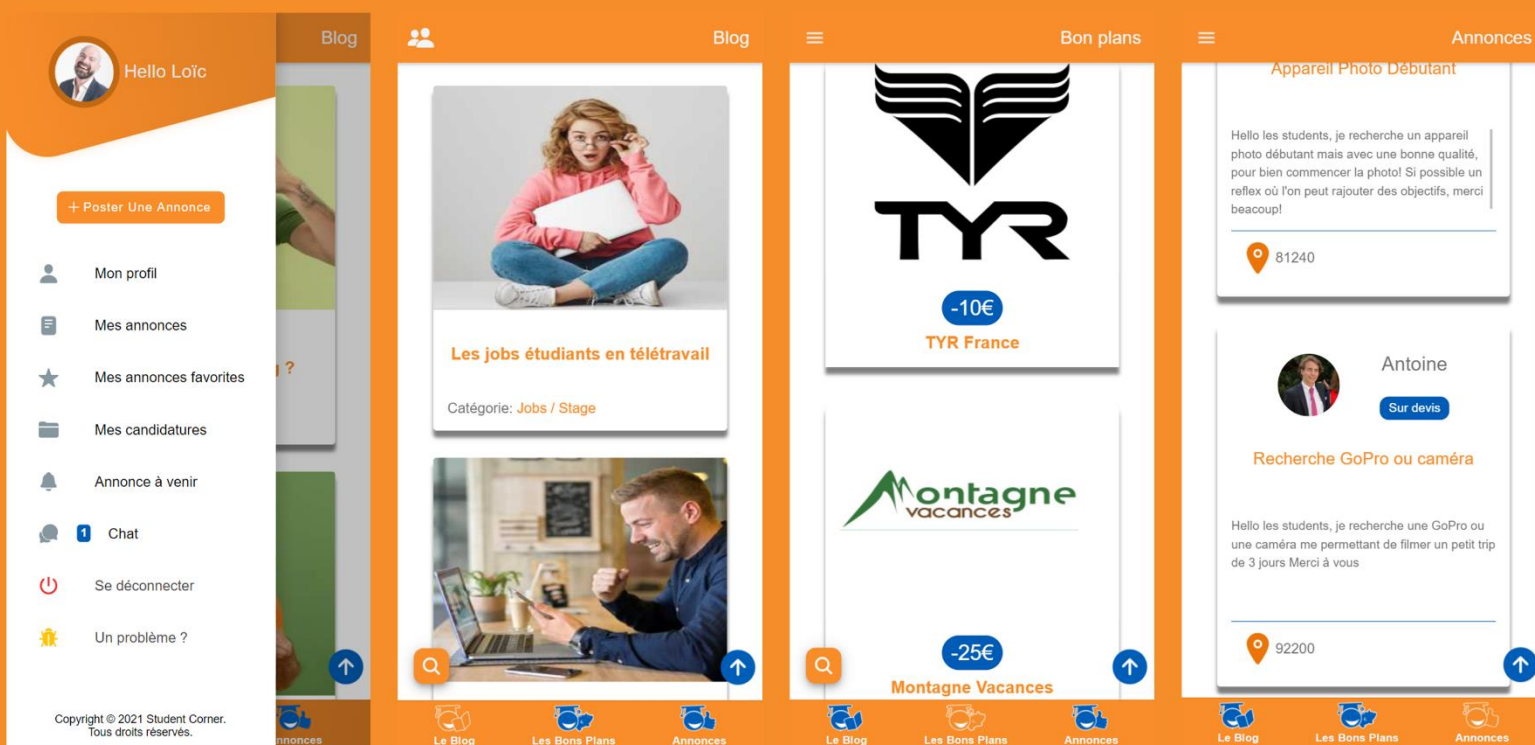




## Création de l'application Student Corner

**Stage** (Télétravail) : Du 01/03/21 au 26/04/21



# SOMMAIRE

Remerciements .....	2
1. Liste des compétences du référentiel qui sont couvertes par le projet : .....	3
Front-end : .....	3
A . Réaliser une interface web statique et adaptable .....	3
B . Développer une interface utilisateur web dynamique .....	4
Back-end : .....	5
C . Créer une base de données.....	5
D . Développer les composants d'accès aux données .....	5
E . Développer la partie back-end d'une application web ou web mobile .....	6
2. Résumé du projet : .....	7
3. Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet : .....	7
4. Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité et le web mobile : .....	10
A . Choix technologiques .....	10
5. Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité et le web mobile : .....	11
6. Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues) .....	35
7. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité .....	35
8. Description d'une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, à partir de site anglophone .....	36
9. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la traduction en français effectuée par le candidat sans traducteur automatique (environ 750 signes). .....	38
English .....	38
Français .....	39
9. Annexe .....	40
Exemples de Query Builder .....	40
Exemple de la mise en place des sockets sur l'application .....	41
Vue sur l'application en mobile et tablette .....	42

## Remerciements

Avant tout développement sur cette expérience professionnelle, il apparaît opportun de commencer ce rapport de stage par des remerciements, à ceux qui m'ont fait confiance, qui m'ont poussé à aller toujours plus loin et à ceux qui m'ont aidé à réaliser ce projet dont je suis si fière

Aussi, je remercie tout d'abord, mon maître de stage, Antoine Lambert ainsi que son associé, Antoine Gollety, qui ont su me faire confiance dès le départ et m'ont permis d'intégrer la startup Student Corner.

Je tiens aussi à remercier mon formateur Alain Merucci, sans qui tout cela n'aurait jamais été possible.

# 1. Liste des compétences du référentiel qui sont couvertes par le projet :

## Front-end :

### A . Réaliser une interface web statique et adaptable

#### Qu'est-ce qu'une interface web statique et adaptable ?

*Une interface web statique est une page web dont le contenu ne varie pas en fonction des caractéristiques de la demande, c'est-à-dire qu'à un moment donné tous les internautes qui demandent la page reçoivent le même contenu.*

L'application Student Corner comporte deux pages web statique et adaptable. Les pages blog et bon Plan appellent du contenu de la base de données via une api et l'intègrent avec une boucle sous forme de Card. Le contenu de ces pages reste le même peu importe l'utilisateur qui les consulte, mais change en fonction des articles et des bons plans qui sont ajoutés dans la base de données.

L'interface a été réalisée avec Vue 3 et Ionic 4 sur Visual Studio Code

J'appelle le contenu dans une méthode qui fait appel à une api que j'ai mise en place sur le serveur et qui push toutes les data dans un tableau que je récupère dans le Template.

J'ai intégré l'ossature dans ce Template avec HTML, les composants Ionic, ainsi que la directive v-for de vue.js pour boucler les cards.

## B . Développer une interface utilisateur web dynamique

### Qu'est-ce qu'une interface utilisateur web dynamique ?

*Une interface web dynamique est générée à la demande et son contenu varie en fonction des caractéristiques de la demande (heure, adresse IP de l'ordinateur du demandeur, formulaire rempli par le demandeur, etc.) qui ne sont connues qu'au moment de sa consultation.*

Sur l'application Student Corner, la page annonce, ainsi que toutes les autres pages de l'application qui sont disponibles uniquement après la connexion ou l'inscription de l'utilisateur sont des pages web dynamiques et s'adaptent en fonction de l'utilisateur actuellement connecté.

Par exemple un utilisateur non connecté sur la page Annonce n'aura accès qu'à la lecture des annonces, il ne pourra pas en ajouter à ses favoris, il ne pourra pas postuler, ni contacter la personne qui a posté l'annonce.

Autre fonctionnalité qui s'ajoute lors de la connexion, le menu déroulant, qui contient toutes les autres pages dynamiques de l'application, notamment la page de l'utilisateur où il peut modifier ses informations et prendre un selfie pour changer sa photo de profil.

Il y a aussi tous les pages relatives à l'utilisateur connecté et les annonces, celles qu'il a ajouté aux favoris, les annonces qu'il a posté, les annonces où il a postulé et bien entendu une page pour poster une annonce.

Se trouve aussi sur cette page le chat que j'ai mis en place dont je vous parlerais plus tard, ainsi qu'un bouton pour nous contacter en cas de soucis techniques, soit par le chat, soit par mail, et pour finir le bouton de déconnexion.

# Back-end :

## C . Créer une base de données

### Qu'est-ce qu'une base de données ?

*Une base de données permet de stocker et de retrouver des données structurées, Semi-structurées ou des données brutes ou de l'information, souvent en rapport avec un thème ou une activité*

L'application Student Corner utilise comme backend le serveur du site éponyme, qui lui tourne sous Symfony 4.

La base de données MySQL était par conséquent déjà créée.

J'ai dû y apporter quelques modifications, notamment lorsque j'ai eu pour mission de créer un Chat pour l'application, ainsi que pour le site web. J'y ai donc introduit trois entités supplémentaires : Message, Conversation, Participant.

L'entité message à une relation ManyToOne avec l'entité User et l'entité Conversation.

L'entité Conversation a une relation OneToMany avec l'entité Participant.

Et par conséquent l'entité participant à une relation ManyToOne avec les entités Conversation et User.

J'ai ensuite mis à jour la base de données pour qu'elle prenne en compte le changement dans mes entités.

## D . Développer les composants d'accès aux données

Afin de récupérer les données dont j'avais besoin pour créer l'application, j'ai intégré Api Platform au serveur afin de générer plus rapidement des liens d'api. J'y ai intégré plusieurs customs Controller afin de réaliser des actions, ou de récupérer certaines données via des Query Builder que j'ai créé dans certain repository, chose qui n'est pas possible de faire avec les fonctionnalités de base d'Api Platform.

## E . Développer la partie back-end d'une application web ou web mobile

Au niveau du Back j'ai dû développer quelque fonctionnalité sur le serveur pour le rendre compatible avec l'application, les plus gros points étant la connexion, où j'ai dû créer un firewall pour autoriser la connexion depuis l'application, ainsi qu'un Event Listener qui se déclenche lorsque la connexion est établie et qui retourne à l'application les données de l'utilisateur qui vient de faire la demande de connexion. J'ai aussi créé une méthode pour l'inscription, qui récupère les données envoyées lorsqu'un utilisateur remplit un formulaire d'inscription. Cette méthode, grâce aux données reçues, encode le password, crée un token propre à l'utilisateur, fait appel à une fonction qui ajoute une notification à l'utilisateur, fait appel à une autre fonction qui envoie un mail à l'utilisateur pour lui permettre de valider son compte, pour finir elle l'enregistre en base de données et retourne les data lié à cet utilisateur à l'application. J'ai aussi créé une méthode qui lorsque l'utilisateur prend un selfie depuis l'application pour changer sa photo de profil, intercepte les données de cette photo envoyée en base64, lui crée un nom unique, la décode, la déplace dans un dossier avatar du dossier public, et enregistre son nom dans la base de données.

J'ai aussi développé une méthode qui lorsqu'un utilisateur postule à l'annonce d'un autre utilisateur, ce dernier soit notifié par mail que quelqu'un a postulé à son annonce.

