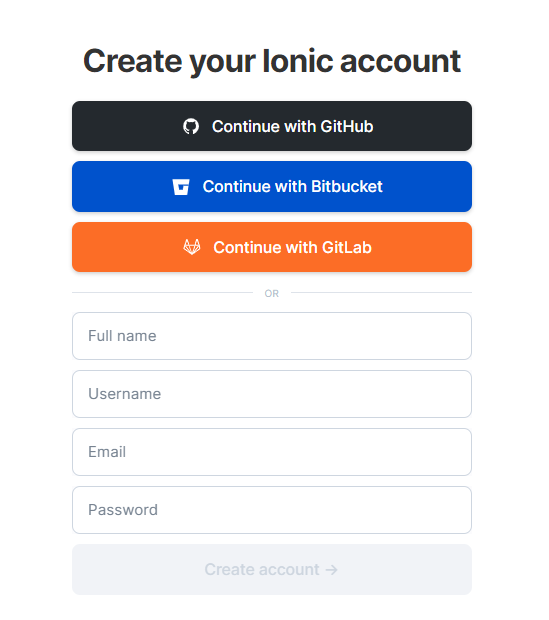
IONIC/Angular

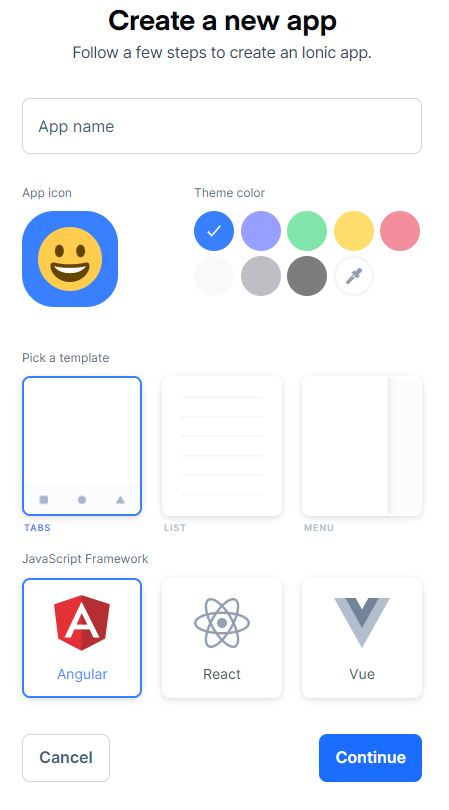
1/ Installer ionic

* Créer un dossier pour les applications ionic nommé appli-ionic
* Se rendre sur le site **ionicframework.com**
* Cliquer sur Get started
* Se connecter avec les identifiants GitHub ou créer un compte

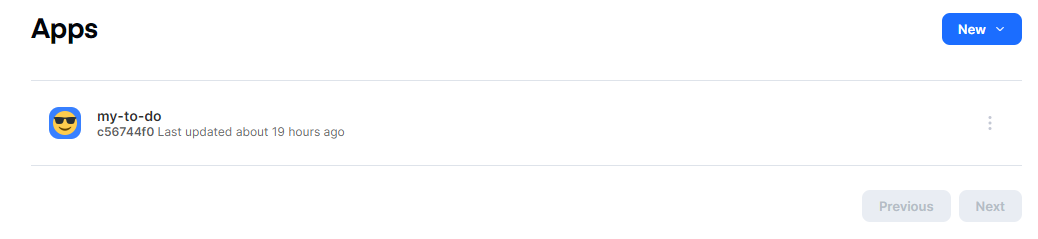


2/ Créer une nouvelle application.

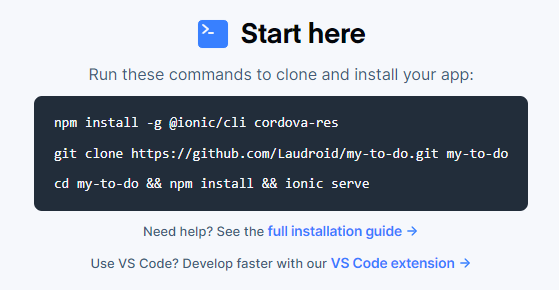
* Donner un nom et choisir le template **tabs** et le framework **Angular.** Puis Continue.



* Dans le tableau de bord, cliquer sur l’application créée.



* Lancer git-bash et suivre les instructions qui s’affichent à l’écran :

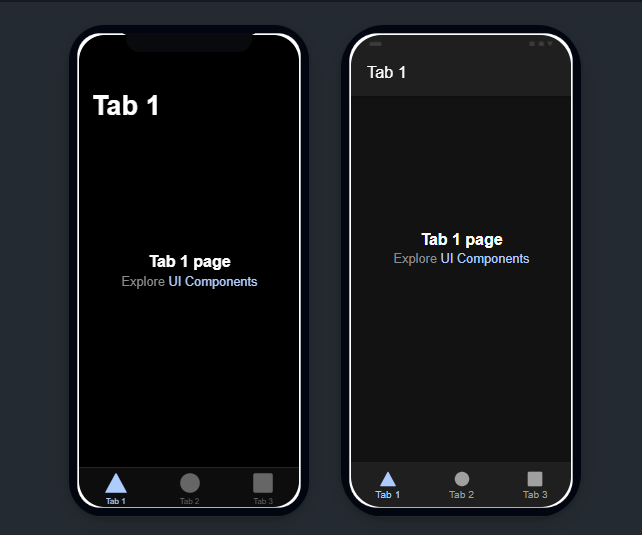


La dernière commande ionic serve crée un serveur local à l’adresse <http://localhost:8100/> et lance automatiquement l’appli dans un navigateur.

