Análisis de frameworks para desarrollo de aplicaciones móviles y web

---

Nicolás Simmonds Samper

Enero, 2019
Versión: 1.0

Bogotá, Colombia

Departamento de Ingeniería de Sistemas y Computación
Facultad de Ingeniería
The Software Design Lab

Tesis de pregrado

Análisis de frameworks para desarrollo de aplicaciones móviles y web

Nicolás Simmonds Samper

Advisor Ph.D. Mario Linares Vásquez
Systems and Computing Engineering Department
Universidad de los Andes, Bogotá, Colombia

Enero de 2019

Abstract

Mobile app development has been growing exponentially in the past few years, and with it, the technologies in order to develop these apps. Usually in order to develop an app, it is required a team for Android development and other for iOS. The problem is that it means high costs for the companies. For this reason, developers have created tools called frameworks, that let write an app in one single language, that works on both operating systems. Nevertheless, there are still limitation in what these applications can do, when compared to a native one. Thus, the goal of this project is to develop the exact same app using three different frameworks (React-Native, ionic y Flutter), and analyze the results considering aspects like development time and performance.
It is expected at the end of the project, to have a detailed analysis of each one of the frameworks and use it to determine which one of them is the one that simulates a native app the better.

El desarrollo de aplicaciones móviles ha ido creciendo sustancialmente con los aaños, y así mismo las tecnologías para crear estas aplicaciones. Normalmente para desarrollar una nueva aplicación, se necesita un equipo que desarrolle para Android y otro para iOS. El problema es que esto implica costos muy altos para las empresas. Por esta razón, se han creado herramientas llamadas frameworks que sirven para desarrollar una aplicación en un solo lenguaje y que sirva adecuadamente en ambos sistemas operativos. Sin embargo, todavía existen limitaciones con respecto a lo que puede hacer una aplicación nativa. El objetivo de este proyecto es entonces, crear la misma aplicación usando tres frameworks de desarrollo diferentes (React-Native, ionic y Flutter) y analizar los resultados obtenidos teniendo en cuenta aspectos como el tiempo de desarrollo o el desempeño.
Se espera que al final del proyecto, se pueda contar con un análisis detallado de cada uno de los frameworks usados y usar esto para determinar cuál es el que mejor se acerca a una aplicación desarrollada nativamente.

Tabla de contenido

1. Introducción

1.1. ¿Qué es un framework?

Un frameworks de desarrollo es una herramienta que pretende facilitar la creación de aplicaciones web o móviles. Esto se logra de diferentes formas. Al usar un framework, se tiene como base un esqueleto que el desarrollador debe seguir. Este esqueleto incluye, entre otras cosas, archivos de configuración básica que siempre son iguales, y componentes previamente creados por los desarrolladores de la herramienta con el fin de que sean reutilizados. En otras palabras, es una receta básica de lo que debe tener un proyecto y que incluye ingredientes de todo tipo que el desarrollador puede elegir usar o no, dependiendo de sus necesidades.

Los frameworks más comunes y usados son los de desarrollo web, los cuales empezaron a tener acogida a principios del 2010 y han venid mejorando con el tiempo. Entre los más usados están React, desarrollado por Facebook y Angular, desarrollado por Google. En términos de desarrollo móvil, estas herramientas son relativamente nuevas pues han tenido que investigar cómo lograr que a partir de un lenguaje que no es nativo para un dispositivo se pueda lograr compilar una aplicación que funcione correctamente tanto para iOS como para Android. Entre los que se van a analizar en este proyecto, el primero de todos fue ionic, que es un framework que se puede usar tanto para web como para móvil, y tiene como base otros frameworks, Angular y Cordova.
React-Native surgió después como la alternativa de Facebook y que pretendía simular aún mejor una aplicación nativa. Como su nombre lo indica, es basado en React.
El más reciente de los tres es Flutter, creado por Google y es muy diferente a los otros pues compila casi 100% el código escrito a nativo y no tiene ninguna base. Se creó desde cero.

1.2. Ionic

Como se mencionó anteriormente, ionic es un framework que utiliza Cordova para encargarse de la parte móvil y Angular para encargarse de la parte web. Aunque la mayoría del código se escribe en TypeScript y HTML, que es la base de Angular, es Cordova el que se encarga de crear el marco de ejecución para las aplicaciones en los dispositivos. Una típica aplicación de ionic tiene 4 archivos por componente que se cree, o 5 si se crea una página desligada de la principal. Estos archivos son el HTML donde se hace el markup de la aplicación, un archivo scss que le da los estilos, un archivo spec, que es autogenerado y por lo general no se modifica, y un archivo component, que es donde se trabaja toda la lógica.
Si en vez de un componente se crea una página, se agrega un archivo adiciona llamado modulo donde se declaran todos los paquetes que provisiona ionic que se van a utilizar en esa página.

1.3. React-Native

React-Native es un framework desarrollado por Facebook y tiene como fundamentos el framework de desarrollo web React. El equipo que desarrollo React Native decidió basarse en React por la facilidad y buen acogimiento que ha tenido entre la comunidad de desarrolladores. El desarrollo de una aplicación de React-Native y React es prácticamente el mismo. Hay dos cambios importantes. Primero, no se usa el lenguaje de React jsx como tal, sino una adaptación de este a lo que necesita típicamente un dispositivo y, además. Es necesario lidiar con particularidades típicas de un dispositivo como permisos que se le soliciten al usuario.

En términos de arquitectura, no funciona como ionic que, al crear una aplicación de React-Native. venga estructurada como debe ser. Sin embargo, es recomendable seguir un patrón MVC, donde se separe por un lado la lógica de la aplicación, el diseño, y todo lo que tenga que ver con conexiones externas.

1.4. Flutter

Flutter es el framework que cambio todo. Si bien en los anteriores era fácil para un desarrollador que tenía un antecedente web, migrar a móvil por medio de estos framework, la forma en la que se escribe una aplicación de Flutter es muy diferente.

Los ingenieros de Google decidieron crear una herramienta que funciona a base de "Widgets", los cuales son componentes gráficos que vienen incluidos con Flutter, y son escogidos basado en lo que típicamente tiene una aplicación móvil. Por ejemplo, si se necesita un botón flotante, hay un widget para eso en Flutter. No hay que escribirlo y posicionarlo como se hace normalmente.

La arquitectura de Flutter también es algo nuevo, pues ellos se inventaron una llamada BloC, que traduce por sus siglas en ingles Business Logic Component y consiste en tener dos capas, una que es puramente el UI de la aplicación y otra que se encarga de la lógica y se comunican por medio de streams. Es decir, suponiendo que existe un botón que realiza una acción X, en vez de realizar esta acción y cambiar la interfaz directamente, este manda un streams a la lógica y este devuelve una acción que debe ejecutar la interfaz si es necesario.

