



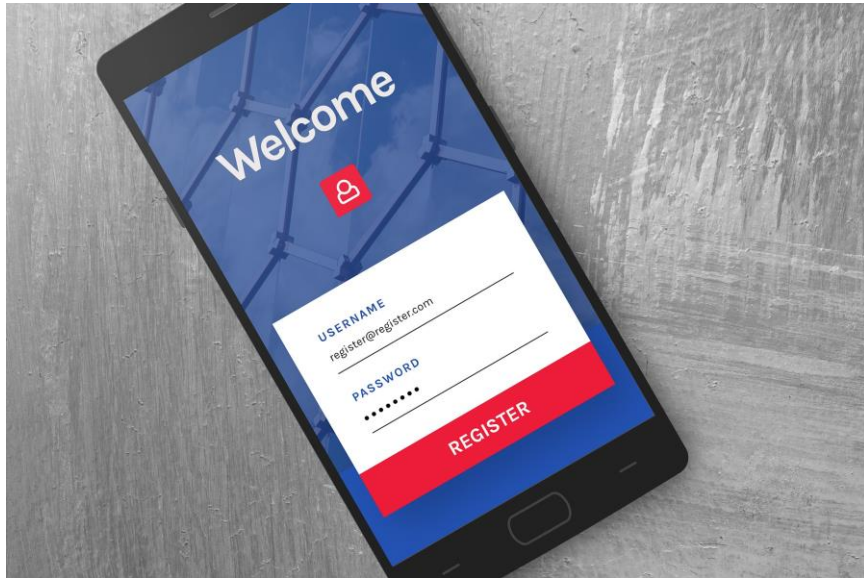
DEVELOPPEMENT MOBILE

M. PESSA MASSENE DAVE ARTHUR

PRESENTATION DU CHAPITRE 4

Les applications mobile qui communiquent avec une base de données locale

SOMMAIRE



- **CONCEPTS DE BD LOCALE**
- **LECTURE, ECRITURE ET SUPPRESSION**
- **EXEMPLES**
- **TRAVAUX PRATIQUES**

CONCEPTS



**IONIC
STORAGE
CRUD**



- Une base de données est dite locale lorsque les informations structurées sont stockées dans la mémoire de l'appareil mobile;
- L'utilisateur n'a pas besoin d'accéder à Internet pour consulter les informations la BD;
- Les opérations de lecture, écriture ou suppression peuvent être utilisé;
- Par contre si l'application est désinstallée, les données seront perdues

CONCEPTS



**IONIC
STORAGE
CRUD**



- Ionic dispose de la bibliothèque `@ionic/storage` utilisé pour stocker les informations en base de données locale mais aussi pour effectuer des opérations de lecture, écriture, suppression, création.
- Il existe plusieurs types de stockage de base de données en locale:
 - **IndexedDB**
 - **SQLite**

