Aplicaciones Distribuídas en Internet:

Trabajo en grupo sobre Ionic Framework

Serhii Vidernikov Y4459773H

Nikita Polyanskiy Y4441167L

Índice

Introducción

El trabajo trata de explicar las bases de Ionic, detallando las características y el porqué es interesante para el desarrollo de las aplicaciones móviles. Después, explicaremos cómo iniciar un proyecto, la estructura, librerías usadas y componentes de Ionic. Respecto a la parte práctica, hemos desarrollado una aplicación móvil sencilla. Va destinada a las personas que quieren acordarse y tener seguimiento de tomar agua. En la parte práctica detallaremos las funcionalidades que se han implementado.

Ionic

Si tenemos que desarrollar una aplicación para Android, debemos saber Java, Kotlin o C/C++ etc. Lo mismo pasa con IOS, para desarrollar debemos conocer Swift o Objective-C. Existen soluciones multiplataforma para poder resolver este problema. Ionic permite desarrollar aplicaciones híbridas, es decir, tanto para Android, como para IOS y por supuesto aplicaciones web, pero ¿por qué elegir Ionic? Puesto que se utiliza tecnología web, es muy fácil aprenderlo. Si ya sabemos desarrollar aplicaciones web, vamos a poder emplear todos nuestros conocimientos y nos será muy fácil aprender nuevos conceptos de Ionic. Otro aspecto importante es que se puede desarrollar en base a Angular, React o Vue y tenemos acceso a todas las librerías npm.

Para desplegar una aplicación web como una aplicación móvil nativa debemos usar capacitor o cordova. Ambos son proyectos open source y se consideran equivalentes. También hay mucha documentación e información para ambos, pero últimamente se recomienda usar capacitor puesto que es más reciente y usa tecnologías innovadoras.

Hay que destacar que la documentación de Ionic es muy completa, está en constante mantenimiento y es muy útil a la hora de desarrollar una con Ionic. Es imprescindible tenerla abierta para poder personalizar y aprovechar los componentes al máximo.

Por último, es importante mencionar que Ionic no proporciona el mejor rendimiento. Si queremos desarrollar una aplicación que requiera muchos recursos, como renderizado en 3D, animaciones, juegos, no es nada recomendable. En cambio, si lo que queremos es una aplicación para realizar operaciones con datos, navegar entre páginas, o por ejemplo, implementar funcionalidades de una tienda, no va a haber ningún problema.

Primeros pasos con Ionic

Para instalar Ionic primero debemos tener el Node.js. Gracias a ello podemos instalar el Ionic en sí con el siguiente comando:

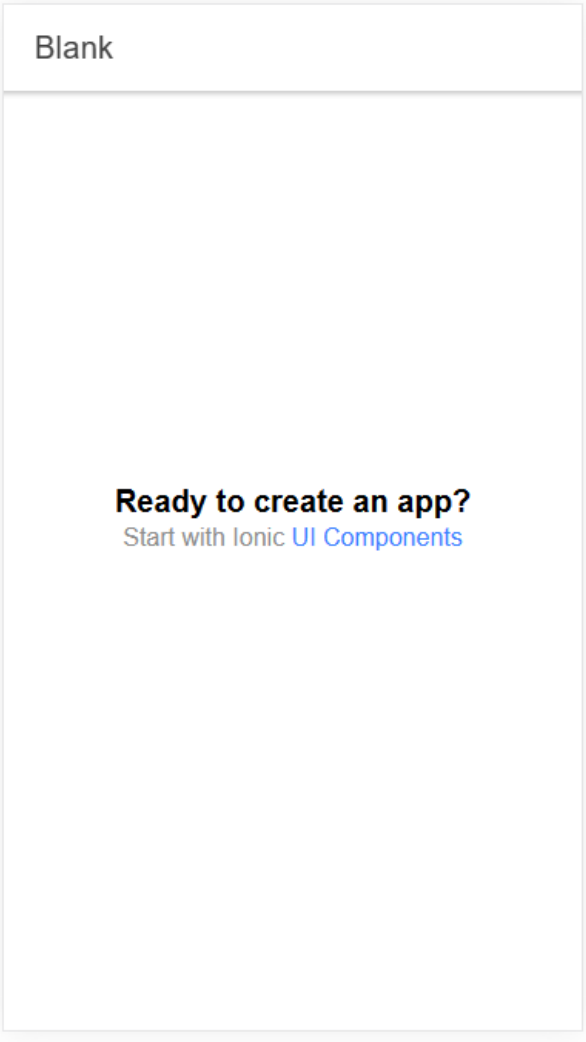
```
npm install -g @ionic/cli
```

A continuación podemos iniciar un nuevo proyecto con:

```
ionic start
```

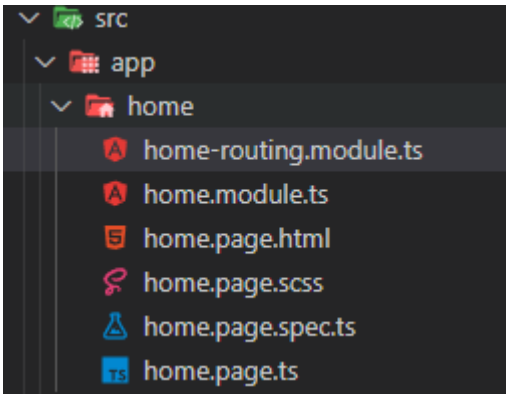
Debemos rellenar algunos datos como el nombre del proyecto, el framework que se utilizará y la plantilla. En este caso hemos puesto de nombre IonicADI, el framework Angular y un proyecto vacío (blank). En cuanto se instale el proyecto, accedemos a la carpeta y con ionic serve ponemos en marcha el proyecto. Por último, también necesitaremos capacitor. Es posible que se instale por defecto, pero se puede añadir con los siguientes comandos:

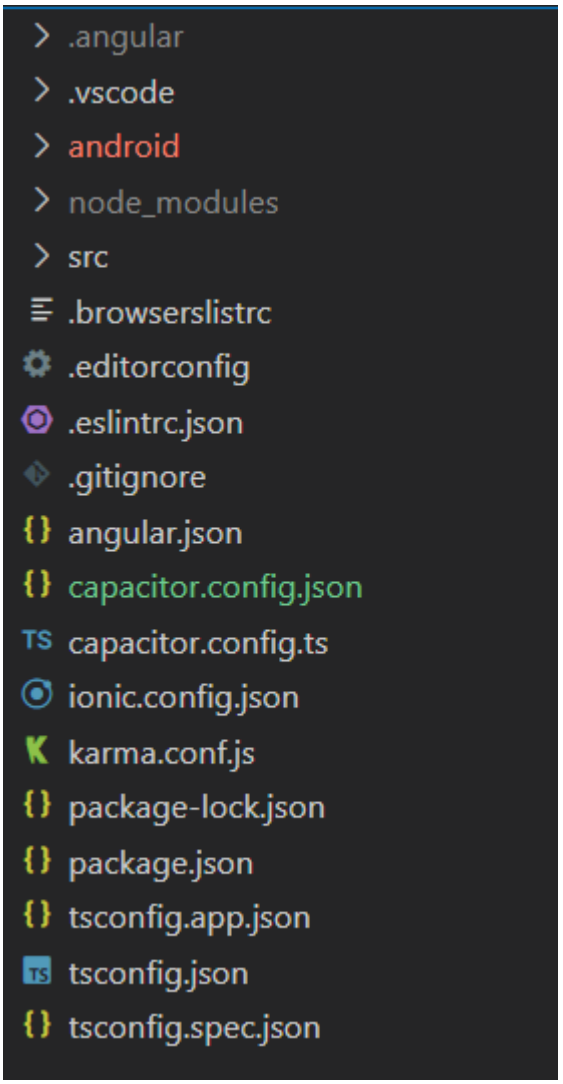
```
ionic integrations enable capacitor
npm i @capacitor/app @capacitor/haptics @capacitor/keyboard
@capacitor/status-bar
```



Una vez realizados estos pasos podemos poner en marcha el proyecto y acceder a él con localhost:8100.

Como el proyecto se basa en angular, las carpetas y archivos que se crearon son de estructura angular. Podemos ver como hay archivos de dependencias y configuración de proyecto. Lo que más nos interesa de aquí es la carpeta src donde estará ubicado todo nuestro código. Aunque el proyecto está vacío, tenemos creados ya algunos archivos y el componente base. El main.ts es el archivo principal que se encarga de poner en marcha la aplicación. Si entramos en la carpeta app es donde encontraremos los componentes y el componente de la app. Una carpeta contiene un componente, como por ejemplo, home que es la página principal, mostrada en la captura anterior.