2. Diseño de la aplicación

2.1. Concepto

La aplicación que se deicidio hacer es una típica aplicación de To-Do, donde básicamente se pueden agregar tareas y tienen dos estados, completada o no completada.

Adicional a esto, tiene otras vistas de las que se va a hablar más adelante en la sección de funcionalidades.

El motivo por el que se decidió hacer esta aplicación es porque así parezca básica, puede reunir muchas funcionalidades (o no) para poder sacar conclusiones en comparación al alcance de una aplicación nativa.

2.2. Interfaz gráfica

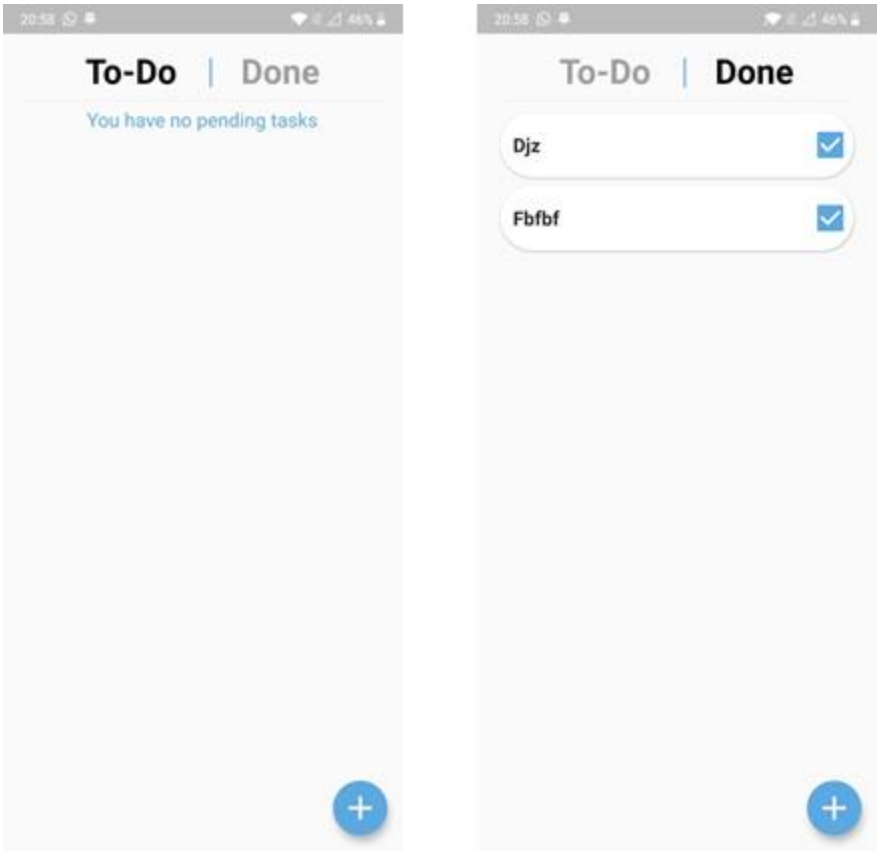


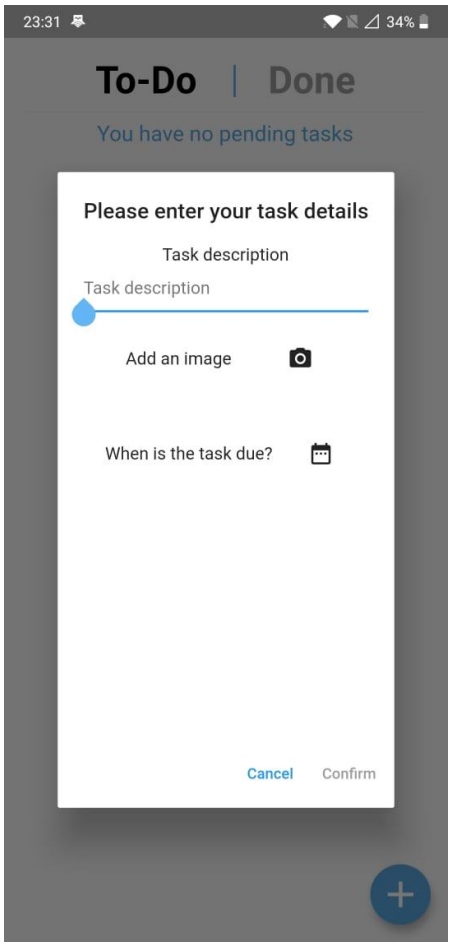
Ilustración 1 Vista principal aplicación

Las dos imágenes anteriores son las principales vistas de la aplicación, en donde se muestran los dos estados en los que puede estar, 'To-DO' son las tareas que se agregaron, pero aún no se han completado, y "Done" las tareas que ya se completaron. Además, el botón flotante sirve para agregar una tarea nueva a la aplicación.

2.3. Funcionalidades

Las funcionalidades que ofrece la aplicación son las siguientes:

1. Agregar una tarea
   - Agregar descripción tarea
   - Agregar imagen, cámara o galería
   - Agregar fecha
   - Agregar ubicación (Se agrega automáticamente con la ubicación actual)
2. Eliminar una tarea
3. Cambiar estado de “To-Do’ A “Done” y viceversa
4. Intercambiar posición de las tareas
5. Ver información de la tarea
6. Notificaciones push



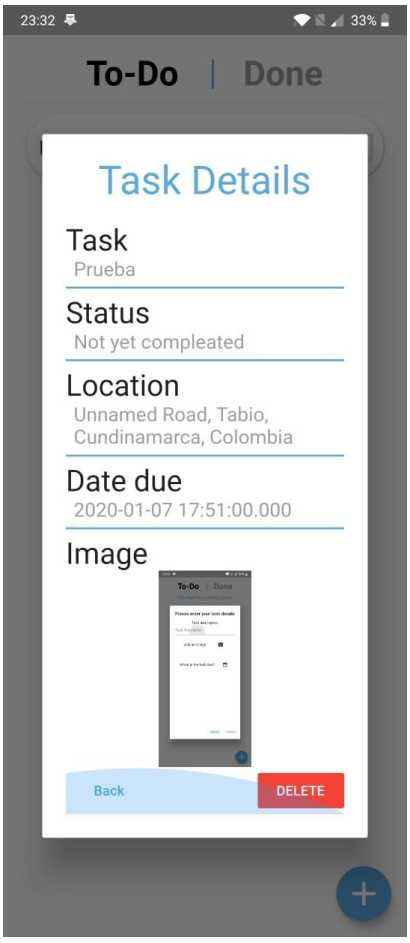


Ilustración 2 Agregar una nueva tarea y ver información de la tarea

La imagen de la izquierda representa el modal que aparece cuando el usuario quiere agregar una tarea nueva. Como se ve, se puede agregar una descripción, una imagen ya sea de la cámara o de la galería de fotos, y una fecha.
Por otro lado, en la imagen de la derecha está el modal que se muestra cuando el usuario hace clic sobre una tarea ya existente y muestra la información de esta.
Esta información es: la descripción, el estatus de la tarea, la cual puede ser completada o no completada, la ubicación de la persona cuando creó la tarea, la fecha límite de la tarea, la imagen que se escogió y unas acciones, que pueden ser ir hacia atrás o eliminar la tarea.
Además, como parte de las funcionalidades hay un sistema de notificaciones push, que se va a explicar en detalle más adelante, pero cuyo funcionamiento se puede ver en la siguiente imagen.

