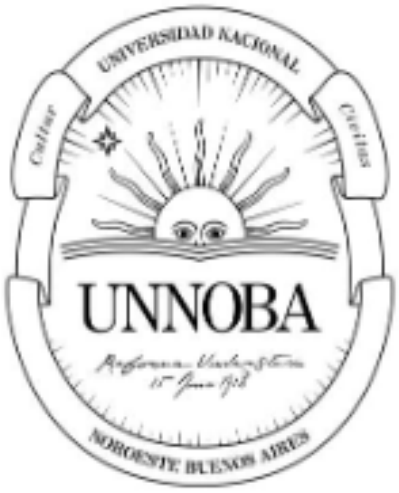

Universidad Nacional del Noroeste de la Provincia de Buenos Aires

Título: Onboarding Empresas ICBC

Carrera: Ingeniería en Informática

Práctica Profesional Supervisada

Alumno: Cellati Nicanor

Tutor Docente: Lic. De la Riva, Diego

Tutor de Empresa: Mg. Bruzzoni, Fernando

1. Introducción

La práctica profesional supervisada se llevó a cabo en el Industrial and Commercial Bank of China (ICBC) Argentina, ubicado en la Ciudad Autónoma de Buenos Aires. Esta experiencia fue supervisada por los Licenciados en Sistemas Diego de la Riva, en su rol de profesor supervisor, y Fernando Bruzzoni, como tutor designado por la empresa.

El presente informe detalla el proceso de aprendizaje y capacitación en el uso de herramientas y tecnologías implementadas en el ICBC. Durante esta práctica, se participó activamente en el desarrollo de un producto de software destinado a la incorporación de clientes empresariales.

El ICBC, reconocido como el banco más grande de China y el mayor banco del mundo por capitalización de mercado, se especializa en la prestación de servicios financieros. En este contexto, la creación de una solución digital para el proceso de onboarding empresarial tuvo un impacto significativo, optimizando los procesos internos del banco y mejorando notablemente la experiencia de sus clientes.

Además, esta práctica representó una valiosa oportunidad para aprender de profesionales altamente experimentados, colaborar en equipos multidisciplinarios y profundizar en la aplicación de los principios de la ingeniería de software en una institución financiera líder a nivel global.

2. Objetivos:

2.1 Objetivos generales:

El objetivo principal de la presente práctica profesional supervisada (PPS) fue facilitar y optimizar la

incorporación de nuevos clientes empresariales a los servicios financieros del Industrial and Commercial Bank of China (ICBC) mediante el desarrollo de un sistema de onboarding para empresas. Este sistema busca simplificar el proceso de apertura de cuentas y la contratación de servicios financieros, reduciendo significativamente el tiempo y los recursos necesarios para que las empresas se conviertan en clientes activos. De esta manera, se pretende mejorar la experiencia del usuario desde el inicio del proceso, permitiendo a las empresas comenzar a operar de manera más eficiente y ágil.

2.2 Objetivos específicos:

1. Adquirir conocimientos sobre las herramientas y metodologías utilizadas por el ICBC en el desarrollo de sus sistemas y procesos.
2. Dominar el framework Angular, con el propósito de aplicar patrones de diseño en la construcción de módulos, componentes y servicios dentro del onboarding para empresas.
3. Desarrollar incrementos en el sistema de onboarding para empresas conforme a los requerimientos planteados por las partes interesadas.
4. Realizar el lanzamiento incremental del sistema a producción y observar su comportamiento, con el objetivo de identificar áreas de mejora continua.

3. Plan de Trabajo y Carga Horaria

En la primera etapa de la práctica, se llevaron a cabo diversas capacitaciones en herramientas tecnológicas y en la metodología empleada por el ICBC para el desarrollo de productos de software. Estos conocimientos fueron fundamentales para poder aplicar las competencias adquiridas en el desarrollo de nuevas funcionalidades. De esta manera, se contribuyó al desarrollo de incrementos en el producto, alineándose con los objetivos estratégicos del banco.

En la tabla 1 se puede observar el diagrama de Gantt con las tareas realizadas cada semana y la duración de las mismas.

N°	***ACTIVIDADES***	***TIEMPO DE DURACIÓN***									
		SEMANAS									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	Capacitación sobre uso de Miro	*									
2	Capacitación en metodología Agile Scrum en ICBC	*									
3	Capacitación en Jira	*									
4	Capacitación sobre herramientas: Lucy, AppFlow, Nexus y la pila ELK		*								
5	Investigación sobre CI/CD en GitLab		*								
6	Investigación sobre la librería RxJS de			*							

	Angular										
7	Investigación sobre la librería NgRx de Angular				*						
8	Desarrollo de tareas utilizando RxJS, Lucy y CI/CD					*	*	*			
9	Desarrollo de tareas utilizando NgRx							*	*	*	
10	Despliegue de nuevas funcionalidades en producción										*
11	Monitoreo y observabilidad de infraestructura usando Dynatrace										*

Tabla 1. Cronograma de tareas

4. Descripción de la práctica profesional:

4.1 Capacitación sobre uso de Miro

Miro es una plataforma de colaboración digital diseñada para facilitar la gestión de proyectos y la comunicación de equipos remotos y distribuidos [3]. En el contexto de la práctica profesional, se utilizó Miro para realizar las siguientes actividades:

- Brainstorming Creativo: Cuando el equipo debía decidir la mejor forma de abordar la resolución técnica de un requerimiento, se presentaban distintos diseños en Miro para consensuar cuál de ellos se llevaría a cabo. Este proceso de lluvia de ideas permitía que el equipo visualizara y comprendiera de manera detallada soluciones técnicas complejas, favoreciendo la toma de decisiones colaborativas.

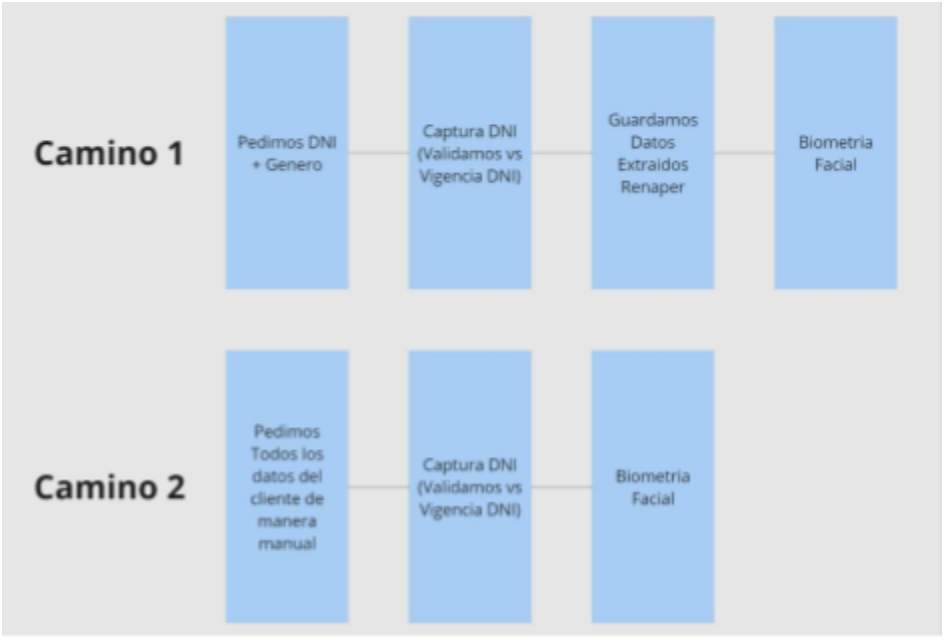

Figura 1: Diseño de solución para escaneo de frente y dorso del DNI (Propia)

- Diseño de Prototipos y conexiones al Backend: El equipo de UX/UI utiliza Miro para presentar al equipo de Frontend los prototipos interactivos que han diseñado. Esto agiliza el proceso de desarrollo, ya que permite anticipar el esfuerzo necesario y analizar la construcción de la solución. Por otro lado, el equipo de Backend documenta la estructura de los servicios asociados a los distintos eventos que pueden generarse mediante la interacción del usuario con los componentes de la aplicación.

Figura 2: Diseño de prototipos y conexiones al backend (Propia)

- Gestión de proyectos intuitiva: Miro facilita la organización de tareas y proyectos de manera eficiente. El Scrum Master utiliza el tablero para gestionar el proyecto, asignar tareas y realizar un seguimiento del progreso en un entorno visualmente atractivo durante cada sprint.

Figura 3: Seguimiento del progreso en el sprint (Propia)

- Aprendizaje colaborativo: La plataforma permite realizar encuentros de aprendizaje colaborativo entre diferentes equipos para explicar de forma gráfica diversas técnicas y

conceptos, promoviendo un entendimiento compartido de los procesos.

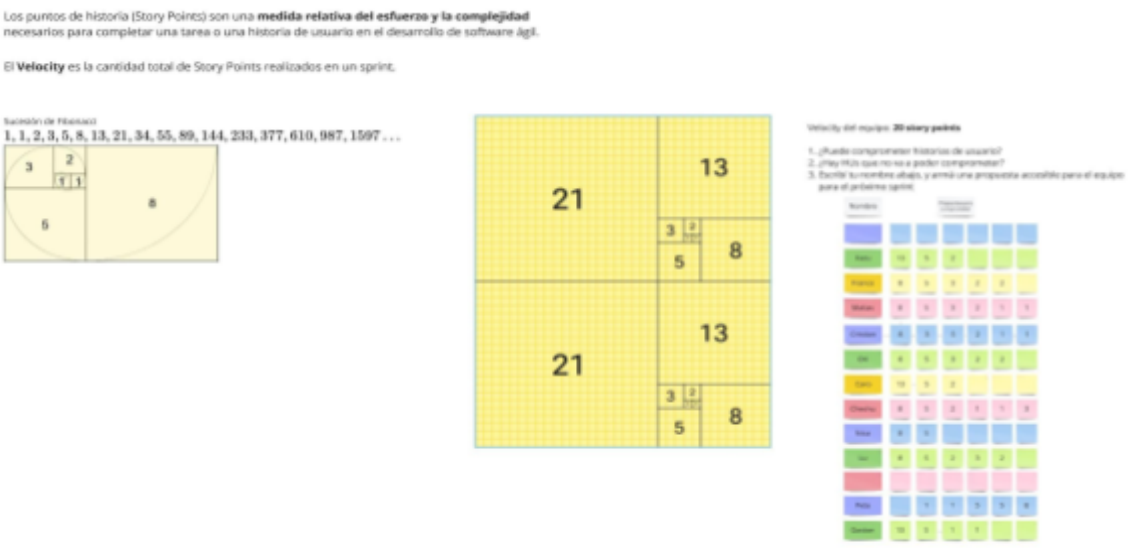

Figura 4: Ejercicio práctico de estimación de puntos de historia (Propia)

4.2 Capacitación en metodología Agile Scrum en ICBC

La metodología Agile Scrum es un enfoque de gestión de proyectos basado en el desarrollo incremental y la entrega continua de valor. Este marco de trabajo permite a los equipos gestionar proyectos complejos a través de ciclos iterativos denominados sprints, en los cuales se desarrolla un conjunto de funcionalidades o tareas específicas [1]. Al finalizar cada sprint, se obtiene un incremento del producto, que se presenta como una versión potencialmente entregable, lista para ser puesta en producción si cumple con los criterios de aceptación previamente definidos por el dueño del producto.

El trabajo dentro de Scrum se organiza a través del Product Backlog, una lista priorizada por las partes interesadas en el producto, el mismo contiene todos los requisitos, historias de usuario y funcionalidades a desarrollar. A partir de esta lista, el equipo selecciona los ítems a abordar durante el sprint, organizándolos en el Sprint Backlog. De esta manera, cada sprint tiene un objetivo claro y medible, permitiendo una colaboración constante entre los miembros del equipo para lograr la entrega continua de valor. [4]

Principales componentes de Agile Scrum

Roles:

- Scrum Master: Es el facilitador del equipo, responsable de eliminar obstáculos y asegurar que el equipo siga las prácticas de Scrum.
- Product Owner: Representa las necesidades del cliente y define las funcionalidades del producto. Se encarga de crear las historias de usuario, gestionar el backlog del producto y supervisar el backlog del sprint.
- Equipo de Desarrollo: Son los profesionales encargados de crear el producto o entregar el proyecto, incluyendo desarrolladores Backend y Frontend.
- Equipo de Testing: Se encarga de probar las funcionalidades entregadas por los desarrolladores para garantizar la calidad e integridad del producto.
- Equipo de UX/UI: Profesionales encargados del diseño de prototipos y la experiencia de usuario, asegurando que los requisitos del producto se reflejen correctamente en el diseño.

Eventos:

- Sprint: Es el ciclo continuo de trabajo en Scrum. Su duración se mantiene constante durante el desarrollo del producto, proporcionando un ritmo constante que permite comparar los resultados a lo largo de los sprints. Cada sprint se planifica para generar un incremento específico del producto. [10]
- Dailys: Breves reuniones diarias en las que, bajo la supervisión del Scrum Master, el equipo informa sobre el progreso de las tareas asignadas y coordina las actividades del día.
- Sprint Refinement: Supervisada por el Product Owner y facilitada por el Scrum Master, esta reunión involucra al equipo de desarrollo, testing y UX/UI. Su objetivo es analizar los elementos del backlog y entender los criterios de aceptación, así como los prototipos asociados. También se realiza un análisis técnico de la viabilidad de las historias de usuario.
- Sprint Planning: Reunión realizada al inicio de cada sprint, donde el equipo Scrum completo revisa el Product Backlog, priorizado por el Product Owner, y selecciona los elementos que trabajarán en el siguiente sprint. El equipo realiza estimaciones del esfuerzo necesario para cumplir con la Definición de Hecho (DoD) de cada Historia de Usuario (HU). [19]
- Retrospectiva del Sprint: Reunión al final del sprint donde se analiza el desempeño del equipo, se revisa si se alcanzaron los objetivos del sprint y se identifican los obstáculos enfrentados. Se discuten los cambios que deben realizarse en el próximo sprint y se evalúa qué procesos deben mantenerse o mejorarse.
- Sprint Review: Reunión final del sprint donde el Product Owner, junto con un miembro del equipo, presenta el incremento final a los stakeholders. Después de la presentación, los stakeholders tienen la oportunidad de hacer preguntas y ofrecer comentarios sobre el producto entregado.

Artefactos:

- Product Backlog: Es una lista priorizada de funcionalidades y requisitos del producto, gestionada por el Product Owner.
- Sprint Backlog: Es una lista de tareas seleccionadas para el sprint, creada por el equipo de desarrollo.
- Incremento: Es la versión del producto que se mejora con cada sprint, representando el avance logrado durante el ciclo. [4]

4.3 Capacitación en Jira

Durante la práctica profesional, se utilizó Jira Software para realizar el seguimiento de las tareas dentro del proyecto, aplicando la metodología ágil de desarrollo. El equipo aprovechó diversas funcionalidades de Jira para gestionar el flujo de trabajo y optimizar la organización de los recursos. El tablero de Jira se utilizó para estructurar el trabajo con las siguientes herramientas:

- **Task**: Las tareas en Jira se utilizan para planificar actividades específicas. Cada tarea puede dividirse en subtareas y puede ser asignada a un miembro del equipo responsable. Las tareas forman parte de un sprint, es decir, están en proceso activo de desarrollo, y pueden ser reenviadas a otros miembros para recibir comentarios. Una vez completadas, se marcan como terminadas.
- **History User (Historia de Usuario)**: Las Historias de Usuario son descripciones breves de los requisitos o solicitudes del proyecto desde la perspectiva del usuario final. Una Historia de Usuario consta de:
    - Un responsable asignado, generalmente un miembro del equipo de desarrollo.
    - Un informador, que suele ser el Product Owner (PO).
    - El equipo o célula responsable del desarrollo.
    - El **Business Value** (valor de negocio) asociado a la Historia de Usuario, el cual representa el beneficio que aporta al producto.
    - Un valor de **Story Points**, que indica el esfuerzo estimado necesario para completar el desarrollo y las pruebas de la historia.
    - La **prioridad** de la Historia de Usuario, clasificada como Highest, Medium, Low o Lowest, lo cual determina el orden en que se trabajará en ella.
- **Epic**: Las épicas son grandes temas o objetivos estratégicos que se dividen en Historias de Usuario y Tareas más pequeñas. Las épicas permiten mantener una visión global del proyecto mientras se gestionan los detalles a nivel de historia o tarea. [9]

Una vez iniciado el sprint, se realiza un encuentro de organización técnica entre los miembros responsables (backend, frontend y testing) de la construcción de una historia o tarea. Este proceso tiene como objetivo dividir el trabajo en subtareas significativas para facilitar el seguimiento del

progreso del equipo a lo largo del sprint.

