

# Ionic 3

Semifir SAS

Roland Laurès – Adrien Vossough



# Avant-propos



## Les différents types d'applications

Une application est dite native, quand le développement d'une application est spécifique à une plateforme.

Le développement natif est couteux car demande un apprentissage spécifique pour chacune des plateformes :

iOS : Swift ou Objective-C

Android : Java ou Kotlin

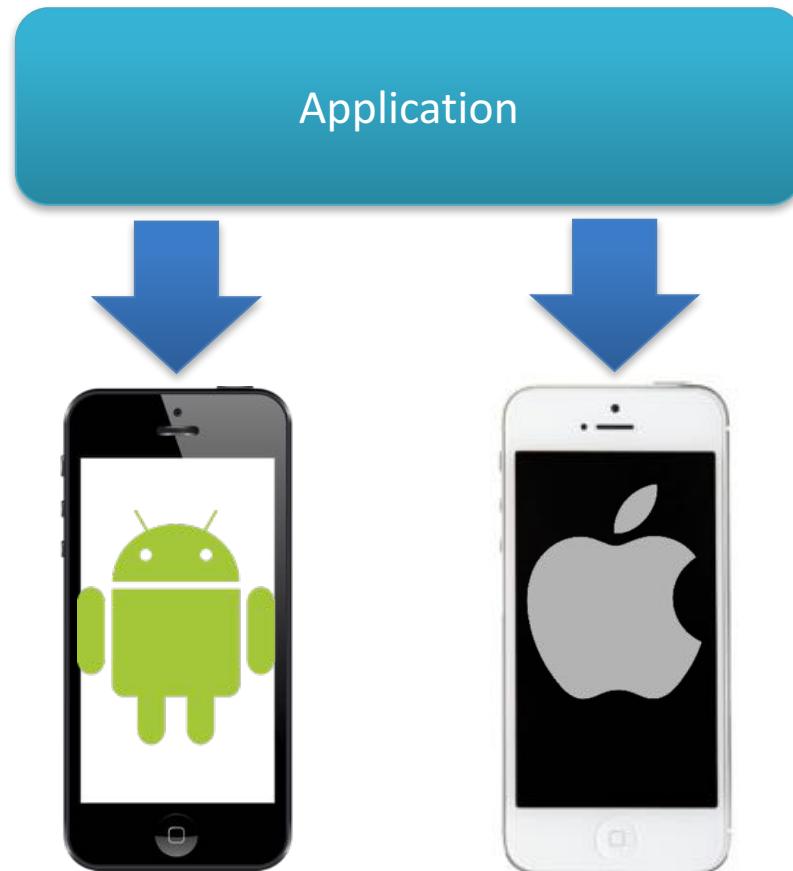
Windows phone : C++, C#, Microsoft Visual Basic et JavaScript

Et chaque plateforme a son propre IDE.

# Avant-propos

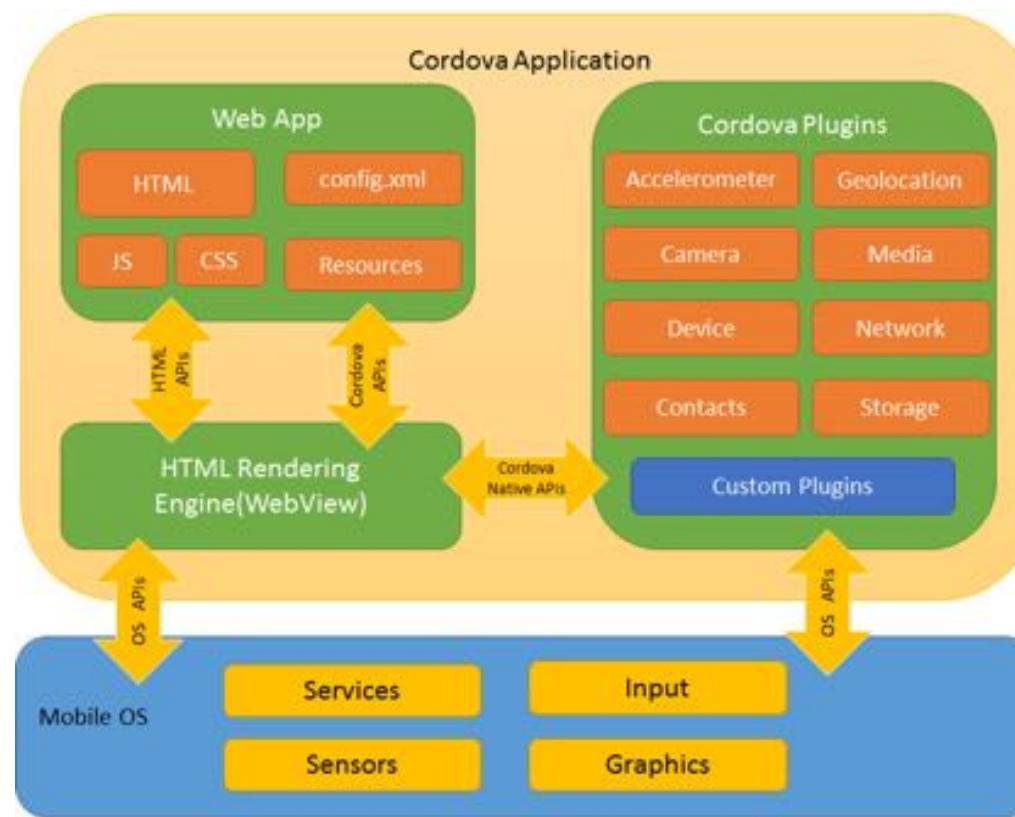
Ionic permet de fabriquer des applications hybrides pour mobile.

Ce qui veut dire que l'application fonctionnera sur Android, iOS et Windows Phone sans avoir à la reprogrammer pour les différentes plateformes.