Il existe une option à cette commande. Dans la console, sous le répertoire de l’application entrer les commandes :

* npm install @ionic/lab
* ionic serve -l

L’appli est lancée en mode lab. Il est alors possible de visualiser sur le même écran les versions Android et iOS de l’appli. Par défaut, le modèle tabs contient 3 onglets. Il est possible d’en rajouter. Le routage est réalisé depuis le fichier **tabs-routing.module.ts**



NB : les icones en bas d’écran peuvent être modifiées dans le fichier **tabs.page.html**. Les icones **ionic** sont disponibles à <https://ionic.io/ionicons>

3/ Création d’une nouvelle appli par lignes de commande.

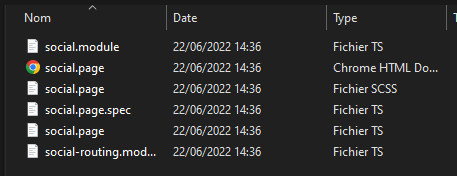
Il existe différents modèles d’appli dans Ionic/Angular. Pour la suite, nous utiliserons le modèle **blank.** Nous allons créer une appli avec 3 g s.

* Sous le répertoire **appli-ionic**, depuis la console, entrer la commande :

ionic start Navigation blank

* Dans le répertoire navigation, lancer le serveur en entrant ionic serve
* Dans le répertoire de l’app créer la page ‘social’ : ionic g page social
* Dans le répertoire de l’app créer la page ‘about’ : ionic g page about

Vérifier qu’un nouveau répertoire est créé sous **src/app/social** et contient :



Même chose pour **src/app/about**

On note la présence de fichiers .ts (typescript). Le langage Typescript est un langage compilé orienté objet basé sur Javascript. Autrement dit Typescript est un sur-ensemble de Javascript :

**Typescript = Javascript + vérification de type + Classe + Héritage**

Les routes de l’appli sont situées dans le fichier **/src/app/app-routing.module.ts**. Ionic a ajouté deux nouvelles routes correspondant aux pages **social** et **about**

* Modifier la page d’accueil pour introduire les routes vers les pages :

<ion-header>

  <ion-toolbar>

    <ion-title>

      Test navigation

    </ion-title>

  </ion-toolbar>

</ion-header>

<ion-content>

  <div *id*="container">

    <ion-item *routerLink*="/social">

      <ion-icon *slot*="start" *color*="medium" *name*="logo-facebook"></ion-icon>

      <ion-label>Réseaux sociaux</ion-label>

    </ion-item>

    <ion-item *routerLink*="/about">

      <ion-icon *slot*="start" *color*="medium" *name*="logo-twitter"></ion-icon>

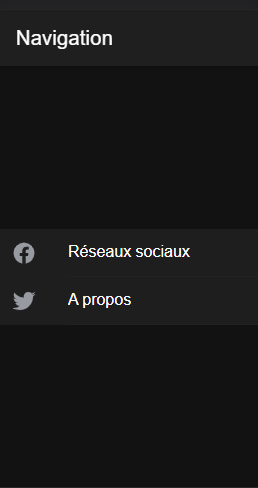
      <ion-label>A propos</ion-label>

    </ion-item>

  </div>

</ion-content>

* Tester l’application



Depuis une page, pour revenir à la page d’accueil, il est nécessaire de rajouter un bouton « retour »

* Dans les pages **social** et **about**, insérer un élément ion-back-button dans la barre de navigation, au-dessus de la balise ion-title :

    <ion-buttons *slot*="start">

      <ion-back-button *defaultHref*="/home" *text*="Accueil"></ion-back-button>

    </ion-buttons>

4/ Gestion des feuilles de style

Nous allons réaliser une ToDoList.

* Créer une nouvelle appli dans le répertoire : **appli-ionic**

ionic start Todolist blank

* Choisir le framework par défaut : Angular
* Changer le titre de l’appli par **Ionic Todolist** (dans home/home.page.html)
* Dans le fichier **src/theme/variable.scss**, remplacer la couleur primary

*:root* {

  /\*\* primary \*\*/

  --ion-color-primary: #5a8ca4;

* Et l’appliquer à l’entête :

<ion-toolbar *color*="primary">

* Placer le fichier **bg.jpg** dans le répertoire **src/assets**.
* Créer une boite **div** dans le bloc ion-content et lui affecter la classe bg-img

<ion-content>

  <div *class* = "bg-img">

  </div>

</ion-content>

* Dans le fichier [home.page.scss](http://home.page.scss) ajouter le style :

*.bg-img* {

  padding: 1px;

  height: 100%;

  width: 100%;

  background-image: url('../../assets/bg.jpg');

  background-repeat: no-repeat;

  background-size: cover;

}

* Ajouter la date du jour dans la balise de classe **bg-img**

<ion-content>

  <div *class* = "bg-img">

    <h1><b>Aujourd'hui</b></h1>

    <h3>{{ currentDate }}</h3>

  </div>

</ion-content>

* Définir la propriété currentDate dans le fichier **home/home.page.ts**, dans la classe HomePage

 currentDate: *string*;

* Dans le constructeur de la classe ajouter une const définissant les options de format :

   const timeformat: *Intl*.*DateTimeFormatOptions* = {

      weekday: 'short',

      year: 'numeric',

      month: 'short',

      day: 'numeric',

      hour12: false

    };

*this*.currentDate = **new** *Date*().toLocaleTimeString('fr-FR', timeformat);

* Modifier la feuille de style :

h1 {

  color: white;

  margin-left: 20px;

  font-size: 30px;

}

h3 {

  color: #e8e8e8;

  margin-left: 20px;

  margin-top: -7px;

  font-size: medium;

  margin-bottom: 15px;

  text-transform: capitalize;

}

5/ Connexion à une base de données Firebase :

Firebase est un service de base de données **propriété de Google**, orienté développement mobile. Il s’agit d’une base no-SQL. Il n’y a pas de notion de colonne et lignes comme dans SQL. Les données sont stockées dans un document.

