

Documentation Technique, réalisé par Bastien Balmes



Description du projet :	2
Équipe :	2
Objectif :	2
Page de connexion :	2
Cas succès :	4
Cas échec :	4
CRUD Vacataire :	5
Créer vacataire :	5
Voir vacataire :	6
Modifier vacataire :	7
Supprimer vacataire :	8
CRUD module :	9
Créer module :	9
Voir module :	11
Modifier module :	12
Supprimer module :	13
Filtres :	13
Filtrage par affectation :	13
Filtrage par module :	14
Compte :	15
Mise en place du projet avec docker (Pour VS CODE) :	15
Pré requis :	15
Mise en place :	16

Description du projet :

Équipe :

- [Bastien BALMES](#) SCRUM master
- [Enzo Mancini](#) Product owner
- [Chrispher MARIE-ANGÉLIQUE](#) DEVDEV
- [Bryce FUERTES](#)
- [Alex JOLAS](#)
- [Marco VALLE](#)
- [Victor THOMPSON](#)
- [Michele FLORIO](#)

Objectif :

La gestion des vacataires à l'IUT actuelle laisse à désirer en termes d'efficacité et d'optimisation. Actuellement, elle repose sur des méthodes manuelles, chaque gestion de vacataire étant consignée sur des fiches en papier. Pour remédier à cette situation, Jean-Michel Bruel a entrepris le développement d'une application web dédiée à cette tâche. Cependant, en raison de contraintes de temps, il nous a confié ce projet, fournissant une maquette de base. Notre mission était donc de poursuivre le développement de cette application en apportant des améliorations substantielles.

Page de connexion :

Dans un premier temps on à un token interceptor, cela va permettre de vérifier si un utilisateur est authentifié ou pas.

- Si l'utilisateur n'est pas sur la page de connexion, il enregistre l'URL actuelle. Cela permet de rediriger l'utilisateur vers la page où il voulait aller après s'être connecté.
- Il vérifie s'il y a un token d'authentification stocké dans l'application.
- Si un token est trouvé, il l'ajoute automatiquement à toutes les demandes sortantes. Cela permet au serveur de savoir que l'utilisateur est authentifié.
- Si une demande échoue avec un code d'erreur 401 (non autorisé), cela signifie que le token est expiré ou invalide. Dans ce cas, il redirige l'utilisateur vers la page de connexion pour qu'il se reconnecte.

```
intercept(  
  req: HttpRequest<any>,  
  next: HttpHandler  
) : Observable<HttpEvent<any>> {  
  if (this.router.url !== "/connexion"){  
    // on enregistre la route actuelle  
    localStorage.setItem("currentRoute",this.router.url)  
  }  
  const token = localStorage.getItem("token")  
  
  if (token) {  
    //Ajout du token dans le header de la requête  
    req = req.clone({  
      setHeaders:{  
        Authorization : `Bearer ${token}`,  
      }  
    })  
  }  
  return next.handle(req).pipe(  
    catchError((err: HttpResponse) => {  
      if(err.status === 401){  
        //Si token expiré ou invalide on renvoie vers connexion  
        this.router.navigate(  
          ["/connexion"],  
          {  
            replaceUrl:true  
          }  
        )  
      }  
      return throwError(err);  
    })  
  )  
}
```

Voici le formulaire de connexion :

Identifiant

Mot de passe

Connexion


Lorsque l'utilisateur va renseigner son identifiant ainsi que son mot de passe, et qu'il va cliquer sur le bouton connexion, la méthode "connect" va être appelée.

- Lorsque l'utilisateur essaie de se connecter avec un nom d'utilisateur (pseudo) et un mot de passe (password), cette méthode appelle un service de login (loginService) pour effectuer une requête HTTP POST sur l'url api/connexion.
- Si la demande de connexion réussit (next), la méthode stocke le token d'authentification renvoyé dans le stockage local (localStorage) qui sera utilisé par le token interceptor. Ensuite, elle redirige l'utilisateur vers la dernière route sauvegardée localement, ce qui signifie qu'il est redirigé vers la page qu'il avait essayé d'accéder avant d'être redirigé vers la page de connexion.
- En cas d'erreur lors de la demande de connexion (error), le code affiche un message d'erreur dans la console (console.error) pour le débogage. Il va aussi afficher un message d'erreur "Identifiants invalides" pendant 3 secondes.

```
connect(pseudo: string, password: string) {
  this.loginService.login(pseudo,password).subscribe({
    next: (response) => {
      //On stock le token en local
      localStorage.setItem("token",(response as res).msg)
      //On redirige vers la dernière route sauvegardée en local
      this.router.navigate(
        [localStorage.getItem("currentRoute")],
        {
          replaceUrl:true
        })
    },
    error: (error) => {
      // Gestion des erreurs
      console.error(error);
      //On cherche l'id du composant d'alert
      const alert = document.getElementById('alertPass');

      if (alert != null ){
        //On affiche l'alert et on attend 3 secondes avant de la desactiver
        alert.style.display = "block"
        setTimeout(() => {
          alert.style.display = "none";
        },3000)
      }
    }
  })
}
```

Cas succès :

UNIVERSITÉ TOULOUSE
Jean Jaurès


VacatairesCoursProfil

Filtres


Rechercher par nom :

Ajouter un vacataire


- Vacataire affecté à un ou plusieurs modules.
- Vacataire désaffecté de tout ses modules.
- Vacataire affecté à aucun cours ou nouveau.




Enzo Mancini
Vacataire n°1
COM SAES



Bastine Balmes
Vacataire n°2
COM SAES



Enzo Moimême
Vacataire n°3



Enzo efef
Vacataire n°4

Cas échec :

Identifiant

Mot de passe

Identifiants invalides

Connexion

CRUD Vacataire :

Créer vacataire :

Nous accédons au formulaire de création d'un vacataire, avec le bouton "Ajouter un vacataire" présent dans la page Vacataires (celle qu'on accède juste après une connexion réussie)

UNIVERSITÉ TOULOUSE Jean Jaurès

Vacataires Cours Profil

Filtres Rechercher par nom : Ajouter un vacataire

- Vacataire affecté à un ou plusieurs modules.
- Vacataire désaffecté de tout ses modules.
- Vacataire affecté à aucun cours ou nouveau.

Avatar	Nom	Vacataire n°	Statut
	Enzo Mancini	Vacataire n°1	COM SAE5
	Bastine Balmes	Vacataire n°2	COM SAE5
	Enzo Moimême	Vacataire n°3	
	Enzo efef	Vacataire n°4	

Ajouter un vacataire :

Prénom :

Le prénom est requis !

Nom de famille :

Le nom de famille est requis !

Numéro de téléphone :

Le numéro de téléphone est requis !

Adresse email :

L'adresse email est requise !

URI du profil GitHub :

Sélectionnez une compétence

Ajouter Compétence

Annuler Ajouter

Pour créer un vacataire, l'utilisateur va devoir respecter les expressions régulières (REGEX):

- Le nom et le prénom ne peuvent pas contenir des chiffres et des caractères spéciaux sauf le " - " et " ' ".

```
<div class="form-floating mb-2">
  <input type="text" class="form-control" id="name" name="name" [(ngModel)]="vacataire.name" #name="ngModel" pattern="^[a-zA-ZÀ-ÿ\s'\-]+$"
  <label for="name">Prénom :</label>
  <div class="form_error" *ngIf="name.invalid && (name.touched || name.dirty)">
    <div *ngIf="name.errors && name.errors['required'];">Le prénom est requis !</div>
    <div *ngIf="name.errors && name.errors['pattern'];">Le prénom ne doit avoir que des lettres !</div>
  </div>
</div>
```

- Le téléphone doit être au format français (10 chiffres) et seulement des chiffres

```
<div class="form-floating mb-2">
  <input type="text" class="form-control" id="phone" name="phone" [(ngModel)]="vacataire.phone" #phone="ngModel" pattern="^(0[1-9])?(?:\d{2}){4}$"
  <label for="phone">Numéro de téléphone :</label>
  <div class="form_error" *ngIf="phone.invalid && (phone.touched || phone.dirty)">
    <div *ngIf="phone.errors && phone.errors['required'];">Le numéro de téléphone est requis !</div>
    <div *ngIf="phone.errors && phone.errors['pattern'];">Le numéro doit être au format FR (sans espace)!</div>
  </div>
</div>
```

