**Escuelita Ionic v7**

¿Qué es Ionic?

Ionic es un SDK de código abierto utilizado para el desarrollo de aplicaciones móviles híbridas.

Escuela Ionic

La escuela de Ionic propuesta busca generar un primer acercamiento a un abanico de funcionalidades y temas que servirán de ayuda para la inserción en el mundo mobile.

Algunos temas propuestos son: simples vistas, integración de services, consumo rest, conexión a base de datos, manejo de promises, aplicación de estilos a las vistas, etc.

Software necesario

Git

Ionic nos inicializa un repositorio Git de manera local, es por ello que se requiere de la herramienta Git instalada.

Una vez garantizada la integración con Git y asegurados los archivos en el Head de la copia local (es decir, que han sido commiteados), podremos subir nuestras aplicaciones firmadas a un repositorio remoto, para lo cual debemos tener acceso a una plataforma de almacenamiento como GitHub o GitLab, por ejemplo.

Website: <https://git-scm.com/>

Link descarga: <https://git-scm.com/download/win>

Node & Npm

La herramienta CLI de Ionic se basa en node y npm para la gestión e instalación de paquetes. Se recomienda el uso de versiones LTS.

Website: <https://nodejs.org/en/>

Versión recomendada: Node 16.20.2 & Npm 8.19.4

*Para ver las versiones que tenemos instaladas, abrir una terminal de comandos y ejecutar node --version para ver la versión de Node, y npm --version para ver la versión de Npm.*

Ionic CLI

Una vez instalado Node & Npm, podremos instalar el *CLI de Ionic*  desde una terminal.

|  |
| --- |
| $ npm install -g @ionic/cli |

*El flag -g hace referencia a una instalación global. La instalación en la mayoría de los casos se realiza en %appdata%/Roaming/npm.*

Temario

1. [Creación de una app Ionic](#_heading=h.gjdgxs).
2. Modificar tabs bar y agregar un componente header[.](#_heading=h.1fob9te)
3. [Generar una vista que contenga una lista de usuarios.](#_heading=h.1fob9te)
4. [Crear página ‘Detalle de usuario’.](#_heading=h.3znysh7)
5. [Aplicar estilos en la vista detalle de usuario.](#_heading=h.2et92p0)
6. [Agregar funcionalidad ‘pull to refresh’ al listado de usuarios.](#_heading=h.tyjcwt)
7. [Conclusiones de la primera parte.](#_heading=h.3dy6vkm)
8. [Crear páginas ‘Login’ y ‘Registro’.](#_heading=h.1t3h5sf)
9. [Provider de autenticación para comunicación con backend(firebase).](#_heading=h.4d34og8)
10. [Formularios y validaciones](#_heading=h.2s8eyo1)
11. [Implementando métodos de autenticación en nuestra page.](#_heading=h.17dp8vu)
12. [Compilar para Android.](#_heading=h.26in1rg)
13. [Compilar para iOS.](#_heading=h.lnxbz9)
14. [Enlaces de referencia.](#_heading=h.35nkun2)
15. [Demo](#_heading=h.1ksv4uv)
16. [Créditos.](#_heading=h.44sinio)

# Creación de una app Ionic

Inicializando nuestra app

Abrir una terminal de comandos en el directorio de trabajo deseado y ejecutar el siguiente comando.

|  |
| --- |
| ionic start escuela-ionic-v7 tabs --**type=**react |

Explicando el comando inicial

ionic start le indica al CLI de Ionic que se desea inicializar una nueva app.

escuela-ionic-v7 es el nombre que se le va a asignar a nuestra app.

tabs indica que se utiliza el template tabs.

--**type=**react se utiliza para indicar que la versión del Framework a inicializar es la 7.

Para lanzar nuestra aplicación debemos usar ionic serve y luego en nuestro navegador, ingresar a las opciones de desarrollo(f12) y seleccionar la vista mobile.

Utilizar como guía los [componentes](https://ionicframework.com/docs/components) provistos por documentación de ionic donde existen ejemplos de cómo utilizarlos.