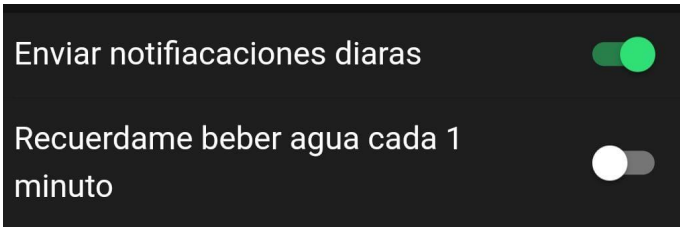
Podemos ver que contiene varios ficheros pero los que nos interesa es home.page.html que contendrá código html y los componentes ionic. Otro fichero es home.page.scss que permite añadir código css. Por último tenemos el home.page.ts que es donde estará ubicada la lógica de la página.

Si queremos crear nueva página debemos usar el comando ionic generate. A continuación elegimos qué es lo que queremos generar. En este caso nos interesa una página pero también hay opciones como componente, servicio, clase, etc. Una vez creado, si ponemos el servidor en marcha, podemos entrar en la página recién creada poniendo el nombre de la misma en la url.

Componentes de Ionic

ion-toggle

Uno de los componentes html que nos proporciona Ionic es el `<ion-toggle>`, es decir un botón de activado/desactivado. Lo hemos utilizado en "home.page.html" para darle la opción al usuario de activar o desactivar notificaciones diarias y recordatorio de beber agua cada minuto.




```
<ion-item>
  <ion-label class="ion-text-wrap">Enviar notifiacaciones diaras</ion-label>
  <ion-toggle color="success" [checked]=enviarNotif (click)="toggle(permiso)" #permiso></ion-toggle>
</ion-item>
<ion-item>
  <ion-label class="ion-text-wrap">Recuerdame beber agua cada 1 minuto</ion-label>
  <ion-toggle color="success" [checked]=reminderAgua (click)="reminder(permiso2)" #permiso2></ion-toggle>
</ion-item>
```

Para que cada opción permanezca en una línea separada se ha utilizado `<ion-item>`, y dentro de él un `<ion-label>` para la etiqueta del botón, y el botón en si `<ion-toggle>`. Como podemos ver dentro del componente de `<ion-toggle>` hay varias opciones:

1) `[checked]=reminderAgua`, es la variable de la cual depende el estado de nuestro botón;
2) `(click)="reminder(permiso2)"` será la función a la que se llama en el evento de click del botón;
3) `permiso2` es un nombre que podemos asignar al botón. Como es una instancia html, para poder ver si está encendido, debemos usar permiso2.checked. En otros casos podemos acceder al valor de, por ejemplo `<ion-input>`, con el .value

```
async reminder(permiso2: any){
  if(!permiso2.checked){
    await this.StorageService.setToggle2(true)
    this.reminderAgua = permiso2.checked
    this.sendLocalNotificationNow()
  }else{
    await this.StorageService.setToggle2(false)
    this.reminderAgua = permiso2.checked
    this.cancelLocalNotificationNow()
  }
}
```

ion-card

Es un contenedor para texto, imágenes, botones, o listas. Ofrece grandes posibilidades de personalización, se puede incluir un `<ion-card-content>`, `<ion-card-title>` etc.

```
<ion-card>
  <ion-card-content>
    <b>{{texto}}{{vecesHoy}} Vasos</b>
  </ion-card-content>
</ion-card>

<ion-card>
  <ion-card-content>
    <div id="myChart" [options]="options" class="demo-chart" echarts></div>
    <ion-button (click)="loadChart()">Reload Chart</ion-button>
  </ion-card-content>
</ion-card>
```

En nuestro caso hemos utilizado una card para mostrar el contador de vasos de hoy, y otro componente igual para el gráfico de los vasos bebidos en otros días. Podemos ver como crea una especie de papel por encima de la pantalla principal con una sombra.

Conclusión

Como hemos visto, Ionic ofrece componentes para poder desarrollar una aplicación muy rápido, puesto que son muy fáciles de personalizar y tienen una apariencia agradable desde que los introducimos sin tener que agregar css. Es decir, es muy fácil introducir nuevos componentes, personalizarlos, conectarlos al código, hacerlos responsive etc.