- L'adresse mail elle doit contenir un "@" et un .com ou .fr par exemple

```
<div class="form-floating mb-2">
  <input type="email" class="form-control" id="email" name="email" [(ngModel)]="vacataire.email" #email="ngModel"
  pattern="^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+[a-zA-Z]{2,}$" required data-cy="email-input">
  <label for="email">Adresse email :</label>
  <div class="form_error" *ngIf="email.invalid && (email.touched || email.dirty)">
    <div *ngIf="email.errors && email.errors['required'];">L'adresse email est requise !</div>
    <div *ngIf="email.errors && email.errors['pattern'];">L'adresse email n'est pas valide !</div>
  </div>
</div>
```

- Le GitHub doit commencer par : "https://github.com/ + le user de la personne + /

```
<div class="form-floating mb-2">
  <input type="text" class="form-control" id="github" name="github" [(ngModel)]="vacataire.github" #github="ngModel"
  pattern=" /^(https?:\V\/)?(www\.)?github\.com\/[a-zA-Z0-9-]+\V\/?$/ " >
  <label for="github">URL du profil GitHub :</label>
  <div class="form_error" *ngIf="github.invalid && (github.touched || github.dirty)">
    <!-- <div *ngIf="github.errors && github.errors['required'];">Le profil github est requis !</div> -->
    <div *ngIf="github.errors && github.errors['pattern'];">L'URL du profil n'est pas bonne !</div>
  </div>
</div>
```

- Tous les champs avec "required" sont requis pour la création d'un vacataire

required

Une fois que tous les champs qui était requis sont remplis et qu'il respecte les REGEX le bouton Ajouter peut enfin être cliquable.

```
<button id="btn-ajout" type="button" class="btn btn-primary" (click)="addVacataire(vacataire.name, vacataire.lastName, vacataire.phone,
vacataire.email, vacataire.github, vacataire.skills)" data-cy="ajouter-input"
data-bs-dismiss="modal" [disabled]="name.errors || lastName.errors || phone.errors || email.errors || github.errors">Ajouter</button>
```

Le code addVacataire est appelé une fois qu'on appuie sur le bouton "Ajouter" dans le modal pour ajouter un nouveau vacataires :

- On a créé une méthode addVacataire qui prend diverses informations sur le vacataire (nom, prénom, téléphone, e-mail, GitHub, compétences) en tant que paramètres.
- On a effectué une vérification et un filtrage des compétences pour s'assurer qu'il n'y a pas d'éléments vides.
- Ensuite, on a appelé un service (vacatairesService) pour ajouter le vacataire en utilisant la méthode addVacataire et on a souscrit à l'Observable résultant pour gérer la réponse du serveur.
- En cas de succès (next), on a rechargé la page (probablement pour afficher les changements).
- En cas d'erreur (error), on a affiché l'erreur dans la console.

```
addVacataire(name: string, lastName: string, phone: string, email: string, github: string, skills: string[]) {  
  console.log(  
    "name : " + name + "\n" +  
    "lastName : " + lastName +  
    "Phone : " + phone + "\n" +  
    "email : " + email + "\n" +  
    "github : " + github + "\n" +  
    "skills : " + skills + "\n"  
  );  
  skills = skills.filter(el => el !== "")  
  this.vacatairesService.addVacataire(name, lastName, phone, email, github, skills).subscribe({  
    next: (response) => {  
      window.location.reload()  
    },  
    error: (error) => {  
      // Gestion des erreurs  
      console.error(error);  
    },  
    complete: () => {  
    }  
  })  
}
```

- On a créé une méthode addVacataire qui prend les mêmes informations que la méthode du composant.
- Cette méthode a construit l'URL de l'API en utilisant this.apiUrl et en ajoutant "/newVacataire".
- Ensuite, on a envoyé une requête POST vers cette URL avec les données du vacataire (nom, prénom, téléphone, e-mail, GitHub, compétences).


```
addVacataire(name: string, lastName: string, phone: string, email: string, github: string, skills: string[]): Observable<any> {  
  const url = this.apiUrl + "/newVacataire";  
  return this.http.post(url, {name, lastName, phone, email, github, skills});  
}
```


- On a un endpoint /newVacataire qui gère la demande POST pour ajouter un vacataire.
- Les données du vacataire sont extraites de la demande (nom, prénom, téléphone, e-mail, GitHub, compétences).
- On a créé un nouvel enregistrement de vacataire dans une base de données avec les données fournies.
- On a renvoyé une réponse avec le statut 200 (OK) et l'objet vacataire créé en tant que réponse JSON.

```
module.exports.addVacataire = async(req, res) => {
  // if(!req.body.name) {
  //   res.status(400).json({message: "Aucun message ! Ajoutez en un..."})
  // }

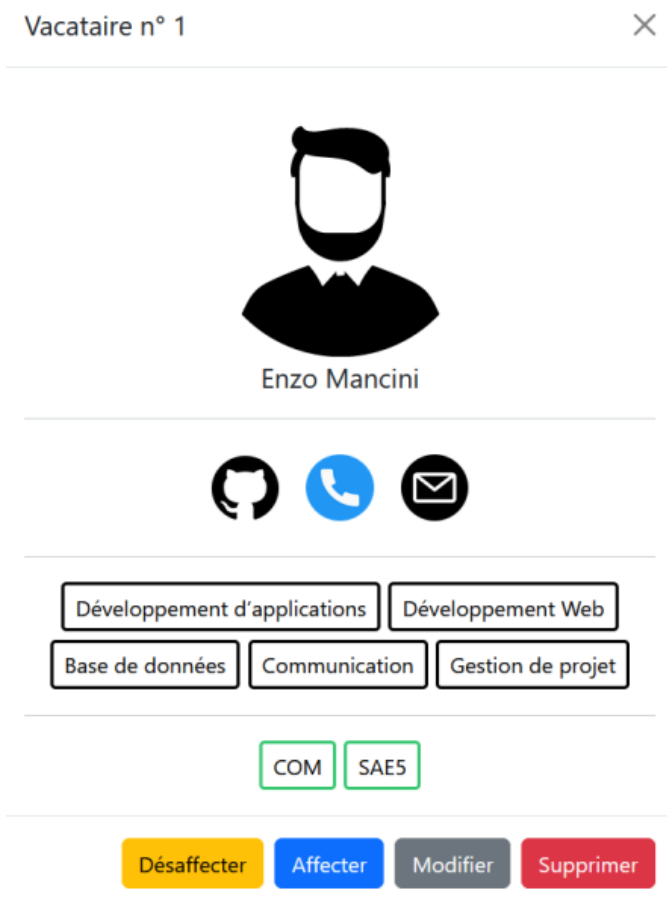
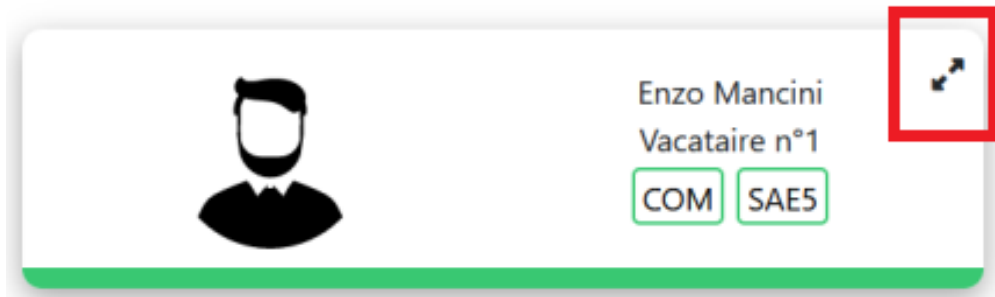
  const vacataire = await VacataireModel.create({
    name: req.body.name,
    lastName: req.body.lastName,
    phone: req.body.phone,
    email: req.body.email,
    github: req.body.github,
    skills: req.body.skills,
    modules: req.body.modules,
    status : "Non Affecté",
  })

  res.status(200).json(vacataire)
}
```

_id	name	lastName	phone	email	github	skills	modules	status	__v
 6539425631a86d979e608796	John	Balmes	0606060606	bastien@gmail.com				Non Affecté	0

Voir vacataire :

Une fois un vacataire créé on obtient une carte d'un vacataire, ou on voit quelque une des ces informations, en cliquant sur le bouton agrandir présent en haut à droite un modal s'ouvre avec toutes les informations du vacataire.