# Modificar tabs bar y agregar un componente header.

Composición de tabs

Se desea modificar los 3 ítems, los cuales son:

* Home
* Posts
* Settings

A su vez es necesario crear un nuevo componente ‘Header.tsx’ el cual renderiza una imagen de assets.

# Generar una vista que contenga una lista de usuarios.

Cuando generamos nuestra aplicación, se nos crea la página ‘Tab1’, la cual renombraremos a ‘Home’ y será la que utilizaremos para este ejercicio.

Consideraciones

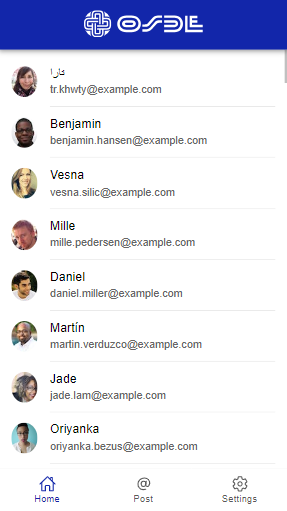
Para llevar a cabo la tarea necesitaremos de los siguientes elementos:

* Una ‘Page’ de Ionic(utilizaremos la que viene por defecto: Home).
* Crear una carpeta ‘api’ donde alojaremos los servicios.
* [Axios](https://axios-http.com/es/docs/intro) la cual nos permite realizar peticiones https.
* Un API público, en nuestro caso utilizaremos: [RandomUser](https://randomuser.me)

Generando el *Service*

Para generar nuestro service es necesario:

* *Crear la carpeta api con su correspondiente archivo, por ejemplo: “randomUserService.ts*
* *Instalar la dependencia de axios.*

Una vez generado veremos el siguiente mensaje:

Realizando una petición Https en nuestro *Service*

Para realizar peticiones https es necesario importar el módulo de axios.

Obteniendo los datos

En nuestro service debemos crear el método get que consuma el servicio de users, el cual nos retorna una promise.

Finalizado el método, dentro de nuestro ‘Home.tsx’ importamos el service, lo declararemos dentro de un useEffect y lo utilizaremos de la siguiente manera getUsers()

Prestar atención a cómo se retornan los datos y recordar que en las promises se puede utilizar promise.then()

Una vez que obtengamos los datos mediante nuestro services, los almacenaremos en un state dentro de ‘Home.tsx’

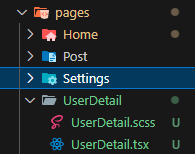
Imprimiendo datos en nuestra vista

Dentro de nuestra vista ‘Home.tsx’ crearemos:

* <IonList>: Para generar un listado de ítems
* <IonItem>: Para generar un ítem.
* <IonAvatar>: Para mostrar la imagen del usuario.
* <h2>: Para el título del ítem.
* <p>: Para el subtítulo del ítem.

# Crear página ‘Detalle de usuario’.

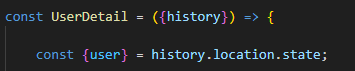
Como hicimos anteriormente con el servicio, ahora vamos a generar una página de una manera similar.



Recibiendo el dato de un usuario.

En nuestra página ‘UserDetail’ recibiremos por props un objeto correspondiente al history de la navegación. Este objeto history se podría utilizar para saber la ubicación actual de la ruta, navegar a otra ruta o reemplazar la actual. En nuestro caso lo utilizaremos para obtener datos enviados al momento de navegar a esta nueva ruta mediante el state

Para depositar el dato recibido en una variable local podríamos hacer algo como lo siguiente. Ej:



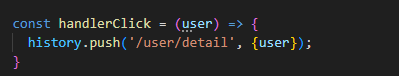
Una vez recibido el dato, podríamos crear un método que vaya asignando una a una las propiedades que deseamos obtener del objeto o simplemente usarlo como lo recibimos.

Enviando un dato a otra página

Como anteriormente se mencionó, necesitaremos enviar datos entre páginas. Este paso se realiza mediante el uso del módulo React Router y su método ‘[history.push()](https://v5.reactrouter.com/web/api/history)’.

En nuestro ‘home.ts’ deberíamos tener un método que se ejecute al hacer click sobre un elemento de la lista, este método debería ser capaz de navegar hacia nuestra página *UserDetail* y enviarle el usuario en el cual se realizó click.