## 2. Résumé du projet :

L'entreprise Student Corner est une startup créée par des étudiants pour des étudiants. L'équipe de Student Corner a créé une plateforme web qui permet aux étudiants de lire un tas d'articles écrits par cette même équipe pour faciliter la vie des étudiants, de trouver des bons plans (beaucoup de codes promos sur pas mal de sites notamment Amazon, Babel etc..), une solution d'échange d'objets et de services entre étudiants ainsi qu'un système pour organiser et participer à des visios sur des thèmes en tous genres...

Le projet que j'ai réalisé au cours de mon stage, était de répondre au besoin de Student Corner de mettre en place une application Android et iOS, cette application reprend donc les fonctionnalités du site éponyme mentionné plus haut, ainsi que son design adapté en mobile et tablette.

## 3. Cahier des charges, expression des besoins, ou spécifications fonctionnelles du projet :

Je vais vous présenter dans cette partie le cahier des charges que l'on m'avait demandé de réaliser à mon début de stage. Celui-ci comprend deux versions, la première comprend ce que Student Corner voulait au minimum dans son application, la deuxième version est plus complète, je devais réaliser celle-ci si mes compétences me permettaient de réaliser la première version assez rapidement et facilement.

À savoir que j'ai fini l'entièreté de ce cahier des charges à la fin du premier mois de stage, c'est donc ajouté à cela une troisième version qui comprend un chat sur l'application ainsi que sur le site web Student corner.

### **Voici le cahier des charges :**

L'application Student Corner sortira en plusieurs versions, en fonction de ce qu'il sera possible de réaliser, et pour être sûr de pouvoir avoir une des versions finies aux termes des deux mois.

#### **V1 :**

Dans cette version, l'application sera composée de deux sections : Blog et BonPlan,



En bas de l'application, on pourra retrouver la sélection des menus sur une navbar en bas avec deux boutons. En haut à gauche figurera le logo de student corner, dans un semblant de navbar.

Dans la section bon plan on affichera la description le titre et la photo de tous les bons plans, avec possibilité de cliquer sur un bon plan pour l'afficher en entier

Exactement la même chose pour les articles, seul le design (et le contenu changera)

## **V2 :**

Pour la V2 on aimerait ajouter à l'application le système d'annonce, pour cela, cette version sera divisée en 2 versions.

### **V2.0**

La première partie concerne uniquement l'espace de connexion/inscription, afficher les annonces sans pouvoir interagir avec n'ayant pas tant d'intérêt.

Pour cela on ajoute une icône d'un bouton connexion Student Corner (comme les photos de profile sur studentcorner.fr) qui sera remplacer par la photo de profil de l'utilisateur lorsqu'il est connecté

Pour ce qui est du clique sur cette icône :

- Si l'utilisateur n'est pas connecté

L'utilisateur est redirigé vers une page d'inscription, (Nom prénom, Email, mdp) ou il peut s'inscrire, il y aura un petit bouton "J'ai déjà un compte", qui lui amènera vers une page de connexion (email, mdp)

- Si l'utilisateur est connecté

Alors une page s'affiche (ou une sidebar) avec des liens (uniquement les pages seront faites, pas le contenu qui sera à faire dans la version 2.1) vers différentes pages :

- Mes annonces,
- Mes annonces favorites
- Mes Candidatures
- Mes notifications
- Déconnexion

Peut-être :

- Mes articles Favoris
- Mes bons plans Favoris
- (Si ces deux pages prévoir le bouton d'ajout d'un article et BP dans les favoris sur la page de l'article)

Ensuite, il faudra rajouter un bouton dans la navbar du bas avec les annonces, puis créer une page où l'on voit un aperçu de toutes les annonces

Ainsi que la page pour afficher entièrement une annonce qui contiendra :

- Toutes les informations relatives à l'annonce comme sur le site,
- Un bouton pour contacter et poster une candidature comme sur le site, qui dans l'idéal affiche un popup ou simplement une Textarea pour envoyer le texte de sa candidature
- + Page ajouter une annonce

## V2.1

Dans cette version-là, il s'agit de finir toutes les pages créées plus ci-dessus ainsi que d'intégrer un design définitif si pas encore fait car tu es arrivé jusqu'ici rapidement :

- Mes annonces : afficher toutes les annonces qu'il a posté et si pas, CTA pour poster
- Mes annonces favorites : afficher toutes les annonces qu'il a mis en favori et si pas, CTA pour poster
- Mes Candidatures : afficher toutes ses candidatures et si pas, CTA vers les annonces
- Mes notifications : afficher toutes les notifs, il faudra faire le pont entre les notifs web et appli dans la table notif

## 4. Spécifications techniques du projet, élaborées par le candidat, y compris pour la sécurité et le web mobile :

### A . Choix technologiques

Au vu des besoins de l'entreprise Student Corner qui était de créer une application à la fois disponible sur les stores de Google et d'Apple, après discussion, il a été convenu pour ce projet d'utiliser côté Front: Ionic framework (V4) combiné avec Vue.js (V3) ainsi que Capacitor.

J'ai ainsi pu utiliser dans un projet Vue.js les nombreux composants Ionic spécialement développé pour mobile, ce qui augmente la rapidité et facilite radicalement le développement.

Quant à Capacitor, c'est grâce à lui que j'ai pu compiler les projets Android et Ios, il fait quasiment tout le travail, une fois le dossier Android ouvert avec Android Studio et le dossier Ios ouvert avec Xcode, il ne reste plus qu'à configurer que vraiment peu de chose avant de pouvoir Build l'Apk et l'Ipa (éventuellement aussi de corriger quelques bugs ne nécessitant que peu de recherche sur Google). Capacitor est aussi très utile pour ses plugins, pour cette application je n'ai eu à m'en servir que d'un seul, celui de la caméra, que j'utilise pour me servir de l'appareil photo du téléphone quand l'utilisateur veut changer sa photo de profil.

Côté back, il a été convenu que j'utilise le serveur du site déjà existant, en récupérant les données grâce à une api.

Le site étant développé avec Symfony, j'ai pu aisément me servir d'Api Platform pour réaliser cela. Je me suis servi d'Api Platform pour récupérer les données comme mentionnées ci-dessus, mais aussi pour en envoyer au serveur, modifier les données déjà existantes et en supprimer, avec les méthodes HTTP : GET, POST, PUT, PATCH et DELETE. J'ai aussi pu créer mes propres Controller et m'en servir avec lui.

Api platform intègre également Nelmio Cors qui permet de sécuriser l'api en lui spécifiant les origines qui auront accès aux données.

## 5. Réalisations du candidat comportant les extraits de code les plus significatifs et en les argumentant, y compris pour la sécurité et le web mobile :

Dans cette partie je vais donc vous présenter toutes les réalisations que j'ai effectué sur ce projet au cours de mon stage, en les argumentant et en vous présentant des exemples.

Pour débiter le projet, j'ai donc suivi le cahier des charges présenté page 7, et ai donc commencé par la première version.

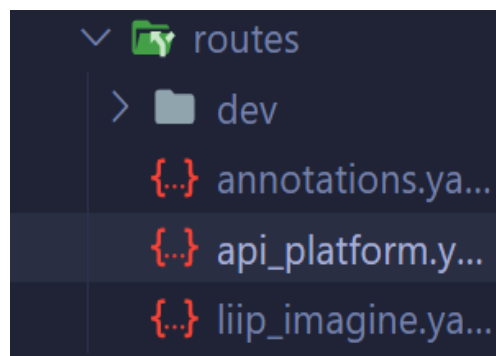
Pour ce faire je devais donc commencer par récupérer les données que j'allais exploiter par la suite.

Le serveur étant développé avec Symfony, j'ai donc décidé d'intégrer Api Platform.

Pour installer cette librairie sur mon projet, j'ai entré cette ligne de commandes dans ma CLI (command line interface) :

```
PS D:\student-corner\Site\student-corner\scgit> composer require api
```

La route par défaut qu'utilise Api Platform étant /api, pour plus de sécurité il est nécessaire de changer celle-ci pour ceci rendons-nous dans le fichier api\_platform.yaml du dossier routes ci-dessous



Et modifions la route comme ceci

```
1  api_platform:
2      resource: .
3      type: api_platform
4      prefix: /api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb
5
```

Pour la suite nous allons maintenant gérer la configuration liée aux CORS (Cross-origin resource sharing) qui permet de sécuriser l'accès aux données en lui spécifiant les origines qui y auront accès.

Api Platform offre une gestion du CORS assez simple grâce au `nelmio cors`. Ce fichier se présente ainsi.

```
1  nelmio_cors:
2      defaults:
3          origin_regex: true
4          allow_origin: ['%env(CORS_ALLOW_ORIGIN)%']
5          allow_methods: ['GET', 'OPTIONS', 'POST', 'PUT', 'PATCH', 'DELETE']
6          allow_headers: ['Content-Type', 'Authorization']
7          expose_headers: ['Link']
8          max_age: 3600
9      paths:
10         '^/': null
```

Dans ce fichier nous pouvons modifier au besoin les méthodes HTTP, les autorisations, et surtout nous pouvons gérer les origines qui auront accès aux données. Pour la suite du développement nous autoriserons toutes les origines en spécifiant dans le fichier `.env` du projet

```
51  ###> nelmio/cors-bundle ###
52  CORS_ALLOW_ORIGIN=/*/
53  ###< nelmio/cors-bundle ###
```

Maintenant qu'Api Platform est installé et configuré sur notre projet, nous allons pouvoir commencer à l'utiliser. Pour ce faire rendons nous dans une Entité et annotons-la de cette ligne

```
27  * @ApiResponse()
```

Sans oublier d'y inclure son Namespace

```
19  use ApiPlatform\Core\Annotation\ApiResource;
```

Cette Entité est désormais utilisable avec toutes les méthodes autorisées via les liens suivants

CodePromo			⌵
GET	/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/code_promos	Retrieves the collection of CodePromo resources.	
POST	/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/code_promos	Creates a CodePromo resource.	
GET	/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/code_promos/{id}	Retrieves a CodePromo resource.	
PUT	/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/code_promos/{id}	Replaces the CodePromo resource.	
DELETE	/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/code_promos/{id}	Removes the CodePromo resource.	
PATCH	/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/code_promos/{id}	Updates the CodePromo resource.	

Pour plus d'optimisations, nous allons mettre en place une pagination, ainsi qu'un groupe qui contiendra uniquement les champs voulus lorsque l'on appellera cette Entité en GET, nous allons aussi lui donner un attribut qui appellera les données en DESC (ordre décroissant) par rapport à leur date d'ajout à la base de données. Nous allons aussi lui spécifier uniquement les méthodes que nous souhaitons mettre en place pour l'instant. Pour ce faire ajouter ses lignes ci aux annotations

```
27 * @ApiResponse(  
28 *     attributes={  
29 *         "order"={"AllDateAjout": "DESC"}  
30 *     },  
31 *     paginationItemsPerPage=5,  
32 *     normalizationContext={"groups"={"read:bp"}},  
33 *     collectionOperations={"get"},  
34 *     itemOperations={"get"}  
35 * )
```

Sans oublier bien sûr de rajouter le Namespace pour les Groups

```
22 use Symfony\Component\Serializer\Annotation\Groups;
```

Nous pouvons maintenant spécifier les champs que l'on veut appeler dans notre groupe en annotant les champs voulus comme ceci

```
* @Groups({"read:bp"})
```

Nous allons maintenant répéter ceci pour les Entités dont nous voulons récupérer les données, c'est-à-dire les Entités Article, BonPlan et Annonce qui représenteront les trois pages principales de l'application.

