

Documentation

Hackat'Event

Quentin Couzinet



Sommaire

01	Contexte	p 3
02	Environnement de travail	p 4
03	Architecture de l'application	p 5
04	Base de données	p 6
05	API	p 8
06	Liste des hackathons et des ateliers liés	p 11
07	Détails d'un atelier	p 16
08	Inscription aux ateliers	p 18
09	Commentaires	p 21
10	Gestion des favoris	p 24
11	Conclusion	p 28

Contexte

L'entreprise Hackat'Agence a pour but de simplifier la gestion de l'organisation d'évènements rassemblant des personnes autour d'un projet informatique, appelés hackathons.

Pour cela plusieurs applications vont être développées dont notamment une application mobile Hackat'Event.

Les fonctionnalités attendues de cette application sont :

- Lister les hackathons
- Lister les atelier liés à un hackathon
- Pouvoir s'inscrire à un atelier
- Pouvoir poster un commentaire sur un atelier
- Lister les commentaires d'un atelier
- Ajouter/retirer un atelier des favoris et lister ceux-ci

L'objet de ce document est de traiter de la réalisation, et du fonctionnement de cette application.

Environnement de travail

Cette application a été développée sur une machine virtuelle Linux Debian 11, créée sur le réseau du BTS grâce à la plateforme VMware vSphere mise à disposition.

L'adresse IP de cette machine est 192.168.51.98

L'application, développée sur le framework Ionic (v 5.4.16), utilise principalement les langages JavaScript, HTML, et CSS.

L'environnement de développement utilisé était Visual Studio Code, et une connexion SSH permettait de coder directement sur la VM.

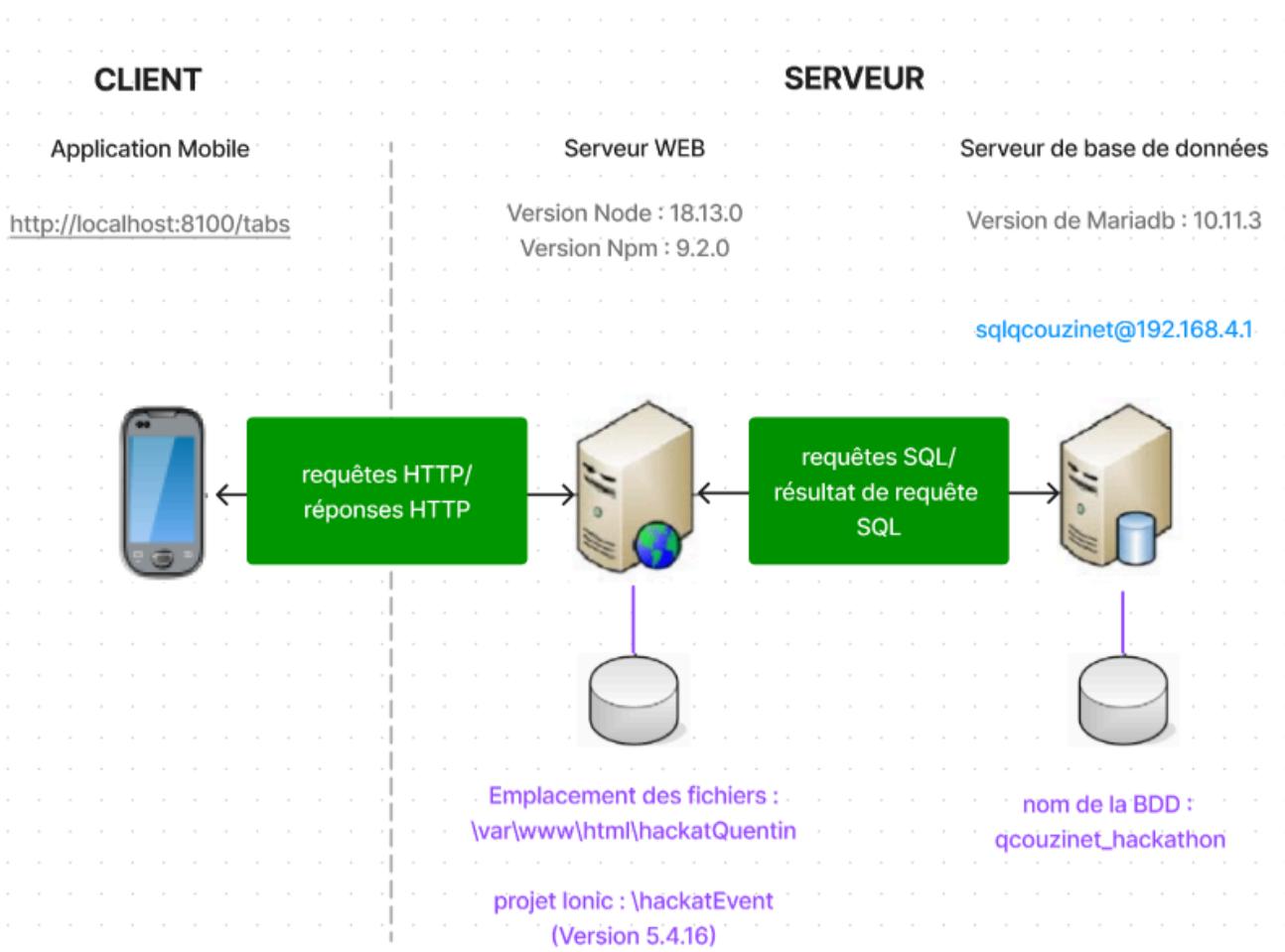
La gestion des versions a été faite grâce à Git et à la création d'un dépôt distant sur Github, accessible à cet adresse :
<https://github.com/soniiix/hackathon/tree/master/hackatEvent> permettant de retrouver l'historique des modifications globales de la solution.

La gestion du projet Hackathon a été faite sur Trello, la roadmap est consultable ici :
<https://trello.com/invite/b/3dQXZIYU/ATTI84f2371045730bfab5fafb3d5a72714F405865E/hackatevent>

Architecture de l'application

A. Schéma d'architecture technique

Voici le schéma de l'architecture technique de l'application Hackat'Event :



Il présente de manière détaillée la disposition, l'interaction, et les versions des différents composants logiciels.