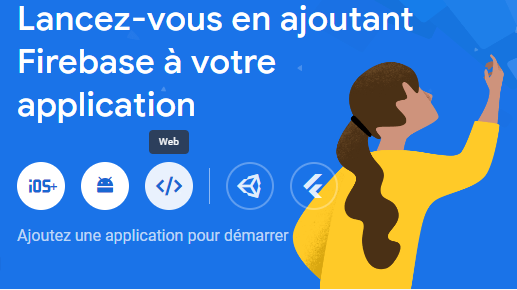
* Se rendre sur le **site de Firebase** pour créer un nouveau projet

<https://console.firebase.google.com/>

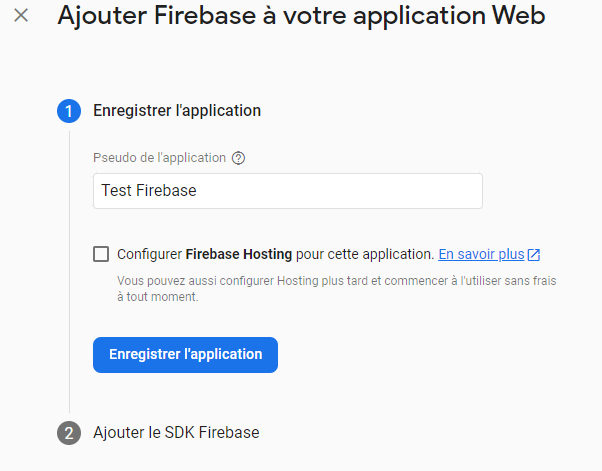
* Cliquer sur ‘Ajouter un projet’
* Entrer un nom de projet.



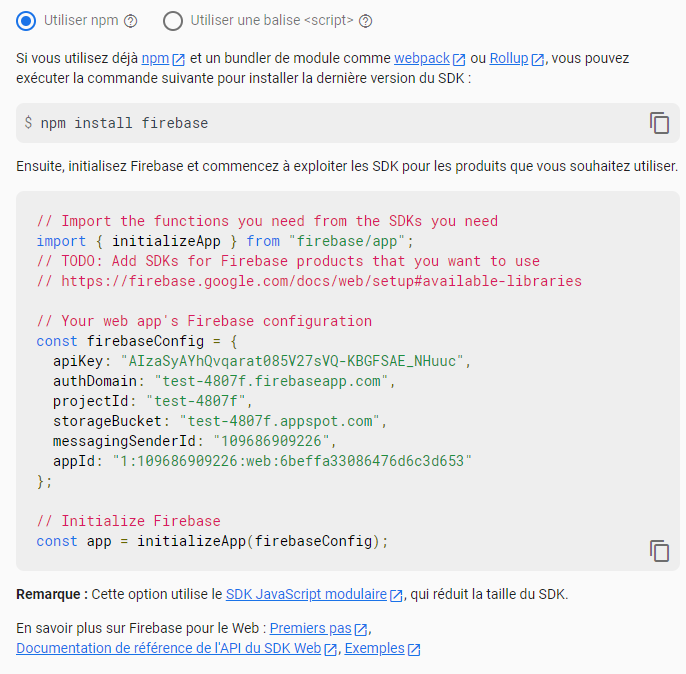
* Cliquer sur ‘Continuer’. Page suivante, désactiver Google Analytics pour ce projet, puis cliquer sur ‘Créer un projet’
* Ajouter Firebase à l’appli en cliquant sur le bouton Web **</>**



* Page suivante, répéter le nom du projet. Ne pas cocher la case ‘Configurer Firebase Hosting’. Cliquer sur ‘Enregistrer l’application’



* Page suivante, cocher ‘Utiliser npm’



Il est possible d’atteindre cet écran en cliquant sur la roue crantée à droite de l’onglet ‘vue d’ensemble du projet’ dans la page d’accueil du projet Firebase, puis ‘Paramètres du projet.

* Installer le plugin **AngularFire** dans le dossier de l’appli Ionic au moyen de la commande suivante :

npm install firebase @angular/fire

* Modifier le fichier **src/app/app.module.ts** en y important deux modules contenant les méthodes qui permettent de communiquer avec la base FireBase.

import { AngularFireModule} from '@angular/fire/compat'

import { AngularFireDatabaseModule } from '@angular/fire/compat/database';

et une const de configuration dans laquelle sont placées les valeurs obtenues sur l’écran de paramètres :

export const firebaseConfig = {

  apiKey: "AIzaSyBE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*88QYj9JjC8",

  authDomain: "ionictodolist-\*\*\*\*\*\*.firebaseapp.com",

  databaseURL: "https://ionictodolist-\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.firebasedatabase.app",

  projectId: "ionictodolist-\*\*\*\*\*",

  storageBucket: "ionictodolist-\*\*\*\*\*\*.appspot.com",

  messagingSenderId: "553\*\*\*\*\*\*\*\*\*",

  appId: "1:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*c6d47a806733"