Si en la planificación del sprint se contempló que algún miembro del equipo debía participar en el desarrollo de más de una Historia de Usuario, la selección de por cuál comenzar se basa en la prioridad asignada a cada historia o tarea. El equipo de trabajo comienza a recoger las subtareas desde la columna de tareas pendientes, moviéndolas a la etapa de "En Progreso" para llevar a cabo la actividad correspondiente.

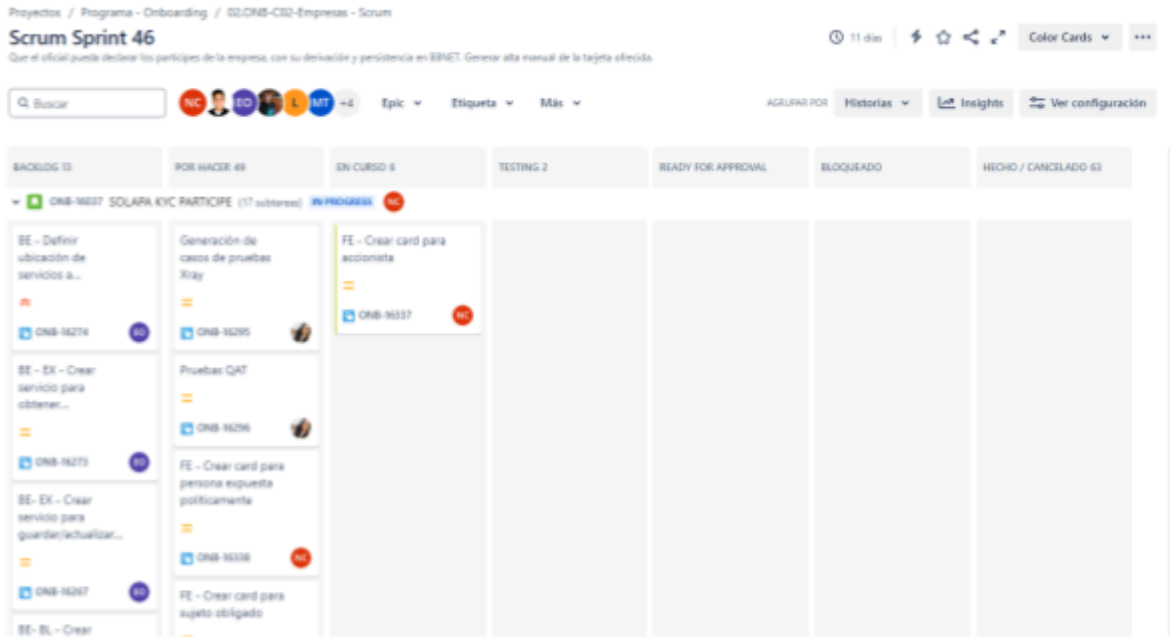

Figura 5: Visualización del estado de las subtareas de una HU en Jira (Propia)

Al finalizar cada subtarea, se actualiza el estado a "Finalizado". Una vez que todas las subtareas asociadas al desarrollo han sido finalizadas, en este punto, se notifica al equipo de QA a través de los comentarios de la tarjeta, informando que el desarrollo ha sido publicado para su respectiva prueba. A continuación, el estado de la Historia de Usuario se actualiza a "Testing". Luego, el responsable asignado quedará pendiente del feedback publicado una vez finalizadas las pruebas. Si todos los casos de prueba son completados exitosamente, el equipo de QA se encarga de cerrar la Historia de Usuario, cambiando su estado a "Finalizado" y adjuntando evidencia que valide que todos los scripts de pruebas se han completado correctamente.

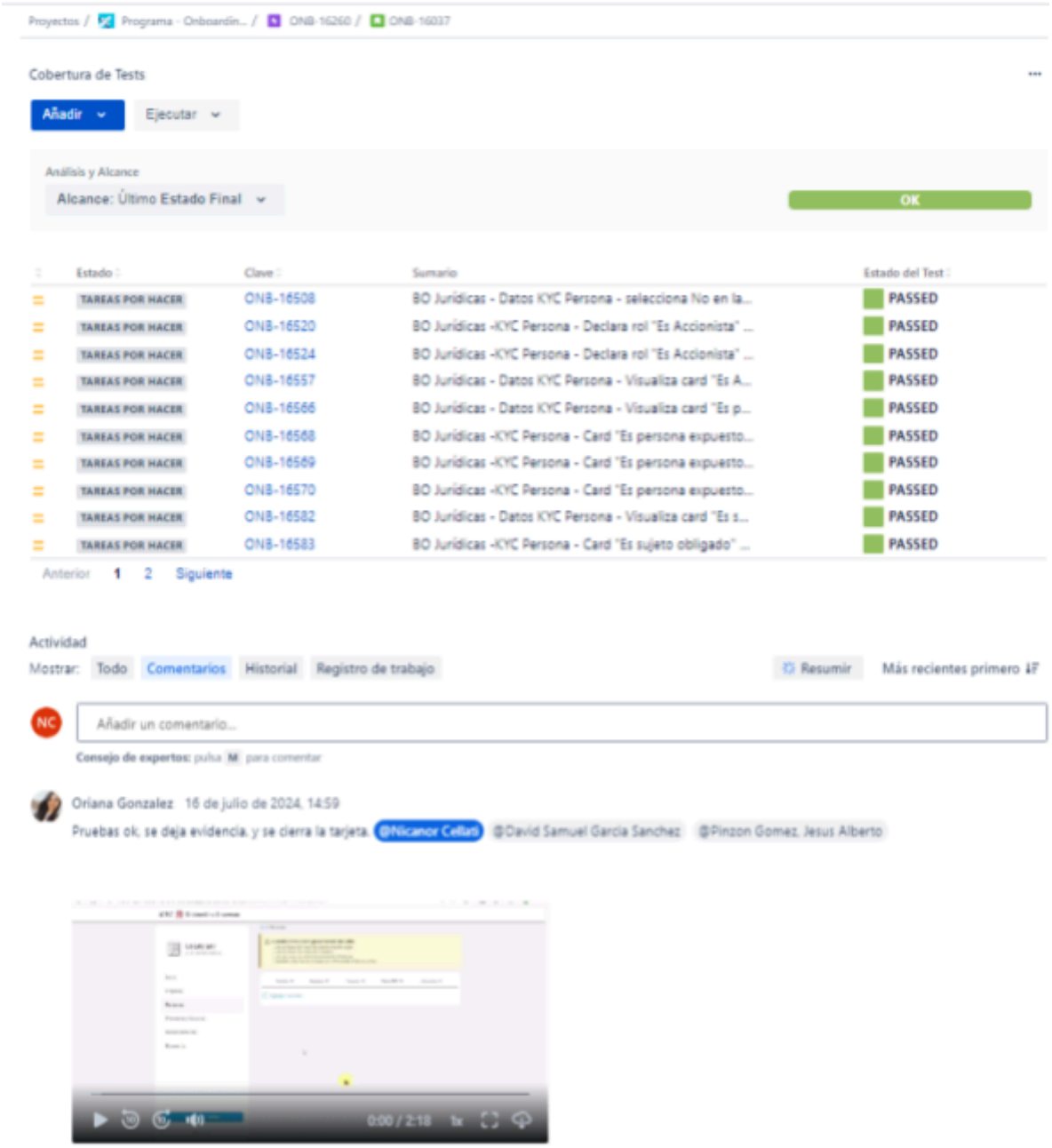

Figura 6: Cobertura de test y evidencia de pruebas (Propia)

A medida que transcurre el sprint, el Scrum Master motiva al equipo a utilizar los diagramas de evolución proporcionados por Jira. Consultar el gráfico de evolución resulta fundamental ya que ofrece una representación visual del trabajo estimado frente al trabajo real completado. Este gráfico se actualiza automáticamente a medida que las historias y tareas se van finalizando, proporcionando una visión clara del progreso general del proyecto.

Figura 7: Reporte de sprint (Propia)

4.4 Capacitación sobre herramientas: Lucy, AppFlow, Nexus y la pila ELK

Lucy

Lucy es un sistema de diseño (design system) que tiene como objetivo generar consistencia en la experiencia de los clientes a través de los distintos canales de atención, además de mejorar la velocidad de desarrollo mediante la estandarización de componentes e interacciones. En ICBC, existen requisitos específicos en términos de identidad visual y funcional que deben ser replicados en cada uno de los puntos de contacto con el usuario. Intentar llevar a cabo este proceso utilizando bibliotecas de componentes visuales genéricas resultaría inadecuado.

Desarrollar bibliotecas propias de componentes permite a la organización adaptar los elementos visuales a sus necesidades particulares, evitando la dependencia de avances propuestos por proveedores externos de sistemas de diseño de acceso abierto, como Material Design o Atomic Design, entre otros. De este modo, ICBC puede gestionar de manera más eficiente los costos y el ciclo de vida de los componentes, de acuerdo con las necesidades cambiantes de la organización.

El proceso de trabajo con Lucy es similar al de otros sistemas de diseño de componentes visuales. En primer lugar, los desarrolladores frontend analizamos los prototipos compartidos en las Historias de Usuario, a través de un enlace proporcionado por el equipo de UX en Figma. Este equipo se encarga de validar previamente la "deseabilidad" de los prototipos por parte de los usuarios, así como de definir los objetivos y los resultados esperados por el área de negocio. Posteriormente, se identifican los componentes de Lucy que serán utilizados en la construcción del diseño. Una vez identificados, se valida la documentación disponible, que proporciona información detallada sobre los componentes, su implementación y ejemplos de aplicación.

AppFlow

AppFlow es una herramienta de gestión de dependencias que se utiliza para la creación de aplicaciones, microservicios, endpoints y entidades, que posteriormente se emplearán en las solicitudes y respuestas de llamadas a APIs REST. Durante el proceso diario de desarrollo, hacemos uso de esta herramienta para verificar la disponibilidad y existencia de las dependencias que necesitamos integrar en el Frontend al trabajar con microservicios y utilizar sus respectivos endpoints.

Además, AppFlow permite inspeccionar los atributos de las entidades utilizadas en las solicitudes (Request) y respuestas (Response) de cualquier endpoint. Esta funcionalidad facilita la validación de las dependencias necesarias, asegurando su correcta integración en el proceso de desarrollo.

Nexus

Nexus de Sonatype es un administrador de repositorios que organiza, almacena y distribuye los artefactos necesarios para el desarrollo. Esta herramienta permite alojar repositorios privados y también funciona como proxy para repositorios públicos. El uso de un proxy reduce el tiempo de acceso a los artefactos y optimiza el ancho de banda.

En nuestra práctica diaria, Nexus facilita la búsqueda y gestión de los artefactos que necesitamos instalar. Las dependencias generadas en AppFlow se almacenan en Nexus, lo que permite verificar que la última versión generada esté correctamente alojada. Al acceder a cada artefacto, podemos consultar la información necesaria para su uso, así como los comandos requeridos para instalarlo mediante el gestor de paquetes que utilicemos, como npm o mvn, entre otros. [13][17]

La pila ELK

La pila ELK es un conjunto de herramientas compuesto por tres proyectos clave: **Elasticsearch**, **Logstash** y **Kibana**. Aunque generalmente se conoce como Elasticsearch, la pila ELK ofrece un conjunto completo para la gestión de registros (logs), su análisis y la visualización de los resultados obtenidos.

El funcionamiento de la pila ELK se puede desglosar de la siguiente manera:

- **Logstash**: Se encarga de ingerir, transformar y enviar los datos al destino correspondiente.
- **Elasticsearch**: Indexa, analiza y permite realizar búsquedas sobre los datos ingeridos.
- **Kibana**: Permite visualizar los resultados del análisis, ofreciendo gráficos y reportes detallados sobre los registros. [2]

En colaboración con el equipo de desarrollo, utilizamos principalmente Kibana para el diagnóstico de errores. A través de la visualización de los registros generados en el servidor, podemos realizar

un seguimiento de las solicitudes a los microservicios y obtener trazabilidad desde el envío de la solicitud hasta la obtención de la respuesta.

En caso de que la respuesta no sea exitosa, Kibana facilita la identificación de errores, permitiendo detectar el punto exacto en el que ocurrió el fallo. Dado que nuestros servicios pueden depender de otros servicios creados y mantenidos por equipos ajenos, esta herramienta es crucial para identificar de manera eficiente si el error proviene de algún servicio externo. Al contar con Kibana, podemos rápidamente identificar el error y notificar al equipo correspondiente para su corrección, agilizando la resolución de problemas.

4.5 Investigación sobre CI/CD en GitLab

- Introducción a CI/CD:

La metodología CI/CD se enmarca dentro de las prácticas de DevOps, que buscan la integración de los equipos de desarrollo y operaciones. CI/CD combina las prácticas de Integración Continua (CI) y Entrega Continua (CD), automatizando gran parte, si no toda, de la intervención humana previamente necesaria para llevar el código de desarrollo a producción. Este proceso abarca fases clave como la construcción, las pruebas (que incluyen pruebas de integración, pruebas unitarias y pruebas de regresión), la implementación y la infraestructura.

Mediante la implementación de un canal de CI/CD, los equipos de desarrollo pueden realizar cambios en el código, los cuales son probados y enviados automáticamente para su entrega e implementación. Esta automatización optimiza el flujo de trabajo, reduciendo la intervención manual y aumentando la eficiencia en el proceso de desarrollo.

- Importancia de CI/CD:

El uso de CI/CD es crucial para la optimización del trabajo conjunto entre los equipos de desarrollo, seguridad y operaciones. Gracias a esta práctica, los equipos de DevOps obtienen retroalimentación más rápida y pueden integrar cambios más pequeños con mayor frecuencia, lo que contribuye a reducir los riesgos de introducir cambios disruptivos. Además, facilita la identificación temprana de problemas, permitiendo su pronta resolución y minimizando su impacto en el ciclo de desarrollo.

- Integración continua (CI):

La Integración Continua (CI) es una práctica que consiste en integrar los cambios de código en la rama principal de un repositorio compartido de manera frecuente y temprana. Esta práctica asegura que cada cambio se pruebe automáticamente al ser confirmado o fusionado, y se ejecute una compilación automática. La integración continua permite identificar y corregir errores y

problemas de seguridad mucho antes en el proceso de desarrollo.

Al fusionar los cambios con regularidad y activar procesos automáticos de prueba y validación, se minimiza la posibilidad de conflictos de código, incluso cuando varios desarrolladores trabajan en la misma parte del proyecto. Tras pasar las pruebas estáticas, las rutinas de CI automatizadas empaquetan y compilan el código, asegurando la calidad del software desde las etapas iniciales del desarrollo. [20]

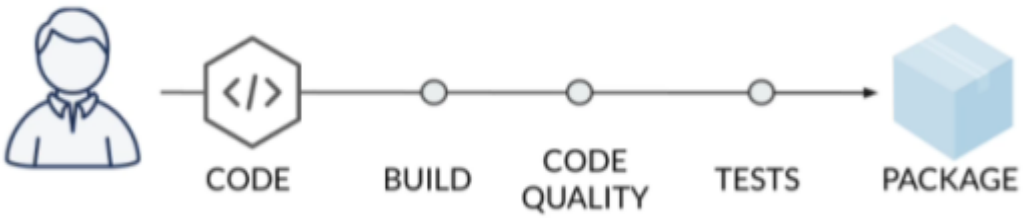

Figura 8: Proceso de integración continua (Udemy: Despa, V. (2025))

En la Figura 8 se ilustra el proceso mediante el cual, a partir del código fuente, se genera un paquete listo para ser desplegado. Este proceso se compone de las siguientes etapas:

1) Build: En esta fase, el código fuente de la aplicación se transforma en un artefacto ejecutable. En el contexto de CI/CD, este proceso se automatiza y se integra en una pipeline, una secuencia de pasos que se ejecutan cada vez que se produce un cambio en el código.
2) Code Quality: Esta etapa se encarga de analizar el código en busca de posibles problemas, tales como errores, vulnerabilidades, código duplicado y malas prácticas. Automatizar este análisis permite detectar y corregir problemas de manera temprana, acelerando el desarrollo y reduciendo costos asociados con la corrección de errores.
3) Tests: En esta fase se ejecutan pruebas automatizadas sobre el código para verificar su funcionamiento, asegurándose de que no se introduzcan nuevos errores y que se cumplan los estándares de calidad establecidos. La automatización de esta etapa es esencial para garantizar la fiabilidad del software. [6]

Entrega continua (CD):

La Entrega Continua (CD) es una práctica complementaria a la Integración Continua, que automatiza el proceso de aprovisionamiento de infraestructura y la puesta en marcha de aplicaciones. Una vez que el código ha sido probado y generado durante el proceso de CI, CD se encarga de las etapas finales, asegurando que el código esté completamente empaquetado y listo para su implementación en cualquier entorno. [20]

