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

IMPULSO DEL ÁREA DE DESARROLLO DE APLICACIONES MÓVILES EN PLATANUS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN

MATÍAS MONTAGNA CARBONELL

PROFESORA GUÍA:
MARÍA CECILIA BASTARRICA PIÑEYRO

MIEMBROS DE LA COMISIÓN:
ANDRÉS MUÑOZ ORDENES
ÉRIC TANTER

Este trabajo ha sido parcialmente financiado por:
Platanus

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL EN COMPUTACIÓN
POR: MATÍAS MONTAGNA CARBONELL
FECHA: 2023
PROF. GUÍA: MARÍA CECILIA BASTARRICA PIÑEYRO

IMPULSO DEL ÁREA DE DESARROLLO DE APLICACIONES MÓVILES EN PLATANUS

Platanus es una empresa que desarrolla productos de software a clientes externos. Para mejorar tanto en calidad como en velocidad de la entrega de sus productos, la empresa ha estandarizado varios de sus procesos tecnológicos, y generado documentación sobre el uso de estas. La mayoría de los procesos estandarizados se refieren al desarrollo de aplicaciones web, mientras que el desarrollo de aplicaciones móviles posee varios aspectos que no son consistentes a lo largo de los diferentes proyectos que ha abordado la empresa. Al no contar con un estándar tecnológico, se incrementa innecesariamente el tiempo que se requiere para llegar al producto mínimo viable.

El presente trabajo de título busca impulsar el área de desarrollo de aplicaciones móviles en Platanus. Para esto, se plantean tres objetivos. El primer objetivo es la generación de estándares, documentación y guías de estilo sobre las tecnologías involucradas. El segundo es la reanudación del desarrollo del generador de proyectos móviles de la empresa, que se encontraba abandonado en un estado no funcional. El tercero es la reestructuración de un proyecto móvil que lleva varios años en Platanus, para dejarlo acorde al nuevo estándar que se plantea.

Con respecto al primer objetivo, se generaron guías de estilo y se estandarizaron ciertas tecnologías para aumentar la consistencia en el área. También se crearon guías que ayudan a utilizar las tecnologías. Sobre el segundo objetivo, se actualizó el generador de proyectos móviles de la empresas, devolviéndolo a un estado funcional y agregándole nuevas funcionalidades (algunas relacionadas con estandarizaciones del primer objetivo). Para el tercer objetivo se efectuó una reingeniería sobre el proyecto, agregando el lenguaje TypeScript al proyecto, modificando la estructura de directorios del proyecto y añadiendo aliases de importación para mejorar la legibilidad de las importaciones de los archivos.

Para evaluar el primer objetivo se contrastaron los resultados obtenidos por encuestas realizadas al inicio y término del proceso a desarrolladores de la empresa.Se obtuvo que el entusiasmo por querer participar en proyectos móviles, y la percepción sobre la cantidad de material disponible mejoraró notablemente. Para el segundo objetivo se inició exitosamente un nuevo proyecto móvil usando el generador, demostrando que esté volvió a un estado funcional. Para el último objetivo se compararon los puntos que cumplía (o no) el proyecto previo a su reingeniería y luego cuántos cumplía después de esta. El proyecto pasó de no cumplir ninguno de los puntos, a cumplir más de la mitad de estos.

Se concluye del trabajo realizado que la estandarización de procesos y tecnologías en organizaciones de software permite disminuir el tiempo que se debe invertir para poder llegar a productos mínimos viables, sin sacrificar la calidad de los mismos. De esta forma las organizaciones pueden crecer, sin perder su sello de calidad en el mercado.

Agradecimientos

A mi familia, por siempre creer en mí y apoyarme en las incontables trasnochadas que tuvo esta aventura.

A mis amigos de la infancia, por siempre estar ahí.

A los amigos que conocí en la universidad, que tomaron este proceso en donde se aprenden muchas cosas técnicas y profesionales, y lo transformaron en una etapa con años intensos de vida, de aciertos, de errores y de aprendizajes que me llevo para todo lo que siga adelante.

A mi profesora guía, por apoyarme con su visión y su experiencia.

A Platanus, por confiar en mí para poder desarrollar este trabajo.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

A la hora de desarrollar productos digitales, un problema recurrente es el de balancear adecuadamente la calidad del trabajo de lo que se contruye, junto a la rapidez con la que se debe entregar el producto.

Inclinarse mucho en en la primera opción lleva a largos tiempos de espera para poder probar el producto, lo que es fatal en un ambiente que se mueve muy rápido por su naturaleza. Además desde el punto de vista tecnológico, invertir mucho tiempo en la calidad de un proyecto puede hacer que la tecnología inicial que se decidió usar se vuelva sub óptima u obsoleta antes de terminar el proyecto.

Por otro lado, enfocarse en entregar lo más rápido posible el producto desmereciendo la calidad del trabajo a realizar, puede desembocar en lanzamientos de productos que se encuentran incompletos, o que bien no son escalables y si se quiere a futuro agregar una nueva funcionalidad, debe rehacerse gran parte de este. Sin ir más lejos, un ejemplo icónico de producto incompleto es la salida al mercado del juego Cyberpunk 2077. Este producto salió al mercado el 10 de diciembre del año 2020, y era el juego más esperado de ese año. Si bien poseía la mayoría de sus funcionalidades terminadas, estaba al debe en lo que respecta a rendimiento en consolas o computadores de baja gama, y poseía varios errores que destruían la experiencia de usuario. El juego tuvo que ser retirado de las tiendas digitales de Sony y se ofrecieron reembolsos a las personas que ya lo habían comprado. El lanzamiento del juego fue considerado un rotundo fracaso [1].

Obtener un balance en la relación velocidad/calidad es de especial importancia para las empresas que ofrecen como servicio el desarrollo de productos digitales, ya que el resultado final debe ser de calidad y extensible para poder ganar reputación con sus clientes y en el mercado. Asimismo, el tiempo invertido en entregar el producto no puede ser muy extenso, porque eso también va en desmedro del nombre de la empresa y es un aspecto importante a considerar por los clientes que se acercan a estas empresas en busca de sus servicios.

Platanus es una empresa de este tipo, y posee mucha experiencia a la hora de abordar esta problemática. Con el paso de los años, las metodologías y las tecnologías que usan para desarrollar aplicaciones web ha sido refinada y estandarizada. Por ejemplo, la empresa desarrolló Potassium, un generador de proyectos web que automatiza varias configuraciones de tecnologías que son usadas en todos los proyectos web que se abordan en la empresa. Por el

lado del frontend, se ha escrito una guía de estilos interna para mantener consistencia sobre lo que se hace y cómo se hace. De esta manera, se acelera el desarrollo de los productos que se abordan, sin sacrificar en la calidad final de los mismos.

Además de aplicaciones web, Platanus también desarrolla aplicaciones móviles, aunque los proyectos móviles que se han abordado son menores en cantidad en relación a los proyectos web. Esta asimetría se ve plasmada en que pocos desarrolladores de la empresa se encuentran familiarizados con las tecnologías que requiere este tipo de desarrollo. A inicios de febrero de 2023 se hizo una encuesta para hacer un diagnóstico del estado actual del desarrollo móvil en Platanus de donde se obtuvieron las respuestas de trece de los diecisiete desarrolladores de la empresa. De la encuesta se obtuvo que el 38,5 % de los desarrolladores encuestados no se sentían preparados para abordar un proyecto móvil, y sólo el 7,5 % de los encuestados decía sentirse muy preparado.

A inicios del año 2023, habían diez proyectos activos que la empresa se encontraba desarrollando o manteniendo. De esa cantidad, solamente uno era un proyecto móvil, al cuál se referirá cómo TaskTracker.

Al haber menos desarrolladores involucrados en esta área, todavía no existe en Platanus una metodología de desarrollo que sea uniforme para todos los proyectos móviles. Esto presenta un gran inconveniente a la hora de buscar el balance óptimo entre calidad y rapidez, y va en desmedro de la empresa.

1.1. Situación Inicial

Platanus usa una mezcla entre la metodología Scrum y la metodología Kanban para trabajar con clientes.

Cuando llega un nuevo cliente, se conforma un nuevo equipo que aborda el proyecto. Este equipo se compone de uno o dos desarrolladores (dependiendo de la magnitud y alcance del proyecto) y un "Scrum Master", el cual está encargado de llevar las reuniones con el cliente y velar por el buen funcionamiento del equipo.

1.1.1. Encuesta de Diagnóstico

Como se mencionó anteriormente, se elaboró una encuesta relacionada con el desarrollo móvil la cual fue respondida por trece desarrolladores de un total de 17 dentro de la empresa. Los resultados arrojados fueron los siguientes:

1. ¿Tienes experiencia en el desarrollo móvil?
    - Si: 61,5 %
    - No: 38,5 %
2. ¿Te gustaría desarrollar móvil en el corto/mediano plazo?
    - Si: 69,2 %

- No: 30,8 %

3. ¿A cuál(es) de estas tecnologías/procesos te has enfrentado antes?

   - Expo: 19,4 %
   - React: 36,1 %
   - React Native: 22,2 %
   - Redux/easy-peasy: 11,1 %
   - Subir app a las tiendas de aplicaciones: 11,1 %

4. De las cosas mencionadas, ¿Cuáles dirías que son las que menos conoces?

   - Expo: 24,24 %
   - React: 6,06 %
   - React Native: 12,12 %
   - Redux/easy-peasy: 33,33 %
   - Subir app a las tiendas de aplicaciones: 24,24 %

5. De las cosas mencionadas, ¿Cuáles dirías que son las más cruciales para sentirte preparado para abordar un proyecto móvil?

   - Expo: 28,13 %
   - React: 18,75 %
   - React Native: 37,5 %
   - Redux/easy-peasy: 12,5 %
   - Subir app a las tiendas de aplicaciones: 3,13 %

6. ¿Cuál es tu percepción sobre el material disponible en Platanus (la guía, presentaciones, notion) para aprender las tecnologías involucradas?

   - Hay mucho material (todas o casi todas las tecnologías están cubiertas): 0 %
   - Hay algo de material (porcentaje no menor de tecnologías cubiertas): 23,08 %
   - Hay poco material (ninguna o casi ninguna de las tecnologías están cubiertas): 76,92 %

7. ¿Qué tan preparado te sentirías si la próxima semana tuvieras que entrar a un proyecto de desarrollo móvil?

   - Muy preparado (estoy familiarizado con varias cosas): 7,6 %
   - Algo preparado (conozco algunas cosas pero me falta harto todavía): 53,8 %
   - Nada preparado (si me asignan a un proyecto de desarrollo móvil me estresaría bastante): 38,5 %

8. Si se realiza una presentación de desarollo móvil, ¿De qué te gustaría que se trate? ¿Cuál encontrarías de mayor utilidad para ti?

   - React: hooks básicos, sintaxis básica (38,5 %)

- React Native: Componentes Nativos más comunes, React Navigation, diferencias con React Web (15,4 %)
- Expo: ¿Qué nos da expo? ¿Qué limitaciones nos impone? Librerías de expo útiles, compilar/subir builds a expo.dev (38,5 %)
- Publicar una App en la App Store(iOS) y Play Store (Android): 7,6 %

Se mencionó que existe Potassium como generador de proyectos web, que ayuda a acelerar la entrega de los productos sin sacrificar la calidad de estos. También existe un generador de proyectos móviles dentro de la empresa, llamado Cavendish. Este se inspira en Potassium y busca automatizar la configuración de tecnologías que son utilizadas en este tipo de proyectos.

En Cavendish se ve reflejada la asimetría de cantidad personas que se encuentran capacitadas en desarrollo móvil en contraste con desarrollo web: en el repositorio de Github de Potassium se puede ver que treinta y ocho personas han contribuido al proyecto, mientras que en el repositorio de Cavendish hay solamente dos contribuidores.

1.2. Problema

La empresa carece de una metodología estandarizada de desarrollo en el área de desarrollo móvil, y esto presenta inconvenientes.

Los proyectos de software poseen un alto grado de incertidumbre y riesgo. Una de las principales formas de disminuir el riesgo es reduciendo lo más posible el tiempo que se debe invertir en llegar al producto mínimo viable (o MVP). Al no tener un estándar tecnológico, los pasos que deberían ser idénticos para cualquier proyecto móvil toman más tiempo del que deberían (por ejemplo, conectar el entorno de la aplicación a entornos de producción o staging). Esto incrementa innecesariamente el tiempo que se requiere para llegar al MVP, y se vuelve una desventaja competitiva frente a otras empresas que ofrezcan el mismo servicio.

Pensando en un MVP veloz y de calidad, el generador de proyectos móviles, Cavendish, fue construido para abordar el problema de poder iniciar un proyecto móvil lo más rápido posible, teniendo en cuenta los estándares de la empresa. Sin embargo, a inicios del año 2023 el proyecto se encontraba en un estado no funcional, sin mantención alguna.

Por otro lado, al no existir un conducto regular en donde conversar sobre estándares de desarrollo móvil, o de metodologías de trabajo en los equipos, se genera una fuga de conocimiento en la empresa. Los integrantes de la empresa involucrados en estos proyectos pueden acumular experiencia y crear formas de resolver los problemas que se presentan, pero esas vivencias no quedan plasmadas en ningún lugar (ya sea físico o digital) por lo que al migrar de proyecto o irse de la empresa, se pierde su experiencia.

Anteriormente se mencionó que a principios del año 2023 el único proyecto móvil que se estaba desarrollando se llamaba TaskTracker. El punto anterior está presente en el proyecto. Desde los inicios de este proyecto en la empresa (2018) hasta finales del año 2022, la modalidad abordada en este proyecto fue de equipos de un desarrollador y un Scrum Master.

Además, debido a la antiguedad del proyecto, los equipos que abordan el proyecto (tanto desarrollador como Scrum Master) han ido rotando. El proyecto ha cambiado de desarrollador 4 veces en aproximadamente 4 años. Esto no sería un problema en proyectos web debido a la cantidad de integrantes en la empresa con el conocimiento necesario para abordar un proyecto que ya se encuentra iniciado, con su propia lógica. Sin embargo, cuando esta situación se combina con la baja disponibilidad de desarrolladores capacitados en el área y la falta de estándares sobre esta, si se convierte en un problema.