};

* Ajouter les modules dans les imports (même fichier)

@NgModule({

  declarations: [AppComponent],

  imports: [

    BrowserModule,

    IonicModule.forRoot(),

    AppRoutingModule,

    AngularFireModule.initializeApp(*firebaseConfig*),

*AngularFireDatabaseModule*

],

* Modifier [home.page.ts](http://home.page.ts) en y important ce module :

import { AngularFireDatabase } from '@angular/fire/compat/database';

et modifier le constructeur en passant le module par injection de dépendance de façon à pouvoir l’utiliser dans toute la page.

constructor(public *afDB*: AngularFireDatabase) {

L’appli Ionic est maintenant associée au projet Firebase grâce au plugin AngularFire

6/ Ajouter un formulaire :

* Utiliser la directive [(ngModel)] pour construire le formulaire dans **home.page.html**. Le champ de saisie est associé à la variable myTask et se place sous la balise <h3>:

<ion-card>

 <ion-item *lines*="none">

   <ion-input *[(ngModel)]*="myTask" *placeholder*="Entrer une tâche"></ion-input>

 </ion-item>

</ion-card>

* Ce champ texte est associé à la propriété myTask à déclarer dans le fichier [home.page.ts](http://home.page.ts) dans la classe HomePage.

myTask = '';

* Ajouter un bouton à droite de la barre de saisie (après la balise <ion-input>). Au clic, le nom de la tâche est envoyé à la base FireBase par la méthode addTaskToFirebase() :

    <ion-button *(click)*="addTaskToFirebase()" *shape*="round" *slot*="end">

      <ion-icon *slot*="icon-only" *name*="add"></ion-icon>

    </ion-button>



* Ajouter la fonction addTaskToFirebase() dans la classe HomePage

    addTaskToFirebase() {

*this*.afDB.list('Tasks/').push({

        text: *this*.myTask,

        date: **new** *Date*().toISOString(),

        checked: false

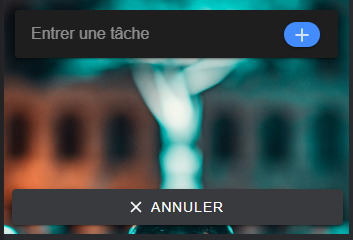
      });

   }

Cette méthode fait appel à l’objet this.afDB (connexion à la base FireBase liée à l’appli) auquel est appliqué la méthode list(‘Tasks/’). Cette méthode se connecte au document ‘Tasks’. A l’objet représentant cette table est appliquée la méthode push() qui attend un objet JSON. L’objet contient le nom des attributs du document (l’équivalent des colonnes en SQL) et les valeurs à insérer en base : le nom de la tâche, l’heure et son statut (false par défaut). AngularFireBase se charge de créer le document et les attributs à la volée.

7/ Afficher / masquer le formulaire

L’idée est de créer un bouton en bas de l’écran qui permet d’afficher le formulaire pour saisir une tâche ou annuler la saisie :

* Dans la classe HomePage, créer une propriété **addTask** (booléen)

  addTask: *boolean*;

* Ajouter dans la même classe, une méthode showForm() dont l’action principale est de renvoyer l’inverse du booléen addTask :

  showForm() {

*this*.addTask = !*this*.addTask;

*this*.myTask = '';

  }

Cette fonction est appelée dans la méthode addTaskToFirebase() de la classe HomePage

Modifier [home.page.ts](http://home.page.ts) en conséquence :

    addTaskToFirebase() {

*this*.afDB.list('Tasks/').push({

        text : *this*.myTask,

        date : **new** *Date*().toISOString(),

        checked : false

      });

*this*.showForm();

   }

Dans le fichier [home.page.html](http://home.page.html), la zone de saisie est affichée dans une card <ion-card>

Documentation sur les cards : [**https://ionicframework.com/docs/api/card**](https://ionicframework.com/docs/api/card)

* Ajouter un test conditionnel **\*ngIf** sur la balise ion-card du formulaire :

<ion-card  *\*ngIf*="addTask">

      <ion-item *lines*="none">

        <ion-input *[(ngModel)]*="myTask" *placeholder*="Entrer une tâche"></ion-input>

        <ion-button *(click)*="addTaskToFirebase()" *shape*="round" *slot*="end">

          <ion-icon *name*="add" *slot*="icon-only" ></ion-icon>

        </ion-button>

      </ion-item>

    </ion-card>

Pour le bouton en bas d’écran (class = ‘’add-button’’), il faut coder deux boutons dont l’affichage du texte et de l’icône est soumis à la valeur du booléen addTask. Au clic sur ces boutons, la fonction showForm() est appelée.