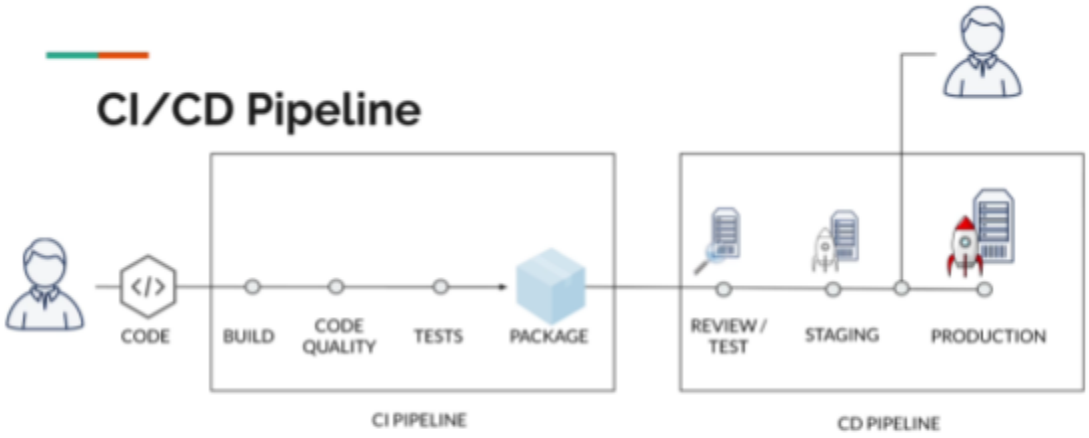

Figura 9: Procesos de integración continua y entrega continua (Udemy: Despa, V. (2025))

El proceso de CD se divide en las siguientes etapas:

- Review/Test: En esta etapa, se revisa la calidad del código y se asegura que los cambios no introduzcan nuevos errores o problemas. Es una revisión final antes de que el código sea desplegado en producción.
- Staging (o QAT en el ICBC): Este entorno de pruebas, que se encuentra entre desarrollo y producción, permite realizar una evaluación más detallada y cercana a las condiciones reales de producción. El equipo de QA ejecuta pruebas manuales adicionales para detectar errores que no hayan sido identificados en las pruebas automatizadas, simulando escenarios de uso real.
- Production: Finalmente, tras pasar las etapas anteriores, el código se despliega en el entorno de producción. En el ICBC, este proceso puede requerir intervención manual por razones de seguridad. Sin embargo, GitLab permite automatizar la mayoría de los despliegues en producción, convirtiendo esta etapa en el objetivo final del proceso de CD.[6]

Beneficios de implementar CI/CD:

- El uso de CI/CD reduce significativamente la cantidad de errores que llegan a producción, mejorando la experiencia de los usuarios y clientes.
- Aceleración del tiempo de obtención de valor: Al permitir despliegues en cualquier momento, CI/CD facilita que los productos y nuevas funcionalidades lleguen al mercado de manera más rápida.
- Mayor confiabilidad en el cumplimiento de fechas: La automatización de los despliegues elimina cuellos de botella y hace que el proceso de implementación sea más predecible, aumentando la confiabilidad en los plazos establecidos.

- Resolución de problemas más eficiente: CI/CD facilita la resolución de problemas y la recuperación de incidentes, reduciendo el tiempo medio de resolución. Los desarrolladores pueden corregir errores rápidamente o revertir cambios para minimizar el impacto sobre los clientes. [20]

4.6 Investigación sobre la librería RxJS de Angular

La librería de programación reactiva RxJS se ha consolidado como una de las herramientas más utilizadas en el desarrollo web moderno. Esta librería introduce conceptos clave y un cambio de paradigma al pasar de un enfoque imperativo a uno declarativo. Las aplicaciones actuales generan una gran cantidad de eventos en tiempo real, lo que enriquece la experiencia interactiva del usuario. Para gestionar estos eventos de manera eficiente, surge la necesidad de herramientas especializadas, siendo la programación reactiva una de las soluciones más adecuadas [7]. En este contexto, me enfoqué en investigar la aplicación de RxJS dentro del marco de trabajo de Angular.

La programación reactiva es un paradigma declarativo que se centra en:

- El manejo de flujos de datos.
- La propagación de cambios a través de los componentes del sistema.

En este enfoque, todos los componentes se modelan como streams o flujos de datos. Un flujo de datos, en términos informáticos, es una secuencia de datos que se emiten a lo largo del tiempo. Este modelo puede conceptualizarse como los elementos de una banda transportadora, procesados uno por uno, en lugar de en bloques grandes. Gracias a esta representación, tanto los datos como los eventos generados durante el ciclo de vida de una aplicación pueden verse como una secuencia temporal. Los sistemas reactivos, por lo tanto, responden a los cambios en los datos, permitiendo su gestión dinámica.

La propagación de cambios se refiere a la manera en que se notifica a una parte del sistema que una de sus dependencias ha cambiado, lo que implica que debe reaccionar adecuadamente a dicho cambio. [5]

En los sistemas reactivos, la relación entre el emisor de los cambios y los receptores de estos cambios se define de manera distinta a la de los sistemas tradicionales. A continuación, se incluye una representación gráfica que ilustra este concepto.

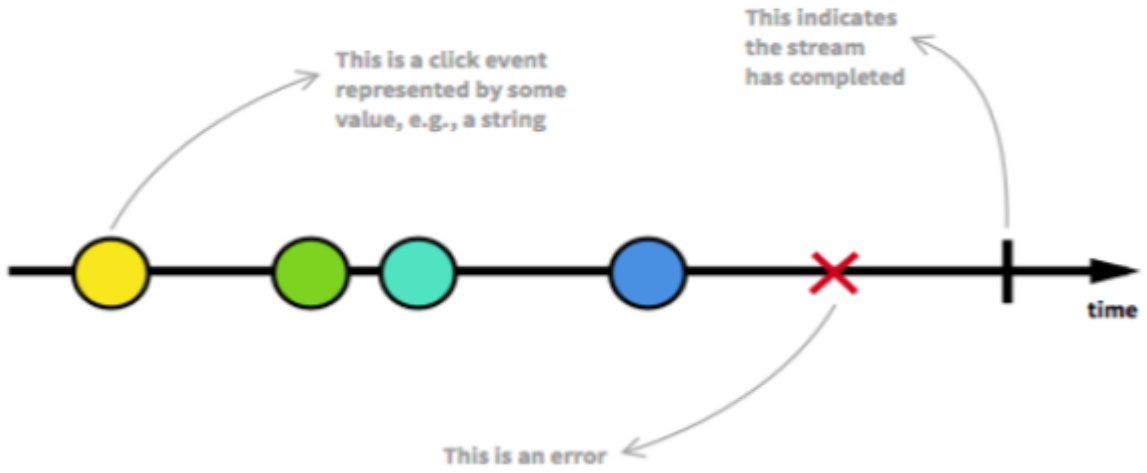

Figura 10: Secuencia de un stream (GitHub: Staltz, A. (2018))

Los flujos de datos, denominados *observables*, pueden representar diversas fuentes, tales como eventos de usuario, solicitudes HTTP o cambios en los datos. Estos observables pueden emitir tres tipos de eventos: un valor (de cualquier tipo), un error o una señal de "completado". Por ejemplo, el evento "completado" ocurre cuando se cierra una ventana o vista que contiene un componente interactivo, como un botón.

Los eventos emitidos por un observable se capturan de forma asincrónica, mediante la definición de funciones que se ejecutan cuando se emite un valor, cuando ocurre un error o cuando se emite el evento "completado". Este proceso de "escuchar" los eventos generados se conoce como *suscripción*, y las funciones que reaccionan a estos eventos son denominadas *observadores*. El sujeto que emite los eventos es el observable, y este patrón sigue el patrón de diseño *Observador*.[18]

El patrón de diseño *Observador* es uno de los patrones de comportamiento, los cuales se centran en la comunicación entre objetos. Su objetivo es definir una relación uno a muchos, de modo que cuando un objeto cambia de estado, todos los objetos dependientes sean notificados y actualizados automáticamente. En otras palabras, permite que los observadores sean informados de los eventos ocurridos en el sistema. [14]

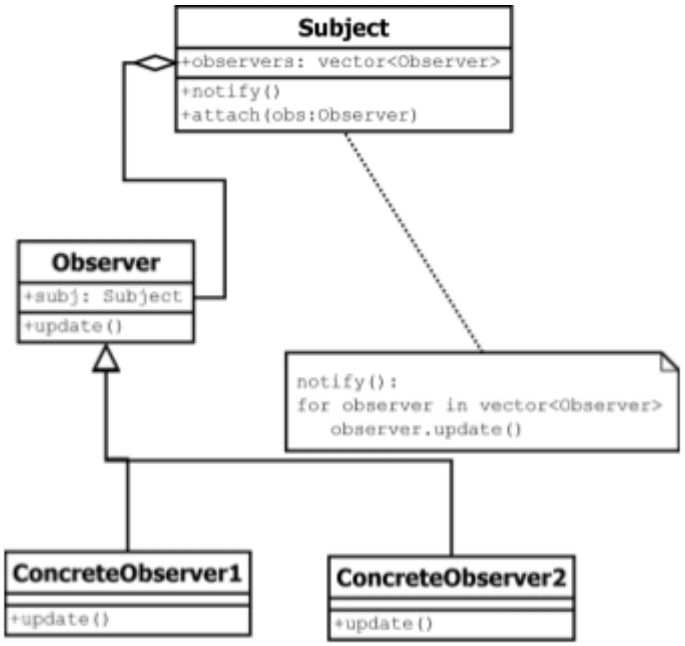

Figura 11: Diagrama de patrón observable (Medium: Poyias, A. (2019))

Los *ConcreteObservers* son clases que contienen información específica de la instancia actual. Estas clases implementan una función de actualización que es invocada por el sujeto a través de la operación *notify()*. Los observadores actualizan su estado de manera independiente, según las condiciones de su instancia.

El observador es la clase principal dentro de los observadores concretos. Esta clase mantiene una referencia al sujeto y, cuando se inicializa, se registra o se adjunta a dicho sujeto. Por su parte, el sujeto (también conocido como *subject*) mantiene una colección de observadores a los que notifica cuando se produce un cambio. Cuando un evento se activa, el sujeto recorre la lista de observadores y llama a su función de actualización.

Cuando nos suscribimos a un observable, continuamos recibiendo los valores emitidos por él hasta que ocurra alguna de las siguientes situaciones: el productor declara que ya no enviará más valores, momento en el cual se emite el evento "completado" (mediante la función *complete()*), o bien, nosotros como consumidores decidimos dejar de recibir valores y nos desuscribimos, lo que proporciona una ventaja importante al evitar posibles pérdidas de memoria.

Transformación de Observables

Los observables pueden ser manipulados utilizando operadores, que son funciones provistas por RxJS. Estos operadores son herramientas clave para transformar y manejar flujos de datos de

manera efectiva. A través de ellos, es posible componer código asincrónico complejo de forma declarativa.

Existen dos tipos principales de operadores:

- **Operadores pipeables**: Estos operadores se aplican a un observable mediante la sintaxis *observableInstance.pipe(operator)* o, más comúnmente, *observableInstance.pipe(operatorFactory())*. Al invocar un operador pipeable, este no modifica el observable original, sino que devuelve un nuevo observable que mantiene la lógica de suscripción del observable inicial. Así, al suscribirse al observable de salida, también se suscribe al observable de entrada.

- **Operadores de creación**: Son funciones utilizadas para crear un observable con un comportamiento predefinido o que se combinan con otros observables para generar nuevos flujos de datos. [12][16]

4.7 Investigación sobre la librería NgRx de Angular

NgRx es una librería de gestión de estado que proporciona una implementación de Redux dentro de la arquitectura Flux. Esta librería está diseñada para ayudar a gestionar el estado de aplicaciones de Angular de manera predecible y eficiente, a través de un enfoque basado en RxJS.

Al desarrollar aplicaciones web para resolver problemas del mundo real, nos encontramos con múltiples componentes organizados en varios grupos. Cada uno de estos grupos posee un componente principal encargado de gestionar el estado correspondiente. Sin embargo, cuando se realizan cambios en uno de los componentes, puede ser necesario volver a renderizar otros componentes. Si estos componentes no comparten un componente principal común, la transmisión de información entre ellos se vuelve difícil de gestionar.

Una solución a este problema fue el desarrollo de la arquitectura Flux, originada a partir del modelo MVC. Flux establece un flujo de datos unidireccional, donde existe un almacén central para toda la aplicación, denominado "store". Este almacén actúa como la fuente única de verdad para el estado de la aplicación, y todos los cambios se gestionan a través de un mecanismo de acciones y despachadores (dispatcher), que serializan las solicitudes de cambio. Además, las vistas pueden suscribirse al almacén y, cada vez que el estado cambia, se actualizan automáticamente.

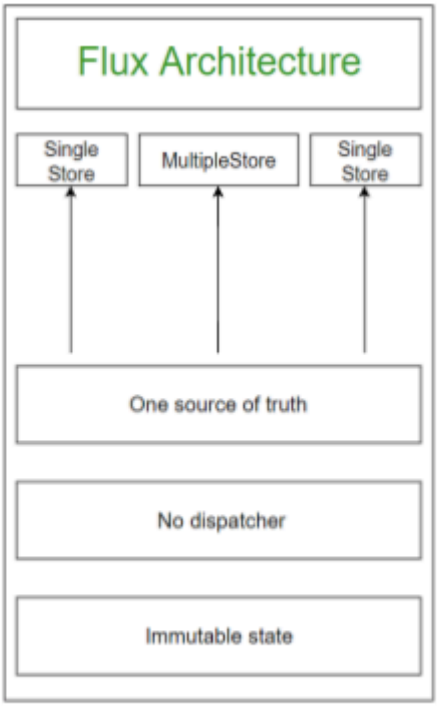

Figura 12: Diagrama de arquitectura Flux (Geeksforgeeks: (2020))

Arquitectura Redux

Redux es un contenedor de estado predecible para aplicaciones JavaScript, y se deriva de la arquitectura Flux. A diferencia de Flux, que maneja múltiples almacenes, Redux utiliza un único almacén centralizado para controlar el estado global de la aplicación.

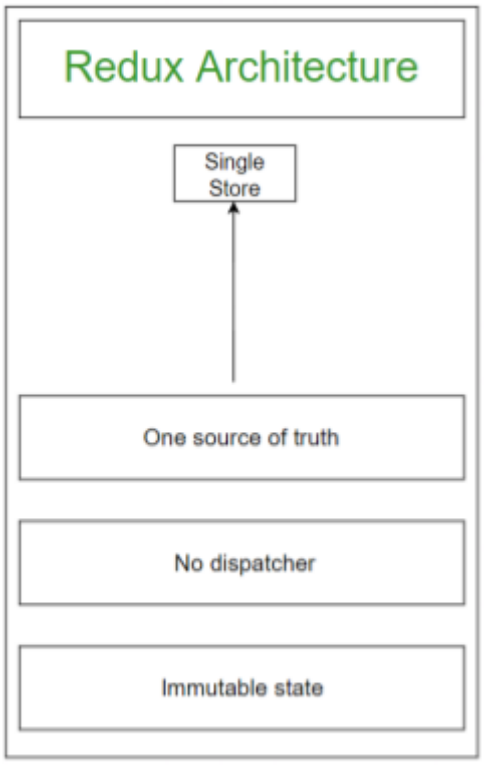

Figura 13: Diagrama de arquitectura Redux (Geeksforgeeks: (2020))

En Redux, los cambios en el estado se realizan mediante acciones y funciones reductoras. Estas funciones toman el estado anterior de la aplicación y la acción específica, y generan el siguiente estado de manera inmutable. Esto significa que el estado anterior no se modifica directamente; en su lugar, se genera un nuevo estado basado en el anterior. [15]

Introducción a NgRx

La librería NgRx toma las ideas fundamentales de Redux y las adapta para su uso en aplicaciones Angular, aprovechando el poder de RxJS para gestionar un estado global y facilitar la comunicación entre componentes. NgRx Store, el componente central de esta librería, permite mantener un estado predecible y consistente en aplicaciones Angular, y es particularmente útil en aplicaciones que requieren alta escalabilidad y rendimiento.

NgRx se enfoca principalmente en dos escenarios clave [21]:

1. **Compartir datos entre diferentes componentes**: facilita la transmisión de datos a través de diferentes partes de la aplicación sin necesidad de pasar información manualmente entre los componentes.
2. **Estado global para la reutilización de datos**: permite mantener un estado centralizado que puede ser compartido y utilizado por diversos componentes de la aplicación.

El siguiente diagrama ilustra el flujo general del estado en una aplicación que utiliza NgRx:

Figura 14: Diagrama de los bloques principales de NgRx (Working Software: Roos, P. (2022))

NgRx ofrece varios componentes esenciales para gestionar el estado, como acciones, reductores, selectores y efectos. Estos componentes trabajan en conjunto para asegurar que el estado se mantenga consistente y que los componentes puedan reaccionar ante cambios de manera eficiente.

