

DÉVELOPPEUR WEB
WEB MOBILE

DOSSIER DE PROJET

Projet internat

Soutenu par :

Bernez Jovany

le 22/07/2024

sous la direction de

Monsieur Bonenfant Gilles

REMERCIEMENTS

Je tiens à remercier nos formateurs, pour leur bienveillance, patience, soutiens et encouragements. Ainsi que l'ESRP Beauvoir en général, pour avoir été une maison, un refuge...

Je tiens particulièrement à remercier Sébastien Platel, mon collègue (et ami) de formation pour m'avoir donné l'heure, même sans lui avoir demandé au préalable.

SOMMAIRE

Table des matières

I . AVANT-PROPOS.....	5
II . PRÉSENTATION DE L'ENTREPRISE.....	6
III . CONTEXTE DU PROJET.....	7
IV . PROJET(S) AU(X)QUEL(S) J'AI PARTICIPÉ.....	8
IV-1 . Cahier des charges.....	8
IV-2 . Maquette(s).....	10
IV-3 . Typologie technique.....	12
IV-3.a . Logiciels utilisés.....	12
IV-3.b . Architecture technique.....	13
IV- 4 . Planning.....	13
IV-5 . Analyse.....	14
IV-5.a . Analyse Orientée Objet avec UML.....	14
IV-5.a.1 : Diagramme de cas d'utilisation (réalisé avec StarUML).....	14
Acteurs.....	14
Cas d'utilisation (Use Cases).....	14
Relations.....	16
Résumé.....	16
IV-5.a.2 : Diagramme(s) de séquence.....	16
IV-5.a.3 : Diagramme de classes (réalisé avec StarUML).....	16
Relations.....	19
Résumé.....	19
IV-5.a . Modèle logique ou physique de données.....	19
Relations.....	20
IV-6 . Implémentation.....	22
IV-6.a . <i>Fonctionnalité 1 Affichage des informations</i> :.....	22
IV-6.b . <i>Fonctionnalité 2 gestion des utilisateurs</i> :.....	23
IV-6.c . <i>Fonctionnalité 3 Bouton et couleur pour le soutirage de l'eau des chambres</i>	25
IV-6.d . <i>Fonctionnalité 4 : 2 formulaires, état des lieux d'entrée et de sortie</i>	27
IV-7.a . <i>Tests unitaires</i>	29
IV-7.a . <i>Tests d'intégration</i>	29
IV-8 . Recherches en anglais – veille technologique.....	30
IV-9 . RGPD.....	30
IV-10 . Déploiement.....	30
V . CONCLUSION.....	31
2° PROJET : LOKAMIGO.....	32
Diagramme de classes :.....	33
User case (cas d'utilisation).....	36
CONCEPTEUR DE MODELE DE DONNEES:.....	40

3 ^e projet : Express-Mongoose.....	42
---	----

I . AVANT-PROPOS

J'ai obtenu un BAC STG GSI (gestion des systèmes informatiques) en 2010, par la suite je me suis dirigé vers une université, où j'ai suivi une licence LLCE, langues, littérature et civilisations étrangères, où je me suis arrêté en 3^e année, en ayant validé 2 ans.

J'ai suivi une formation de community manager sur OpenClassroom, et vu que le domaine informatique m'a toujours intéressé j'ai souhaité me former au développement web, avec comme objectif, de développer ma propre application une fois que j'aurai les compétences nécessaires.

II . PRÉSENTATION DE L'ENTREPRISE

si stage en entreprise

Ce projet s'est fait en distanciel dans le centre de formation, avec une autre collègue en formation, et sous la tutelle de notre formateur référent.

N'ayant pas trouvé de stage, il m'a été confié le projet que je vais vous présenter par la suite. Nous avons des horaires fixes, comme en entreprise, et des délais à respecter pour chaque tâche. De plus nous faisons régulièrement des bilans, avec mon formateur et ma binôme, afin de statuer sur l'avancée, par rapport au planning établi sur Trello.fr.

III . CONTEXTE DU PROJET

Objectif(s) du projet

Client : Monsieur Barré et Infirmière de l'ESRP Beauvoir, Anne Seprey

Objectifs : Développer une application web pour gérer les chambres de l'internat.

Technologies utilisées : Framework Symfony, PostgreSQL

Le projet vise à créer une application web pour gérer l'occupation des chambres de l'internat de l'ESRP Beauvoir, un établissement du groupe UGECAM Ile-de-France. Actuellement, la gestion se fait via des tableaux Excel et des documents Word, ce qui est inefficace et sujet aux erreurs. L'application permettra de numériser ce processus, offrant une gestion plus fluide et précise.

IV . PROJET(S) AU(X)QUEL(S) J'AI PARTICIPÉ

Projet internat

IV-1 . Cahier des charges

contexte :

Le centre B souhaite numériser le système de gestion de son internat. Actuellement, l'internat de l'établissement compte 122 chambres. La gestion de l'occupation de ces chambres est répertoriée dans un tableau excel. À chaque changement de résident, un état de lieu d'entrée et de sortie est effectué. Ceux-ci sont également rédigés grâce à des documents Word qui sont archivés. Ces documents sont disponibles en annexe.

Le tableau Excel représente en même temps l'organisation des chambres par bâtiment et par étage. Chaque chambre représentée indique la capacité du lit (simple ou double) et le handicap éventuel de l'élève (mobilité réduite, malentendant, malvoyant). À l'heure actuelle on se contente de remplir chaque case avec le nom de l'étudiant et la formation qu'il suit.

Objectifs de l'application :

garder cette représentation spatiale pour plus de facilité.

L'utilisateur pourra cliquer sur une chambre pour en changer le contenu, c'est à dire le nom de l'occupant et sa formation, ainsi que son handicap éventuel. Cliquer sur une chambre permettra également de consulter les états des lieux archivés, relatifs à cette chambre. On pourra envisager un dashboard dédié pour afficher toutes les informations relatives à une chambre. Il n'est pas nécessaire d'archiver les occupants des chambres. En ce qui concerne la fonctionnalité « état des lieux », il faudra prévoir 1 formulaire qui permettra de reprendre les données précisées dans

les documents fournis par les clients. On pourra alors créer un état des lieux (sortie ou entrée) qui sera archivé et pourra être consulté plus tard. Les états des lieux sortie et entrée correspondant à une période d'occupation devront être reliés pour être retrouvés plus facilement. L'état des lieux de sortie précisera aussi si la caution a été rendue ou non. On pourra exporter un tableau récapitulatif de l'occupation des chambres sous forme de listing clair au format excel. Les états des lieux pourront également être imprimés

Fonctionnalités :

- Mise à jour des informations des chambres en temps réel (occupant, formation, handicap).
- Représentation spatiale des chambres pour une gestion visuelle facile.
- Archivage des états des lieux avec possibilité de consultation.
- Exportation des données en format Excel.
- Impression des états des lieux.
- Suivi de l'état des chambres (libres/occupées).
- Enregistrement des dates de formation des stagiaires occupant une chambre.
- Incrémentation automatique de l'état des lieux lors de l'attribution d'une chambre.
- Modification des états des lieux.
- Intégration du planning des chambres avec le planning de soutirage de l'eau.

Charte graphique/style :

Logo Ugecam et logo ESRP Beauvoir

Le site doit être aux couleurs du logo UGECAM


Police sobre

IV-2 . Maquette(s)

Les maquettes ont été réalisées à l'aide de Figma,

Administrateur / Gestionnaire et utilisateur

Page 1 : Connection

 Connectez-vous!


Email :

Mot de passe :

UGECAM - Beauvoir

Maquette de connexion

Page 8 : Voir chambre / Gestionnaire

 Déconnexion Bonjour! gestionnaire, vous êtes connecté

Accueil Les chambres

Que faire ? VOIR CHAMBRE

Chambre : 223

Bâtiment : A

Étage : 2

Capacité : lit simple

Accès PMR : non

État des lieux entrée : effectué

État des lieux sortie : en attente

Occupant :
Date entrée : 01/03/2024

BONENFANT Gilles

Formation : web 4

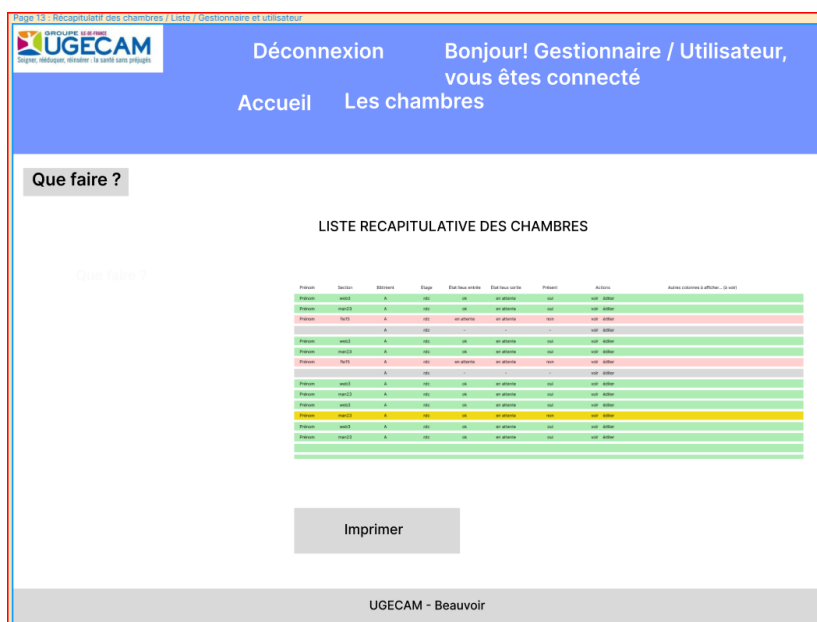
Handicap : -

Présent : oui

Entrée : 01/05/2022	Sortie : 25/02/2024	DUPONT Michel	<input type="button" value="Voir"/>
Entrée : 01/02/2022	Sortie : 16/04/2022	LAMBERT Gérard	<input type="button" value="Voir"/>
Entrée : 17/07/2021	Sortie : 20/01/2022	DURAND Jacky	<input type="button" value="Voir"/>
Entrée : 01/03/2020	Sortie : 29/06/2021	LEGRAND Kévin	<input type="button" value="Voir"/>

UGECAM - Beauvoir

Maquette de visualisation des informations d'une chambre.