    <ion-button *\*ngIf*="!addTask" *(click)*="showForm()" *expand*="block" *class*="add-button">

      <ion-icon *name*="add" *slot*="start"></ion-icon>

      Ajouter une tâche

    </ion-button>

    <ion-button *\*ngIf*="addTask" *(click)*="showForm()" *expand*="block" *class*="add-button">

      <ion-icon *name*="close" *slot*="start"></ion-icon>

      Annuler

    </ion-button>

Modifier la feuille style en ajoutant :

*.add-button* {

  --background: #383a3e;

  position: absolute;

  bottom: 5px;

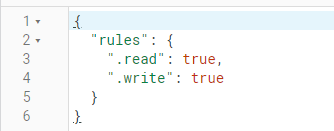
  width: 96%;

  --color: #ffffff;

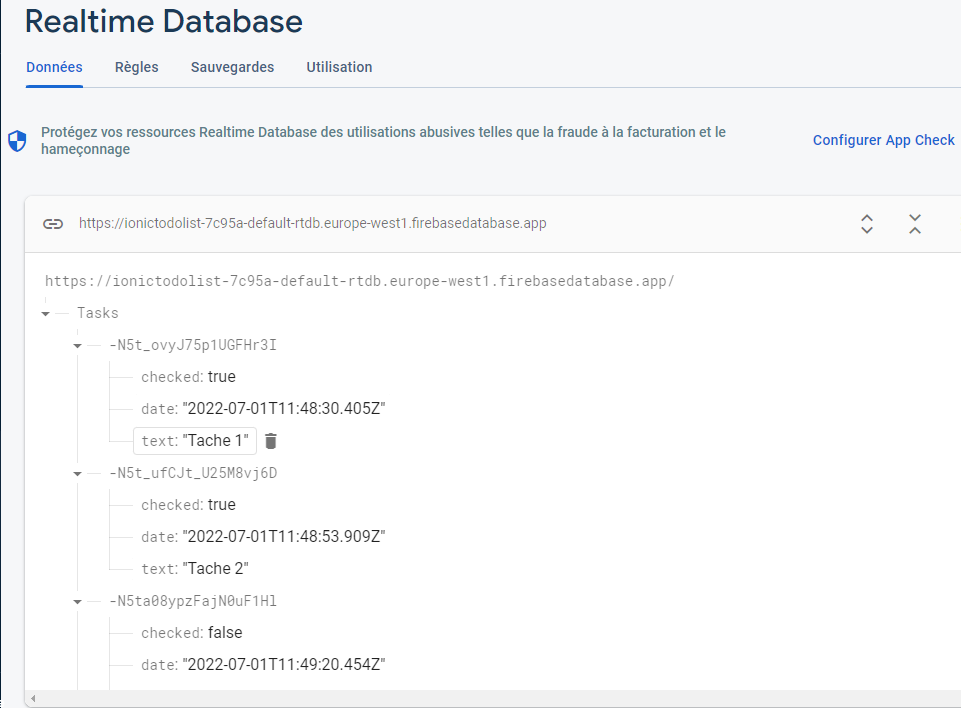
  margin-left: 2%;

}

Dans la console firebase, cliquer sur l’onglet Realtime Database > Règles, modifier les règles pour donner les privilèges de lecture et d’écriture :



Tester la création de tâches en base en allant sur l’interface Firebase à l’onglet Realtime Database > Données :



8/ Gestion des tâches :

La fonction getTasks() récupère les tâches depuis le document ‘/Tasks’ de la Firebase.

this.afDB.list(‘Tasks/’).snapshotChanges() retourne un objet Observable correspondant à un instantané (snapshot) de la base, et plus précisément du document Tasks/ qui contient toutes les tâches.

On applique à cet objet la méthode subscribe() qui attend pour paramètre la fonction de callback pour récupérer les entités depuis la Firebase et les envoyer dans la propriété this.tasks en vue de leur affichage. Penser à vérifier que la propriété tasks est bien définie dans la classe HomePage.

getTasks() {

*this*.afDB.list('Tasks/').snapshotChanges().subscribe(*actions* => {

*this*.tasks = [];

*actions*.forEach(*action* => {

*this*.tasks.push({

          key: *action*.key,

          text: *action*.payload.exportVal().text,

          hour: *action*.payload.exportVal().date.substring(11, 16),

          checked: *action*.payload.exportVal().checked

        });

      });

    });

  }

Les tâches sont affichées les unes en dessous des autres au moyen d’une balise <ion-card></ion-card> et au moyen d’une boucle \*ngFor= ‘let task of tasks’ .

* Afficher la description et l’heure de chaque tâche.

<ion-card  *\*ngFor*="let task of tasks">

        <ion-item *lines*="none">

          <ion-label>

            <h2 *\*ngIf*="!task.checked">{{ task.text }}</h2>

            <p>{{ task.hour }}</p>

          </ion-label>

        </ion-item>

    </ion-card>

* Ajouter l’appel à la méthode d’affichage dans le constructeur :

    *this*.getTasks();

* Ajouter une case à cocher à droite (slot=’end’) de chaque tache permettant de changer son statut. Au clic, la fonction changeCheckState(task) est appelée :

<ion-checkbox *(ionChange)*="changeCheckState(task)" *color*="success" *[(ngModel)]*="task.checked" *slot*="end"></ion-checkbox>

La méthode changeCheckState() effectue la mise à jour du document dans la Firebase.

  changeCheckState(*ev*: *any*) {

*this*.afDB.object('Tasks/' + *ev*.key + '/checked/').set(*ev*.checked);

  }

* Modifier la balise <ion-label> de façon à ce que la tâche soit barrée quand la case est cochée :