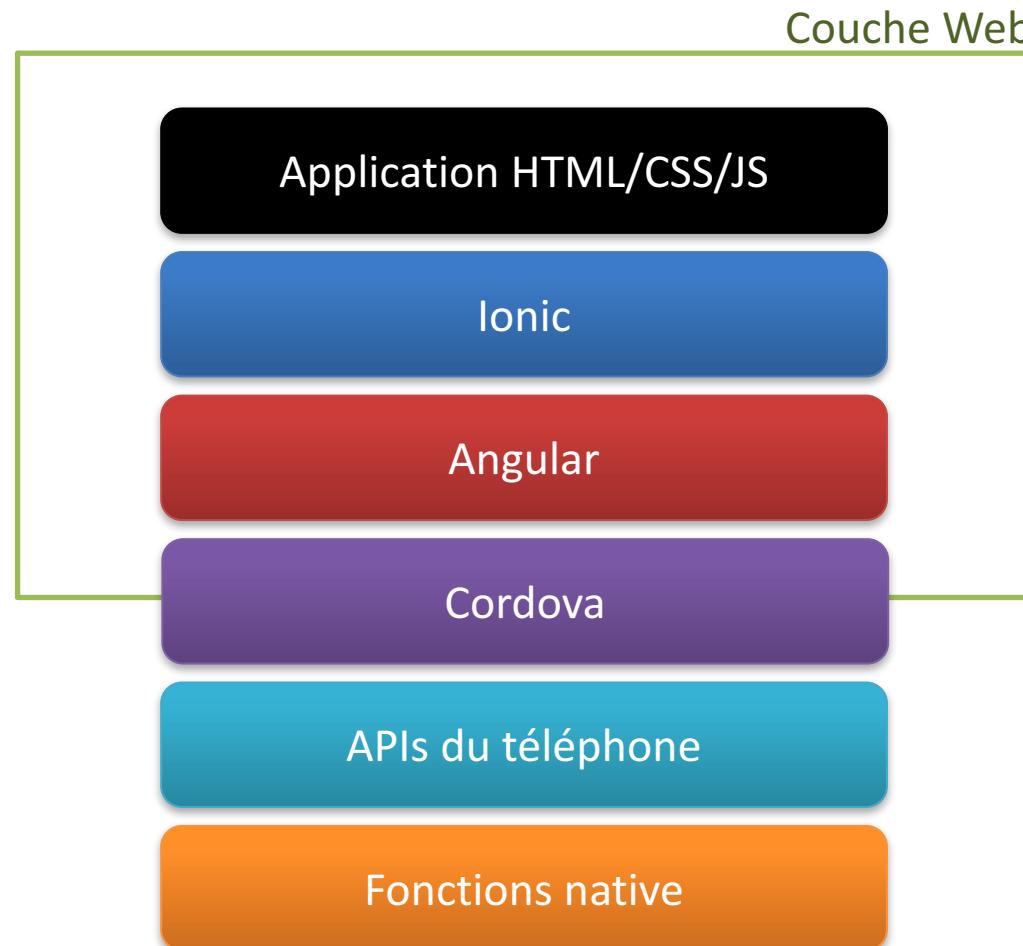


Ionic est basé sur Apache Cordova qui était un projet d'Adobe et qui s'appelait auparavant PhoneGap.

Aujourd'hui Adobe PhoneGap représente un projet apportant un ensemble de services et de fonctionnalités à Cordova.



Ionic ajoute le framework Angular ainsi qu'un ensemble de composants pour simplifier le développement



# Avant-propos

Cordova embarque un navigateur en interne pour charger la couche Web et met en forme pour donner l'impression d'avoir une vraie application native.

Ce n'est pas juste du responsive car Cordova embarque un certain nombre de plugin pour communiquer avec les fonctionnalités natives du téléphone.



Cordova permet d'utiliser quasiment l'ensemble des fonctionnalités des différentes plateformes : Boussole, Vibration, Accéléromètre, Géolocalisation, Notification, etc.

# Avant-propos

Cordova a toutefois un défaut majeur c'est qu'un projet doit commencer par une page blanche, et il faut alors mettre en place toute la stack technique.

Ionic propose un socle technique : Angular avec sa navigation, ses services etc.

Attention : les versions 2 et 3 de Ionic ne sont pas compatibles avec la version 1.

Ceci est dû :

1. au passage vers Angular 2+ qui n'est plus en JavaScript mais en TypeScript
2. et à l'histoire du développement initial de Ionic.

Ionic 1 avait des problèmes d'architecture car il a beaucoup évolué par les retours des développeurs.

L'arrivée à la version 2 fait suite à de multiples évolutions et une réécriture complète du framework.

La migration est donc difficile, on utilise donc Ionic 2+ lors de la réécriture d'un projet existant ou un nouveau projet.

# Installation



Ionic utilise Node.js pour mettre en place sa pile technique pour le développement

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Important security release for 8.x, please update now!

Download for Windows (x64)

**6.11.4 LTS**  
Recommended For Most Users

**8.7.0 Current**  
Latest Features

Other Downloads | Changelog | API Docs    Other Downloads | Changelog | API Docs

Or have a look at the LTS schedule:

Sign up for Node.js Everywhere, the official Node.js Weekly Newsletter.

Avec Ionic, il est conseillé d'utiliser la version LTS (Long Term Support) car l'évolution est plus lente. L'évolution des packages sur la version LF peut provoquer des instabilités.



: Télécharger Git car l'outil de dépendances "npm" aura besoin de se connecter sur le serveur Github

**Une fois ceci fait, aller dans une console et taper :**

**npm install -g cordova**

**npm install -g ionic**

**Cela va installer Cordova et Ionic, c'est une installation globale –g, il ne sera plus la peine de réinstaller ces paquets.**

**La commande ionic vous donnera la liste des commandes possibles.**

# Premier Projet



Dans une console, placez-vous dans un répertoire et taper :

**ionic start**

Cela va permettre d'installer la build technique et un template de projet.

1. Choisir un nom de projet : « tp1 » par exemple.
2. Choisir le template : blank

Cette commande va télécharger les bibliothèques avec des composants et installer les packages

3. A la question qui sera posée, répondre : « n » (Install the free Ionic Pro SDK and connect your app?)



La commande **ionic cordova platform** permet d'ajouter, modifier, supprimer la liste des plateformes mobiles vers lesquelles notre projet va être compiler

Pour ajouter la plateforme Android : **ionic cordova platform add android**

**ionic cordova run android** : permet de lancer l'application en test (connexion vers un smartphone en mode dév)

**ionic upload** : permet d'uploader une application vers une URL de test pour partager, avec des clients ou des collaborateurs par exemple.