Si desarrolladores inexpertos quieren sumarse a algún proyecto móvil poseerán una barrera de entrada que podría ser menor si existiera un repositorio con las respuestas a los obstáculos a la que sus colegas ya se enfrentaron. Pero esto no existe actualmente y deben resolver problemas que otro integrante ya puede haber encontrado una buena forma de abordarlo y resolver. Este punto también aumenta los tiempos de desarrollo en promedio, ya que se enfrenta un mismo problema más de una vez.

Además, en los casos en que desarrolladores con experiencia en desarrollo móvil deciden (o tienen que) cambiarse de un proyecto móvil a otro de la misma índole, sus experiencias acumuladas en un proyecto pudiera no servirle en el nuevo proyecto ya que los estándares ocupados pueden ser radicalmente distintos.

1.3. Objetivos

1.3.1. Objetivo General

El objetivo principal de la presente memoria es la mejoría del área de desarrollo de aplicaciones móviles en Platanus, mejorando las herramientas y materiales a disposición de desarrolladores que deben involucrarse en proyectos de esta índole, así como también mejorando la mantenibilidad, escalabilidad y uniformidad del código de estos proyectos.

1.3.2. Objetivos Específicos

Para cumplir el objetivo general propuesto, se plantean los siguientes objetivos específicos:

1. Estandarización y documentación de tecnologías y procesos existentes en la empresa para desarrollo de aplicaciones móviles
2. Actualización del generador de proyectos móviles de la empresa
3. Reestructuración de proyecto TaskTracker que aborde deuda técnica, aplicando los nuevos estándares móviles

Capítulo 2

Marco Teórico

2.1. Transmisión de Conocimiento en Organizaciones

El trabajo de esta memoria consiste en mejorar los procesos y el conocimiento sobre un área en específico dentro de una organización; desarrollo móvil en Platanus. Debido a eso, se vuelve indispensable hablar de transmisión de conocimiento, ya que es a través de esto que se logra un aumento sostenible en el tiempo.

Inicialmente se pensaba en la transmisión de conocimiento cómo un proceso unidireccional, pero este pensamiento fallaba en capturar lo que ocurría en organizaciones en donde existen varias personas interactuando [2].

Una forma conocida de transferencia de conocimiento en equipos ágiles es hacer *pair programming* en donde dos desarrolladores trabajan juntos. Sin embargo, la viabilidad de esta técnica no es alta en Platanus debido a que la cantidad de personas capacitadas para poder transmitir conocimiento de desarollo móvil es muy baja: en la encuesta que se mencionó en la sección 1.1.1 solamente el 7,6 % de los desarrolladores encuestados respondieron que se encontraban "muy preparados" en esta área. Entonces, hacer pair programming generaría una sobrecarga importante en las personas capacitadas.

Otra método que se utiliza para la transmisión de conocimiento es usar un sistema tipo Wiki, el cual consiste en un conjunto de páginas web que son creadas y editadas por un grupo de usuarios, que buscan proveer información y conocimiento sobre un tema en específico. Este sistema genera transmisión de conocimiento a través de la exposición de información valiosa (inicialmente por los integrantes más experimentados de la organización), y la colaboración que ocurre entre los integrantes, al ir constantemente actualizando la información disponible [3]. Esta manera de transferir conocimiento ya existe en Platanus, y se le denomina "La Guía" (en la sección 2.7 se entra en detalle sobre ésta).

2.2. Desarrollo de Aplicaciones Móviles

El desarrollo de aplicaciones móviles es el proceso mediante el cual se desarrolla software que se ejecuta principalmente en un dispositivo digital portátil, como un teléfono móvil o una tablet [4].

Cuando se desarrolla una aplicación para que sea usada en el teléfono por un usuario, se deben considerar ciertos aspectos que no están presentes en el desarrollo web. El almacenamiento disponible que tienen los usuarios se vuelve un factor relevante, por lo que se debe considerar eso a la hora de ver cómo la aplicación guarda los datos necesarios para funcionar. También se debe tener en cuenta la variación en las dimensiones de las pantallas, que pueden ir desde dispositivos pequeños de alrededor de 4,7 pulgadas hasta dimensiones de 12,9 pulgadas en los dispositivos de mayor tamaño, con son las tablet [5]. Finalmente, un factor fundamental a tener en cuenta es el sistema operativo de los teléfonos.

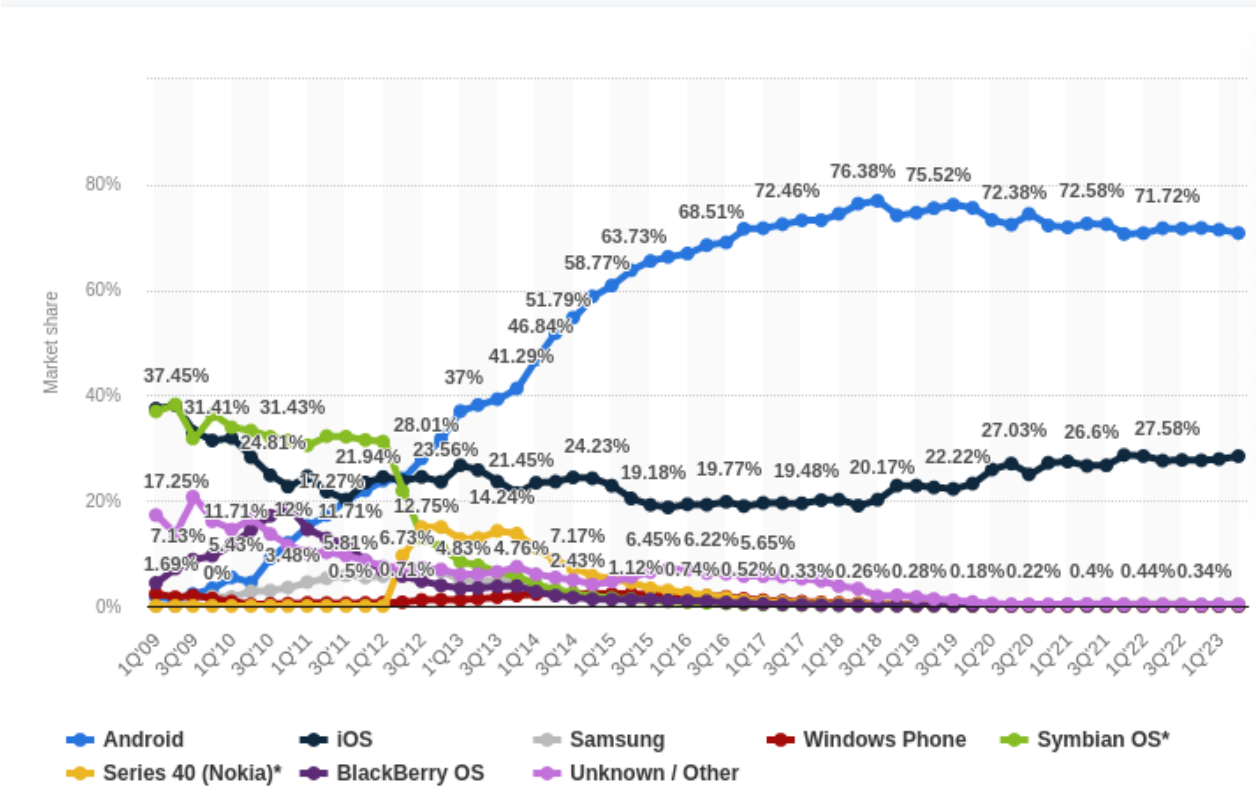

Figura 2.1: Participación de mercado de sistemas operativos en dispositivos móviles desde 2009 hasta segundo cuarto (Q2) de 2023. Fuente: Statista [6].

Los dos sistemas operativos más usados son Android e iOS, aunque también existen otros menos conocidos como Symbian OS y Windows Phone. En la figura 2.1 se puede ver como ha evolucionado la participación de mercado (market share en inglés) de estos sistemas operativos desde 2009 hasta 2023. Android e iOS poseen en conjunto el 99,2 % del mercado, con un 70,8 % y 28,4 % respectivamente. Debido a la abrumadora dominancia que poseen, es que solamente se referirá a estos dos a la hora de hablar del espectro de dispositivos móviles desde este punto en adelante.

Debido a la diferencia fundamental entre Android y iOS, es que existen dos formas de abordar el desarrollo móvil:

- Desarrollo nativo
- Desarrollo multiplataforma

2.2.1. Desarrollo Móvil Nativo

En esta metodología de desarrollo se construye la aplicación móvil usando el lenguaje de programación que es compatible con el sistema operativo del teléfono. En este enfoque *nativo*, se emplean las herramientas proporcionadas por el propio sistema para construir la aplicación.

Android posee una máquina virtual llamada Dalvik [7]. Debido a esto, existen dos opciones a la hora de desarrollar aplicaciones nativas para Android: Java y Kotlin.

Kotlin es un lenguaje de programación relativamente nuevo, que fue lanzado en 2016. Este puede ser usado para desarrollar en sistemas Android ya que se puede correr en una JVM (Java Virtual Machine). En un estudio publicado el año 2018 que comparaba a Java con Kotlin en el ambiente de desarrollo de aplicaciones móviles, se concluyó que Kotlin era el lenguaje preferido desde el punto de vista de desarrolladores con experiencia [8]. Se esgrimieron, entre otras, las siguientes razones:

- Código escrito en Kotlin posee mayor grado de seguridad. Kotlin está diseñado para evitar errores comunes en la programación. Por ejemplo, posee una característica llamada *Null Safety* que resuelve en la mayoría de los casos uno de los errores más comunes de Java, la excepción *NullPointerException*
- Kotlin es más conciso que Java. Permite resolver los mismos problemas con menos líneas de código, lo que mejora la lectura y mantenibilidad del mismo, y aumenta la productividad de desarrolladores.

Al ser un lenguaje reciente, la gran mayoría de las aplicaciones móviles nativas de Android usan Java. Aun así, la proyección de este lenguaje es prometedora, con Google anunciando en la conferencia Google I/O 2019 que este es ahora el lenguaje preferido para abordar el desarrollo de aplicaciones en Android [9].

Por el lado de los teléfonos con sistemas iOS, también existen dos alternativas de lenguajes de programación para programar nativamente en la plataforma, los cuales son Objective-C y Swift. Estos dos lenguajes comparten una relación similar a la de Java con Kotlin: Objective-C fue el lenguaje inicial y único sobre el cual programar aplicaciones nativas en iOS. Sin embargo, el 2014 aparece Swift con la intención de ofrecer una alternativa más simple y novedosa a nuevos desarrolladores [10]. Ambos Swift y Objective-C usan el compilador LLVM (Low Level Virtual Machine) [10].

Al desarrollar aplicaciones nativas, se puede acceder directamente al sistema operativo y por ende existe un alto grado de customización y de acceso a todas las funcionalidades que proveen los sistemas operativos. Otro punto fuerte de este tipo de aplicaciones es su rendimiento en términos del espacio en el celular que ocupan, y la rapidez a la que se ejecutan.

A pesar de lo anterior, un gran problema que posee este tipo de desarrollo es que al ser nativo está estrechamente ligado a la plataforma sobre la que está construida. Una aplicación nativa de Android construida en Kotlin o Java no puede ser ejecutada en un dispositivo iOS y viceversa. Este tipo de desarrollo es costoso: si se quiere acceder a todo el mercado de usuarios se debe tener un equipo de desarrolladores construyendo la aplicación para Android y otro equipo construyendo la aplicación para iOS. Si bien hay cierto conocimiento y tecnologías

que pueden compartirse (como un servidor común que maneje el almacenamiento y procesamiento de datos, o el diseño y experiencia de usuario), se deben construir dos aplicaciones móviles en lenguajes diferentes.

2.2.2. Desarrollo Móvil Multiplataforma

Como su nombre lo sugiere, el enfoque de desarrollo móvil multiplataforma busca abarcar las dos plataformas existentes en el mercado (Android e iOS) construyendo una sola aplicación que sea compatible en ambas plataformas. Para lograr esto, se abstrae de las funcionalidades nativas que estos sistemas proveen, agregando capas que se encargan de traducir el código de la aplicación a código nativo de la plataforma que corresponde. Esta abstracción no es gratis, y por ende el rendimiento de este tipo de aplicaciones es inferior al de las nativas.

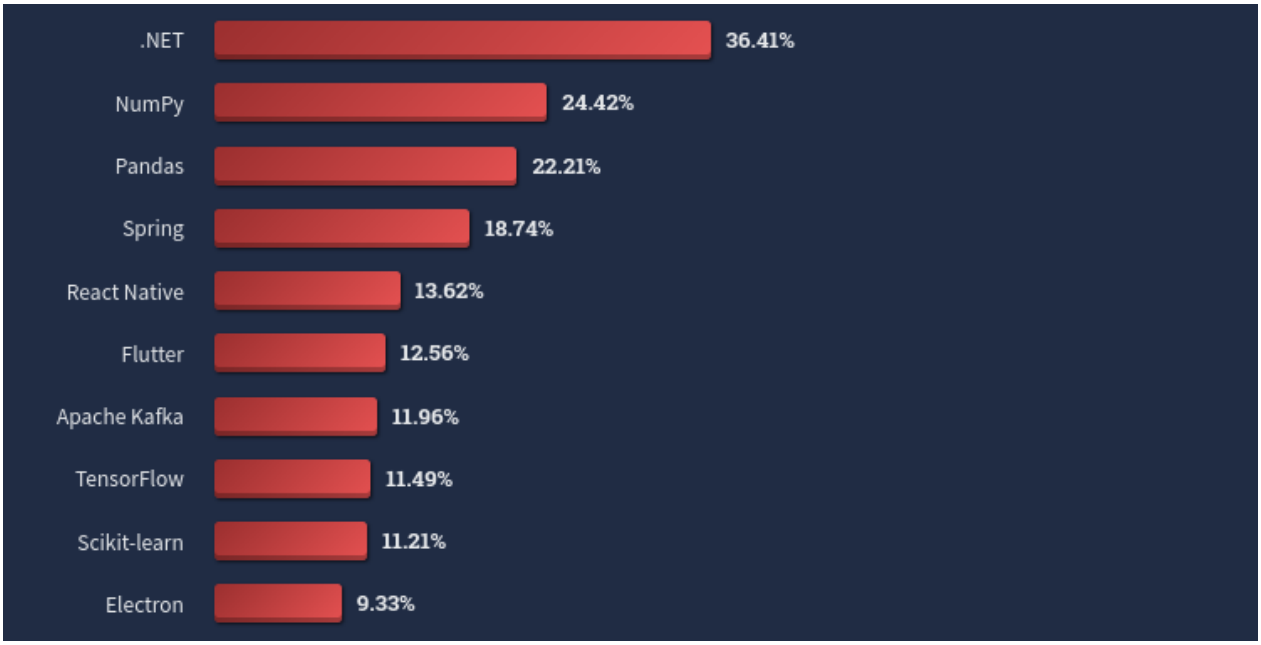

Figura 2.2: Top 10 tecnologías no web preferidas. Developer Survey 2022 de Stack Overflow

Los dos frameworks más populares para desarrollo móvil multiplataforma son Flutter y React Native. Ambos poseen el respaldo de grandes compañias, con el primero siendo respaldado por Google y el segundo por Facebook. En la "Developer Survey" de Stack Overflow del año 2022, en la sección de "Otras técnologías o librerías favoritas" (es decir, no web), React Native obtuvo el 13.62 % de los votos y Flutter obtuvo el 12.56 % en las votaciones que sólo consideran a desarrolladores profesionales de los votos [11]. En la figura 2.2 se pueden ver los primeros 10 resultados de está categoría. Además, en la figura 2.3 se puede ver cómo ha evolucionado la popularidad de estas tecnologías en los últimos años.