Nous sommes maintenant capables de récupérer les données requis au développement de la première version de l'application.

Nous allons donc pouvoir initialiser le projet Ionic.

Pour pouvoir installer Ionic sur sa machine, on aura besoin tout d'abord de NPM (Node Package Manager) donc dans un premier temps il nous faudra installer Node.js sur notre machine. Rendons-nous donc sur le site officiel de Node.js et téléchargeons la version LTS (Long-Term Support). Une fois cela fait, nous sommes maintenant capables d'utiliser NPM sur notre machine. Ouvrons un terminal peu importe où, et entrons la commande suivante

```
PS D:\student-corner\Site\student-corner\scgit> npm install -g @ionic/cli
```

Cette commande nous permet d'installer la CLI d'Ionic globalement sur notre machine.

Une fois cela fait nous pouvons initialiser le projet avec la commande suivante

```
PS C:\Users\33666\Desktop> ionic start myProject
```

À la suite de cela Ionic nous proposera plusieurs choix, celui du framework JS que l'on souhaite utiliser, et celui de la base sur laquelle on veut initier notre projet (avec un menu déjà tout prêt, une tab bar, commencer avec un projet vide...)

```
PS C:\Users\33666\Desktop> ionic start myProject
```

Pick a framework!

Please select the JavaScript framework to use for your new app. To bypass this prompt next time, supply a value for the `--type` option.

? Framework:

Angular		<a href="https://angular.io">https://angular.io</a>
React		<a href="https://reactjs.org">https://reactjs.org</a>
> Vue		<a href="https://vuejs.org">https://vuejs.org</a>

Let's pick the perfect starter template!

Starter templates are ready-to-go Ionic apps that come packed with everything you need to build your app. To bypass this prompt next time, supply `template`, the second argument to `ionic start`.

? Starter template:

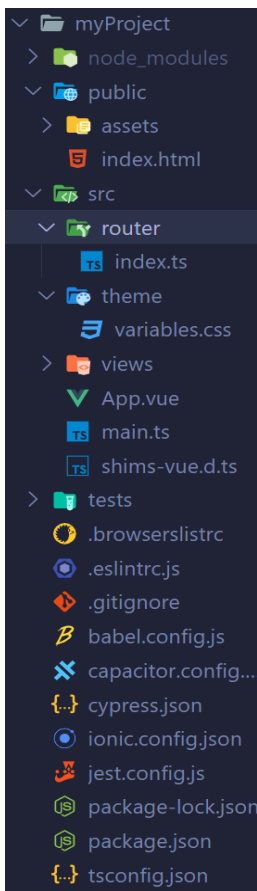
tabs		A starting project with a simple tabbed interface
sidemenu		A starting project with a side menu with navigation in the content area
> blank		A blank starter project
list		A starting project with a list

Dans notre cas nous allons choisir comme framework Vue.js et partir sur une base vide (Blank). Une fois cela fait, Ionic va s'installer, une fois fini il ne reste plus qu'à se déplacer dans le dossier de notre projet avec la commande suivante

```
PS C:\Users\33666\Desktop> cd myProject
```

A présent nous allons pouvoir configurer notre projet pour pouvoir le développer en JS, Ionic et Vue.js étant installé de base pour pouvoir être développer en TypeScript, et ne sachant pas développer en TypeScript, j'ai fait le choix de le passer en JS.

Pour ce faire nous devons réaliser quelques petites modifications. Voyons d'abord comment se présente un projet Ionic/Vue.js après sa création.



Nous avons un dossier Public qui contiendra le index.html.

Et le dossier sur lequel nous allons le plus travailler, le dossier src, celui-ci contiendra le fichier router/index.ts où se trouvera les routes de notre application, le dossier theme qui contiendra tout le style de notre application, le dossier views qui va contenir toutes les vues de l'application. Se trouve aussi dans ce dossier, le fichier App.vue qui va contenir la Template de base parente à toutes les vues que l'on va créer. Et pour finir se trouve le fichier main.ts qui lui va s'occuper d'importer toutes les dépendances dont l'application aura besoin globalement, puis de monter l'application dans la div #app.

Dans un premier temps pour convertir notre projet en JS nous allons devoir exécuter une commande dans notre projet pour pouvoir supprimer les dépendances TypeScript

```
PS C:\Users\33666\Desktop\myProject> npm uninstall --save typescript @types/jest @typescript-eslint/eslint-plugin @typescript-eslint/parser @vue/cli-plugin-typescript @vue/eslint-config-typescript
```

Une fois cela fait nous pouvons changer l'extension des fichiers router/index.ts à .js, nous allons faire de même pour le fichier main.ts . Nous avons aussi besoin de supprimer le fichier src/shims-vue.d.ts. Une fois fait rendons nous dans le fichier .eslintrc.js à la racine du projet, ou nous allons supprimer les lignes suivante :

```
@vue/typescript/recommended
```

```
@typescript-eslint/no-explicit-any: 'off',
```

Il ne nous reste plus qu'à supprimer le `lang="ts"` dans le fichier App.vue, ainsi que le `Array<RouteRecordRaw>` dans le fichier router/index.js.

À présent notre projet est complètement développable en JS.



À partir de maintenant nous pouvons commencer à développer le projet, nous allons commencer par créer un dossier Components et y inclure un fichier BaseLayout.vue, dans ce component nous allons créer une architecture de base avec les composants Ionic pour éviter de se répéter à chaque page. Cette architecture se composer ainsi

```

1  <template>
2    <ion-page>
3      <ion-header>
4        <ion-toolbar v-if="isNotInConfig()">
5          <ion-icon slot="start" color="light" v-if="nom == ''
6            :src="people" @click="() => router.push('/login')" class="loginIcon"></ion-icon>
7          <ion-buttons slot="start" v-if="nom != ''>
8            <div v-if="backButton === true"></div>
9            <ion-menu-button color="light" v-else"></ion-menu-button>
10         </ion-buttons>
11         <ion-buttons v-if="backButton === true" slot="start">
12           <ion-back-button :default-href="pageDefaultBackLink" color="light"></ion-back-button>
13         </ion-buttons>
14         <ion-title slot="end" color="light">{{ pageTitle }}</ion-title>
15       </ion-toolbar>
16     </ion-header>
17     <ion-content fullscreen="true" ref="main">
18       <slot />
19       <!-- <ion-icon :src="arrowUpCircle" class="fixed goTop" @click="ScrollToTop()" v-if="backToTop === true"></ion-
20       <ion-icon :src="arrowUpCircle" class="fixed goTop" @click="ScrollToTop()" v-if="showButton === 1"></ion-icon>
21     </ion-content>
22   </ion-page>
23 </template>

```

Nous avons donc le component IonPage qui contient le component IonHeader ainsi que le component IonContent dans lequel nous allons venir greffer un <slot /> ou viendra se placer nos différentes vues appeler selon la route sélectionnée.

En plus de cela nous allons y introduire des props grâce auquel nous pourrons faire passer à ce component des valeurs depuis d'autre vue, notamment le IonTitle.

```

props: ['pageTitle', 'pageDefaultBackLink', 'bool', 'bbBool'],

```

Nous avons donc ici le premier props qui concerne le titre de la page, la deuxième props concerne le lien de la page ou devra mener le bouton de retour en arrière, la troisième ainsi que la quatrième sont des booléen que je fais passer à ce component pour afficher ou non le bouton pour revenir en arrière et le bouton pour faire un scroller vers le haut de la page.

Maintenant ce component pourra être appelé comme ceci dans d'autre vue.

```

<base-layout page-default-back-link="/blog" :page-title="title" :bb-bool="true">

```

Nous ouvrons la balise comme ceci, nous lui passons des props, nous pouvons ensuite la refermer, puis y insérer notre contenu à l'intérieur.

Nous allons dès à présent nous occuper de mettre en place le style. Nous allons inclure un fichier `core.css` au sein du dossier `theme`, dans ce fichier se trouvera tout le code CSS que l'on mettra en place pour l'application. Une fois cela fait nous avons besoin de l'importer dans le fichier `main.js` comme ceci

```
29 | import './theme/core.css';
```

Ionic dispose également d'un fichier `variable.css` dans le dossier `theme`, où l'on pourra y mettre en place des variables de couleur réutilisable dans l'ensemble de l'application sur les composants Ionic, il y en avait une par exemple dans la page ci-dessus, dans le composant `IonTitle`, il y a `color="light"` de mentionner, cela fait référence à la variable `light` du fichier `variable.css`. J'ai donc récupéré le thème couleur du site Student Corner et généré le fichier `variable.css` en fonction de celle-ci. Voici un exemple de variable

```
5  :root {
6  |     --ion-color-primary: #f98d2a;
7  |     --ion-color-primary-rgb: 249,141,42;
8  |     --ion-color-primary-contrast: #ffffff;
9  |     --ion-color-primary-contrast-rgb: 255,255,255;
10 |     --ion-color-primary-shade: #db7c25;
11 |     --ion-color-primary-tint: #fa983f;
```

Nous pouvons donc voir ici que la variable `primary` comporte plusieurs couleurs, il y a sa couleur principale, son contraste, son ombre ainsi que `primary-tint`, couleur un peu plus claire qui dont je ne me suis pas servi durant ce projet.

Nous allons donc maintenant passer à la création des vues, pour que cela soit plus explicite, j'ai supprimé le dossier `views` et l'ai remplacé par un dossier `pages`.

J'y ai inclus dans celui-ci trois dossiers, une pour chaque page du site, un dossier `Blog`, `Bp` (pour bon plan) et un dossier `Annonce`. J'y ai inclus dans chaque dossier un fichier du même nom avec pour extension `.vue`, également dans chaque dossier j'y ai rajouté un fichier `detail.vue`. Chaque vue avec `Vue.js` se présente ainsi, il y a en haut deux balises `template`, qui doivent obligatoirement contenir une `div` qui elle contiendra le contenu de la vue. Et plus bas se trouve une balise `script` qui elle importera des dépendances ou des composants si elle en a besoin, et ensuite exportera le reste du script qui contiendra les composants importés plus haut, les méthodes, les datas, les props ect...

Une fois les pages créées nous allons pouvoir créer leurs routes, Ionic est très bien fait pour cela et nous à préparer les bases de `Vue.js`, notamment le router, nous n'avons plus qu'à l'utiliser et à créer les routes, pour cela direction le fichier `router/index.js` où nous allons devoir dans un premier temps importer la vue que l'on souhaite relier à une route

```
2 | import blog from '../pages/blog/blog.vue';
```

Une fois la vue importer, il ne nous rester plus qu'à la relier à une route comme ceci

```
27   const routes = [  
28     {  
29       path: '/',  
30       redirect: '/blog'  
31     },  
32     {  
33       name: 'blog',  
34       path: '/blog',  
35       component: blog  
36     },
```

Ici nous avons donc une route qui a le chemin "/" qui redirige vers le chemin "/blog", ainsi qu'une route qui a pour nom "blog", comme chemin "/blog" et qui utilisera la vue blog importer plus haut. Il ne nous reste plus qu'à répéter cela avec les trois vues que nous avons créé.