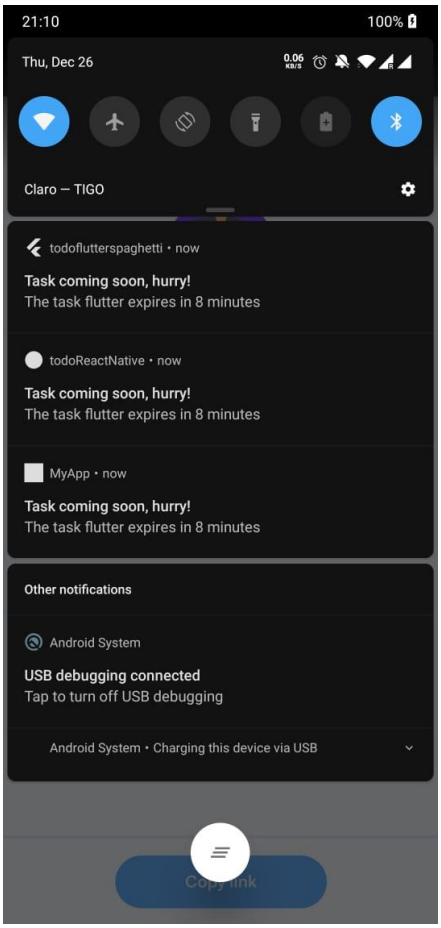


Ilustración 3 Notificaciones push

Como se aprecia en la imagen, cuando el sistema detecta que una tarea esta próxima a vencer, se manda una notificación al usuario recordándole que la complete. En caso de que se haya pasado la fecha límite, dicha notificación muestra que la tarea ha vencido.

3. Descripción Backend

Para esta aplicación se decidió hacer un solo backend que las tres aplicaciones pudieran compartir. Esto se decidió así ya que lo importante era analizar el cada framework como tal, y no una aplicación robusta. Además, en el proceso de desarrollo se evidencio que esta decisión ayudo a ver la reactividad de las aplicaciones al agregar una tarea a la vez, y ver si automáticamente la cargaba en otra o era necesario volver a lanzarla.

Dicho esto, el backend esta creado con Node.JS, y MongoDB. En términos generales permite hacer un CRUD sobre los datos guardados. A continuación, se presentan los componentes.

```
const dataSchema = new mongoose.Schema(
  {
    task: String,
    active: Boolean,
    picture: String,
    date: String,
    location: String,
    position: Number,
  },
  { timestamps: true },
);
```

Ilustración 4 Esquema de la base de datos

En la imagen anterior se presenta el esquema de datos que se definió. Como es de esperarse incluyen todos los datos que la aplicación permite guardar además de un timestamp que genera mongo automáticamente para saber la última fecha de modificación.

```
router.get('/data', dataController.index);
router.get('/data/:id', dataController.show);
router.post('/data', dataController.create);
router.put('/data/:id', dataController.update);
router.delete('/data/:id', dataController.delete);
```

Ilustración 5 Operaciones posibles sobre la base de datos

Esta imagen representa las definiciones de las rutas de las operaciones CRUD que se pueden hacer sobre la base de datos. Como se puede ver, se puede hacer un get de todas las tareas, un get por id, un post para crear una tarea, un put para modificar una tarea dado un id y un delete para eliminar una tarea dado un id.

Para ver el código completo del backend se puede ver el enlace (9.4) de GitHub donde esta toda la aplicación.

4. Descripción Firebase API

4.1. Cloud Functions

El API Cloud Functions de Firebase, es una herramienta muy poderosa que permite ejecutar una función o funciones, generalmente escritas en JavaScript, cada cierto tiempo determinado. Hay dos formas de crearlas. La primera es generarla directamente desde la consola de Google donde uno escribe la función que se quiere ejecutar y el intervalo de tiempo que se desee. La segunda, y es la que se implementó en este proyecto, es por medio del CLI de Firebase. En este caso, se instala este cliente en la máquina que el usuario desee y se escribe un programa que se quiere ejecutar cada cierto tiempo. La ventaja de esta alternativa es que permite generar un set de instrucciones más robusto que por medio de la consola de Google.

4.2. Cloud Messaging

El API Cloud Messaging es otra herramienta que proporciona Firebase la cual permite mandar notificaciones push a todo tipo de aplicaciones y segmentarlas por diferentes reglas predefinidas para apuntar a cierto tipo de usuarios. En este caso, se creó un servicio de Cloud Messaging por cada aplicación y este es el segmento al que el API apunta.

4.3. Integración

Lo que se hizo en el proyecto fue integrar estas dos APIS para poder generar un servicio en el que cada cierto tiempo, si se cumplen ciertas condiciones, se mande una notificación push al dispositivo. Como demostración, se muestra a continuación un fragmento de código de este servicio hecho para la aplicación de Flutter.

```
exports.taskDue = functions.pubsub.schedule("*/5 * * * *").onRun(() => {
  gettoken()
    .then(token => {
      const headers = {
        Authorization: "Bearer " + token
      };

```

Ilustración 6 Declaración Cloud Functions

En la imagen anterior se puede apreciar la creación de la función cloud. En este caso, se ejecuta cada 15 minutos y este tiempo se define en formato CRON. Luego se pide el token y se crea el header que se va a mandar posteriormente en la notificación push.

```
const dataAfter = {
  message: {
    topic: "push",
    notification: {
      title: "Task Due Date Passed",
      body: `The task ${task} is past it's due date`
    }
  }
};
axios.post(
  "https://fcm.googleapis.com/v1/projects/flutterspaghetti-cd14f/messages:send",
  dataAfter,
  {
    headers: headers
  }
```

Ilustración 7 Mandar notificación push

En el cuerpo de la función se hace un get de toda la base de datos y se calcula la fecha guardada y la fecha actual. En caso de cumplirse que falten 15 minutos o menos, o se haya vencido la tarea, se procede a enviar un mensaje push al dispositivo. Este proceso es el que se ejemplifica en la ilustración 7. Se crea el cuerpo del mensaje como un objecto y luego se hace una petición POST al API de Cloud Messaging, apuntando a la aplicación que se desee, que en este caso es la de Flutter.

Para ver a fondo cómo funciona el servicio, se puede ver alguna de las implementaciones de las tres aplicaciones descritas en los enlaces (9.5, 9.6, 9.7) de GitHub.