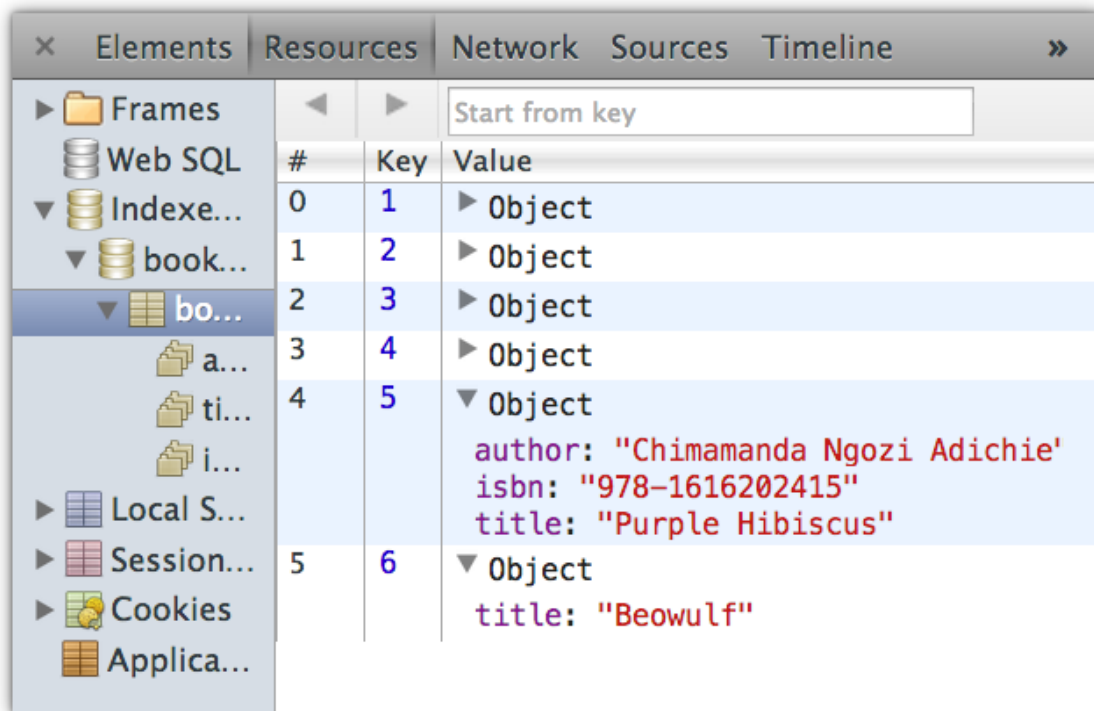
CONCEPTS: INDEXEDDB

IndexedDB



- C'est une API Web utilisé pour le stockage des données côté client.
- C'est aussi un système de gestion de base de données.
- A l'inverse des SGBD relationnelles qui utilise des tables et colonnes, lui il utilise un système de **clé – valeur**, un système orientée objet JavaScript.

CONCEPTS: INDEXEDDB



#	Key	Value
0	1	▶ Object
1	2	▶ Object
2	3	▶ Object
3	4	▶ Object
4	5	▼ Object author: "Chimamanda Ngozi Adichie" isbn: "978-1616202415" title: "Purple Hibiscus"
5	6	▼ Object title: "Beowulf"

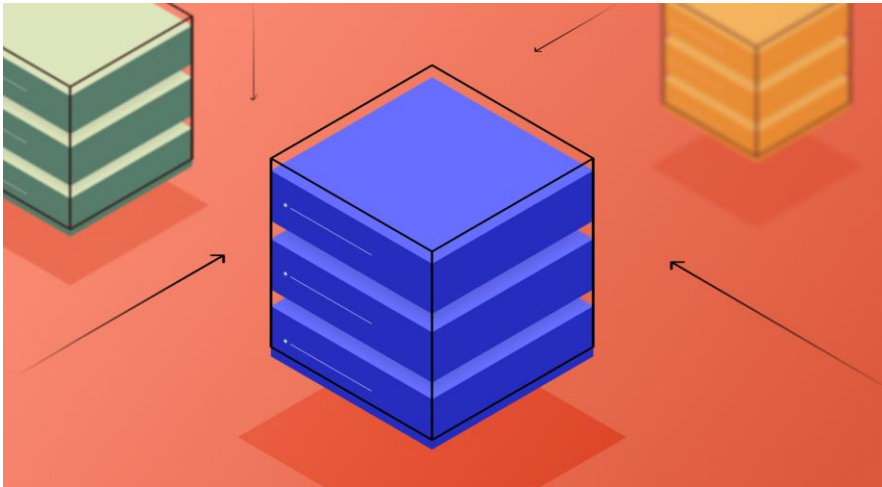
- Les données sont stockées en mémoire locale
- Les opérations (lecture, écriture, ajout) effectuées avec IndexedDB sont asynchrone.
- Le type de stockage est persistant car les données sont stockées pendant une longue période.

CONCEPTS: SQLITE



- SQLite est une base de données relationnelles.
- Ce SGBD ne reproduit pas le schéma client – serveur. Il est directement intégré dans une application
- Les opérations (ajout, lecture, écriture, recherche) sur ce SGBD utilise le langage SQL.
- La taille maximale de stockage en base de données est de 140 Tera octets

CONCEPTS: IONIC STORAGE



- **Ionic Storage** est une blibliothèque regroupant un ensemble de technologies utilisés pour la gestion du stockage des données de manière permanente.
- Il utilise également un système de clé - valeur
- Il prend en charge les SGBD de stockages : IndexedDB, SQLite, ou encore Secure Storage.