Librerías usadas

Local Notifications

Esta librería se ha utilizado para poder enviar y planificar notificaciones locales al móvil. Para instalarla en nuestro proyecto introducimos los siguientes comandos en la terminal:

```
npm install @capacitor/local-notifications
ionic generate
```

Como hemos mencionado anteriormente, usando “ionic generate” elegimos el tipo service. Con ello se generará el fichero .service.ts donde vamos a implementar funciones que nos convengan para conectar con la api de la librería, en este caso “local-notifications”. Una vez hecho esto se nos generará automáticamente algunos ficheros necesarios para el funcionamiento del servicio. En nuestro caso hemos creado el fichero “./app/local-notification.service.ts”, en primer lugar añadiremos el import de la librería:

```
import { LocalNotifications } from '@capacitor/local-notifications';
```

Dentro de la clase correspondiente implementaremos las funciones que necesitemos, por ejemplo la función que aparece a continuación se utiliza para planificar enviar una notificación cada 1 minuto.

```
constructor() { }

async showLocalNotification(id: number, title: string, text: string){
  this
  await LocalNotifications.schedule({
    notifications: [
      {
        title: title,
        body: text,
        id: id,
        autoCancel: true,
        schedule: { every: 'minute'}
      }
    ]
  })
}
```

Otro ejemplo sería la siguiente función, en la cual se planifican varias notificaciones para ser enviadas a determinadas horas del día, como se puede ver en la siguiente captura, se enviará una notificación cada día a las 9 y a las 11 de la mañana.

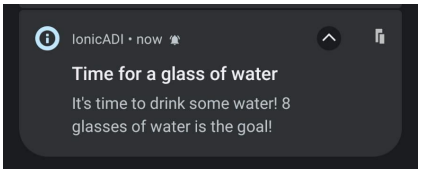
```
40      async DefaultNotification(){
41        await LocalNotifications.schedule({
42          notifications: [
43            {
44              // 1
45              title: "Buenos dias",
46              body: "Es importante hidratarse bien despues de dormir",
47              id: 1,
48              schedule: {
49                on: {hour:9, minute: 0},
50                allowWhileIdle: true,
51                //at: new Date(m),
52                // every: 'day'
53              }
54            },
55            {
56              // 2
```

La opción “allowWhileIdle: true” nos permite enviar la notificación aunque el móvil esté con la pantalla bloqueada.

Luego para poder cancelar estas notificaciones planificadas se han creado algunas funciones, una de ellas es la siguiente. Se encarga de cancelar las notificaciones diarias de la captura anterior, debemos pasarle el id de las notificaciones a cancelar a la función cancel() de la siguiente forma:

```
async cancelLocalNotification() {
  let option:CancelOptions={
    notifications:[
      {id:1},
      {id:2},
      {id:3},
      {id:4},
      {id:5},
      {id:6},
      {id:7},
      {id:8}
    ]
  }
  LocalNotifications.cancel(option);
}
```

Las notificaciones en cuestión se verán de la siguiente forma:



Storage

Es una librería de almacenamiento de datos simple utilizando clave-valor. Nosotros la utilizamos para guardar variables dentro de la aplicación o sitio web. De esta manera si cerramos la aplicación o la página, se guardarán los datos. Si iniciamos el proyecto y accedemos mediante navegador, podemos entrar a inspeccionar página, a continuación "aplicación" → "almacenamiento" → "IndexedDB" → "ionicstorage" podemos ver todos los datos guardados para esta dirección url:

0	"10/01/23"	▶ 3
1	"11/01/23"	▶ 5
2	"12/01/23"	▶ 7
3	"13/01/23"	▶ 6
4	"7/01/23"	▶ 1
5	"8/01/23"	▶ 3
6	"9/01/23"	▶ 0
7	"toggle"	▶ false

Para instalarla se utiliza el siguiente comando:

```
npm install @ionic/storage-angular
```

Luego creamos un fichero que contendrá las funciones relacionadas con el almacenamiento, "./app/storage.service.ts", al cual le añadiremos el import de la librería y luego inicializamos el Storage:

```typescript
import { Injectable } from '@angular/core';
import { Storage } from '@ionic/storage-angular';

const STORAGE_KEY = "lista"

@Injectable({
  providedIn: 'root'
})

export class StorageService {

  constructor(private storage: Storage) {
    this.init()
  }

  async init(){
   this.storage.create();
  }
```

Un ejemplo de uso del Storage serían las funciones de la siguiente captura, las cuales se usan para guardar un valor en la fecha y obtener el valor de la fecha.

```typescript
async getData(key: string) {
  const resul = await this.storage.get(key)

  if (resul == null || Number.isNaN(resul)) {
    return 0
  }
  return resul
}

async addData(key: string, item: number) {
  return this.storage.set(key, item)
}
```

Luego para poder acceder a estas funcionalidades desde nuestro fichero typeScript, añadiremos el siguiente import:

```typescript
import { StorageService } from '../storage.service'
```

Y se pasará la variable al constructor junto con el servicio de notificaciones:

```typescript
constructor(private localNotification: LocalNotifiactionService, private StorageService: StorageService) {
```