5. Análisis por framework

5.1. Flutter

5.1.1. Estructura del proyecto

En la imagen de la derecha se puede ver la estructura general del proyecto de Flutter. Lo importante a analizar es lo que se encuentra dentro de la carpeta “lib” pues es donde el desarrollador escribe todo el código de la aplicación. El resto es lo que Flutter genera automáticamente y que anteriormente se mencionó como el esqueleto o “receta”.

Dentro de la carpeta lib hay dos archivos: main.dart y listComponent.dart. El primero es donde esta lo principal de la aplicación y el segundo archivo es widget personalizado de una sola tarea. Este widget se renderiza por cada tarea que se encuentre en la base de datos y se pasan por parámetro la información de la tarea.

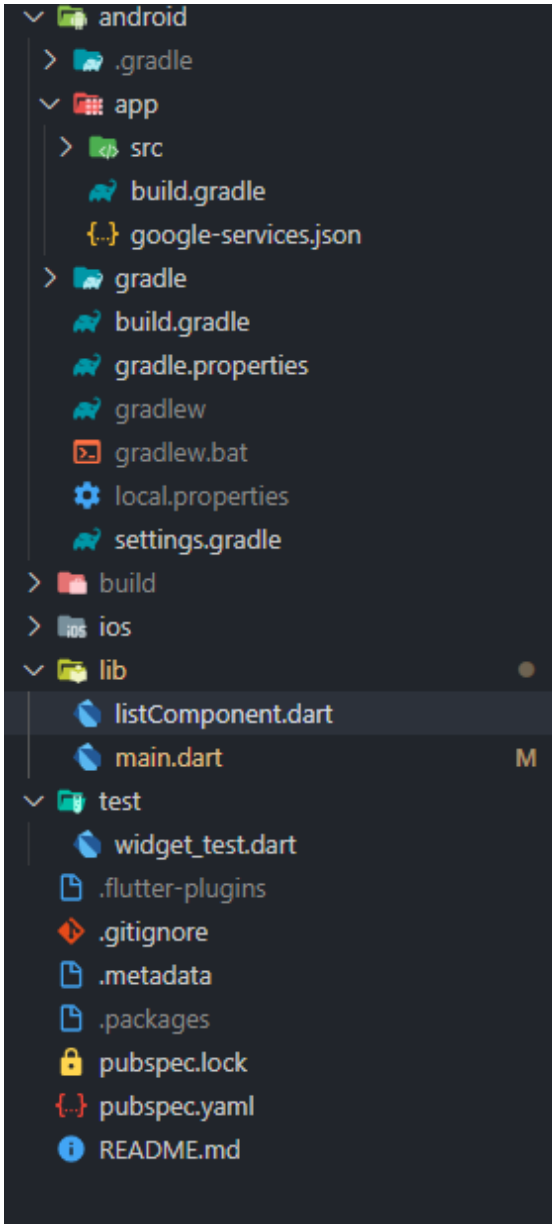


Ilustración 8 Estructura proyecto Flutter

5.1.2. Librerías externas usadas

Por defecto Flutter trae muchas librerías que el desarrollador puede usar, pero en algunos casos es necesario instalar algunas externas para lograr ciertas funcionalidades. A continuación, se describen este tipo de librerías.

```
geolocator: ^5.1.4+1
geocoder: ^0.2.1
image_picker: ^0.6.0+9
firebase_core:
firebase_messaging:
flutter_datetime_picker: ^1.2.8
```

Ilustración 9 Librerías externas Flutter

- Geolocator: Esta librería es usada para saber la ubicación actual del dispositivo, siempre y cuando haya conexión a internet. Esta ubicación se da como un arreglo [latitud, longitud].

```
_getCurrentLocation() async {
  final Geolocator geolocator = Geolocator()..forceAndroidLocationManager;

  geolocator
      .getCurrentPosition(desiredAccuracy: LocationAccuracy.best)
      .then((Position position) {
    _getLocationName(position);
  }).catchError((e) {
    throw e;
  });
}
```

Ilustración 10 Uso de Geolocator

- Geocoder: Por medio de esta librería se puede ingresar las coordenadas previamente extraídas y convertirlas a una dirección real, por ejemplo, si se ingresan las coordenadas [4.602835, -74.065039], por medio del Geocoder se obtiene la dirección Cra. 1 #18a 12, Bogotá, Cundinamarca, Colombia.

```
_getLocationName(Position pos) async {
  final coordinates = new Coordinates(pos.latitude, pos.longitude);
  var addresses =
      await Geocoder.google('API-KEY-GOOGLE')
          .findAddressesFromCoordinates(coordinates);
  var first = addresses.first;
  setState(() {
    location = "${first.addressLine}";
  });
}
```

Ilustración 11 Uso de Geocoder

- Image_picker: Esta librería permite otorgar acceso a ver archivos guardados en el dispositivo y a la cámara. Entonces, es la que permite seleccionar una imagen guardada en el dispositivo o tomar una foto para agregarla a una tarea.

```
  picker(bool action, BuildContext context) async {
    File img = action
        ? await ImagePicker.pickImage(source: ImageSource.camera)
        : await ImagePicker.pickImage(source: ImageSource.gallery);
    if (img != null) {
      Navigator.of(context).pop();
      setState(() {
        _image = img.path;
      });
    }
  }
}
```

Ilustración 12 Uso de image_picker

- Firebase_core y Firebase_messaging: Estas dos librerías permiten hacer la conexión a los servicios de Firebase y en este caso, recibir notificaciones push.

```
  firebaseMessaging.requestNotificationPermissions(
      const IosNotificationSettings(sound: true, alert: true, badge: true));
  firebaseMessaging.onIosSettingsRegistered
      .listen((IosNotificationSettings settings) {
    debugPrint('ios');
  });

  firebaseMessaging.getToken().then((onValue) {
    debugPrint(onValue);
  });

  firebaseMessaging.subscribeToTopic('push');
}
```

Ilustración 13 Uso de Firebase_core y Firebase_messaging

- Flutter_datetime_picker: Por medio de esta librería se puede declarar un widget que da la opción al usuario de seleccionar una fecha y hora.

```
_selectDate() {
  DatePicker.showDateTimePicker(
    this.context,
    showTitleActions: true,
    minTime: DateTime.now(),
    onChanged: (e) {
      setState(() {
        date = e.toString();
      });
    },
    onConfirm: (e) {
      setState(() {
        date = e.toString();
      });
    },
    currentTime: DateTime.now(),
  );
}
```

Ilustración 14 uso Flutter_datetime_picker

5.1.3. Análisis

5.1.3.1 Facilidad de desarrollo

Desarrollar una aplicación en Flutter es relativamente sencillo. La forma en la que se crea la interfaz gráfica es intuitiva, pues está basado en lo que el equipo de desarrollo de Google llama árbol de widgets. Esto significa que se van anidando widgets para ir construyendo un elemento. Por ejemplo, si se quiere construir un contenedor que tenga una imagen y un botón, primero se declara un widget Container, y en los hijos de ese container se declara al mismo nivel de anidación el widget Text y FlatButton. Cada uno de estos widgets tiene un sinfín de opciones de personalización que van desde estilos hasta el tipo de comportamiento que se quiere.