<ion-label>

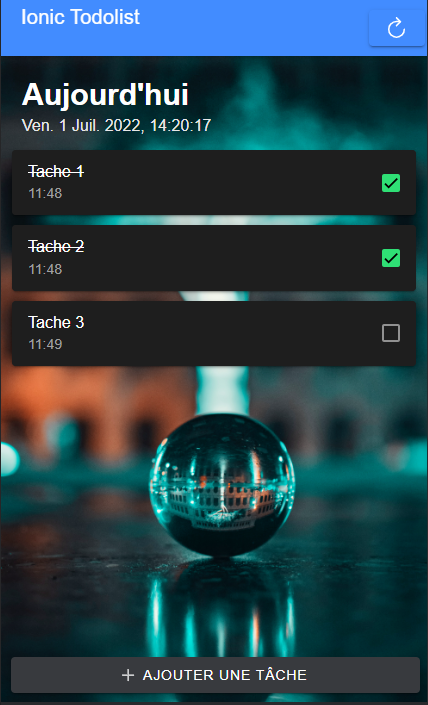
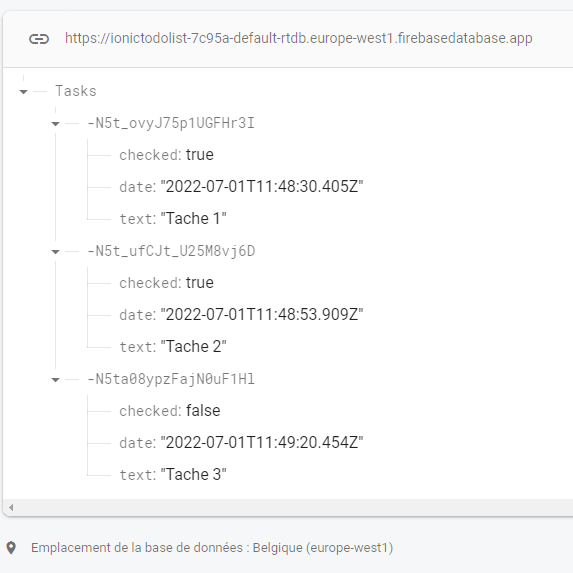
   <h2 *\*ngIf*="!task.checked">{{ task.text }}</h2>

  <h2 *\*ngIf*="task.checked" *style*="text-decoration:line-through;">{{ task.text }}</h2>

   <p>{{ task.hour }}</p>

</ion-label>

Dans la console Firebase, onglet Realtime Database, on peut voir ce que contient la base :

Pour la suppression des tâches, la méthode dans le fichier [home.page.ts](http://home.page.ts) s’écrit :

  deleteTask(*task*: *any*) {

*this*.afDB.list('Tasks/').remove(*task*.key);

  }

Pour la partie front, utiliser une balise <ion-item-sliding></ion-item-sliding> qui permet de faire glisser la tâche à l’écran. Placer une corbeille à l’extrémité gauche de la tâche (<ion-item-option side=’start’>). Elle apparait en faisant glisser la tache vers la droite. Au clic sur la corbeille, on appelle la fonction deleteTask().

* Modifier la balise <ion-card></ion-card> :

    <ion-card  *\*ngFor*="let task of tasks">

      <ion-item-sliding>

        <ion-item-options *side*="start">

          <ion-item-option *color*="danger" *(click)*="deleteTask(task)">

            <ion-icon *name*="trash" *slot*="icon-only"></ion-icon>

          </ion-item-option>

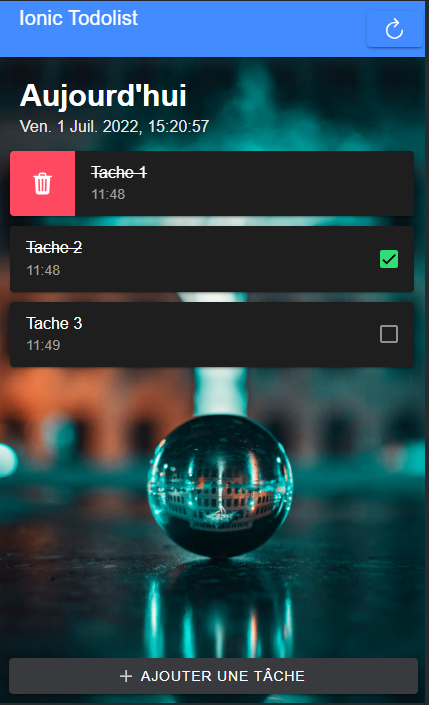
        </ion-item-options>

        <ion-item *lines*="none">

…

      </ion-item-sliding>

    </ion-card>



9/ CRUD PHP MySQL sous Ionic/Angular

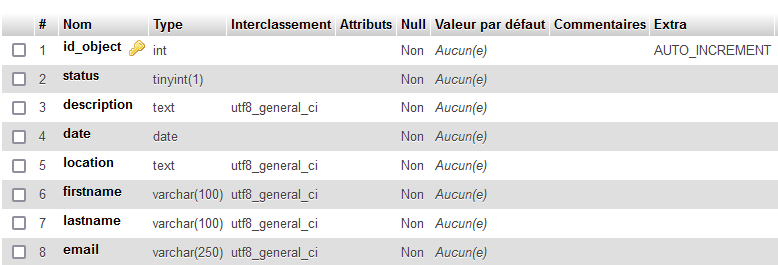
Il n’est pas forcément commode ou souhaité d’avoir ses bases de données sous serveurs Google.

Le but de ce projet est de créer une application Ionic/Angular qui adresse une base MySQL.

Un utilisateur signale qu’il a perdu ou trouvé un objet en renseignant un formulaire. Les déclarations des utilisateurs sont stockées en base de données. Ils peuven consulter la liste des objets perdus et des objet trouvés.