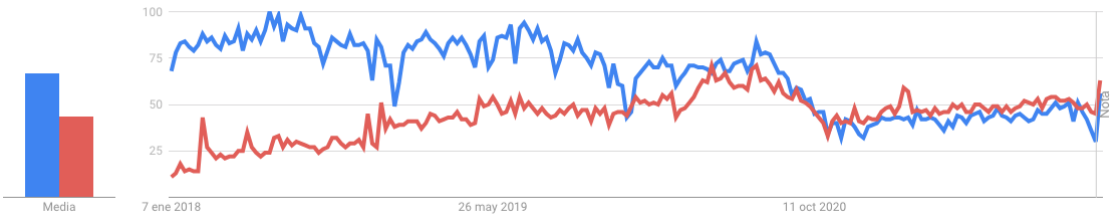

Figura 2.3: Popularidad de React Native (azul) y Flutter (rojo) en Estados Unidos desde 2018 hasta 2023. Fuente: Google Trends [12].

Flutter es un framework que se basa en el lenguaje de programación Dart. Este no es un lenguaje muy común y eso se ve reflejado en que obtiene el décimo sexto lugar en la categoría de lenguajes de programación favoritos, en la encuesta ya mencionada de Stack Overflow, con un 9.16 % de los votos. Por otro lado, React Native es un framework basado en el conocido lenguaje Javascript, que obtuvo el primer lugar en la misma categoría con un 63.36 % de los votos. En la figura 2.4 se pueden ver los primeros 20 lenguajes más populares en esta encuesta.

El mayor beneficio de este tipo de desarrollo es poder acceder a la totalidad del mercado de usuarios Android e iOS construyendo una sola aplicación. Es por esto que este método es el preferido a la hora de empezar un producto en una startup ya que abarata considerablemente los costos y disminuye el riesgo.

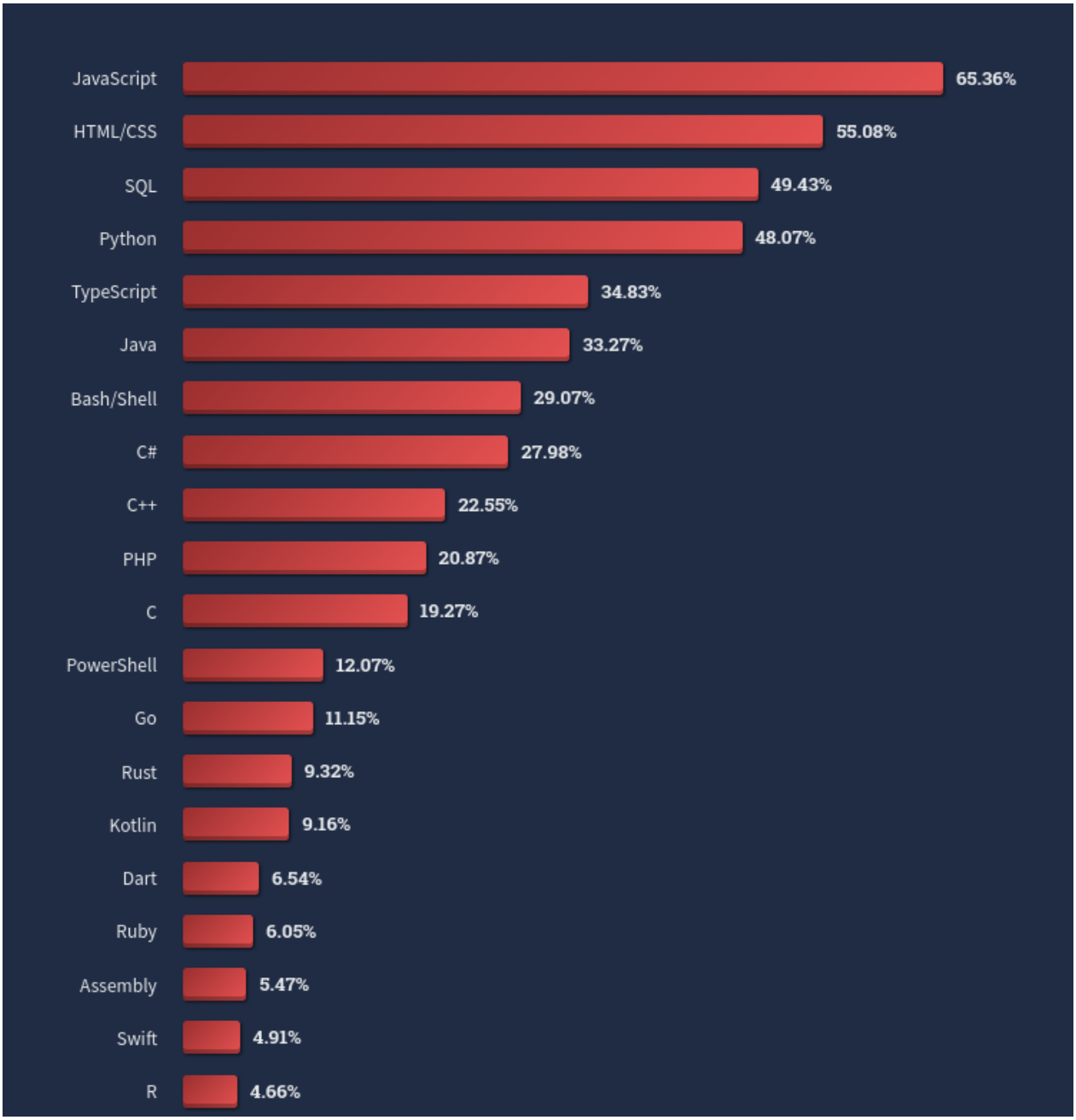

Figura 2.4: Top 20 lenguajes más populares. Developer Survey 2022 de Stack Overflow

En Platanus se sigue este enfoque de desarrollo móvil multiplataforma por las razones esgrimidas en el parrafo anterior. A la hora de elegir entre Flutter y React Native, existe una ventaja comparativa en usar React Native debido a que está construido sobre Javascript, lenguaje que tambien se usa en los proyectos de desarrollo web de la empresa. Debido a todos estos puntos, en Platanus a la hora de desarrollar aplicaciones móviles se opta por React Native como la tecnología a usar.

2.3. React

Antes de comenzar a detallar el framework móvil multiplaforma que usa Platanus en sus proyectos (React Native), se debe indagar en el framework original que revolucionó la manera en que hoy en día se desarrollan páginas webs y sobre el cuál React Native está fuertemente basado: React.

React, también conocido como React.js o ReactJS, es una biblioteca de JavaScript de código abierto para el desarrollo de interfaces de usuario (UI en inglés) en el lado del cliente web. Es mantenido por Facebook y una comunidad de desarrolladores individuales y empresas [13]. El código que compone esta librería se encuentra en un repositorio de Github, en el link https://github.com/facebook/react.

La plataforma Github posee una opción de permitir a usuarios "estrellar" un repositorio como una forma de demostrar interés por estos, y poder seguir sus avances. El repositorio de React posee más de doscientos nueve mil estrellas en esta misma plataforma, convirtiéndolo en el décimo repositorio con más estrellas de toda la plataforma a la fecha de agosto de 2023 [14].

En su lanzamiento introdujo, entre otras cosas, el concepto de JSX, una extensión de sintaxis de Javascript, que puede considerarse como una combinación de JavaScript y XML, que se asemeja de cerca a HTML. Con JSX, se puede escribir como el código 2.1, lo que abre la puerta a combinar lógica de Javascript de manera muy estrecha para modificar el html. Esto aumenta la customización del código y la capacidad de manejar diferentes situaciones de la UI (ver 2.2) [15].

Código 2.1: Declaración de un elemento usando JSX.

```
1 const element = <h1>Hello, world!</h1>;
```

Código 2.2: Elemento JSX con lógica de Javascript incorporada.

```
1 const name = 'Matías Montagna';
2 const text = name.length > 20 ? 'Your name is very long!' : '';
3 const element = <h1>Hello, {name}!{text}</h1>;
```

La introducción de esta sintaxis no es algo netamente estético. En el núcleo de cualquier sitio web básico se encuentran los documentos HTML. Los navegadores web leen estos documentos y los muestran en un computador, tableta o teléfono como páginas web. Durante este proceso, los navegadores crean algo llamado *Document Object Model (DOM)*, un árbol representativo de cómo se organiza la página web. Si se usa JSX para manipular el DOM, React crea un DOM virtual para revisar qué partes del sitio web deben cambiar, para luego cambiar solamente esas partes en vez de renderizar de nuevo la totalidad de la página web [13].

Sobre el concepto de JSX se construye el concepto de componente. Un componente es una función de Javascript que retorna una agrupación de elementos JSX (ver 2.3) [16].

Código 2.3: Componente que representa una Navbar, en React.

```
1
2  const Navbar = () => {
3    return (
4      <nav>
5        <ul>
6          <li><a href="/">Home</a></li>
7          <li><a href="/about">About</a></li>
8          <li><a href="/services">Services</a></li>
9          <li><a href="/contact">Contact</a></li>
10       </ul>
11     </nav>
12   );
13 };
14
15 export default Navbar;
```

React apunta a construir UI's descomponiéndolas en varios componentes pequeños y reutilizables. Por ejemplo, en una aplicación en que el usuario puede comprar productos y ya posee varios items en su carrito de compra, el código de React que represente a su carrito de compra podría ser el que aparece en 2.4. Podemos ver que el componente ShoppingCart está constantemente reutilizando el componente ProductDetail para construir la lista de productos que verá el usuario en el carrito de compras.

Código 2.4: Carrito de compras construido en React (simplificado).

```
1  import ProductDetail from 'components/Product';
2
3  const ShoppingCart = (data) => {
4    const productsInShoppingCart = data.productsInShoppingCart;
5
6    return (
7      <div>
8        {productsInShoppingCart.map(product => (
9          <ProductDetail id={product.id} name={product.name} price={product.price} />
10       ))}
11     </div>
12   );
13 }
14
15 export default ShoppingCart;
```

Al ver el código en 2.4 se puede apreciar otra característica de React que es de gran relevancia, y es que al declarar el componente *ProductDetail* se puede ver que tiene los campos *id*, *name* y *price*. Estos campos se conocen como *props*. Si un componente es una función que retorna elementos JSX, los props (abreviación de *properties* en inglés) son los argumentos de esa función [16]. Esto le agrega variabilidad de customización a los componentes y a la aplicación en general, haciéndola flexible a diferentes datos.

2.4. React Native

React Native fue introducido el 2015 por Facebook con la premisa "Aprende una vez, escribe en cualquier lugar" (Learn Once, Write Anywhere) haciendo análogía a que desde ese momento se iba a poder desarrollar en React aplicaciones para plataformas web, Android e iOS, pudiendo en la práctica abarcar todo el mercado de usuarios con ese lenguaje.

El framework permite desarrollar aplicaciones móviles multiplaforma usando Javascript, JSX y React como API para comunicarse con el sistema nativo de cada plataforma móvil. Esto lo hace a través de una interfaz denominada "Bridge" (puente) que une la parte nativa del teléfono con el código Javascript escrito. De esta manera, eventos nativos del teléfono cómo presionar la pantalla en cierto punto o deslizar el dedo a través de esta, son serializados y enviados a través de "Bridge" en formato JSON [17]. Los eventos serializados son recibidos por el código Javascript, que actualiza la UI según corresponda. En la figura 2.5 puede verse un diagrama simplificado de cómo funciona esta relación.

Figura 2.5: Diagrama simplificado de cómo funciona React Native. Se ve la relación entre entre React, Javascript, "Bridge" y código nativo.

2.5. Redux

Se mencionó en 2.3 que los componentes de React tienen props. Las props son muy útiles, pero poseen una restricción grande en el sentido de que necesitan ser transportadas desde el componente padre hacia el componente hijo. ¿Que ocurre cuando sería conveniente que todos los componentes de la aplicación tengan cierta prop a su alcance? Que todos los componentes creados en la aplicación deban tener esa prop suena tedioso, restrictivo y una fuente segura de bugs, ya que basta que un componente maneje mal el valor de la prop para que toda la cadena de transmisión de la prop deje de funcionar de la manera esperada.

Son varias las aplicaciones que actualmente poseen el "modo oscuro", que al ser encendido por el usuario, hace que toda la aplicación se oscurezca. En una aplicación de React, para lograr esto se necesitaría que todos (o la mayoría) de los componentes de la aplicación tuvieran una prop "darkMode"que fuera una variable booleana que representara esto.

También pueden existir aplicaciones en que necesitan saber cual es el usuario que está actualmente en la sesión. Para estos casos también se debería dar como prop el id de este usuario a cada uno de estos componentes.

Para solucionar estos casos en que no se desea tener que darle a todos los componentes la misma prop, es que se crea Redux. Redux centraliza el estado de la aplicación, y la lógica para modificar ese estado, dentro de lo que se denomina la *store*. Los componentes entonces pueden acceder a esta store y enviar señales para modificarla. De esta manera no se tiene que manejar la lógica del cambio de estado en cada componente, sino que se aísla la lógica

dentro de esta store, y los componentes solamente deben ir a consultar con la store el valor del dato que necesitan. En la figura 2.6 se contrasta cómo se comporta una aplicación sin una store (sólo React), y una aplicación con store (React con Redux).