A partir de esto, se pueden crear también widgets personalizados, que es el caso del ListElement.dart. Este es un widget que se creó que tiene por dentro muchos otros widgets, pero se puede reutilizar las veces que quiera, lo que aumenta significativamente la reutilización de código

A continuación, se presenta un fragmento de código que muestra un árbol de widgets.

```
class ListElementState extends State<ListElement> with WidgetsBindingObserver {
  @override
  Widget build(BuildContext ctx) {
    bool _checked = widget.active;
    return InkWell(
      onTap: () => widget.displayInfo(this.widget),
      child: Container(
        padding: EdgeInsets.symmetric(horizontal: 20),
        child: SizedBox(
          height: 75,
          child: Card(
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(100.0),
            ), // RoundedRectangleBorder
            elevation: 2,
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
```

Ilustración 15 Ejemplo Árbol de widgets

Se ve como se inicia declarando un widget Inkwell que da la funcionalidad de reaccionar a clics del usuario, y adentro de ese un Container, y así sucesivamente.

Un problema que se encontró con este sistema, que no es un problema realmente si se estructura bien el proyecto, es que se pueden crear arboles excesivamente grandes que pueden resultar difíciles de leer incluso para el desarrollador que lo creo. Es por esto por lo que es altamente recomendable crear widgets personalizados para mitigar este problema.

5.1.3.2 Desempeño

Para el desempeño se van a analizar dos métricas, uso de CPU y uso de memoria (RAM).

En cuanto al uso de CPU, se puede ver que es bajo, pues el promedio es de 1.3% y en algunos momentos alcanzo hasta 2.3%.

Ilustración 16 Uso de CPU Flutter

En cuanto a la memoria usada se vio que el promedio usado es de 75MB con picos de hasta 85MB. Teniendo en cuenta que es un framework de desarrollo no nativo es un valor bastante bueno, sin embargo, es mayor al que se consigue normalmente en una aplicación nativa.

Ilustración 17 Uso de RAM Flutter

5.2. React-Native

5.2.1. Estructura del proyecto

La estructura del proyecto de React-Native es la típica estructura de un proyecto de React. Se tienen 4 principales carpetas. Componentes, donde hay elementos que se reutilizan varias veces en el código. Containers, que archivos que sirven de contenedores, pero no representan las vistas principales de la aplicación, en este caso, dichos contenedores son el modal donde se muestra la información de una tarea existente y el modal para agregar una tarea nueva.

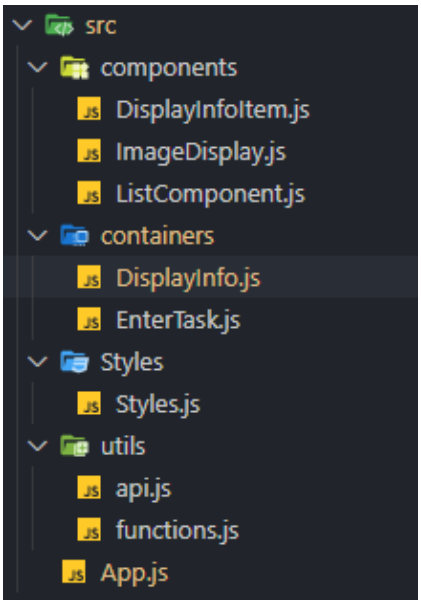


Ilustración 18 Estructura proyecto React-Native

En la carpeta Styles esta el archivo de los estilos que se manejan a lo largo de la aplicación. En este caso es un archivo js por que se usan StyledComponents, pero podría ser cualquier archivo de estilos compatibles con React-Native como css o scss.

La carpeta utils aloja funciones que manejan la llógica de la aplicación. Por un lado, el archivo api.js funciona como intermediario entre la aplicación y el backend para hacer todas las peticiones y la carpeta Functions, aloja funciones de lógica general como pedir permisos.

Por último, el archivo App.js es el núcleo de la aplicación. Ahí se maneja la vista y se renderizan los containers y los componentes necesarios para que funcione correctamente la aplicación, además del manejo del estado.

5.2.2. Librerías externas usadas

Así como Flutter, React-Native incluye librerías necesarias para que corra la aplicación, así como algunas básicas que el usuario puede elegir usar o no. En el archivo package.json se pueden ver las librerías externas que se tuvieron que agregar.

```
"react-native-elements": "^1.2.7",
"react-native-geocoding": "^0.4.0",
"react-native-geolocation-service": "^3.1.0",
"react-native-image-picker": "^1.1.0",
"react-native-modal-datetime-picker": "8.x.x",
"react-native-paper": "^3.1.1"
```

Ilustración 19 Librerías externas de React-Native

- Draggable-flatlist: Esta librería sirve para agregar la funcionalidad de intercambiar elementos en una lista.

```
<DraggableFlatList
  data={tasks2}
  renderItem={renderItem}
  keyExtractor={(item) => `draggable-item-${item.key}`}
  scrollPercent={5}
  onMoveEnd={({ data }) => { setTasks2(data); reorder(data); }}
/>
```

Ilustración 20 Implementación de draggable-flatlist

Se declara el componente y se le agregan los parámetros necesarios para que la funcionalidad sirva correctamente. El ejemplo total es bastante largo por lo que no se incluye todo el código, pero se puede ver en el proyecto de React-Native en el enlace 9.2

- Elements: Esta librería se usa para agregar componentes visuales como checkbox personalizados. En este caso se uso para agregar el checkbox de cada lista, e iconos de cámara y calendario.

```
<CheckBox
  checked={item.active}
  onPress={() => onChangeEvent(item.position)}
  checkedColor={'rgba(92, 168, 224, 1)'}
  checkedIcon="check-square"
  size={35} />
```

Ilustración 21 Implementación Elements

- Geolocation-Service: Esta librería es usada para saber la ubicación actual del dispositivo, siempre y cuando haya conexión a internet. Esta ubicación se da como objecto al que se pueden acceder a muchas propiedades entre las que están la latitud y la longitud.