Un ejemplo de cómo se podría construir el método es:



Como podemos observar, en el segundo parámetro de history.push, enviamos un objeto que contiene el usuario a transferir.

> *history.push(‘path\_name’, state)*

# Aplicar estilos en la vista detalle de usuario.

En una aplicación Ionic, muchas clases de estilos vienen predefinidas dependiendo del elemento que se quiera usar. También existen selectores que se pueden aplicar para obtener un control de los estilos si tener que escribir código css. (*ver* [*Css Utilities*](https://ionicframework.com/docs/layout/css-utilities)).

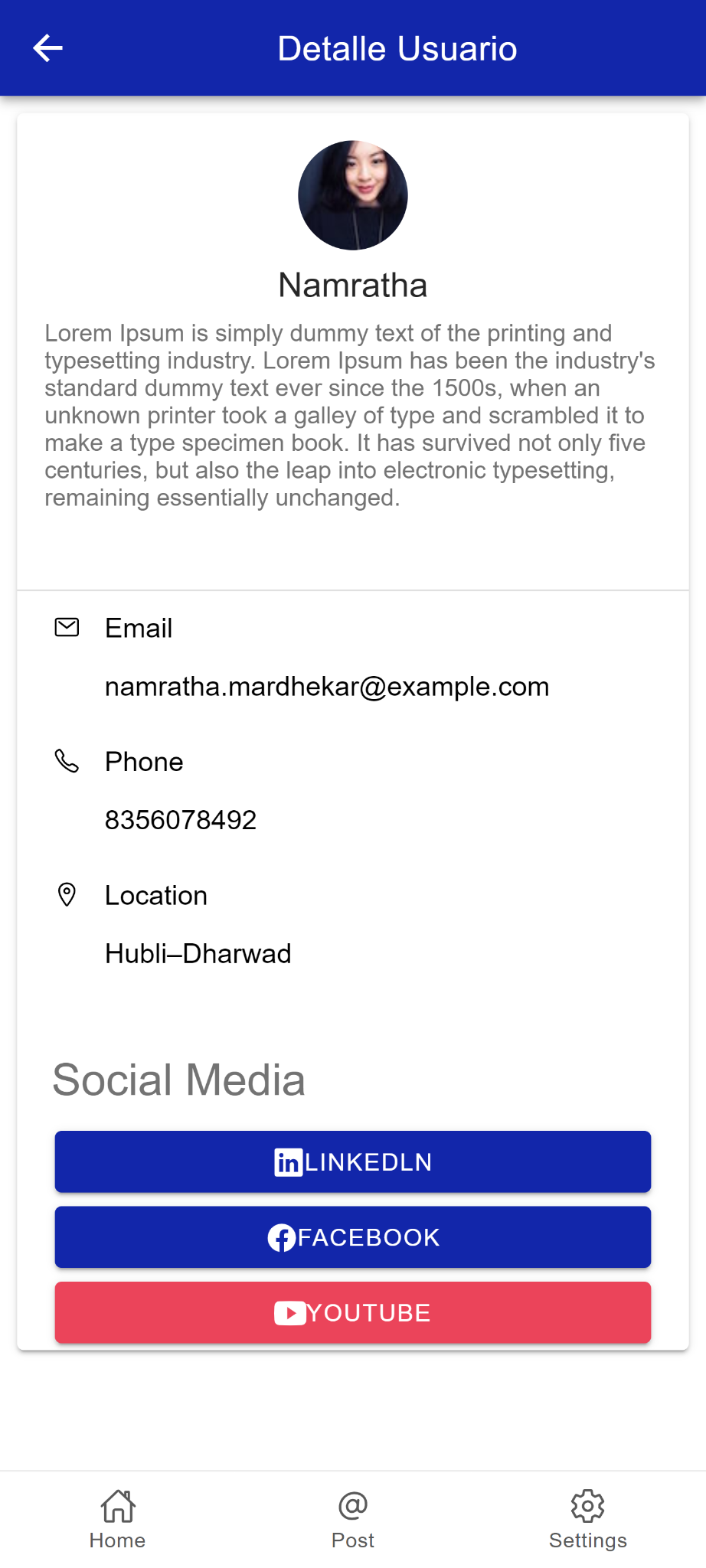
En nuestro caso, vamos a ser nosotros quienes vamos a decidir el estilo que tendrá nuestra vista *‘user-detail.html’*.