Base de données

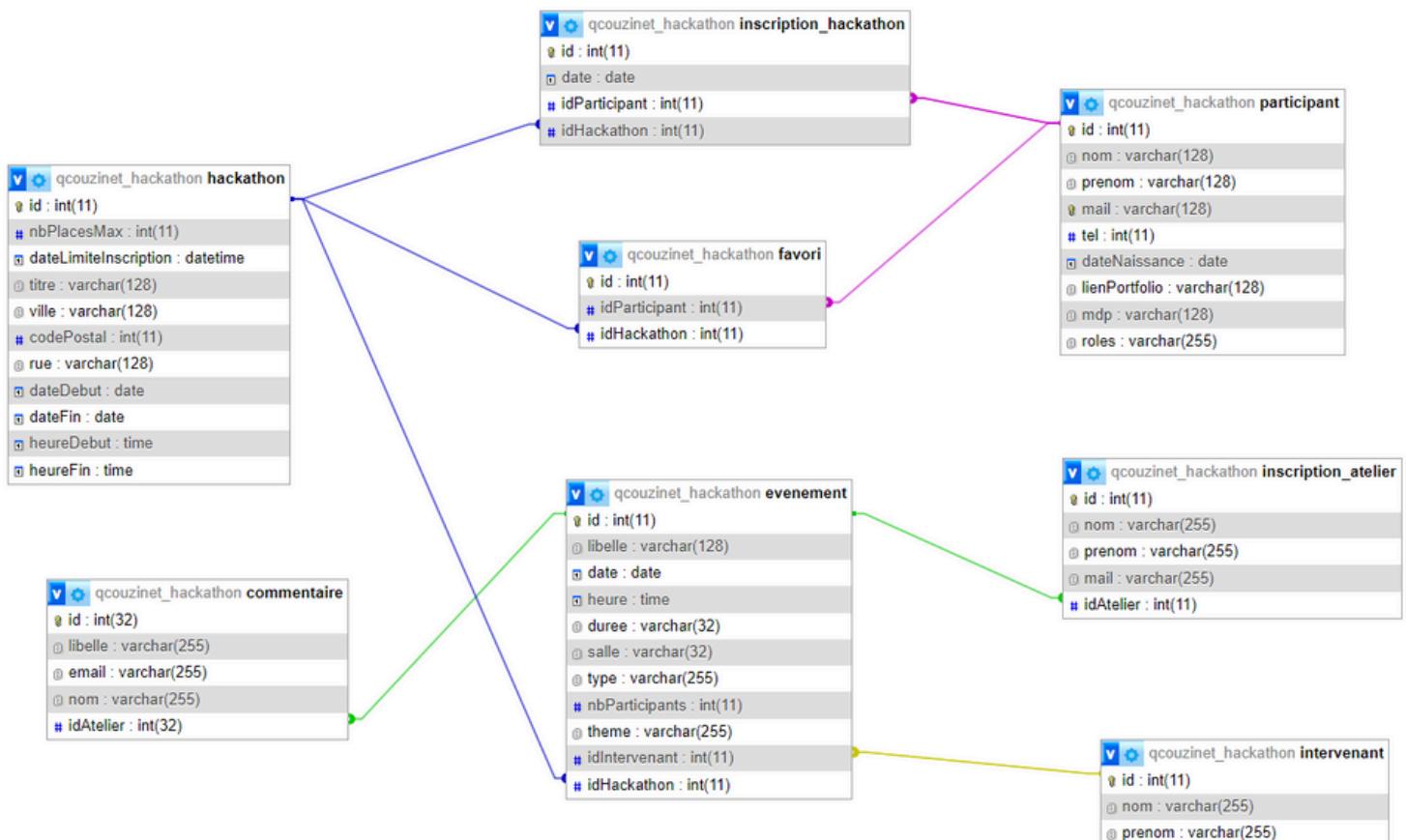
La base de données est essentielle au fonctionnement de l'application Hackat'Event.

Elle sert pour stocker et gérer toutes les informations relatives aux hackathons et aux ateliers.

Retrouvez ci-après le schéma de la base de données, les différentes tables qui la composent, ainsi que les relations entre ces tables.

A. Schéma conceptuel

Voici un schéma de la base de données :



Les tables utilisés pour l'application mobile sont :
hackathon, evenement, inscription_atelier, et commentaire

Un événement est soit un atelier soit une conférence. C'est une situation d'héritage où evenement est la mère et atelier et conference sont deux filles.

Comme nous avons utilisé la méthode d'héritage Single Table, nous avons donc une seule table avec un champ discriminant 'type' permettant de savoir si l'évènement est une conférence ou un atelier. C'est ce dernier qui est utilisé pour l'application mobile.

Si l'évènement est une conférence, alors il aura comme champs propres : le thème et l'identifiant de l'intervenant.

Si l'évènement est un atelier, alors il aura comme champ propre, uniquement le nombre de participants.

Exemples :

id	libelle	date	heure	duree	salle	type	nbParticipants	theme	idIntervenant	idHackathon
1	Atelier sur le leadership	2023-12-08	16:44:00	2	B009	atelier	25	NULL	NULL	1
2	Blockchain : Applications et implications futures	2023-12-08	14:47:00	4	G934	conference	NULL	Cryptomonnaies	2	2
3	Sécurité informatique : Pratiques avancées	2023-12-16	15:57:00	2	C890	atelier	33	NULL	NULL	1
4	Conférence sur l'IA	2023-12-08	14:08:00	2	M260	conference	NULL	Intelligence artificielle	3	3

Enfin, dans la table hackathon, l'adresse a été séparée en plusieurs champs (ville, code postal, rue) pour des soucis de clarté et de lisibilité et pour simplifier les modifications éventuelles.