- Geocoding: Por medio de esta librería se puede ingresar las coordenadas previamente extraídas y convertirlas a una dirección real, por ejemplo, si se ingresan las coordenadas [4.602835, -74.065039], por medio del Geocoder se obtiene la dirección Cra. 1 #18a 12, Bogotá, Cundinamarca, Colombia.

```
Geolocation.getCurrentPosition(e => {

  Geocoder.from(e.coords.latitude, e.coords.longitude).then(res => {
    const results = res.results[0].address_components;
    results.forEach(addresses => {
      if (addresses.types.includes('route') && !dir.includes(addresses.long_name)) {
        dir.unshift(addresses.long_name);
      }
      if (addresses.types.includes('street_number') && !dir.includes(addresses.long_name)) {
        dir.push(addresses.long_name);
      }
      if (addresses.types.includes('country') && !dir.includes(addresses.long_name)) {
        dir.push(addresses.long_name);
      }
      if (addresses.types.includes('administrative_area_level_1') && !dir.includes(addresses.long_name)) {
        dir.push(addresses.long_name);
      }
      setLocation(dir);
    });
  });
```

Ilustración 22 Implementación Geolocation-services and Geocoding

En la imagen anterior se puede ver el uso de las dos ultimas librerías pues se usan en conjunto, primero se le pide a Geolocation que, de la posición actual, y luego a partir de las coordenadas se va construyendo la dirección actual por medio de la respuesta que da Geocoder.

- Image-picker: Esta librería permite otorgar acceso a ver archivos guardados en el dispositivo y a la cámara. Entonces, es la que permite seleccionar una imagen guardada en el dispositivo o tomar una foto para agregarla a una tarea.

```
const chooseImage = () => {
  requestStorage();
  const options = {
    title: 'Please choose between camera or gallery',
    storageOptions: {
      skipBackup: true,
      path: 'images',
    },
  };
  ImagePicker.showImagePicker(options, (response) => {
    if (!response.didCancel && !response.error) {
      setImage(response.path);
    }
  });
};
```

Ilustración 23 Implementación Image-picker

- Modal-datetime-picker: Por medio de esta librería se puede declarar un widget que da la opción al usuario de seleccionar una fecha y hora.

```
<DateTimePickerModal
  isVisible={isDatePickerVisible}
  mode="datetime"
  onConfirm={handleConfirm}
  onCancel={hideDatePicker}
/>
```

Ilustración 24 Implementación modal-datetime-picker

- Paper: Esta librería sirve para agregar diferentes elementos de UI inspirados en material. En este caso se uso para implementar el botón flotante para agregar una tarea.

```
<FAB
  style={styles.fab}
  icon="plus"
  color="white"
  onPress={() => setVisible(true)}
/>
```

Ilustración 25 Implementación paper

5.2.3. Análisis

5.2.3.1 Facilidad de desarrollo

Si se viene de un entorno de desarrollo web, y el desarrollador está familiarizado con React, aprender este framework es relativamente fácil. Sin embargo, partiendo del punto de que no tiene este conocimiento, es de una complicidad media. Empezando por el ciclo de vida de React, es algo a lo que los programadores no están acostumbrados y aprender puede ser un poco tedioso.

Además, al ser un lenguaje basado en JavaScript, implica todos los problemas conocidos que vienen con ese lenguaje, aunque se puede mitigar si se decide implementarlo con TypeScript. Por otro lado, crear el markup (jsx), no es algo difícil de aprender, por lo que en un principio se podría empezar construyendo interfaces estáticas e ir agregándole interacciones para que la curva de aprendizaje se haga más fácil.

5.2.3.2 Desempeño

Para el desempeño se van a analizar dos métricas, uso de CPU y uso de memoria (RAM).

En cuanto al uso de CPU, se puede observar en la siguiente imagen que es de 12% en promedio, es una carga alta y se puede deber a la forma en la que el framework compila el código y librerías a código nativo.

Ilustración 26 Uso CPU React-Native

En el caso de la RAM, el consumo de la aplicación también es alto, pues tiene un consumo promedio de 140 MB con picos de 172 MB. El problema es que React-Native no suelta memoria de la misma forma en la que la hace una aplicación nativa. Esto se puede mitigar soltando dicha memoria en el AppCycle cuando la aplicación entra en background.

Ilustración 27 Uso RAM React-Native

5.3. Ionic

5.3.1. Estructura del proyecto

La estructura del proyecto del proyecto de ionic es extensa pues, como se mencionó anteriormente, al crear componentes genera varios archivos por cada uno.

Hay dos componentes, el primero add-task es el que se encarga de agregar una nueva tarea. El rol de cada archivo se explico en el anexo 1.2.

Luego esta display-info, que es el componente donde se renderiza la información de la tarea ya creada.

List-element es otro componente y es donde se trabaja todo lo relacionado con una tarea creada.

Después sigue home, que no es un componente sino una página, y en este caso donde esta la vista principal de la aplicación.

Los otros archivos son generados por el framework, pero hay uno muy importante, que es app.module.ts. En este archivo se declaran todas las importaciones de librerías externas.

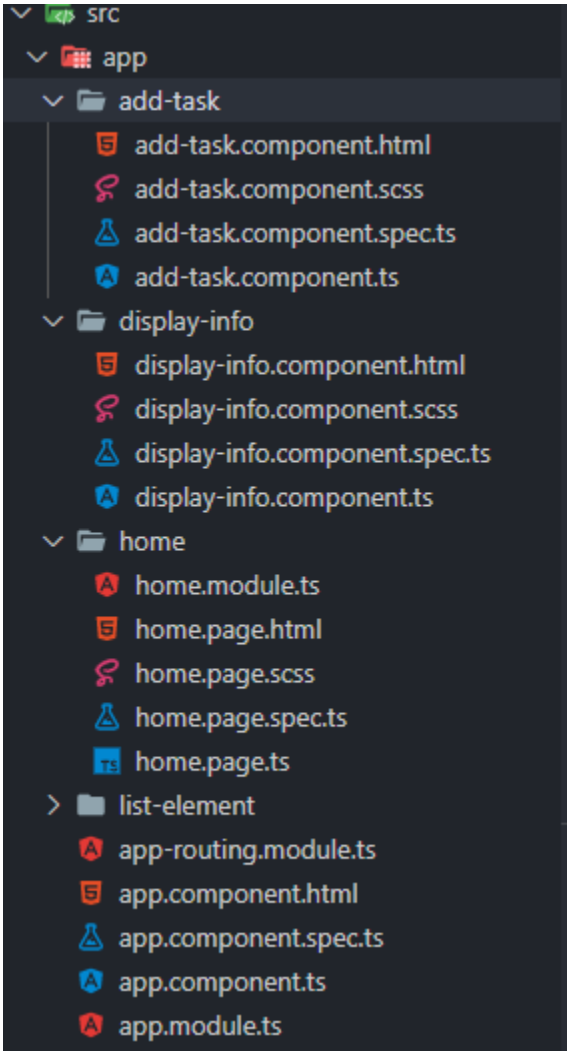


Ilustración 28 Estructura proyecto ionic

5.3.2. Librerías externas usadas

```
"@ionic-native/camera": "^5.18.0",
"@ionic-native/date-picker": "^5.18.0",
"@ionic-native/fcm": "^5.18.0",
"@ionic-native/file": "^5.18.0",
"@ionic-native/geolocation": "^5.18.0",
"@ionic-native/http": "^5.18.0",
"@ionic-native/native-geocoder": "^5.18.0",
"@ionic-native/status-bar": "^5.18.0",
```