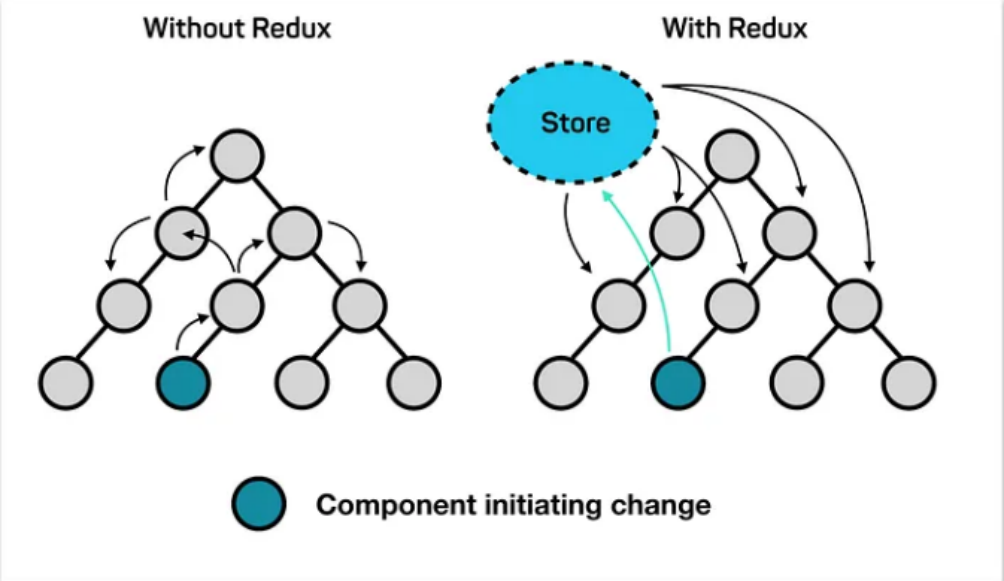

Figura 2.6: Representación de los cambios de estado de una aplicación sin Redux (izq.) y una con Redux (der.). Fuente: Level Up Coding [18].

2.6. Otras tecnologías

Existen variadas tecnologías de diferente índole que se necesitan para el desarrollo de aplicaciones móviles. En esta sección se pretende dar una breve explicación de estas, para proveeer mayor contextualización.

2.6.1. Expo

React Native permite escribir código, pero el proceso de desarrollar una aplicación posee más etapas que sólo escribir código. Además la configuración de React Native tiene una curva de aprendizaje alta debido a que se debe tener conocimiento de cómo operan los sistemas Android e iOS. Expo es una plataforma que abstrae mucha complejidad a la hora de configurar React Native, simplificando aún más el desarrollo de aplicaciones móviles.

Al iniciar una aplicación usando React Native sin Expo, se debe configurar tanto para Android como para iOS. Al usar Expo, esta librería se encarga de manejar la compatibilidad con ambas plataformas, por lo que los desarrolladores no necesitan saber esos detalles.

2.6.1.1. Expo Application Services (EAS)

Una parte crucial del desarrollo de aplicaciones móviles es la compilación del proyecto a un ejecutable binario compatible con la respectiva plataforma nativa. Para dispositivos Android

estos son los archivos apk y aab, mientras que para iOS son los archivos en formato ipa. Para crear un ejecutable en Android se debe usar la aplicación *Android Studio* y para iOS se debe usar la aplicación Xcode (la cuál sólo puede usarse en un Macbook).

Expo Application Services (EAS) es un conjunto de servicios que ofrece hacer esta compilación de los proyectos en la nube. De esta manera se puede compilar el proyecto sin necesidad de tener las herramientas instaladas en el computador de cada desarrollador. Sin esta herramienta, un desarrollador que no tiene un notebook de la marca Apple no podría generar el archivo ejecutable compatible con el sistema iOS.

EAS no sólo ofrece compilar proyectos, también permite automatizar el envío del ejecutable resultante a las tiendas de Google (Android) y Apple (iOS), agilizando este flujo.

2.6.2. CircleCI: Integración Continua

La integración continua (CI) es una práctica de desarrollo de software en la que se realizan integraciones de código frecuentes y automatizadas en un repositorio central. El objetivo de usar CI es detectar errores de integración temprano y reducir el tiempo y el riesgo asociados con la entrega de productos de software.

Circle CI es el servicio de CI que usa Platanus. Cada vez que un desarrollador abre una solicitud de cambios (pull request en inglés), Circle CI ejecuta el código verificando que cumpla los estandáres establecidos y que pase los tests pertinentes (unitarios y/o de sistema). Se puede configurar su comportamiento en el archivo *config.yml* de los proyectos.

2.6.3. Sentry: Monitoreo de Errores

Existen herramientas que monitorean errores en producción de las aplicaciones construidas. Estas herramientas son útiles para identificar y resolver problemas lo más rápido posible, reduciendo el impacto en el rendimiento y la experiencia del usuario de la aplicación.

Existen varias alternativas en el mercado que proveen este servicio de monitoreo de errores de las apliciones, como Dynatrace, Datadog, Sentry, entre otras. Platanus usa Sentry. Esto se debe principalmente a que es open-source y posee una buena integración con Heroku (la plataforma de hosteo que se usa en la empresa).

Sentry además puede integrarse a las aplicaciones móviles construidas sobre Expo, con el módulo *sentry-expo* [19]. Una vez configurado, este permite que los errores que usuarios tienen en la aplicación móvil sean enviados a través de Sentry, además de añadir información adicional como el archivo Javascript (o Typescript) que ocasionó el error, lo que permite poder abordar estos errores con más detalles de lo ocurrido.

2.6.4. TypeScript

TypeScript es un superset de Javascript que le agrega sistema de tipado al mismo. Todo código Javascript es válido también cómo código de Typescript, pero no viceversa. Sin embargo, TypeScript compila a código JavaScript, lo que permite que los proyectos escritos en TypeScript puedan ser ejecutados en cualquier entorno que soporte JavaScript. Durante el

proceso de compilación, TypeScript realiza la verificación de tipos y transforma el código con tipado estático en JavaScript estándar, lo que garantiza que la aplicación sea compatible con diferentes navegadores y dispositivos.

Debido a la capacidad de poder trabajar con código Typescript y Javascript en un mismo proyecto, y la mejora en la detección temprana de errores hacen que TypeScript sea una opción popular para proyectos de desarrollo web y aplicaciones más grandes y complejas [20].

Cómo React Native permite desarrollar aplicaciones móviles usando Javascript, también puede usarse Typescript debido a las razones anteriormente descritas. Para iniciar un proyecto nuevo de Expo con React Native y Typescript, se puede usar el comando *npx create-expo-app -t expo-template-blank-typescript*.

Para comenzar a usar TypeScript en un proyecto de Expo ya existente que usa Javascript, se debe hacer lo siguiente:

1. Cambiar la extensión del archivo que contiene al componente padre de toda la aplicación. Si este archivo se nombra "App", el cambio consistiría en renombrar el archivo de *App.js* (extensión de Javascript) a *App.tsx* (extensión de TypeScript con JSX).

2. Instalar el sistema de tipado que provee TypeScript. Esto puede hacerse usando el comando *npx expo start*.

2.7. La Guía

Platanus posee un alto grado de transmisión de conocimiento en varias aristas de la empresa, tales cómo calidad de código, trabajo con clientes, y cultura. Esto lo logra a través de un sistema centralizado de documentación que se actualiza periódicamente con nueva información.

Los documentos que abordan metodologías organizacionales e información sensible son mantenidos en privado (información de la composición de los equipos, formas de tratar con clientes, entre otras) pero la documentación que se refiere a las prácticas y acuerdos tecnológicos que se han adquirido con el pasar de los años se encuentra pública con el nombre de *la-guía* y se encuentra en la url https://la-guia.platan.us/.

Cualquier integrante interno de Platanus puede agregar nuevas secciones o modificar existentes. Posterior a una revisión estos son actualizados en el sitio web.

Al entrar a la guía, se puede ver un README que muestra las diferentes secciones que hay en la página. La primera sección se denomina"Stack" y comprende todo el stack tecnológico que usa la empresa. Esta sección tiene las siguientes subsecciones:

- Getting Started: ayuda con la creación de un nuevo proyecto web y da links útiles a tecnologías usadas en los proyectos web. No aparece nada de desarrolo móvil

- Ruby/Rails: Esta es la sección que más información posee. Tiene todo lo relacionado a cómo manejar el backend de las aplicaciones que se desarrollan en Platanus.

- Javascript: Aquí hay información sobre Vue, el framework utilizado en la empresa para el frontend de las aplicaciones web. Además, hay información sobre Alpine, que es un framework que se usa para customizar las páginas de administrador generadas por Rails (también relacionado con el desarrollo web).
- CSS: Se especifica que para proyectos web se usa Tailwind para manejar el estilo de las páginas web. Esta sección es chica, y no posee información relacionada a aplicaciones móviles.
- Mobile: Se menciona el stack técnológico que usa la empresa para aplicaciones móviles. Además se da una breve explicación de no más de tres párrafos para cada tecnología involucrada en este tipo de proyectos.

Una de las secciones más importantes es la *Guía de Estilo*. En la figura 2.7 se puede ver el esquema temático que comprende a esta guía. De todas esas secciones, ninguna tiene relación directa con el desarrollo móvil.

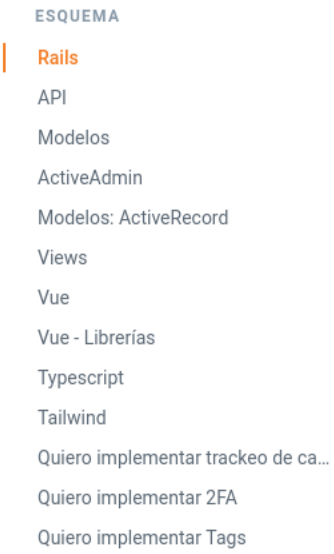

Figura 2.7: Esquema de la guía de estilos de Platanus

Si bien existe traspaso de conocimiento en La Guía en lo que respecta a varios procesos tecnológicos que aparecen en el desarrollo web, no puede decirse lo mismo del desarrollo móvil. En la pregunta 6 de la encuesta de diagnóstico sobre desarrollo móvil en Platanus (ver 1.1.1), el 76,92 % de los desarrolladores respondieron que hay poco material de desarrollo móvil en La Guía.

Al haber una baja cantidad de información en lo que respecta a este ámbito, se vuelve difícil para un integrante de la empresa el poder ser transferido a proyectos de esta índole sin tener que depender totalmente de los colegas que ya tienen experiencia en estos temas.

Capítulo 3

Estándar de Desarrollo de Aplicaciones Móviles

La guía es el medio en que se traspasa y comparte el conocimiento en Platanus. Por lo tanto, para poder generar una mayor colaboración de conocimiento en el área de desarrollo móvil se procedió a crear guías que sirven para iniciarse en el tema, y estándares sobre temas que son comunes a todos los tipos de proyectos móviles que aborda la empresa.

3.1. Guía de Estilos

Si bien es importante dar a los desarrolladores que abordan un proyecto la libertad de poder decidir cómo van a implementar su solución al problema que tienen al frente, estos pueden verse abrumados por el abanico de elecciones que tienen a su disposición. Además, esta libertad de acción es deseable en ciertas áreas de un proyecto, pero existen otras en las que no lo es (por razones de seguridad, uniformidad, calidad de código, entre otras).

Por el otro lado, sí cada detalle del trabajo que hará un desarrollador se encuentra estandarizado, esto puede ser desmotivador y puede inhibir su capacidad creativa lo que va en desmedro del avance del proyecto.

Debido a las razones expuestas, se optó por estandarizar parcialmente la elección de tecnologías nucleares en proyectos móviles, así cómo también la forma en que son configurados. De esta manera los desarrolladores de aplicaciones móviles dentro de Platanus siguen teniendo libertad de decisión creativa en la mayoría de lo que será su día a día en un proyecto, y al mismo tiempo aumentando la uniformidad a lo largo de los diferentes proyectos que puede tener la empresa.

Una consecuencia directa de esto es la disminución en la inicialización de un nuevo proyecto móvil, ya que muchas de las decisiones iniciales ya se encuentran tomadas (más detalles en 3.1.2, 3.2). Además, el tiempo y el esfuerzo que toma la preparación del entorno de desarrollo cuando un desarrollador debe sumarse a un proyecto ya existente también disminuye, debido a la uniformidad en las bases arquitectónicas de este con respecto al resto de proyectos móviles en la empresa.

A continuación se entrará en detalle en las tecnologías que se estandarizaron y los estándares que se decidieron. Cada una de estas secciones fue añadida a La Guía.

3.1.1. TypeScript es el Nuevo Estándar

Typescript posee varias ventajas sobre Javascript a la hora de desarrollar y mantener proyectos de software (ver sección 2.6.4). Platanus migró a TypeScript por sobre Javascript a inicios de 2022 en los proyectos web. Sin embargo, los proyectos móviles seguían utilizando Javascript.

Para mantener consistencia en la empresa, se decidió aplicar el mismo enfoque de usar TypeScript para los proyectos móviles.

3.1.2. Estructura de un Proyecto Móvil

Los proyectos generados por Expo no fuerzan una estructura de carpetas sobre el proyecto ya que prefieren que los usuarios decidan su propia estructura. Esta libertad de elección se plasmaba en Platanus en una diferente estructura para cada proyecto, dependiendo del gusto personal del equipo de desarrolladores que estaba a cargo de este. Si bien esta flexibilidad puede aportar en términos de empoderar a los desarrolladores, viene con la desventaja de no darle una uniformidad a los proyectos de la empresa, lo que genera un roce adicional para los desarrolladores cuando deben cambiar de proyecto.

Todos los proyectos móviles en React Native usan los mismos bloques de construcción, sin importar el tipo de aplicación que se busca construir. Por ende, tener que pensar sobre cómo organizar estos bloques al inicio de cada proyecto sería un proceso que ocupa tiempo innecesariamente.

Estandarizar la estructura de un proyecto aporta en la dirección de reducir el roce de dificultad, y disminuye el tiempo de desarrollo eliminando un problema que es común en todos los proyectos móviles que aborda Platanus.

Se propuso la siguiente estructura para los proyectos móviles en Platanus: en la raíz del proyecto

- **assets**: carpeta con información estática que necesita la aplicación (imágenes, archivos json, svg, entre otros)
- **src**: carpeta principal con el código de la aplicación.
- **App**: archivo con el componente principal de React que contiene toda la aplicación

La carpeta **src** encapsula toda la lógica de la aplicación, y por ende también está estructurada. La estructura es la siguiente:

- **components**: Componentes compartidos de React que se usan en más de una pantalla.
- **hooks**: Aquí se encuentran todos los hooks customizados.