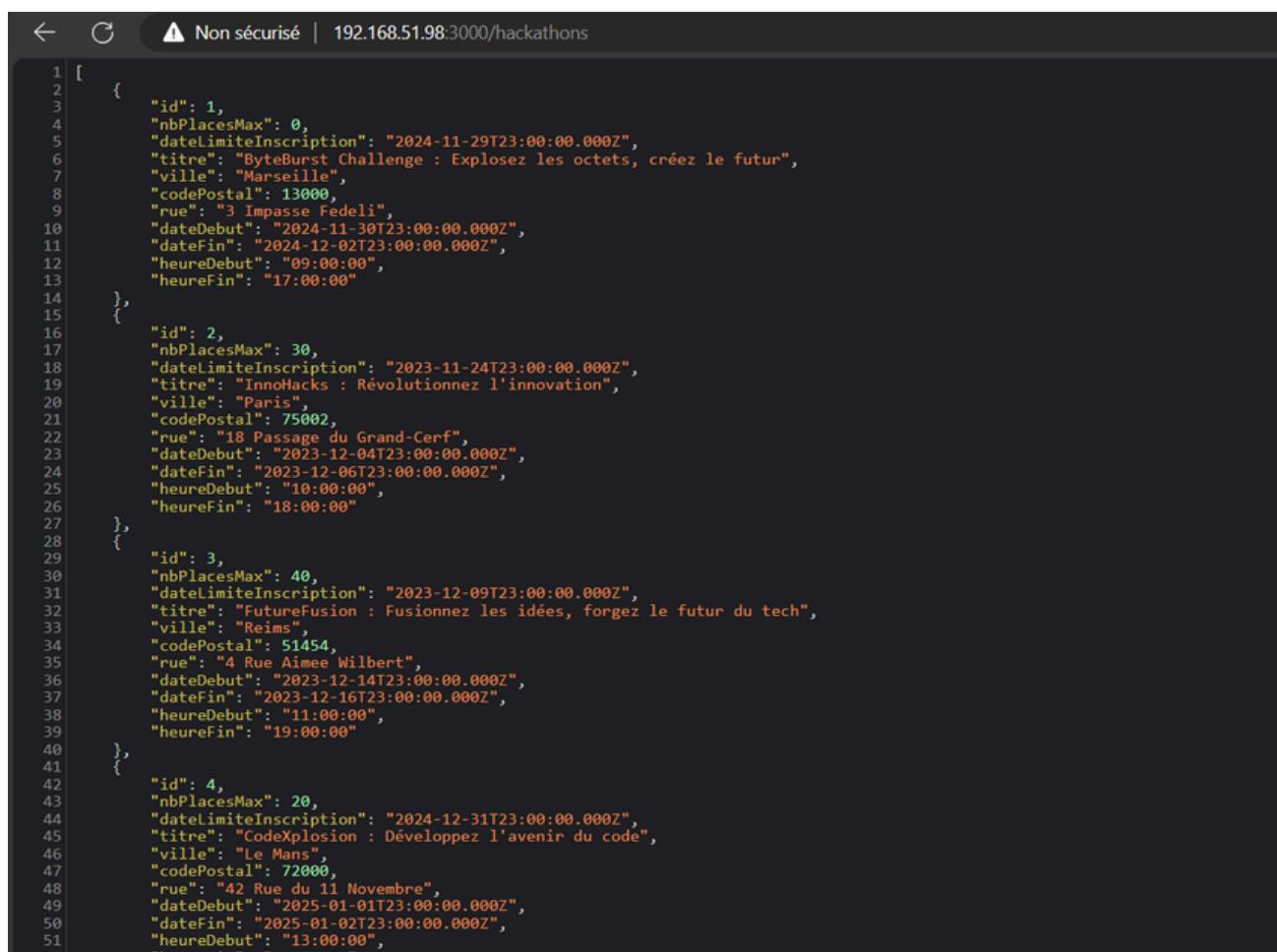
API

Pour accéder aux données de la base de données, nécessaires pour le fonctionnement de l'application, j'ai créé une API en Node JS.

Celle-ci se trouve dans un dossier au-dessus du projet et se démarre avec la commande « nodemon ».

Les données sont ensuite accessibles en lecture à l'adresse : <http://192.168.51.98:3000/> suivi de la route souhaitée.

Par exemple, pour la liste des hackathons il faut saisir <http://192.168.51.98:3000/hackathons> et on obtient dans le navigateur :



The screenshot shows a browser window with the URL <http://192.168.51.98:3000/hackathons>. The page content is a JSON array of four objects, each representing a hackathon. The objects are numbered 1 through 4. Each object contains fields such as id, nbPlacesMax, dateLimiteInscription, titre, ville, codePostal, rue, dateDebut, dateFin, heureDebut, and heureFin. The data is presented in a monospaced font, with line numbers on the left side of the JSON structure.

```
1 [  
2 {  
3     "id": 1,  
4     "nbPlacesMax": 0,  
5     "dateLimiteInscription": "2024-11-29T23:00:00.000Z",  
6     "titre": "ByteBurst Challenge : Explitez les octets, créez le futur",  
7     "ville": "Marseille",  
8     "codePostal": 13000,  
9     "rue": "3 Impasse Fedeli",  
10    "dateDebut": "2024-11-30T23:00:00.000Z",  
11    "dateFin": "2024-12-02T23:00:00.000Z",  
12    "heureDebut": "09:00:00",  
13    "heureFin": "17:00:00"  
14 },  
15 {  
16     "id": 2,  
17     "nbPlacesMax": 30,  
18     "dateLimiteInscription": "2023-11-24T23:00:00.000Z",  
19     "titre": "InnoHacks : Révolutionnez l'innovation",  
20     "ville": "Paris",  
21     "codePostal": 75002,  
22     "rue": "18 Passage du Grand-Cerf",  
23     "dateDebut": "2023-12-04T23:00:00.000Z",  
24     "dateFin": "2023-12-06T23:00:00.000Z",  
25     "heureDebut": "10:00:00",  
26     "heureFin": "18:00:00"  
27 },  
28 {  
29     "id": 3,  
30     "nbPlacesMax": 40,  
31     "dateLimiteInscription": "2023-12-09T23:00:00.000Z",  
32     "titre": "FutureFusion : Fusionnez les idées, forgez le futur du tech",  
33     "ville": "Reims",  
34     "codePostal": 51454,  
35     "rue": "4 Rue Aimée Wilbert",  
36     "dateDebut": "2023-12-14T23:00:00.000Z",  
37     "dateFin": "2023-12-16T23:00:00.000Z",  
38     "heureDebut": "11:00:00",  
39     "heureFin": "19:00:00"  
40 },  
41 {  
42     "id": 4,  
43     "nbPlacesMax": 20,  
44     "dateLimiteInscription": "2024-12-31T23:00:00.000Z",  
45     "titre": "CodeExpllosion : Développez l'avenir du code",  
46     "ville": "Le Mans",  
47     "codePostal": 72000,  
48     "rue": "42 Rue du 11 Novembre",  
49     "dateDebut": "2025-01-01T23:00:00.000Z",  
50     "dateFin": "2025-01-02T23:00:00.000Z",  
51     "heureDebut": "13:00:00",  
52     "heureFin": "17:00:00"  
53 }
```

Voici un tableau récapitulatif de l'ensemble des routes de l'API :

URL	Méthode HTTP	Données transmises	Données récupérées	Statut
/hackathons	GET	-	tous les hackathons	200
/ateliers/:id	GET	l'identifiant d'un hackathon	tous les ateliers liés à un hackathon	200
/inscription-atelier	POST	nom, prénom, email, et identifiant d'un atelier	-	200
/commentaires/atelier/:idAtelier	GET	l'identifiant d'un atelier	tous les commentaires relatifs à un atelier	200
/commentaire/post	POST	libellé, email, nom, et identifiant d'un l'atelier	-	200