Ilustración 29 Librerías externas ionic

- Camera: Esta librería es la que permite seleccionar una imagen guardada en el dispositivo o tomar una foto para agregarla a una tarea.

```
openDeviceCamera(option: number) {
  const options: CameraOptions = {···
  };
  this.camera.getPicture(options).then(
    (imageData: string) => {
      if (imageData.includes('?')) {
        this.image = imageData.substring(7, imageData.indexOf('?'));
      } else {
        this.image = imageData.substring(7);
      }
      this.cameraOptionsMenu = false;
    },
    err => {
      console.log(err);
    }
  );
}
```

Ilustración 30 Implementación camera ionic

- Date-picker: Esta librería permite otorgar acceso a ver archivos guardados en el dispositivo y a la cámara. Entonces, es la que permite seleccionar una imagen guardada en el dispositivo o tomar una foto para agregarla a una tarea.

```
showDatePicker() {
  this.datePicker.show({
    date: new Date(),
    mode: 'datetime',
    androidTheme: this.datePicker.ANDROID_THEMES.THEME_HOLO_LIGHT,
    is24Hour: false,
    minDate: new Date().getTime()
  }).then(
    dateTime => {
      this.date = dateTime;
    },
    err => console.log('Error occurred while getting dateTime: ', err)
  );
}
```

Ilustración 31 Implementación date-picker

- Fcm: Esta librería permite hacer la conexión a los servicios de Firebase y en este caso, recibir notificaciones push.

```
    this.fcm.getToken().then(token => {
      console.log(token);
    });
    this.fcm.subscribeToTopic('push');
  }
```

Ilustración 32 Implementación fcm ionic

- File: Esta librería permite otorgar acceso a la aplicación a leer y escribir archivos en el dispositivo del usuario. Como tal no hay una implementación en específico, pero a continuación se muestra un ejemplo.

```
  await this.file
    .readAsDataURL(path, imageName)
    .then(data => {
      this.imageUrl = data;
    })
    .catch(err => {
      this.imageUrl = undefined;
    });
}
```

Ilustración 33 Implementación file ionic

- Geolocation: Esta librería es usada para saber la ubicación actual del dispositivo, siempre y cuando haya conexión a internet. Esta ubicación se da como objecto al que se pueden acceder a muchas propiedades entre las que están la latitud y la longitud.

```
getGeolocation() {
  this.geolocation
    .getCurrentPosition()
    .then(resp => {
      this.getGeoencoder(resp.coords.latitude, resp.coords.longitude);
    })
    .catch(error => {
      alert('Error getting location' + JSON.stringify(error));
    });
}
```

Ilustración 34 Implementación Geolocation ionic

- http: Como ionic puede ser usado tanto para web como para móvil, la librería que viene incluida con el framework no funciona para hacer peticiones http en un dispositivo móvil, por esta razón es necesario instalarla.

```
this.tasks.forEach(task => {
  if (task._id) {
    this.http.put(
      `https://backendthesis.herokuapp.com/data/${task._id}`,
      task,
      {}
    );
  } else {
    this.http.post(
      `https://backendthesis.herokuapp.com/data/`,
      task,
      {}
    );
  }
});
```

Ilustración 35 Implementación http ionic

- Native-Geocoder: Por medio de esta librería se puede ingresar las coordenadas previamente extraídas y convertirlas a una dirección real, por ejemplo, si se ingresan las coordenadas [4.602835, -74.065039], por medio del Geocoder se obtiene la dirección Cra. 1 #18a 12, Bogotá, Cundinamarca, Colombia.

```
getGeoencoder(latitude: number, longitude: number) {
  this.nativeGeocoder
    .reverseGeocode(latitude, longitude, this.geoencoderOptions)
    .then((result: NativeGeocoderResult[]) => {
      this.address = this.generateAddress(result[0]);
    })
    .catch((error: any) => {
      alert('Error getting location' + JSON.stringify(error));
    });
}
```

Ilustración 36 Implementación native-geocoder ionic

- Status-bar: Al crear una aplicación de ionic por defecto no se muestra el status bar de los dispositivos móviles, por esto es necesario instalar esta librería para que sea parte de la aplicación. La implementación es simplemente declarar el color y el estilo.

```
this.statusBar.backgroundColorByHexString('#A9A9A9');
this.statusBar.styleLightContent();
```

5.3.3. Análisis

5.3.3.1 Facilidad de desarrollo

La facilidad de desarrollo de ionic depende de si el desarrollador tiene conocimientos previos de Angular. Pero partiendo del punto de que no es el caso, es un framework difícil de aprender. El hecho de que por cada componente se generen cuatro archivos diferentes, y en cada uno toque escribir líneas de código para que pueda servir ese componente, lo hace algo tedioso, complicado y largo. Lo mismo pasa si se implementan múltiples páginas, con la diferencia de que en este caso es necesario lidiar con un enrutamiento que no es intuitivo y toca prestarle mucha atención. Algo positivo es que el estilo y markup es igual que una aplicación pura de HTML y CSS.

5.3.3.2 Desempeño

Para el desempeño se van a analizar dos métricas, uso de CPU y uso de memoria (RAM).

En cuanto al uso de CPU, se puede observar en la siguiente imagen que es de 2% en promedio, lo cual es sorpresivo pues es un framework que no compila a código nativo, sino que actúa como un wrapper de cordova, esto significa que hay tareas puramente nativas que se las delega a cordova, pero muchas de las tareas son código JavaScript que esta corriendo en el dispositivo.

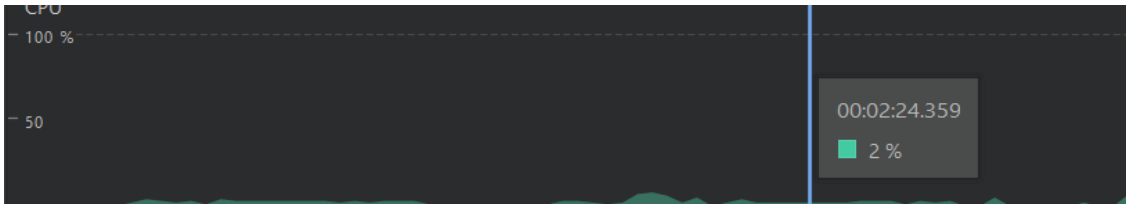


Ilustración 37 Uso CPU ionic

En el caso de la RAM, el resultado es más esperado pues tiene un consumo promedio de 162MB con picos de hasta 187MB. Este comportamiento se le puede atribuir en su mayoría a lo explicado en el caso del uso de CPU.