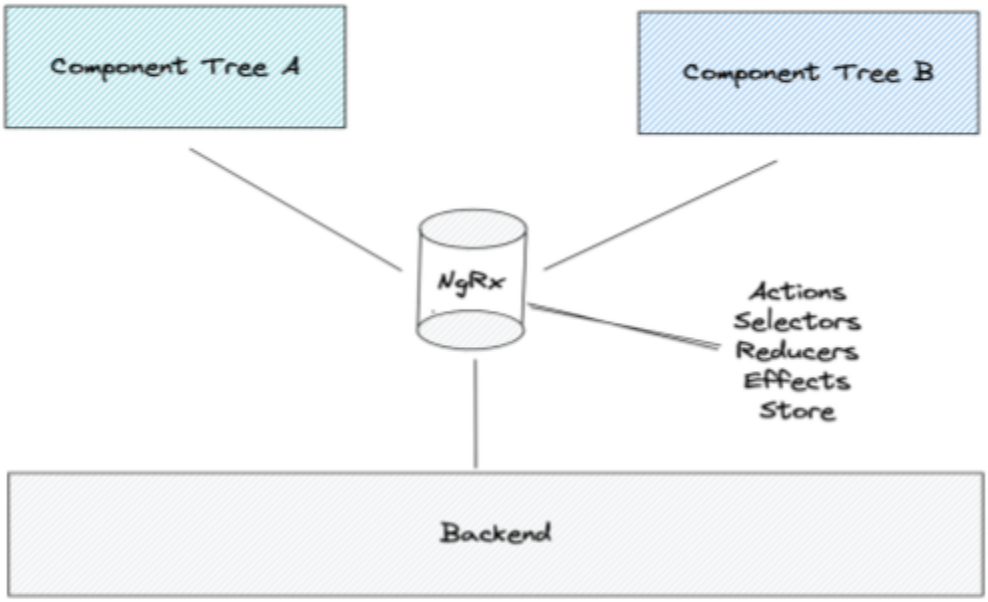

Figura 15: Tienda global de NgRx (Working Software: Roos, P. (2022))

Funcionamiento de NgRx

NgRx implementa el patrón de flujo unidireccional de datos, siguiendo el principio de que los datos deben fluir en una sola dirección, lo que garantiza que el estado sea predecible y fácil de seguir. Los principales elementos de este flujo son los siguientes:

- **Almacén (Store)**: El almacén es el contenedor central del estado, que mantiene los datos en una estructura inmutable. Los cambios en el estado sólo se pueden realizar a través de **acciones**, las cuales son procesadas por **reductores**.
- **Reductor (Reducer)**: El reductor es responsable de procesar las acciones y generar el nuevo estado de la aplicación. Recibe el estado actual y la acción a aplicar, y devuelve el nuevo estado.
- **Acción (Action)**: Las acciones son objetos que describen qué cambios deben realizarse en el estado. Estas acciones se envían al almacén para que sean procesadas por el reductor correspondiente. Además pueden ser observadas por uno o más efectos.
- **Selectores (Selectors)**: Los selectores son funciones que permiten acceder a partes específicas del estado almacenado. Proporcionan una forma eficiente de obtener solo los datos necesarios, lo que mejora la modularidad y el rendimiento de la aplicación.
- **Efectos (Effects)**: Los efectos son responsables de gestionar tareas asincrónicas o efectos secundarios, como la realización de peticiones HTTP o la interacción con APIs externas. Los efectos permiten que las acciones se procesen de manera asíncrona sin bloquear el flujo de la aplicación. [11]

4.8 Desarrollo de tareas utilizando RxJS, Lucy y CI/CD

En esta sección se describe la aplicación práctica de los conocimientos adquiridos en las herramientas Lucy, RxJS y los principios de CI/CD a través de la resolución de tareas específicas. A continuación, se exponen las soluciones implementadas para las distintas tareas asignadas, destacando cómo cada herramienta y concepto fue aplicado estratégicamente para alcanzar los objetivos planteados.

Durante el encuentro de planificación (Planning) previo al inicio del sprint, el equipo de desarrollo, junto con el líder técnico, determinó la asignación de las siguientes dos tareas:

- Construir la pantalla de inicio para el proceso de Onboarding de Empresas.
- Desarrollar la pantalla para el ingreso del CUIT.

El objetivo de ambas tareas era garantizar que las pantallas a desarrollar fueran completamente responsivas, adaptándose a las diferentes dimensiones de los dispositivos que utilizan los usuarios, desde teléfonos móviles hasta monitores de escritorio, asegurando una experiencia de usuario consistente y satisfactoria.

Una vez iniciado el sprint, se comenzó con el análisis de viabilidad de construir las pantallas como componentes *responsive*. Esto se debía a que es un requisito esencial que el producto final se adapte automáticamente a distintos tamaños y resoluciones de pantalla. Para lograrlo, el primer paso fue investigar la documentación de Lucy para verificar si existía alguna forma de configurar el comportamiento *responsive* a nivel de aplicación, mediante algún servicio.

Figura 16: Sección Servicios en documentación de Lucy (Propia)

El siguiente paso consistió en integrar el servicio proporcionado por Lucy al constructor del componente principal (*AppComponent*), implementando una función que detectara el tipo de dispositivo que utiliza la aplicación.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss'],
})
export class AppComponent {

  constructor(
    private changeDetectorRef: ChangeDetectorRef,
    private responsiveService: LyResponsiveService
  ) {}

  ngOnInit() {
    this.detectDevice();
  }

  detectDevice() {
    this.responsiveService.screen$
      .pipe(
        map((screen: ScreenConfig) => screen.screen),
        distinctUntilChanged()
      )
      .subscribe((screen: ScreenType) => {
        if (screen === ScreenType.mobile || screen === ScreenType.tablet) {
          this.store.dispatch(SetDeviceMood({ isMobile: true }));
        } else {
          this.store.dispatch(SetDeviceMood({ isMobile: false }));
        }
        this.changeDetectorRef.markForCheck();
      });
  }
```

Figura 17: Función para detectar tipo de dispositivo (Propia)

En dicha función, se utilizó el operador *pipeable map* de RxJS, que transforma cada valor emitido por el observable, aplicando una función que retorna solo el atributo *screen* de la clase *ScreenConfig*, el cual contiene el tipo de pantalla del dispositivo. Con este nuevo observable, nos suscribimos a los cambios y, mediante una condición que verifica si el dispositivo tiene resoluciones propias de un teléfono móvil o tablet (en cuyo caso asumimos que es un dispositivo móvil), almacenamos un valor booleano en un objeto denominado *isMobile*, el cual fue creado en el *store* de NgRx (que se detallará más adelante).

Este valor booleano nos permitió aplicar el Patrón Declarativo de Acceso a Datos (Declarative Data Access Pattern) en las vistas (templates) de los componentes de la aplicación, con el objetivo de construir una interfaz responsiva para cada tipo de dispositivo.

En Angular, al usar un patrón declarativo de acceso a datos, podemos delegar la ejecución de código para recuperar los datos cuando sea necesario en el template. Gracias a la canalización asíncrona (*async* pipe), no necesitamos suscribirnos ni cancelar las suscripciones a un observable, ya que Angular se encarga de ello de manera automática. Al cargar el template, la canalización obtiene los datos y los muestra tal como se reciben.

A continuación, se muestra cómo se implementó este patrón en la pantalla inicial de la aplicación:

Figura 18: Obtención de observable que contiene valor de tipo de dispositivo (Propia)

En el código del componente, se utiliza el decorador *@Component* para transformar una clase TypeScript en un componente de Angular. En los metadatos se incluyen el selector, la plantilla y los estilos. Se declaró la clase *SelectFlowComponent*, que implementa la interfaz *OnInit*, asegurando que la clase cumpla con el contrato de tener el método *ngOnInit*. Esto garantiza que Angular llame automáticamente a dicho método durante la inicialización del componente, lo que permite la asignación de variables antes de que el componente se renderice completamente.

Dentro de la clase, se declaró una variable de tipo observable llamada *isMobileIndicator$*, la cual, al ejecutar el método *ngOnInit*, se asigna con el valor del objeto *isMobileDevice* almacenado en el *store*. En el template, se utilizó la directiva *ngIf* de Angular para mostrar u ocultar un elemento del DOM dependiendo de si la condición evaluada es verdadera. Si el valor del observable *isMobileIndicator$* es *true*, se renderiza la vista para dispositivos móviles o tablets; de lo contrario, se muestra una vista diseñada para computadoras de escritorio o laptops.

select-flow.component.html M

src > app > modules > components > select-flow > select-flow.component.html > ...

Go to component | You, 1 second ago | 1 author (You)