Extrait du code :

```

39 //Route pour accéder à l'ensemble des ateliers liés à un hackathon
40 app.get('/ateliers/:id', (req, res) => {
41   var idHackathon = req.params.id;
42   var sql = "SELECT * FROM evenement WHERE type='atelier' AND idHackathon = ?";
43   connection.query(sql, [idHackathon], function(err,resultat){
44     console.log(resultat)
45     res.send(resultat);
46   })
47 })
48
49 //Route pour enregistrer une inscription à un atelier
50 app.post('/inscription-atelier', (req, res) => {
51   const { nom, prenom, email, idAtelier } = req.body;
52   var sql = "INSERT INTO inscription_atelier (nom, prenom, mail, idAtelier) VALUES (?, ?, ?, ?)";
53   connection.query(sql, [nom, prenom, email, idAtelier], function(err, resultat){
54     if (err) {
55       console.error("Erreur lors de l'insertion de l'inscription :", err);
56       res.status(500).send("Erreur lors de l'insertion de l'inscription");
57     } else {
58       console.log("Inscription ajoutée avec succès !");
59       res.status(200).send("Inscription ajoutée avec succès !");
60     }
61   })
62 })

```

A. Service Ionic

Pour accéder à cette API dans l'application, j'ai créé un service Ionic. Cela permet de simplifier le code en définissant des fonctions globales pouvant être utilisées n'importe où dans l'application. Chaque fonction correspond à une route de l'API.

Un client HTTP permet de faire le lien avec celle-ci.

Voici un extrait du code :

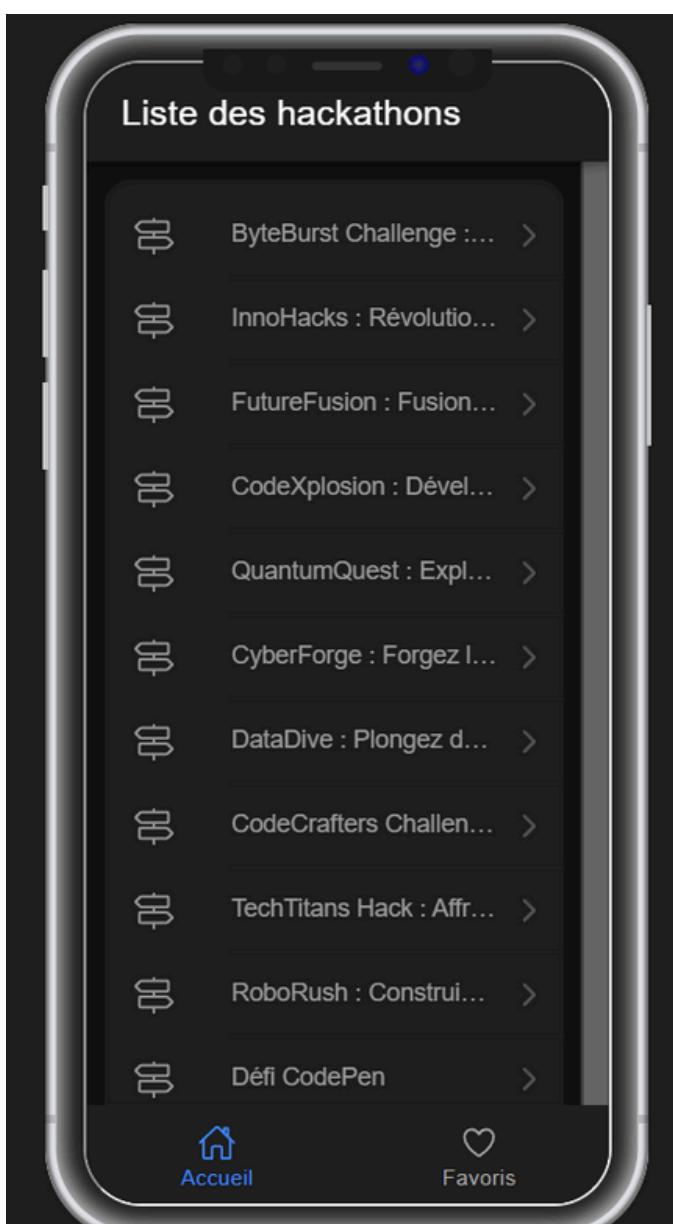
```
ts hackat-service.service.ts ✘

hackatQuentin > hackatEvent > src > app > ts hackat-service.service.ts > ...
1 import { Injectable } from '@angular/core';
2 import { HttpClient } from "@angular/common/http";
3
4 @Injectable({
5   providedIn: 'root'
6 })
7 export class HackatServiceService {
8
9
10  private hackathons:any;
11  private ateliers:any;
12  private inscriptionAtelier:any;
13
14  constructor(private http : HttpClient) {
15  }
16
17  getHackathons(){
18    return new Promise((resolve) => {
19      var url = "http://192.168.51.98:3000/hackathons";
20      this.http.get(url).subscribe((data) => {
21        resolve(data);
22      });
23    })
24  }
25
26  getAteliersByIdHackathon(idHackathon:any){
27    return new Promise((resolve) => {
28      var url = "http://192.168.51.98:3000/ateliers/" + idHackathon;
29      this.http.get(url).subscribe((data) => {
30        resolve(data);
31      });
32    })
33  }
34
35  postInscriptionAtelier(nom:string, prenom:string, email:string, idAtelier:number){
36    return new Promise((resolve) => {
37      var url = "http://192.168.51.98:3000/inscription-atelier";
38      const body = { nom, prenom, email, idAtelier };
39    })
40  }
41}
```

Liste des hackathons

La première page de l'application est celle qui liste les hackathons.

On retrouve, pour chaque hackathon : une icône d'illustration, le titre du hackathon, et une flèche en fin de ligne indiquant que c'est un élément sur lequel on peut cliquer. En effet, en cliquant dessus cela renvoie sur la page des détails du hackathon, qui sera expliquée après.