Maquette de la liste des chambres

IV-3 . Typologie technique

IV-3.a . Logiciels utilisés

Durant ce stage de 8 semaines, et pour mener à bien les missions qui m'étaient confiées, j'ai utilisé les logiciels suivants :

LISTE :

FIGMA

Pour la réalisation des maquettes

StarUML

Pour réaliser le UserCase, et le diagramme de classes

XAMPP

Pour l'hébergement local et les tests de développement.

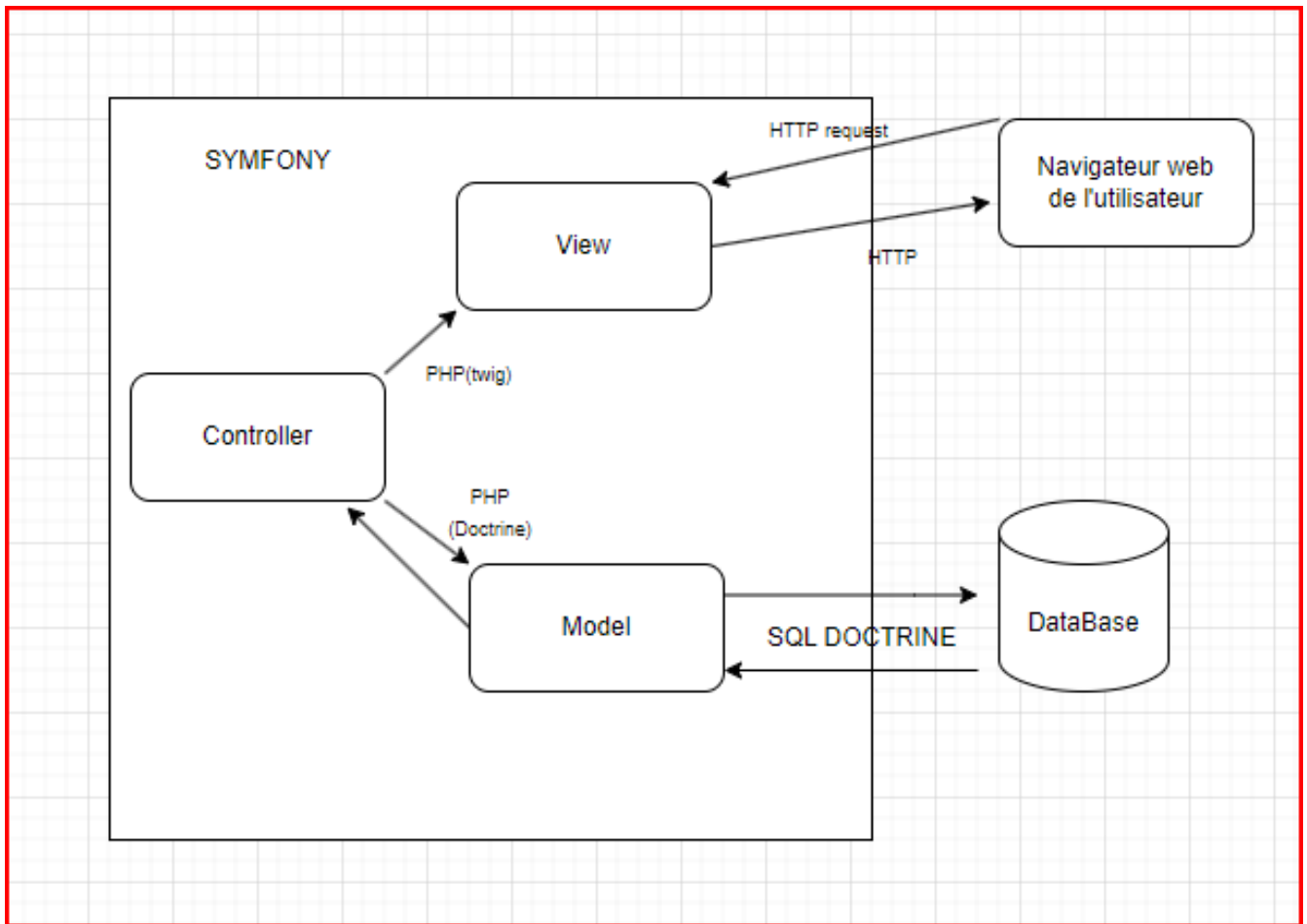
VisualStudioCode

Pour l'édition de code

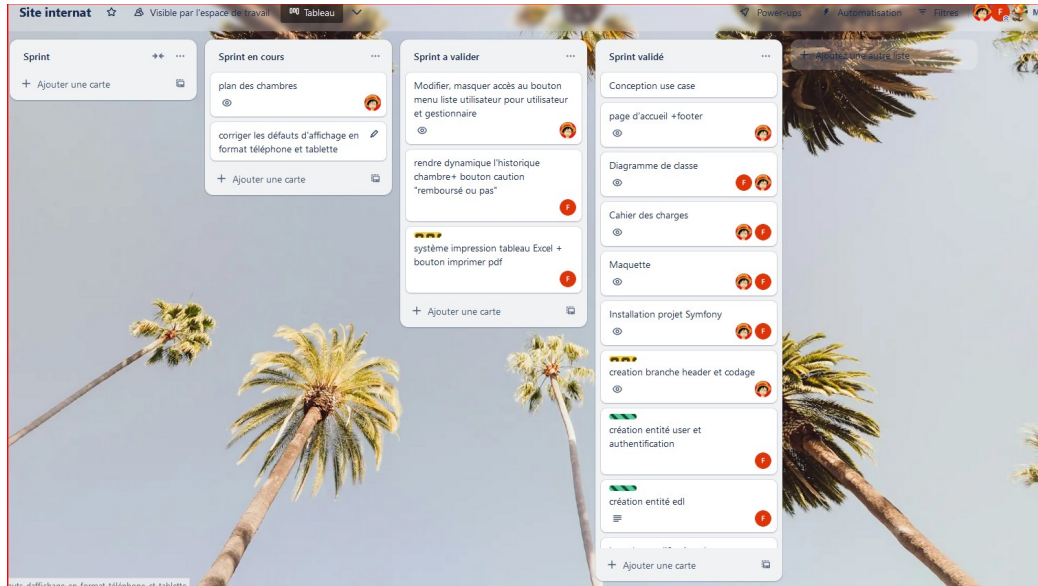
Trello

Pour le partage des tâches organisé

IV-3.b . Architecture technique



IV- 4 . Planning

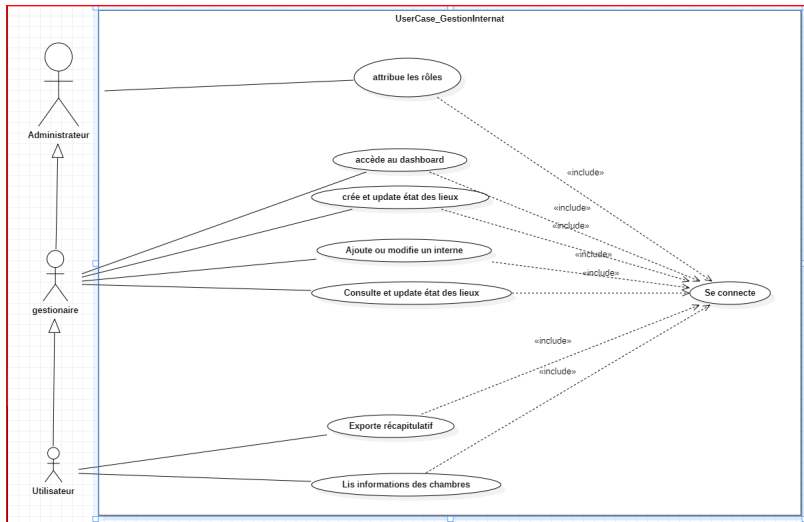


J'ai travaillé en collaboration avec ma collègue Fatima, nous nous sommes assignés les tâches, nous avons fait un partage équitable.

IV-5 . Analyse

IV-5.a . Analyse Orientée Objet avec UML

IV-5.a.1 : Diagramme de cas d'utilisation (réalisé avec StarUML)



Acteurs

- **Administrateur** : Responsable de la gestion des rôles et des accès.
- **Gestionnaire** : Gère les opérations courantes telles que la mise à jour des états des lieux et la gestion des internes.
- **Utilisateur** : Accède aux informations des chambres et peut exporter des récapitulatifs.

Cas d'utilisation (Use Cases)

1. Attribue les rôles

- Acteur : Administrateur
- Description : L'administrateur attribue différents rôles aux utilisateurs du système.

2. Accède au dashboard

- Acteur : Gestionnaire
- Description : Le gestionnaire accède à un tableau de bord pour voir les informations globales et les statistiques.

3. Crée et update état des lieux

- Acteur : Gestionnaire
 - Description : Le gestionnaire crée de nouveaux états des lieux ou met à jour les existants.
 - **Include** : Se connecte
4. **Ajoute ou modifie un interne**
- Acteur : Gestionnaire
 - Description : Le gestionnaire ajoute ou modifie les informations des internes (étudiants, résidents).
 - **Include** : Se connecte
5. **Consulte et update état des lieux**
- Acteur : Gestionnaire
 - Description : Le gestionnaire consulte et met à jour les états des lieux.
 - **Include** : Se connecte
6. **Exporte récapitulatif**
- Acteur : Utilisateur
 - Description : L'utilisateur peut exporter un récapitulatif des informations pertinentes.
 - **Include** : Se connecte
7. **Lis informations des chambres**
- Acteur : Utilisateur
 - Description : L'utilisateur lit les informations concernant les chambres disponibles ou occupées.
 - **Include** : Se connecte
8. **Se connecte**
- Acteur : Tous les acteurs (Administrateur, Gestionnaire, Utilisateur)
 - Description : Tous les utilisateurs doivent se connecter au système pour accéder à leurs fonctionnalités respectives.