```
<ng-container *ngIf="!(isMobileIndicator$ | async)">
    <app-header [showLittleArrow]="false"></app-header>
    <ly-flex-layout data-cy="flex-layout-space-3-7" [space]="[2, 6, 2]">
        <div></div>
        <div>
            <ly-block-container class="custom-block-container" type="paddingHorizontalLarge">
                <app-select-flow-form (onSubmitEmitter)="onSubmit($event)"></app-select-flow-form>
            </ly-block-container>
        </div>
        <div></div>
    </ly-flex-layout>
</ng-container>
<ng-container *ngIf="(isMobileIndicator$ | async)">
    <app-header [showLittleArrow]="false"></app-header>
    <ly-block-container class="custom-mobile-container" type="paddingHorizontal">
        <app-select-flow-form (onSubmitEmitter)="onSubmit($event)"></app-select-flow-form>
    </ly-block-container>
</ng-container>
```

Figura 19: Implementación de Patrón Declarativo de Acceso a Datos (Propia)

Figura 20: Pantalla inicial para tipo de dispositivo desktop (Propia)

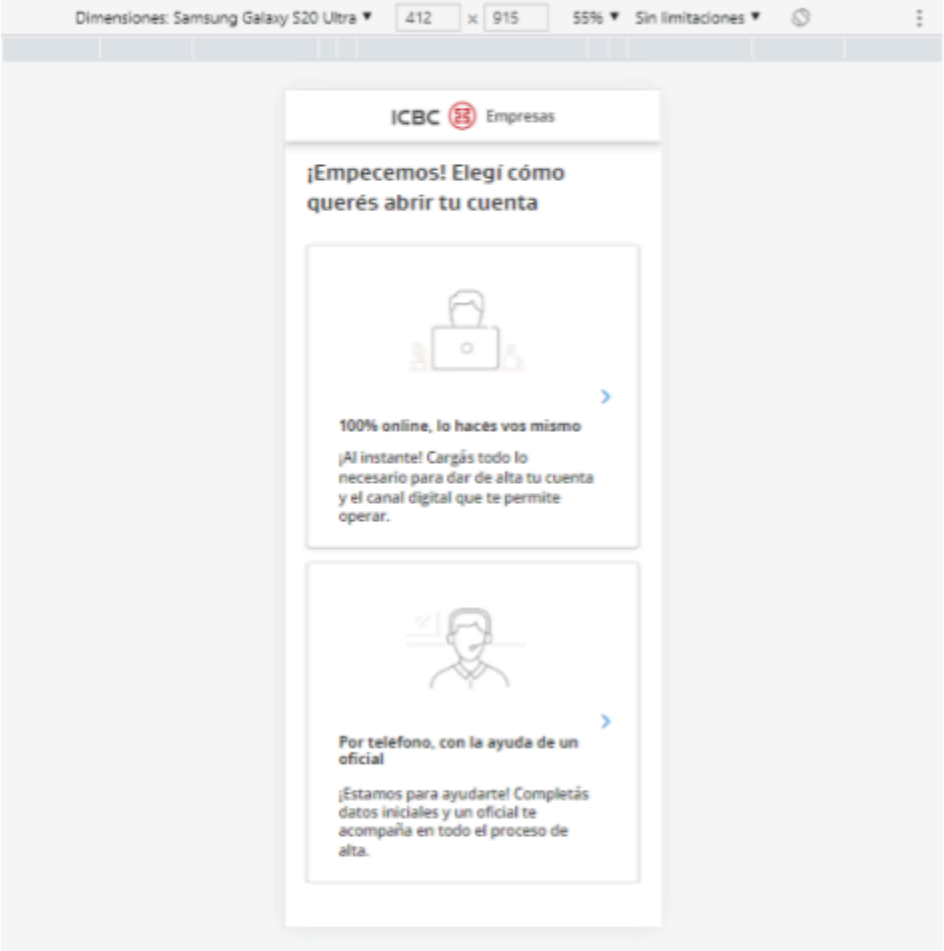

Figura 21: Pantalla inicial para tipo de dispositivo mobile (Propia)

Una vez completada esta tarea y tras realizar pruebas en el entorno local para validar el funcionamiento correcto, el siguiente paso fue enviar los cambios al repositorio remoto del proyecto en GitLab. Después de hacer el *push* de los cambios, se creó un Merge Request (MR) para fusionar la nueva funcionalidad en la rama principal de desarrollo (*dev*).

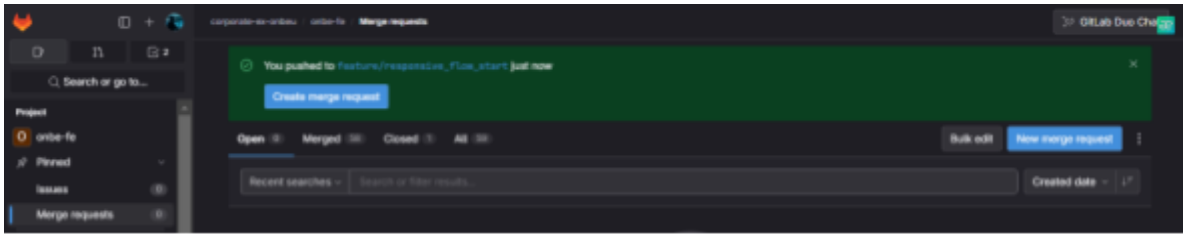

Figura 22: Creación de merge request en Gitlab (Propia)

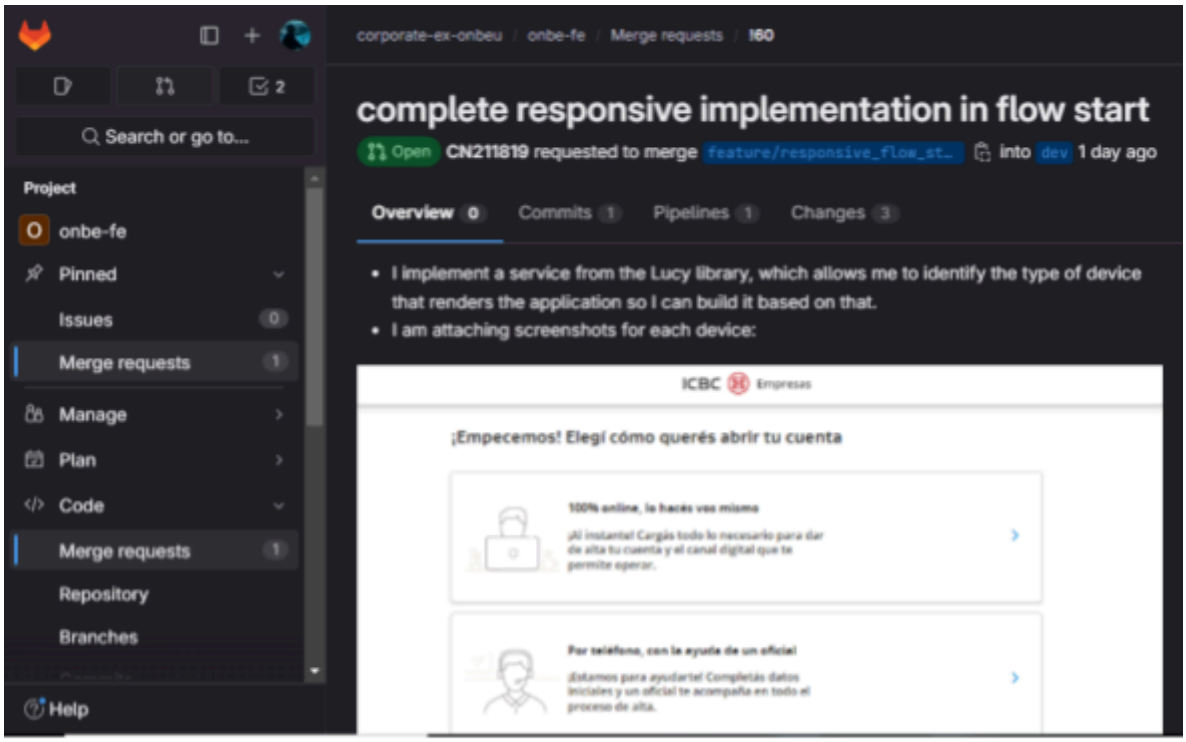

Figura 23: Visualización de merge request creado (Propia)

Antes de que los revisores comiencen el proceso de *code review*, verificamos que los tres pipelines iniciales de configuración se hayan completado con éxito:

A continuación se describe la función de cada uno en el orden en el que se disponen en la imagen:

1) **Pre-config**: Trae todos los repositorios con alguna configuración y establece configuraciones adicionales, como el puerto del servidor de *k3s*.
2) **Check-dependencies**: Valida que el archivo *package.json* no contenga dependencias con versiones incrementales.
3) **Quality**: Evalúa el cumplimiento del porcentaje mínimo de cobertura de pruebas unitarias mediante SonarQube (en este caso, 70%).

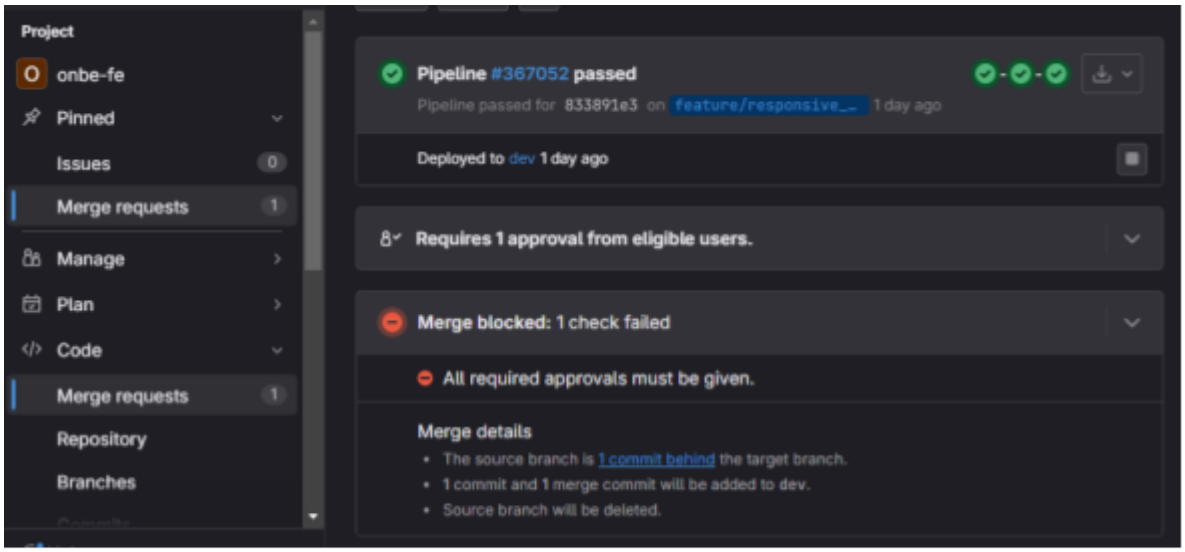

Figura 24: Pipelines iniciales de configuración (Propia)

Si todos los pipelines pasan sin problemas, el MR es revisado por otro miembro del equipo. Si el código sigue las convenciones y estándares establecidos, el MR es aprobado.

Una vez aprobado, el paquete es entregado al ambiente de *dev* mediante la implementación de nuevos pipelines:

Figura 25: Despliegue en ambiente dev (Propia)

1) **Build-dev**: Se encarga de compilar los archivos estáticos.
2) **Deploy-nexus**: Publica los archivos en el servidor *nexus*.
3) **Deploy-dev**: Publica los archivos en el servidor web (*nginx*) de *dev*.

Luego, se pudo continuar con el segundo requerimiento de la tarea: desarrollar la pantalla para el ingreso del CUIT de la empresa.

Para ello, se utilizó el mismo patrón declarativo de acceso a datos implementado anteriormente para construir la vista, adaptada a dispositivos móviles y de escritorio.

- Para dispositivos desktop:

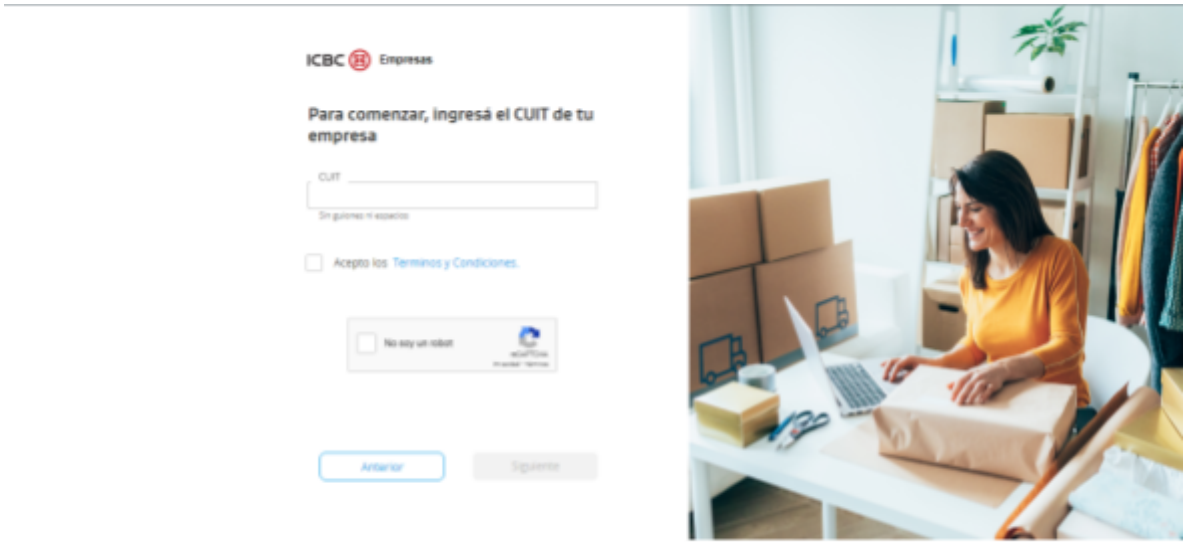

Figura 26: Pantalla de ingreso de cuit para dispositivos desktop (Propia)

- Para dispositivos mobile:

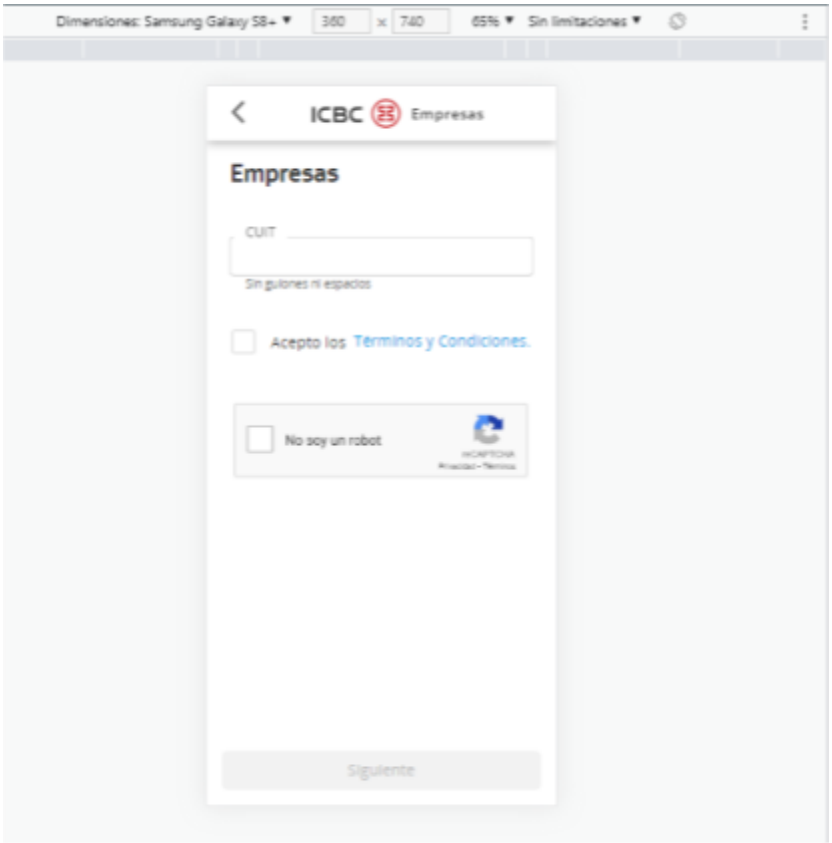

Figura 27: Pantalla de ingreso de cuit para dispositivos mobile (Propia)

Finalmente, una vez que todo estuvo integrado correctamente y pasó las pruebas locales, se realizó un nuevo *merge request* para integrar los cambios en la rama *master* (rama de pruebas).

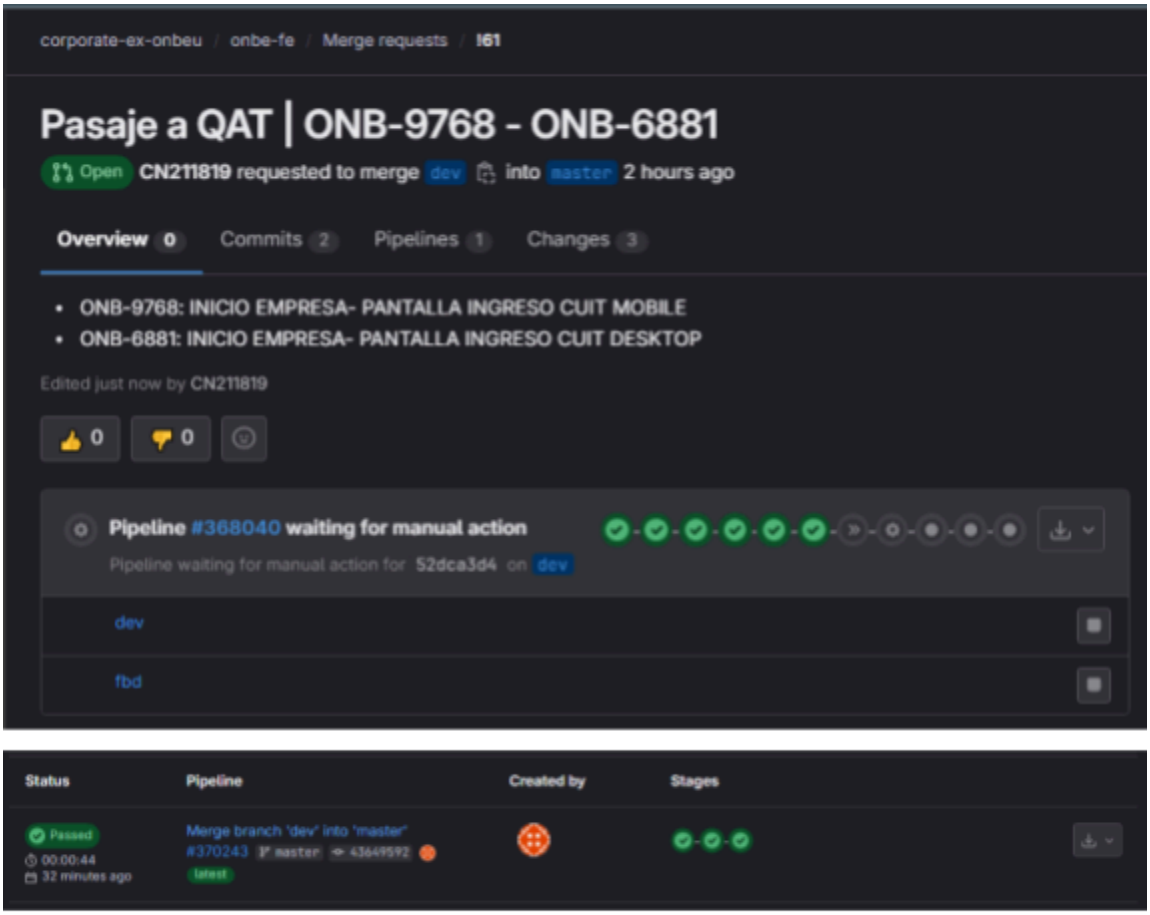

Figura 28: Merge request desde el branch dev al branch master (Propia)

Tras la aprobación de este MR y la ejecución exitosa de los pipelines, se procedió con la creación de un *tag release*, el cual se vincula con las historias de usuario correspondientes, según el sistema Jira.

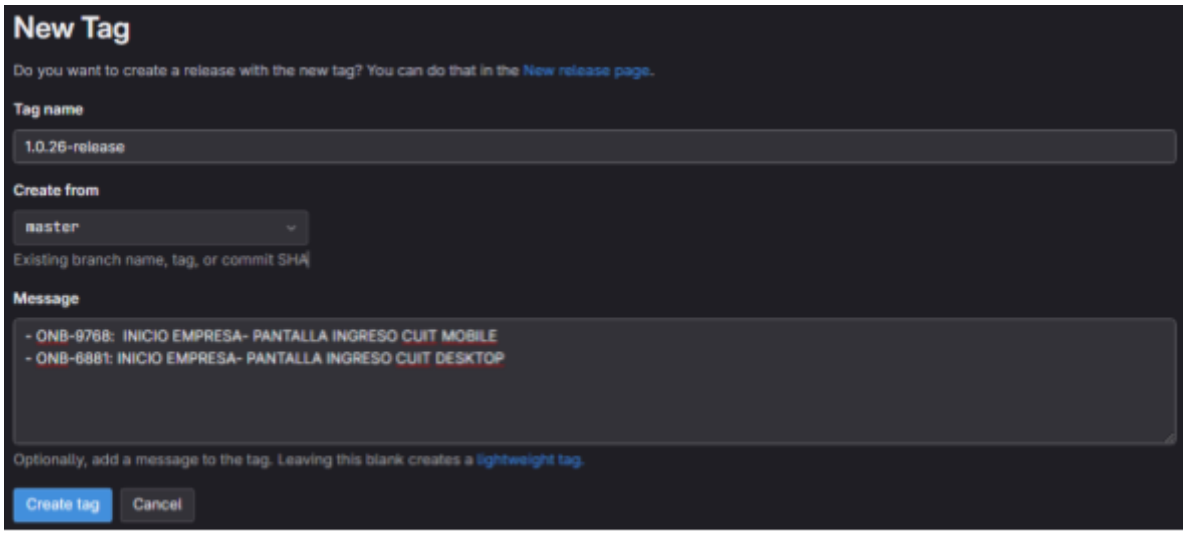

Figura 29: Creación de tag release (Propia)

Esto hace que se ejecuten automáticamente una serie de pipelines, donde los 3 primeros son los ya mencionados Pre-config, Check-dependencies, Quality y el resto, que comienzan con el build-qat solo pueden ser ejecutados manualmente por el equipo de Operaciones.

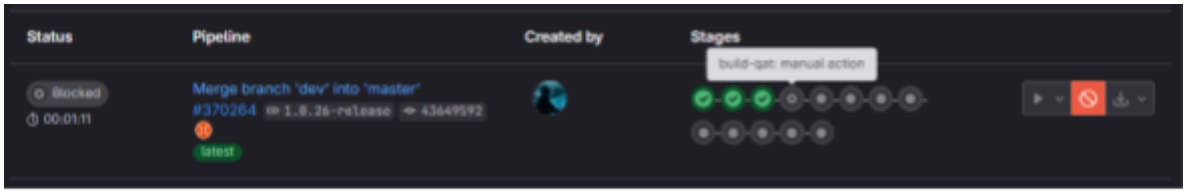

Figura 30: Despliegue en ambiente qat (Propia)

Una vez concluidas las pruebas, si el equipo de QA pudo terminar cada script de prueba con éxito, está en condiciones de dar por finalizadas las historias.

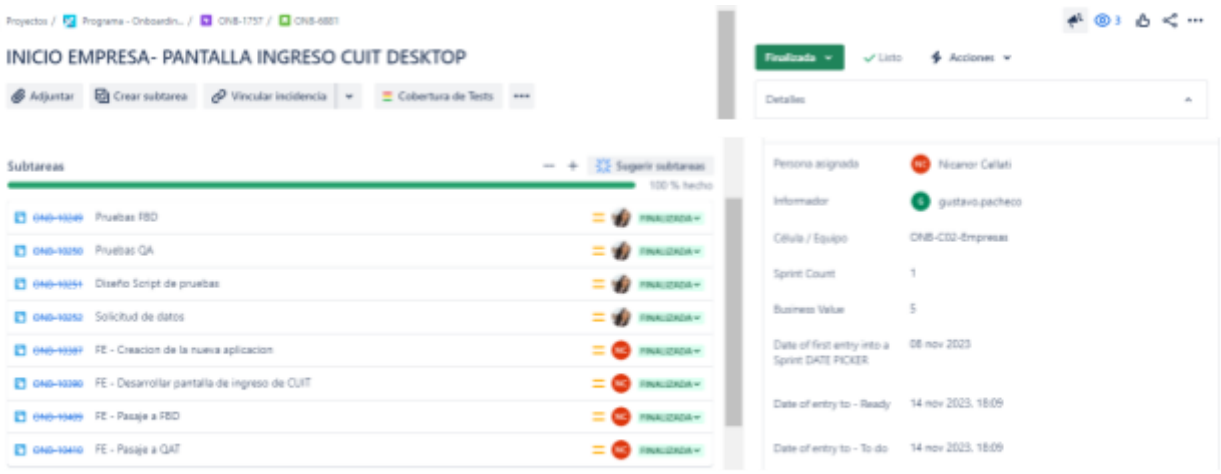

Figura 31: Finalización de historia de usuario pantalla de ingreso cuit desktop (Propia)

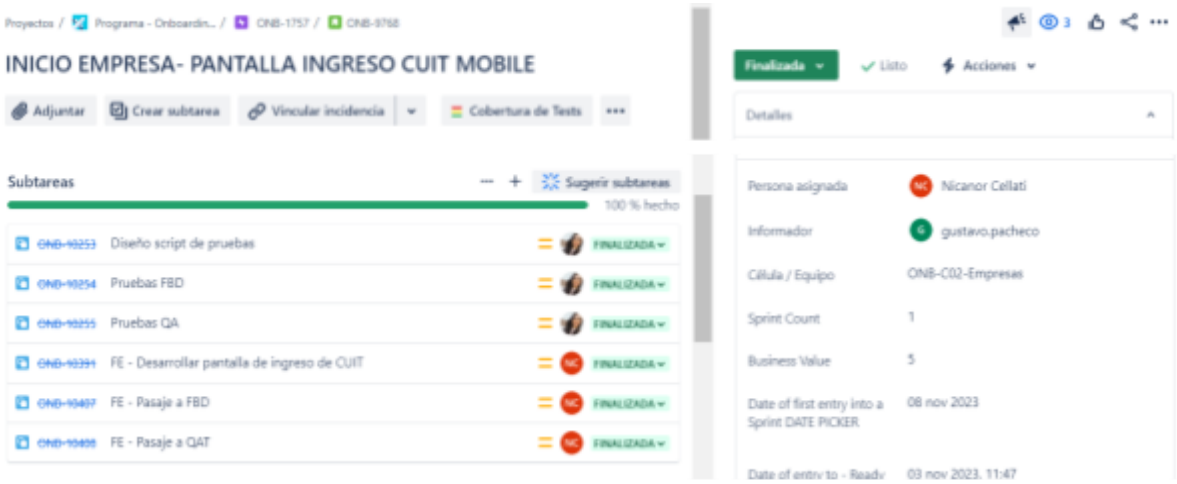

Figura 32: Finalización de historia de usuario pantalla de ingreso cuit mobile (Propia)

4.9 Desarrollo de tareas utilizando NgRx

Con el objetivo de abordar los desafíos asociados a las tareas asignadas en este periodo, se decidió aplicar los principios fundamentales de NgRx para construir una solución escalable y fácilmente mantenible. En esta sección se describirá cómo se pusieron en práctica los conceptos de Store, Actions, Reducers, Effects, y Selectors.

En el contexto de un nuevo sprint, se me asignó la construcción de una nueva Historia de Usuario (HU), la cual se presenta en la pantalla inicial desarrollada en la sección anterior. Esta opción se puede seleccionar al acceder a la segunda opción de la interfaz.

Figura 33: Descripción de historia de usuario proceso asistido (Propia)

Figura 34: Opción para iniciar el proceso asistido (Propia)

Para considerar completada la HU, se deben cumplir con los siguientes criterios de aceptación, que detallan las funcionalidades requeridas para el desarrollo del proceso asistido.

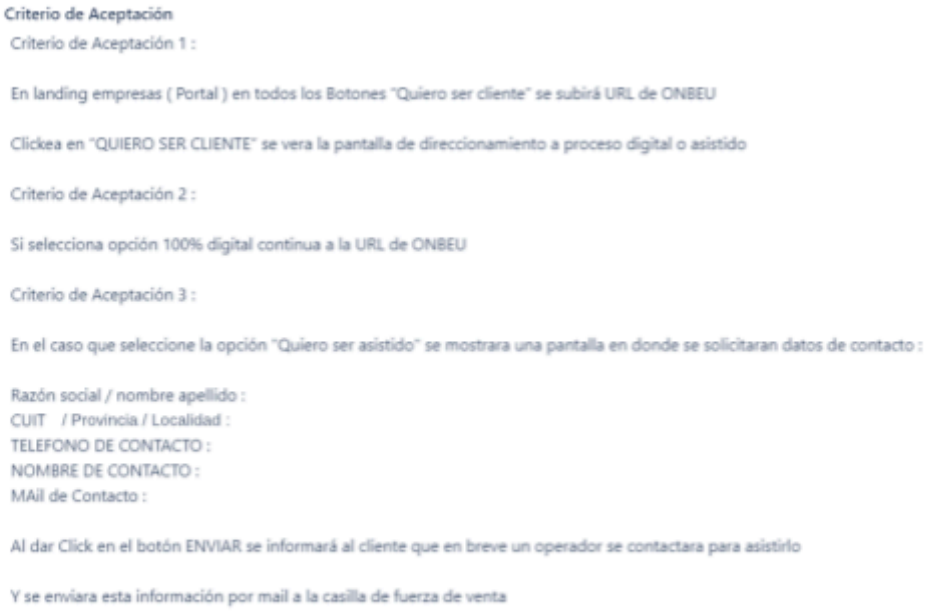

Figura 35: Criterios de aceptación para historia de usuario de proceso asistido (Propia)

Aplicación de NgRx en la Resolución de Criterios de Aceptación

La aplicación de NgRx se destaca en el tercer criterio de aceptación, cuyo primer paso consistió en desarrollar los templates que contienen los datos solicitados en la HU. Para ello, se utilizaron los prototipos adjuntos por el equipo de diseño.

Figura 36: Template con formulario de datos relativos a la empresa (Propia)

Uno de los elementos clave es un formulario de dos pasos. En el primer paso, se recogen los datos relativos a la empresa. Dentro de este formulario, se solicitó al usuario seleccionar la provincia en la que se encuentra la empresa. Para facilitar esta interacción, se construyó un componente de tipo **Select**, el cual al ser desplegado permite al usuario elegir entre una lista de provincias.

Dado que el listado de provincias debe ser consultado desde un servicio backend, se optó por almacenar los datos obtenidos en la **Store** de NgRx. Esta decisión permite evitar realizar múltiples consultas a la API, ya que el listado de provincias puede ser consumido desde el frontend las veces que sea necesario sin la necesidad de realizar una nueva solicitud cada vez.

Implementación de NgRx para Almacenar el Listado de Provincias