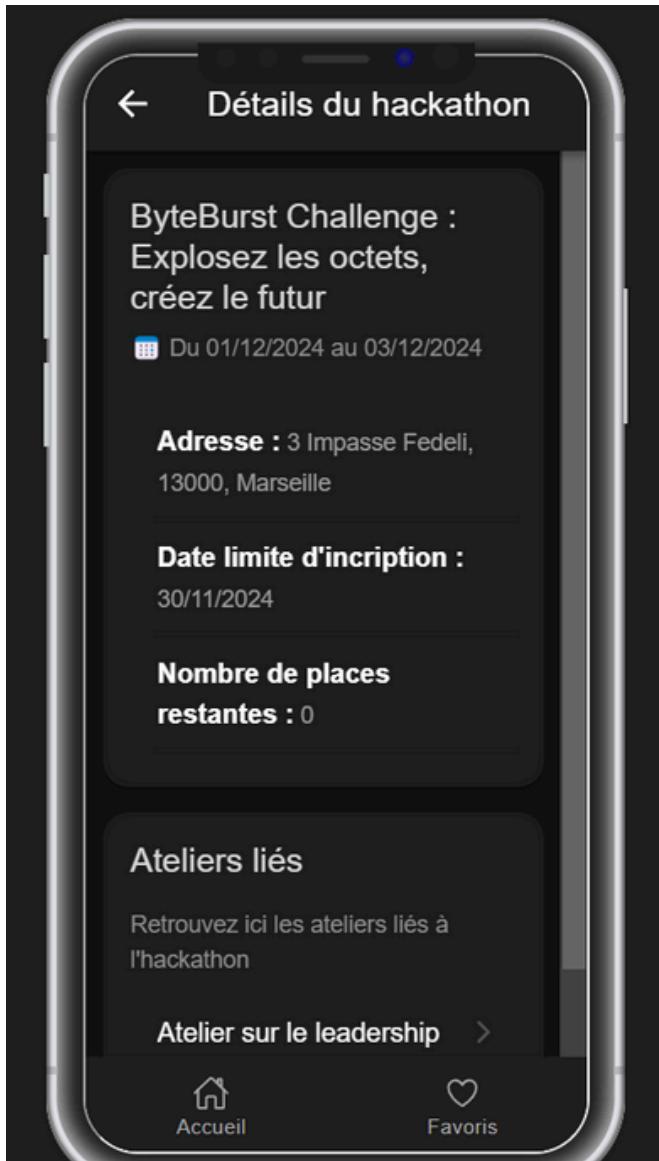
Pour obtenir cette liste, on utilise le service **HackatService** et dans le fichier TypeScript (contrôleur) de la page on les récupère grâce à la méthode `getHackathons`, créée précédemment.

```
export class HackathonsPage {  
  
  hackathons:any;  
  unHackathon:any;  
  
  constructor(public http : HttpClient, public hackatService : HackatServiceService, pr  
{  
  //récupération des hackathons de la BDD  
  this.hackatService.getHackathons().then(results => {  
    this.hackathons = results;  
  });  
}  
  
//fonction qui redirige vers les détails d'un hackathon après avoir cliqué dessus  
detailHackathonClick(unHackathon:any){  
  let navigationExtras: NavigationExtras = {  
    state : {  
      item: unHackathon  
    }  
  };  
  this.router.navigate(['/tabs/details'], navigationExtras)  
}
```

Le fichier HTML (vue) affiche ensuite ces données sous forme de liste grâce à “ion-list” et l'utilisation de “*ngFor” pour les parcourir :

```
<!-- liste des hackathons -->  
<ion-card>  
  <ion-list>  
    <ion-item button *ngFor="let unHackathon of hackathons" (click)="detailHackathonClick(unHackathon)" detail="true">  
      <ion-icon name="trail-sign-outline" slot="start"></ion-icon>  
      <ion-label>  
        <p>{{unHackathon.titre}}</p>  
      </ion-label>  
    </ion-item>  
  </ion-list>  
</ion-card>
```

En cliquant sur un hackathon, on arrive sur cette page :



On retrouve les détails du hackathon avec :

- le titre
- la date de début et de fin
- l'adresse
- la date limite d'inscription
- le nombre de places restantes

Ces informations sont passées de la page précédente à celle-ci grâce à l'utilisation de `NavigationExtras`
(<https://angular.io/api/router/NavigationExtras>)

Envoi :

```
//fonction qui redirige vers les détails d'un hackathon après avoir cliqué dessus
detailHackathonClick(unHackathon:any){
  let navigationExtras: NavigationExtras = {
    state : {
      item: unHackathon
    }
  };
  this.router.navigate(['/tabs/details'], navigationExtras)
}
```

Récupération :

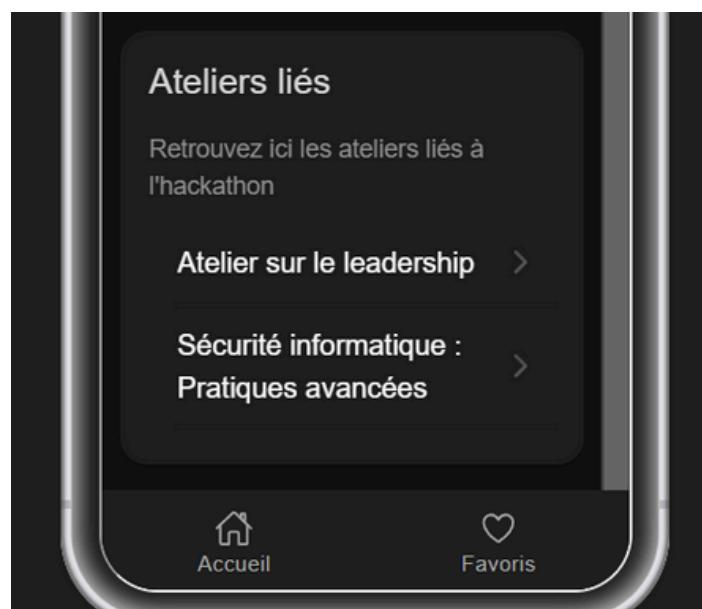
```
export class DetailsPage {
  leHackathon:any;
  ateliers:any;
  unAtelier:any

  constructor(private router: Router, public hackatService : HackatServiceService, private ac
  //récupération du hackathon sur lequel on a cliqué sur la page précédente
  this.activeRoute.queryParams.subscribe(param=>{
    let navigation:any = this.router.getCurrentNavigation()?.extras.state;
    this.leHackathon = navigation.item;
```

Le principe est le même pour les autres pages

A. Liste des ateliers

Sous les informations du hackathon sont affichés, à la manière de la liste des hackathons sur la page précédente, la liste des ateliers liés :



Ceux-ci sont récupérés grâce à la méthode `getAteliersByIdHackathon` du service Ionic, qui utilise l'API (voir 2ème route du tableau).

```
//récupération en BDD des ateliers correspondants
this.hackatService.getAteliersByIdHackathon(this.leHackathon.id).then(results => {
  |  this.ateliers=results;
  });
})
```

Détails d'un atelier

Lorsqu'on clique sur un atelier, on arrive sur la page affichant les détails de celui-ci. Toutes les informations sont affichées, à savoir le titre, la date, et la salle où a lieu l'atelier.