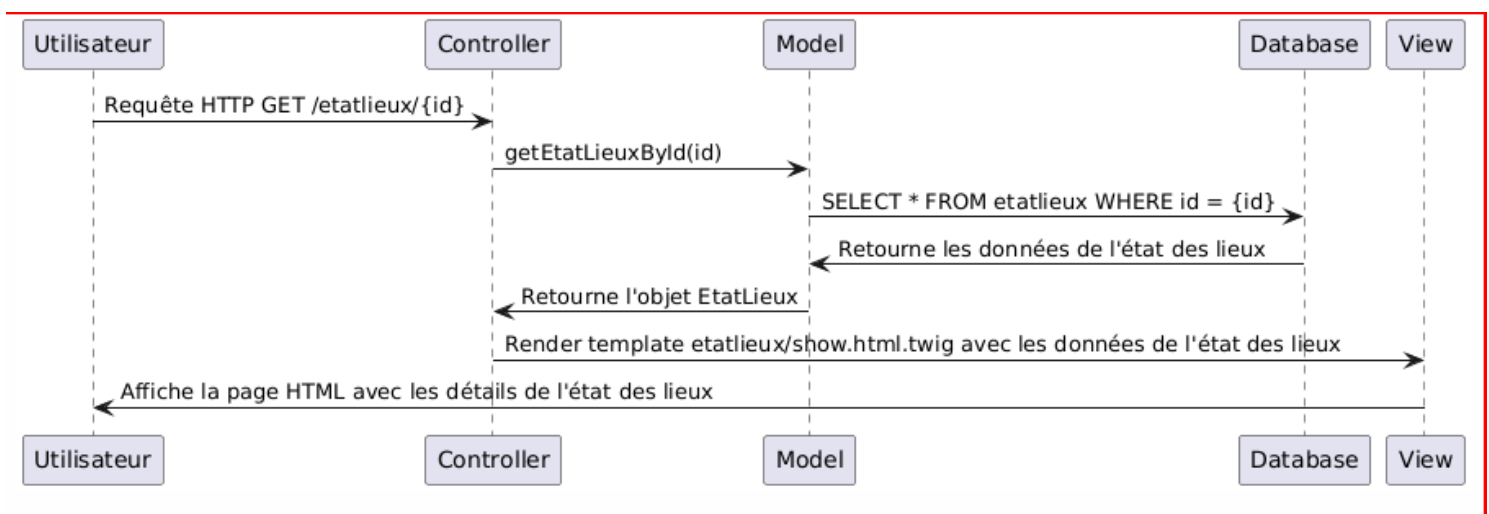
Relations

- Les relations **include** indiquent que certaines actions (comme "Créer et update état des lieux", "Ajoute ou modifie un interne", etc.) nécessitent une connexion préalable au système ("Se connecte").

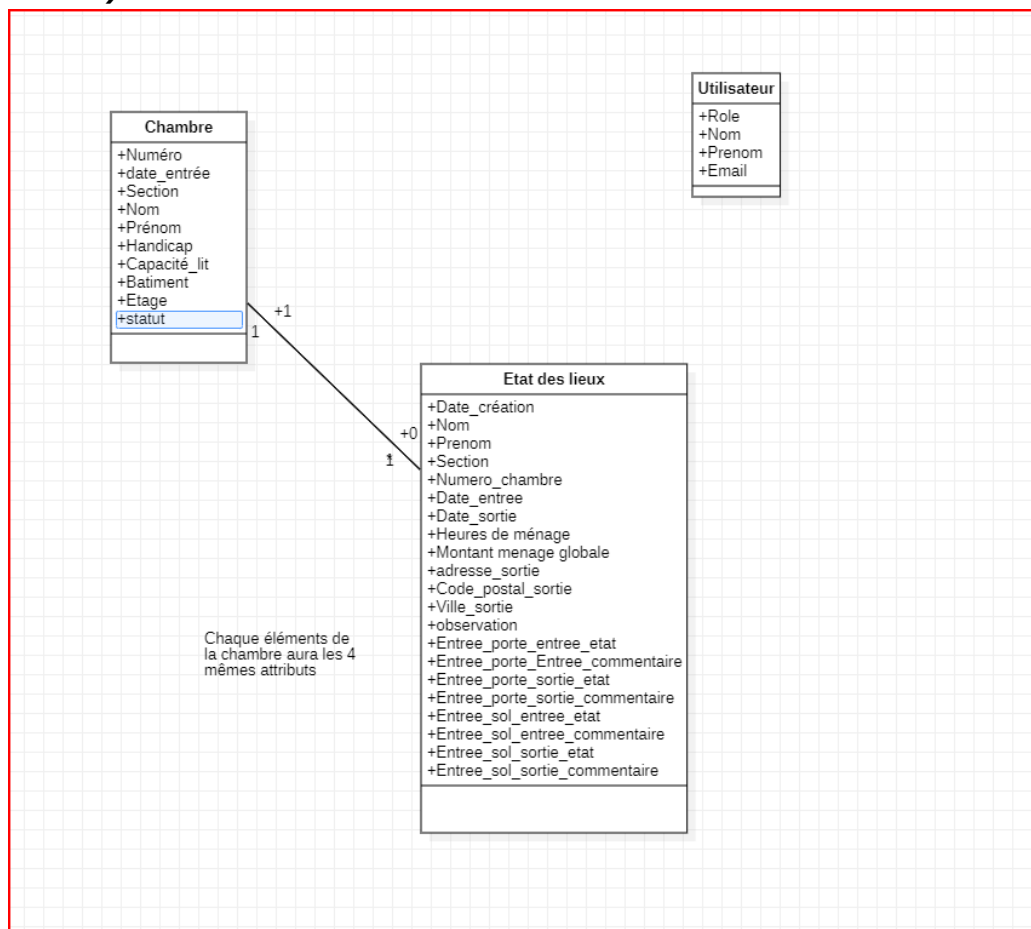
Résumé

En résumé, ce diagramme montre comment différents utilisateurs interagissent avec le système de gestion d'internat. L'administrateur gère les rôles, le gestionnaire s'occupe des opérations liées aux états des lieux et à la gestion des internes, et les utilisateurs peuvent consulter des informations et exporter des récapitulatifs, avec une connexion au système étant une étape nécessaire pour chaque action.

IV-5.a.2 : Diagramme(s) de séquence



IV-5.a.3 : Diagramme de classes (réalisé avec StarUML)



Classe Chambre

• Attributs :

- Numéro : Identifiant unique de la chambre.
- date_entrée : Date d'entrée dans la chambre.
- Section : Section ou division à laquelle appartient la chambre.
- Nom : Nom de l'occupant de la chambre.
- Prénom : Prénom de l'occupant de la chambre.
- Handicap : Informations sur le handicap de l'occupant, le cas échéant.
- Capacité_lit : Nombre de lits dans la chambre.
- Bâtiment : Nom ou numéro du bâtiment où se trouve la chambre.
- Étage : Étage où se trouve la chambre.

- **statut** : Statut de la chambre (occupée, libre, en nettoyage, etc.).

2. Classe État des lieux

• **Attributs :**

- **Date_création** : Date de création de l'état des lieux.
- **Nom** : Nom de la personne pour laquelle l'état des lieux est effectué.
- **Prénom** : Prénom de la personne pour laquelle l'état des lieux est effectué.
- **Section** : Section à laquelle la personne appartient.
- **Numéro_chambre** : Numéro de la chambre associée.
- **Date_entrée** : Date d'entrée dans la chambre.
- **Date_sortie** : Date de sortie de la chambre.
- **Heures de ménage** : Nombre d'heures de ménage effectuées.
- **Montant ménage globale** : Coût total du ménage.
- **adresse_sortie** : Adresse de sortie après le séjour.
- **Code_postal_sortie** : Code postal de l'adresse de sortie.
- **Ville_sortie** : Ville de l'adresse de sortie.
- **observation** : Remarques ou observations diverses.

• **État des éléments de la chambre (Porte, Sol) :**

- **Entree_porte_entree_etat** : État de la porte à l'entrée.
- **Entree_porte_Entree_commentaire** :
Commentaire sur l'état de la porte à l'entrée.
- **Entree_porte_sortie_etat** : État de la porte à la sortie.
- **Entree_porte_sortie_commentaire** :
Commentaire sur l'état de la porte à la sortie.
- **Entree_sol_entree_etat** : État du sol à l'entrée.
- **Entree_sol_entree_commentaire** :
Commentaire sur l'état du sol à l'entrée.
- **Entree_sol_sortie_etat** : État du sol à la sortie.

- `Entree_sol_sortie_commentaire` :
Commentaire sur l'état du sol à la sortie.

3. Classe Utilisateur

- **Attributs :**
 - `Role` : Rôle de l'utilisateur (administrateur, gestionnaire, résident).
 - `Nom` : Nom de l'utilisateur.
 - `Prénom` : Prénom de l'utilisateur.
 - `Email` : Adresse email de l'utilisateur.

Relations

- **Relation entre Chambre et État des lieux**
 - Il y a une relation 1 à N entre la classe `Chambre` et la classe `État des lieux`, ce qui signifie qu'une chambre peut avoir plusieurs états des lieux, mais chaque état des lieux est associé à une seule chambre.