A continuación, se describe el proceso de implementación de NgRx para almacenar el listado de provincias.

Paso 1: Construcción del Store

El primer paso consistió en definir el Store de la aplicación, que en este caso se estructuró de la siguiente manera:

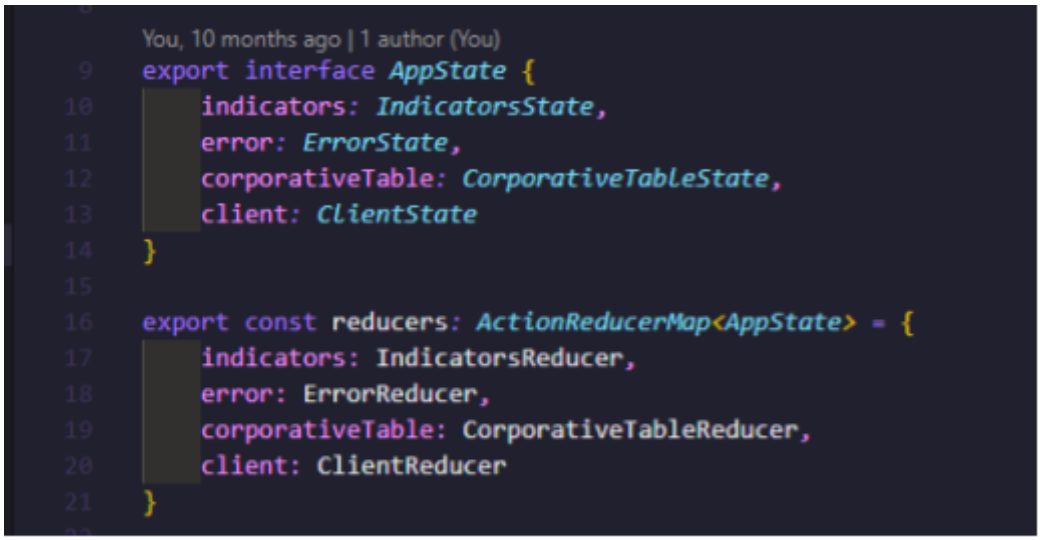

Figura 37: Definición del Store (Propia)

- AppState contiene el estado global de la aplicación, e incluye estados particulares para cada módulo que utilice el Store.
- En este caso, se optó por almacenar el listado de provincias dentro del corporativeTableState, que hace referencia al microservicio correspondiente.

Para ello, se utilizó el objeto actionReducerMap, que asocia los tipos de acción con las funciones reductoras correspondientes. Este enfoque facilita una gestión más eficiente del estado de la aplicación, al evitar el uso de un único reductor gigante y difícil de mantener.

Paso 2: Implementación del Reductor

El siguiente paso fue la creación del reductor CorporativeTableReducer, en el cual se definieron los siguientes componentes:

Figura 38: Creación del reductor CorporativeTableReducer (Propia)

1. **Interfaz** para CorporativeTableState: En este caso, la interfaz incluye únicamente el objeto listProvinces.
2. **Estado inicial** (initialState): El estado inicial define un array vacío para el listado de provincias.
3. **Reductor**: La constante CorporativeTableReducer se asigna a la función createReducer, provista por NgRx, la cual toma el estado actual y una acción (por ejemplo, SuccessGetProvinces), y devuelve un nuevo estado actualizado.

Paso 3: Definición de Acciones

Se definieron tres acciones clave para el flujo de datos de la aplicación:

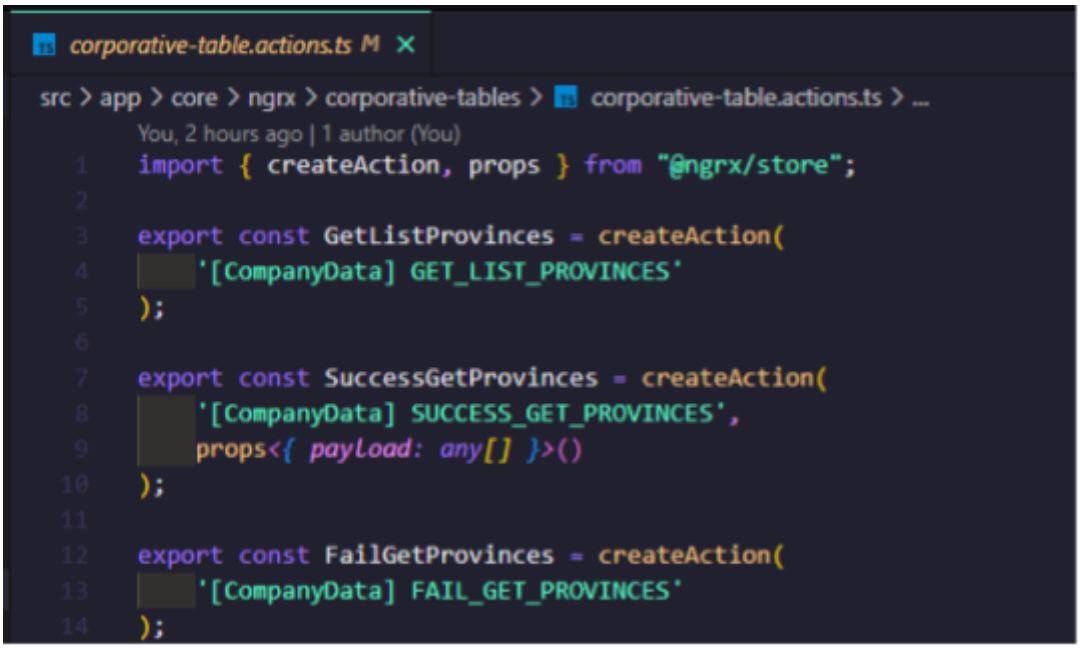

Figura 39: Definición de acciones (Propia)

1. **GetListProvinces**: Acción despachada cuando se solicita el listado de provincias.
2. **SuccessGetProvinces**: Acción despachada cuando el servicio backend devuelve una respuesta exitosa con el listado de provincias.
3. **FailGetProvinces**: Acción despachada cuando ocurre un error en la solicitud, con el fin de informar al usuario sobre lo ocurrido.

Las acciones se definieron utilizando la función createAction, la cual toma dos parámetros: el tipo de acción (un identificador único) y una carga útil (opcional) que puede incluir cualquier dato adicional relacionado con la acción.

Paso 4: Implementación del Efecto

El siguiente paso consistió en definir el Effect que maneja la llamada a la API para obtener el listado de provincias. Se creó el efecto getListProvinces$, que es de tipo observable y se define mediante la función createEffect.

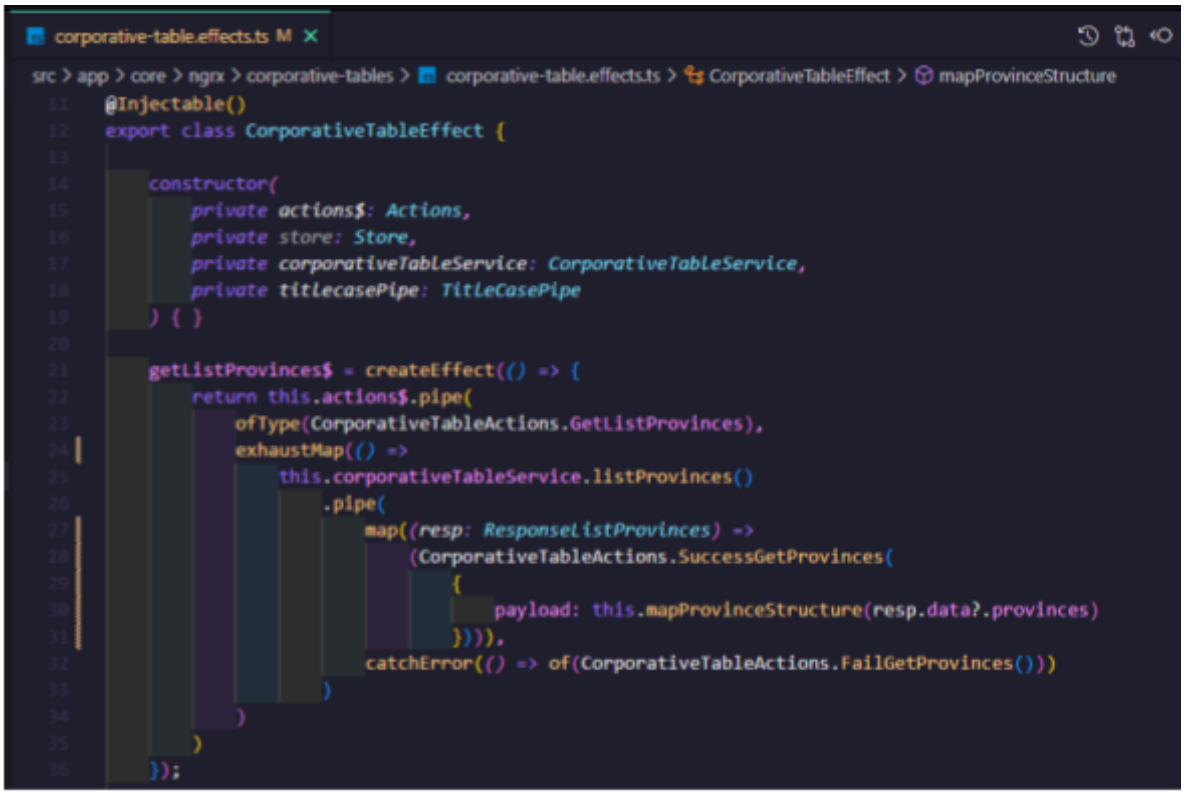

Figura 40: Creación del efecto getListProvinces$ (Propia)

Dentro del flujo de acciones (this.actions$), se utiliza el operador ofType para escuchar la acción **GetListProvinces**. Luego, el operador exhaustMap asegura que no se envíen solicitudes simultáneas, evitando la ejecución de múltiples solicitudes antes de que la primera termine. En caso de éxito, se despacha la acción **SuccessGetProvinces**, pasando los datos recibidos como payload. Si ocurre un error, se despacha la acción **FailGetProvinces**.

Paso 5: Definición del Selector

Para extraer el listado de provincias del estado, se definieron dos selectores utilizando las funciones createFeatureSelector y createSelector. Estos selectores permiten acceder a la porción específica del estado correspondiente a las provincias y facilitar su consumo en el componente correspondiente.

Figura 41: Creación de selectores (Propia)

Paso 6: Consumo del Listado en el Componente

Finalmente, se utilizó el ciclo de vida ngOnInit para suscribirse al selector **getListOfProvinces**. Al recibir la actualización del estado, el listado de provincias se asigna a una variable local y se utiliza para cargar el componente Select en el template.

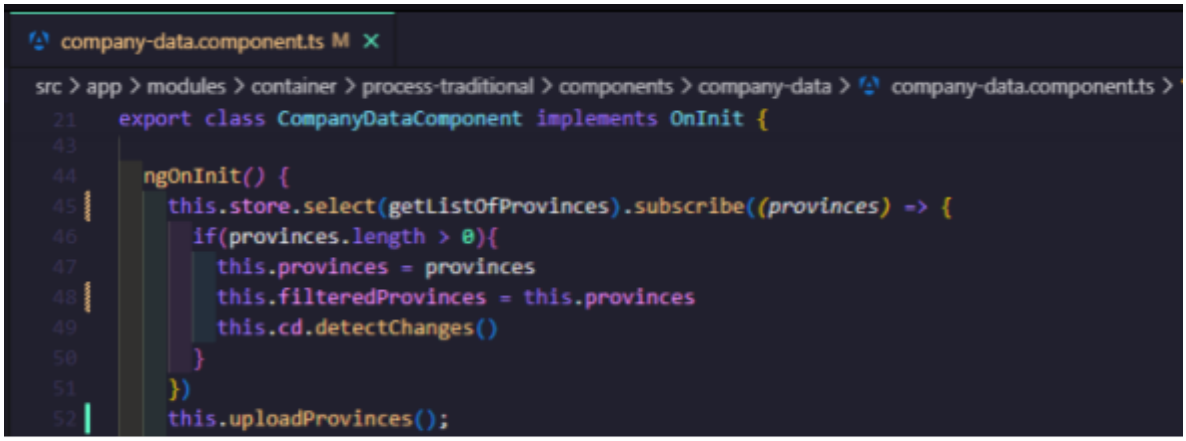

Figura 42: Utilización de selector getListOfProvinces (Propia)

Además, se implementó la función uploadProvinces, que valida si el listado de provincias ya ha sido cargado desde la Store. Si aún no se ha despachado la acción GetListProvinces, se realiza dicha acción.

Figura 43: Lógica de la función uploadProvinces (Propia)

Una vez que el usuario selecciona una provincia, el componente Select se actualiza correctamente, mostrando la lista de provincias de forma eficiente.

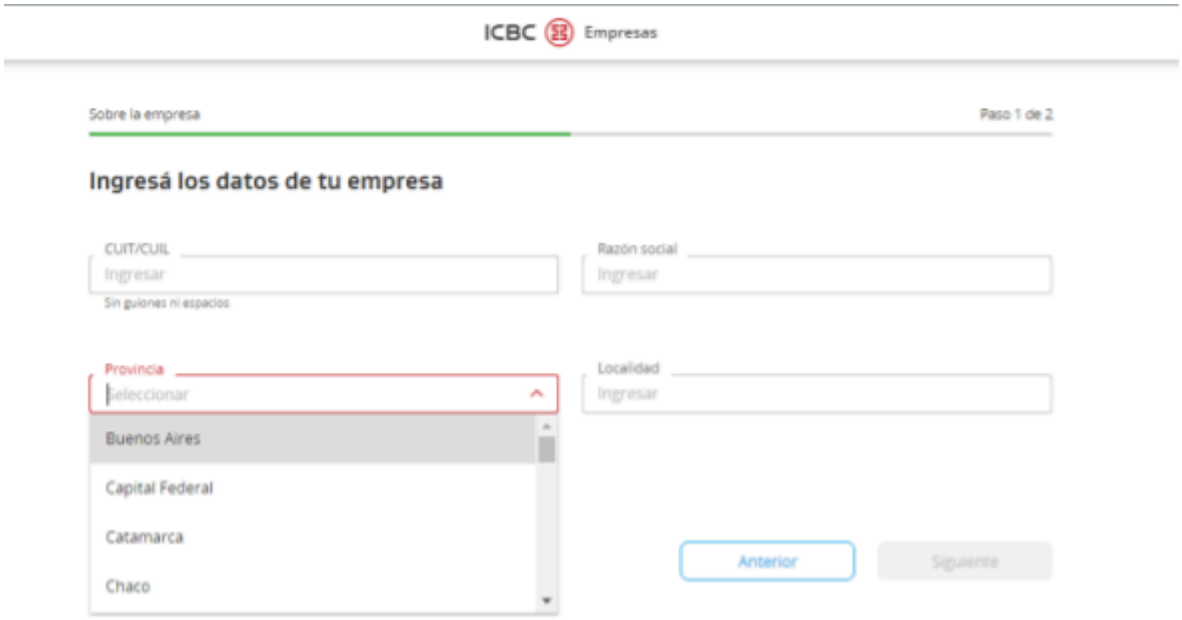

Figura 44: Visualización del listado de provincias en el template (Propia)

Conclusión del Desarrollo

Concluido el desarrollo del primer paso del formulario, se continuó con la implementación del segundo paso, donde el usuario proporciona más detalles relacionados con el proceso asistido. Al finalizar la carga de datos y hacer clic en "Siguiente", se muestra un feedback al usuario notificando que su proceso ha finalizado.

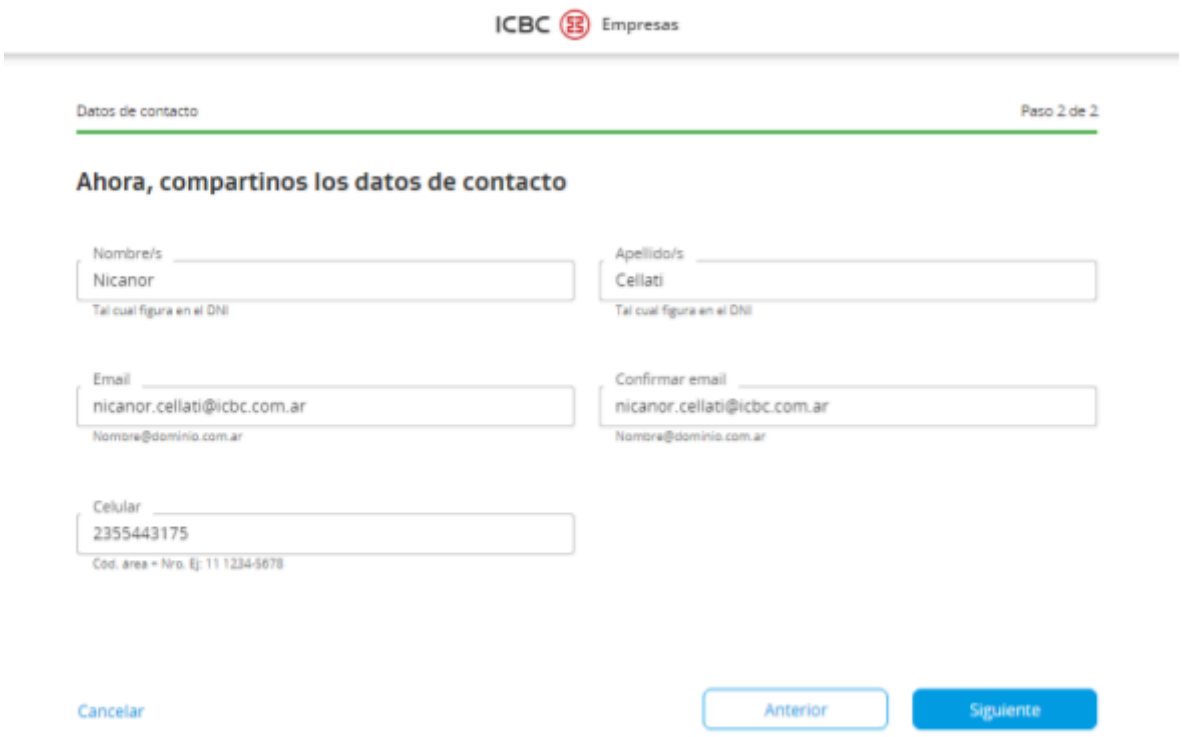

Figura 45: Template con formulario de datos de contacto (Propia)

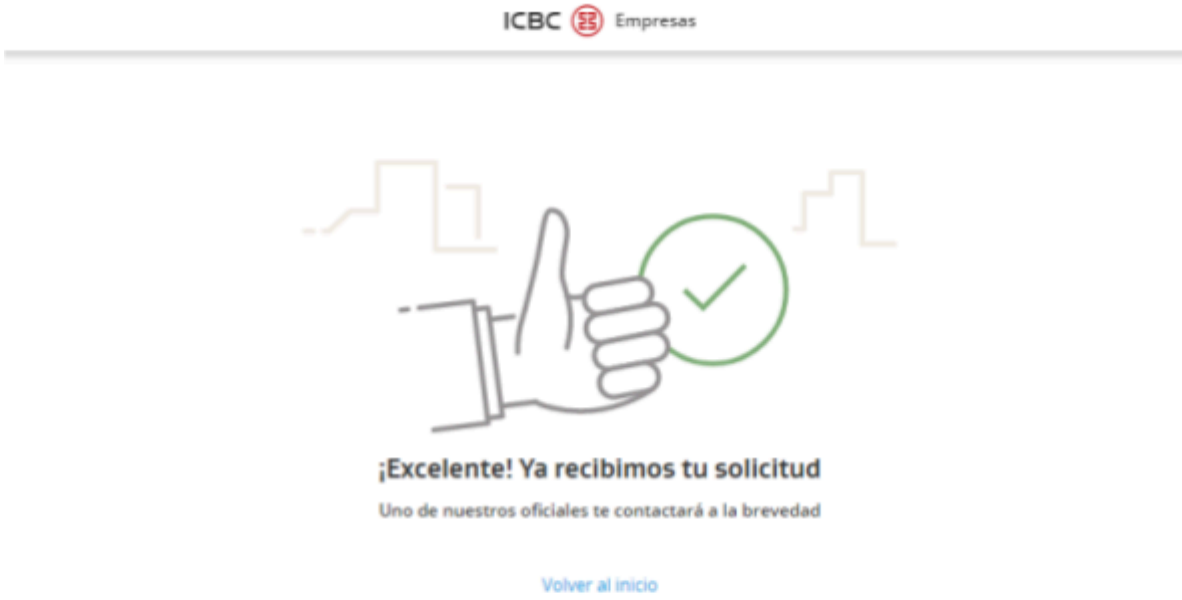

Figura 46: Feedback de proceso finalizado con exito (Propia)

El desarrollo fue finalmente publicado en el ambiente de qat mediante el proceso ya descrito en la sección anterior. Posteriormente, se notificó al equipo de QA a través de la tarjeta en Jira. Tras realizar las pruebas correspondientes y verificar el funcionamiento, la Historia de Usuario (HU) fue completada con éxito.

Figura 47: Finalización de historia de usuario de proceso asistido (Propia)

4.10 Despliegue de nuevas funcionalidades en producción

Con el fin de organizar de manera efectiva la puesta en producción y asegurar una entrega continua de valor, nos aseguramos de cumplir con una serie de requisitos definidos en colaboración con los referentes de cada área del equipo. Además, nos apoyamos en herramientas como el tablero de Miro para gestionar y coordinar las actividades.

Introducción

La puesta en producción de un incremento representa una fase crucial dentro del proceso de desarrollo ágil utilizando Scrum. En esta sección, se describirán los pasos necesarios para llevar a cabo este proceso, con el objetivo de asegurar que todo el equipo esté alineado y entienda las acciones a realizar para garantizar una transición fluida al entorno de producción.

Objetivo

El principal objetivo de la puesta en producción es desplegar el incremento desarrollado durante el Sprint al entorno de producción, de forma que los usuarios finales puedan beneficiarse de las nuevas funcionalidades o mejoras. Este proceso debe realizarse de manera eficiente y segura, minimizando las interrupciones para los usuarios finales.

Pasos para la Puesta en Producción

1. Revisión de Sprint

Responsables: Scrum Master y Product Owner
Descripción: Durante la reunión de revisión del Sprint, se revisa el trabajo completado y se confirma que todos los incrementos cumplen con la Definición de Hecho (DoD). Asimismo, se verifica que los incrementos están listos para ser desplegados en el entorno de producción.
Resultado esperado: Aprobación del incremento para su despliegue en producción.

2. Preparación para el Despliegue

Responsables: Equipo de Desarrollo