De esta manera podemos invocar las funciones, pasando los parámetros requeridos:

```typescript
async addOne() {
  this.vecesHoy = this.vecesHoy + 1
  await this.StorageService.addData(this.day, this.vecesHoy)
  this.asignarTexto()
}
```

ngx-echarts

Para instalar esta aplicación se utilizan los siguientes comandos:

```
npm install echarts -S
npm install ngx-echarts -S
```

En nuestro caso se ha utilizado para poder utilizar gráficos estadísticos de los vasos bebidos.
Para utilizarla en nuestra aplicación, en el fichero module de la página en la que vayamos a utilizarla introducimos lo siguiente:

```
import { NgxEchartsModule } from 'ngx-echarts';
@NgModule({
  imports: [
    CommonModule,
    FormsModule,
    IonicModule,
    HomePageRoutingModule,
    NgxEchartsModule.forRoot({
      echarts: () => import('echarts'),
    }),
  ],
  declarations: [HomePage]
})
export class HomePageModule {}
```

Luego en el fichero html introducimos el gráfico de esta forma:

```
44      <div [options]="options" [loading]="true" class="demo-chart" echarts></div>
```

La opción `[loading]="true"` permite al gráfico mostrar que se está cargando mientras espera para obtener los datos, y la opción `[options]="options"` será la variable en donde se guardan todas las opciones y datos necesarios para el gráfico, la cual se declara de la siguiente forma:

```
14  export class HomePage {
15    options: any
```

Podemos ver diferentes opciones y ajustes para el gráfico, pero lo más importante serán los datos, para el eje X se introducen en `xAxis:[data: week]`, y para el eje Y en `series:[{data: values}]` con los datos previamente sacados del almacenamiento.

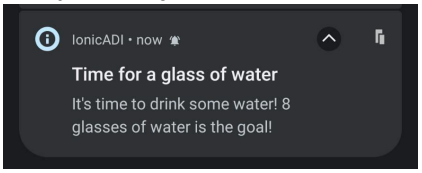

```
63    this.options = {
64      title: {
65        text: 'Seguimiento:',
66        show: true,
67      },
68      color: ['#3398DB'],
69      grid: {
70        left: '3%',
71        right: '4%',
72        bottom: '3%',
73        containLabel: true
74      },
75      xAxis: [
76        {
77          type: 'category',
78          data: week,
79          axisTick: {
80            alignWithLabel: true
81          }
82        }
83      ],
84      yAxis: [
85        {
86          type: 'value'
87        }
88      ],
89      series: [
90        {
91          name: 'Test',
92          type: 'bar',
93          barWidth: '60%',
94          data: values
95        }
96      ]
97    };
```