CONCEPTS: IONIC STORAGE

Installer et configurer Ionic Storage:

1. Installation via l'invite de commande

```
$ npm install @ionic/storage
```

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

2. Importer le module Ionic Storage dans le fichier main.ts

```
import { IonicStorageModule } from '@ionic/storage-angular';  
import { Drivers } from '@ionic/storage';
```

CONCEPTS: IONIC STORAGE

Installer et configurer Ionic Storage:

3. Ajouter IonicStorageModule dans le tableau « imports »

```
IonicStorageModule.forRoot({  
    name: 'nom_database',  
    driverOrder: [CordovaSQLiteDriver._driver, Drivers.IndexedDB]  
}),
```

4. Créer un service qui sera utilisé pour les tâches de lecture, écriture, ajout, suppression

```
$ ionic g service db
```

CONCEPTS: IONIC STORAGE (SQLITE

Ionic Storage a pour rôle de choisir avec pertinence le type de stockage pour la plateforme approprié. Etant donné que nous travaillons sur des applications mobiles, nous allons utiliser SQLite.

- S'il s'agit d'une application web, alors Ionic Storage choisit IndexedDB
- Si c'est une application mobile alors Ionic storage choisit SQLite comme SGBD

5. Pour intégrer le pilote SQLite pour sa prise en charge par Ionic Storage

Nous allons installer Cordova SQLite

```
$ npm install cordova-sqlite-storage
```

```
$ npm install localforage-cordovasqlitedriver
```

CONCEPTS: IONIC STORAGE (SQLITE

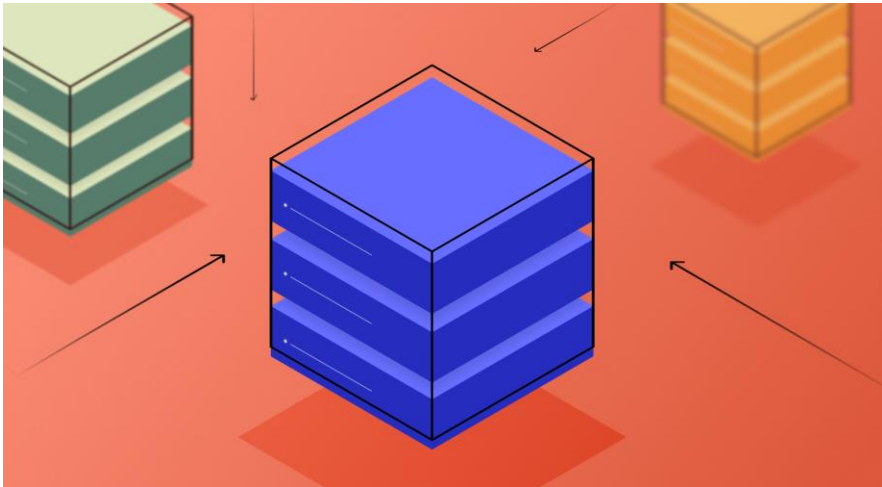
6. Ajouter le pilote SQLite parmi les paramètres du module IonicModuleStorage

```
import * as CordovaSQLiteDriver from 'localforage-cordovasqlitedriver';
```

Ajouter le pilote **CordovaSQLiteDriver._driver** dans le tableau **driverOrder**.

```
IonicStorageModule.forRoot({  
  name: 'nom_database',  
  driverOrder: [CordovaSQLiteDriver._driver, Drivers.IndexedDB]  
}),
```

FONCTIONNALITES: LECTURE, ECRITURE, SUPPRESSION



- Nous allons voir l'usage des méthodes Ionic Storage pour les fonctionnalités de : Lecture, Ecriture et Suppression.
- Mais avant, il est important d'initialiser Ionic Storage (création de la base de données,) afin d'exploiter.
- Cette initialisation est implémenté dans le constructeur de notre service **bdService**.