Comme pour les pages précédentes, il s'agit d'une liste:

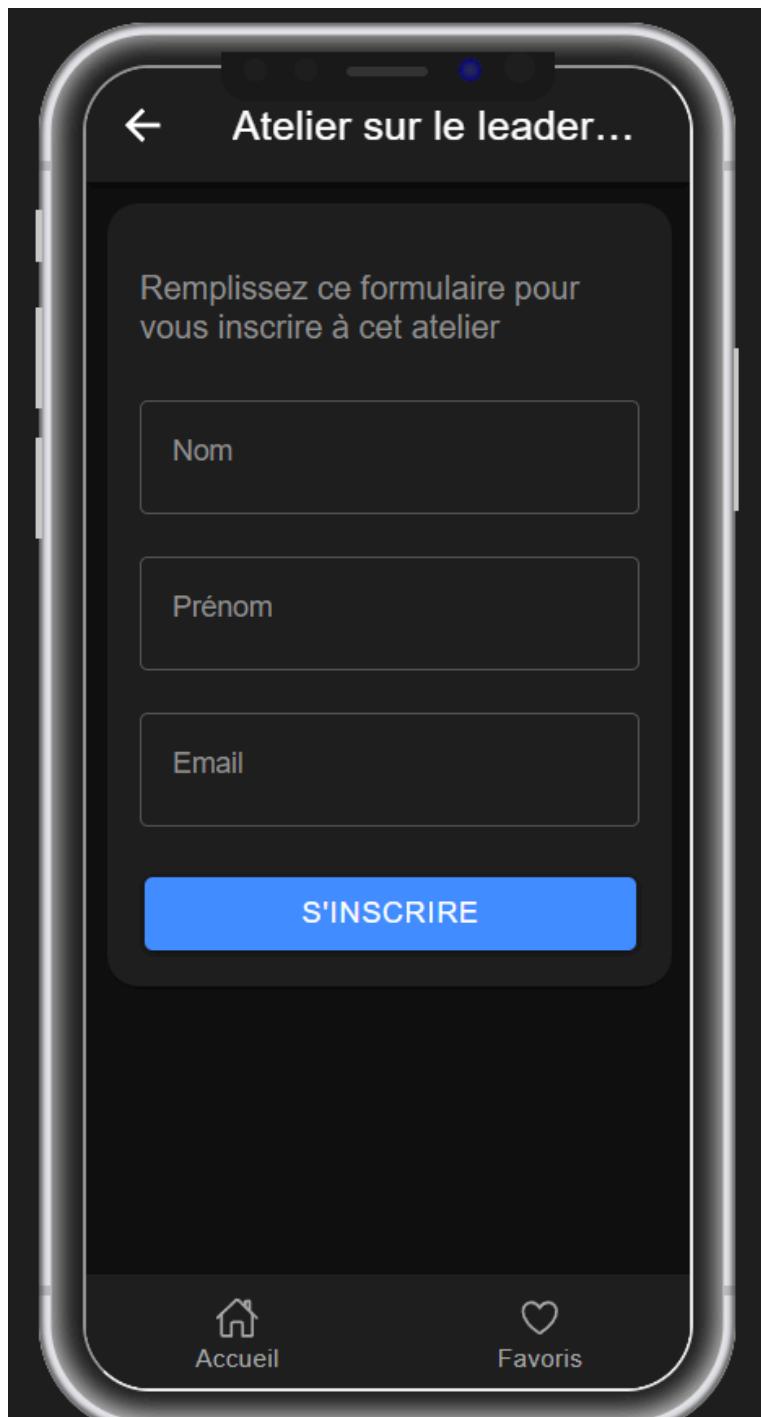
```
<!-- détails de l'atelier -->
<ion-card-content>
  <ion-list lines="none">
    <ion-item detail="false">
      <ion-icon name="calendar-outline" slot="start"></ion-icon>
      <ion-label>Date :&nbsp;{{leAtelier.date | date: 'dd/MM/yyyy'}}</ion-label>
    </ion-item>
    <ion-item detail="false">
      <ion-icon name="location-outline" slot="start"></ion-icon>
      <ion-label>Salle :&nbsp; {{leAtelier.salle}}</ion-label>
    </ion-item>
```

Inscription aux ateliers

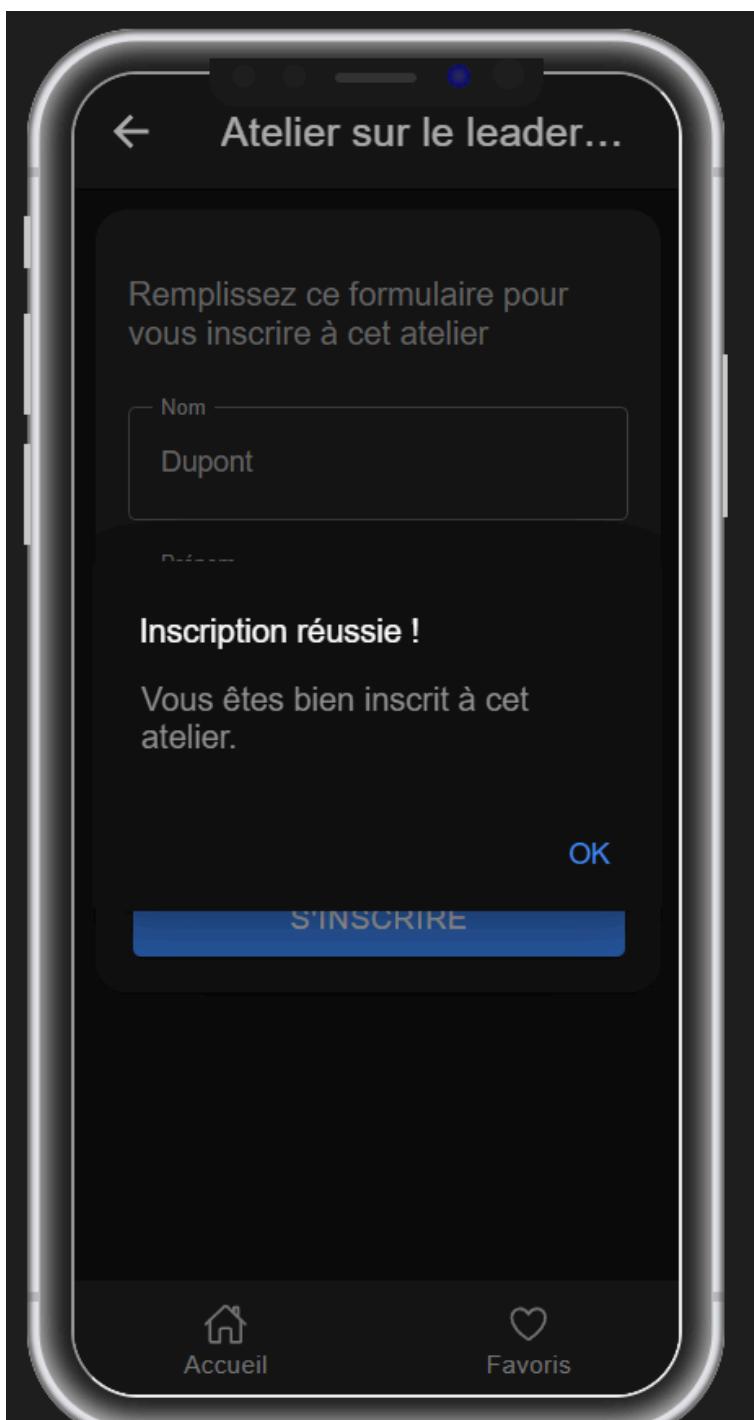
Sur cette page de détail d'un atelier, est affiché un bouton "S'inscrire".