## Architecture du projet :

**package.json** : contient la liste des dépendances

**editorconfig** : configuration de l'éditeur de code

**config.xml** : gestion des droits + fonctionnement de l'app sur le mobile :

[http://cordova.apache.org/docs/en/dev/config\\_ref/index.html#The config.xml File](http://cordova.apache.org/docs/en/dev/config_ref/index.html#The%20config.xml%20File)

**ionic.config.json** : fichier de configuration de la CLI ionic

**tsconfig.json** : configuration du compilateur Typescript

**tslint.json** : configuration pour l'analyse de la syntaxe et les règles de développement

**/plugins** : Modules pour la gestion des éléments natifs par Cordova (Géolocalisation, Caméra, etc.)

**/hooks** : ajouter du comportement avant, pendant et après le build

**/node\_modules** : dépendances téléchargées par npm

**ressources** : Différentes ressources (notamment images) pour les différentes plateformes

**www** : utiliser pour stocker le résultat de la compilation

**/src** : sources du projet

## Architecture du projet :

**src/index.html** : on y touche pas, il permet de charger nos dépendances JavaScript

**src/service-worker.js** : gestion des tâches de fond sur le mobile

**src/app** : tous les fichiers de notre application, c'est ici que l'on développe

**src/app/app.module.ts** : Liant entre les modules Angular et Ionic (module principal de l'application)

**src/assets** : ressources de l'application comme les fonts et les icônes

**src/pages** : Contient les différentes « pages » de notre application (ou écrans de l'application)

**src/theme** : Contient des variables css permettant de personnaliser visuellement l'application



Une fois l'installation finie, taper :

**ionic cordova run browser**

**ionic serve**

Cela lance un serveur http. recompile le projet à chaque modification de l'application et recharge la page dans le navigateur (comme pour angular : ng serve).

Dans le navigateur, passer en vue device :

Sous Chrome, taper F12 cliquer sur l'icone device

Sous Safari, aller dans « développement » puis « Passer en mode conception adaptative »

La page principale est contenu dans home.html

The image shows a code editor on the left and a browser window on the right. The code editor displays the file `src\pages\home\home.html`. The browser window shows the rendered output of this file, titled "Ionic Blank". The rendered output contains the text "The world is your oyster." and a link "docs" with the href "http://ionicframework.com/docs/v2". Red arrows point from the title and the rendered text back to the corresponding code in the home.html file.

```
src\pages\home\home.html
<ion-header>
  <ion-navbar>
    <ion-title>
      Ionic Blank
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  The world is your oyster.
  <p>
    If you get lost, the
    <a href="http://ionicframework.com/docs/v2">docs</a> will be your guide.
  </p>
</ion-content>
```

<https://ionicframework.com/docs/ionicons/>

src\pages\home\home.html

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Exemple d'application Ionic
    </ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
  <h1>Les boutons</h1>
  <button ion-button>Suivant</button> bouton simple
  <button ion-button color="danger">Supprimer</button> bouton coloré
  <button ion-button color="danger" block>Supprimer</button> bouton coloré pleine largeur
  <button ion-button color="danger" clear block>Supprimer</button> bouton nu
  <button ion-button color="dark" small block>Suivant</button> bouton réduit

  <button ion-button color="dark" small block icon-left>
    <ion-icon name="add-circle" color="danger"></ion-icon> ajout d'une icone
    Ajouter un utilisateur
  </button>
</ion-content>
```



Les couleurs : "danger", "dark", etc., proviennent du fichier src\theme\variables.scss

Les cards sont des zones de contenu

```
src\pages\home\home.html

<ion-header>
    <!--!... garder le code précédent -->
</ion-header>

<ion-content padding>
    <h1>Mon application Ionic</h1>

    <ion-card>
        <ion-card-header> en-tête
            <h2>En-tête de ma zone</h2>
        </ion-card-header>

        <ion-card-content> contenu
            <p>C'est la lune ! </p>

            <button ion-button color="dark" small block icon-left>
                Image suivante
            </button>
        </ion-card-content>
    </ion-card>
</ion-content>
```

Les listes "ion-list" est un élément structurel pour indiquer une liste de ce que l'on veut

src\pages\home\home.html

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Exemple d'application Ionic
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <h1>Mon application Ionic</h1>

  <ion-list> Une liste de carte
    <ion-list-header>Mes contacts</ion-list-head> Titre de ma liste
    <ion-card>
      <!-- contenu d'une card -->
    </ion-card>

    <ion-card>
      <!-- contenu d'une card -->
    </ion-card>
  </ion-list>

</ion-content>
```

Nous fabriquons une liste d'utilisateur :

src\pages\home\home.ts

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';

@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {
  users: Array<Object>;    Une liste d'objets

  constructor(public navCtrl: NavController) {
    this.users = new Array<Object>();
    remplissage de la liste des utilisateurs
    this.users.push({ name: "Adrien", email: "adrien.vossough@semifir.com" });
    this.users.push({ name: "Roland", email: "roland.laures@semifir.com" });
    this.users.push({ name: "Hervé", email: "herve@gmail.com" });
  }
}
```

## Affichage de la liste des utilisateurs

```
src\pages\home\home.html
```

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Exemple d'application Ionic
    </ion-title>
  </ion-navbar>
</ion-header>
<ion-content padding>
  <h1>Mon application Ionic</h1>

  <ion-list>
    <ion-list-header>Mes contacts</ion-list-header>
      <ion-card *ngFor="let user of users">
        <ion-card-header>
          <ion-icon name="contact">{{user.name}}</ion-icon>
        </ion-card-header>
        <ion-card-content>
          <p><span>email: </span><span>{{user.email}}</span></p>
        </ion-card-content>
        <button ion-button color="dark" small block>Editer</button>
      </ion-card>

    </ion-list>
  </ion-content>
```

affichage de la liste des contacts dans des cards

Le styles est configuré en Sass : <http://sass-lang.com/>

src\pages\home\home.scss

```
page-home {  
    ion-card {  
        ion-icon {  
            font-size: 30px;  
            display: block;  
            text-align: center;  
            color: green;  
        }  
    }  
}
```

La syntaxe est très proche du CSS et simplifie la lecture et l'écriture de ce dernier.