FONCTIONNALITES: LECTURE, ECRITURE, SUPPRESSION

- Dans le fichier db.service.ts, importer le service IonicStorage ainsi que le pilote SQLite

```
import { Storage } from '@ionic/storage';  
import * as CordovaSQLiteDriver from 'localforage-cordova-sqlitedriver';
```

- Instantier Storage dans le constructeur et définir une méthode asynchrone dans laquelle nous allons insérer les instructions d'initialisation

```
await this.storage.create();  
await this.storage.defineDriver(CordovaSQLiteDriver);
```

FONCTIONNALITE: ECRITURE

La fonction Ecriture

- la méthode **set()** permet d'enregistrer un objet Javascript
- `storage.set('<table>', list_students);` //où list_students est un tableau d'objets ou un objet Javascript et table le nom de la table

Dans notre exemple nous aurons:

```
let list_students = [ {id: 1, name:'Sunshine', class : 'ING5' }, ... ];  
this.storage.set('students', list_students);
```

FONCTIONNALITES: SUPPRESSION

La fonction Suppression

- la méthode (asynchrone) **remove()** permet de supprimer une table dans notre BD.
- `storage.remove('<nom_table>');` //où <nom_table>, le nom de la table à supprimer
- `storage.clear();` //permet de supprimer la base de données

Dans notre exemple nous aurons:

```
this.storage.remove('students');
```

```
this.storage.clear();
```

FONCTIONNALITES: LECTURE

La fonction Lecture

- la méthode (asynchrone) **get()** est utiliser pour lire les informations stockées dans une table dans notre BD. Cette méthode retourne Promise tout comme set(), remove() et clear();
- `storage.get('<nom_table>');` //où <nom_table>, le nom de la table dont nous souhaitons récupérer les données

Dans notre exemple nous aurons:

```
const list_students = await this.storage.get('students');
```

Notes: `list_students` sera un tableau d'objets. Dans le cas contraire, la méthode `get()` va retourner la valeur **undefined** (lorsqu'il n'existe pas une table (clé) du nom « *students* »).

CHAP 4: SQLITE

D'autres part, nous pouvons exploiter les bases de données relationnelles via SQLite.

- A travers le plugin Capacitor SQLite
- La lecture, l'écriture, la modification ou encore la suppression se feront via des requêtes SQL;
- Avec ce plugin, les données seront encryptées conformément aux plateformes Android et iOS;
- Ce plugin utilise SQLCipher pour crypter les informations stockées en base de données.
- <https://github.com/capacitor-community/sqlite>

CHAP 4: SQLITE

- Comme IonicStorage, le plugin Capacitor SQLite est utilisé pour stocker les données de manière permanente;
- **Contexte:** nous voulons construire une mini application pour gérer les étudiants en utilisant une base de données locale;
- Implémentons la démarche suivante :

CHAP 4: SQLITE

- Installer le plugin Capacitor SQLite:

`npm install --save @capacitor-community/sqlite`

- Créer un nouveau service :

`ionic g service providers/sql`

Nous allons procéder à la configuration de la base de données

CHAP 4: SQLITE

- Configuration pour la plateforme iOS. Dans le fichier, capacitor.config.ts, insérer l'objet plugins avec la configuration suivante (afin que la bd soit visible par iTunes et sauvegarder par iCloud):

```
plugins: {  
  CapacitorSQLite: {  
    "iosDatabaseLocation": "Library/CapacitorDatabase"  
  }  
}
```

CHAP 4: SQLITE

- Sous « Android », la base de données est sauvegardé dans « `data/data/YOUR_PACKAGE/databases` »
- Le plugin va ajouter un suffixe « SQLite » au nom de la base de donnée. Le nom de la base de données devrait ressembler à: **etudiant_dbSQLite.db**; où « etudiant_db » est le nom que vous avez défini
- Sur le Web, les données sont sauvegardées via le SGBD IndexedDB

CHAP 4: SQLITE

- Editons le service **sql.service.ts**
- Définir le nom de la base de données
- **private readonly DB_NAME: string = "students_db";**
- Définir également l'interface étudiant ayant les propriétés nom, niveau, filière, age

CHAP 4: SQLITE

- **Etape 2:**

- Définir la propriété qui sera affecté à l'objet SQLConnection

```
private sqlObjet : SQLiteConnection = new  
SQLiteConnection(CapacitorSQLite);  
  
private sqlDB!: SQLiteDBConnection;  
  
private etudiants: any = signal<Etudiant[]>([]);
```