- **navigators**: Todos los navegadores de la app. Estos navegadores se encargan de las transiciones entre diferentes pantallas.
- **providers**: Componentes que "Proveen" de alguna herramienta a sus componentes hijos.
- **screens**: Pantallas completas, que se montan directamente en un navegador. Una pantalla está compuesta de varios componentes.
- **services**: Configuración de servicios que usa la aplicación. Un caso común es la configuración de la API que se usará para comunicarse con el servidor, y la configuración del servicio *i18n* para manejar la traducción a diferentes idiomas.
- **store**: En esta carpeta se encuentran los archivos que componen y configuran el estado centralizado de la aplicación.
- **styles**: Archivos de estilo (colores, márgenes, espaciado, tamaño de texto) que se comparten en la aplicación.
- **types**: Todas las interfaces y tipos (de Typescript) que se usan en más de un archivo se ubican en esta carpeta.
- **utils**: Todas las cosas (funciones, constantes, formateadores) que no caigan en las carpetas anteriores y que se usen en más de un archivo van a esta carpeta.

3.1.3. Uso de aliases en Importación de Archivos

A medida que un producto digital crece, también crecen las carpetas del proyecto que lo compone. Un problema frecuente que puede ocurrir es el de requerir varios componentes y funciones que se encuentran en diferentes carpetas del proyecto, y para llegar a estos archivos se debe apuntar a su PATH teniendo como punto de partida el PATH en el que se encuentra el archivo que los quiere usar. Esto se conoce comúnmente como importaciones relativas, y en la figura 3.1 se ve cómo lucen las importaciones de un archivo usando esta aproximación.

Hay dos problemas con este tipo de declaración de importaciones: son verbosas y son confusas. Verbosas debido a los largos PATHS de importación de cada archivo, y confusos debido a que no se sabe a primeras el contexto de cada archivo debido a los consecutivos "../../".

```
import * as ImagePicker from 'expo-image-picker';
import React, { useState } from 'react';
import styled from 'styled-components/native';

import CameraComponent from '../../../../components/Camera/Camera';
import ExpandableImage from '../../../../components/ExpandableImage/ExpandableImage';
import { ActionButton } from '../../../../components/buttons';
import { CameraPermissionsModal } from '../../../../components/modals';
import { usePartialAnswer, useTask } from '../../../../hooks';
import TaskContainer from '../TaskContainer/TaskContainer';
import { StyleUtils } from '../../../../styles/styles';
import compressImage from '../../../../utils/compress-image';
import { logSentryIssue } from '../../../../utils/event-logger';
import { CHOOSE_PICTURE_TASK_IDENTIFIER } from '../../../../utils/task-type-identifiers';
```

Figura 3.1: Imports relativos en un archivo

Existe otra aproximación a la hora de importar archivos, y se conoce como importación absoluta. Esta se refiere a usar el PATH absoluto del archivo que se quiere importar, usando la raíz del proyecto como base. Si aplicamos la estructura de carpetas que se detalló en 3.1.2, el resultado que se obtiene puede verse en la figura 3.2. Esta aproximación ya no es confusa, debido a que si entendemos previamente el contexto de la estructuración del proyecto, es fácil contextualizar los archivos que se importan. Sin embargo, este método no resuelve el problema de verbosidad en las importaciones: podemos ver que el texto "src" se repite constantemente en las declaraciones de las importaciones.

```
import * as ImagePicker from 'expo-image-picker';
import React, { useState } from 'react';
import styled from 'styled-components/native';

import CameraComponent from '/src/components/Camera/Camera';
import ExpandableImage from '/src/components/ExpandableImage/ExpandableImage';
import { ActionButton } from '/src/components/buttons';
import { CameraPermissionsModal } from '/src/components/modals';
import { usePartialAnswer, useTask } from '/src/hooks';
import TaskContainer from '/src/screens/Routine/tasks/TaskContainer/TaskContainer';
import { StyleUtils } from '/src/styles/styles';
import compressImage from '/src/utils/compress-image';
import { logSentryIssue } from '/src/utils/event-logger';
import { CHOOSE_PICTURE_TASK_IDENTIFIER } from '/src/utils/task-type-identifiers';
```

Figura 3.2: Imports absolutos sin alias en un archivo

Es en este punto en que se vuelve especialmente útil el uso de "aliases". En este contexto, un alias es una forma de nombrar de una manera diferente una porción del PATH del proyecto. De esta forma podemos personalizar el PATH de los archivos que importamos.

Para abordar la verbosidad expuesta en la figura 3.2 teniendo en cuenta la estructura de carpetas descrita en la sección 3.1.2, se proponen aliases para cada una de las carpetas que componen el proyecto. En la tabla 3.1 se pueden ver las carpetas mencionadas y sus correspondientes aliases.

Tabla 3.1: Carpetas de un proyecto y sus respectivos aliases

Carpetas	Alias
src	@
assets	@assets
components	@components
hooks	@hooks
navigators	@navigators
providers	@providers
screens	@screens
services	@services
store	@store
styles	@styles
types	@types
utils	@utils

Al aplicar estos aliases sobre las importaciones absolutas de la figura 3.2, se obtiene la figura 3.3. Estas importaciones no son confusas: poseen el contexto en su PATH de importación. Además, no son verbosas: no repiten texto innecesariamente, y con los aliases eliminan el uso de la frase "src".

```
import * as ImagePicker from 'expo-image-picker';
import React, { useState } from 'react';
import styled from 'styled-components/native';

import CameraComponent from '@components/Camera/Camera';
import ExpandableImage from '@components/ExpandableImage/ExpandableImage';
import { ActionButton } from '@components/buttons';
import { CameraPermissionsModal } from '@components/modals';
import { usePartialAnswer, useTask } from '@hooks';
import TaskContainer from '@screens/Routine/tasks/TaskContainer/TaskContainer';
import { StyleUtils } from '@styles/styles';
import compressImage from '@utils/compress-image';
import { logSentryIssue } from '@utils/event-logger';
import { CHOOSE_PICTURE_TASK_IDENTIFIER } from '@utils/task-type-identifiers';
```

Figura 3.3: Imports absolutos con aliases

3.2. Módulo de Estados

Las aplicaciones móviles basadas en React Native, al ir adquiriendo tamaño y complejidad comúnmente necesitan en su arquitectura un módulo que sea responsable de mantener el estado de la aplicación (nombre de usuario, token de autenticación, entre otros). A este módulo se le conoce como la *store* de la aplicación.

Existen diferentes librerías para lograr esto, cómo Easy-Peasy (https://easy-peasy.vercel.app/) o Redux (tecnología descrita en sección 2.5). Se decidió usar Redux debido a su

popularidad y mayor volumen de contenido en línea a la hora de querer consultar información.

3.3. Material de Apoyo

3.3.1. Uso de Tailwind en React Native

Cómo se mencionó en 2.7, Platanus usa Tailwind para estilizar las páginas en proyectos web. Sin embargo, React Native no soporta nativamente esta tecnología y se debe utilizar una librería que haga la conexión entre Tailwind y React Native. La librería que se usa para esto se llama *tailwind-rn*.

Esta librería necesita ciertos pasos adicionales que deben tomarse antes de poder efectivamente usar tailwind en los componentes de React Native. Se debe iniciar un servidor aparte, que escucha los cambios hechos al proyecto para actualizar las funcionalidades de tailwind que se están usando. Además se debe importar una función llamada *useTailwind* para poder usar tailwind en los componentes.

Se creó una guía que entra en mayor detalle sobre los puntos mencionados, y da ejemplos sobre cómo usar esta librería en proyectos móviles. Esta se encuentra en el link https://la-guia.platan.us/stack/mobile/styling/usando_tailwind_en_react_native, y también está disponible en la sección A.1 del anexo.

3.3.2. Creación y Conexión de una Slice de Redux

En la sección 3.2 se estableció Redux cómo la librería encargada de manejar el estado centralizado de las aplicaciones móviles en Platanus. Para complementar esta estandarización, se creo una guía que muestra el paso a paso de crear una partición (slice en inglés) de información en esta store, cómo crear la lógica que modifica esta data y finalmente cómo conectar esta partición al resto de la aplicación.

Esta guía es de visibilidad pública y puede accederse a ella en La Guía, en el siguiente link: https://la-guia.platan.us/stack/mobile/redux/crear_y_conectar_una_slice_en_redux. Además, se encuentra disponible en A.2 en el anexo.

Capítulo 4

Cavendish: El Generador de Proyectos Móviles de Platanus

Si bien en la sección 2.7 se mencionó que había varios aspectos que no se encontraban estandarizados en lo que respectaba a proyectos móviles en Platanus, si existían acuerdos sobre ciertas tecnologías involucradas en este tipo de proyectos.

Cuando arribaba un nuevo proyecto móvil, se sabía de antemano que este iba a ser abordado usando React Native en conjunto con Expo como base; que usaría Tailwind para abordar la estilización de los componentes (por sobre la alternativa nativa de React Native); ESLint para controlar la gramática del código; CircleCI como proveedor de integración continua. Estos acuerdos se debían a que Platanus ya usaba Tailwind, ESLint y CircleCI en los proyectos web, lo que permite transferir conocimiento que ha sido acumulado con las experiencias de esos otros proyectos.

Configurar las tecnologías mencionadas al inicio de cada proyecto es repetitivo y consume tiempo. Es un proceso que puede ser automatizado, y en Platanus esto se hace a través de Cavendish, un generador open source de proyectos móviles mantenido dentro de la empresa. Este generador es una gema de Ruby que crea un proyecto en Expo y le agrega capas de configuración para acelerar el inicio este. El repositorio de Cavendish se encuentra en Github en el link https://github.com/platanus/cavendish.

4.1. Situación Inicial

El desarrollo de Cavendish se encontraba detenido. A inicios de febrero de 2023, la última solicitud de cambios que se le había hecho al repositorio en GitHub databa de diciembre del año 2021. Debido al tiempo en que la gema estuvo sin mantención, esta poseía paquetes desactualizados que ya no eran compatibles entre sí. Además, en el tiempo que dejó de desarrollarse Cavendish, surgieron nuevas herramientas para hacer deploy en Expo, que requería agregar capas extra de configuración.

El proyecto se encontraba inutilizable en ese estado.

4.2. Solución

Dentro del proyecto, en el archivo lib/cavendish/cli.rb se encuentra la función create-command-steps que determina el orden de las configuraciones que se hacen. La lógica de cada paso se encuentra encapsulada por comandos. En el código 4.1 se puede ver los comandos existentes previo comienzo de la reanudación del desarrollo de Cavendish.

Código 4.1: Comandos de configuración en Cavendish (versión inicial).

```
1
2   def create_command_steps
3     [
4       Cavendish::Commands::CreateExpoProject,
5       Cavendish::Commands::AddTailwind,
6       Cavendish::Commands::AddReactNavigation,
7       Cavendish::Commands::AddEslint,
8       Cavendish::Commands::AddTesting,
9       Cavendish::Commands::AddCiConfig,
10      Cavendish::Commands::AddReadme,
11      Cavendish::Commands::ConfigureGit
12    ]
13  end
```

Entonces, el trabajo en Cavendish de actualizar cierta configuración se refiere a actualizar el comando pertinente a esa configuración. Asimismo, si se quiere agregar configuraciones implica agregar nuevos comandos, en el orden que se estime conveniente.

En la tabla 4.1 se pueden ver los comandos iniciales que se encontraban en el proyecto, y sus funciones

Tabla 4.1: Comandos iniciales en Cavendish

Comando	**Descripción**
CreateExpoProject	Crea un nuevo proyecto Expo.
AddTailwind	Agrega y configura la librería tailwind-rn al proyecto.
AddReactNavigation	Integra React Navigation en el proyecto, permitiendo la navegación entre diferentes pantallas/componentes.
AddEslint	Configura ESLint para el proyecto, proporcionando reglas de linting y calidad de código.
AddTesting	Configura el ambiente de testeo, permitiendo elegir entre los frameworks enzyme y jest.
AddCiConfig	Agrega configuración de integración continua.
AddReadme	Crea un archivo README con documentación e instrucciones del proyecto.
ConfigureGit	Configura Git para el proyecto, incluyendo la eliminación de archivos .git generados en pasos anteriores y la creación del primer commit.

4.2.1. Actualización del Proyecto

El primer paso para poder tener un generador de aplicaciones móviles que refleje el estándar de la empresa, es actualizarlo para que vuelva a funcionar con las funcionalidades que proveía inicialmente. Habían tres aristas que no estaban permitiendo que el generador ya construido estuviera funcional:

1. Expo había cambiado la forma en que se usaba su CLI, y Cavendish seguía usando la forma antigua.

2. A la hora de configurar la librería de testing, Cavendish permitía elegir entre Enzyme y React Testing Library (RTL). Lamentablemente Enzyme dejó de funcionar y por ende se estaba ofreciendo una opción que ya no funcionaba.

3. Para poder usar Tailwind en React Native, se estaba usando la librería *tailwind-rn*, en específico la versión 2 de la librería. Esta versión se encontraba deprecada y la versión más reciente requería una configuración diferente a la que estaba implementada.

4.2.1.1. Actualización de Expo CLI

Inicialmente para usar Expo se debía descargar una CLI global llamada *expo-cli* usando Node Package Manager (NPM), un administrador de paquetes para Javascript. Esto generaba ciertas inconveniencias ya que se tenía que controlar manualmente la versión de la CLI, y también ocasionaba que el CI se demorará más debido a que se debía instalar esta herra-

mienta antes de comenzar a ejecutar los chequeos sobre el proyecto .

Para mitigar estos problemas, el equipo de Expo decidió migrar desde esta herramiento global, a una local que venía incorporada a cada proyecto. En el blog que publicó la organización al momento de anunciar el cambio mencionaban: "*A diferencia de la CLI global, la CLI local se instala en cada proyecto y se mantiene en la misma versión que el código ejecutado dentro del proyecto, lo que significa que los proyectos son más fáciles de configurar, siguen funcionando durante más tiempo y las actualizaciones de las herramientas son menos frecuentes.*"[21]

Cavendish inicialmente usaba la versión global de Expo, y en su archivo *lib/cavendish/-version.rb* controlaba manualmente su versión.