L'équivalent en CSS serait :

```
page-home ion-card ion-icon {  
    font-size: 30px;  
    display: block;  
    text-align: center;  
    color: green;  
}
```

Ce qui revient à :

L'icone qui se trouve dans la card qui elle même est sur la page home



La syntaxe Sass n'est pas compréhensible par les navigateurs, elle doit être transformée en CSS par un préprocesseur (transpileur)

Documentation pour les composants pour les formulaires :

<https://ionicframework.com/docs/components/#inputs>

src\pages\home\home.html

```
<!-- code de l'en tête, avant -->
<ion-content padding>
  <h1>Mon application Ionic</h1>
  <ion-item> ion-item est un bloc de ce que l'on veut. Équivalent du "div" en ionic
    <ion-label floating>Rechercher</ion-label>
    <ion-input type="text"></ion-input>
  </ion-item>
  <ion-list radio-group> Pour que les boutons radio soient liés, les placer dans une ion list avec l'attribut
    <ion-item> radio-group
      <ion-label>Par nom</ion-label>
      <ion-radio checked="true" value="name"></ion-radio>
    </ion-item>
    <ion-item>
      <ion-label>Par email</ion-label>
      <ion-radio value="email"></ion-radio>
    </ion-item>
  </ion-list>
  <ion-list>
    <ion-list-header>Mes contacts</ion-list-header>
<!-- reste du code ... -->
```

ce label est lié à l'input.

floating n'a pas de rapport avec le float css. C'est une façon de s'afficher en ionic (voir doc)

Pour que les boutons radio soient liés, les placer dans une ion list avec l'attribut  
radio-group

```
src\pages\home\home.ts
```

```
import { Component } from '@angular/core';
import { NavController } from 'ionic-angular';

@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {
  users: Array<Object>;
  searchType: 'name' | 'email' = 'name'; searchType ne peut valoir que la chaîne 'name' ou 'email'
  searchValue: string = ''; searchValue : valeur de notre recherche

  constructor(public navCtrl: NavController) {
    this.users = new Array<Object>();
    this.users.push({ name: 'Adrien', email: 'adrien.vossough@semifir.com' });
    this.users.push({ name: "Roland", email: "roland.laures@semifir.com" });
    this.users.push({ name: "Hervé", email: "herve@gmail.com" });
  }

  search() { filter est une méthode de array qui crée un nouveau tableau
    this.users = this.users.filter( (user) => {
      let name = user[this.searchType].toLowerCase();
      return name.startsWith(this.searchValue.toLowerCase());
    })
  }
}
```

filter doit retourner true/false.

Si c'est true le user est conservé sinon il ne sera pas présent dans le nouveau tableau

src\pages\home\home.html

```
<!-- CODE AVANT -->
<ion-item>
  <ion-label floating>Rechercher</ion-label> Two ways data binding de la valeur de recherche
  <ion-input type="text" [(ngModel)]="searchValue" ></ion-input>
</ion-item>
<ion-list radio-group [(ngModel)]="searchType" > Ce binding se fera sur les valeurs des boutons
  <ion-item> radio
    <ion-label>Par nom</ion-label>
    <ion-radio checked="true" value="name"></ion-radio>
  </ion-item>
  <ion-item>
    <ion-label>Par email</ion-label>
    <ion-radio value="email"></ion-radio>
  </ion-item>
</ion-list>
<button ion-button color="primary" block (click)="search()">Rechercher</button>
<!-- CODE APRES -->
```

Two ways data binding de la valeur de recherche

Ce binding se fera sur les valeurs des boutons  
radio  
les valeurs sont : name ou email

appel de la méthode search() lors d'un clic sur  
le bouton rechercher

Nous pouvons remarquer qu'une seule recherche est possible.

src\pages\home\home.ts

```
// CODE AVANT
search() {
    this.users = this.users.filter( user) => {
        let name = user[this.searchType].toLowerCase();
        return name.startsWith(this.searchValue);
    })
}
// CODE APRES
```

La méthode search utilise la méthode filter.

filter crée un nouveau tableau qui est filtré et remplace l'ancien. Nous perdons donc tous les utilisateurs filtrés

Nous pouvons remarquer qu'une seule recherche est possible.

src\pages\home\home.ts

```
export class HomePage {
    users: Array<Object>;
    buffer: Array<Object>;      buffer gardera une référence vers tous les contacts
    searchType: 'name' | 'email' = 'name';
    searchValue: string = '';

    constructor(public navCtrl: NavController) {
        this.users = new Array<Object>();
        this.users.push({ name: 'Adrien', email: 'adrien.vossough@semifir.com' });
        this.users.push({ name: "Roland", email: "roland.laures@semifir.com" });
        this.users.push({ name: "Hervé", email: "herve@gmail.com" });

        this.buffer = this.users;
    }

    search() {          on crée un tableau filtré depuis buffer et on le sauvegarde dans contact
        this.users = this.buffer.filter( (user) => {
            let name = user[this.searchType].toLowerCase();
            return name.startsWith(this.searchValue);
        })
    }
}
```

# Premier projet

## Alerter l'utilisateur

Si aucun utilisateur n'est trouvé, on affiche une alerte

src\pages\home\home.ts

```
import { Component } from '@angular/core';
import { NavController, AlertController } from 'ionic-angular';

// CODE PRECEDENT
export class HomePage {
  // LISTE DES ATTRIBUTS ICI
  constructor(public navCtrl: NavController, public alertDialog: AlertController) {
    // CODE PRECEDENT
  }

  search() {
    this.users = this.buffer.filter((user) => {
      let name = user[this.searchType].toLowerCase();
      return name.startsWith(this.searchValue);
    })
    if (this.users.length == 0) {
      this.alertModal.create({
        title: 'Hum !',
        subTitle: 'Aucun utilisateur trouvé',
        buttons: ['ok']
      }).present();
    }
  }
}
```

Nous ajoutons un nouveau service : AlertController  
Permet d'afficher un modal

Nous utilisons l'injection de dépendance d'Angular  
nous avons donc un attribut alertCtrl

Si la liste des contacts est vide  
nous fabriquons une alerte

Nous demandons l'affichage de l'alerte

# Premier projet : la commande ionic generate

## Création d'un service pour récupérer les utilisateurs

Nous allons créer un service pour simuler la récupération des utilisateurs depuis un serveur.

Dans la console en se plaçant dans le répertoire du projet, taper :

ionic generate

choisir : provider

et le nommer : ChatApi

Cette commande nous propose de générer différents éléments Angular :

- component
- directive
- page : Composant structurel, représentant une page de l'application avec son module
- pipe
- provider : Création d'un service. Simple classe de type singleton qui peut être partagée entre différents éléments
- tabs

Le service est généré automatiquement dans :

src\providers\chat-api\chat-api.ts



Il est ajouté dans le module principal de notre application :

src\app\app.module.ts

dans la section providers.