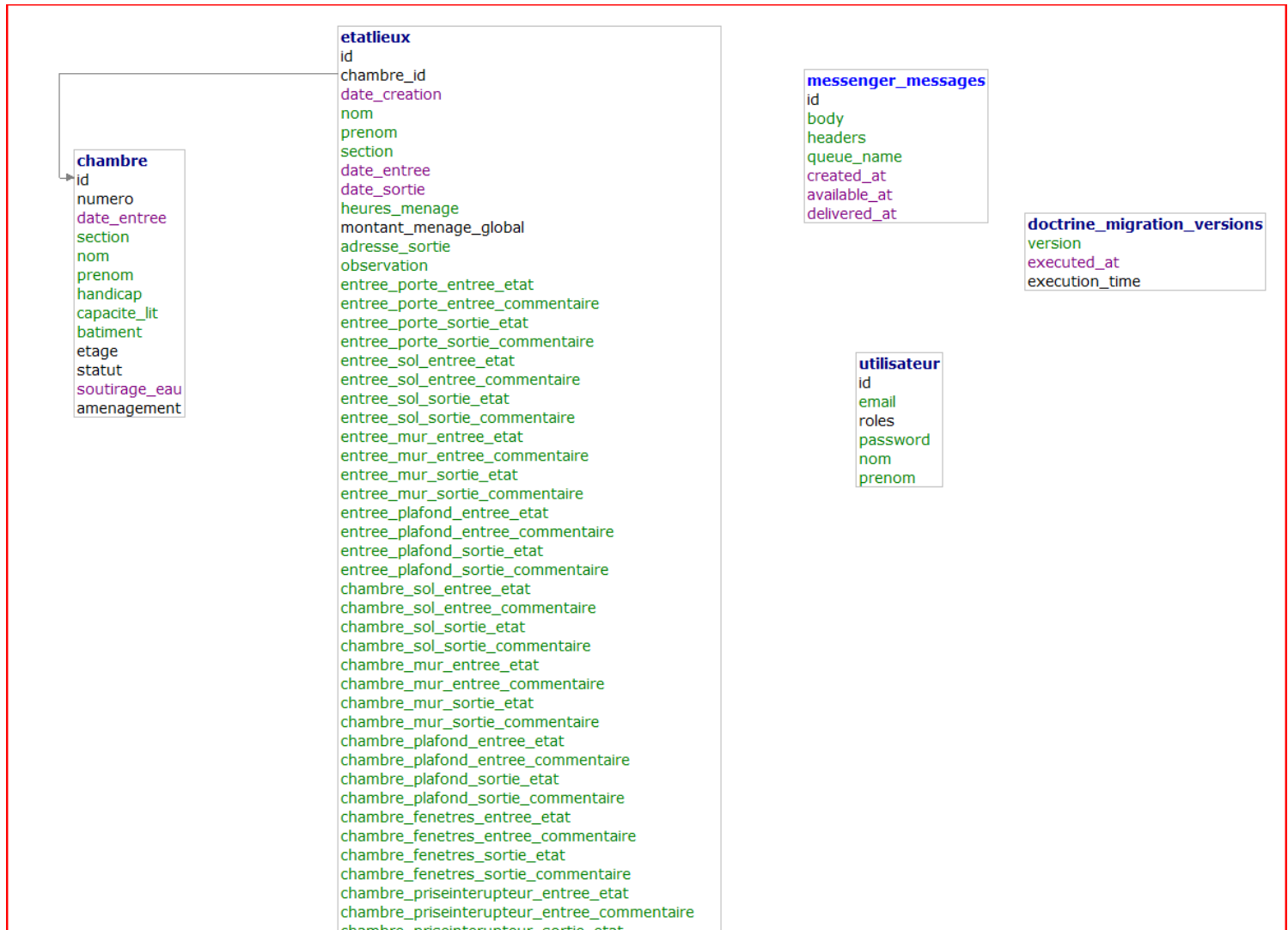
Résumé

Ce diagramme de classes illustre la structure d'un système de gestion des chambres et des états des lieux. Les chambres sont identifiées par leur numéro et ont des attributs tels que le nom, le prénom, la capacité des lits, etc. Les états des lieux enregistrent des détails spécifiques à chaque chambre, y compris les dates d'entrée et de sortie, les observations et l'état des éléments tels que la porte et le sol. La classe Utilisateur décrit les utilisateurs du système avec leurs rôles et coordonnées. La relation entre les chambres et les états des lieux montre que chaque chambre peut avoir plusieurs états des lieux associés.

Malheureusement le diagramme de classe de ce projet ne présente pas de relation ManyToMany, mais je présente une relation de ce type dans mon 2^e projet « LOKAMIGO ».

IV-5.a . Modèle logique ou physique de données

schéma de base de données relationnelle obtenu sur PostgrésSQL



Relations

- La table **etatlieux** est liée à la table **chambre** via la clé étrangère **chambre_id**.
- L'autres tables (**utilisateur**) n'a pas de relations explicites avec les tables **chambre** et **etatlieux**.

IV-6 . Implémentation

IV-6.a . Fonctionnalité 1 Affichage des informations :

les utilisateurs peuvent accéder aux informations des chambres sans pouvoir les modifier, comme indiqué dans le fichier de template index.html.twig.

```
<div class="mb-3 mt-4 p-2">
  <h1 class="row h-100 justify-content-center fw-bold">Liste des chambres</h1>
  <a href="{{ path('app_chambre_indexexport')}}" class="btn btn-success">Afficher avant l'export en excel</a>
  <table class="table table-bordered table-striped mt-3 table-hover" data-toggle='tooltip' data-placement='top' title="Couleur
rouge = soutirage d'eau à faire, vert=fait">
    <thead>
      <tr>
        <th class="bg-primary text-white">Numéro</th>
        <th class="bg-primary text-white">Date d'entrée</th>
        <th class="bg-primary text-white">Nom</th>
        <th class="bg-primary text-white">Statut</th>
        <th class="bg-primary text-white">Soutirage d'eau</th>
        <th class="bg-primary text-white">actions</th>
      </tr>
    </thead>
    <tbody>
      {% for chambre in chambres %}
        <tr>
          {% if date(chambre.soutirageEau) < date('today')|date_modify('-15 days') or (chambre.soutirageEau == null and chambre.
statut==false) %}
            class='table-danger'
          {% else %}
            class='table-success'
          {% endif %}
          <td>{{ chambre.numero }}</td>
          <td>{{ chambre.dateEntree ? chambre.dateEntree|date('d-m-Y') : '' }}</td>
          <td>{{ chambre.nom }}</td>
          <td>{{ chambre.statut ? "Occupé" : "Vide" }}</td>
          <td>{{ chambre.soutirageEau ? chambre.soutirageEau|date('d-m-Y') : '' }}
          <td>
        </tr>
      {% endfor %}
    </tbody>
  </table>
  <div class="text-align-right">
    You, 2 months ago • modifcrudchambre2
  </div>
</div>
```

Voici le résultat obtenu dans le navigateur :

					Bonjour doe Admin, vous êtes connecté !			
ESRP BEAUVOIR					Accueil	Utilisateurs	Chambres	Déconnexion
Liste des chambres								
Afficher avant l'export en excel								
Numero	Date d'entrée	Nom	Statut	Soufflage d'air	actions			
1	14-06-2024	Guillaume	Vide	02-07-2024	 Occupier	 02/07/2024	 Modifier chambre	 Modifier occupant
2	21-06-2024	Sophie2	Occupé		 Vider	 02/09/2024	 Modifier chambre	 Modifier occupant
3	27-06-2024	aboudrar2	Occupé		 Vider	 02/08/2024	 Modifier chambre	 Modifier occupant
4			Vide	03-06-2024	 Occupier	 03/06/2024	 Modifier chambre	 Modifier occupant
5			Vide	03-06-2024	 Occupier	 03/06/2024	 Modifier chambre	 Modifier occupant
6			Vide	03-06-2024	 Occupier	 03/06/2024	 Modifier chambre	 Modifier occupant
7			Vide	03-06-2024	 Occupier	 03/06/2024	 Modifier chambre	 Modifier occupant
8			Vide		 Occupier	 02/06/2024	 Modifier chambre	 Modifier occupant
9			Vide		 Occupier	 02/08/2024	 Modifier chambre	 Modifier occupant
10			Vide		 Occupier	 02/08/2024	 Modifier chambre	 Modifier occupant
11			Vide		 Occupier	 02/08/2024	 Modifier chambre	 Modifier occupant
12			Vide		 Occupier	 02/06/2024	 Modifier chambre	 Modifier occupant

IV-6.b . Fonctionnalité 2 gestion des utilisateurs :
 Le projet internat donne 3 niveaux d'affectation pour les rôles :

Créer un nouvel utilisateur

Formulaire

Nom

Prénom

Mot de passe

Email

Rôle

☐Administrateur☐Gestionnaire☐Utilisateur

Envoyer

[Retour à la liste des utilisateurs](#)

Voici la hiérarchisation des rôles

Administrateur : Peut créer des utilisateurs, en supprimer, en modifier et a les droits du rôle gestionnaire en plus.

Gestionnaire :Attribué aux utilisateurs qui ont des permissions de gestion. Les utilisateurs avec ce rôle peuvent accéder à des routes spécifiques et effectuer des actions administratives.

Utilisateur : Ont la possibilité d'accéder aux informations des chambres et des états des lieux, sans pouvoir les modifier, les supprimer ou en créer, comme précisé dans la fonctionnalité 1. Ils ont aussi la possibilité d'exporter les informations des chambres et états des lieux en format Excel.

Le rôle gestionnaire est utilisé pour contrôler l'accès aux actions spécifiques que seuls les utilisateurs ayant ce rôle peuvent effectuer.

Par exemple, les boutons *modifier chambre*, *modifier occupant*, *supprimer* (visible dans le template show), ne seront visibles que pour les utilisateurs ayant le rôle gestionnaire et administrateur.

```
{% if is_granted('ROLE_GESTIONNAIRE') %}
<div class="btn-group" role="group">
  <form action="{{ path('app_chambre_show', {'id': chambre.id}) }}" onsubmit="return confirm('Vous allez accéder aux informations de la chambre :')">
    <button class="btn btn-info me-1" data-toggle="tooltip" data-placement="top" title="Voir la chambre n°{{chambre.numero}}"><i class="fa-solid fa-eye"></i></button>
  </form>
  {{include("chambre/_change_status.html.twig")}}
  {{include("chambre/_change_soutirage_eau.html.twig")}}

  <form action="{{ path('app_chambre_edit', {'id': chambre.id}) }}">
    <button class="btn btn-warning me-1"><i class="fa-solid fa-bed"></i> Modifier chambre</button>
  </form>
  <form action="{{ path('app_chambre_occupant_edit', {'id': chambre.id}) }}">
    <button class="btn btn-warning"><i class="fa-solid fa-person-half-dress"></i> Modifier occupant</button>
  </form>
</div>
```

Dans cet extrait de code, nous pouvons voir l'utilisation de *IsGranted*, afin d'afficher ces fonctionnalités qu'aux utilisateurs ayant le rôle gestionnaire.

```
#[IsGranted('ROLE_ADMIN')]
#[Route('/new', name: 'app_utilisateur_new', methods: ['GET', 'POST'])]
14 references | 0 overrides
public function new(Request $request, EntityManagerInterface $entityManager, UserPasswordHasherInterface $userPasswordHasher):
Response
{
    $utilisateur = new Utilisateur();
    $form = $this->createForm(UtilisateurType::class, $utilisateur);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {

        //---
        $utilisateur->setPassword(
            $userPasswordHasher->hashPassword(
                $utilisateur, faboudrar, 3 months ago * creation auth user terminee
                $utilisateur->getPassword()
            )
        );

        //----

        $entityManager->persist($utilisateur);
        $entityManager->flush();

        return $this->redirectToRoute('app_utilisateur_index', [], Response::HTTP_SEE_OTHER);
    }

    return $this->render('utilisateur/new.html.twig', [
        'utilisateur' => $utilisateur,
        'form' => $form,
    ]);
}
Codeium: Refactor | Explain | Generate Function Comment | X
#[IsGranted('ROLE_ADMIN')]
#[Route('/{id}', name: 'app_utilisateur_show', methods: ['GET'])]
```

Explication de la méthode new contenue dans Utilisateur controleur :

Pour la création d'un nouvel utilisateur, dans le controleur, on déclare la méthode new, qui crée un nouvel utilisateur. Elle prend trois paramètres : \$request : L'objet Request qui contient des données de la requête HTTP.