Para realizar el cambio, se tuvo que cambiar el comando *Command::CreateExpoProject* para que *npx create-expo-app* en vez de de *expo init*. Debido a esto, ya no erá que Cavendish instalara la CLI global de expo y por ende se eliminó esa instalación.

Estos cambios poseen una gran ventaja en términos de mantenibilidad del proyecto, debido a que ya no es necesario controlar el versionamiento de la CLI de expo mediante una variable global dentro del proyecto, sino que expo a través a del comando *npx create-expo-app* controla la versión de la CLI local.

4.2.1.2. Eliminación de Enzyme y Limitación a la Configuración

Cavendish es open-source, por lo que cualquier persona puede usarlo para generar un nuevo proyecto móvil. Con eso en mente, inicialmente se construyó pensando en que usuarios pudieran elegir qué partes querían configurar y qué partes no. También proveía opciones sobre posibles configuraciones en cierto ámbito; este era el caso de la configuración de la librería de testing para el proyecto, que permitía elegir entre Enzyme y React Testing Library (RTL). Enzyme dejó de funcionar desde desde React 18 [22] por lo que se debía eliminar la opción de poder elegirla (además era la opción por defecto).

Se aprovechó esta instancia para reflexionar sobre la idea de permitir que los usuarios pudieran elegir qué querían configurar, y qué no. Se concluyó que si bien Cavendish es open-source y usuarios con diferentes necesidades pueden ocuparlo, en la práctica el uso de Cavendish se encuentra fuertemente concentrado dentro de la empresa. Debido a esto, se cambió el rumbo de permitir configurabilidad de cada tecnología por separado, y se optó por uno en que simplemente se configuran todas las tecnologías sin permitir configurabilidad sobre cuál agregar o cuál no. Esta decisión puede ir en detrimento de los usuarios que quieren sólo una porción de las tecnologías que se ofrecen, ya que no podrán elegir solamente las que necesitan. Sin embargo, restringir la elección de las tecnologías a configurar disminuye considerablemente la complejidad sobre el proyecto a la hora de querer modificar algo existente o agregar una característica nueva al generador.

Debido a lo anterior, se eliminó Enzyme del generador, y también se eliminó la opción de preguntarle al usuario cuál librería de testing prefiere. En el rumbo a futuro del proyecto tampoco se tiene pensado agregar la opción de que el usuario pueda elegir entre ciertas con-

figuraciones.

4.2.1.3. Actualización de Tailwind

En la sección 3.3.1 se mencionó que para implementar Tailwind en React Native se usa la librería *tailwind-rn*. Cavendish automatizaba la configuración de esta librería en los proyectos móviles que generaba, pero estaba usando una versión antigua.

Tailwind (no confundir con *tailwind-rn*) tuvo un gran cambio al subir una de sus versiones. Esto implicó qué *tailwind-rn* también tuvo que actualizar varias procesos de su configuración para poder seguir manteniendo la compatibilidad entre Tailwind y React Native, lo que desembocó en la versión 4 de la librería *tailwind-rn*. El cambio implicaba una diferencia radical a la hora de configurar y usar la librería. Al inicio de la guía A.1 del anexo se comenta brevemente sobre la diferencia de comportamiento de la librería en versiones inferiores a la 4.

Para actualizar la librería *tailwind-rn* a su versión 4, se hace lo siguiente:

1. Posterior a instalar la librería con el comando *yarn add tailwind-rn*, se debe correr en la terminal el comando *npx setup-tailwind-rn*, que ejecuta un script que creó la librería para agilizar la configuración.

2. Se elimina el comando que instalaba Tailwind con un versionamiento manual, ya que esto lo maneja el script que se describe en el paso anterior. Este cambio posee implicancias similares a las de eliminar el versionamiento manual de la CLI de Expo (ver 4.2.1.1) en términos de mantenibilidad del proyecto.

3. Se agrega el componente TailwindProvider de forma que "envuelve" a toda la aplicación (ver figura 4.1). Este componente permite que todos sus componentes hijos tengan acceso a la función *useTailwind*.

```
import {TailwindProvider} from 'tailwind-rn';
import utilities from './utilities.json';

const App = () => (
        <TailwindProvider utilities={utilities}>
                <MyComponent />
        </TailwindProvider>
);
```

Figura 4.1: Componente *TailwindProvider* de la librería *talwind-rn* envolviendo toda la aplicación.

4. Cavendish posee una plantilla que se añade a los proyectos móviles a modo de que posean una pantalla inicial cuando se inicia el proyecto. Se modifica esta plantilla para que ahora ocupe Tailwind con el nuevo comportamiento, es decir, a través de la función *useTailwind* mencionado en el paso previo. Enla figura 4.2 se pueden ver los cambios hechos a esta plantilla.

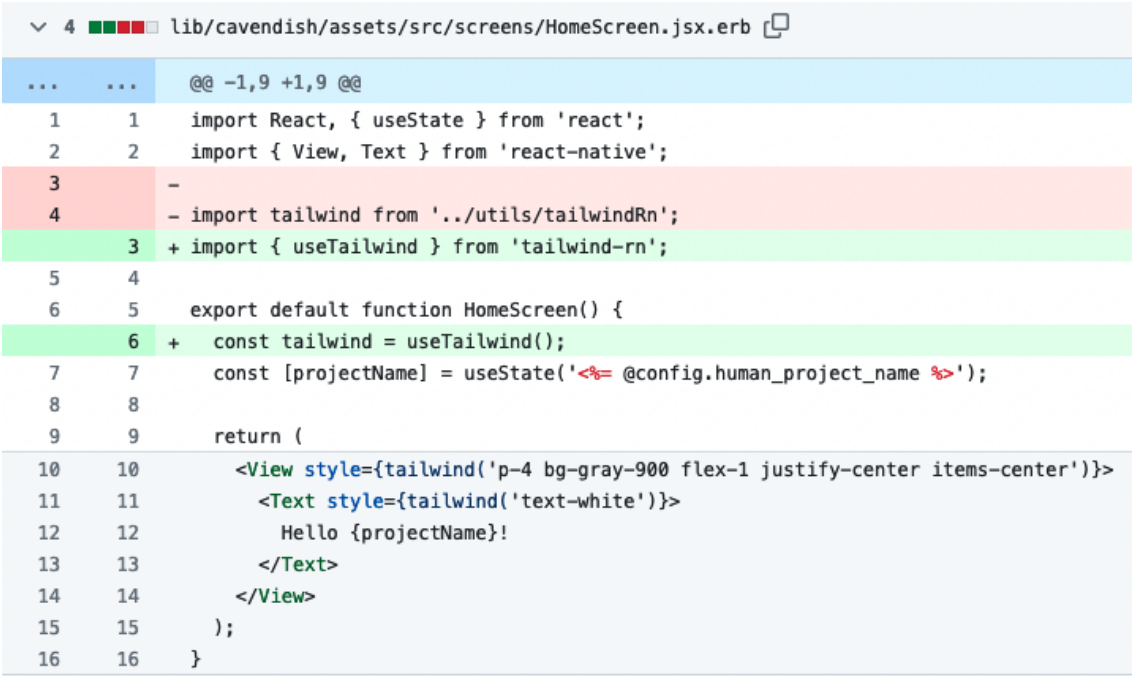

Figura 4.2: Modificación hecha a la pantalla de inicio que inyecta Cavendish en los proyectos para que ocupe *tailwind-rn* acorde a la versión 4 de la librería. La variable *tailwind* se obtiene a través de invocar la función *useTailwind* dentro de los componentes.

Los cambios descritos fueron realizados en su mayoría en el comando AddTailwind de Cavendish (ver tabla 4.1). Además, se agregaron dos nuevos comandos para mantener un enfoque modular en el proyecto:

- AddTemplateFiles: comando encargado de manejar la inyeccción de plantillas al proyecto móvil generado.
- PatchDependencies: comando encargado de parchar dependencias de terceros (librerías). Esto fue agregado ya que *tailwind-rn* debió ser parchado debido a un error de tipos que había con TypeScript. Este error posee un issue en github el cuál puede ser revisado en la siguiente url: https://github.com/vadimdemedes/tailwind-rn/issues/169.

4.2.2. Conexión con EAS

Cómo fue descrito en la sección 2.6.1.1, los proyectos generados con Expo tienen la posibilidad de conectarse con Expo Application Services (EAS) para poder compilar el proyecto, enviarlo a las tiendas de Apple y Google, y enviar actualizaciones inálambricas (*over-the-air* u OTA). Cavendish no configuraba este aspecto de los proyectos ya que pausó su desarrollo antes de que EAS fuera adoptado en la empresa. Actualmente todos los proyectos móviles de Platanus usan EAS por lo que hace sentido configurarlo a través de este generador.

Con la introducción de EAS, también vino su propia CLI llamada *eas-cli* la cual maneja aspectos cómo las credenciales del usuario y del proyecto. La CLI también permite especificar para qué plataforma compilar el proyecto (Android o iOS) y para qué ambiente hacerlo (development, staging, producción), así como también las mismas opciones para enviar ac-

tualizaciones OTA.

Para conectar los proyectos generados por Cavendish con EAS, se crea el comando *AddEASConfiguration* que encapsula la lógica de esto. El comando registra y configura el proyecto en EAS siguiendo estos pasos:

- Se instala *eas-cli*.
- Se muestra un mensaje en donde se le dice al usuario que si no tiene una cuenta de usuario en EAS, debe crearse una. Junto al mensaje aparece una url que lo direcciona al lugar en donde puede hacer eso. Se optó por esta aproximación debido a que al momento de hacer esta configuración, la CLI de EAS no poseía una opción de crear una nueva cuenta a través de la CLI.
- Se pueden crear organizaciones para colaborar entre usuarios de EAS. Esta es la forma recomendada para trabajar en un proyecto. Al igual que en el paso anterior, se muestra un mensaje junto a una url, donde el usuario puede crear una organización. La CLI de EAS no soporta crear organizaciones a través de esta, y por eso se decidió abordarlo de esta forma.
- Se hace login en la CLI de EAS, para que los siguientes pasos sean con las credenciales del usuario.
- Se configura al dueño del proyecto en EAS. En este paso el usuario puede elegir donde crear el proyecto: dentro de su cuenta usuario o dentro de alguna organización a la que pertenezca.
- Se agrega la configuración inicial de EAS para poder enviar a la nube la compilación del proyecto.
- Se inyecta una configuración avanzada que crea tres ambientes de desarrollo sobre los que el usuario a futuro podrá elegir para compilar el proyecto: development, staging y producción.

Se puede ver el archivo que contiene al comando AddEASConfiguration en la siguiente url: https://github.com/platanus/cavendish/blob/1c77946c7de09d536f4ae0d0b780ce6aa4f3336e/lib/cavendish/commands/add_eas_configuration.rb, y en la sección B.1 del anexo.

4.2.3. Configuración de Redux

En la sección 3.2 se mencionó que se eligieron redux como la tecnologías encargada de manejar el estado centralizado de la aplicación (store). La configuración de esta tecnología no se encontraba en cavendish debido a que no existía un concenso claro en este punto. Al haber ahora un concenso, se procedió a automatizar la configuración de redux con Cavendish.

Se creó el comando *Cavendish* :: *Commands* :: *AddRedux* que encapsula parte del proceso de configurar redux. En este comando se instalan las dependencias necesarias las cuales son:

- redux: librería principal

- redux-thunk: librería que añade soporte para actualizar la store con llamadas asíncronas
- @reduxjs/toolkit: librería auxiliar que simplifica varios procesos de redux

Luego, se modifican ciertos archivos existentes y se agregan nuevos. Para que la aplicación pueda acceder a la store de redux, se debe "envolver" en un componente proveedor de redux. Este proceso es análogo al que aparece en la figura 4.1 que muestra cómo se debe envoler la aplicación con un componente proveedor.

Además, se agrega la configuración inicial de la *store* que provee redux en la carpeta *src/store/index.ts* y *src/store/reducers.ts*. La ubicación de estos archivos es consistente con la estructura de carpetas que se plantea en la sección 3.1.2.

Capítulo 5

TaskTracker

TaskTracker es una aplicación de carácter "Business to Business", que permite a jefes de operaciones crear rutinas de tareas customizadas, que luego sus empleados deben ejecutar en su día a día en el trabajo.

La aplicación posee dos tipos de usuarios: los usuarios administradores que construyen y envían las rutinas de tareas, y los usuarios no administradores que son los que tienen que usar la aplicación móvil para ejecutar estas rutinas. Como los usuarios administradores no usan la aplicación móvil para crear las rutinas (usan un panel de administrador creado con Ruby on Rails), cuando se hable de "los usuarios de TaskTracker" se referirá a los usuarios no administradores que deben usar la aplicación móvil.

Entonces, los usuarios de TaskTracker usan la aplicación para ejecutar estas rutinas de tareas, y los datos de sus ejecuciones son subidas a un servidor donde los jefes de operaciones pueden ver el desempeño. Algunas de las empresas que usan TaskTracker tienen bonos monetarios que están sujetos al desempeño de los usuarios en esta aplicación. Debido a lo anterior, si la aplicación falla y un usuario no puede completar exitosamente sus rutinas, esto puede impactar en la remuneración que recibe a final del mes. La aplicación posee alrededor de 2.000 usuarios activos en su plataforma.

5.1. Situación Inicial

El proyecto TaskTracker fue desarrollado desde cero en Platanus, comenzando su desarrollo a mitad del 2017. No hay planes de que termine pronto. El proyecto lleva casi 6 años en Platanus y si bien ha crecido muchísimo en volumen de código y cantidad de usuarios que lo usan, también ha aumentado su deuda técnica.

Al haber pasado varios años desde sus inicios, algunas tecnologías que se usan en este proyecto no se usan en otros proyectos, como lo es el caso de Tailwind para estilizar componentes, y TypeScript como lenguaje de programación. El proyecto tampoco poseía la estructura planteada en la sección 3.1.2 y los alias que se discuten en la sección 3.1.3. Esta situación va en desmedro de la intención de querer mejorar la consistencia entre diferentes proyectos.

5.2. Reingeniería de TaskTracker

En esta sección se describen los temas que se abordaron en el proyecto TaskTracker para volverlo consistente con el nuevo estándar que se está impulsando a través de nuevos acuerdos y la automatización de estos.

5.2.1. Introducción de TypeScript al Proyecto

El primer paso en la dirección de querer garantizar mayor seguridad y traspaso de conocimiento entre desarrolladores, es de agregar Typescript a la aplicación. Typescript provee un sistema de tipado fuerte lo que permite encontrar posibles bugs antes de que ocurran. Esta decisión va en línea con lo mencionado en 3.1.1.