Code HTML body du modal :

```
<div class="modal-body">
  <div class="container-img">
    <div class="profile-pic-2" style="display: block;"><img src='.././../assets/img/user-avatar.svg' alt="profile-pic"></div>
  </div>
  <h5 class="modal-title" id="exampleModalLabel">{{ vacataire.name }} {{ vacataire.lastName }}</h5>
  <!-- Contenu du modal pour ce vacataire -->
  <hr>
  <div class="contact-logo">
    <a [hidden]="vacataire.github===''" target="_blank" href="{{vacataire.github}}"><img src='.././../assets/img/logo_github.png' alt="logo-github">
    <img style="cursor: pointer;" src='.././../assets/img/logo_tel.png' alt="logo-tel" (click)="logoTelClick(vacataire.phone)" title="Copie de la carte de contact">
    <img style="cursor: pointer;" src='.././../assets/img/logo_mail.png' alt="logo-mail" (click)="logoMailClick(vacataire.email)" title="Copie de la carte de contact">
  </div>
  <hr [hidden]="vacataire.skills.length===0">
  <div class="skills-containers">
    <ng-container *ngFor="let skill of vacataire.skills">
      <div *ngIf="skill!='" class="skill-show" >{{ skill }}
      <!-- <span class="delete-icon" (click)="removeSkill(skill)"><i class="fa-solid fa-xmark"></i></span> -->
      </div>
    </ng-container>
  </div>
  <hr [hidden]="vacataire.modules.length===0">
  <div class="skills-containers">
    <ng-container *ngFor="let module of vacataire.modules">
      <div *ngIf="module!='" class="module-show">{{module}}</div>
    </ng-container>
  </div>
  <!-- <p [hidden]="vacataire.github === '">GitHub : {{ vacataire.github }}</p> -->
  <!-- <p [hidden]="vacataire.skills.length === 0">Skills : {{ vacataire.skills }}</p> -->
  <!-- <p [hidden]="vacataire.modules.length === 0">Modules : {{ vacataire.modules }}</p> -->
  <!-- Ajoutez d'autres informations du vacataire si nécessaire -->
</div>
```


Code HTML Footer du modal :




```
<div class="modal-footer">
  <button *ngIf="vacataire.modules && vacataire.modules.length > 0" type="button" class="btn btn-warning"
    [attr.data-bs-target]="'#exampleModalToggle3-' + vacataire._id" data-bs-toggle="modal" data-bs-dismiss="modal">Désaffecter</button>
  <button type="button" class="btn btn-primary" [attr.data-bs-target]="'#exampleModalToggle2-' + vacataire._id"
    data-bs-toggle="modal" data-bs-dismiss="modal">Affecter</button>
  <button type="button" class="btn btn-secondary" [attr.data-bs-target]="'#editVacataireModal-' + vacataire._id"
    data-bs-toggle="modal" data-bs-dismiss="modal" (click)="initializeFormWithId(vacataire._id)">Modifier</button>
  <button type="button" class="btn btn-danger" (click)="deleteVacataire(vacataire._id)">Supprimer</button>
</div>
```

Modifier vacataire :

Pour modifier les informations d'un vacataire, ça se fait dans le modal où on voit toutes les informations d'un vacataire, on voit tout en bas du modal un bouton "Modifier". En cliquant sur ce bouton un nouveau modal apparaît, tous les champs du formulaire sont automatiquement pré-remplis avec les informations du vacataire. Pour modifier ces informations il suffit d'écrire dans le champ que nous souhaitons modifier, tout en respectant les "REGEX" (se sont les même que pour la création).

Vacataire n° 1 ×


Enzo Mancini

Développement d'applications

Développement Web

Base de données

Communication

Gestion de projet

COM

SAES

Désaffecter

Affecter

Modifier

Supprimer

Modification du vacataire ×

Les champs avec une * sont obligatoires.

Prénom* :

Bastine5

Le prénom ne doit avoir que des lettres !

Nom de famille* :

Balmes(

Le nom de famille ne doit avoir que des lettres !

Numéro de téléphone :

09089899995

Le numéro doit être au format FR (sans espace)!

Adresse email* :

bastin.balmes@gmail.com5

L'adresse email n'est pas valide !

URL du profil GitHub :

Sélectionnez une compétence

▼

Ajouter Compétence

- On a créé une méthode editVacataire qui prend un ID de vacataire et diverses informations de mise à jour (nom, prénom, téléphone, e-mail, GitHub, compétences) en tant que paramètres.
- La méthode envoie ces informations au service vacatairesService en appelant la méthode editVacataire du service, puis on a souscrit à l'Observable résultant pour gérer la réponse du serveur.
- En cas de succès (next), on a affiché la réponse dans la console.
- En cas d'erreur (error), on a affiché l'erreur dans la console.
- Après avoir terminé (complete), on recharge la page (window.location.reload())

```
editVacataire(id: String, name: string, lastName: string, phone: string, email: string, github: string, skills: string[]) {
    console.log(name);

    this.vacatairesService.editVacataire(id, name, lastName, phone, email, github, skills).subscribe({
        next: (response) => {
            // Traitement du succès
            console.log(response);
        },
        error: (error) => {
            // Gestion des erreurs
            console.error(error);
        },
        complete: () => {
            window.location.reload()
        }
    });
}
```

- On a créé une méthode editVacataire qui prend un ID de vacataire et les informations de mise à jour.
- Cette méthode a construit l'URL de l'API en utilisant this.apiUrl et l'ID du vacataire pour identifier la ressource spécifique à mettre à jour.
- On a créé un objet body contenant les données de mise à jour (nom, prénom, téléphone, e-mail, GitHub, compétences).
- Ensuite, on a envoyé une requête HTTP PUT à cette URL avec le corps de la requête contenant les données de mise à jour.

```
editVacataire(id: String, name: string, lastName: string, phone: string, email: string, github: string, skills: string[]): Observable<any> {
    const url = this.apiUrl + '/editVacataire/' + id;
    const body = {
        name: name,
        lastName: lastName,
        phone: phone,
        email: email,
        github: github,
        skills: skills
    }
    return this.http.put(url, body);
}
```

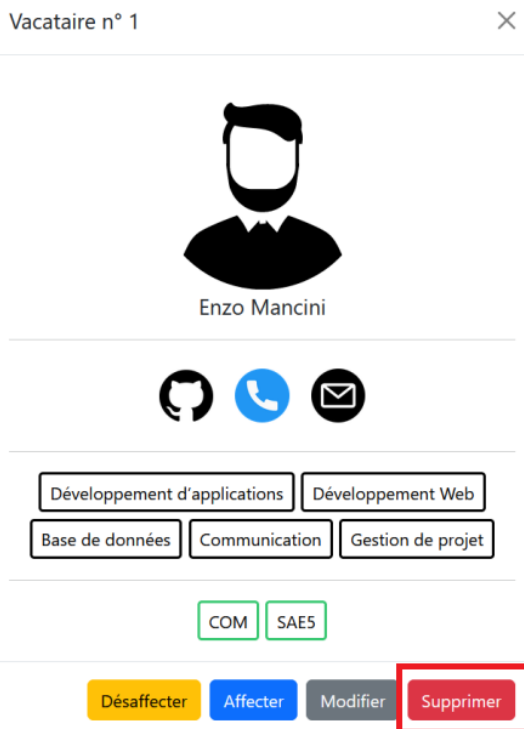
- On a un endpoint /editVacataire/:id qui gère la demande PUT pour mettre à jour un vacataire spécifique en fonction de son ID.
- L'API commence par chercher le vacataire dans la base de données en utilisant l'ID fourni dans la demande. Si le vacataire n'existe pas, on a renvoyé une réponse avec un statut 400 (Bad Request) et un message indiquant que le vacataire n'existe pas.

```
module.exports.editVacataire = async (req, res) => {
  const vacataire = await VacataireModel.findById(req.params.id)

  if(!vacataire) {
    res.status(400).json({
      message: "Ce vacataire n'existe pas"
    })
  }
}
```

Supprimer vacataire :

Pour supprimer les informations d'un vacataire, ça se fait dans le modal où on voit toutes les informations d'un vacataire, on voit tout en bas du modal un bouton "Supprimer". Et le vacataire est automatiquement supprimé du site.