# Premier projet :

## Création d'un service pour récupérer les utilisateurs

src\providers\chat-api\chat-api.ts

```
import { Injectable } from '@angular/core';
import { Http } from '@angular/http';

@Injectable()
export class ChatApiProvider {
  private users: Array<Object>;

  constructor() {
    this.users = [
      { name: 'Adrien', email: 'adrien.vossough@semifir.com' },
      { name: 'Roland', email: 'roland.laures@semifir.com' },
      { name: 'Hervé', email: 'herve@gmail.com' }
    ];
  }

  getUsers() {
    return this.users;
  }
}
```

Nous fabriquons nos contacts ici en attendant d'avoir un serveur avec une base de données

# Premier projet :

## Utilisation du service contact

src\pages\home\home.ts

```
// garder les imports précédents
import { ChatApiProvider} from '../../../../../providers/chat-api/chat-api';
// garder le décorateur du composant
export class HomePage {
// garder les attributs
    constructor(
        public navCtrl: NavController,
        public alertCtrl: AlertController,
        public api: ChatApiProvider) {
            this.users = api.getUsers();
}
search() {
    this.users = this.api.getUsers().filter((user) => {
        let name = user[this.searchType].toLowerCase();
        return name.startsWith(this.searchValue);
});
// bloc pour l'alerte ici
}
```

        nous importons le service

        nous demandons l'injection du service

        nous récupérons les contacts depuis le service



L'attribut buffer n'est plus utile car c'est le service qui nous fournit les contacts

# Premier projet :

## Création d'une entité pour les contacts

Nous fabriquons un modèle pour les utilisateurs, ce qui permet de définir un comportement et diminuer le risque d'erreur lors de la création d'un contact

src\models\user.model.ts

```
class User {  
    constructor(public name:string, public email:string) {  
    }  
}  
  
export default User;
```

Indique que nous autorisons l'import de la classe User

'l'écriture est identique à :

```
class User {  
    public name: string;  
    public email: string;  
  
    constructor(name: string, email: string) {  
        this.name = name;  
        this.email = email;  
    }  
}  
  
export default User;
```

# Premier projet :

## Utilisation du "model contact"

User est dorénavant une entité bien définie.

src\providers\chat-api\chat-api.ts

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

import User from '../../models/user.model'; Import de la classe User

@Injectable()
export class ChatApiProvider {
    private users : Array<User>

    constructor() {
        this.users = [ Création des objets depuis une classe
            new User('Adrien', 'adrien.vossough@semifir.com' ),
            new User('Roland', 'roland.laures@semifir.com' ),
            new User('Hervé', 'ab@gmail.com' )
        ];
    }

    getUsers(): Array<User> {
        return this.users;
    }
}
```

## Premier projet : Utilisation du "model contact"

```
src\pages\home\home.ts
```

```
import { Component } from '@angular/core';
import { NavController, AlertController } from 'ionic-angular';

import User from '../../models/user.model';
import { ChatApiProvider} from '../../providers/chat-api/chat-api';

@Component({
  selector: 'page-home',
  templateUrl: 'home.html'
})
export class HomePage {
  users: Array<User>;
```

```
// CODE A LA SUITE
```

# Premier projet :

## Création d'une nouvelle page

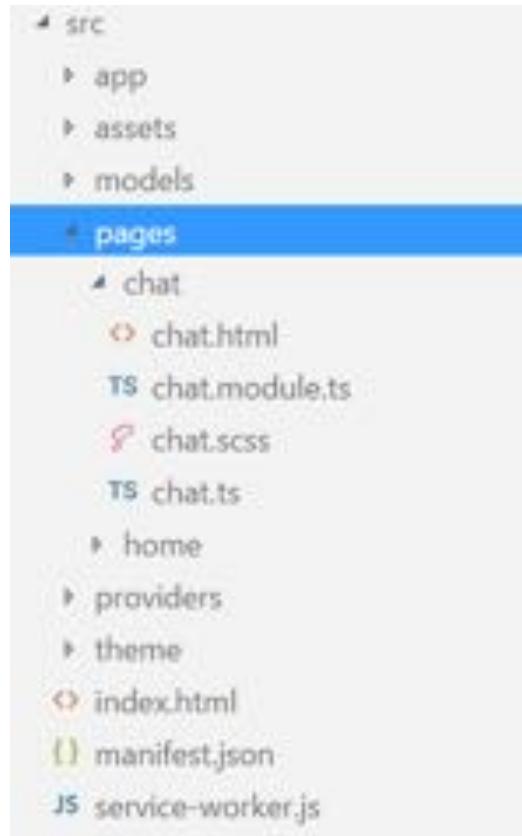
Dans la console en se plaçant dans le répertoire du projet, taper :

ionic generate

**choisir : page**

**et la nommer : Chat**

Nous avons une nouvelle page. C'est un module avec un composant.



# Premier projet :

## Importation de la nouvelle page

La page est ajoutée à son module mais ce dernier n'est pas activé dans notre application. Pour cela :

src\app\app.module.ts

```
/* garder les imports précédents */
// mes pages
import { HomePage } from '../pages/home/home';
import { ChatPage } from '../pages/chat/chat';

@NgModule({
  declarations: [
    MyApp,
    HomePage,
    ChatPage
  ],
  imports: [
    BrowserModule,
    IonicModule.forRoot(MyApp)
  ],
  bootstrap: [IonicApp],
  entryComponents: [
    MyApp,
    HomePage,
    ChatPage
  ],
  providers: /* garder les services précédents */
})
export class AppModule { }
```

# Premier projet :

## Changer de page

src\pages\home\home.ts

```
/* garder les imports précédents */
import { ChatPage } from '../chat/chat';

/* garder le décorateur*/
export class HomePage {
/* garder le reste du code */
    goToChat() {
        this.navCtrl.push(ChatPage);
    }
}
```

le service NavController nous permet de naviguer  
Il suffit de lui donner un composant en utilisant la méthode push()

src\pages\home\home.html

```
<ion-header>
  <ion-navbar>
    <ion-title>
      Exemple d'application Ionic
    </ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <h1>Mon application Ionic</h1>
  <button ion-button color="primary" block (click)="goToChat()">Chatter</button>
  /* Reste du code */

```

création d'un bouton qui appelle goToChat()

## Premier projet :

### Ajout d'un formulaire login/email

La page est ajoutée à son module mais ce dernier n'est pas activé dans notre application. Pour cela :

src\pages\home\home.html

```
<!-- code l'en-tête -->

<ion-content padding>
  <h1>Mon application Ionic</h1>
  <ion-card>
    <form (ngSubmit)="goToChat(loginForm)" #loginForm="ngForm">
      Nous ajoutons la possibilité de fournir un pseudonyme et un mail
      nous envoyons le formulaire à la méthode de changement de page
      <ion-item>
        <ion-label floating>Login</ion-label>
        <ion-input type="text" name="login" ngModel></ion-input>
      </ion-item>      ne pas oublier : ngModel et l'attribut name=" " pour lier le champ au formulaire.