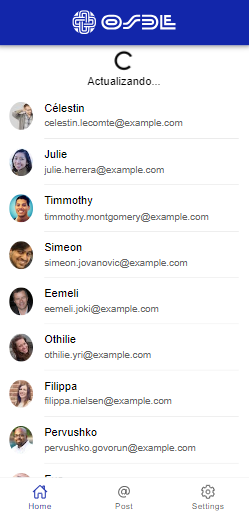
En el ejemplo que se ve a la derecha, usamos solamente 4 clases css y algunos selectores de *Css Utilities*.

* .profile-description{ }: Para la descripción e info de perfil.
* .container-buttons{ }: Para los detalles de iconos, label y detalle de numeros.

Tener en cuenta que:

* La imagen utilizada para el avatar es dinámica



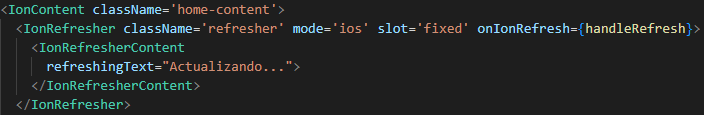


# Agregar funcionalidad *‘pull to refresh’* al listado de usuarios.

Una funcionalidad básica de las vistas que presentan un listado de datos es tener la posibilidad de actualizar el conjunto de datos. Para ello Ionic provee de un componente llamado [Refresh](https://ionicframework.com/docs/v7/api/refresher).

Agregando el componente *<IonRefresher>*

Uno de los requisitos es que nuestro componente <IonRefresher> sea el primer hijo de nuestro <IonContent>



Entendiendo *<IonRefresher>*

El componente *refresher* presenta una amplia personalización, llegando incluso a poder modificar por completo su apariencia.

Pero en nuestro caso, que utilizaremos el que viene por defecto, nos centraremos en el *<IonRefresherContent>.*

Explicando sus propiedades.

|  |
| --- |
| <IonRefresher className='refresher' mode='ios' slot='fixed' onIonRefresh={handleRefresh}">  <IonRefresherContent  pullingIcon = Icono a mostrar cuando se hace pull en la vista.  pullingText = Texto a mostrar cuando se hace pull en la vista.  refreshingSpinner = Icono a mostrar mientras actualiza.  refreshingText = Texto a mostrar mientras actualiza.>  </IonRefresherContent>  </IonRefresher> |

*\*\* El nombre de los iconos a utilizar se pueden obtener de ionicons v7.*

Implementando onIonRefresh()

Cuando nuestro componente detecte el evento de refresco, lanzará nuestro evento onIonRefresh() pasándole como parámetro dicho evento.

Este método debemos implementarlo en nuestra lógica de página. Dentro de éste, se invocará al método que nos retorna el listado actualizado, y, al finalizar la operación, llamaremos a event.detail.complete().

# Conclusiones de la primera parte.

Hasta ahora, hemos estado trabajando con varios aspectos que nos ofrece el framework de Ionic, entre ellos su CLI, archivos de lógica y estructura (.jsx o .tsx), archivos de estilos (.css o .scss), entre otros. Con todo este conocimiento adquirido y los que adquiriremos más adelante, tendremos las herramientas necesarias para la mayoría de los requerimientos. En caso de no tener en claro el panorama en este punto, investigar sobre los siguientes ítems debería poder ayudarte:

* ¿Qué es un CLI?
* ¿Qué son Ionic y Capacitor? ¿Son lo mismo?
* ¿Qué es NodeJs? ¿Y NPM?
* ¿Qué es un hook? ¿Cómo se usa useEffect?
* ¿Por qué en las importaciones de dependencias usamos import { Component }...?
* ¿Cuáles son las diferentes formas de navegar entre páginas utilizando React Router?
* ¿Qué es una Promise? Diferencias entre Async Await y Promise
* ¿Qué es una arrow function?
* ¿Cómo se lleva a cabo la inyección de dependencias en React?
* ¿Conoces los ciclos de vida de un componente en React? ¿Para qué nos sirven?
* ¿Conoces TypeScript? ¿Por qué consideras que se usa?
* ¿Sabes qué función cumple el package.json?
* ¿Qué es un custom hook? ¿Se te ocurre un caso donde sea aplicable usarlos?
* ¿Sabes cómo exportar variables y funciones?