Depuis phpMyAdmin, créer une base ionicfoundlost qui contient une table foundlost. La table contient :

* Un identifiant (id\_object) unique et auto-incrémenté de type int(3)
* Un statut (status) de type BOOLEAN (0=perdu, 1=trouvé)
* Une description (description) de type TEXT
* Une date (date) de type DATE
* Un lieu (location) de type TEXT
* Un prénom (firstname) VARCHAR(100)
* Un nom (lastname) VARCHAR(100)
* Un email (email) VARCHAR(250)



* Créer un répertoire sous Laragon/www nommé ionicserver
* Dans ce répertoire, créer deux fichiers PHP

**manage-data.php** pour gérer l’ajout, la mise à jour et la suppression des enregistrements de la table foundlost.

**retrieve-data.php** pour la lecture des données de la table foundlost.

<?php

header('Access-Control-Allow-Origin: \*');

header('Access-Control-Allow-Headers: Content-Type');

TODO : Définir les paramètres de connexion à la base

TODO : Créer une instance de la classe PDO

// Récupérer le paramètre d’action de l’URL du client depuis $\_GET[‘key’]

// et nettoyer la valeur

$key = strip\_tags($\_GET['key']);

// Récupérer les paramètres envoyés par le client vers l’API

$input = file\_get\_contents('php://input');

if (!empty($input)) {

    $data = json\_decode($input, true);

    $description = strip\_tags($data['description']);

$status = strip\_tags($data['status']);

    $date = strip\_tags($data['date']);

    $location = strip\_tags($data['location']);

    $firstname = strip\_tags($data['firstname']);

    $lastname = strip\_tags($data['lastname']);

    $email = strip\_tags($data['email']);

// En fonction du mode d’action requis

switch ($key) {

        //Ajoute un nouvel enregistrement

    case "create":

        TODO : Filtrer les valeurs entrantes

        TODO : Préparer la requête dans un try/catch

        break;

        // Mettre à jour un enregistrement existant

    case "update":

        TODO : Nettoyer les valeurs en provenant de l’URL client

        TODO : Préparer et exécuter la requête (dans un try/catch)

        break;

        // Supprimer un enregistrement existant

    case "delete":

        TODO : Nettoyer les valeurs de l’URL client (id\_task)

        TODO : Préparer et exécuter la requête (dans un try/catch)

        break;

} // fin switch

} // fin if

Et le fichier retrieve-data.php

<?php

header('Access-Control-Allow-Origin: \*');

TODO : Définir les paramètres de connexion

TODO : Créer une instance de la classe PDO (connexion à la base)

TODO : Prépare et exécute la requête de lecture de la table (try/catch)

NOTE : les headers en début de fichiers permettent de s’affranchir de la « CORS policy » qui empêche deux serveurs différents de communiquer par défaut.

Passons à la réalisation du frontend :

* Créer un nouveau répertoire : ionicfoundlost
* Dans ce répertoire, créer une nouvelle application :

Ionic start foundlost blank

L’appli comporte 6 pages :

* Accueil – home.page
* J’ai trouvé (un objet) - found.page
* J’ai perdu (un objet) – lost.page
* Détail de la fiche objet – viewentry.page
* Liste des objets trouvés – foundlist.page
* Liste des déclarations des objets perdus – lostlist.page
* Modifier le fichier app.module.ts de façon à importer tous les modules nécessaires :

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 { HttpClientModule } from  '@angular/common/http';

import { FormsModule } from '@angular/forms';

@NgModule({

  declarations: [AppComponent],

  imports: [BrowserModule, IonicModule.forRoot(), AppRoutingModule, HttpClientModule, FormsModule],

  providers: [{ provide: RouteReuseStrategy, useClass: IonicRouteStrategy }],

  bootstrap: [AppComponent],

})

export class AppModule {}

La page d’accueil se présente comme suit :



|  |  |
| --- | --- |
|  | Donne l’accès au formulaire de déclaration de perte d’un objet. |
|  | Donne l’accès au formulaire de déclaration d’un objet trouvé. |
|  | Donne l’accès à la liste des objets perdus |
|  | Donne l’accès à la liste des objets trouvés |

* La connexion à la base de données MySQL est réalisée à l’aide d’un service Ionic. Pour le créer, entrer la commande suivante dans le répertoire du projet front :

ionic generate service api/user

* Modifier comme suit le fichier /src/app/api/user.service.ts :

import { Injectable } from '@angular/core';

import { HttpClient, HttpHeaders } from '@angular/common/http';

@Injectable({

  providedIn: 'root'

})

export class UserService {

  // On définit une propriété de type HttpHeaders qui représente les options de configuration d’un entête d'une requête http (la requête vers le serveur)

  headers: HttpHeaders;

// HttpClient représente une requête html. Le service est récupéré par injection de dépendance dans le constructeur

  constructor(public *http*: HttpClient) {

    // On instancie le service

*this*.headers = **new** HttpHeaders();

    // On définit les options de configuration et la CORS policy

*this*.headers.append(‘Accept’, 'application/json');

*this*.headers.append('Content-Type', 'application/json');

*this*.headers.append('Access-Control-Allow-Origin', '\*');

*this*.headers.append('Access-Control-Allow-Headers', 'Content-Type');

   }

// On définit la méthode qui envoie les données du formulaire au serveur backend.