\$entityManager : l'interface EntityManagerInterface pour interagir avec la base de données.

\$userPasswordHasher : l'interface UserPasswordHasherInterface pour hacher les mots de passe des utilisateurs.

Ensuite la méthode retourne un objet Response ligne 31

```
<?php
{
    $utilisateur = new Utilisateur();
```

création d'une nouvelle instance de l'entité Utilisateur

```
<?php
    $form = $this->createForm(UtilisateurType::class, $utilisateur);
```

Création d'un formulaire basé sur la classe UtilisateurType et lié à l'objet \$Utilisateur.

```
<?php
    $form->handleRequest($request);
```

Traitement de la requête HTTP pour remplir le formulaire avec les données soumises.

```
<?php
```

```
if ($form->isSubmitted() && $form->isValid()) {
```

Vérification si le formulaire a été soumis et s'il est valide.

```
<?php
```

```
    $utilisateur->setPassword(
```

```
        $userPasswordHasher->hashPassword(
```

```
            $utilisateur,
```

```
            $utilisateur->getPassword()
```

```
        )
```

```
    );
```

Hachage du mot de passe de l'utilisateur

appel de la méthode setPassword de l'objet Utilisateur

Utilisation de l'interface

UserPasswordHasherInterface pour hacher le mot de passe.

Le mot de passe en clair est récupéré de l'objet Utilisateur

Le mot de passe haché est retourné par hashPassword

Le mot de passe Haché est défini dans l'objet Utilisateur.

```
<?php
```

```
    $entityManager->persist($utilisateur);
```

Persistence de l'objet Utilisateur dans le contexte de l'EntityManager.

```
<?php
```

```
    $entityManager->flush();
```

Enregistrement des modifications dans la base de données.

```
<?php
```

```
    return $this->redirectToRoute('app_utilisateur_index', [],  
    Response::HTTP_SEE_OTHER);
```

Redirection vers la route app_utilisateur_index après la création réussie de l'utilisateur.

```
<?php
```

```
}
```

Fin du block if

```
<?php
```

```
return $this->render('utilisateur/new.html.twig', [
```

```
'utilisateur' => $utilisateur,
```

```
'form' => $form,
```

```
]);
```

Rendu du template new.html.twig avec les variables utilisateur et form.

Retourne la réponse rendu.

Passe l'objet Utilisateur à la vue.

Passe le formulaire à la vue.

Fin du tableau des variables.

Dernière ligne : fin de la méthode new.

IV-6.c . Fonctionnalité 3 Bouton et couleur pour le soutirage de l'eau des chambres

Numéro	Date d'entrée	Nom	Statut	Soutirage d'eau	actions
1	14-06-2024	Guillaume	Vide	02-07-2024	 Occuper  02/07/2024  Modifier chambre  Modifier occupant
2	21-06-2024	Sophie2	Occupé		 Vider  02/08/2024  Modifier chambre  Modifier occupant
3	27-06-2024	aboudrar2	Occupé		 Vider  02/08/2024  Modifier chambre  Modifier occupant
4			Vide	03-06-2024	 Occuper  03/06/2024  Modifier chambre  Modifier occupant

En plus du bouton pour le soutirage de l'eau, reconnaissable par une goutte suivie d'une date, j'ai mis en place un processus de code couleur pour les lignes, c'est à dire que les lignes dont la couleur apparaît en rouge, signifie que le soutirage n'a pas été fait, au contraire si la ligne apparaît en vert, le soutirage a été fait.

Pour le bouton, nous avons crée un formulaire
`_change_soutirage_eau.html.twig` :

```
templates > chambre > ./ _change_soutirage_eau.html.twig
```

```
You, 2 months ago | 1 author (You)
```

```
1 <form method="post" action="{{ path('app_chambre_change_soutirage_eau', {'id': chambre.id}) }}" onsubmit="return confirm('Vous
  allez actualiser la date du soutirage d'eau')">
2   <button class="btn btn-custom btn-primary me-1" data-toggle='tooltip' data-placement='top' title='Changer la date du soutirage
  d eau {{ chambre.soutirageEau|date('d/m/Y') }}"> <i class="fa-solid fa-droplet"></i> {{ chambre.soutirageEau|date('d/m/Y') }}
3   </button>
4 </form>
5
```

Ce formulaire est utilisé pour envoyer une requête au serveur via la méthode POST lorsque l'utilisateur clique sur le bouton pour changer le statut du soutirage d'eau. Lorsque le formulaire est soumis, une requête est envoyée à la route `app_chambre_change_soutirage_eau`, qui est définie dans le fichier routes de Symfony est associée à une méthode dans le contrôleur :

```
<?php
```

```
// src/Controller/ChambreController.php
```

```
#[Route('/chambre/change-soutirage-eau/{id}', name:
'app_chambre_change_soutirage_eau', methods: ['POST'])]
```

```
public function changeSoutirageEau(Request $request, Chambre $chambre,
EntityManagerInterface $entityManager): Response
```

```
{
```

```
    // Vérification du token CSRF
```

```
    if (!$this->isCsrfTokenValid('change_soutirage_eau' . $chambre->getId(), $request-
>request->get('_token')) {
```

```
        // Changement du statut du soutirage d'eau
```

```
        $chambre->setSoutirageEau(!$chambre->getSoutirageEau());
```

```
        $entityManager->flush();
```

```
    }
```

```
    return $this->redirectToRoute('app_chambre_index');
```

```
}
```

Dans la méthode `changeSoutirageEau` du contrôleur, se produit un changement du statut (si c'était « fait », ça devient « à faire » et vice et versa). Ensuite vient la sauvegarde dans la base de données : grâce à Doctrine on persiste les changements dans la

base de données. La méthode flush() enregistre les modifications dans la base de données.

Après avoir changé le statut et sauvegardé les modifications, l'utilisateur est redirigé vers la liste des chambres. Ce processus garantit que les changements sont correctement enregistrés dans la couche physique (base de données)

IV-6.d . Fonctionnalité 4 : 2 formulaires, état des lieux d'entrée et de sortie

Formulaire d'Entrée

Le formulaire d'entrée est utilisé pour enregistrer les informations lorsqu'un nouvel occupant entre dans une chambre. Voici les étapes typiques et les éléments inclus dans ce formulaire :

-démarrage du formulaire : `{{ form_start(form) }}`

-Affichage du numéro de la chambre `<p>Chambre N° : {{ chambre.numero }}</p>`

- Affichage de la date de création de l'état des lieux : `<p>Date de création : {{ etatlieux.dateCreation|date('d-m-Y') }}</p>`

-Affichage des informations de l'occupant telles que le nom, le prénom, la section ect :


```

<div class="col-6">
  <div class="m-3">
    <p>Date de création : {{ etatlieux.dateCreation|date('d-m-Y') }}</p>
  </div>

  <p class="fs-4 fw-bold">Occupant : </p>

  <table class="w-100">
    <tbody>
      <tr>
        <th class="w-25">Nom</th>
        <td><div>{{ etatlieux.nom }}</div></td>
      </tr>
      <tr>
        <th>Prénom</th>
        <td>{{ etatlieux.prenom }}</td>
      </tr>
      <tr>
        <th>Section</th>
        <td>{{ etatlieux.section }}</td>
      </tr>
      <tr>
        <th>Date d'entrée</th>
        <td>{{ etatlieux.dateEntree|date('d-m-Y') }}</td>
      </tr>
    </tbody>
  </table>
</div>

```

-bouton pour soumettre le formulaire :

```
<button type="submit" class="btn btn-primary">Enregistrer</button>
```

-Fin du formulaire : {{ form_end(form) }}

Les deux formulaires, d'entrée et de sortie, sont utilisés pour gérer les informations des occupants dans les chambres. Le formulaire d'entrée enregistre les informations lorsqu'un nouvel occupant entre dans une chambre, tandis que le formulaire de sortie enregistre les informations lorsqu'un occupant quitte une chambre. Les deux formulaires affichent des informations similaires mais adaptées au contexte spécifique (entrée ou sortie) et incluent des boutons pour soumettre les données.

Mise à jour des états et commentaires :Le controlleur EtatlieuxControlleur gère la mise à jour des états et des commentaires des différents éléments des chambres et des salles de bain.

Les méthodes de ce contrôleur permettent de copier les commentaires et les états des éléments à l'entrée vers les champs correspondants à la sortie.

IV-7 . Tests

IV-7.a . Tests unitaires

Afin de réaliser, que la connexion utilisateur fonctionne bien, j'ai vérifié que l'utilisateur peut se connecter avec des informations d'identification valides, et vérifié que l'utilisateur est redirigé vers la page appropriée après la connexion.

J'ai ensuite réalisé des tests de connexion avec des informations incorrectes. Je me suis assuré que l'utilisateur avait le bon message d'erreur adéquat.

J'ai réalisé des tests de connexion avec chacun des 3 rôles, en m'assurant qu'ils ont bien accès aux informations qui leur correspondent.

J'ai vérifié que chaque utilisateur peut se déconnecter, et redirigé vers la page de connexion.

Je me suis assuré qu'un utilisateur non connecté est redirigé vers la page de connexion lorsqu'il essaie d'accéder aux informations des chambres.

Je me suis assuré que les informations spécifiques des chambres sont correctement affichés.

IV-7.b . Tests d'intégration

Un test d'intégration s'assure que les différents composants de l'application fonctionnent ensemble comme prévu, en simulant des scénarios réels d'utilisation. Dans notre projet, ma binôme a travaillé sur la partie concernant l'état des lieux, tandis que je me suis concentré sur la partie concernant les chambres. Pour vérifier l'intégration de ces deux parties, j'ai effectué plusieurs tests d'intégration pour l'affichage et l'enregistrement des chambres.

Création et Affichage des Chambres

J'ai commencé par créer des chambres avec les informations correspondantes, y compris l'ajout d'un occupant/stagiaire. J'ai ensuite vérifié que ces informations étaient correctement enregistrées dans la base de données. Pour cela, je me suis rendu dans l'index des chambres, où la liste des chambres est affichée, et j'ai confirmé que les nouvelles chambres apparaissaient correctement.