Descripción: En esta fase, el equipo de desarrollo prepara todos los artefactos necesarios para el despliegue, lo cual incluye la creación de paquetes de despliegue y la actualización de la documentación correspondiente. La planificación de las tareas se gestiona a través de un tablero en Miro, donde se asignan responsabilidades específicas a cada miembro del equipo y se realiza un seguimiento del avance de las tareas. Una vez las tareas están definidas, se formalizan en Jira, enlazando aquellas implementaciones que están relacionadas o dependen de otros equipos.
Resultado esperado: Paquetes de despliegue listos y equipo alineado

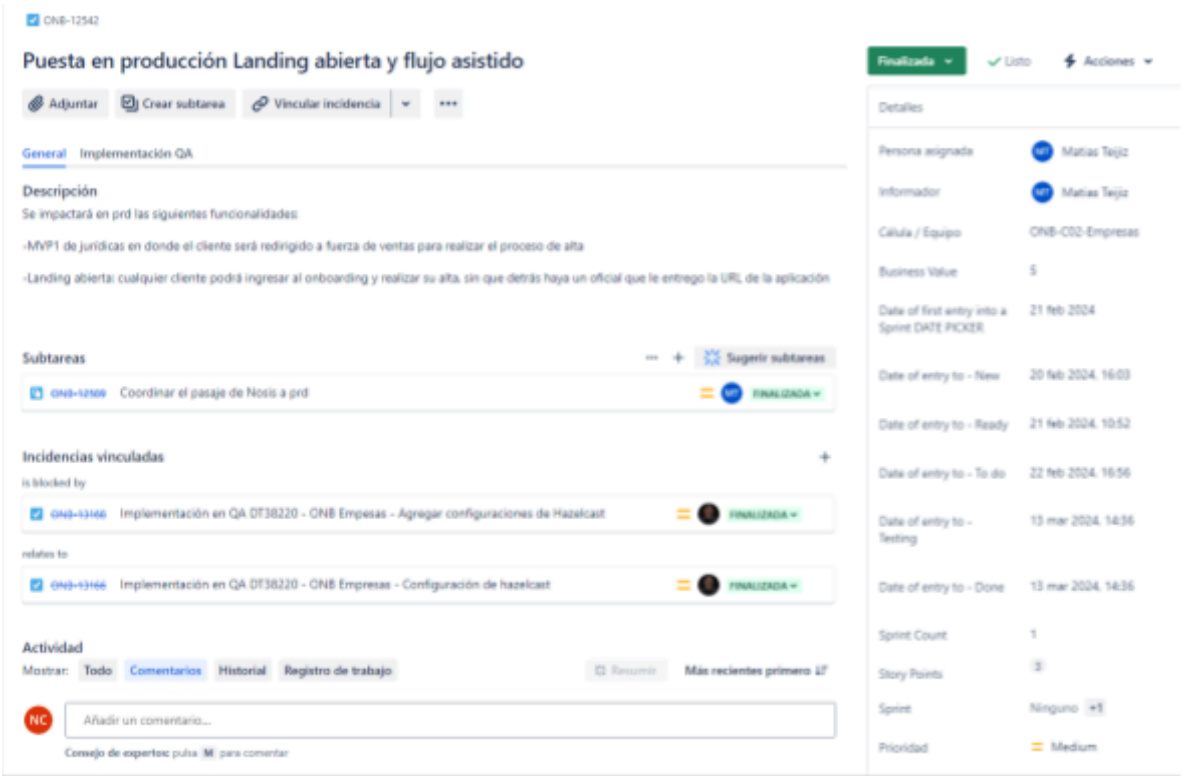

Figura 48: Tarjeta de implementación en producción (Propia)

3. Pruebas de Pre-Producción

Responsables: Equipo de QA
Descripción: El equipo de QA ejecuta pruebas exhaustivas en el entorno de pre-producción (ambiente de QA) con el fin de asegurar que el incremento funciona como se espera y que no existen errores críticos.
Resultado esperado: Validación del incremento en el entorno de pre-producción.

4. Planificación del Despliegue

Responsables: Arquitecto y DevOps
Descripción: El equipo de Arquitectura y DevOps se encarga de planificar el momento adecuado para realizar el despliegue, teniendo en cuenta el impacto mínimo para los usuarios finales. Además, se debe crear un plan de contingencia que contemple un respaldo en caso de que surjan problemas durante el proceso de despliegue.
Resultado esperado: Plan detallado de despliegue y plan de contingencia.

5. Despliegue al Entorno de Producción

Responsables: DevOps y Equipo de Desarrollo
Descripción: En esta fase, el equipo de DevOps y Desarrollo ejecuta el despliegue siguiendo el plan establecido. Es fundamental monitorizar el proceso para identificar y resolver cualquier problema que pueda surgir durante la implementación.
Resultado esperado: Incremento correctamente desplegado en el entorno de producción.

Conclusión

La puesta en producción de un incremento es un proceso colaborativo que requiere la participación activa de todo el equipo Scrum. Siguiendo estos pasos, podemos asegurar una transición suave y eficiente de los incrementos al entorno de producción, brindando valor continuo a nuestros usuarios finales y garantizando una entrega exitosa y sin contratiempos.

4.11 Monitoreo y observabilidad de infraestructura usando Dynatrace

Monitoreo versus Observabilidad

El monitoreo y la observabilidad son conceptos clave en la gestión de infraestructuras tecnológicas, pero cumplen funciones diferentes. El monitoreo se centra en detectar si un sistema está funcionando correctamente o no, y en identificar los problemas cuando algo sale mal. Esto se

realiza mediante el uso de un conjunto predefinido de métricas y registros. El monitoreo tiene un enfoque reactivo, ya que permite una respuesta rápida ante incidentes, abarcando paneles de control y alertas basadas en métricas ya definidas.

Por otro lado, la observabilidad proporciona una comprensión más profunda de cómo funciona un sistema y por qué algo no está funcionando correctamente. Es de naturaleza proactiva y se enfoca en el comportamiento del sistema. La observabilidad reduce el impacto y la duración de los incidentes, ofreciendo una visión más holística. Se fundamenta en tres pilares: métricas, registros y seguimientos, que abarcan una variedad de eventos, tales como registros de aplicaciones, registros del sistema, métricas de infraestructura y red, y seguimientos de transacciones. [8]

Relación entre Monitoreo y Observabilidad

El monitoreo y la observabilidad están estrechamente relacionados y se complementan mutuamente. Un sistema debe ser observable para poder ser monitoreado de manera efectiva. La observabilidad, de hecho, comienza donde termina el monitoreo. Para entender mejor la relación entre estos dos conceptos, es útil considerar dos aspectos clave: la cantidad de datos disponibles y el nivel de comprensión que se tiene sobre los sistemas en cuestión. [8]

Métricas Definidas

Dentro del monitoreo y la observabilidad, las métricas juegan un papel fundamental. A continuación, se detallan algunas métricas clave utilizadas en la supervisión de sistemas:

- **Latencia**: Mide el tiempo necesario para completar una acción. Esta métrica ayuda a identificar cuellos de botella en el rendimiento del sistema.
- **Tráfico**: Evalúa la carga o demanda que los componentes del sistema deben soportar. Permite correlacionar la degradación del servicio o las fallas con la necesidad de incrementar recursos en distintas etapas de carga.
- **Errores**: Proporcionan información sobre el estado de los componentes del sistema. Algunos errores deberían activar alertas inmediatas, mientras que otros pueden ser aceptables hasta que la tasa de error supere un umbral predefinido.
- **Saturación**: Mide la utilización de los recursos del sistema. Esta métrica es esencial para identificar cuándo los recursos pueden estar al límite, lo que podría afectar el rendimiento general del sistema.

Dynatrace

Dynatrace es una herramienta integral que facilita la supervisión de infraestructura, aplicaciones y la experiencia digital del usuario, todo en una única plataforma automatizada. Gracias a su capacidad para integrar la infraestructura de la nube, el rendimiento de las aplicaciones y el

monitoreo de la experiencia del usuario, Dynatrace se posiciona como una "fuente de verdad" para las organizaciones. [8]

Entre las principales capacidades de Dynatrace se destacan:

- Supervisión de la pila completa (infraestructura, aplicaciones, y experiencia digital).
- Gestión y análisis de registros en tiempo real.
- Depuración de código en producción.
- Identificación y resolución de cuellos de botella en el rendimiento de aplicaciones.

Además, Dynatrace ofrece paneles personalizables que permiten visualizar las métricas de manera clara y acceder rápidamente a los datos relevantes para la supervisión de los entornos de trabajo.

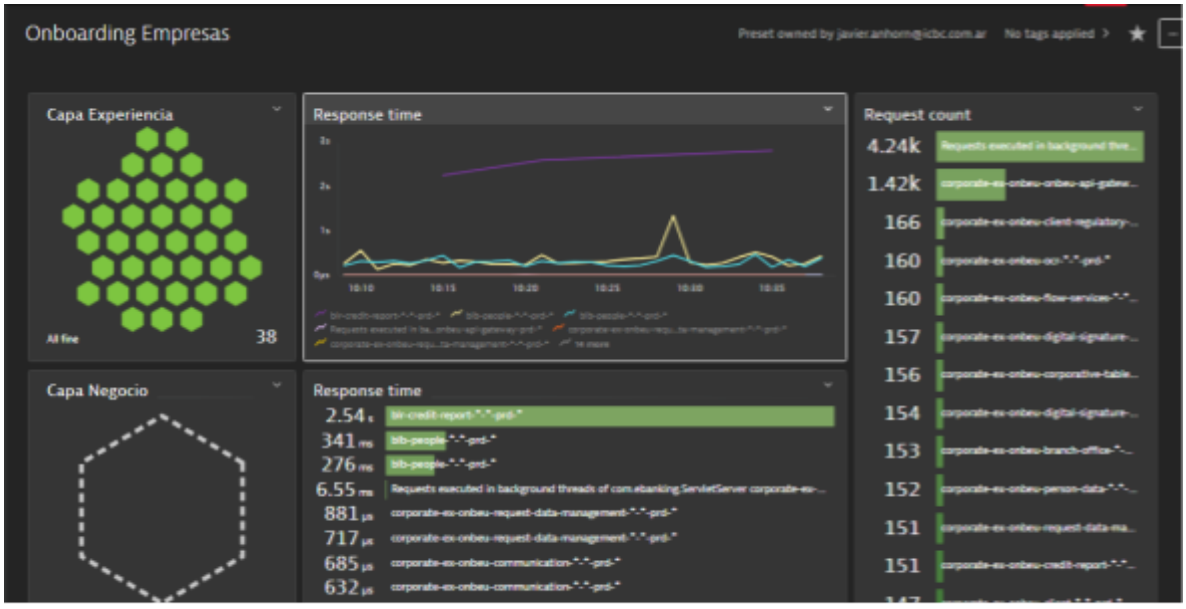

Figura 49: Panel en Dynatrace de Onboarding Empresas (Propia)

Integración del Real User Monitoring (RUM)

Dynatrace ofrece el monitoreo de la experiencia del usuario en tiempo real a través de su herramienta Real User Monitoring (RUM). Para implementarlo en aplicaciones frontend, es necesario agregar un fragmento de código JavaScript a las aplicaciones web. Este fragmento debe ser insertado en el encabezado (<head>) del archivo index.html.

Una vez integrado, se puede configurar opciones avanzadas en RUM, como excluir ciertas páginas o establecer reglas de privacidad, según lo acordado con el equipo de cumplimiento del ICBC. Esto garantiza que no se capturen datos sensibles de los usuarios.

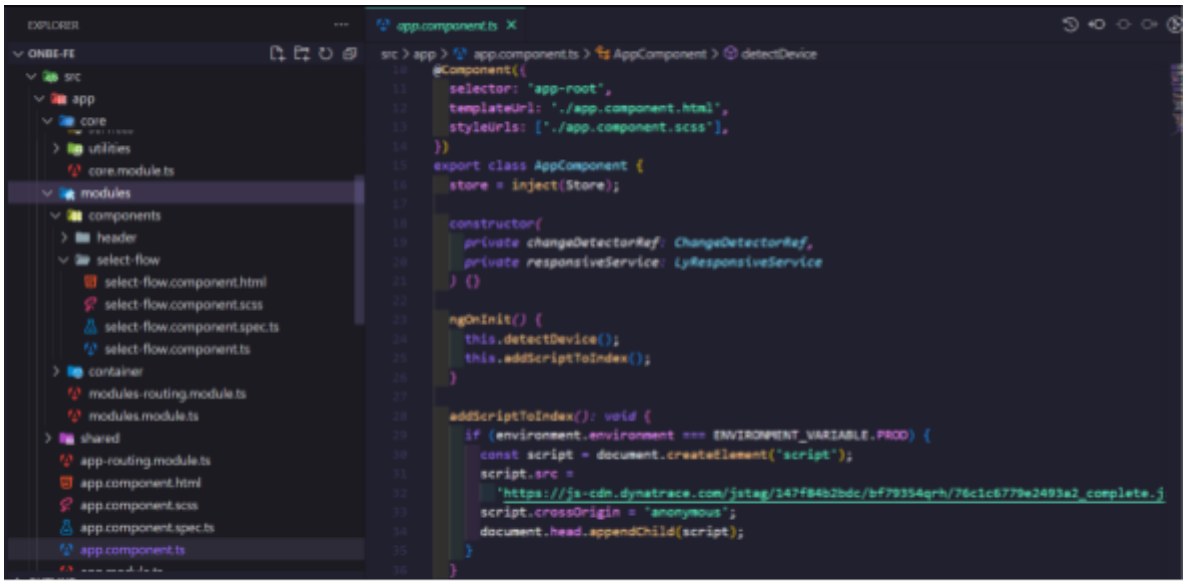

Figura 50: Integración de RUM en el index de aplicación frontend (Propia)

Validación de la Integración

Tras agregar el fragmento de código, es necesario validar la integración abriendo la aplicación y verificando que los datos del usuario real se están recopilando correctamente en la consola de Dynatrace.

Monitoreo y Análisis

Para analizar los datos recopilados por RUM, es posible acceder a la consola de Dynatrace y navegar a la sección correspondiente. Allí se puede obtener información detallada sobre el rendimiento de la aplicación, incluidos los tiempos de carga, errores y la experiencia general del usuario.