Estas son algunas de las preguntas que te pueden ayudar a entender un poco el ecosistema en el cual se mueve Ionic y React.

# Crear páginas ‘Login’ y ‘Registro’.

Comenzaremos por generar ambas páginas, generamos las vistas que incluyen los [formularios](https://formik.org/docs/tutorial).

Podemos observar una primera aproximación al formulario de login, donde nos interesa implementar la funcionalidad de crear una cuenta (register), y de ingresar a la aplicación (log in).

Autenticando con Firebase

Toda nuestra funcionalidad de login valida contra nuestro backend en Firebase.

Toda nuestra información de usuario puede ser manipulada desde la consola que nos provee Firebase en la sección de autenticación.

En la misma podremos generar nuevos usuarios manualmente, inhabilitar cuentas e incluso eliminarlas.

Cabe mencionar que en esta guía no se enseña a utilizar Firebase. Esta ofrece muchas funcionalidades y la documentación es completa y extensa.

En caso de que no se quiera utilizar la plataforma para este proyecto, se puede optar por crear datos mockeados desde el lado de la aplicación.

Consola de autenticación



# Service de autenticación para comunicación con backend(firebase).

Con la incorporación de Firebase a la aplicación buscamos darle capacidad de almacenar datos sensibles por fuera de nuestros dispositivos móviles. Particularmente queremos incluir la funcionalidad de login y creación de cuentas.

La configuración de Firebase escapa a los fines de este curso, por lo que nos limitaremos a explicar cómo integrar nuestra aplicación con la plataforma.

Para poder comunicarnos con el [Api de Firebase](https://firebase.google.com/docs/auth/), necesitaremos crear un service, en el cual inyectamos nuestras dependencias de firebase.

Utilizaremos [firebase](https://www.npmjs.com/package/firebase) para este procedimiento:

* firebase: Es la librería que nos permite acceder a todas las características que ofrece [Google Firebase](https://firebase.google.com/).

Para ello:

* Para ello crearemos un hook personalizado ‘useFirebase.ts’.
* Importamos la librería de firebase.

|  |
| --- |
| import { getAuth, createUserWithEmailAndPassword, signInWithEmailAndPassword, signOut, UserCredential } from 'firebase/auth';  import { app } from '../config/firebaseConfig';  export const useFirebase = () => {  const auth = getAuth(app); |

Métodos de sesión

Generamos los métodos que utilizaremos en nuestras páginas de Login y Registro, se recomiendan login(), register(), logOut().

Estos métodos al estar en un service, es recomendable que retornen una Promise, de esta manera podremos manejar todas nuestras solicitudes de manera asincrónica.

Métodos de autenticación de firebase.

Como se mencionó antes, utilizaremos los métodos que nos provee firebase para comunicarnos con el Api, algunos de ellos son:

Firebase:

* signInWithEmailAndPassword()
* createUserWithEmailAndPassword()
* signOut()

Para brindar ayuda, dejo un ejemplo de cómo implementar el login() dentro de nuestro hook:

|  |
| --- |
| const login = (email: string, password: string) => {  return new Promise<UserCredential>(async (resolve, reject) => {  try {  const signIn = await signInWithEmailAndPassword(auth, email, password);  resolve(signIn);  }  catch (error) {  reject(error);  }  })  } |

En este ejemplo, si la Promise es resuelta retorna el resultado, si da error retorna el error.

# Formularios y validaciones

Validando Inputs

Sería una buena práctica considerar la incorporación de métodos de validación para los inputs de los formularios de login y registro del usuario. Para realizar las validaciones crearemos un nuevo hook personalizado.