Nous pouvons maintenant utiliser ses routes, pour cela nous allons créer une tabs de navigation en bas de la page à l'aide du component Ionic IonTabs. Dans cette tabs se trouvera trois boutons qui chacun utilisera la fonction du router .push() qui redirigera sur chacune des routes créées précédemment. Chacun des boutons aura aussi deux petits logos que l'on m'a fourni pour chacune des pages. Un étant orange et l'autre bleu, je peux comme ceci écouter la route sur laquelle se trouve l'utilisateur et changer la couleur du logo, pour que l'utilisateur ne soit pas perdu en navigant sur l'application.

Voici un extrait de la tabs

```
<ion-tabs v-if="isNotInConfig()">  
  <ion-tab-bar slot="bottom" id="tabs">  
  
    <div id="divTabs1" class="divTabs">  
      <ion-tab-button ref="ombre1" @click="() => router.push('/blog')" class="tabsBtn ombre">  
        <div ref="logo3b" class="">  
          <br>  
          <ion-label class="colorPrimaryLogoTabs">Blog</ion-label>  
        </div>  
  
        <div class="none" ref="logo3o">  
          <br>  
          <ion-label class="colorPrimaryLogoTabs">Blog</ion-label>  
        </div>  
      </ion-tab-button>  
    </div>
```

On peut y voir ici le component IonTabButton qui s'occupe de push sur la route "/blog" avec à l'intérieur les deux logos que j'ai mentionné plus haut

Suite à cela, il a fallu créer des fonctions qui allait rajouter un display none sur les boutons voulu, puis grâce à la méthode watch de Vue.js il a été relativement facile d'écouter les routes comme ceci

```
watch:{
  $route(){
    if(this.$route.name == "blog"){
      this.logo3b();
      if(localStorage.getItem("User") !== null){
        this.removeAll();
        this.loadedConvs();
      }
    }
  }
}
```

On peut voir ici le méthode watch qui écoute les routes de l'application, et avec une condition, si le nom de la route est égal à blog alors l'application lance la fonction logo3b() qui rendra le logo de cette route orange, et rendra les autres logos bleu.

Maintenant que la navigation est opérationnelle, nous allons pouvoir commencer à ajouter du contenu aux trois pages, chaque page possédant le même schéma, je vais vous expliquer une fois la méthode utilisée.

Dans un premier temps nous allons installer Axios qui permet d'effectuer des requête http GET assez facilement. Pour installer Axios il suffit de taper cette ligne de commande dans le projet

```
PS D:\student-corner\Appli\student-corner> npm install axios
```

Une fois cela fait nous pouvons l'importer comme ceci dans notre vue

```
88 import axios from 'axios';
```

Maintenant grâce à Axios et les liens d'api précédemment créés, nous allons pouvoir appeler les données relatives à chaque page et les mettre dans un tableau JS comme ceci

Dans un premier temp nous allons créer ce tableau

```
data(){
  return{
    url: 'https://www.studentcorner.fr/media/cache/blogPetit/blog/logo-',
    blogs: [],
    page: 1,
  }
}
```

Dans les data de la vue nous allons donc y insérer le tableau blogs, nous allons aussi y placer le chemin vers le dossier où se trouve les images du blog sur le serveur, ainsi qu'une variable page qui sera initialisée à 1

Une fois cela fait, nous n'avons plus qu'à créer la méthode qui va push les données dans le tableau

```
124     methods: {
125         loadedData(page){
126             setTimeout(async() => {
127                 let url = 'https://www.studentcorner.fr/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/articles?page='+page;
128                 axios.get (url) .then ((response) => {
129                     var count = response.data['hydra:member'].length
130                     for(var i = 0; i < count; i++){
131                         this.blogs.push(response.data['hydra:member'][i])
132                     }
133                     this.date = new Date();
134                 }). catch (error => {
135                     console.log (error);
136                     this.errorVue = true;
137                     this.loading = false;
138                     setInterval(this.reloadAfterCounter, 1000);
139                 }).finally(() => this.loading = false, this.errorVue = false );
140             })
141         },
```

Cette méthode prend donc un paramètre qui sera le numéro de la page à appeler, qui par défaut comme mentionné précédemment sera de 1. Elle s'exécute de façon asynchrone, c'est-à-dire que même si cette fonction met du temps à s'exécuter, elle ne bloquera pas l'exécution d'autres méthodes qui pourront le faire en parallèle. Cette méthode lance donc une requête au lien d'api spécifié grâce à Axios et nous donne une réponse, la prochaine étape est de créer une boucle avec les éléments récupérés en initialisant une variable count qui récupèrera le nombre d'article que contiendra la réponse, puis dans la boucle le tableau créé précédemment sera incrémenté d'un article à chaque itération de la boucle. Vient ensuite l'initialisation d'une variable date qui prendra comme valeur la date du moment de l'exécution de la requête, Ce qui nous permet de n'afficher que les articles où la date du jour est inférieure à la date de publication. Vient ensuite la gestion des erreurs, si la requête ne réussit pas son exécution, alors une vue apparaîtra, celle-ci demandera à l'utilisateur de vérifier sa connexion internet, il y a aussi un timer qui relance la requête toutes les 10 secondes. Si la requête s'exécute correctement la variable booléenne loading passe à false, sachant qu'à l'arrivée sur la page celle-ci est à true est affiche une page blanche avec un loader que j'ai créée en CSS.

Nous pouvons ensuite lancer l'exécution de cette méthode au chargement de la vue grâce à la méthode mounted() de Vue.js

```
120     mounted () {
121         this.loadedData(this.page)
122         this.getCategoryBp()
123     },
```

Maintenant que notre tableau est rempli de données, nous pouvons donc les intégrer visuellement dans la template, pour ce faire nous allons utiliser le component IonCard de Ionic et boucler les éléments du tableau grâce à la directive v-for de Vue.js

```
<ion-grid fixed>
  <ion-row>
    <ion-col size="12" size-xs="12" size-sm="6" size-md="6" size-lg="4"
      v-for="article in blogs.filter((article) => formatCompareDate(date) >
        formatCompareDate(article.dateAjout))" :key="article.id">
      <ion-card>
        <div @click="() => router.push(`/blog/${article.id}`)">
          
          <ion-card-header>
            <ion-card-subtitle>
              Posté le {{ format_date(article.dateAjout) }}<br>
            </ion-card-subtitle>

            <div class="annonceTitle"><h2 class="colorPrimaryBlog">{{article.title}}</h2></div>

          </ion-card-header>
        </div>
        <ion-card-content @click="() => router.push(`/blog/category/${article.category.id}`)">
          <div class="contentCard">
            <h2> Catégorie: <span class="tags">{{ article.category.name }}</span></h2>
          </div>
        </ion-card-content>
      </ion-card>
    </ion-col>
  </ion-row>
</ion-grid>
```

J'ai donc mis en place une responsive grid grâce au composants IonGrid, IonRow et IonCol, ainsi le design et l'ergonomie sont aussi adaptable sur moyenne et grandes tablettes. J'ai donc bouclé sur les IonCol chaque élément du tableau blogs, en y filtrant avec une méthode que j'ai mis en place qui format une date donnée à un certain format. Je l'utilise pour formater la date de la requête instancié dans une variable précédemment ainsi que la date de publication de l'article, si cette dernière est supérieure à la date de la requête, l'élément ne s'affiche pas. Vient ensuite l'IonCard qui aux cliques de celle-ci redirige vers la page détail en passant comme paramètre à la route l'id de l'article. Puis j'affiche le contenu (son image, sa date de publication ainsi que sa catégorie).

À la suite de cela, pour appeler le contenu suivant (page suivante) j'ai mis en place un système d'infinite scroll grâce au component Ionic éponyme ainsi qu'à une méthode que j'ai mise en place. Pour ce faire nous devons premièrement appeler ce component dans la template pour que quand l'utilisateur arrive en bas de la page, le component lance la fonction que l'on lui a précisé.

```
<ion-infinite-scroll threshold="5%" id="infinite-scroll" @ionInfinite="loadedMoreData($event)">
  <ion-infinite-scroll-content loading-spinner="lines" loading-text="Chargements...">
  </ion-infinite-scroll-content>
</ion-infinite-scroll>
```

Ainsi une fois que l'utilisateur a atteint le bas de la page, la méthode loadedMoreData() s'exécute.

La méthode `loadedMoreData()` incrémentera la variable `page` de 1 et relancera simplement la méthode `loadedData()`, ainsi le tableau `blogs` se retrouvera implémenté de 5 articles en plus, et ainsi de suite.

```
144         loadedMoreData(event){
145             this.page++
146             this.loadedData(this.page)
147             setTimeout(() => {
148                 event.target.complete();
149             }, 1500);
150         },
```

Une fois les datas chargées l'animation du loader s'arrête 1 seconde et demie après.

Pour éviter de répéter des méthodes inutilement dans chaque page de l'application, j'ai mis en place une mixin dans le fichier `main.js`, grâce à celle-ci nous pouvons écrire des méthodes ainsi que des variables à l'intérieur qui seront global, c'est-à-dire réutilisable dans chaque page de l'application.

```
36     app.mixin({
37 >     data(){ ...
49     },
50     methods:{
51 >     user(){ ...
64     },
65 >     marquerCommeLu(linkConvId){ ...
82     },
83 >     format_date(value){ ...
97     },
98 >     navigate(url){ ...
100     },
101     }
102
103 });
```

J'y ai placé plusieurs méthodes, notamment une pour formater la date, une pour ouvrir le navigateur du téléphone au clique d'un lien, et d'autre méthode dont je vous parlerais par la suite, qui me serve à mettre en place une sorte de session pour l'utilisateur.

Pour finir cette première version, il ne restait plus qu'à intégrer le contenu dans les pages détails de chaque page. Pour ce faire comme mentionné précédemment, j'ai mis une méthode qui demande au router d'aller sur la vue détail, en lui passant en paramètre l'id de l'article cliqué. Une fois sur cette page je n'ai plus qu'à récupérer cet Id comme ceci

```
122 blogId: this.$route.params.id,
```

Une fois cet Id récupérer, je n'ai plus qu'à envoyer une requête de la même façon que précédemment, la seule différence, c'est qu'au lieu de faire passer un numéro de page au lien, je lui fais passer l'id, exemple : /article/1.

Pour le reste je récupère les valeurs et les intègre dans la template de la même façon que précédemment, le infinite scroll en moins bien sûr.

Pour finaliser cette première version j'y ai ajouté un peu de design, je n'ai pas vraiment réalisé de maquette, j'y suis allé un peu au feeling, en fonction des composants Ionic, et surtout bien entendu en respectant l'esprit de site déjà existant.