Resultado de la desarrollo

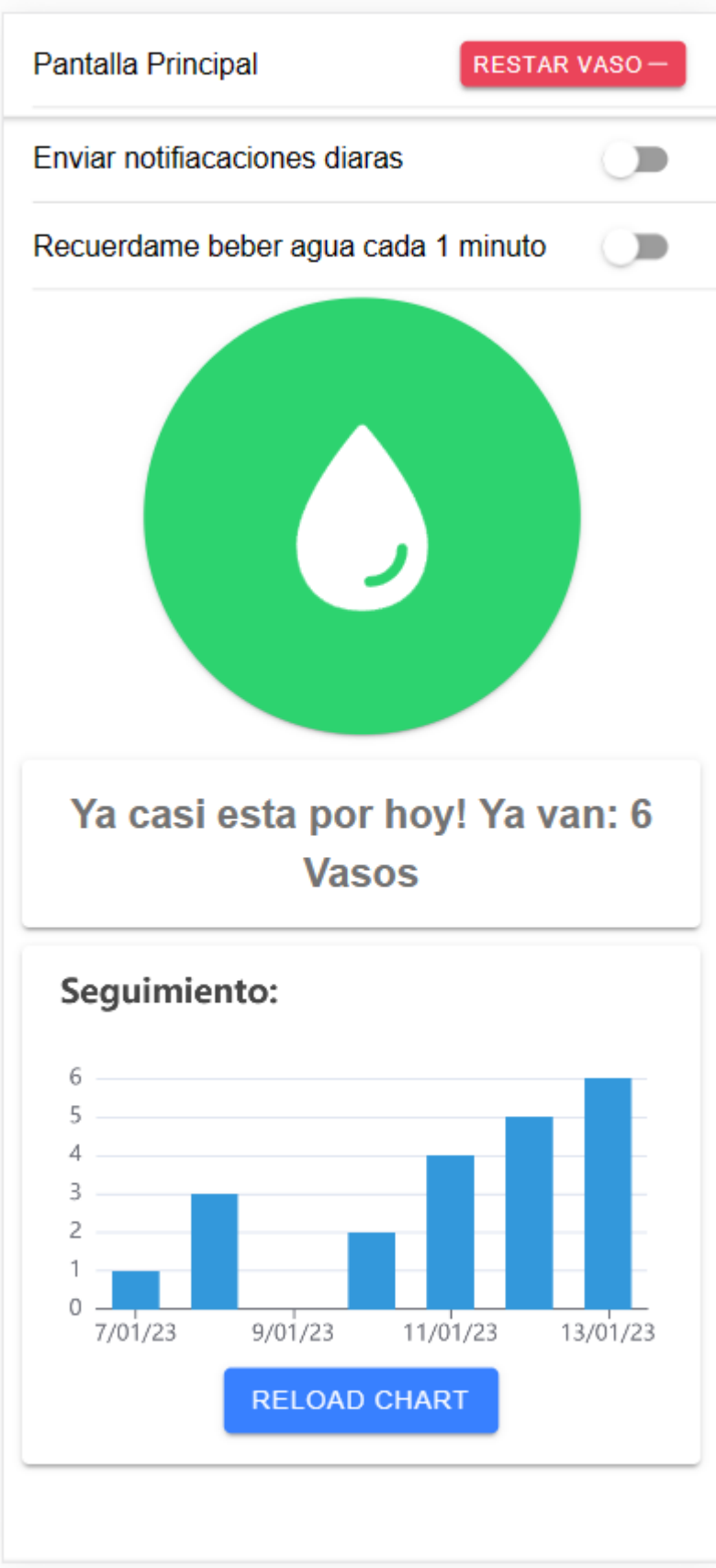
Como ya se ha mencionado en la introducción, la aplicación ayudará a las personas que quieren acordarse de tomar agua y tener un seguimiento. Solo consiste en una pantalla, aunque se podría dividir en más para que no estuviera sobrecargada de información.

Para empezar, lo que más llama la atención es el botón verde. Si le damos, indicamos que hemos tomado un vaso de agua y se sumará 1 al contador. La idea es llegar a tomar al menos 8 vasos al día, por lo que dependiendo de cuantos van, se cambiara el mensaje de debajo del botón.

Arriba podemos observar un botón de restar vaso, por si el usuario se ha equivocado. A continuación tenemos dos opciones de configuración. La opción de recordar tomar agua cada minuto es para mostrar el funcionamiento de las notificaciones y no entraría en la versión final del proyecto. Ejemplo de notificación:



Por último, gracias a la librería de echarts, hemos conseguido añadir un gráfico de seguimiento, que muestra los últimos 7 días y los vasos que se han tomado. Se ha agregado un botón para recargar la gráfica por si hay algún error o queremos actualizarla.



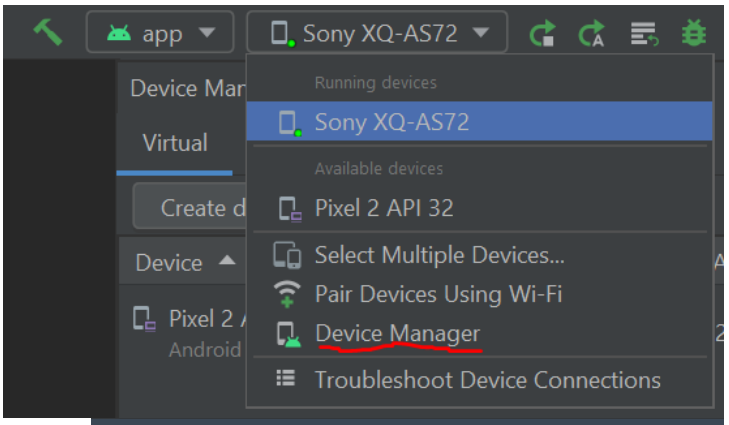
Exportar la aplicación

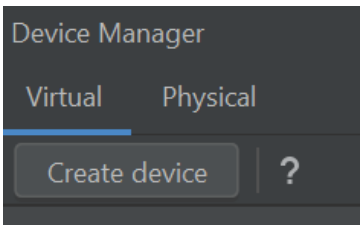
Android Studio

Android Studio es un entorno de desarrollo de aplicaciones móviles para Android. Se ha utilizado para probar nuestra aplicación en un entorno de móvil real, usando tanto un emulador de móvil que nos proporciona el mismo Android Studio, como uno de nuestros móviles. Para instalar Android Studio primero debemos descargarlo de la página oficial.
Si nos situamos en el directorio de proyecto y ejecutamos los siguientes comandos se abrirá el Android Studio con nuestro proyecto.