Debido a la compatibilidad que existe entre Javascript y TypeScript, es posible migrar el proyecto parcialmente, componente a componente. Inicialmente se comenzó migrando el componente raíz del proyecto, *App*. Luego, se establecieron dos acuerdos en el equipo de desarrollo. El primero es que nuevos archivos deben ser creados en TypeScript. El segundo se refiere a que cuando se debe actualizar un archivo ya existente, se debe pasar a TypeScript dentro de lo posible (plazos de tiempo, complejidad de la migración).

5.2.1.1. Tipado de las Tareas

Agregar Typescript abrió la puerta a poder agregarle tipado a ciertos modelos recurrentes dentro de la aplicación. Se priorizó el tipado del modelo que representa a las tareas en la aplicación (*Task* en inglés).

Se eligió este modelo de datos debido a que es el modelo con el que los usuarios interactúan la mayor cantidad del tiempo, cuando están ejecutando tareas. En la figura 5.1 se puede ver la definición del tipo *Task* en TypeScript.

Una vez habiendo definido el tipo, se debe hacer que los componentes tengan acceso a este. Una opción es migrar todas las tareas a TypeScript para que puedan usar este tipo, pero no es práctico en términos de tiempo, ya que existen diecinueve tipos de tareas diferentes.

Para lograr agregarlo tipado a todas las tareas sin necesidad de tener que migrar todas las tareas, primero se centralizó la llamada a las tareas en una sola función. Para eso se refactorizaron todas las tareas existentes para cambiar la forma en que accedían al modelo de datos de la tarea. Si bien esto implicó una inversión de tiempo no menor, sigue siendo más conveniente que migrar todos los archivos que contienen las tareas de la aplicación. Finalmente, la función que llama a las tareas se migra a TypeScript en un archivo diferente, y se le agrega tipado a su firma.

5.2.2. Reestructuración del Proyecto y Alias en Importaciones

En la sección 3.1.2 se entró en detalle sobre la nueva estructura de carpetas modularizada que poseen los nuevos proyectos de Platanus. Esta estructura no se reflejaba con el estado inicial de TaskTracker.

Para aumentar la uniformidad entre proyectos móviles, se reestructuró el proyecto para que quedara coherente con la estructura propuesta, y en el proceso se aprovechó también de

```typescript
import type DynamicList from '@/types/tasks/dynamicList';
import type TaskOutputType from '@/types/tasks/taskOutputType';

interface Task {
  id: number,
  title: string,
  description: string,
  index: number,
  outputType: TaskOutputType,
  options: string[] | null,
  conditionalTaskId: number | null,
  conditionalValue: string,
  documentUrl: string,
  regex: string,
  code: string,
  optional: boolean,
  valueCondition: string,
  nested: boolean,
  fullAnswer: boolean,
  justifiable: boolean,
  userValueAttribute: string,
  fetchEndpoint:string,
  fetchHeaders: string,
  highlighted: boolean,
  highlightTitle: string,
  useRestrictions: boolean,
  restrictions: string,
  updatedAt: string,
  randomizeOptions: boolean,
  dynamicList: DynamicList | null,
  localUri?: string,
}
```

Figura 5.1: Definición de tipo *Task* usando TypeScript.

configurar el estilo de importación basado en aliases descrito en 3.1.3.

Para lograr esto, se creó la carpeta *src*, y se movieron las carpetas ya existentes hacía dentro de la nueva creada. Al cambiar la ubicación de las carpetas, se tuvo que actualizar el PATH de importacion de todos los archivos contenidos en estas. Se hizo uso de las herramientas disponibles en editores de código para que estos actualizaran los PATH automaticamente.

En conjunto con la reestructuración del proyecto, se implementaron los aliases de importaciones que fueron descritos en la sección 3.1.3, siguiendo la nomenclatura descrita en la tabla 3.1.

Capítulo 6

Evaluación

En la sección 1.1.1 se presentaron los resultados de la encuesta de diagnóstico inicial sobre el estado del desarrollo móvil en Platanus. La encuesta comprendía preguntas que se basaban en experiencias personales cómo las preguntas 1 y 3, así cómo también preguntas más enfocadas a la percepción qué existía sobre el área dentro de la empresa (preguntas 6 y 7).

Para validar el trabajo realizado en este ámbito, se realizó la misma encuesta a los desarrolladores de la empresa, para después proceder a contrastar los resultados obtenidos. La encuesta fue respondida por 12 desarrolladores de un total de 16. Es importante mencionar que entre la realización de la encuesta de diagnóstico y la realización de la encuesta de validación, dos desarrolladores se fueron de la empresa, e ingresó una nueva desarrolladora. De los desarrolladores que se fueron, uno era el creador de Cavendish.

A continuación se muestran los datos arrojados de la encuesta, así cómo también la variación porcentual de las respuestas respecto a la encuesta inicial de diagnóstico.

1. ¿Tienes experiencia en el desarrollo móvil?
   - Si: 58,3 % ($\downarrow 3,2\,\%$)
   - No: 41,7 %. ($\uparrow 3,2\,\%$)
2. ¿Te gustaría desarrollar móvil en el corto/mediano plazo?
   - Si: 100 % ($\uparrow 30,8\,\%$)
   - No: 0 % ($\downarrow 30,8\,\%$)
3. ¿A cuál(es) de estas tecnologías/procesos te has enfrentado antes?
   - Expo: 20,0 % ($\uparrow 0,6\,\%$)
   - React: 33,3 % ($\downarrow 2,8\,\%$)
   - React Native: 23,3 % ($\uparrow 1,1\,\%$)
   - Redux/easy-peasy: 10,0 % ($\downarrow 1,1\,\%$)
   - Subir app a las tiendas de aplicaciones: 13,3 % ($\uparrow 2,2\,\%$)
4. De las cosas mencionadas, ¿Cuáles dirías que son las que menos conoces?
   - Expo: 22,6 % ($\downarrow 1,7\,\%$)

- React: 9,7 % (↑ 3, 6 %)
- React Native: 9,7 % (↓ 2, 4 %)
- Redux/easy-peasy: 32,3 % (↓ 1, 1 %)
- Subir app a las tiendas de aplicaciones: 25,8 % (↑ 2, 2 %)

5. De las cosas mencionadas, ¿Cuáles dirías que son las más cruciales para sentirte preparado para abordar un proyecto móvil?

   - Expo: 26,7 % (↓ 1, 5 %)
   - React: 13,3 % (↓ 5, 4 %)
   - React Native: 36,7 % (↓ 0, 8 %)
   - Redux/easy-peasy: 16,7 % (↑ 4, 2 %)
   - Subir app a las tiendas de aplicaciones: 6,7 % (↑ 3, 5 %)

6. ¿Cuál es tu percepción sobre el material disponible en Platanus (la guía, presentaciones, notion) para aprender las tecnologías involucradas?

   - Hay mucho material: 8,3 % (↑ 8, 3 %)
   - Hay algo de material: 75,0 % (↑ 51, 9 %)
   - Hay poco material: 16,7 % (↓ 60, 3 %)

7. ¿Qué tan preparado te sentirías si la próxima semana tuvieras que entrar a un proyecto de desarrollo móvil?

   - Muy preparado:16,7 % (↑ 9, 1 %)
   - Algo preparado: 50,0 % (↓ 3, 8 %)
   - Nada preparado: 33,3 % (↓ 5, 2 %)

8. Si se realiza una presentación de desarrollo móvil, ¿De qué te gustaría que se trate? ¿Cuál encontrarías de mayor utilidad para ti?

   - React: 27,3 % (↓ 11, 2 %)
   - React Native: 36,4 % (↑ 21, 0 %)
   - Expo: 36,4 % (↓ 2, 1 %)
   - Publicar una App en la App Store(iOS) y Play Store (Android): 0 % (↓ 7, 6 %)

Se puede ver que las preguntas 1, 3 y 4 que tienen directa relación con la experiencia de cada desarrollador (tecnologías usadas y no usadas) no poseen una gran variación. Esto es esperable ya que la respuesta del desarrollador depende de sí son asignados o no a algún proyecto móvil, y en el periodo comprendido entre la encuesta inicial y la final la asignación de desarrolladores a proyectos móviles no ha fluctuado notablemente.

También es posible ver que las respuestas a las preguntas relacionados con percepción del área de desarrollo móvil en la empresa (2, 6 y 7) son las que presentan mayor variación porcentual. Se aprecia que el entusiasmo por querer participar de proyectos de desarrollo móvil ha aumentado en un 30,8 %, la percepción de que existe poca documentación disponible en la empresa bajó un 60,3 % y el porcentaje de desarrolladores que se sienten “Muy preparado” para comenzar un proyecto móvil aumentó en un 9,1 %.

6.1. Documentación y Guía de estilos

A partir de los resultados de la encuesta, en específico de los resultados de la pregunta 6, se puede ver que la percepción de los desarrolladores de la empresa sobre la documentación de las tecnologías involucradas en los proyectos de desarrollo móvil ha evolucionado positivamente, siendo entonces bien evaluado este objetivo por el resto de la empresa.

6.2. Cavendish

En el periodo comprendido entre el inicio y término de esta memoria, arribó un nuevo proyecto móvil a la empresa, al cuál se le nombrará Organizer en este documento. La aplicación móvil del proyecto fue generada usando Cavendish, donde se pudo validar que las configuraciones automatizadas funcionaban acorde a lo esperado, y demostrar que el generador había vuelto a un estado funcional. Organizer se encuentra actualmente en producción y ya se efectuó exitosamente el traspaso de la mantención del proyecto a su organización.

Indirectamente, los resultados de la encuesta final también sirven para evaluar positivamente Cavendish, debido a que algunas de las guías escritas mencionan a Cavendish (ver guías A.1 y A.2).

6.3. TaskTracker

La razón de querer hacer una reingeniería sobre TaskTracker era para que el proyecto quedara consistente con los nuevos cambios que se propusieron en el primer objetivo y se automatizaron en el segundo. Los nuevos proyectos móviles en Platanus usan TypeScript como lenguaje de programación, usan Tailwind para estilizar el diseño, poseen una estructura de carpetas descrita en la sección 3.1.2, y poseen alias para la importación de archivos. Previo al proceso de reingeniería, el proyecto TaskTracker no cumplía ninguna de estas características.

Una vez finalizada la reingeniería, TaskTracker posee un 5,6 % de sus archivos en el lenguaje TypeScript, lo que corresponde a 16 archivos que usan este lenguaje, en comparación con 238 que usan Javascript. De esos 16 archivos, 12 corresponden a nuevos archivos y 4 corresponden a archivos migrados. Además de Typescript, el proyecto ahora posee la estructura descrita en 3.1.2 y el alias de importación. El proyecto sigue sin usar Tailwind para los estilos de los componentes.

Debido a lo anterior, se evalúa que la reingeniería cumple completamente el objetivo en 2 puntos (estructura de proyecto y alias de importación), parcialmente en el punto de migrar a TypeScript, y no logra cumplir en el punto de usar Tailwind.

Capítulo 7

Conclusiones

Este trabajo buscaba impulsar el área de desarrollo de aplicaciones móviles en Platanus creando conocimiento sustentable en el tiempo, que nutriera a la empresa y a los trabajadores de la misma. Para obtener ese tipo de impacto no basta con solamente capacitarse y especializarse en el área mencionada, también se debe plasmar ese conocimiento en los canales existentes de la empresa, y crear nuevos donde no existan.

Para abordar el objetivo propuesto de una manera palpable, es que se decidieron tres objetivos específicos. El primer objetivo fue la generación de documentación y guías de estilo sobre las tecnologías involucradas, el segundo fue la reanudación del desarrollo del generador de proyectos móviles de la empresa, y el tercero fue la reestructuración de un proyecto móvil que llevaba varios años en Platanus.

El primer objetivo era necesario para poder renovar el conocimiento en el área, y canalizarlo hacia el resto de los desarrolladores de la empresa. La mayor parte de las tecnologías mencionadas en desarrollo móvil no eran conocidas en profundidad por los integrantes de la empresa, por lo que hubo un fuerte componente de investigación para capacitarse en el área. Ejemplos de este proceso son tecnologías cómo Redux, EAS (2.5 y 2.6.1.1)y *tailwind-rn* (ver 3.3.1). Este objetivo permitió especializarse en el área al mismo tiempo que se transmitía conocimiento.

El segundo objetivo fue la reanudación del generador de proyectos móviles, con el propósito de poder automatizar ciertas configuraciones horizontales a todos los proyectos de desarrollo móvil. Este objetivo permitió reducir la cantidad de configuraciones que los desarrolladores de proyectos móviles deben saber configurar, aumentando la uniformidad en los proyectos de la empresa, y disminuyendo el tiempo requerido en llegar a un MVP (ya que no se pierde tiempo en configuraciones iniciales). Este objetivo también sirvió para automatizar acuerdos establecidos en el objetivo anterior (ver 3.2 y 4.2.3).

El tercer y último objetivo consistió en reestructurar un proyecto móvil que lleva varios años en la empresa, esto se hizo ya que si se quiere renovar el área, no sólo se debe mejorar lo que está por venir, también se debe impulsar lo que ya existe. En este proyecto el foco estuvo en hacer una reingeniería sobre el proyecto para que quedara consistente con los nuevos estándares que se tomaron en el primer objetivo.

Por el lado de la documentación, los resultados obtenidos en la encuesta final hecha a desarrolladores de la empresa (ver sección 6) arrojaron que la percepción sobre la documentación existente ha mejorado notoriamente, así como también el entusiasmo sobre querer participar en este tipo de proyectos.

En el ámbito del generador de proyectos móviles, un proyecto que arribó en marzo de 2023 fue generado desde cero usando la nueva versión funcional del generador (ver sección 6.2), y se planea seguir usando este generador para los futuros proyectos móviles que lleguen a la empresa.

Sobre la reingeniería que se efectuó sobre el proyecto, este pasó de no cumplir ninguno de los estándares definidos, a dos completos y uno parcial, de un total de cuatro.

Finalmente, se valora el proceso de crear acuerdos dentro de organizaciones del mundo del desarrollo de software con el fin de mantener un estándar de calidad a lo largo de la organización. Estos acuerdos también son los cimientos sobre los que se puede escribir código que los automatice. De esta forma, las organizaciones pueden escalar, disminuyendo el tiempo de sus procesos sin dejar de garantizar el sello de calidad que las caracteriza, como es el caso de la empresa Platanus.