En cliquant dessus l'utilisateur est redirigé vers un formulaire d'inscription à l'atelier.

Il est ainsi invité à saisir son nom, son prénom, et son email.



Une fois qu'il a rempli ses informations et validé, un message de confirmation de son inscription s'affiche.



Voici le fonctionnement.

Le formulaire sur la page HTML comporte un attribut "ngSubmit" renvoyant vers la fonction inscriptionSubmit().

```
<ion-card-content>
  <form (ngSubmit)="inscriptionSubmit()" novalidate>
    <ion-input label="Nom" name="nom" [(ngModel)]="nom" label-placement="floating" fill="outline">
      <br>
    <ion-input label="Prénom" name="prenom" [(ngModel)]="prenom" label-placement="floating" fill="outline">
      <br>
    <ion-input label="Email" name="email" [(ngModel)]="email" type="email" label-placement="floating" fill="outline">
      <br>
    <ion-button expand="block" id="present-alert" type="submit">S'inscrire</ion-button>
    <ion-alert trigger="present-alert" subHeader="Inscription réussie !" message="Vous êtes inscrit."></ion-alert>
  </form>
</ion-card-content>
```

Cette fonction est ainsi exécutée lors de la soumission du formulaire. La voici :

```
//fonction qui soumet le formulaire d'inscription
inscriptionSubmit(){
  const nom = this.nom;
  const prenom = this.prenom;
  const email = this.email;
  const idAtelier = this.leAtelier.id;

  //enregistrement des données
  this.hackatService.postInscriptionAtelier(nom, prenom, email, idAtelier);
}
```

Les valeurs des champs du formulaire sont récupérées, et en utilisant la méthode "postInscriptionAtelier" du service Ionic qui utilise l'API (voir 3ème route du tableau), les données sont enregistrées dans la base de données.

 Supprimer 29 Dupont Pierre dupontpr@gmail.com

1

Commentaires

La page des détails d'un atelier offre aussi une fonctionnalité de commentaires.

En effet, en dessous des informations de l'atelier, on retrouve une rubrique "Commentaires" affichant les commentaires relatifs à celui-ci déjà postés par des visiteurs. Ceux-ci sont composés du nom du visiteur ainsi que de son message.



Cette liste est récupérée grâce au service Ionic et à l'API (voir 4ème route du tableau).

```
<ion-card>
  <ion-card-header>
    <ion-card-title>Commentaires</ion-card-title>
  </ion-card-header>
  <!-- affichage des commentaires sur l'atelier -->
  <ion-card-content>
    <p *ngIf="lesCommentaires.length === 0">Aucun commentaire pour le moment.</p>
    <ion-list lines="full">
      <ion-item *ngFor="let unCommentaire of lesCommentaires">
        <ion-avatar slot="start">
          
        </ion-avatar>
        <ion-label class="ion-text-wrap">
          <h3>{{unCommentaire.nom}}</h3>
          <p>{{unCommentaire.libelle}}</p>
        </ion-label>
      </ion-item>
    </ion-list>
```

Sous cette rubrique, un bouton "Commenter" offre la possibilité à l'utilisateur, après avoir rempli le formulaire sur la page suivante, de publier lui aussi un commentaire sur l'atelier.



Comme le formulaire d'inscription, un message de confirmation s'affiche.

Même principe pour l'enregistrement des données en BDD grâce à la méthode postCommentaire().

```
<ion-card-content>
  <form (ngSubmit)="postCommentaire()" novalidate>
    <ion-input color="primary" label="Commentaire" name="libelle" [(ngModel)]="libelle">
      <br>
    <ion-input label="Nom" name="nom" [(ngModel)]="nom" label-placement="floating" type="text">
      <br>
    <ion-input label="Email" name="email" [(ngModel)]="email" type="email" label-placement="floating">
      <br>
    <ion-button expand="block" id="present-alert" type="submit">Valider</ion-button>
    <ion-alert trigger="present-alert" subHeader="Commentaire publié !" [buttons]="[<button>OK</button>]">
  </form>
</ion-card-content>
```

Contrôleur de la page :

```
//fonction qui soumet le formulaire et donc le commentaire
postCommentaire(){
  const nom = this.nom;
  const email = this.email;
  const libelle = this.libelle;
  const idAtelier = this.leAtelier.id;

  //enregistrement des données
  this.hackatService.postCommentaire(libelle, email, nom, idAtelier);
}
```

Service Ionic :

```
postCommentaire(libelle:string, email:string, nom:string, idAtelier:number){
  return new Promise((resolve) => {
    var url = "http://192.168.51.98:3000/commentaire/post";
    const body = { libelle, email, nom, idAtelier };
    this.http.post(url, body).subscribe((data) => {
      resolve(data);
    });
  })
}
```

Gestion des favoris

Pour finir, l'application permet également de mettre en favoris ou retirer des favoris un atelier.

C'est possible grâce à un bouton sur la page de détails d'un atelier.



Cette fonctionnalité est différente du reste de l'application car les données ne sont pas enregistrées en base. En effet, on ne gère pas la persistance des favoris d'un appareil à un autre.

Pour intégrer cette fonctionnalité, j'ai utilisé le système de stockage en local d'Ionic (<https://ionicframework.com/docs/angular/storage>).

Pour simplifier les choses, j'ai créé un service Ionic **StorageService**.