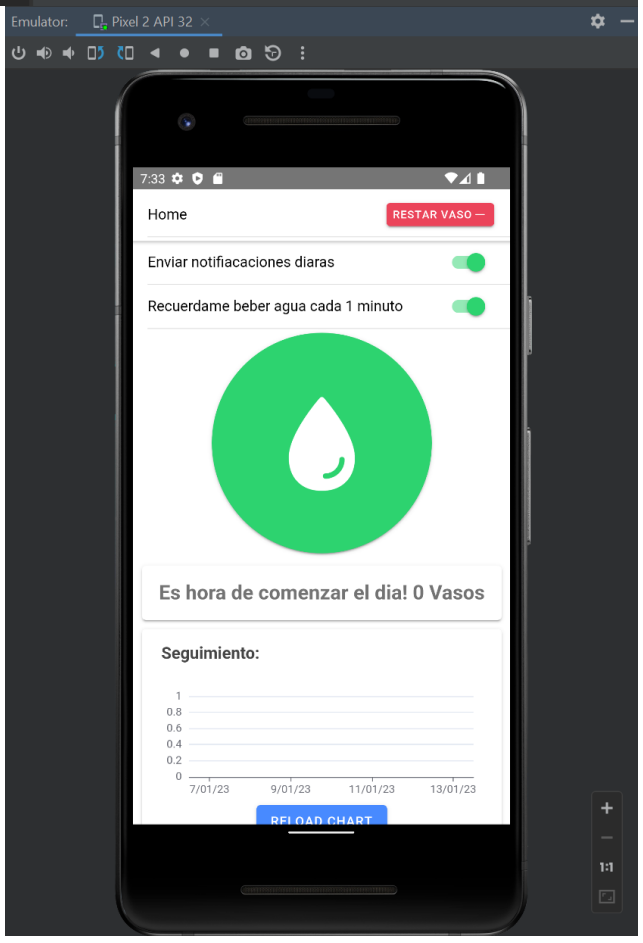
```
ionic build
rm -rf ./android/
npx cap add android
npx cap open android
```

Después de ejecutar los comandos se nos abrirá una ventana de Android Studio, en la parte superior derecha seleccionamos Device Manager y podremos crear un emulador de móvil o conectarnos a uno real.



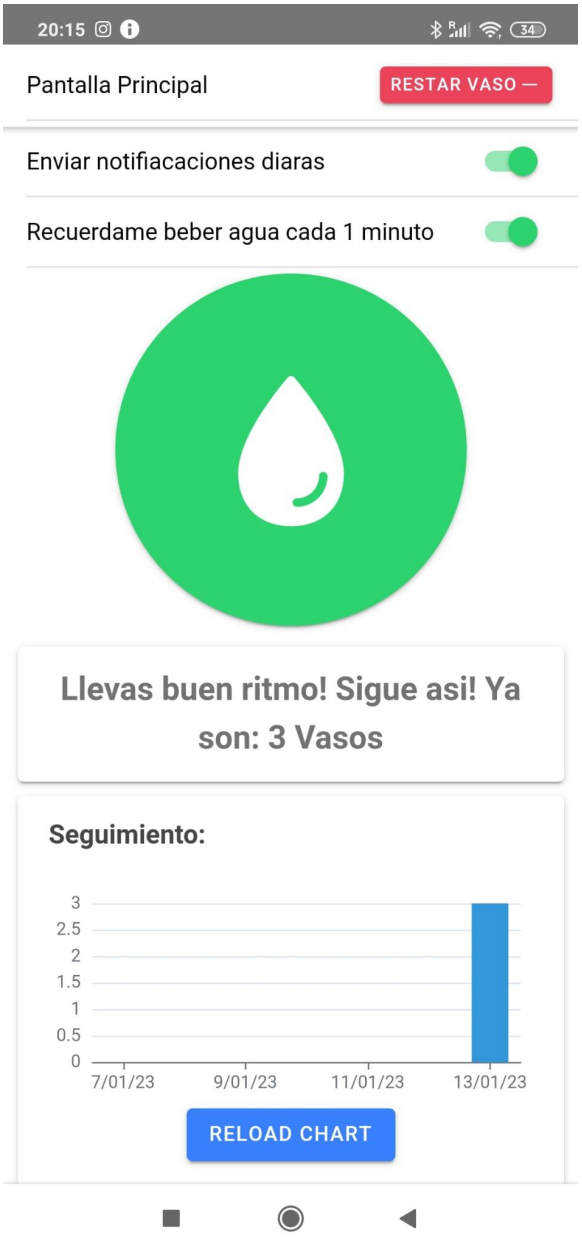


En el caso de un emulador, seguiremos las instrucciones y elegiremos el modelo y sistema operativo de nuestro móvil virtual, al finalizar, podremos ejecutar nuestra aplicación en nuestro emulador:



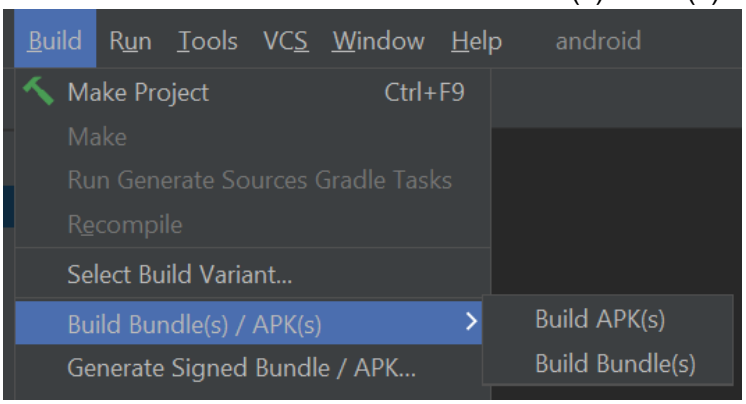
En el caso de conectarnos a un móvil real, primero deberemos activar opciones de desarrollado en nuestro móvil. Puede variar dependiendo de la marca y modelo del móvil, pero en general suele activarse en Settings → About Phone → Build Number, pulsaremos sobre el Build Number 7 veces y se activaran las opciones de desarrollador, luego iremos a System → Developer Options → USB Debugging, y lo activaremos.

Luego en Android Studio desde el mismo menú de administrador de dispositivos podemos conectarnos a un móvil real desde WiFi o con un cable USB. Al ejecutar nuestra aplicación desde Android Studio, se instalará y se abrirá en nuestro dispositivo móvil, no hace falta que sigamos conectados a Android Studio para poder utilizar nuestra app en nuestro móvil, ya que seguirá instalada.

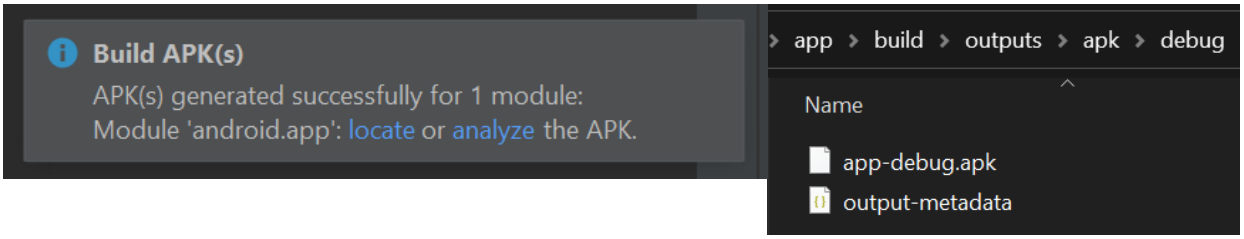


Generar APK

Para generar el APK con Android Studio seleccionaremos la opción de Build APK que se encuentra en el menú Build → Build Bundle(s) / APK(s) → Build APK(s)



Una vez se termina de generar el APK, nos saldrá una notificación en la esquina inferior izquierda, si le damos a “locate” nos abrirá el directorio donde se encuentra el APK.



Una vez tenemos el APK, debemos transferirlo a nuestro móvil (usando un cable USB, por correo, etc), y utilizando un instalador de APKs podremos instalar la aplicación en nuestro móvil.

Bibliografía

UI Components | User Interface Application Building Components (ionicframework.com)
https://capacitorjs.com/docs/android
ngx-echarts - npm (npmjs.com)
ngx-echarts-demo documentation (xieziyu.github.io)
Ionic | Generar Apk de Android con Ionic - YouTube
ionic-team/ionic-storage: Ionic Storage module for Ionic apps (github.com)
Local Notifications Capacitor Plugin API | Capacitor Documentation (capacitorjs.com)
Using with Ionic Framework | Capacitor Documentation (capacitorjs.com)