La première version étant finie, nous pouvons maintenant commencer la seconde. Les principales nouveautés de celle-ci tourneront principalement autour d'un seul point, la possibilité pour l'utilisateur de se connecter/s'inscrire et d'interagir avec l'application avec les mêmes fonctionnalités que le site web existant. Toutes ses fonctionnalités supplémentaires se trouveront dans un menu déroulant créé avec le composant IonMenu et qui ne sera accessible que si l'utilisateur est connecté, grâce à une méthode donc je vous parlerais plus bas

Dans un premier temps, mettons donc en place l'inscription. Pour ce faire, j'ai donc créé un dossier security au sein du dossier pages, dans lequel se trouvera une page pour l'inscription, ainsi qu'une autre pour la connexion. Pour l'instant nous allons donc mettre en forme la page inscription dans laquelle nous allons placer un formulaire dont nous récupérerons la valeur des champs grâce à la directive v-model de Vue.js qui seront placés dans des variables.

```
37 <ion-item class="width80">
38   <ion-label color="dark">Email</ion-label>
39   <ion-input required color="dark" type="mail" placeholder="Adresse email" v-model="email"></ion-input>
40 </ion-item>
```



```

75     data(){
76         return{
77             loading: false,
78             email: "",
79             password: "",
80             nom: "",
81             prenom: "",
82             error: "",
83             success: ""
84         };
85     };

```

Une fois les champs remplis par l'utilisateur, au clique du bouton de validation, cela activera une fonction register()

```

88     register(){
89         if(this.password.length < 7){
90             this.error = "Votre mot de passe doit contenir au minimum 7 caractères"
91         }else{
92             setTimeout(async() => {
93                 // this.loading = true;
94                 let dataRegister = {
95                     Nom: this.nom,
96                     Prenom: this.prenom,
97                     email: this.email,
98                     password: this.password
99                 };
100                 var dataPost = JSON.stringify(dataRegister);
101                 let xmlhttp = new XMLHttpRequest()
102                 xmlhttp.open("POST", "https://www.studentcorner.fr/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/users/create");
103                 xmlhttp.onload = () =>{
104                     const res = JSON.parse(xmlhttp.response);
105                     if(xmlhttp.status == 400){
106                         this.success = "";
107                         this.error = res['hydra:description'];
108                     }else{
109                         this.error = "";
110                         this.success = "Hello "+res['hydra:member'][1].Prenom+", ton compte a bien été enregistré !";
111                         const user = {
112                             id: res['hydra:member'][1].id,
113                             nom: res['hydra:member'][1].Nom,
114                             prenom: res['hydra:member'][1].Prenom,
115                             avatar: res['hydra:member'][1].avatar,
116                             roles: res['hydra:member'][1].roles
117                         }
118                         localStorage.setItem("User", JSON.stringify(user));
119                     }
120                 }
121                 xmlhttp.setRequestHeader("Content-Type", "application/json");
122                 xmlhttp.send(dataPost)
123             })
124         }
125     },
126     reload(){
127         window.location.assign("/");
128     }

```

Dans un premier temps nous pouvons voir dans cette fonction qu'elle refuse la requête si le mot de passe entré par l'utilisateur comporte moins de 7 caractères, si c'est le cas elle lancera donc une fonction asynchrone qui créera un tableau associative contenant les valeurs des champs renseignés par l'utilisateur, puis les convertira au format JSON (JavaScript Object Notations), format auquel je dois envoyer les données au serveur. J'instancie ensuite un nouvel objet XMLHttpRequest, puis j'ouvre une requête sur le lien qui enverra au controller que j'ai créé sur le serveur les datas au format JSON ensuite

selon la réponse que l'on va recevoir. Si l'on reçoit une erreur alors nous l'afficherons dans un message d'erreur et si tout se déroule comme prévu, sera alors afficher un message de confirmation d'inscription. Plus haut dans la template, un autre message aussi s'affiche en cas de succès et prévient l'utilisateur qu'il vient de recevoir un E-mail pour confirmer son compte. Par suite de cela un autre tableau associatif (user) est créé avec les données renvoyé par le serveur, nous enverrons ensuite ses données transformées au format JSON dans le localStorage, ce qui nous permettra de mettre en place une session pour l'utilisateur grâce à la mixin créé précédemment.

Rendons-nous donc dans la mixin sur le fichier main.js ou nous allons y créer une méthode ainsi qu'ajouter plusieurs valeurs à ses datas

```
36 app.mixin({
37   data(){
38     return{
39       tabsColor: 'primary',
40       toolBarColor: 'primary',
41       id: 0,
42       nom: "",
43       prenom: "",
44       avatar: "",
45       convsIds: [],
46       convCounter: 0,
47       newArticleCounter: 0
48     };
49   },
50   methods:{
51     user(){
52       if(localStorage.getItem("User") != null){
53         const user = JSON.parse(localStorage.getItem("User"));
54         this.id = user.id;
55         this.nom = user.nom;
56         this.prenom = user.prenom;
57         if(user.avatar != ''){
58           this.avatar = user.avatar;
59         }else{
60           this.avatar = 'default.png';
61         }
62       }
63     }
64   },
```

Cette méthode user() (qui sera exécutable partout dans l'application je le répète), vérifiera si la valeur User existe dans le localStorage, si oui elle récupérera donc ses valeurs et les placera dans des valeurs de la mixin. Ainsi nous pourrions récupérer les informations de l'utilisateur connecté sans avoir à refaire des requêtes, cela nous permet aussi de limiter certaine fonctionnalité seulement à l'utilisateur connecté, ainsi que comment mentionné plus haut. Elle sert aussi à ne pas afficher le menu si l'utilisateur n'est pas connecté.

Nous pouvons maintenant nous rendre du côté du serveur et créer un controller qui s'occupera lui de récupérer les datas, encryptera le mot de passe, créera un token propre à l'utilisateur, créera un nouvel user en base de données et exécutera une fonction qui lui enverra un E-mail de confirmation.

Nous allons commencer par ajouter un dossier api au sein du dossier Controller, puis nous lui créerons un fichier CreateUser.php. Suite à cela, rendez-vous dans l'Entité User pour relier notre custom controller avec api platform en rajoutant ceci dans l'annotation @ApiResponse()

```
38      *      collectionOperations={
39      *          "get"={},
40      *          "post"={},
41      *          "create_user"={
42      *              "method"="POST",
43      *              "path"="/users/create",
44      *              "controller"=App\Controller\Api\CreateUser::class
45      *          }
46      *      }
```

Cette nouvelle opération (create\_user) utilisera une méthode POST, lui donnera un chemin, et lui indiquera le controller que l'on vient de créer. Nous allons maintenant créer une Class User dans un dossier que nous allons créer aussi, src/manager.

Cette class importera donc d'autres méthodes, d'autres Class et les utilisera dans son constructeur. Nous pouvons ensuite créer la méthode registerAccount() que voici

```
67      public function registerAccount(User $user)
68      {
69          if ($this->userRepository->findOneBy(['email' => $user->getEmail()])) {
70              throw new BadRequestHttpException('Cette adresse email est déjà utilisé.');
```

```
71          }
72
73
74          $pass = $this->passwordEncoder->encodePassword($user, $user->getPassword());
75          $user->setPassword($pass);
76          $user->setDateInscription(new \DateTime());
77          $user->setDateLastCo(new \DateTime());
78          $roles[] = 'ROLE_USER';
79          $user->setRoles($roles);
80          $user->setAvatar("default.png");
81          $token = $this->generateToken();
82          $user->setToken($token);
83
84          $this->em->persist($user);
85          $this->em->flush();
86
87          $userSub = $this->em->getRepository(User::class)->findOneBy(['email' => $user->getEmail()]);
88          $this->InscriptionConfirmMail->notify($user->getEmail(), $token, $userSub);
89
90          $notif = new Notification();
91          $notif->setUser($user);
92          $notif->setType(0);
93          $notif->setTitle("Confirmation D'inscription");
94          $notif->setContent('Pour pouvoir poster des annonces et profiter pleinement de ton compte, confirme-le avec le mail q
95          $notif->setDateAjout(new \DateTime());
96          $this->em->persist($notif);
97          $this->em->flush();
98
99          return [
100              'message' => 'Création de compte enregistrée.',
101              'user' => $user,
102              'notif' => $notif
103          ];
104      }
```

Cette méthode dans un premier temps fait donc appel au repository user et utilise sa méthode find() afin de vérifier si l'email est déjà utilisé ou pas, si tout est ok, nous récupérerons ensuite le mot de passe pour l'encoder, nous initialisons le valeur DateInscription et DateLastCo à la date de la requête, nous lui injectons un rôle user, un avatar par défaut puis générons un token unique à l'utilisateur. Une fois cela fait l'utilisateur peut être enfin enregistré, une fois l'enregistrement de l'utilisateur effectué, nous pouvons aller le chercher dans le repository, puis faire appel à une méthode qui lui enverra un mail de confirmation, puis plus loin à une autre méthode qui elle lui ajoutera une notification et la persistera. Pour finir cette méthode retourne un message de succès, les données de l'utilisateur, ainsi que la notification.

Une fois cette méthode prête à l'emploi, nous pouvons retourner dans la Class CreateUser.php créée précédemment, et ajouter cette méthode à son constructeur puis invoquer les datas que l'on va recevoir via l'api et utiliser cette méthode en lui passant comme paramètre les datas invoquer.

```
6 use App\Manager\UserManager;
7 use Doctrine\ORM\EntityManagerInterface;
8
9 class CreateUser
10 {
11     /**
12      * @var UserManager
13      */
14     protected $userManager;
15
16     /**
17      * CreateUser constructor.
18      * @param UserManager $userManager
19      */
20     public function __construct(UserManager $userManager)
21     {
22         $this->userManager = $userManager;
23     }
24
25     /**
26      * @param $data
27      * @return mixed
28      * @throws \Exception
29      */
30     public function __invoke($data)
31     {
32         return $this->userManager->registerAccount($data);
33     }
34 }
```

Une fois l'inscription faite, nous pouvons mettre en place la connexion. Sur l'application j'effectue une requête exactement de la même façon que pour l'inscription, nous n'allons donc pas nous attarder dessus. Passons donc directement sur le serveur, ou pour plus de sécurité j'y ai intégré lexik/jwt-authentication-bundles qui permet d'effectuer une connexion sécurisée par token. Installons donc ce bundle en entrant dans notre projet la commande suivante

```
composer require lexik/jwt-authentication-bundle
```

Nous allons ensuite devoir générer deux clés, une public ainsi qu'une privée, donc lexik a besoin pour fonctionner en toute sécurité. Nous allons les générer grâce à openssl (téléchargeable depuis le site officiel si vous ne l'avez pas sur votre ordinateur), nous avons ensuite besoin de créer un dossier jwt dans le dossier config, puis nous pouvons maintenant entrer les commandes suivantes.

```
openssl genpkey -out config/jwt/private.pem -aes256 -algorithm rsa -pkeyopt  
rsa_keygen_bits:4096
```

```
openssl pkey -in config/jwt/private.pem -out config/jwt/public.pem -pubout
```

Direction ensuite le fichier security.yaml ou nous allons devoir créer un firewall pour la connexion avec lexik

```
18     api:  
19         pattern:    ^/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb  
20         provider:   app_user_provider  
21         stateless:  true  
22         anonymous:   true  
23         guard:  
24             authenticators:  
25                 - lexik_jwt_authentication.jwt_token_authenticator  
26  
27     main:  
28         anonymous:   true  
29  
30         json_login:  
31             check_path: /login  
32             username_path: email  
33             password_path: password  
34             success_handler: lexik_jwt_authentication.handler.authentication_success  
35             failure_handler: lexik_jwt_authentication.handler.authentication_failure  
36
```

Nous pouvons donc voir ci-dessus le firewall api que j'ai ajouté et qui utilise comme authenticators lexik, j'ai également modifié le firewall main pour lui ajouter json\_login qui lorsqu'un utilisateur cherchera à se connecter depuis la route login exécutera un event listener que j'ai mis en place et qui retournera les données de l'utilisateur si l'authentification est un succès. Puis après comme mentionné précédemment je traite les données reçues de la même façon que pour l'inscription.