Creando las validaciones en useFormValidation.ts

export interface FormValues {

email: string;

password: string;

}

//En este método se puede agregar todas las validaciones que se crean necesarias.

const validate = (values: FormValues) => {

const errors: Partial<FormValues> = {};

if (!values.email) {

errors.email = 'El email es requerido';

} else if (!/\S+@\S+\.\S+/.test(values.email)) {

errors.email = 'El email es erroneo';

}

if (!values.password) {

errors.password = 'El password es requerido';

} else if (values.password.length < 6) {

errors.password = 'El password debe contener 6 caracteres como minimo';

}

return errors;

};

const useFormValidation = (initialValues: FormValues, onSubmit: (values: FormValues) => Promise<void>) => {

const formik = useFormik<FormValues>({

initialValues,

validate,

onSubmit,

});

return formik;

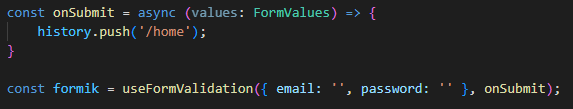
};

Utilizando los validadores en nuestro form

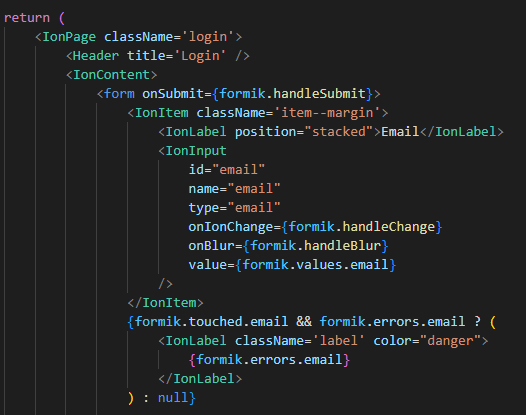
Para poder utilizar nuestros validadores en el formulario, es necesario importar nuestro nuevo hook creado.

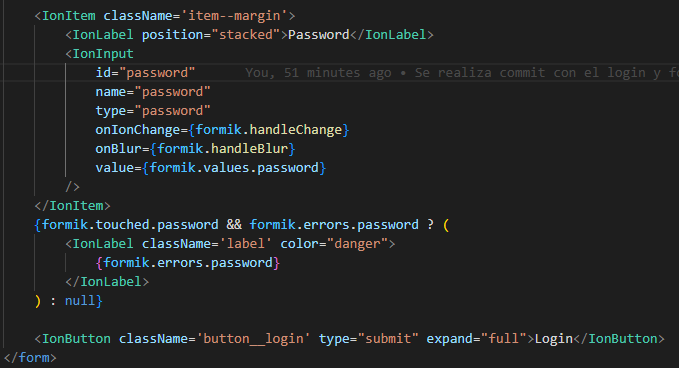


Luego es necesario hacer el llamado de nuestro useFormValidation pasándole por parámetros el initialValues y el método onSubmit.



Finalmente hay que integrar el form dentro de nuestro IonContent de la siguiente manera.



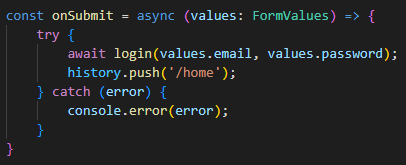


# Implementando métodos de autenticación en nuestra page.

A continuación, se implementan los métodos relacionados a las distintas funcionalidades presentadas en la vista (métodos para login y register, como mínimo). Estos métodos se comunican con los del service de useFirebase.ts para invocar las distintas operaciones sobre la base de datos de firebase.

Cabe recordar que nuestros métodos de autenticación que creamos anteriormente, retornan promesas, las cuales se manejan de manera asincrónica (tener esto muy en cuenta).

Para ejemplificar, en Login.tsx :



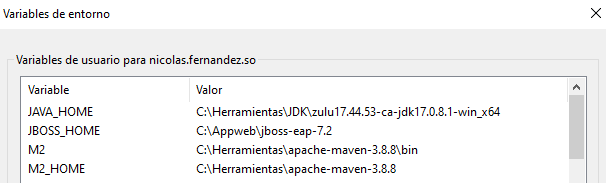
Gracias a que generamos un hook para el manejo de autenticación, con llamar los métodos correspondientes, ya se podría hacer el login de manera efectiva.