Bibliografía

[1] M. Keza, "Cyberpunk 2077: how 2020's biggest video game launch turned into a shambles." https://www.theguardian.com/games/2020/dec/18/cyberpunk-2077-how-2020s-biggest-video-game-launch-turned-into-a-shambles, December, 18 2020.

[2] V. Morales Lopez, "La transferencia de conocimiento en las organizaciones," *Revista Estudios Interdisciplinarios de la Organización*, Enero/Junio 2012.

[3] W.-H. Hung and W.-H. Wang, "Design principles of wiki system for knowledge transfer and sharing in organizational education and training," *Sustainability*, vol. 12, no. 17, 2020.

[4] A. Syeed, S. Bhat, and D. Kaur, "Study of mobile app development industry," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, pp. 154–170, 12 2021.

[5] Perfecto, "Perfecto test coverage index magazine, edition 11.2." PDF document, 2020. Página 12.

[6] Statista, "Mobile operating systems' market share worldwide from 1st quarter 2009 to 2nd quarter 2023." Statista, 2023.

[7] E.-R. Latifa and E. K. M. Ahmed, "Android: Deep look into dalvik vm," in *2015 5th World Congress on Information and Communication Technologies (WICT)*, pp. 35–40, 2015.

[8] S. Bose, "A comparative study: Java vs kotlin programming in android application development," *International Journal of Advanced Research in Computer Science*, vol. 9, pp. 41–45, 06 2018.

[9] A. Developers, "Android's commitment to kotlin." https://android-developers.googleblog.com/2019/12/androids-commitment-to-kotlin.html, December 2019.

[10] C. González García, J. Espada, B. Pelayo García-Bustelo, and J. Cueva Lovelle, "Swift vs. objective-c: A new programming language," *International Journal of Artificial Intelligence and Interactive Multimedia*, vol. 3, pp. 74–81, 01 2015.

[11] "Stack overflow developer survey 2022."

[12] "React native y flutter, google trends." https://trends.google.com/trends/explore?date=2018-01-02%202022-01-02&geo=US&q=/g/11h03gfxy9,/g/11f03_rzbg.

[13] B. V. S. Indla and Y. Puranik, "Review on react js," *International Journal of Trend in Scientific Research and Development (ijtsrd)*, vol. 5, no. 4, pp. 1137–1139, 2021.

[14] E. Li, "Github ranking." https://github.com/EvanLi/Github-Ranking, 2023. Repositorio de GitHub.

[15] React Documentation, "Introducing JSX." https://legacy.reactjs.org/docs/introducing-jsx.html, Legacy.

[16] React Documentation, "Components and props." https://legacy.reactjs.org/docs/components-and-props.html, Legacy.

[17] J. Cook, "How the react native bridge works and how it will change in the near future." https://dev.to/wjimmycook/how-the-react-native-bridge-works-and-how-it-will-change-in-the-near-future-4ekc, August, 9 2020.

[18] S. De Lima, "A reduced explanation of redux. (reducers, actions, and store)." https://levelup.gitconnected.com/a-reduced-explanation-of-redux-reducers-actions-and-store-9b7819c5d064, January, 30 2020.

[19] Expo, "Using sentry." https://docs.expo.dev/guides/using-sentry/, 2022. A guide on installing and configuring Sentry for crash reporting.

[20] J. Bogner and M. Merkel, "To type or not to type? a systematic comparison of the software quality of javascript and typescript applications on github," MSR '22, (New York, NY, USA), p. 658–669, Association for Computing Machinery, 2022.

[21] E. Bacon, "The new expo cli." https://blog.expo.dev/the-new-expo-cli-f4250d8e3421, August, 11 2022.

[22] W. Maj, "Enzyme is dead. now what?." https://dev.to/wojtekmaj/enzyme-is-dead-now-what-ekl, December, 20 2021.

Anexos

Anexo A. La Guía

En esta sección se encuentran traducciones de guías que fueron escritas pensando en el resto de los desarrolladores de Platanus cómo público objetivo. Debido a esto, poseen jergas informales en ciertas partes de su contenido. Las versiones originales se encuentran disponibles en la página web de La Guía, https://la-guia.platan.us/.

A.1. Uso de Tailwind en React Native

Para usar Tailwind en los proyectos móviles, utilizamos la biblioteca `tailwind-rn`[1].

Si estás utilizando una versión anterior a la 4 de `tailwind-rn`, hay varias diferencias importantes. La mayoría de las cosas en esta página no te serán útiles. Se recomienda realizar la migración utilizando esta guía.

Se deben seguir ciertos pasos previos para poder utilizar Tailwind en los componentes de React Native. Si el proyecto se inició con `cavendish`[2], esto ya está configurado y puedes omitir la sección de setup.

Setup

Debes ejecutar el comando `npx setup-tailwind-rn`, que instalará la mayoría de las dependencias. Luego, el mensaje final del script contendrá los siguientes pasos a seguir. Estos pasos también están descritos en el `README` del repositorio de GitHub de la biblioteca[3].

Uso

Dado que Tailwind no es compatible nativamente con React Native, esta biblioteca hace que funcione creando un archivo `tailwind.json` en la raíz del proyecto. El archivo guarda las clases y las traduce al sistema de stylesheets de React Native. El archivo puede verse en el código A.1

[1] https://github.com/vadimdemedes/tailwind-rn
[2] https://github.com/platanus/cavendish
[3] https://github.com/vadimdemedes/tailwind-rn#install

Código A.1: Archivo *tailwind.json* que traduce las clases de Tailwind a estilos de React Native.

```
  /*
 ...
*/

"items-center": {
    "style": {
      "alignItems": "center"
    }
  },
  "rounded": {
    "style": {
      "borderTopLeftRadius": 4,
      "borderTopRightRadius": 4,
      "borderBottomRightRadius": 4,
      "borderBottomLeftRadius": 4
    }
  },
  "rounded-full": {
    "style": {
      "borderTopLeftRadius": 9999,
      "borderTopRightRadius": 9999,
      "borderBottomRightRadius": 9999,
      "borderBottomLeftRadius": 9999
    }
  },
  "bg-gray-900": {
    "style": {
      "--tw-bg-opacity": 1,
      "backgroundColor": "rgb(17 24 39 / var(--tw-bg-opacity))"
    }
  },
/*
 ...
*/
```

Este archivo no incluye todas las clases existentes de Tailwind. De hecho, cuando se inicia el proyecto, este archivo estará vacío. Si intentas usar una clase de Tailwind que no esté en este archivo, no funcionará.

¿Cómo se actualiza este archivo? Hay dos formas:

1. Cuando desees utilizar una nueva clase de Tailwind que no hayas usado antes, puedes ejecutar el comando:

   ```
   yarn build:tailwind
   ```

   Esto actualizará el archivo `tailwind.json` con todas las clases que se utilizan en el proyecto. Este proceso debe repetirse cada vez que se desee agregar una nueva clase, lo cual

puede resultar tedioso, especialmente al inicio de un proyecto.

2. Puedes utilizar el siguiente comando:

   ```
   yarn dev:tailwind
   ```

   Esto iniciará un servidor que escuchará los cambios en el proyecto y actualizará automáticamente el archivo `tailwind.json` cada vez que se agregue una nueva clase de Tailwind. Esta opción es mucho más conveniente que la primera y es la que recomendamos utilizar.

En los componentes, debes importar el *hook* `useTailwind` e invocarlo para obtener la variable `tailwind`. Finalmente, en la propiedad `style` de los componentes, puedes utilizar esta variable para aplicar las clases de Tailwind en React Native.

Código A.2: Componente de React Native estilizado usando la librería *tailwind-rn*

```
import { View, Text } from 'react-native';
import { useTailwind } from 'tailwind-rn';

export default function MyTailwindComponent() {
 const tailwind = useTailwind();

 return (
  <View style={tailwind('bg-gray-900 flex-1 items-center justify-center')}>
   <View style={tailwind('bg-white h-40 w-40 rounded-full items-center justify-center')}>
    <Text style={tailwind('text-lg font-black')}>
     Tailwind in React Native!
    </Text>
   </View>
  </View>
 );
}
```

Figura A.1: UI generada a partir del código A.2.

TLDR

- Inicia el servidor de Tailwind con `yarn dev:tailwind`.
- Importa el *hook* `useTailwind` de la biblioteca `tailwind-rn`.
- `const tailwind = useTailwind()` dentro del componente que estás escribiendo.
- En la prop `style` de los componentes, utiliza `tailwind('clases de Tailwind aquí')`.

A.2. Crear y Conectar una slice en Redux

¿Qué es una *slice*?

Una *slice* contiene una porción de data del estado de la aplicación y métodos para procesar, cambiar y acceder a esa data.

Las *slices* de la app van en la carpeta `src/store/slices` con el nombre del modelo de datos que quieren representar, en *camelCase*. Por ejemplo, la *slice* que maneja los datos del

usuario debería llamarse `user.ts` (o `currentUser.ts`) y su path de importación debería ser `src/store/slices/user` (o `@/store/slices/user` usando el alias `@`).

Hay 3 cosas importantes que proporciona una *slice*:

- El estado inicial de la data de ese modelo.
- Cómo debería modificarse el estado de la data (*actions* y *reducers*).
- *Shortcuts* personalizables para acceder a la data (*selectors*).

Crear una *slice*

Importamos la función `createSlice` de la librería `redux-toolkit`, que simplifica muchísimo todo el proceso de definir la data y escribir los *reducers*. Primero definimos un estado inicial de la *slice* y le damos un nombre. Vamos a crear una *slice* para un contador, que va a partir con cero como valor inicial:

```
import { createSlice } from '@reduxjs/toolkit';

const initialState = {
  value: 0,
};

const counterSlice = createSlice({
  name: 'counter',
  initialState,
  // otras cosas...
});
```

Ahora debemos especificar en la *slice* cómo va a cambiar el estado de la data (en este caso, del valor del contador). La función `createSlice` viene con un campo *reducers* donde podemos especificar justamente eso. En teoría, para cambiar la data necesitamos definir *actions* y *reducers*, pero la función `createSlice` nos permite preocuparnos solamente de escribir las funciones *reducers* y a partir de eso genera las *actions* correspondientes. Por ejemplo, si queremos tener un *reducer* que incremente el valor del contador en uno, debemos definirlo de la siguiente manera:

```
const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    increment: (state) => {
      state.value += 1;
    },
  }
});
```

Además, estas funciones pueden recibir información adicional de lo que deben actualizar a través del campo payload de una action. Por ejemplo, imaginemos que ahora queremos crear un reducer incrementBy, que le suma al contador un número variable:

```
import { createSlice, type PayloadAction } from '@reduxjs/toolkit';

// ...

const counterSlice = createSlice({
  name: 'counter',
  initialState,
   reducers: {
     // ...
     incrementBy: (state, action: PayloadAction<number>) => {
       state.value += action.payload;
     },
   }
});
```

Agregando un reducer para disminuir el valor del contador por un número arbitrario, y también otro reducer para devolver a cero el contador, la versión final del archivo src/store/slices/counter.ts queda así:

```
import { createSlice, type PayloadAction } from '@reduxjs/toolkit';

const initialState = {
  value: 0,
};

export const counterSlice = createSlice({
  name: 'counter',
  initialState,
  reducers: {
    reset(state) {
      state.value = 0;
    },
    incrementBy(state, action: PayloadAction<number>) {
      state.value += action.payload;
    },
    decrementBy(state, action: PayloadAction<number>) {
      state.value -= action.payload;
    },
  },
});
```

Ya tenemos una slice que tiene data inicial y provee formas de manipularla a través de los reducers que definimos, pero todavía no está conectada a la store de la app. Para eso debemos agregar los reducers que definimos al archivo src/store/reducers.ts en donde se encuentra el reducer de toda la aplicación (el appReducer). El archivo src/store/reducers.ts se debería ver parecido a algo como esto:

```
import { combineReducers } from '@reduxjs/toolkit';

import { userSlice } from '@/store/slices/user';
import { postsSlice } from '@/store/slices/posts';

const appReducer = combineReducers({
  user: userSlice.reducer,
  posts: postsSlice.reducer,
});

export default appReducer;
```

Aquí se encuentra el *appReducer*, que cómo bien dice su nombre, es el reducer que maneja los cambios de estado de toda nuestra aplicación. Este reducer se obtiene usando la función *combineReducers* de *redux-toolkit*. Para agregar el slice que creamos, tenemos que dentro del objeto que se le pasa a combineReducers agregar el campo counter y que su valor sea el reducer de nuestra slice.

OJO! el campo que se agrega a combineReducers tiene que coincidir con el nombre que se le dió a la slice dentro de *createSlice*.
Finalmente el archivo con el appReducer queda así:

```
import { combineReducers } from '@reduxjs/toolkit';

import { counterSlice } from '@/store/slices/counter';
import { userSlice } from '@/store/slices/user';
import { postsSlice } from '@/store/slices/posts';

const appReducer = combineReducers({
  counter: counterSlice.reducer,
  user: userSlice.reducer,
  posts: postsSlice.reducer,
});

export default appReducer;
```

Listo! Ahora tenemos acceso a la data, las acciones y los reducers que definimos en nuestro archivo original, y ya podemos llamarlos dentro de nuestros componentes de React.

Anexo B. Cavendish

B.1. Comando *AddEASConfiguration*

Código B.1: Clase Cavendish::Commands::AddEASConfiguration.

```
module Cavendish
  module Commands
    class AddEASConfiguration < Cavendish::Commands::Base
      def perform
        install_eas_cli
        register_and_create_organization_prompt
        login_to_expo
        configure_project_owner
        set_basic_project_config
        inject_advanced_eas_config
      end

      private

      def install_eas_cli
        run_in_project('npm install -g eas-cli')
      end

      def register_and_create_organization_prompt
        ask(REGISTER_MESSAGE)
        ask(CREATE_ORGANIZATION_MESSAGE)
      end

      def login_to_expo
        run_in_project('eas login')
      end

      def configure_project_owner
        run_in_project('eas project:init')
      end

      def set_basic_project_config
        run_in_project('eas build:configure')
      end

      def inject_advanced_eas_config
        rename_preview_to_staging
        inject_to_json_file('eas.json', EAS_ADVANCED_CONFIG)
      end

      def rename_preview_to_staging
        rename_key_in_json('eas.json', 'build.preview', 'staging')
      end
    end
  end
end
```