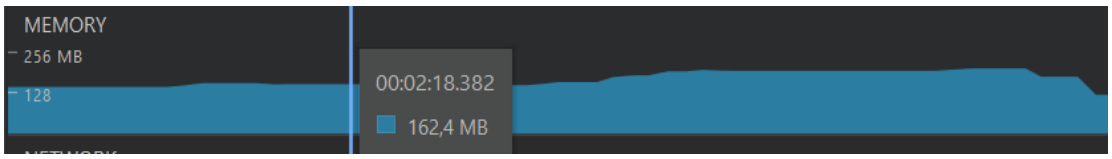


Ilustración 38 Uso RAM ionic

6. Comparación General

6.1. Estructura de los proyectos

Si se tiene en cuneta la estructura de los proyectos aquí analizados, es claro que el más optimo es el de Flutter, seguido por React-Native. Pensando que la aplicación es relativamente sencilla, si se escala y empiezan a agregar nuevas vistas y páginas, el proyecto de Flutter no crece mucho, pues como se mostró con anterioridad no es necesario crear muchos archivos por componente o funcionalidad que se le quiera agregar a la aplicación. En el caso de ionic, esto se es un problema, pues si se piensa en escalar la aplicación se vuelve un proyecto con un sinfín de archivos de componentes y páginas que en algún momento puede resultar complicado de manejar. Por otro lado, React-Native También tiene una estructura grande, pero tiene mucha mas libertad de la forma en la que se quiere estructurar la aplicación. Además, no se necesitan generar mas archivos por componente o página. Esto es decisión del desarrollador.

6.2. Facilidad de desarrollo

Partiendo de la base que el desarrollador no conoce ninguno de los frameworks, o en su defecto ninguno de los frameworks sobre los que están construidos. Flutter es sin duda el que mas facilidad tiene de desarrollo. Por un lado, la forma en la que se estructuran los arboles de widgets se logra de una forma bastante intuitiva. Manejar la lógica de la aplicación es también bastante simple pues solo se declaran variables, y si se quiere modificar algo, se hace por medio de la función setState y se modifica la variable deseada, al hacer esto la aplicación se actualiza de forma automática. Además, la documentación es excelente. El equipo de desarrollo se encargo de construirla de una forma que fuera fácil de entender para los que no están familiarizados, y aparte de eso, por cada widget que existe, hay videos y ejemplos de su implementación e integración con otros widgets en su página oficial.

React-Native e ionic están en el mismo nivel, React-Native tiene una forma, no tan simple, de manejar el estado, pero en ionic esta funcionalidad no viene por defecto con el framework, por lo que a veces al actualizar alguna variable que requiera actualizar la interfaz gráfica, no lo hace y toca recurrir a otros métodos. Por otro lado, crear la estructura, el markup de la aplicación, es mas sencillo en ionic que en React-Native, pues se hace de la misma forma que se escribe un archivo de HTML, pero con ventajas como tener obtener parámetros de la lógica desde el mismo tag.

6.3. Desempeño

Teniendo en cuenta el análisis previamente hecho de los tres frameworks, es claro que el que mejor desempeño tiene es Flutter en ambos casos, pues su uso de CPU es muy bajo y el uso de memoria RAM es la mitad que el de los otros dos. Además, al usar la aplicación, se siente que las interacciones con los elementos son mucho mas fluidas. Por ejemplo, en React-Native, hay un pequeño retraso cuando se agrega una tarea y se muestra en la aplicación, en Flutter ese proceso es prácticamente instantáneo. Esto se atribuye a que Flutter, es el único de los tres frameworks que compila su lenguaje (dart) 100% a código nativo para ambas plataformas.

Otra vez React-Native y ionic tienen comportamientos muy parecidos en términos de desempeño, pero si hay uno mejor, tendría que ser React-Native. Si bien el uso de CPU es bastante mayor, el consumo de RAM en general es menor y es un aspecto más importante pues la aplicación puede seguir corriendo sin problemas con un porcentaje de CPU alto, pero si se llena la memoria, la aplicación puede quedar totalmente paralizada o incluso perder datos importantes del usuario que no han sido guardados todavía.

7. Conclusiones

Teniendo en cuenta la comparación hecha previamente, el mejor de los tres frameworks es Flutter. Es el que tiene la curva de aprendizaje mas corta, y es el que mejor se asemeja en términos de desempeño y "touch and feel", a una aplicación nativa, sin ser todavía perfecto. Por otro, el catálogo de widgets que tienen es muy amplio y suficiente. Prácticamente no se necesitan librerías externas, y en caso de hacerlo en su mayoría están hechas por el mismo equipo de desarrollo de Flutter, mientras que en las otras toca recurrir en muchos casos a librerías hechas por terceros.

React-Native es una herramienta muy poderosa para desarrollo cross-platform también, pero tiene ciertas fallas que lo hacen caer por debajo de Flutter. Como por ejemplo agregar la funcionalidad de hacer una lista intercambiable, que en Flutter se hace simplemente con un solo widget, pero en React-Native se tiene que instalar una librería de externa, hecha por un tercero, y su implementación es complicada.

Ionic es el mas viejo de los tres, y por eso probablemente el que está detrás de los otros dos. Al haberse creado hace bastantes años han tenido que ir construyendo encima de todo lo anterior para agregar funcionalidades nativas para seguir compitiendo en el mercado, pero al hacer esto se ha vuelto difícil construir una aplicación con este Framework. Por esta razón es que ionic recientemente lanzo una versión basada en React, que puede ser mucho mejor que la original, basada en Angular.

8. Trabajo futuro

El proyecto se puede seguir trabajando bastante. Por un lado, se puede crear una aplicación de Flutter que siga estrictamente el patrón de arquitectura BloC y analizar si en efecto se genera una mejora en términos de desempeño y facilidad de manejar las interacciones con la aplicación. Además, se pueden crear nuevas características que expandan la aplicación general para ver como el escalamiento influye en el análisis y conclusiones hechas previamente.

También se pueden incluir nuevos frameworks en la comparación. Ionic normalmente ha sido trabajado con Angular, pero también es posible crear una aplicación con JavaScript puro, o recientemente sacaron la versión de Ionic React. Se puede incluir estos nuevos frameworks para seguir analizando y comparando.

Lo importante es extender el proyecto para que pueda ser una referencia para los desarrolladores a la hora de escoger un framework de desarrollo, teniendo en cuenta las capacidades y alcance de estos.

9. Links proyectos en GitHub

9.1. Flutter https://github.com/Nicolas9415/todoflutterspaghetti
9.2. React-Native https://github.com/Nicolas9415/reactNative
9.3. Ionic https://github.com/Nicolas9415/ionicTodo
9.4. Back-End https://github.com/Nicolas9415/backTesis
9.5. Firebase-Flutter https://github.com/Nicolas9415/cloudFunctionsFlutter
9.6. Firebase React-Native https://github.com/Nicolas9415/cloudFunctionsRN
9.7. Firebase ionic https://github.com/Nicolas9415/cloudFunctinsIOnic

10. Referencias

https://es.reactjs.org/. (s.f.).

https://es.stackoverflow.com/. (s.f.).

https://flutter.dev/. (s.f.).

https://ionicframework.com/. (s.f.).