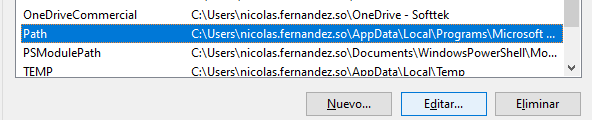
Los demás métodos se deberán crear siguiendo el ejemplo brindado para el login.

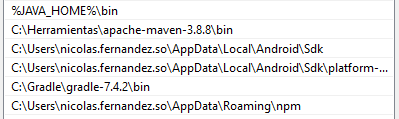
# Compilar para Android.

Para llevar a cabo la compilación para Android necesitamos tener descargado JDK de Java, SDK de Android y el [Gradle7.5.1](https://downloads.gradle-dn.com/distributions/gradle-7.4-all.zip), luego modificar la variable de entorno Path de la cuenta de usuario de modo que incluya las rutas a JDK, Gradle y Android SDK, para este último es necesario instalar [Android Studio](https://developer.android.com/studio) el cual al abrirlo nos recomienda realizar una descarga del SDK y por defecto lo podremos encontrar en:

C:\Users\usuario.so\AppData\Local\Android\Sdk







Agreagamos las plataformas:

npx cap add android

npx cap add ios

en nuestro caso solo android

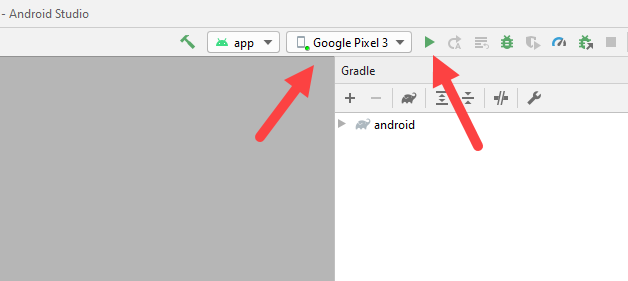
Dicho esto, el próximo paso consiste en abrir una consola en la raíz de nuestra aplicación y ejecutar el siguiente comando:

ionic build

Una segunda opción que tenemos es desplegar la aplicación a un dispositivo de forma directa a través del comando:

ionic cap open android

Utilizando Android Studio, podremos descargar emuladores para distintos dispositivos y utilizarlos para levantar la aplicación.



# Compilar para iOS.

Si bien la compilación en iOS es importante, en este documento no abordaremos este tema por varios factores, entre ellos está la complejidad en las configuraciones a realizarse en el xCode, la necesidad de tener una mac y una cuenta de apple developer, entre otras.

# Enlaces de referencia.

* Ionic Framework Docs: <https://ionicframework.com/docs/v7/>
* Git: <https://git-scm.com/book/es/v2>
* React Docs: <https://react.dev/reference/react>
* NodeJs: [https://nodejs.org/](https://nodejs.org/en/)
* NpmJs: [https://npmjs.com/](https://www.npmjs.com/)
* Google Firebase: <http://firebase.google.com/docs>
* Android Studio: <https://developer.android.com/studio>
* TypeScript: [https://typescriptlang.org/docs/](https://www.typescriptlang.org/docs/)
* SasS: <https://sass-lang.com/documentation>
* GitHub: <https://help.github.com/es#dotcom>

# Demo

La sección de Posts fue excluida de esa guía para acortar su complejidad.

Para la utilización de a demo, se puede generar una nueva cuenta, o usar el siguiente usuario, el cual tiene ‘posts’ creados:

User: [test@test.com](mailto:test@test.com)

Pass: 123456

Para ver la demo de la escuelita acceder a:

<https://escuelita-ionic-stk.firebaseapp.com/>

# Créditos.

|  |  |
| --- | --- |
| Autor | REC2 - Cristian Reyes |
| Colaborador | GOCO1 - Gonzalo Collazo |
| Version 5 | JOMU1 - Jonatan Müller |
| Frente | Urgencias |