      <ion-item>
        <ion-label floating>Email</ion-label>
        <ion-input type="email" name="email" ngModel></ion-input>
      </ion-item>
      <button ion-button type="submit" block>Chatter</button>
    </form>
  </ion-card>

  <!-- placer dans une ion-card le code du formulaire de filtre -->
```

# Premier projet :

## Envoie d'informations d'une page à l'autre

Nous allons transmettre à la page de Chat le pseudonyme et le mail de l'utilisateur

src\pages\home\home.ts

```
goToChat(form: NgForm) {  
    let user:User = new User(form.value.login, form.value.email);  
    this.navCtrl.push(ChatPage, {user: user});  
}  
}
```

NavController : La méthode push peut recevoir un deuxième paramètre qui sont les données transmises à la nouvelle page

l'import pour NgForm est : `import { NgForm } from '@angular/forms';`

src\pages\chat\chat.ts

```
import { Component } from '@angular/core';  
import { IonicPage, NavController, NavParams } from 'ionic-angular';  
import User from '../../models/user.model';
```

```
@IonicPage()  
@Component({  
    selector: 'page-chat',  
    templateUrl: 'chat.html',  
})  
export class ChatPage {  
    user: User;
```

```
    constructor(public navCtrl: NavController, public navParams: NavParams) {  
        this.user = this.navParams.get('user');  
        console.log(this.user);  
    }  
    /* Laisser le reste du code */
```

On récupère dans le composant Chat, le paramètre fourni par Home, qui est "user

# Premier projet :

## Création d'une entité Message

L'entité message contiendra les informations relatives à un message.

src\models\message.model.ts

```
import User from './user.model'

class Message {
    public date = Date.now();

    constructor(public text: string, public user: User) {
    }
}

export default Message;
```

# Premier projet :

## Récupération des messages

Le service chat-api nous fournit les messages et permet de les envoyer.

src\providers\chat-api\chat-api.ts

```
import { Injectable } from '@angular/core';
import User from '../models/user.model';
import Message from '../models/message.model';

@Injectable()
export class ChatApiProvider {
    private users: Array<User>;

    constructor() {
        this.users = [
            new User('Adrien', 'adrien.vossough@semifir.com' ),
            new User('Roland', 'roland.laures@semifir.com' ),
            new User('Hervé', 'herve@gmail.com' ),
            new User('Dédé', 'ab@gmail.com' ),
        ];
    }

    getUsers(): Array<User> {
        return this.users;
    }
    saveMessage(msg: Message) {
        console.log("message envoyé");
    }
    getMessages(): Array<Message> {
        return [
            new Message('texte de test', new User("adrien", "adrien@test.fr")),
            new Message('autre texte', new User("Roland", "roland@test.fr"))
        ]
    }
}
```

Le service continue de simuler le serveur.  
C'est un bouchon. (stub en anglais)

# Premier projet :

## Récupération des messages

src\pages\chat\chat.ts

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-angular';
import User from '../../models/user.model';
import Message from '../../models/message.model';
import { NgForm } from '@angular/forms';
import { ChatApiProvider } from '../../providers/chat-api/chat-api';
@IonicPage()
@Component({
  selector: 'page-chat',
  templateUrl: 'chat.html',
})
export class ChatPage {
  user: User;
  messages: Array<Message>;
  constructor(public navCtrl: NavController,
    public navParams: NavParams, private api: ChatApiProvider) {
    this.user = this.navParams.get('user');
    this.messages = api.getMessages();
  }
  ionViewDidLoad() {
    console.log('ionViewDidLoad ChatPage');
  }
  sendMessage(msgForm: NgForm) {
    let msg = new Message(msgForm.value.text, this.user);
    this.api.sendMessage(msg);
  }
}
```

Nous récupérons le service ChatApi

Nous demandons les messages à l'API

cette méthode reçoit un formulaire contenant le texte à envoyer

Nous fabriquons un objet message et l'envoyons

# Premier projet :

## Affichage et envoie des messages

```
src\pages\chat\chat.html
```

```
<ion-header>
  <ion-navbar>
    <ion-title>Chat</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-list>
    <ion-item *ngFor="let message of messages">
      {{message.user.name}}: {{message.text}}
    </ion-item>
  </ion-list>

  <ion-card>
    <form (ngSubmit)="sendMessage(msgForm)" #msgForm="ngForm">
      <ion-input type="text" name="text" ngModel></ion-input>
      <button ion-button type="submit" block>Envoyer</button>
    </form>
  </ion-card>
</ion-content>
```