Suppression des Occupants

Ensuite, j'ai vidé plusieurs chambres de leurs occupants pour m'assurer que la suppression fonctionnait correctement. J'ai vérifié que les chambres étaient bien mises à jour dans la base de données et que l'interface utilisateur reflétait ces changements.

Fonctionnalité de Soutirage d'Eau

J'ai également testé le bouton de soutirage d'eau pour m'assurer qu'il fonctionnait comme prévu. J'ai vérifié que le système de couleurs (vert/rouge) fonctionnait correctement : la ligne correspondante s'affichait en rouge si le soutirage d'eau n'était pas fait et en vert s'il était fait. Cela m'a permis de confirmer que le changement de statut était bien pris en compte et affiché correctement.

Édition des Chambres et des Utilisateurs

Enfin, j'ai vérifié que l'édition des chambres et des utilisateurs fonctionnait correctement. J'ai modifié les informations d'une chambre et d'un utilisateur, puis j'ai vérifié que ces modifications étaient bien enregistrées dans la base de données et reflétées dans l'interface utilisateur.

Conclusion

Ces tests d'intégration m'ont permis de m'assurer que les différentes fonctionnalités liées aux chambres fonctionnent correctement ensemble, de la création à l'affichage, en passant par la suppression et l'édition. Ils garantissent que les interactions entre les composants de l'application sont cohérentes et fiables, offrant ainsi une expérience utilisateur fluide et sans erreurs.

IV-8 . Recherches en anglais – veille technologique

Quand je ne trouvais pas de solution, j'allais sur des sites comme, MDN Web Docs (<https://developer.mozilla.org/fr/>), qui couvre tout, du CSS aux dernières API JavaScript et Stack Overflow (<https://stackoverflow.blog/>), afin de trouver des réponses, car la plupart des questions ont déjà été posées. J'allais aussi sur d'autres sites divers

IV-9 . RGPD

IV-10 . Déploiement

Si je voulais déployer ce projet, je le ferai grâce à l'hébergeur O2Switch sur lequel j'ai un abonnement. J'importerais la BDD, dans phpMyAdmin sur mon espace personnel O2Switch. Vu que j'ai déjà un nom de domaine (mondouvert), je créerais un sous-domaine, par exemple, internatbeauvoir. A l'aide de Filezilla, je transférerais le dossier contenant le projet dans le dossier public_html à l'aide du gestionnaire de fichiers de O2Switch.

Je lierais mon sous domaine à mon nom de domaine, afin d'obtenir par exemple « internatbeauvoir.mondouvert.com » que je lierais au dossier que j'ai transféré précédemment dans public_html.

Pour finir, je créerais un certificat SSL pour cette adresse à l'aide de l'outil LetsEncrypt de O2Switch, afin d'avoir une URL sécurisée avec HTTPS.

V . CONCLUSION

Ce projet était particulièrement intéressant pour plusieurs raisons : Le côté travail en binôme était motivant, et enrichissant, on s'entraidait quand on en ressentait le besoin. Le fait que ce projet était pour l'ESRP Beauvoir, destiné aux professionnels de l'établissement, était d'autant plus motivant, j'avais réellement l'impression d'être en stage au cours des différentes étapes, notamment lors des interviews des professionnels afin d'obtenir les informations nécessaires par rapport aux attentes du projet. J'ai apprécié également le fait d'avoir le soutien de nos formateurs, qu'ils se montrent aussi présents. Les problèmes que j'ai rencontrés sont plus dus à ma problématique de santé, fatigue, douleurs, concentration.

2^e PROJET : LOKAMIGO

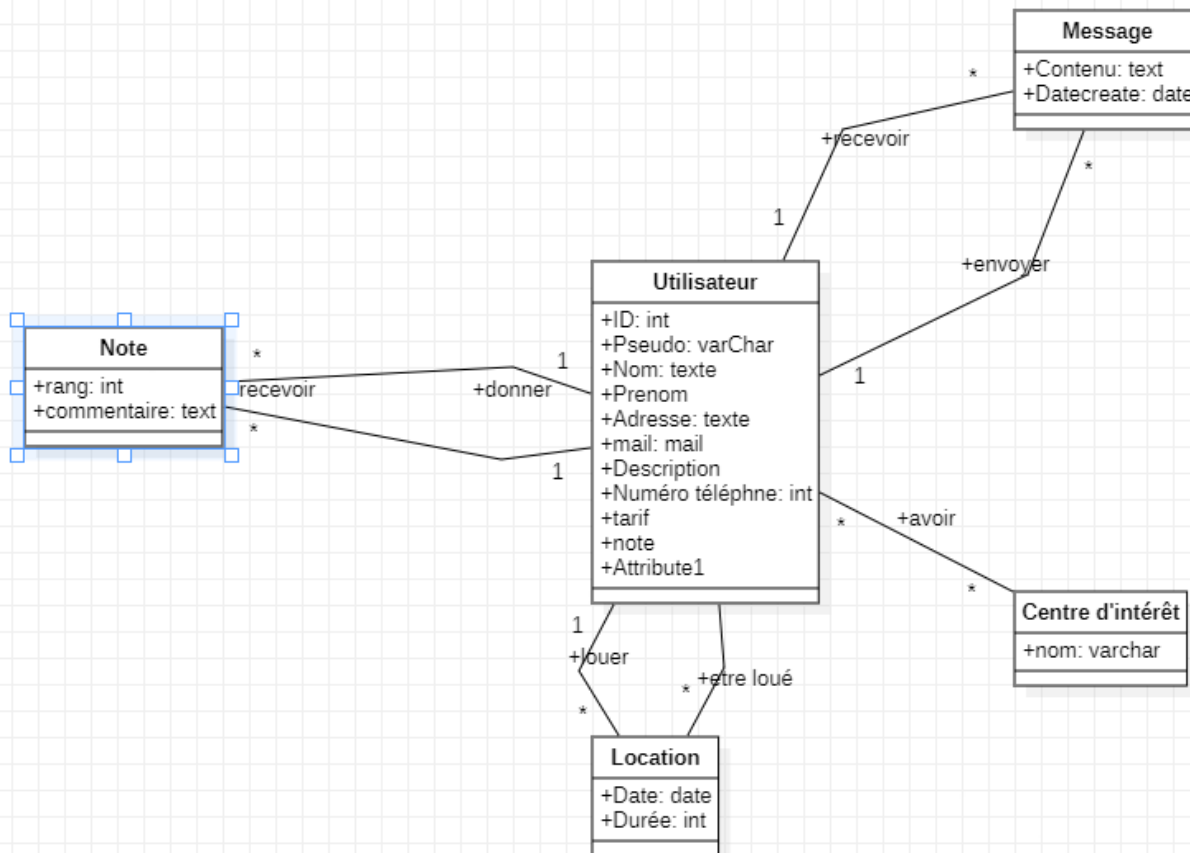
J'ai choisi de présenter mon projet LOKAMIGO, dont le modèle de données est plus complexe, afin d'avoir plus de matière à présenter.

Mon application permet de louer des amis. Vous pouvez créer un compte en tant que client pour louer un ami, ou en tant que "produit" - un ami disponible à la location. Que ce soit pour une activité, un événement, ou simplement pour sortir de la solitude, l'application offre une solution pratique.

L'objectif est de proposer une plateforme où vous pouvez être le produit ou simplement louer des services amicaux. Le service met en relation les utilisateurs avec des amis temporaires pour différentes occasions.

Ce projet a été développé avec Symfony, mais il n'a pas été mené à terme. Je vais donc présenter uniquement la partie conception, qui comprend un modèle relationnel de données complexe que le projet internat.

Diagramme de classes :



Explication :

Classes Principales et leurs Attributs

1. Utilisateur

- **ID** : int (Identifiant unique de l'utilisateur)
- **Pseudo** : varChar (Pseudo de l'utilisateur)
- **Nom** : texte (Nom de l'utilisateur)
- **Prenom** : texte (Prénom de l'utilisateur)
- **Adresse** : texte (Adresse de l'utilisateur)
- **mail** : mail (Adresse e-mail de l'utilisateur)
- **Description** : texte (Description de l'utilisateur)
- **Numéro téléphone** : int (Numéro de téléphone de l'utilisateur)
- **tarif** : (Tarif associé à l'utilisateur)
- **note** : (Note de l'utilisateur)

2. Message

- **Contenu** : text (Contenu du message)
- **Datecreate** : date (Date de création du message)

3. Note

- **rang** : int (Rang ou score de la note)
- **commentaire** : text (Commentaire associé à la note)

4. Location

- **Date** : date (Date de la location)
- **Durée** : int (Durée de la location)

5. Centre d'intérêt

- **nom** : varchar (Nom du centre d'intérêt)

Associations et Cardinalités

1. Utilisateur et Message

- Un utilisateur peut envoyer plusieurs messages.
(cardinalité OneToMany)
- Un utilisateur peut recevoir plusieurs messages.
(cardinalité de OneToMany)

2. Utilisateur et Note

- Un utilisateur peut recevoir plusieurs notes. (de OneToMany)
- Un utilisateur peut donner plusieurs notes. (cardinalité de OneToMany)

3. **Utilisateur et Location**

- Un utilisateur peut louer plusieurs locations. (cardinalité de OneToMany)
- Une location peut être louée par plusieurs utilisateurs. (cardinalité de ManyToMany)

4. **Utilisateur et Centre d'intérêt**

- Un utilisateur peut avoir plusieurs centres d'intérêt. (cardinalité de OneToMany)
- Un centre d'intérêt peut être associé à plusieurs utilisateurs. (cardinalité de ManyToMany) Dans cet exemple, contrairement au 1^{er} projet, nous avons ici une relation ManyToMany entre les classes User, et centre_dinteret. La relation ManyToMany sera représentée par une table intermédiaire qui permet de traiter cette relation dans la base de données.