- On a créé une méthode deleteVacataire qui prend l'ID du vacataire que l'on souhaite supprimer.
- La méthode appelle le service vacatairesService pour supprimer le vacataire en utilisant la méthode deleteVacataire. On a souscrit à l'Observable résultant pour gérer la réponse du serveur.
- En cas de succès (next), on a affiché la réponse dans la console.
- En cas d'erreur (error), on a affiché l'erreur dans la console.
- Après avoir terminé (complete), on recharge la page (window.location.reload())

```
deleteVacataire(id: string) {
  this.vacatairesService.deleteVacataire(id).subscribe({
    next: (response) => {
      // Traitement du succès
      console.log(response);
    },
    error: (error) => {
      // Gestion des erreurs
      console.error(error);
    },
    complete: () => {
      window.location.reload()
    }
  });
}
```

- On a créé une méthode editVacataire qui prend un ID de vacataire et les informations de mise à jour.
- Cette méthode a construit l'URL de l'API en utilisant this.apiUrl et l'ID du vacataire pour identifier la ressource spécifique à mettre à jour.
- On a créé un objet body contenant les données de mise à jour (nom, prénom, téléphone, e-mail, GitHub, compétences).
- Ensuite, on a envoyé une requête HTTP PUT à cette URL avec le corps de la requête contenant les données de mise à jour.
-

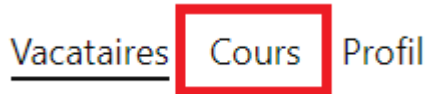
```
deleteVacataire(id: string): Observable<any> {
  const url = this.apiUrl + '/deleteVacataire/' + id;
  return this.http.delete(url);
}
```

- On a un endpoint /deleteVacataire/:id qui gère la demande DELETE pour supprimer un vacataire spécifique en fonction de son ID.
- L'API commence par chercher le vacataire dans la base de données en utilisant l'ID fourni dans la demande. Si le vacataire n'existe pas, on a renvoyé une réponse avec un statut 400 (Bad Request) et un message indiquant que le vacataire n'existe pas.
- Si le vacataire est trouvé, l'API utilise la méthode deleteOne pour supprimer le vacataire de la base de données en fonction de son ID.
- Enfin, on a renvoyé une réponse avec un status 200 (OK) et un message indiquant que le vacataire a été supprimé.

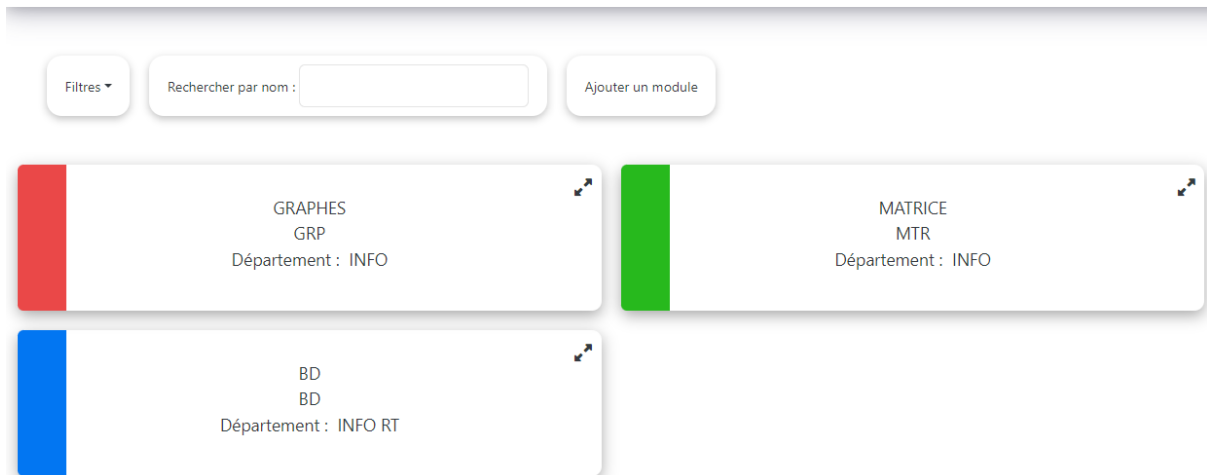
```
module.exports.deleteVacataire = async (req, res) => {  
  const vacataire = await VacataireModel.findById(req.params.id)  
  
  if(!vacataire) {  
    res.status(400).json({  
      message: "Ce vacataire n'existe pas"  
    })  
  }  
  
  await vacataire.deleteOne({ _id: req.params.id })  
  res.status(200).json("Message supprimé " + vacataire)  
}
```


CRUD module :

Pour s'occuper de la gestion des modules il faut changer de page et aller sur la page "Cours", on y accède en cliquant sur Cours dans la barre de navigation.



Un fois qu'on à cliquer sur "Cours" on arrive sur une page avec tous les modules créés.



Récupération des modules :

- On a un endpoint /getModules dans notre API Node.js géré par la fonction getModules.
- Cette fonction utilise le modèle ModuleModel pour récupérer les modules à partir de la base de données avec find().
- Une fois les modules récupérés, on renvoie la réponse au format JSON avec un statut 200 (OK).

```
module.exports.getModules = async (req, res) => {  
  const module = await ModuleModel.find()  
  res.status(200).json(module)  
}
```

Dans le service Angular, on a créé une méthode getModule() qui envoie une requête HTTP GET à l'URL de l'API (this.apiUrl) pour récupérer les modules.

```
getModule() {  
  return this.http.get(this.apiUrl)  
}
```

- Dans le composant “parent”, cours component, on utilise la méthode ngOnInit() pour initialiser la récupération de données depuis l'API.
- On appelle le service modulesService pour exécuter la méthode getModule().
- En utilisant .subscribe(), on attend la réponse de l'API.
- Lorsque la réponse est reçue, on stocke les modules dans la propriété this.modules du composant parent.

```
ngOnInit() {
  this.modulesService.getModule().subscribe((data: unknown) => {
    this.modules = data as Module[];

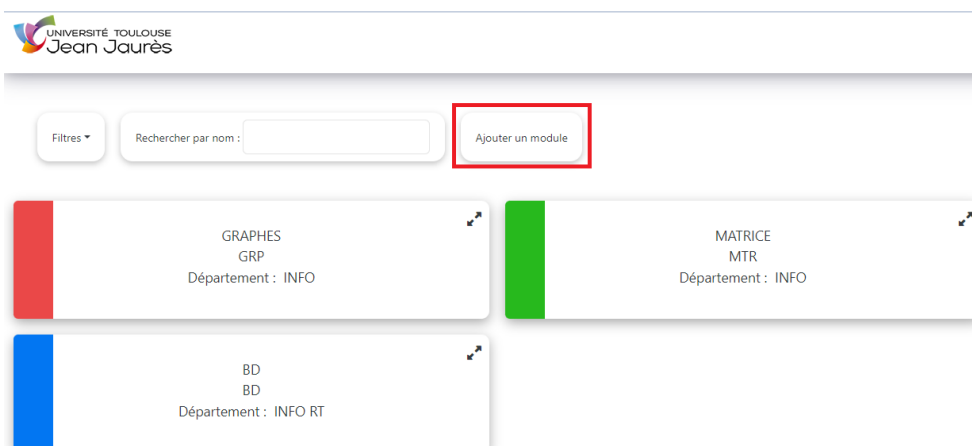
    // Récupération des matières et des départements distincts
    for (const c of this.modules) {
      if (!this.matières.includes(c.matiere)) {
        this.matières.push(c.matiere);
      }
      for (const departement of c.departement) {
        if (!this.departements.includes(departement.toUpperCase())) {
          this.departements.push(departement.toUpperCase());
        }
      }
    }
  });
}
```

- Dans le composant “enfant”, liste-cours component, on a une propriété @Input() modules: Module[] = []; qui nous permet de recevoir les données des modules du composant parent.
- On peut maintenant utiliser ces données dans le composant pour afficher les informations des modules.

```
@Input() modules: Module[] = [];
```

Créer module:

Nous accédons au formulaire de création d'un module, avec le bouton “Ajouter un module” présent dans la page Cour



Un fois qu'on à cliquer sur le bouton "Ajouter un module" un modal avec un formulaire de création de module apparaît, nous devons remplir ici tous les champs présents, et respecter certaines conditions :

- Le nom, le nom réduit ainsi que la matière ne peuvent pas contenir des chiffres et des caractères spéciaux sauf le " - " et " ' ".

```
<div style="text-align: center;">Les champs avec une * sont obligatoires.</div>
<div class="form-floating mb-3">
  <input type="text" class="form-control" name="name" id="name" [(ngModel)]="form.name" #name="ngModel" pattern="^[a-zA-ZÀ-ÿ\s'\-\d]+$" required>
  <label for="name">Nom* :</label>

  <div class="form_error" *ngIf="name.invalid && (name.touched || name.dirty)">
    <div *ngIf="name.errors && name.errors['required'];">Le nom est requis !</div>
    <div *ngIf="name.errors && name.errors['pattern'];">Le nom ne doit pas avoir de symbole (sauf -) !</div>
  </div>
</div>
```

- Tous les champs avec "required" sont requis pour la création d'un module

required

Une fois que tous les champs qui était requis sont remplis et qu'il respecte les REGEX le bouton Ajouter peut enfin être cliquable.

```
<button type="button" class="btn btn-primary" (click)='addModule(form.name, form.name_reduit,
form.color_hexa, form.departement, form.matiere)'
[disabled]="name.errors || name_reduit.errors || departement.errors || matiere.errors">Ajouter</button>
```

Le code addModule est appelé une fois qu'on appuie sur le bouton "Ajouter" dans le modal pour ajouter un nouveau vacataires :