Affiche la liste des messages reçus

Formulaire permettant l'envoie des messages

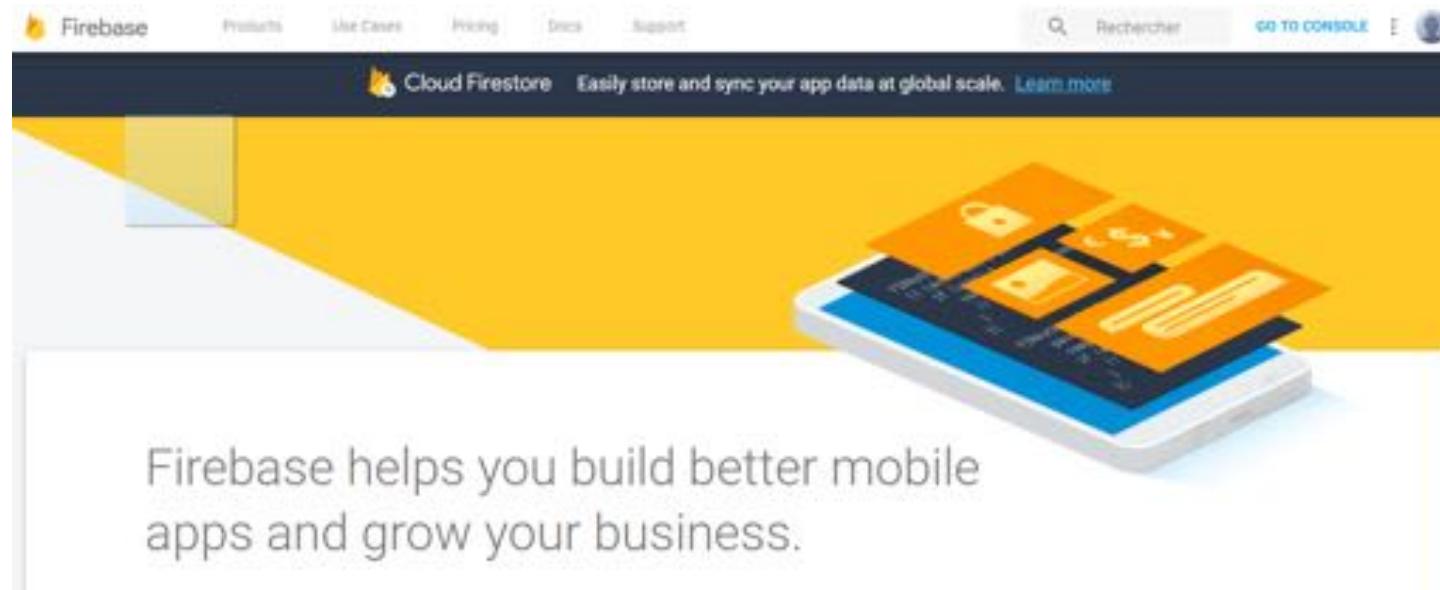
# Firebase

## configuration d'une base de données

Firebase est un ensemble de services en ligne permettant de tester ses applications.

Il propose un serveur NoSQL, des serveurs, etc.

Il appartient à Alphabet (Google) ; il faut donc un compte google pour s'y connecter.



## configuration d'une base de données

1. Cliquer sur "console" en haut à droite.
2. Créer un projet
3. Donner un nom et choisir un pays pour le serveur
4. Dans les règles de la base de données, placer les règles à "true" puis cliquer sur publier

Placer les règles à true permet à n'importe qui de lire ou écrire dans la base.

The screenshot shows the Firebase console interface. On the left, there's a sidebar with various services: Analytics, Create, Authentication, **Database**, Storage, Hosting, Functions, Test Lab, Crash Reporting, and Performance. The 'Database' item is highlighted with a red box. The main area has tabs at the top: DONNÉES, RÈGLES (which is also highlighted with a red box), SAUVEGARDES, and UTILISATION. Below these tabs, there's a sub-header with 'Modifications non publiées' and buttons for 'PUBLIER' and 'SUPPRIMER'. A note says: 'Les règles de sécurité par défaut nécessitent que les utilisateurs soient authentifiés.' Below the note is a code editor containing the following JSON:

```
1+ {
2+   "rules": {
3+     ".read": true,
4+     ".write": true
5+   }
6+ }
```

## Premier projet : Connexion à Firebase

Installer les dépendances pour s'y connecter : `npm install angularfire2 firebase promise-polyfill --save`

**angularfire2 firebase** sont des modules qui permettent de se connecter à l'API de Firebase.

Dans Firebase, cliquer sur Overview et sélectionner : Ajouter Firebase à votre application Web



Les informations fournies vont être utilisé pour configurer nos modules Angular

## Premier projet :

### Connexion à Firebase

Créer le fichier suivant qui va configurer le module firebase en utilisant les informations fournies par Firebase

src\app\firebase.conf.ts

```
export const firebaseConfig = {  
  apiKey: "AIzaSyBfQcwZT3P9sT0G1T2H*****",  
  authDomain: "chat-****.firebaseapp.com",  
  databaseURL: "https://chat-****..firebaseio.com",  
  projectId: "chat-****",  
  storageBucket: "chat-****.appspot.com",  
  messagingSenderId: "932946*****"  
};
```

src\app\app.module.ts

```
// firebase  
import { AngularFireModule } from 'angularfire2';  
import { AngularFireDatabase } from 'angularfire2/database';  
import { AngularFireAuthModule } from 'angularfire2/auth';  
import { firebaseConfig } from './firebase.conf';  
  
@NgModule({  
  declarations: [  
    MyApp,  
    HomePage,  
    ChatPage  
  ],  
  imports: [  
    BrowserModule,  
    IonicModule.forRoot(MyApp),  
    AngularFireModule.initializeApp(firebaseConfig),  
    AngularFireDatabaseModule,  
    AngularFireAuthModule  
  ],  
  /* RESTE DE LA CONFIGURATION */
```

## Premier projet :

### Connexion du service à Firebase

src\providers\chat-api\chat-api.ts

```
import { Injectable } from '@angular/core';
import User from '../../models/user.model';
import Message from '../../models/message.model';

import { AngularFireDatabase } from 'angularfire2/database';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class ChatApiProvider {
  private obsMessages: Observable< Array<Message> >;
  private users: Array<User>;
  messages: Array<Object>;

  constructor(private db: AngularFireDatabase) {
    this.obsMessages = db.list('messages').valueChanges();

    this.users = [new User('Adrien', 'adrien.vossough@gmail.com')];
  }

  getUsers(): Array<User> {
    return this.users;
  }

  sendMessage(msg: Message) {
    this.db.list('messages').push(msg);
  }

  subscribe( fct: (data: Array<Message>) => void ) {
    this.obsMessages.subscribe(fct);
  }
}
```

Nous nous connectons au nœud "message" de notre base de données et demandons de nous retourner un observable.

Cet observable est un objet à laquelle on peut s'abonner. Cela veut dire fournir des fonctions à exécuter lorsqu'il y a du changement

nous envoyons un message au nœud messages de notre BDD

Nous avons retiré getMessages par un subscribe nous fournissons une fonction qui sera appelé à chaque modification de la BDD

## Premier projet :

### Connexion du service à Firebase

src\pages\chat\chat.ts

```
import { Component } from '@angular/core';
import { IonicPage, NavController, NavParams } from 'ionic-angular';
import User from '../../models/user.model';
import Message from '../../models/message.model';
import { NgForm } from '@angular/forms';
import { ChatApiProvider } from '../../providers/chat-api/chat-api';

@IonicPage()
@Component({
  selector: 'page-chat',
  templateUrl: 'chat.html',
})
export class ChatPage {
  user: User;
  messages: Array<Message>;

  constructor(public navCtrl: NavController,
    public navParams: NavParams, private api: ChatApiProvider) {
    this.user = this.navParams.get('user');

    this.api.subscribe( (data:Array<Message>) => {
      this.messages = data;
    });
  }

  sendMessage(msgForm: NgForm) {
    let msg = new Message(msgForm.value.text, this.user);
    this.api.sendMessage(msg);
  }
}
```

nous fournissons une fonction au service chat-api  
cette fonction sera automatiquement appelé à  
chaque changement.