Interprétation Générale :

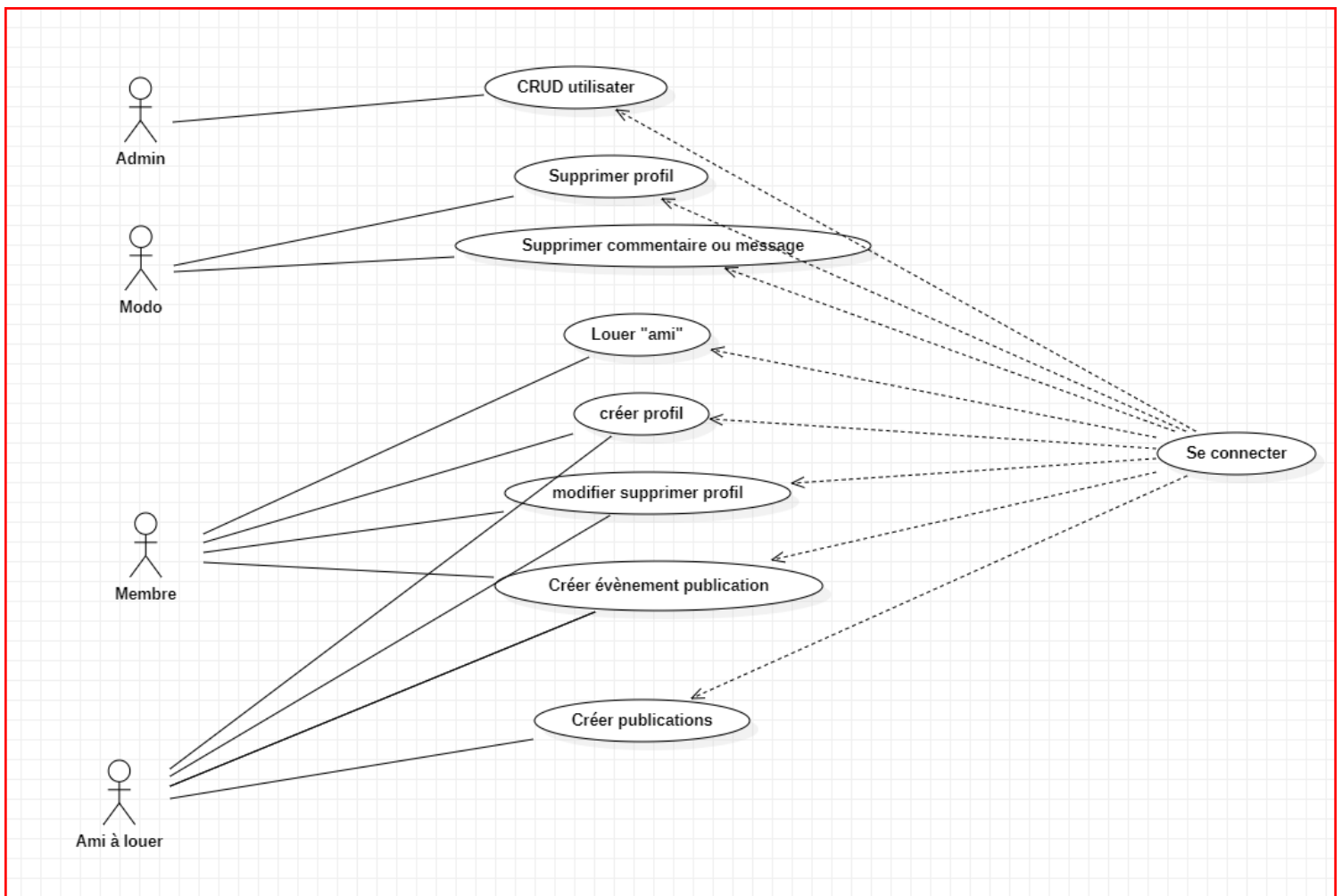
Ce diagramme de classes montre un système où des utilisateurs peuvent interagir de plusieurs manières :

- **Communication** : Les utilisateurs peuvent s'envoyer des messages entre eux.
- **Évaluation** : Les utilisateurs peuvent noter et être notés par d'autres utilisateurs, avec des commentaires associés à ces notes.
- **Location** : Les utilisateurs peuvent louer des biens/services et ces locations peuvent être associées à des dates et des durées spécifiques.
- **Intérêts** : Les utilisateurs peuvent avoir des centres d'intérêt multiples qui peuvent être partagés avec d'autres utilisateurs.

Ce diagramme modélise un système complexe d'interactions entre

utilisateurs, comprenant la communication, l'évaluation, les locations, et les intérêts personnels.

User case (cas d'utilisation)



EXPLICATION :

Ce diagramme de cas d'utilisation représente les interactions possibles entre différents rôles d'utilisateurs (Admin, Modo, Membre, et Ami à louer) et les différentes actions qu'ils peuvent effectuer dans un système. Voici une explication détaillée de chaque rôle et des actions associées :

Rôles d'utilisateur

1. Admin

- **CRUD utilisateur** : L'admin a la capacité de créer, lire, mettre à jour et supprimer les profils d'utilisateurs. Il a un contrôle total sur la gestion des utilisateurs du système.
- **Supprimer profil** : L'admin peut également supprimer les profils d'utilisateurs.
- **Supprimer commentaire ou message** : L'admin peut supprimer des commentaires ou des messages postés par les utilisateurs.

2. Modo (Modérateur)

- **Supprimer commentaire ou message** : Le modérateur peut supprimer des commentaires ou des messages postés par les utilisateurs, mais n'a pas les mêmes privilèges étendus que l'admin pour la gestion complète des utilisateurs.

3. Membre

- **Se connecter** : Les membres peuvent se connecter au système.
- **Créer profil** : Les membres peuvent créer leur propre profil.
- **Modifier/supprimer profil** : Les membres peuvent modifier ou supprimer leur propre profil.
- **Créer évènement/publication** : Les membres peuvent créer des évènements ou des publications dans le système.

- **Créer publications** : Les membres peuvent créer des publications.
 - **Louer "ami"** : Les membres peuvent louer des "amis".
4. **Ami à louer**
- **Se connecter** : Les "amis à louer" peuvent se connecter au système.
 - **Créer profil** : Ils peuvent créer leur propre profil.
 - **Modifier/supprimer profil** : Ils peuvent modifier ou supprimer leur propre profil.
 - **Créer évènement/publication** : Ils peuvent créer des évènements ou des publications dans le système.
 - **Créer publications** : Ils peuvent créer des publications.
 -

Actions détaillées

1. **CRUD utilisateur (Admin)**
 - **Créer** : Ajouter un nouveau profil utilisateur dans le système.
 - **Lire** : Voir les détails des profils utilisateurs.
 - **Mettre à jour** : Modifier les informations des profils utilisateurs.
 - **Supprimer** : Effacer les profils utilisateurs du système.
2. **Supprimer profil (Admin)**
 - Supprimer définitivement un profil utilisateur du système.
3. **Supprimer commentaire ou message (Admin, Modo)**
 - Enlever des commentaires ou des messages inappropriés ou non conformes aux règles du système.
4. **Louer "ami" (Membre)**
 - Permet aux membres de louer les services d'un "ami" pour une certaine période ou activité.
5. **Créer profil (Membre, Ami à louer)**
 - Les utilisateurs peuvent enregistrer leurs informations personnelles pour créer un profil dans le système.
6. **Modifier/supprimer profil (Membre, Ami à louer)**

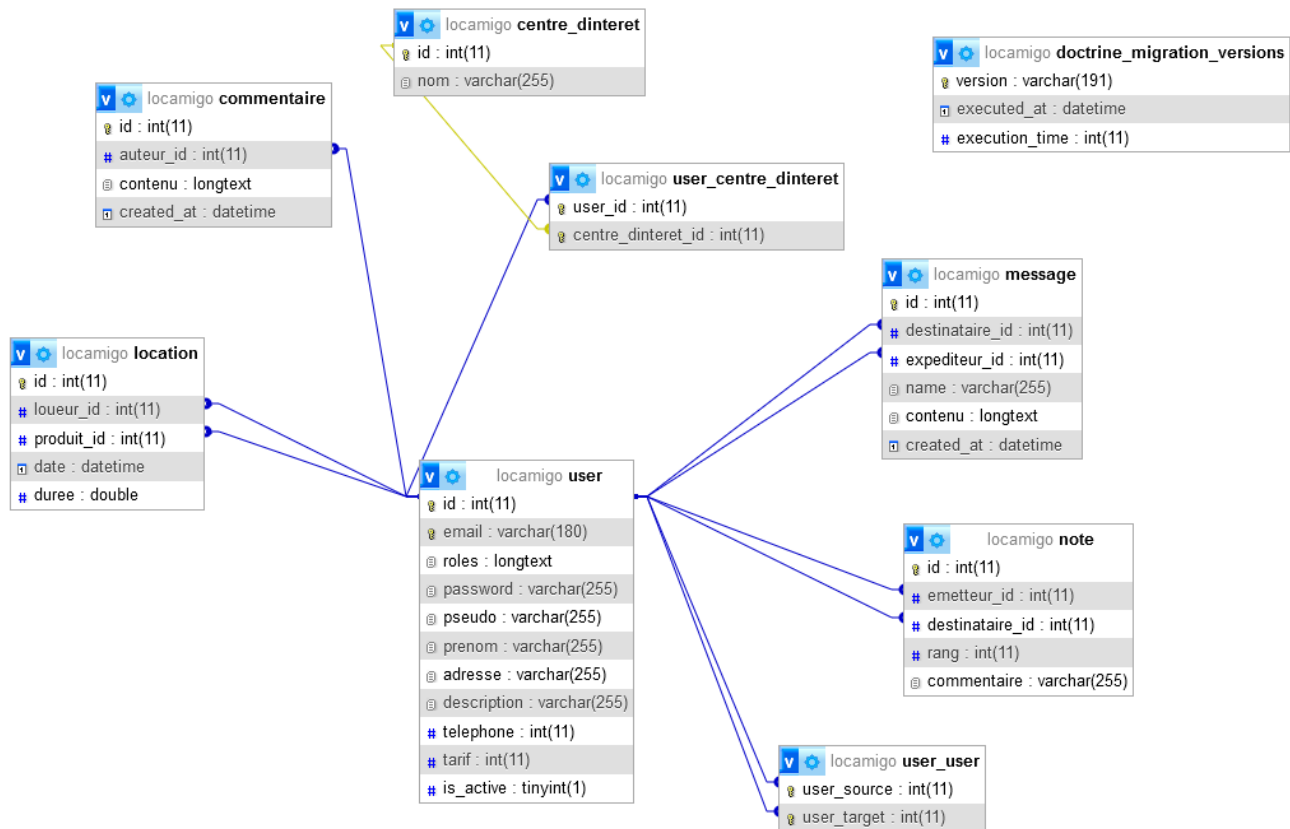
- Les utilisateurs peuvent mettre à jour leurs informations personnelles ou supprimer leur profil du système.
- 7. **Créer évènement/publication (Membre, Ami à louer)**
 - Les utilisateurs peuvent créer des évènements (par exemple, des réunions, des activités) ou des publications (par exemple, des articles, des annonces).
- 8. **Créer publications (Membre, Ami à louer)**
 - Les utilisateurs peuvent créer des publications spécifiques dans le système.
- 9. **Se connecter (Tous)**
 - Tous les utilisateurs doivent se connecter au système pour accéder aux fonctionnalités correspondantes à leur rôle.