- On a une méthode appelée addModule dans le composant TypeScript.
- On l'utilise pour appeler le service modulesService afin d'ajouter un module en utilisant la méthode addModule.
- On souscrit à l'Observable résultant pour gérer la réponse du serveur.
- En cas de succès (next), on recharge la page avec window.location.reload().
- En cas d'erreur (error), on affiche les erreurs dans la console.

```
addModule(name: string, name_reduit: string, color_hexa: string, departement: string[], matiere: string) {
  this.modulesService.addModule(name, name_reduit, color_hexa, departement, matiere).subscribe({
    next: (response) => {
      window.location.reload()
    },
    error: (error) => {
      // Gestion des erreurs
      console.error(error);
    },
    complete: () => {
    }
  });
}
```

- On a créé une méthode `addModule` dans le service Angular, qui prend les mêmes paramètres que la méthode du composant.
- Cette méthode construit l'URL de l'API en utilisant `this.apiUrl` et l'endpoint spécifique `/newModule`.
- Ensuite, on envoie une requête HTTP POST à cette URL avec un objet contenant les données du module, telles que le nom, le nom réduit, la couleur, le département et la matière.

```
addModule(name: string, name_reduit: string, color_hexa: string, departement: string[], matiere: string ): Observable<any> {
    const url = this.apiUrl + "/newModule";
    return this.http.post(url, {name, name_reduit, color_hexa, departement, matiere});
}
```

- Dans notre API Node.js, on a un endpoint `/newModule` qui est géré par la fonction `addModule`.
- Cette fonction commence par vérifier si le champ `name` est fourni dans la requête. Si ce n'est pas le cas, on renvoie une réponse avec un statut 400 (Bad Request) et un message d'erreur.
- Si le champ `name` est fourni, on crée un nouveau module en utilisant le modèle `ModuleModel` et les données fournies dans la requête.
- Enfin, on renvoie une réponse avec un statut 200 (OK) et le module créé au format JSON.

```
module.exports.addModule = async(req, res) => {
    if(!req.body.name) {
        res.status(400).json({message: "Aucun message ! Ajoutez en un..."})
    }

    const module = await ModuleModel.create({
        name: req.body.name,
        name_reduit: req.body.name_reduit,
        color_hexa: req.body.color_hexa,
        departement: req.body.departement,
        matiere: req.body.matiere,
    })
    res.status(200).json(module)
}
```

Ajouter un module :



Nom :
Dev JAVA

Nom réduit :
DEV

Départements :

INFO

RT

CS

Matière :
Développement

Couleur du module :

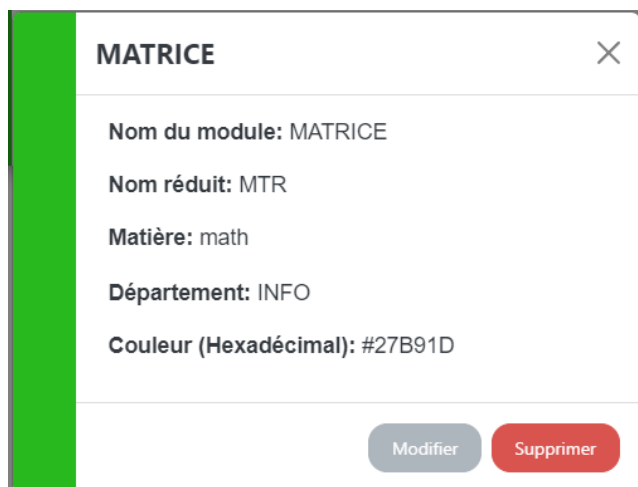


Annuler

Ajouter

Voir module :

Une fois un module créé on obtient une carte d'un module, ou on voit quelque une des ces informations, en cliquant sur le bouton agrandir présent en haut à droite un modal s'ouvre avec toutes les informations du module.



Code HTML du modal :

```
<div class="modal fade" [id]='exampleModal' + cours._id tabindex="-1" aria-labelledby="exampleModallabel" aria-hidden="true">
  <div class="modal-dialog modal-dialog-centered">
    <div class="modal-content">
      <div class="cours-color" [style.background-color]="cours.color_hexa"></div>
      <div class="modal-header" style="margin-left: 50px;">
        <h5 class="modal-title" id="exampleModallabel">{{ cours.name }}</h5>
        <button type="button" class="btn-close" data-bs-dismiss="modal" aria-label="Close"></button>
      </div>
      <div class="modal-body" style="text-align: left;margin-left: 60px;">
        <!-- Contenu du modal pour ce module -->
        <p style="font-family: Arial, Helvetica, sans-serif;"><strong>Nom du module:</strong> {{ cours.name }}</p>
        <p style="font-family: Arial, Helvetica, sans-serif;"><strong>Nom réduit:</strong> {{cours.name_reduit}}</p>
        <p style="font-family: Arial, Helvetica, sans-serif;"><strong>Matière:</strong> {{cours.matiere}}</p>
        <strong style="font-family: Arial, Helvetica, sans-serif;">Département:</strong>
        <div *ngFor="let departement of cours.departement" class="departement" style="display:inline-block;font-family: Arial, Helvetica, sans-serif;">
          <li style="text-transform: uppercase;">&nbsp;&nbsp;&nbsp;&{{departement}}</li>
        </div><p></p>
        <p><strong style="font-family: Arial, Helvetica, sans-serif;">Couleur (Hexadécimal):</strong> <font style="font-family: Arial, Helvetica, sans-serif;">{{cours.color_hexa}}</font></p>
        <!-- Ajoutez d'autres informations du module si nécessaire -->
      </div>
      <div class="modal-footer">
        <!-- <button type="button" class="btn btn-primary" [attr.data-bs-target]='"#exampleModalToggle2-' + cours.id"' data-bs-toggle="modal" data-bs-dismiss="modal">Ajouter</button> -->
        <button type="button" class="btn btn-secondary" data-bs-toggle="modal" [attr.data-bs-target]='"#updateModule'+cours._id" data-bs-dismiss="modal">Modifier</button>
        <button type="button" class="btn btn-danger" (click)="deleteModule(cours._id)">Supprimer</button>
      </div>
    </div>
  </div>
</div>
```

Modifier module :

Pour modifier les informations d'un module, ça se fait dans le modal où on voit toutes les informations d'un vacataire, on voit tout en bas du modal un bouton "Modifier". En cliquant sur ce bouton un nouveau modal apparaît, tous les champs du formulaire sont automatiquement pré-remplis avec les informations du vacataire. Pour modifier ces informations il suffit d'écrire dans le champ que nous souhaitons modifier, tout en respectant les "REGEX".

MATRICE

Nom du module: MATRICE

Nom réduit: MTR

Matière: math

Département: INFO

Couleur (Hexadécimal): #27B91D

Modifier

Supprimer

- On a une méthode `updateModule` dans le composant TypeScript.
- On l'utilise pour appeler le service `modulesService` afin de mettre à jour un module en utilisant la méthode `updateModule`.
- On souscrit à l'Observable résultant pour gérer la réponse du serveur.
- En cas de succès (`next`), on affiche la réponse dans la console.
- En cas d'erreur (`error`), on affiche les erreurs dans la console.
- Après avoir terminé (`complete`), on recharge la page avec `window.location.reload()` pour refléter les modifications.

```
updateModule(id: string, name: string, name_reduit: string, color_hexa: string, departement: string[], matiere: string ){
  this.modulesService.updateModule(id,name,name_reduit,color_hexa,departement,matiere).subscribe({
    next: (response) => {
      // Traitement du succès
      console.log(response);
    },
    error: (error) => {
      // Gestion des erreurs
      console.error(error);
    },
    complete: () => {
      window.location.reload()
    }
  });
}
```

- On a créé une méthode `updateModule` dans le service Angular, qui prend les mêmes paramètres que la méthode du composant.
- Cette méthode construit l'URL de l'API en utilisant `this.apiUrl` et l'endpoint `/editModule/id`, où `id` est l'identifiant unique du module que l'on souhaite modifier.
- Ensuite, on envoie une requête HTTP PUT à cette URL avec un objet contenant les données du module à mettre à jour.

```
updateModule(id: string, name: string, name_reduit: string, color_hexa: string, departement: string[], matiere: string ): Observable<any> {
  const url = this.apiUrl + '/editModule/' + id;
  return this.http.put<any>(url,{name, name_reduit, color_hexa, departement, matiere});
}
```

- Dans notre API Node.js, on a un endpoint /editModule/:id qui est géré par la fonction editModule.
- Cette fonction commence par chercher le module en utilisant l'ID fourni dans la requête en utilisant ModuleModel.findById().
- Si le module n'est pas trouvé, on renvoie une réponse avec un statut 400 (Bad Request) et un message d'erreur.
- Si le module est trouvé, on utilise ModuleModel.findByIdAndUpdate() pour mettre à jour le module avec les données fournies dans la requête. On utilise l'option {new: true} pour renvoyer le module mis à jour.
- Enfin, on renvoie une réponse avec un statut 200 (OK) et le module mis à jour au format JSON.

```
module.exports.editModule = async (req, res) => {
  const module = await ModuleModel.findById(req.params.id)

  if(!module) {
    res.status(400).json({
      message: "Ce module n'existe pas"
    })
  }

  const updateModule = await ModuleModel.findByIdAndUpdate(
    module,
    req.body,
    {new: true}
  )

  res.status(200).json(updateModule)
}
```

Modifier le module :



Nom :
MATRICE

Nom réduit :
PLOP

Départements :

INFO

RT

CS

Matière :
math

Couleur du module :



Annuler

Valider

Supprimer module :

Pour supprimer les informations d'un module, ça se fait dans le modal où on voit toutes les informations d'un module, on voit tout en bas du modal un bouton "Supprimer". Et le module est automatiquement supprimé du site.