La connexion et l'inscription étant mis en place et opérationnelle, nous pouvons dès à présent nous occuper de la déconnexion, qui est relativement simple, Il me suffit juste de vider l'objet User du localStorage avec la méthode localStorage.removeItem() , puis d'actualiser l'application et le tour est joué.

Cette partie étant fini, nous allons maintenant pouvoir mettre en place certaine fonctionnalité relatif à l'utilisateur. Nous allons commencer par créer une page de profil. Il y aura deux types de page de profil, une page que les autres utilisateurs peuvent consulter, qui recensent les informations que l'utilisateur à renseigner, sa photo de profil ainsi qu'un bouton contacter qui ouvre une conversation dans le chat dont je vous parlerais plus tard. La seconde est la page de profile de l'utilisateur connecté, il a la possibilité de prendre un selfie, j'utilise une fonctionnalité native du téléphone grâce à Capacitor puis j'envoie la photo en base64 au serveur Symfony ou j'ai créé un controller qui la réceptionne, qui la normalise, la renomme et la redirige dans le dossier voulu. Ce controller me renvoie ensuite le nouveau nom de la photo de l'utilisateur que je push ensuite dans le localStorage.

J'importe le plugin camera de Capacitor comme ceci

```
93 import { Plugins, CameraResultType, CameraSource } from '@capacitor/core';
94 const { Camera } = Plugins;
```

Et je m'en sers dans cette méthode

```
152 async takePicture(){
153     const photo = await Camera.getPhoto({
154         resultType: CameraResultType.Base64,
155         source: CameraSource.Camera,
156         quality: 60,
157     });
158     this.newPhoto = photo.base64String
159     setTimeout(async() => {
160         let data = {
161             avatar: this.newPhoto
162         };
163         var dataPut = JSON.stringify(data);
164         let xmlhttp = new XMLHttpRequest()
165         xmlhttp.open("PATCH", "https://www.studentcorner.fr/api/d88DhaVji9GJA6n4dBcW6P78Z6m4Pb/users/update/"+this.id,
166         xmlhttp.onload = () =>{
167             if(xmlhttp.status == 200){
168                 const res = JSON.parse(xmlhttp.response);
169                 var dataUser = res['hydra:member'][2];
170                 const user = {
171                     id: dataUser.id,
172                     nom: dataUser.Nom,
173                     prenom: dataUser.Prenom,
174                     avatar: dataUser.avatar,
175                     roles: dataUser.roles
176                 };
177                 localStorage.setItem("User", JSON.stringify(user));
178                 this.user();
179                 setTimeout(function(){window.location.reload();}, 100);
180             }else{
181                 this.error = "Erreur"
182                 console.log("erreur")
183                 this.loading = false;
184             }
185         }
186         xmlhttp.setRequestHeader("Content-Type", "application/merge-patch+json");
187         xmlhttp.send(dataPut)
188     })
189 }
```

Dans cette méthode, nous pouvons voir que nous commençons dans un premier temps par initialiser une constante photo qui prendra un opérateur await, c'est-à-dire que la méthode (asynchrone) devra attendre la résolution de cette promesse avant de pouvoir continuer son exécution. Cette constante photo utilise donc le plugin importer plus haut, elle lui demande d'avoir pour source la caméra du téléphone, d'avoir une qualité de 60 ( sur 100) pour optimiser le poids, et de nous donner un résultat en base64.

Une fois la photo prise par l'utilisateur, elle vient donc se placer dans une variable `newPhoto`, pour la suite nous envoyons une requête avec la méthode PATCH pour mettre à jour uniquement l'avatar de l'utilisateur sur le serveur, puis avec la réponse nous mettons à jour l'item `User` dans le `localStorage`.

Maintenant coté serveur nous n'avons plus qu'à réceptionner l'image et la traiter, j'ai donc rajouté une fonction `updateAvatar()` dans le `userManager`

```
110 | public function updateAvatar(User $user){
111 |     $image = $user->getAvatar();
112 |     $ext='jpg';
113 |     $fileName= md5(uniqid()) . '.' . $ext;
114 |
115 |     $file=fopen('avatar/'.$fileName, 'wb');
116 |     fwrite($file, base64_decode($image));
117 |     fclose($file);
118 |
119 |     $user->setAvatar($fileName);
120 |
121 |     $this->em->persist($user);
122 |     $this->em->flush();
123 |
124 |     return [
125 |         'message' => 'Profil modifié avec succès',
126 |         'file' => $fileName,
127 |         'user' => $user
128 |     ];
129 | }
```

Dans cette méthode dans un premier temps nous récupérons l'image reçue, lui donnons une extension, un nom unique en hachant en md5 un `uniqid()` qui génère un identifiant unique basé sur la date et l'heure courante en microsecondes. Nous ouvrons un fichier dans le dossier `avatar` qui aura comme nom, le nom donné à l'image précédemment, nous lui précisons que nous voulons l'ouvrir en écriture binaire (`wb`), puis nous pouvons écrire notre image décodée à l'intérieur, il ne nous reste plus qu'à fermer le fichier. Nous pouvons ensuite persister l'utilisateur avec son nouvel avatar. La méthode retourne ensuite un message de confirmation, le nom du nouvel avatar, ainsi que l'utilisateur.

Retournons maintenant sur l'application et sur la page de profil. Sur cette page l'utilisateur a également accès à un bouton qui redirige vers une page où l'utilisateur peut modifier ses informations, pour ce faire j'utilise la méthode PUT de l'api que j'ai générée, que je ne vous présenterais pas car très similaire à la méthode POST de l'inscription.

Le menu de l'application comporte aussi plusieurs pages liées à l'utilisateur et à ses interactions avec les annonces.

#### **La page mes annonces :**

Cette page contient toutes les annonces que l'utilisateur a posté, il a la possibilité de les supprimer pour ce faire j'ai posé sur chaque annonce de cette page une petite croix, aux clics de celle-ci, une confirmation apparaît, si la confirmation est acceptée alors une méthode DELETE s'envoie via Api et supprime l'annonce, puis la page se vide et se remplit à nouveau avec les bonnes annonces. Dans le détail de chaque annonce, nous pouvons aussi voir chaque utilisateur qui a postulé à cette annonce et les acceptés ou les refusés, si tel est le cas alors une méthode PATCH entre en jeu et envoie avec l'api le changement qu'elle doit effectuer dans cette annonce.

#### **La page Mes annonces favorites :**

J'ai intégré à l'application une autre fonctionnalité qui est aussi présente sur le site, celle de mettre en favoris des annonces, pour ce faire dans la page annonces/detailAnnonces, l'utilisateur a la possibilité de cliquer sur une étoile qui enverra une pré requête au serveur, qui lui demandera si l'utilisateur a déjà ajouté cette annonce à ses favoris sur le site, si oui je push l'annonce (son id) dans le local Storage ainsi l'étoile devient jaune, si non j'envoie une requête au serveur pour lui demander d'ajouter cette annonce à ses favoris et je rajoute également l'annonce dans le localStorage. Pour en revenir à la page mes annonces favorites, cette page contient donc toutes les annonces favorites de l'utilisateur avec la possibilité comme sur la page mes annonces, de supprimer celle-ci, à savoir que l'utilisateur peut aussi la supprimer depuis la page annonces/detailAnnonces en cliquant sur l'étoile jaune, dans tous les cas quand l'annonce est supprimée, le localStorage s'actualise, l'étoile n'est plus jaune et le serveur supprime l'annonce dans les favoris de l'utilisateur.

#### **La page Mes candidatures :**

Cette page contient trois boutons, un bouton pour chaque statut de candidatures que l'utilisateur a effectuées. Il pourra ainsi voir les candidatures qui ont été acceptées, celle qui ont été refusées, et celle qui sont toujours en attente.

#### **La page Poster une annonce :**

L'utilisateur connecté a la possibilité de poster une annonce, sur cette page l'utilisateur a deux choix demander un objet ou un service, au clic de celui-ci une requête s'effectue et va chercher les catégories du choix précédent, et les boucles sous forme de SELECT, au choix d'une catégorie comme précédemment une requête s'effectue et refait la même chose pour les sous-catégorie, une fois cela fait l'utilisateur peut renseigner le titre de son annonce, son contenu, son prix, son département, puis au clic du bouton de validation, une requête s'envoie en POST au serveur.



### Postuler à une annonce :

Si l'utilisateur est connecté, il aura la possibilité de postuler à une annonce dans le détail de celle-ci, il entre alors son message et celui-ci est envoyé en POST avec l'api au serveur, si l'utilisateur n'est pas connecté, dans le détail de l'annonce il aura alors à la place un bouton pour qui le redirige vers la page de connexion/inscription.

### Ergonomie :

De même que pour les boutons de la tabs de navigation en bas de l'application, j'ai répété le processus, ainsi quand l'utilisateur est sur l'une des routes du menu, son icône devient orange, il se décale vers la droite par rapport aux autres et une barre orange apparaît à gauche de l'écran sur le même axe x que l'icône et le titre de la page.

### Recherche par catégorie :

J'y ai rajouté aussi sur cette version un système de recherche par catégorie. Sur les pages blog et bonplans vous avez maintenant la possibilité de faire une recherche par catégorie, j'ai récupéré les catégories sur le serveur, et les ai bouclés sous forme de fab button, aux cliques d'une catégorie une requête est alors envoyée au serveur, sur un controller que j'ai créé qui utilise un query builder qui renvoie les article/bonplans de la catégorie demandée.

Ayant terminé la seconde version de l'application à la fin du premier mois, l'équipe de Student Corner a donc décidé de partir sur une troisième version, qui contiendrait un chat sur l'application ainsi que sur le site, mais avant toutes choses, il a été décidé de lancer une bêta test sur les stores de la version actuelle de l'application. Pour faire cela, il a fallu tout d'abord compiler un dossier Android ainsi qu'un dossier ios ouvrable par respectivement Android Studio et Xcode. Pour ce faire nous allons utiliser Capacitor (qui s'est installé dans notre projet en même temps qu'Ionic). Ouvrons d'abord un terminal dans notre projet puis entrons-y la commande

```
PS D:\student-corner\Appli\student-corner> ionic build
```

Cette commande créera un dossier dist et compilera à l'intérieur tout les assets Ionic. La seconde commande à exécuter est `npx cap add android` et `npx cap add ios` qui créera un dossier Android prêt à l'emploi avec toutes les dépendances dont il a besoin, et la même chose pour le dossier ios. La troisième commande à exécuter est (seulement lorsque nous feront ceci pour la deuxième fois ainsi que celle qui suivent, donc pour les mises à jour ect...) `npx cap sync` qui met à jour les dépendances ainsi que les web assets. Ayant déjà installer Android Studio sur ma machine au préalable, il ne me reste donc plus qu'à lancer la commande suivante pour ouvrir le projet dans cet IDE (environnement de développement intégré, comme par exemple VS Code)

```
npx cap open android
```

Une fois sur android studio, nous pouvons installer un SDK (Kit de développement) avec la version Android de notre choix, puis nous pouvons ensuite émuler notre application.