- NB: ne pas oublier d'importer les différents objets du plugin SQLite

CHAP 4: SQLITE

- Etape 3 : initialiser la base de données
- Définir la méthode pour initialiser la base de données

```
async initializeDB(){  
    this.sqlDB = await  
this.sqlObjet.createConnection(this.DB_NAME,  
false, "no-encryption", 1, false);  
    await this.sqlDB.open();  
}
```


CHAP 4: SQLITE

- Etape 4 : créer une table

```
async createTable(){  
  const schema = `CREATE TABLE IF NOT EXISTS etudiants(  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    nom TEXT NOT NULL,  
    filiere TEXT,  
    niveau INTEGER DEFAULT 1  
    age INTEGER NOT NULL  
  )`;  
  this.sqlDB.execute(schema);  
}
```

CHAP 4: SQLITE

- Etape 4 : récupérer les données d'une table

```
async retrieveData(){  
    const data = await this.sqlDB.query('SELECT * from  
etudiants');  
    this.etudiants.set(data.values || []);  
}
```

CHAP 4: SQLITE

- **Etape 5 : Appeler la méthode d'initialisation (app.component.ts).** La méthode `initializeApp()` est appelé dans le corps du constructor

```
async initializeApp(){  
    await this.sqlServ.initializeDB();  
    SplashScreen.hide();  
}
```

CHAP 4: SQLITE

Etape 6 : Insérer des données dans la BD

```
async addEtudiant(obj_etudiant: Etudiant){  
    const req = `INSERT INTO etudiants  
(nom,filiere,niveau,age) VALUES ('${obj_etudiant.nom}',  
    '${obj_etudiant.filiere}', ${obj_etudiant.niveau},  
    ${obj_etudiant.age})`;   
    const resultat = await this.sqlDB.query(req);  
    return resultat;  
}
```

CHAP 4: SQLITE

Etape 7 : Modifier des données dans la BD

```
async updateEtudiantById(id: number, niveau: number){  
    const req = `UPDATE etudiants SET niveau=${niveau}  
WHERE id=${id}`;  
    const resultat = await this.sqlDB.query(req);  
  
    return resultat;  
}
```

CHAP 4: SQLITE

Etape 8 : Supprimer des données dans la BD

```
async deleteEtudiantById(id: number){  
    const req = `DELETE FROM etudiants WHERE id=${id}`;  
    const resultat = await this.sqlDB.query(req);  
  
    return resultat;  
}
```