- On a une méthode deleteModule dans le composant TypeScript.
- Cette méthode prend un seul paramètre, id, qui est l'identifiant du module que l'on souhaite supprimer.
- On l'utilise pour appeler le service modulesService afin de supprimer un module en utilisant la méthode deleteModule.
- On souscrit à l'Observable résultant pour gérer la réponse du serveur.
- En cas de succès (next), on affiche la réponse dans la console.
- En cas d'erreur (error), on affiche les erreurs dans la console.
- Après avoir terminé (complete), on recharge la page avec window.location.reload() pour refléter les modifications.

```
deleteModule(id: string) {  
  this.modulesService.deleteModule(id).subscribe({  
    next: (response) => {  
      // Traitement du succès  
      console.log(response);  
    },  
    error: (error) => {  
      // Gestion des erreurs  
      console.error(error);  
    },  
    complete: () => {  
      window.location.reload()  
    }  
  });  
}
```

- On a créé une méthode `deleteModule` dans le service Angular, qui prend l'id du module à supprimer comme paramètre.
- Cette méthode construit l'URL de l'API en utilisant `this.apiUrl` et l'endpoint `/deleteModule/id`, où `id` est l'identifiant unique du module à supprimer.
- Ensuite, on envoie une requête HTTP DELETE à cette URL.

```
deleteModule(id: string): Observable<any> {
  const url = this.apiUrl + '/deleteModule/' + id;
  return this.http.delete(url);
}
```

- Dans notre API Node.js, on a un endpoint `/deleteModule/:id` qui est géré par la fonction `deleteModule`.
- Cette fonction commence par chercher le module en utilisant l'ID fourni dans la requête en utilisant `ModuleModel.findById()`.
- Si le module n'est pas trouvé, on renvoie une réponse avec un statut 400 (Bad Request) et un message d'erreur.
- Si le module est trouvé, on utilise `module.deleteOne()` pour supprimer le module de la base de données en utilisant son ID.
- Enfin, on renvoie une réponse avec un statut 200 (OK) et un message indiquant que le module a été supprimé.

```
module.exports.deleteModule = async (req, res) => {
  const module = await ModuleModel.findById(req.params.id)

  if(!module) {
    res.status(400).json({
      message: "Ce module n'existe pas"
    })
  }

  await module.deleteOne({ _id: req.params.id })
  res.status(200).json("Message supprimé " + module)
}
```

Filtres :

Sur notre site il y a la possibilité de rechercher les vacataires et les modules en fonction de certain filtre

Filtre pour les vacataire :

cette ligne de code conditionne l'affichage d'un élément HTML avec la classe CSS "vacataire" en fonction du résultat de la fonction `isInFilter(vacataire)`. Si la fonction renvoie true, l'élément est affiché ; sinon, il est masqué. Cela permet de filtrer dynamiquement les éléments à afficher en fonction des critères de filtrage définis dans la fonction `isInFilter`.

```
<div class="vacataire" *ngIf="isInFilter(vacataire)">
```

- La fonction `isInFilter` prend un objet `vacataire` de type `Vacataire` en tant qu'argument.
- Au début de la fonction, une variable `state` est initialisée à `true`. Cela signifie que par défaut, le "vacataire" est considéré comme validé.
- Ensuite, la fonction parcourt chaque filtre contenu dans l'objet `filters` à l'aide d'une boucle `for...of`. Chaque filtre est une paire clé-valeur, où la clé (`filterName`) est le nom de la propriété du "vacataire" que l'on souhaite filtrer, et la valeur (`filterValue`) est la valeur que cette propriété doit satisfaire pour que le "vacataire" soit considéré comme validé.
- Si la clé du filtre est "search", la fonction effectue une recherche textuelle dans les propriétés `name` et `lastName` du "vacataire". Ces propriétés sont converties en majuscules, tout comme la valeur du filtre. Ensuite, la fonction vérifie si la valeur du filtre (en majuscules) est incluse dans l'une des deux propriétés (nom ou prénom). Si c'est le cas, la variable `found` devient `true`, indiquant que le filtre de recherche est satisfait. Finalement, l'état `state` est mis à la valeur de `found`.
- Si la clé du filtre n'est pas "search", la fonction récupère la propriété correspondante du "vacataire" en utilisant la clé du filtre comme clé d'accès. Si la propriété existe (property n'est pas nulle), la fonction vérifie si elle est de type tableau (`Array`). Si c'est le cas, la fonction parcourt le tableau pour vérifier si la valeur du filtre est incluse dans le tableau. Si la valeur du filtre n'est pas trouvée dans le tableau, l'état `state` est défini sur `false`.
- Si la propriété n'est pas un tableau, la fonction compare la valeur de la propriété (convertie en majuscules) à la valeur du filtre (également convertie en majuscules). Si elles ne correspondent pas, l'état `state` est mis à `false`.
- La fonction continue de parcourir tous les filtres, mettant à jour l'état `state` en fonction des critères de filtrage.
- En fin de compte, la fonction retourne la valeur de l'état `state`, qui indique si le "vacataire" satisfait tous les critères de filtrage. Si au moins un filtre n'est pas satisfait, la valeur `state` sera `false`, indiquant que le "vacataire" n'est pas validé selon les critères de filtrage.

```

isInFilter(vacataire: Vacataire) {
  let state = true; // Est validé par défaut

  for (const [filterName, filterValue] of Object.entries(this.filtres)) {
    if (filterName === 'search') {
      let found = false;

      // Propriétés de la recherche
      const name = vacataire.name.toUpperCase();
      const lastName = vacataire.lastName.toUpperCase();

      if ((name + ' ' + lastName).includes(filterValue.toUpperCase()) || (lastName + ' ' + name).includes(filterValue.toUpperCase())) {
        found = true;
      }

      state = found;
    } else {
      const property = vacataire?.[filterName as keyof Vacataire];

      if (property) {
        if (property instanceof Array) {
          if (property.findIndex(e1 => e1.toUpperCase() === filterValue.toUpperCase()) === -1) {
            state = false;
          }
        } else if (property.toUpperCase() !== filterValue?.toUpperCase()) {
          state = false;
        }
      }
    }
  }

  return state;
}

```

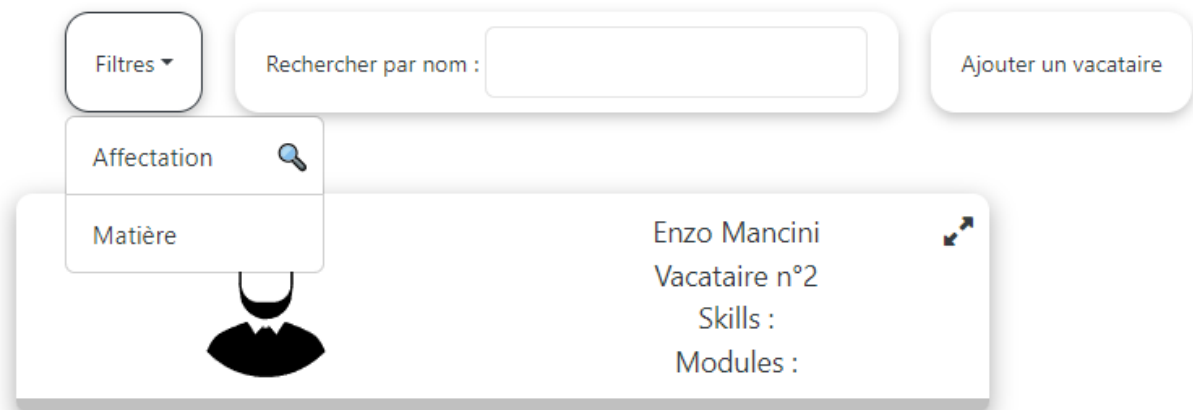
Filtrage par affectation :

Dans notre site un vacataire peut avoir plusieurs état d'affectation, "Affecté" sur on lui à affecter un module, "Non affecté" si il n'a aucun module et n'a jamais eue de module, et "En attente" si il à déjà eu un module mais actuellement n'a pas d'affectation.