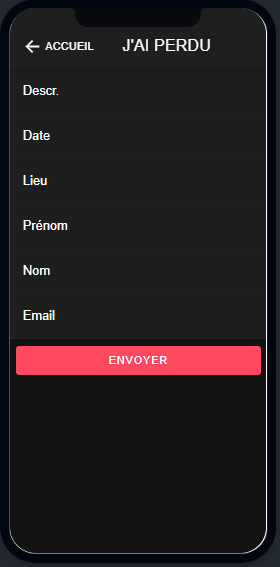
  submitForm(*data*) {

    return *this*.http.post('http://localhost/ionic\_techno/manage-data.php?key=create', *data*);

  }

}

La page de déclaration de perte (lost.page) contient :

* Un formulaire
* Un bouton d’envoi
* Un bouton de retour vers l’accueil

Important : Après envoi du formulaire :

* L’utilisateur est prévenu du succès ou de l’échec de l’enregistrement en base de sa déclaration.
* Le formulaire est vidé

Pour la connexion au service api (et la connexion à la base de données):

* Dans le fichier lost.page.ts, importer le composant api

import { UserService } from '../api/user.service';

* Récupérer le service par injection de dépendances dans le constructeur :

constructor(public *apiService*: UserService) {

* Créer une méthode submitForm() dans la classe **LostPage**
* Récupérer les données du formulaire dans une variable data sous la forme d’un objet JSON
* Passer la variable data à la méthode submitForm de **apiService**.
* Le service renvoi un Observable, donc terminer en ajoutant la promesse subscribe()

export class LostPage implements *OnInit* {

  constructor(public *apiService*: UserService) { }

ngOnInit() { }

…

  submitForm() {

    let data = {

      description: <valeur du formulaire>,

status : 0,

      …

    }

*this*.apiService.submitForm(data).subscribe((*res*) => {

      console.log("SUCCES ===", *res*);

    })

  }

}

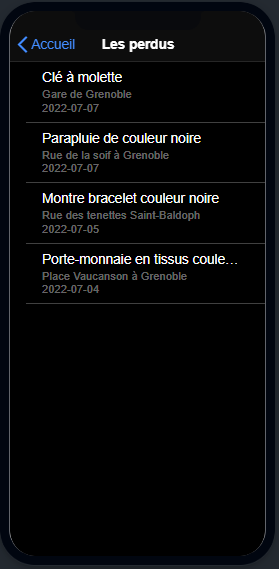
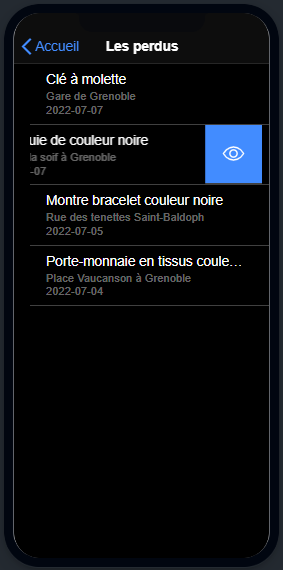
* Lancer les serveurs Ionic, Laragon et tester l’appli en création de déclaration d’objet perdu.
* Procéder de même avec la page de déclaration d’objet trouvé (found.page)

La page liste d’objets perdus (lostlist.page) contient :

- La liste des objets de statut 0 (perdu) en base de données classés par ordre chronologique inverse.

- Un bouton retour vers l’accueil

- Un glissement à gauche fait apparaître un bouton.

Pour afficher la liste des objets perdus, modifier le fichier listlost.page.ts

* Importer le module HttpClient

import { HttpClient } from '@angular/common/http';

* Adapter la classe ListlostPage

Pour récupérer les données de la base de données, utiliser la méthode native du module HttpClient : get(< url-du-serveur-backend>) qui renvoie un Observable à traiter par la méthode subscribe()

export class ListlostPage implements *OnInit* {

// Créer deux propriétés

// URL du serveur backend

  bdUrl = 'http://localhost/ionicserver/retrieve-data.php';

// Un tableau

  entryData = [];

// Récupérer le service par injection de dépendance

  constructor(public *http* : HttpClient) {

*this*.getEntry();

  }

  ngOnInit() {

  }

  getEntry() {

*this*.readAPI(*this*.bdUrl).subscribe((*data*) => {

      for (let i=0; i<*Object*.keys(*data*).length; i++) {

*this*.entryData[i] = {"id": *data*[i].id\_object,

                            "status": *data*[i].status,

                            "description": *data*[i].description,

                            "date": *data*[i].date,

                            "location": *data*[i].location,

                            "firstname": *data*[i].firstname,

                            "lastname": *data*[i].lastname,

                            "email": *data*[i].email};

      } // fin boucle for

    }); // fin subscribe

  }

  readAPI(*URL*: *string*) {

    return *this*.http.get(*URL*);

  }

}

En faisant glisser un objet vers la gauche, un bouton apparait à droite de l’objet. Un clic sur ce bouton fait apparaître la fiche complète de l’objet (tous les champs en base) - viewentry.page.

Depuis cette fiche, il est possible de **mettre à jour** la déclaration (update) ou de la **supprimer** (delete).

Prévoir un champ statut dans le formulaire de la fiche de déclaration : un objet perdu qui est retrouvé peut passer au statut objet trouvé bien que cela ne soit pas systématique.

* Procéder de même avec la liste des objets trouvés (foundlist.page).

BONUS :

Ajouter un champ visuel de l’objet qui sera optionnel.

Le visuel est uploadé dans un dossier upload/. Le nom du visuel est modifié en id\_object.jpg, où id\_object est l’identifiant de la déclaration.

Le visuel sera visible dans les listes et dans les fiches objets.