Dans ce service, j'ai créé 4 méthodes, les voici (avec en commentaire leur fonction) :

```
export class StorageServiceService {

  constructor() { }

  //Fonction pour ajouter un atelier dans le tableau d'ateliers favoris
  addToFavorites(unAtelier: any) {
    //Récupération des favoris actuels
    let lesFavoris = this.getFavorites();

    //On vérifie si l'atelier est déjà en favori
    const existeDeja = lesFavoris.some(a => a.id === unAtelier.id);

    //Si l'atelier n'existe pas déjà dans la liste des favoris, on l'ajoute
    if (!existeDeja) {
      lesFavoris.push(unAtelier);
      localStorage.setItem('ateliersFavoris', JSON.stringify(lesFavoris));
    } else {
      console.log("Cet atelier est déjà ajouté aux favoris !");
    }
  }

  //Fonction pour supprimer un atelier des favoris
  removeFromFavorites(idAtelier: number) {
    let lesFavoris = this.getFavorites();
    lesFavoris = lesFavoris.filter(atelier => atelier.id !== idAtelier);
    localStorage.setItem('ateliersFavoris', JSON.stringify(lesFavoris));
  }

  //Fonction pour récupérer le tableau d'ateliers favoris
  getFavorites(): any[] {
    const lesFavoris = localStorage.getItem('ateliersFavoris');
    return lesFavoris ? JSON.parse(lesFavoris) : [];
  }

  //Fonction pour vérifier si un atelier est en favori
  isFavorite(unAtelier: any): boolean {
    let lesFavoris = this.getFavorites();
    return lesFavoris.some(a => a.id === unAtelier.id);
  }
}
```

Ces méthodes sont ensuite utilisés sur la page comme ceci :

Lorsqu'on est sur la page, le bouton sera différent en fonction de si l'atelier est déjà en favori ou non (grâce à la méthode `isFavorite()`).

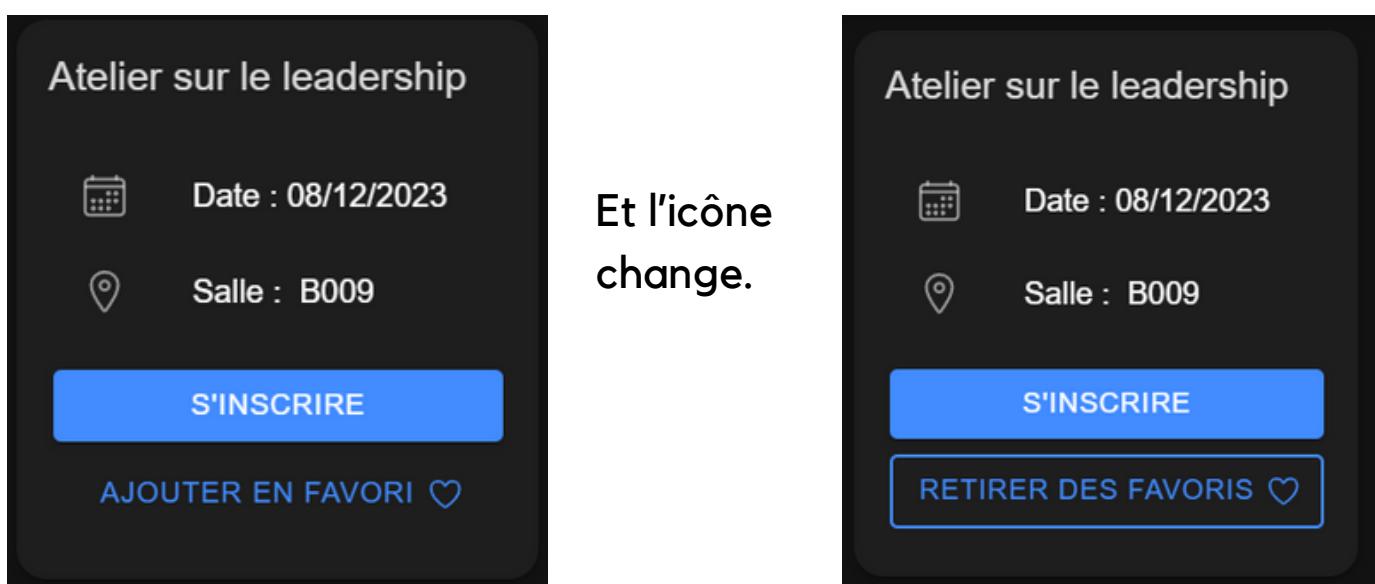
```
<!-- si l'atelier est déjà en favori, affichage du bouton de favori correspondant -->
<ion-button *ngIf="isFavorite(leAtelier) == false" fill="clear" expand="block" (click)="addToFavorites(leAtelier)">
  <ion-icon name="heart-outline" slot="end"></ion-icon>
  Ajouter en Favori
</ion-button>
<ion-button *ngIf="isFavorite(leAtelier)" fill="outline" expand="block" (click)="removeFromFavorites(leAtelier.id)">
  <ion-icon name="heart-outline" slot="end"></ion-icon>
  Retirer des favoris
</ion-button>
```

Le contrôleur exécutera ensuite la méthode correspondante : soit `addToFavorites()` soit `removeFromFavorites()`.

```
isFavorite(leAtelier: any): boolean {
  return this.storageService.isFavorite(leAtelier);
}

//fonction qui ajoute l'atelier en favoris et qui redirige vers la page qui les liste
addToFavorites(leAtelier: any){
  this.storageService.addToFavorites(leAtelier);
}

//fonction qui supprime l'atelier des favoris
removeFromFavorites(idAtelier: number){
  this.storageService.removeFromFavorites(idAtelier);
}
```



Enfin, dans le menu de navigation il y a un onglet "Favoris".
Celui-ci permet de diriger vers la page qui liste tous les ateliers mis en favoris par l'utilisateur.
Cette page est mise à jour automatiquement.



Sur cette page, c'est la méthode `getFavorites()` du service Ionic qui est utilisée. Elle parcourt l'ensemble des favoris stockés localement. Puis la vue les affiche. Ils sont évidemment cliquables.

```
this.lesFavoris = this.storageService.getFavorites();
```

Conclusion

Ce projet, réalisé en groupe, a été effectué durant toute la 2ème année de BTS SIO (2023-2024) durant les ateliers de professionnalisation et à la maison.

Il nous a permis de mettre en pratique les connaissances théoriques acquises en classe et de les appliquer dans un contexte réel. Cela nous a donné une expérience précieuse qui nous servira dans le monde professionnel.