Affecté :

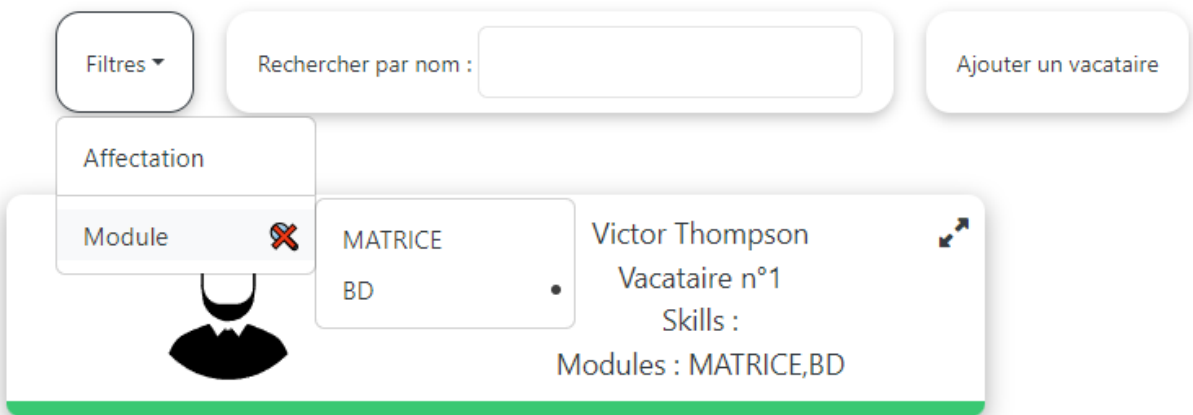
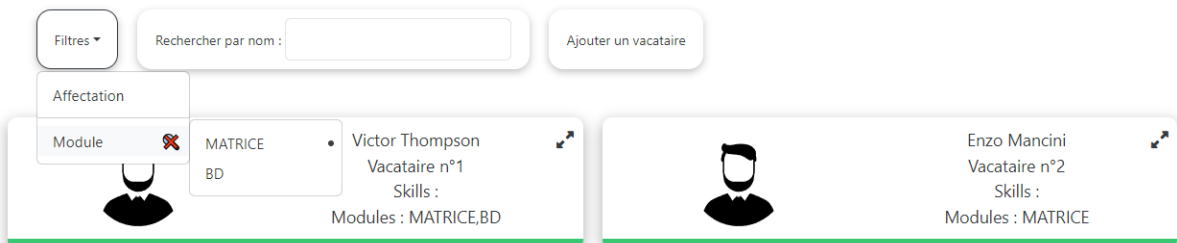
Non affecté :

En attente :



Filtrage par module :

Chaque vacataire sur notre site peut être affecté à un ou plusieurs modules, on peut donc rechercher tous les vacataires affectés à un certain module.



Cette ligne de code conditionne l'affichage d'un élément HTML avec la classe CSS `cours-card` en fonction du résultat de la fonction `isInFilter(cours)`. Si la fonction renvoie `true`, l'élément est affiché ; sinon, il est masqué. Cela permet de filtrer dynamiquement les éléments à afficher en fonction des critères de filtrage définis dans la fonction `isInFilter`.

Filtre pour les modules :

```
<div *ngIf="isInFilter(cours)" class="cours-card">
```

- La fonction `isInFilter` prend un objet `cours` de type `Module` comme argument.
- Au début de la fonction, une variable `state` est initialisée à `true`. Cette variable servira à suivre l'état de filtrage du cours.
- Ensuite, la fonction itère sur les paires clé-valeur dans l'objet `filtres`. Les filtres semblent être stockés dans cet objet. Chaque filtre est associé à un nom de propriété (clé) et à une valeur (valeur) que le cours doit satisfaire pour être considéré comme filtré.
- La première vérification se fait pour le filtre "search" (recherche). Si le filtre a pour nom "search", la fonction effectue une recherche dans les propriétés du cours (nom et nom réduit) pour voir si la valeur du filtre est présente dans l'une de ces propriétés. Si c'est le cas, la variable `found` devient `true`, sinon, elle reste à `false`. Finalement, l'état `state` est mis à la valeur de `found`.
- Si le filtre n'est pas "search", la fonction récupère la propriété du cours correspondant au nom du filtre (par exemple, si le filtre est "departement", la propriété correspondante serait `cours.departement`).
- Ensuite, la fonction vérifie si la propriété est définie (property n'est pas nulle). Si la propriété est un tableau (Array), elle parcourt le tableau pour vérifier si la valeur du filtre est présente. Si la valeur n'est pas trouvée dans le tableau, l'état `state` est mis à `false`.
- Si la propriété n'est pas un tableau, la fonction vérifie si la valeur de la propriété (convertie en majuscules) correspond à la valeur du filtre (également convertie en majuscules). Si elles ne correspondent pas, l'état `state` est mis à `false`.
- La fonction continue de parcourir tous les filtres et met à jour l'état `state` en conséquence.
- En fin de compte, la fonction retourne la valeur de l'état `state`, qui indique si le cours satisfait tous les filtres spécifiés.

```
isInFilter(cours: Module) {  
  let state = true; // Est valide par défaut  
  
  for (const [filterName, filterValue] of Object.entries(this.filtres)) {  
    if (filterName === 'search') { // Si barre de recherche  
      let found = false;  
  
      // Propriétés de la recherche  
      const name = cours.name.toUpperCase();  
      const nameReducit = cours.name_reduit.toUpperCase();  
  
      if (name.includes(filterValue.toUpperCase()) || nameReducit.includes(filterValue.toUpperCase())) {  
        found = true;  
      }  
  
      state = found;  
    } else {  
      const property = cours?.[filterName as keyof Module];  
  
      if (property) {  
        if (property instanceof Array) {  
          if (property.findIndex(el => el.toUpperCase() === filterValue.toUpperCase()) === -1) {  
            state = false;  
          }  
        } else if (property.toUpperCase() !== filterValue?.toUpperCase()) {  
          state = false;  
        }  
      }  
    }  
  }  
}
```

GRAPHES
GRP
Département : INFO RT

MATRICE
MTR
Département : INFO

DEV AVANCER
DEV
Département : CS

Filtre par matières :

Chaque module (Graphe,Matrice,Dev avancer) est associé à une matière (math,développement).

Filtres ▾

Rechercher par nom :

Ajouter un module

Matière

Département

math

Développement

GRAPHES
GRP
Département : INFO RT

MATRICE
MTR
Département : INFO

Filtres ▾

Rechercher par nom :

Ajouter un module

Matière

Département

math

Développement

DEV AVANCER
DEV
Département : CS

Filtre par département :

Chaque module (Graphe,Matrice,Dev avancer) est associé à un ou plusieurs département (INFO,RT)

Filtres ▾

Rechercher par nom :

Ajouter un module

Matière

Département

INFO

RT

CS

GRAPHES
GRP
Département : INFO RT

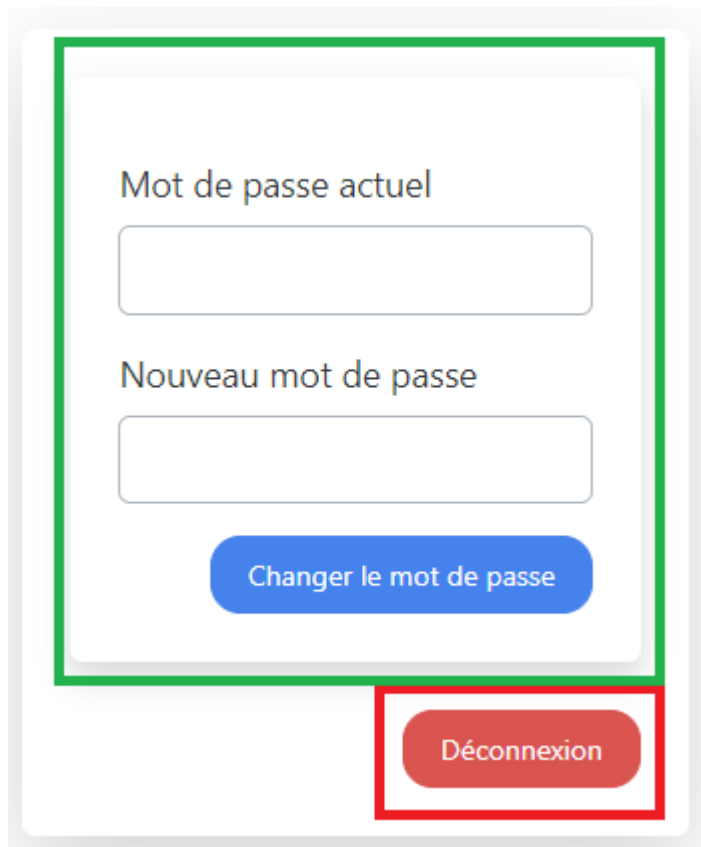
MATRICE
MTR
Département : INFO

Compte :

Pour son compte il faut changer de page et aller sur la page "Profil", on y accède en cliquant sur Profil dans la barre de navigation.