5. Conclusiones:

En conclusión, el desarrollo del sistema de onboarding ha permitido optimizar el proceso de apertura de cuentas y contratación de servicios financieros para empresas, logrando una mayor eficiencia en términos de tiempo y recursos. La implementación del onboarding contribuye no solo a la seguridad y conformidad del proceso, sino también a una experiencia de usuario más fluida desde el inicio de la relación con el ICBC. Este enfoque permite a las empresas comenzar a operar de manera más rápida y efectiva, apoyando así su desarrollo y competitividad.

Puedo concluir también, que todo lo aprendido en la Universidad Nacional del Noroeste de la Provincia de Buenos Aires, en la carrera de Ingeniería en informática, fue determinante y de gran importancia para la realización de la práctica profesional y desempeño como desarrollador Frontend, me brindó una rápida adaptación al aprendizaje de nuevos conocimientos y facilidad para aplicarlos en requerimientos para el producto, ya que es una práctica que se lleva a cabo durante toda la carrera.

Desde el rol de desarrollador frontend, la experiencia de aprendizaje con el framework Angular junto con la utilización de importantes librerías y el esfuerzo por aplicar prácticas de programación eficientes han sido fundamentales para entregar productos bien estructurados y funcionales. Al integrar las mejores prácticas de desarrollo, junto con herramientas de gestión de proyectos y control de versiones, se logró mejorar continuamente tanto el producto como el trabajo en equipo. De este modo, el proyecto no solo alcanzó sus objetivos iniciales, sino que también sentó una base sólida para futuros incrementos en el producto.

6. Bibliografía:

1. Agile Scrum methodology: What to know. (s/f). *Inflectra.com*. Recuperado el 20 de junio de 2024, de https://www.inflectra.com/Solutions/Methodologies/Scrum.aspx
2. Amazon.com. (s/f). Recuperado el 20 de junio de 2024, de https://aws.amazon.com/es/what-is/elk-stack/
3. Appmaster.io. (s/f). ¿Qué es Miro y cómo utilizarlo en su empresa? *Appmaster.io*. Recuperado el 20 de junio de 2024, de https://appmaster.io/es/blog/que-es-miro-y-como-utilizarlo-en-su-empresa
4. Bhaskar, S. (2022, diciembre 23). What is scrum methodology? & scrum project management. *NimbleWork, Inc.* https://www.nimblework.com/agile/scrum-methodology/
5. Cea, O. (2018, agosto 18). Programación reactiva en JavaScript. *Medium*. https://osmancea.medium.com/programaci%C3%B3n-reactiva-en-javascript-997478d45bfb
6. Despa, V. (2025). *Learn GitLab CI/CD from a GitLab Hero. Obtain valuable DevOps skills. Build pipelines and Deploy to AWS* [Curso en línea]. Udemy. https://icbc.udemy.com/course/gitlab-ci-pipelines-ci-cd-and-devops-for-beginners/learn/quiz/6625363#overview
7. Digital55. (s.f.). *Programación reactiva con RxJS*. Digital55. https://digital55.com/blog/programacion-reactiva-rxjs/
8. Gupta, T. (2021, agosto 4). Infrastructure monitoring and observability using Dynatrace — Part 1. *Engineered @ Publicis Sapient*. https://medium.com/engineered-publicis-sapient/infrastructure-monitoring-and-observability-using-dynatrace-part-1-e9e69d8be74d
9. Jira documentation. (s/f). *Atlassian.com*. Recuperado el 20 de junio de 2024, de https://confluence.atlassian.com/jira

10. Las 5 ceremonias Scrum: Claves para la gestión de procesos. (2017, diciembre 20). *Deloitte Spain*. https://www2.deloitte.com/es/es/pages/technology/articles/ceremonias-scrum.html
11. Martins, I. (2023, agosto 2). State management with NgRx in Angular - Igor Martins. *Medium*. https://medium.com/@igorm573/state-management-with-ngrx-in-angular-66ddc61cdf14
12. Motto, T. (s/f). RxJS: Observables, Observers and operators introduction.*Ultimatecourses.com*. Recuperado el 9 de noviembre de 2024, de https://ultimatecourses.com/blog/rxjs-observables-observers-operators
13. Nanarkar, A. (2020, marzo 5). Nexus repository manager. *Medium*. https://medium.com/@abhisheknanarkar/nexus-repository-manager-53fc29bfa592
14. Poyias, A. (2019, enero 7). *Design patterns: A quick guide to observer pattern*. Medium. https://medium.datadriveninvestor.com/design-patterns-a-quick-guide-to-observer-pattern-d0622145d6c2
15. Redux and the Flux architecture. (2019, octubre 31). *GeeksforGeeks*. https://www.geeksforgeeks.org/redux-and-the-flux-architecture/
16. RxJS. (s/f). *Rxjs.dev*. Recuperado el 9 de noviembre de 2024, de https://rxjs.dev/guide/operators
17. Sonatype Nexus repository. (s/f). *Onap.org*. Recuperado el 20 de junio de 2024, de https://nexus3.onap.org/
18. Staltz, R. (2015). *Introduction to Functional Programming with JavaScript* [gist]. GitHub. https://gist.github.com/staltz/868e7e9bc2a7b8c1f754
19. West, D. D. (s/f). Planificación de sprints. *Atlassian*. Recuperado el 20 de junio de 2024, de https://www.atlassian.com/es/agile/scrum/sprint-planning
20. What is CI/CD? (2022, enero 26). *GitLab*. https://about.gitlab.com/topics/ci-cd/
21. Working Software. (2023, septiembre 27). What is NgRx and why is it used in Angular? *Working Software*. https://www.workingsoftware.dev/what-is-ngrx-and-why-is-it-used-in-angular/

7. Anexos:

- Anexo A: Tablas

- Anexo B: Figuras

8. Agradecimientos:

En primer lugar, quiero expresar mi más sincero agradecimiento a la UNNOBA, que me brindó la oportunidad de crecer y aprender sin limitaciones. Fue en sus aulas, bajo la guía de sus docentes y junto a compañeros que compartieron mi misma pasión, donde encontré el camino que me permitió acercarme a la realización de mi sueño de convertirme en ingeniero. La universidad me abrió sus puertas y me ofreció cada rincón de sus edificios como un segundo hogar, un lugar lleno de recuerdos y anécdotas que marcaron mi vida de manera imborrable.

A mi familia, cuya presencia constante fue un pilar fundamental en mi vida académica. Cada uno, a su manera, me mostró que mi esfuerzo y dedicación por terminar la carrera los hacía inmensamente felices. Mi madre, que siempre me alentó a no rendirme y a seguir luchando hasta alcanzar el objetivo; mi tía, quien me transmitió su amor por las ciencias exactas y su pasión por el conocimiento; mi hermana, quien, con su fe inquebrantable en mí, se convirtió en mi mayor motor en los momentos más difíciles; y, por último, a mi padre, quien hizo posible que cada viaje a Junín fuera una experiencia significativa. Desde el primer curso de ingreso hasta cada paso que di hasta el final de la carrera, puedo concluir que cada kilómetro recorrido valió la pena gracias a su esfuerzo y apoyo constante.

Finalmente, quiero agradecer profundamente a mi pareja, quien ha sido mi compañera incondicional en la etapa final de este proceso. Cada vez que perdía la motivación por terminar de construir la práctica profesional supervisada, ella hacía todo su esfuerzo para ayudarme a reencontrarme con mi antigua ilusión de darle un cierre a esta etapa. Además, tomaba como propias esas ganas de continuar y las utilizaba para poder finalizar sus propios exámenes finales. Ha sido un trabajo en equipo muy agradable y no tengo dudas de que, sin su presencia y la tranquilidad que siempre logra transmitirme, nada de esto hubiese sucedido de la forma en que lo hizo.