CHAP 4: SQLITE

Etape 9 : Configurer la vue étudiants

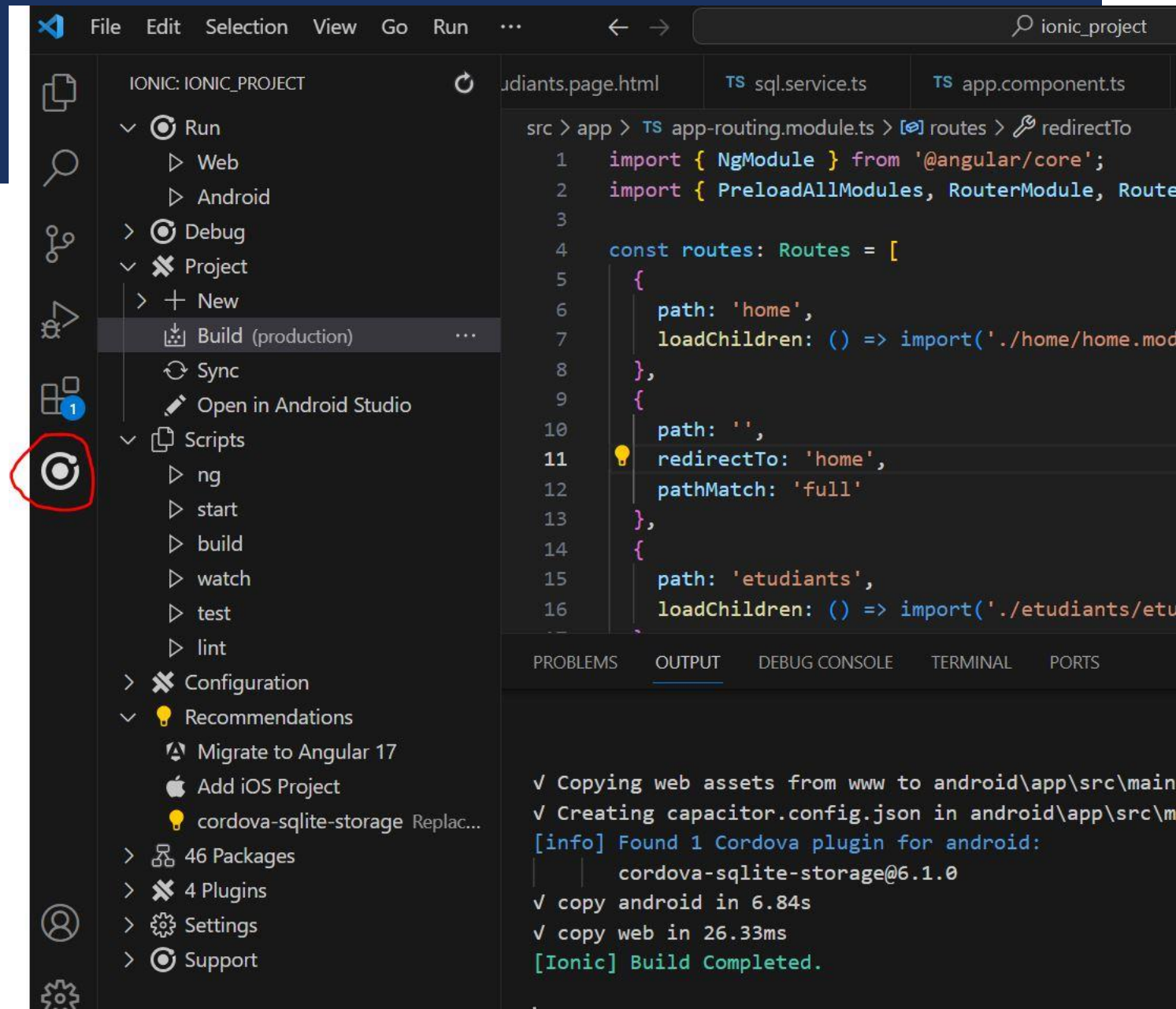
- Implémenter la fonctionnalité liste étudiants
- Enregistrer ou Modifier un étudiant : Vous pouvez utiliser le composant UI Ion-Modal de Ionic
- Supprimer un étudiant

CHAP 4: IONIC VS CODE

- Ionic dispose d'un plugin qui facilitera l'aide au développement
- Les différentes fonctions ou tâches telles que l'exécution, la compilation, la synchronisation, la configuration d'un projet Ionic, ainsi que les tests depuis un terminal mobile, sont réunies en une seule extension
- Ionic VS Code :
<https://marketplace.visualstudio.com/items?itemName=ionic.ionic-vs-code>

CHAP 4: IONIC VS CODE

- Via l'IDE VS, après avoir installer l'extension, il faudra redémarrer VS. Ensuite vous verrez l'icône Ionic. Ce menu propose un ensemble de fonctionnalités



CHAP 4: TD

Nous allons exploiter notre projet sur la liste des fruits. Il sera question de ne plus exploiter les fichiers json ou encore le tableau de fruits stockés dans le fichier ts.

Exercice 1: Créer un service db dont le rôle sera d'implémenter les fonctionnalités de lecture, écriture et suppression dans la base de données

Exercice 2: Ajout d'un fruit

Sur la liste des fruits, Ajouter un bouton + au niveau de l'en-tête. Lorsque l'utilisateur va cliquer sur ce bouton, ceci va ouvrir un *ModalController* (composant UI Ionic): il s'agit d'un formulaire qui va servir à ajouter un fruit. Le fruit aura les propriétés id, nom, description

CHAP 4: TD

Exercice 3: Liste des fruits

Récupérer la liste des fruits qui sont stockées dans la base de données. Lorsque je clique sur un élément de la liste, ceci m'ouvre un `ActionSheet` avec 2 actions : « supprimer » et « modifier ».

- Supprimer va retirer l'élément de la liste des fruits ainsi que dans la BD
- Modifier va modifier l'élément de la liste des fruits ainsi qu'au niveau de la BD



THANK YOU

ARTHUR.PESSA@INSTITUTSAIN
TJEAN.ORG