Vacataires Cours **Profil**

On accède directement à un formulaire pour changer son mot de passe et un bouton de déconnexion :

The image shows a web interface for a user profile. It features a green-bordered box containing a form with two input fields: "Mot de passe actuel" (Current password) and "Nouveau mot de passe" (New password). Below these fields is a blue button labeled "Changer le mot de passe" (Change password). To the right of this box, outside the green border, is a red button labeled "Déconnexion" (Disconnect).

Mot de passe actuel

Nouveau mot de passe

Changer le mot de passe

Déconnexion

- La fonction `changePassword` est déclenchée lorsqu'un utilisateur souhaite changer son mot de passe. Elle prend deux paramètres : l'ancien mot de passe (`password`) et le nouveau mot de passe (`newPassword`).
- Cette fonction utilise le service `loginService` pour envoyer une requête HTTP POST à l'API avec les données de l'ancien et du nouveau mot de passe.
- En cas de succès de la requête (dans la fonction `next`), une alerte est affichée pendant 3 secondes pour informer l'utilisateur que son mot de passe a été changé avec succès. Une autre alerte est affichée en cas d'erreur, indiquant que les anciennes informations d'identification ne sont pas valides.


```

changePassword(password: string, newPassword: string) {
  this.loginService.putNewPassword(password,newPassword).subscribe({
    next: (response) => {
      const alert = document.getElementById('alertCreated');

      if (alert != null ){
        //On affiche l'alert et on attend 3 secondes avant de la desactiver
        alert.style.display = "block"
        setTimeout(() => {
          alert.style.display = "none";
        },3000)
      }
    },
    error: (error) => {
      // Gestion des erreurs
      console.error(error);
      //On cherche l'id du composant d'alert
      const alert = document.getElementById('alertPass');

      if (alert != null ){
        //On affiche l'alert et on attend 3 secondes avant de la desactiver
        alert.style.display = "block"
        setTimeout(() => {
          alert.style.display = "none";
        },3000)
      }
    }
  });
}

```

La fonction putNewPassword envoie une requête HTTP POST à l'URL de l'API /changePassword avec les données du mot de passe actuel et du nouveau mot de passe.

```

putNewPassword(password: string, newPassword: string) {
  return this.http.post(this.apiUrl+'/changePassword', {password,newPassword})
}

```

- L'API reçoit la requête POST avec les données password (ancien mot de passe) et newPassword (nouveau mot de passe).
- L'API recherche l'utilisateur correspondant à la requête en utilisant findById et vérifie si l'utilisateur existe.
- Si l'utilisateur existe, l'API compare le mot de passe actuel (password) avec le mot de passe haché stocké dans la base de données en utilisant la fonction bcrypt.compare.
- Si la comparaison est réussie (le mot de passe actuel est correct), l'API génère un nouveau sel (salt) avec bcrypt.genSalt, puis hache le nouveau mot de passe (newPassword) avec ce sel. Le nouveau mot de passe haché est ensuite enregistré dans la base de données à la place de l'ancien mot de passe haché.
- Si la comparaison échoue (le mot de passe actuel est incorrect), l'API renvoie une réponse d'erreur indiquant que les informations d'identification ne sont pas valides.
- Si l'utilisateur n'existe pas, l'API renvoie une réponse d'erreur indiquant que l'utilisateur n'existe pas.
- En cas de succès, l'API renvoie une réponse indiquant que le mot de passe a été changé avec succès.

```
module.exports.putNewPassword = async (req, res) => {
  const {password, newPassword} = req.body;
  try {
    const user = await loginModels.findById(req.userId);
    console.log(req.userId)
    if (user) {
      bcrypt.compare(password, user.password, function (err, result) {
        if (result) {
          bcrypt
            .genSalt()
            .then(salt => {
              return bcrypt.hash(newPassword, salt)
            })
            .then(async hash => {
              user.password = hash
              user.save()
              res.status(200).json({message: "Password changed"})
            })
            .catch(err => console.error(err.message))
        } else {
          res.status(500).json({error: "Credentials not valid"});
        }
      })
    } else {
      res.status(404).json({error: "User not exist"});
    }
  } catch (error) {
    res.status(500).json({message: "message"});
  }
}
```

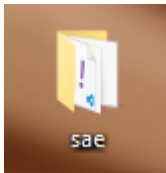
Mise en place du projet avec docker (Pour VS CODE) :

Pré requis :

- Visual Studio Code
- Git Bash
- WSL (si windows)
- Docker desktop

Mise en place :

Pour commencer il faudra créer un dossier :



Ensuite il faudra cloner les repositories github :

```
git clone https://github.com/SAE-IUT/sae5.01-gestion_vacataires.git
git clone https://github.com/SAE-IUT/sae5.01-gestion_vacataires-API.git
```

Une fois les repositories cloné et placé dans le fichier créé du départ, il va falloir créer un fichier nommé compose.yaml et il va falloir y mettre ce contenu, ce fichier doit aller dans le dossier créé précédemment :

```
# Use root/example as user/password credentials
version: '3.1'
services:
  mongo:
    image: mongo
    volumes:
      - mongodb_volume:/data/db
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: example
      MONGO_INITDB_DATABASE: test
  mongo-express:
    image: mongo-express
    ports:
      - 8081:8081
    environment:
      ME_CONFIG_MONGODB_ADMINUSERNAME: root
      ME_CONFIG_MONGODB_ADMINPASSWORD: example
      ME_CONFIG_MONGODB_URL: mongodb://root:example@mongo:27017/
```

api:

build: sae5.01-gestion_vacataires-API

depends_on:

- mongo

ports:

- 3000:3000

angular:

build: sae5.01-gestion_vacataires/s5.01-app-gestion-vacataires

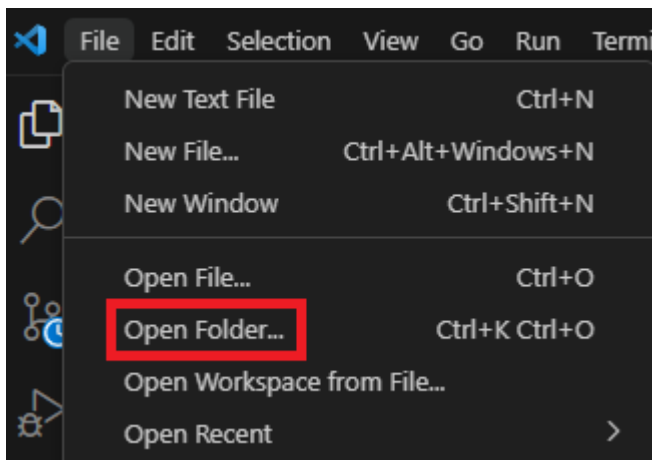
ports:

- 80:4200

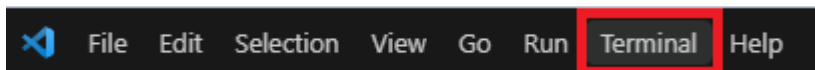
volumes:

mongodb_volume:

ensuite il faut lancer docker desktop et visual studio code, dans visual studio code on ouvre le dossiers créés précédemment :



Une fois le dossier ouvert, on va venir ouvrir un terminal :



Et pour venir on va venir inséré ces 3 commandes les une à la suite des autres :

```
docker compose up mongo mongo-express api -d --build
cd sae5.01-gestion_vacataires/s5.01-app-gestion-vacataires/
ng serve --host 0.0.0.0 --proxy-config proxy.confdev.json
```

Cela nous permettra d'avoir accès au front :

<http://localhost:4200/>

A l'API :

<http://localhost:3000/>

Et à la base mongodb avec user admin et password pass sur :

<http://0.0.0.0:8081/>