La liste des messages sera mise à jour si la BDD est  
modifiée

## Premier projet :

### Mise en place de la caméra pour prendre des photos

Dans la console taper :

```
ionic cordova plugin add cordova-plugin-camera
```

```
npm install --save @ionic-native/camera
```

Nous ajoutons le module à notre projet

src\app\app.module.ts

```
...
import { Camera } from '@ionic-native/camera';
...
@NgModule({
...
providers: [
...
Camera
...
]
...
})
export class AppModule { }
```

## Premier projet :

### Mise en place de la caméra pour prendre des photos

Pour centraliser la configuration de la caméra, nous allons créer un proxy.

Proxy est un design pattern où un objet se fait passer pour un autre tout en apportant un nouveau comportement

Dans la console, taper : **ionic generate** et sélectionner provider. Le nommer Camera

src\providers\camera\camera.ts

```
import { Injectable } from '@angular/core';
import { Camera, CameraOptions } from '@ionic-native/camera';

@Injectable()
export class CameraProxy extends Camera {

  constructor() {
    super()
  }

  getPicture(options?: CameraOptions): Promise<any> {
    const defaultOptions: CameraOptions = {
      quality: 75,
      destinationType: this.EncodingType.JPEG,
      encodingType: this.EncodingType.JPEG,
      mediaType: this.MediaType.PICTURE
    }
    return super.getPicture(defaultOptions);
  }
}
```

Un proxy a la même interface que son parent et doit appeler les méthodes de son parent.

Nous ajoutons une configuration par défaut.

## Premier projet :

### Mise en place de la caméra pour prendre des photos

```
src\pages\chat\chat.ts
```

```
// reste des imports

import { CameraProxy } from '../../providers/camera/camera';

@IonicPage()
@Component({
  selector: 'page-chat',
  templateUrl: 'chat.html',
})
export class ChatPage {
  user: User;
  messages: Array<Message>;

  constructor(public navCtrl: NavController, public navParams: NavParams,
    private api: ChatApiProvider, private camera: CameraProxy) {
// garder le contenu du constructeur
  }

// reste des méthodes

  takePicture() {
    this.camera.getPicture()
      .then(() => {
        console.log("OK");
      }).catch((err) => {
        console.log(err);
      });
  }
}
```

Nous importons notre service CameraProxy

getPicture() est une méthode de camera qui permet de prendre des photos. Elle peut recevoir des options mais nous avons déjà fait cela dans le CameraProxy

then( méthode si tout se bien).catch(méthode s'il y a une erreur

## Premier projet :

### Mise en place de la caméra pour prendre des photos

src\models\message.model.ts

```
import User from './user.model'

class Message {
    public date = Date.now();
    public image = '';Nous ajoutons la possibilité d'envoyer une image à notre message

    constructor(public text: string, public user: User) {
    }

}

export default Message;
```

## Premier projet :

### Mise en place de la caméra pour prendre des photos

src\models\message.model.ts

```
// imports
// décorateurs
export class ChatPage {
  user: User;
  messages: Array<Message>;
  msg: Message;

  constructor(public navCtrl: NavController, public navParams: NavParams,
    private api: ChatApiProvider, private camera: CameraProxy) {

    this.user = this.navParams.get('user');
    this.msg = new Message('', this.user);
    Nous fabriquons le message immédiatement pour
    pouvoir ajouter une image

    this.api.subscribe((data: Array<Message>) => {
      this.messages = data;
    })
  }
  sendMessage(msgForm: NgForm) {
    this.msg.text = msgForm.value.text;
    this.api.sendMessage(this.msg);
    this.msg = new Message('', this.user);
    Dès que le message est envoyé, nous en fabriquons un
    autre qui attend d'être rempli à son tour
  }

  takePicture() {
    this.camera.getPicture()
      .then((image) => {
        this.msg.image = 'data:image/jpeg;base64,' + image;
      }).catch((err) => {
        console.log(err);
      });
  }
}
```

Nous ajoutons la possibilité d'envoyer une image à notre message

Nous fabriquons le message immédiatement pour pouvoir ajouter une image

Dès que le message est envoyé, nous en fabriquons un autre qui attend d'être rempli à son tour

Nous ajoutons l'image au message sous forme de chaîne de caractère en base64

## Premier projet :

### Mise en place de la caméra pour prendre des photos

src\pages\chat\chat.html

```
<ion-header>
  <ion-navbar>
    <ion-title>Chat</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-list>
    <ion-item *ngFor="let message of messages">
      {{message.user.name}}: {{message.text}}
    </ion-item>
  </ion-list>

  <ion-card>
    <form (ngSubmit)="sendMessage(msgForm)" #msgForm="ngForm">
      <ion-input type="text" name="text" ngModel></ion-input>
      <button ion-button type="submit" block>Envoyer</button>
    </form>
    <button ion-button (click)="takePicture()" block>Ajouter une photo</button>
    <img *ngIf="msg.image!==''" [src]="msg.image" />
  </ion-card>
</ion-content>
```

Ajouter un bouton qui appelle la fonction de prise d'image

Nous affichons l'image si ce n'est pas une chaîne vide

## Premier projet :

### Mise en place de la caméra pour prendre des photos

src\pages\chat\chat.html

```
<ion-header>
  <ion-navbar>
    <ion-title>Chat</ion-title>
  </ion-navbar>
</ion-header>

<ion-content padding>
  <ion-list>
    <ion-item *ngFor="let message of messages">
      {{message.user.name}}: {{message.text}}
      <img *ngIf="msg.image!=''" [src]="message.image" />
    </ion-item>
  </ion-list>

  <ion-card>
    <form (ngSubmit)="sendMessage(msgForm)" #msgForm="ngForm">
      <ion-input type="text" name="text" ngModel></ion-input>
      <button ion-button type="submit" block>Envoyer</button>
    </form>
    <button ion-button (click)="takePicture()" block>Ajouter une photo</button>
    <img *ngIf="msg.image!=''" [src]="msg.image" />
  </ion-card>
</ion-content>
```

Ajouter un bouton qui appelle la fonction de prise d'image

Nous affichons l'image si ce n'est pas une chaîne vide

## Premier projet :

### Tester l'application sous android

Avec un téléphone Android :

**Le passer en mode développeur**

télécharger Android SDK :

<https://developer.android.com/studio/index.html#command-tools>

Ajouter à notre application la plateforme Android :

**ionic cordova platform add android**

Brancher le mobile avec un cable USB.

Taper :

**cordova platform update android**

**ionic cordova run android**

# Projet

## TP 2