Une fois que l'on a vérifié que tout fonctionne bien, et éventuellement corrigé quelques bugs (ne nécessitant que peu de recherche), et aussi sans oublier d'ajouter le splash screen ainsi que les différents formats de logo, nous pouvons dès à présent build l'APK (Android Package) qui lui pourra être installé et faire fonctionner l'application sur les smartphones Android. Pour ce faire nous avons juste à générer une clé signée, qui contiendra les informations de l'entreprise ou du développeur, puis à cliquer sur le bouton générer une APK signée. Nous pouvons maintenant la déployer sur le Play store, après avoir rempli toutes les informations demandées à la création d'un projet, ainsi que les captures d'écran demandées pour mobile, moyenne et grande tablette. Une fois que tout est bon, nous pouvons maintenant déployer la bêta fermée. Ne disposant pas de Mac, et Xcode étant seulement utilisable sur un PC Apple, Mon maître de stage, a trouvé un Mac de son côté, et nous avons répété le même processus (à peu de chose près, Apple étant vraiment mais vraiment un tout petit peu plus carré) en partage d'écran. Nous avons eu des retours des testeurs tout le long du mois et ainsi corrigé plusieurs bugs (principalement d'affichage). Parallèlement à cela, j'ai commencé à développer le Chat.

Pour le Chat premièrement direction le serveur où j'y ai ajouté trois Entités : Conversation, Message, Participant, ayant pour relation celle que je vous ai décrite dans le C de la page 5. J'y ai aussi créé les contrôleurs grâce auquel je pourrais persister les données dans la base de données, ainsi que plusieurs query builders (j'en mettrais un ou deux dans l'annexe en fin de rapport, pour pas que ce rapport ne s'éternise trop).

De retour sur l'application, j'y ai créé une page chat/conversation.vue ainsi qu'une page message.vue dans le même dossier. Je me suis inspiré des systèmes de messagerie instantané, et des couleurs de Student Corner pour créer le design, puis j'y ai créé des méthodes pour appeler le contenu ainsi que pour envoyer des messages. Une fois cela fait, le chat est opérationnel, à la seule condition de reload la page afin de pouvoir voir les nouveaux messages, il faut donc rendre ce système plus dynamique, pour ce faire j'ai dû faire une recherche afin de trouver la meilleure façon d'aborder cette problématique. J'ai pour commencer essayé le protocole mercure, développer par la coopérative les Tilleuls, celle-là même qui a développé Api Platform. À la suite de gros problèmes au niveau du Cors (j'ai perdu une semaine à essayer de résoudre ce souci) j'ai décidé de ne plus perdre de temps et de partir sur une autre technologie, les Web Sockets. Pour ce faire, sur le serveur je me suis servi de Ratchet qui permet d'exécuter un script auquel on peut se connecter via un lien `ws://studentcorner.fr:8080` ce lien se connecte donc au serveur sur le port que l'on aura précisé au script Ratchet, le 8080 en l'occurrence ici.

Une fois cela terminé, nous pouvons donc nous connecter aisément à ce port, puis y écouter les messages qui y transit, ainsi qu'en envoyer. Il ne reste plus qu'à trier les messages que l'utilisateur reçoit, pour qu'il ne reçoit que les messages qui lui sont destinés en mettant une simple condition. Puis nous pouvons en même temps que d'envoyer un message à Symfony pour qu'il le persiste, envoyer le même message au port 8080.

Tout allait bien jusque-là, en locale cela fonctionné parfaitement, puis cela s'est un peu compliqué lors de la mise en ligne, Student Corner étant héberger sur un mutualisé, il était impossible d'ouvrir le port 8080 ni d'ailleurs d'en ouvrir aucun. J'ai donc dû trouver une autre solution. Puis suite à plusieurs recherches, je suis tombé sur le St-graal, le VPS, une machine virtuel distante tournant avec Ubuntu 16.04, sur ce VPS peut y être installer n'importe quoi (enfin par là je veux dire que vous avez les mêmes possibilités qu'en locale), donc aucuns soucis pour l'ouverture de n'importe quel port, faire tourner Node.js ect... J'ai ensuite suivi à un tutoriel pour installer un projet Symfony en SSH sur VPS. Grâce à cela j'ai appris à me servir des lignes de commande linux, ainsi j'ai configuré cette machine, j'y ai ajouté un nouvel utilisateur, installé PHP, Composer, Mysql, j'y ai importer le repository du site avec GIT, et généré un certificat SSL en configurant un fichier \*\*\*\*\*.conf . Une fois connecté sur le domaine du VPS, tout fonctionné enfin correctement en ligne. Après cela, j'ai pu régénéré une APK ainsi qu'une IPA et faire le nécessaire pour mettre à jour les release sur les stores

Une fois cela fait j'ai donc ensuite intégré le chat au site, pour ce faire j'ai utilisé le VPS pour faire tourner node.js sur Symfony avec Encore, qui nous servira à installer Webpack (Webpack est un modules bundle open source. Son objectif principal est de regrouper des fichiers JavaScript pour les utiliser dans un navigateur. Cet outil est également capable de transformer, regrouper ou empaqueter à peu près n'importe quelle ressource) et pourvoir générer le chat à l'aide de Vue.js, j'ai rencontré un petit souci car le VPS étant en https, il ne voulait pas m'accepter les lien ws:// j'ai dû trouver un moyen de les passer en wss:// en ajoutant ses lignes au fichier \*\*\*\*\*.conf

ProxyRequests Off

ProxyPass "/ws/" "ws://studentcorner.fr:8080/ "

Et en l'appelant comme ceci var ws = new WebSocket('wss://studentcorner.fr/ws')

## 6. Présentation du jeu d'essai élaboré par le candidat de la fonctionnalité la plus représentative (données en entrée, données attendues, données obtenues)

J'ai fait beaucoup de test sur l'application Student Corner, mais la plus grande partie des tests effectués ont été fait tout au long du développement de l'application. Pour l'appel de données par exemple, je bien vérifié que chaque article était présent, notamment avec la méthode mis en place pour que seul les articles où la date de publication était inférieure à la date d'aujourd'hui n'apparaisse. Il y a eu aussi beaucoup de tests de fait sur le chat, pour faire ceci j'ai créé deux comptes, et à chaque fois que je réaliser des améliorations, j'envoyais un message pour vérifier que le résultat attendu soit le bon. Nous avons également mis en place une bêta test fermée. Pour ce faire, nous avons lancé cette bêta test depuis les stores, d'où une dizaine de personnes ont pu tester l'application, suit à cela nous avons eu plusieurs retours que j'ai pu corriger par la suite avant d'envoyer l'application en production. Pour d'éventuel futurs erreurs, j'ai placé un bouton " Un problème ? " dans le menu de navigation de l'application, ou l'utilisateur qui rencontre un problème peut nous contacter soit par e-mail, ou soit directement sur le chat, ou un compte admin reçoit les messages.

Pour résumer les tests réalisés ont été faits tout au long du développement de l'application, à chaque fois que je développais une fonctionnalité, je vérifiais si les résultats étaient ceux attendu. Nous avons lancer une bêta test en avant-première pour avoir les retours de plusieurs utilisateurs. Un bouton pour nous contacter en cas de problème une fois l'application mise en production à était intégré.

## 7. Description de la veille, effectuée par le candidat durant le projet, sur les vulnérabilités de sécurité

Pour cette veille, j'ai choisi de la réaliser sur node.js et ses vulnérabilités. Je vais donc vous exposer ici ce que j'ai trouvé à ce propos ainsi que d'une façon de trouver ses failles.

Lorsque quelque chose devient populaire dans les technologies, ils sont exposés à des millions de professionnels, y compris des experts en sécurité, attaquants, hackers, etc. Pour offrir plus de sécurité à nos applications, nous pouvons utiliser Snyk, qui est l'une plates-formes d'analyse de sécurité les plus populaires de node.js. Pour avoir une idée, **83%** des utilisateurs de Snyk ont trouvé une ou plusieurs vulnérabilités dans leur application.

Qu'est-ce que Snyk ? Comme mentionné précédemment Snyk est une plate-forme d'analyse de sécurité pour node.js. Vous avez la possibilité de l'intégrer à GitHub, Jenkins, Circle CI, Travis, Code Ship, Bamboo pour trouver et corriger les vulnérabilités connues.

À un niveau élevé, Snyk offre une protection de sécurité complète, y compris les éléments suivants.

- Trouver des vulnérabilités dans le code
- Surveillez le code en temps réel
- Corrigez les dépendances vulnérables
- Soyez averti lorsqu'une nouvelle faiblesse affecte votre application.
- Collaborez avec les membres de votre équipe

Snyk maintient aussi sa propre base de données de vulnérabilités, et actuellement, il prend en charge Node.js, Ruby, Scala, Python, PHP, .NET, Go, etc.

Pour résumer afin de mieux sécuriser une application, nous pouvons utiliser des services tel que propose Snyk, une fois cela fait cela rajoute une touche de sécurité à notre application en nous offrant la possibilité de corriger quelques failles s'il y a.

## 8. Description d'une situation de travail ayant nécessité une recherche, effectuée par le candidat durant le projet, à partir de site anglophone

Lors de mon stage, chaque recherche que j'ai effectuée étant écrite en anglais, je suis donc naturellement tombé sur principalement que des sites anglophones. Peu de ressource n'étant accessible en français, il est donc préférable de parler anglais à Google, ainsi nous trouvons beaucoup de plus de ressource, et par conséquent la solution à notre problème, optimisant ainsi beaucoup (sur plusieurs recherche) le temps que nous mettons à développer notre projet. Pour cette description je vais donc vous raconter la première recherche que j'ai fait durant mon stage. Ayant préféré passer mon projet de TypeScript à Javascript, je me suis donc rendu sur le site officiel de Ionic Framework pour y rechercher la façon de procéder. Dans la partie suivante je vais donc vous présenter cet extrait de la documentation que j'ai trouvé et vous la traduire.



convert ionic project ts to js



[Tous](#) [Shopping](#) [Vidéos](#) [Actualités](#) [Images](#) [Plus](#) [Paramètres](#) [Outils](#)

Environ 1020 000 résultats (0,51 secondes)

<https://forum.ionicframework.com> > ... > Traduire cette page

### Help to convert ionic 2 JS project to TS one - ionic-v3 - Ionic ...

30 juin 2016 — Ionic 2 does not support JS any more so I'm trying to convert the app I'm working on in TS. But I have a lot of problems. Basicly my app show a ...

