Creating a File Manager Application in Ionic Capacitor

Creating an Ionic Capacitor mobile app with file management capabilities involves several steps. We'll use the **@capacitor/filesystem** plugin to interact with the device's filesystem, allowing us to create, read, update, and delete files.

Below are the instructions and necessary code snippets for building a simple file manager app using Ionic and Capacitor.

Prerequisites

1. **Node.js** installed on your machine.
2. **Ionic CLI** installed (**npm install -g @ionic/cli**).
3. **Capacitor** installed (**npm install -g @capacitor/cli**).

Step 1: Create a New Ionic Project

First, create a new Ionic project using Angular (or any other framework of your choice):

```
ionic start FileManagerApp blank --type=angular
cd FileManagerApp
```

Step 2: Add Capacitor to Your Project

If Capacitor is not already added to your project, you can add it by running:

```
npm install @capacitor/core @capacitor/cli
```

You will be prompted to enter the app name and app ID (e.g., **com.example.filemanagerapp**).

Step 3: Install the Filesystem Plugin

To enable android support and to manage files on the device, we need the **@capacitor/filesystem** plugin:

```
npm install @capacitor/android
npm install @capacitor/filesystem
```

Step 4: Add Platforms (Android)

Add the platforms you want to target:

```
npx cap add android
```

Step 5: Modify the App Code

Now, let's modify the app to include basic file management functionality.

1. Update home.page.ts

Open **src/app/home/home.page.ts** and add the following code:

```
import { Component } from '@angular/core';
import { Directory, Filesystem } from '@capacitor/filesystem';

@Component({
  selector: 'app-home',
  templateUrl: 'home.page.html',
  styleUrls: ['home.page.scss'],
})
export class HomePage {
  fileName: string = '';
  fileContent: string = '';
  fileList: string[] = [];

  constructor() {}

  async createFile() {
    try {
      await Filesystem.writeFile({
        path: this.fileName,
        data: this.fileContent,
        directory: Directory.Documents,
      });
      alert('File created successfully!');
      this.refreshFileList();
    } catch (error) {
      console.error('Error creating file', error);
    }
  }

  async readFile() {
    try {
      const contents = await Filesystem.readFile({
        path: this.fileName,
        directory: Directory.Documents,
      });
      this.fileContent = contents.data;
      alert('File content loaded!');
    } catch (error) {
      console.error('Error reading file', error);
    }
  }

  async deleteFile() {
    try {
```

```
      await Filesystem.deleteFile({
        path: this.fileName,
        directory: Directory.Documents,
      });
      alert('File deleted successfully!');
      this.refreshFileList();
    } catch (error) {
      console.error('Error deleting file', error);
    }
  }

  async refreshFileList() {
    try {
      const result = await Filesystem.readdir({
        path: '',
        directory: Directory.Documents,
      });
      this.fileList = result.files.map(file => file.name);
    } catch (error) {
      console.error('Error reading directory', error);
    }
  }
}
```

2. Update home.page.html

Open **src/app/home/home.page.html** and replace its content with the following:

```
<ion-header>
  <ion-toolbar>
    <ion-title>File Manager</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <ion-item>
    <ion-label position="floating">File Name</ion-label>
    <ion-input [(ngModel)]="fileName"></ion-input>
  </ion-item>

  <ion-item>
    <ion-label position="floating">File Content</ion-label>
    <ion-textarea [(ngModel)]="fileContent"></ion-textarea>
  </ion-item>

  <ion-button expand="full" (click)="createFile()">Create File</ion-button>
  <ion-button expand="full" (click)="readFile()">Read File</ion-button>
```

```
  <ion-button expand="full" (click)="deleteFile()">Delete File</ion-button>

  <ion-list>
    <ion-item *ngFor="let file of fileList">
      {{ file }}
    </ion-item>
  </ion-list>
</ion-content>
```

3. Update app.module.ts

Ensure that **FormsModule** is imported in **src/app/app.module.ts** to enable two-way data binding with **[(ngModel)]**.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RouteReuseStrategy } from '@angular/router';

import { IonicModule, IonicRouteStrategy } from '@ionic/angular';

import { AppComponent } from './app.component';
import { AppRoutingModule } from './app-routing.module';
import { FormsModule } from '@angular/forms'; // Import FormsModule

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    IonicModule.forRoot(),
    AppRoutingModule,
    FormsModule, // Add FormsModule here
  ],
  providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],
  bootstrap: [AppComponent],
})

export class AppModule {}
```

Step 6: Run the App

For Android:

1. Build the app:

```
ionic build
```

2. Open the Android project in Android Studio:

```
npx cap open android or npx cap run android
```

3. Run the app on an emulator or physical device.

Step 7: Test the App

- You should be able to create, read, and delete files in the **Documents** directory.
- The file list will be refreshed after each operation, showing the current files in the directory.

Additional Notes

- **Permissions** : On Android, you may need to request storage permissions in the **AndroidManifest.xml** file. For iOS, you may need to configure the **Info.plist** file to request access to the file system.

For Android, add the following permissions to **android/app/src/main/AndroidManifest.xml**:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

- **Filesystem API** : The **@capacitor/filesystem** plugin provides a simple API for interacting with the device's filesystem. You can explore more advanced features like appending to files, checking file existence, etc., in the official documentation .

Conclusion

You now have a basic file manager app built with Ionic and Capacitor. This app allows you to create, read, and delete files on the device's filesystem. You can further enhance this app by adding features like file upload/download, folder management, and more.