1 réponse · Meilleure réponse: Thank for your response , I have found an other way to do that

Create new app with JavaScript instead of TypeScript - Ionic Vue 8 déc. 2020

Typescript project files named js - ionic-v3 - Ionic Forum 13 juin 2016

Ionic React Without Typescript - ionic-v3 - Ionic Forum 6 nov. 2019

Problem files js/ts - ionic-v3 - Ionic Forum 23 juin 2016

[Autres résultats sur forum.ionicframework.com](#)

<https://ionicframework.com> > intro > Traduire cette page

### Migration - Ionic Framework

To convert this to controllerAs syntax, you only have to change a few things. ... you can easily convert it to a TypeScript class like this. `app.js .controller('MainCtrl' ...`

<https://ionicframework.com> > docs > Traduire cette page

### Ionic Vue Quickstart - Ionic Documentation - Ionic Framework

Write your app once using familiar technologies (HTML, CSS, JavaScript) and deploy to ... In a blank Ionic Vue app, this should only be router/index.ts and main.ts . ... top: 50%; transform: translateY(-50%); } #container strong { font-size: 20px; ...

Vous avez consulté cette page 5 fois. Dernière visite : 01/05/21

## 9. Extrait du site anglophone, utilisé dans le cadre de la recherche décrite précédemment, accompagné de la traduction en français effectuée par le candidat sans traducteur automatique (environ 750 signes).

Documentation d'Ionic :

<https://ionicframework.com/docs/vue/quickstart>

### English

#### **Build your way with TypeScript or JavaScript**

We love TypeScript at Ionic, and have believed for quite some time now that it's a great tool for building scalable apps. That said, we know how much the Vue community values simplicity – in their tooling, languages, and more. In fact, it's likely what drew you to Vue in the first place. Start simple – then scale up as needed.

So, if you'd prefer to use JavaScript instead of TypeScript, you can. After generating an Ionic Vue app, follow these steps :

1. Remove TypeScript dependencies :

```
npm uninstall --save typescript @types/jest @typescript-eslint/eslint-plugin @typescript-eslint/parser @vue/cli-plugin-typescript @vue/eslint-config-typescript
```

2. Change all .ts files to .js. In a blank Ionic Vue app, this should only be router/index.ts and main.ts.
3. Remove @vue/typescript/recommended and @typescript-eslint/no-explicit-any: 'off', from .eslintrc.js.
4. Remove Array<RouteRecordRaw> from router/index.js.
5. Delete the shims-vue.d.ts file.
6. Remove lang="ts" from the script tags in any of your Vue components that have them. In a blank Ionic Vue app, this should only be App.vue and views/Home.vue.

### **Créez votre chemin avec TypeScript ou JavaScript**

Nous adorons TypeScript chez Ionic et nous croyons depuis un certain temps déjà qu'il s'agit d'un excellent outil pour créer des applications évolutives. Cela dit, nous savons à quel point la communauté Vue valorise la simplicité – dans ses outils, ses langages et plus encore. En fait, c'est probablement ce qui vous a attiré vers Vue en premier lieu. Commencez simplement, puis augmentez au besoin.

Donc, si vous préférez utiliser JavaScript au lieu de TypeScript, vous pouvez le faire. Après avoir généré une application Ionic Vue, suivez ces étapes :

1. Supprimez les dépendances TypeScript :

```
npm uninstall --save typescript @types/jest @typescript-eslint/eslint-plugin @typescript-eslint/parser @vue/cli-plugin-typescript @vue/eslint-config-typescript
```

2. Remplacez tous les fichiers .ts par .js. Dans une application Ionic Vue vierge, il ne doit s'agir que de router/index.ts et main.ts.
3. Supprimez @vue/typescript/recommended et @typescript-eslint/no-explicit-any: 'off', de .eslintrc.js.
4. Supprimez Array<RouteRecordRaw> de router/index.js.
5. Supprimez le fichier shims-vue.d.ts.
6. Supprimez lang="ts" des balises de script dans n'importe lequel de vos composants Vue qui en ont. Dans une application vierge, cela ne devrait être que App.vue et views/Home.vue.



## 9. Annexe

### Exemples de Query Builder

```
90 public function findConversationsByUser(int $userId){
91     $qb = $this->createQueryBuilder('c');
92     $qb
93         ->select('otherUser.Prenom', 'otherUser.Nom', 'otherUser.avatar', 'c.id as conversationId', 'lm.content', 'lm.creat
94         ->innerJoin('c.participants', 'p', Join::WITH, $qb->expr()->neq('p.user', ':user'))
95         ->innerJoin('c.participants', 'me', Join::WITH, $qb->expr()->eq('me.user', ':user'))
96         ->leftJoin('c.lastMessage', 'lm')
97         ->innerJoin('lm.user', 'lmu')
98         ->innerJoin('me.user', 'meUser')
99         ->innerJoin('p.user', 'otherUser')
100         ->where('meUser.id = :user')
101         ->setParameter('user', $userId)
102         ->orderBy('lm.createdAt', 'DESC')
103     ;
104     return $qb->getQuery()->getResult();
105 }
```

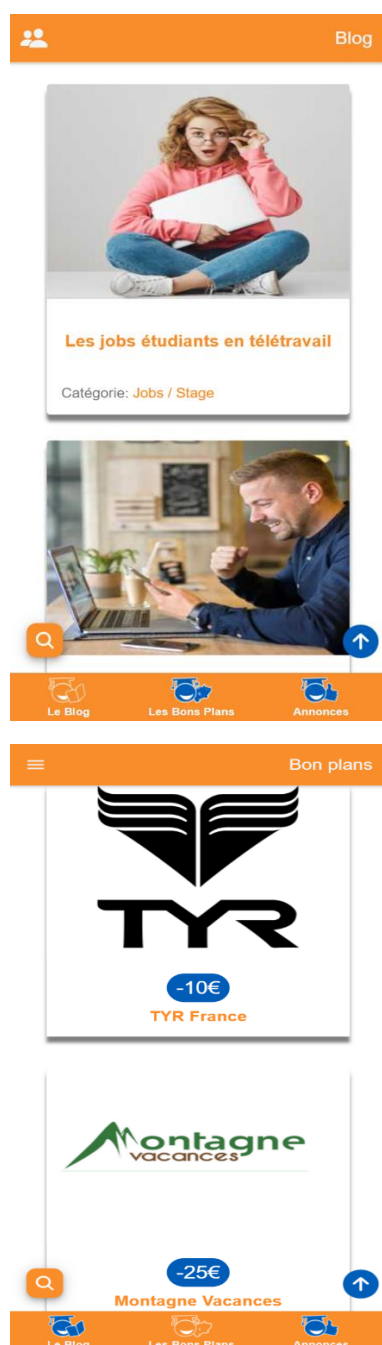
```
23 /**
24  * @return Conversation[] Returns an array of Conversation objects
25  */
26 public function findLastConversation(int $myId, int $otherUserId)
27 {
28     $qb = $this->createQueryBuilder('c');
29     $qb
30         ->innerJoin('c.participants', 'p')
31         ->where(
32             $qb->expr()->orX(
33                 $qb->expr()->eq('p.user', ':me'),
34                 $qb->expr()->eq('p.user', ':otherUser')
35             )
36         )
37         ->groupBy('p.conversation')
38         ->having(
39             $qb->expr()->eq(
40                 $qb->expr()->count('p.conversation'),
41                 2
42             )
43         )
44         ->setParameters([
45             'me' => $myId,
46             'otherUser' => $otherUserId
47         ])
48     ;
49     return $qb->getQuery()->getResult();
50 }
```


## Exemple de la mise en place des sockets sur l'application

```
760     data(){
761         return{
762             url: 'https://www.studentcorner.fr/avatar/',
763             senders: [],
764             conn: new WebSocket('ws://152.228.172.110:8080'),
765         }
766     }
767 });
```

```
608     connSocket(){
609         var self = this
610         this.conn.onopen = function(e) {
611             console.log("Connection established!" + e);
612         };
613         this.conn.onmessage = function(e) {
614             var res = JSON.parse(e.data);
615             var sender = res.senderId;
616             if(self.isNotInConfigChat()){
617                 if(self.id == res.destinataire){
618                     if(self.senders.indexOf(sender) == -1){
619                         self.senders.push(sender);
620                         var convId = res.conversation;
621                         var Nom = res.user.Nom;
622                         var Prenom = res.user.Prenom;
623                         var indexOfSender = self.senders.indexOf(sender)
624                         self.openToastOptions(convId, Nom, Prenom, indexOfSender);
625                     }else{
626                         console.log('notif deja presente')
627                     }
628                     if(self.convsIds.indexOf(sender) == -1){
629                         self.convsIds.push(sender);
630                         self.convCounter = self.convsIds.length
631                     }
632                 }
633             }else{
634                 if(self.convsIds.indexOf(sender) == -1){
635                     self.convsIds.push(sender);
636                     self.convCounter = self.convsIds.length
637                 }
638             }
639         }
640     }
641 }
642 },
```

## Vue sur l'application en mobile et tablette





Hello Loïc

Blog

+ Poster Une Annonce

Mon profil

Mes annonces

Mes annonces favorites

Mes candidatures


Annonce à venir

Chat


Se déconnecter

Un problème ?

Copyright © 2021 Student Corner.  
Tous droits réservés.




Blog




**Quels papiers administratifs garder ?**

Catégorie: Budget étudiant




**Comment trouver un logement étudiant ?**

Catégorie: étude




**Les sites internet pour trouver un job rapidement**


Catégorie: Budget étudiant




**Que faut-il savoir avant de signer un contrat de location ?**


Catégorie: étude









Le Blog



Les Bons Plans



Annonces



Hello Loïc

[+ Poster Une Annonce](#)

Mon profil

Mes annonces

Mes annonces favorites

Mes candidatures


Annonce à venir

1 Chat

Se déconnecter

Un problème ?


Blog



Posté le 15/04/2021

**Pourquoi envisager la freelance quand on est étudiant ?**

Catégorie: Jobs / Stage






Posté le 13/04/2021

**Comment travailler en freelance quand on est étudiant?**


Catégorie: Jobs / Stage

Copyright © 2021 Student Corner. Tous droits réservés.



Annonces




Antoine

Sur devis

Recherche GoPro ou caméra

Hello les students, je recherche une GoPro ou une caméra me permettant de filmer un petit trip de 3 jours Merci à vous

92200




Charles

5€ / fixe

Boitier de charge airpods APPLE

Hello les students, je recherche un boitier de charge pour airpods APPLE, j'en ai besoin que pendant 5 minutes afin d'associer mes nouveaux airpods, (j'ai un boitier qui n'est pas officiel APPLE et qui ne fait pas d'association)

92360




Louis

100€ / fixe

Création application native

Hello les students, je recherche quelqu'un qui maitrise des logiciels de création d'application comme goodbarber, happy pie etc !

92200



Edouard

20€ / fixe

Cherche Malette de poker, Paris 6.

Hello les students, je recherche une mallette de Poker complète le plus rapidement possible ! Covidement votre :)

75006