Connexions

- **Se connecter** : Cette action est le point commun pour tous les utilisateurs avant de pouvoir effectuer d'autres actions spécifiques à leurs rôles respectifs.
- Les lignes pleines représentent les actions directement accessibles par les rôles spécifiques.
- Les lignes en pointillés indiquent les actions accessibles après la connexion initiale.

Ce diagramme montre clairement les capacités et les permissions de chaque rôle utilisateur, ce qui aide à comprendre la structure de gestion des utilisateurs et des contenus dans ce système.

Modèle physique de données:



- **Relation entre user et message :**

- **Colonnes :**

- destinataire_id : Référence à user.id (utilisateur qui reçoit le message).
- expediteur_id : Référence à user.id (utilisateur qui envoie le message).

- **Explication :**

- Dans cette relation, un utilisateur peut être à la fois l'expéditeur et le destinataire d'un message. Chaque message a un utilisateur qui l'a envoyé (expediteur_id) et un utilisateur qui le reçoit (destinataire_id).

Relation entre user et note :

- **Colonnes :**
 - `emetteur_id` : Référence à `user.id` (utilisateur qui a émis la note).
 - `destinataire_id` : Référence à `user.id` (utilisateur qui reçoit la note).
- **Explication :**
 - Ici, un utilisateur peut évaluer un autre utilisateur. L'utilisateur qui émet la note (`emetteur_id`) évalue l'utilisateur qui reçoit la note (`destinataire_id`). Cette relation permet d'avoir une évaluation ou une critique entre utilisateurs.

Relation entre `user` et `user_user` :

- **Colonnes :**
 - `user_source` : Référence à `user.id` (utilisateur source).
 - `user_target` : Référence à `user.id` (utilisateur cible).

Les doubles relations permettent de modéliser des interactions complexes entre les utilisateurs. Voici comment elles fonctionnent dans l'application locamigo :

- **Messagerie :**
 - Un utilisateur peut envoyer des messages à d'autres utilisateurs, et il peut également recevoir des messages d'autres utilisateurs. Cela crée une interaction bidirectionnelle entre les utilisateurs via les messages.
- **Évaluation :**
 - Les utilisateurs peuvent s'évaluer les uns les autres. Cela permet de garder une trace des évaluations émises et reçues par chaque utilisateur, facilitant ainsi une réputation basée sur les notes.
- **Relations entre utilisateurs :**
 - Les relations entre utilisateurs permettent de modéliser des connexions sociales ou d'autres types de relations

spécifiques entre les utilisateurs. Cela peut être utilisé pour des fonctionnalités sociales au sein de l'application.

3^e projet : Express-Mongoose

je vais vous présenter l'exercice que l'on a réalisé en cours de formation « **node_js_mongodb** », afin d'exposer un projet NoSQL. Ce projet est une API (Application Programming Interface) **Node.js** qui utilise Express pour gérer les requêtes HTTP et **MongoDB** pour stocker les données.

MongoDB est un système de base de données non relationnelle.

L'avantage par rapport aux bdd relationnelles, est le fait d'être flexible, pouvoir être déployé sur plusieurs serveurs contrairement au modèle SQL

J'ai configuré Node.js, voici les étapes :

j'ai créé un dossier nommé node_1, j'ai ouvert ce dossier dans VS Code, et j'ai fait la manipulation **npm init -y** dans le terminal, afin d'initialiser rapidement mon nouveau projet, cette commande va créer un fichier package.json dans mon repertoire de projet avec des valeurs par défaut.

Npm est le gestionnaire de paquets pour Node.js

init : sous commande pour initialiser un nouveau projet

-y : flag qui accepte automatiquement toutes les valeurs par défaut.

J'ai ensuite installé les packages nécessaires :

je les ai importés, par exemple ici dans mon fichier central, index.js en tapant :

```
const express = require('express');
```

```
const cors = require('cors');
```

Express : qui est un framework flexible pour Node.js, il simplifie le développement d'applications web et d'APIs en fournissant un ensemble robuste de fonctionnalités pour les applications web et mobiles, il va servir à gérer mon serveur Node.

Cors (cross-origin Ressource Sharing): mécanisme de sécurité implémenté par les navigateurs web pour contrôler l'accès aux ressources, comme les APIs, situées sur un domaine différent de celui de l'application. Il va donc me servir à ajouter des entêtes http afin d'accéder à des ressources d'un autre serveur.

Voici la configuration de cors dans le projet, pour que le front puisse utilisé l'API.

```
app.use(cors({
  origin: "*",
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  allowedHeaders: "Origin, X-Requested-With, x-access-token, role, Content, Accept, Content-Type, Authorization"
}))
```

jeremy-di, 4 months ago • documentation et users

J'ai ensuite initialisé mon API :

```
const app = express()
app.listen(8080, () => {
  console.log("Vous êtes connecté sur le port 8080")
})
```

On a crée un dossier models dans lequel on ajoute le fichier User.js qui contient les propriétés de mes utilisateurs. La BDD aura besoin de ce modèle pour connaître les données traitées par l'AP

Dans l'exemple suivant, le fichier User.JS de mon projet utilise Mongoose pour définir un schéma de données pour les utilisateurs dans une base de données MongoDB.

```

1  const mongoose = require('mongoose');
2  const bcrypt = require('bcrypt')
3
4  const userSchema = new mongoose.Schema(
5    {
6      email : {
7        type : String,
8        require : true
9      },
10     password : {
11       type : String,
12       require : true
13     },
14     role : {
15       type: String,
16       require : true,
17       default : "utilisateur"
18     }
19   },
20   {
21     timestamps : true
22   }
23 )
24
25 userSchema.pre("save", async function(){
26   if (this.isModified("password")) {
27     this.password = await bcrypt.hash(this.password, 10)
28   }
29 })
30
31 const User = mongoose.model("User", userSchema)
32
33 module.exports = User

```

1- On importe les bibliothèques:

```
const mongoose = require('mongoose');  
const bcrypt = require('bcrypt');
```

mongoose pour interagir avec MongoDB

bcrypt pour chiffrer les mots de passe

2- On définit le schéma :

```
const userSchema = new mongoose.Schema({  
  email: {  
    type: String,  
    require: true  
  },  
  password: {  
    type: String,  
    require: true  
  },  
  role: {  
    type: String,  
    require: true,  
    default: "utilisateur"  
  },  
  timestamps: true  
});
```

email, password et rôle sont les champs

3- On chiffre le mot de passe avec bcrypt

```
userSchema.pre("save", async function() {  
  if (this.isModified("password")) {  
    this.password = await bcrypt.hash(this.password, 10);  
  }  
});
```

Avant de sauvegarder un utilisateur, le mot de passe est chiffré.

4- On crée le modèle

```
const User = mongoose.model("User", userSchema);
```

User est le modèle ici, utilisé pour interagir avec la collection « users » dans MongoDB.

5- On exporte le modèle

```
module.exports = User;
```

Ici le modèle User est exporté pour être utilisé dans d'autres parties de l'application.

Le fichier userController.js du projet :

```
jeremy-di, 4 months ago | 1 author (jeremy-di)
const express = require('express');
const router = express.Router();
const User = require('../models/User');
const bcrypt = require('bcrypt');
const jwt = require('jsonwebtoken');

// Route d'inscription
router.post('/register', async(req, res) => {
  try {
    const searchUser = await User.findOne({email : req.body.email})
    if ( searchUser ) {
      return res
        .status(403)
        .json({status : 403, message : `L'utilisateur portant l'email : ${req.body.email} existe déjà`})
    }
    const user = new User(req.body)
    console.log(req.body)
    const newUser = await user.save()
    return res
      .status(201)
      .json({status : 201, message : `L'utilisateur ${newUser.email} à bien été créé`})
  } catch (error) {
    res.sendStatus(500)
    console.log(error)
  }
})

// Route de connexion
router.post('/login', async(req, res) => {
  try {
    const user = await User.findOne({email : req.body.email})
    if ( !user ) {
      return res
        .status(400)
        .json({status : 400, message : "Identifiants invalide"})
    }
    const isMatch = await bcrypt.compare(req.body.password, user.password)
```

Il utilise Express.js pour gérer les routes d'inscription et de connexion des utilisateurs.

On peut y retrouver une route d'inscription : `/register`

qui permet à un nouvel utilisateur de s'inscrire. Cette route cherche un utilisateur existant avec l'email (req.body.email) grâce à la méthode *findOne*. Si l'utilisateur existe, elle renvoie une erreur 403, sinon crée un nouvel utilisateur avec les données fournies. Ensuite elle sauvegarde le nouvel utilisateur dans la BDD, et envoie un statut 201 avec un message de succès. Pour finir en cas d'erreur, renvoie un statut 500 et affiche l'erreur dans la console.

On y trouve ensuite une route de connexion : `/login`

Qui permet à un utilisateur de se connecter.

Elle cherche un utilisateur avec l'email fourni (req.body.email). Si l'utilisateur n'existe pas, renvoie un statut 400 avec un message d'erreur. Vérifie également si le mot de passe est correct, si oui génère un token JWT et le renvoie.

La route pour récupérer tous les utilisateurs :

```
router.get('/all_users', async (req, res) => {  
  try {  
    const userList = await User.find({}, ["email", "role"]);  
    if (userList.length == 0) {  
      return res.status(200).json({ status: 200, message: "Pas  
d'utilisateurs trouvés" });  
    }  
    return res.status(200).json({ status: 200, result: userList });  
  } catch (error) {  
    res.sendStatus(500);  
    console.log(error);  
  }  
});
```

GET /all_users : Récupère tous les utilisateurs.

- Renvoie une liste d'utilisateurs avec leurs emails et rôles.

La route pour récupérer un utilisateur par son identifiant :

```
router.get('/user_by_id/:id', async (req, res) => {
  try {
    const user = await
    User.findById(req.params.id).select(["email", "role"]);
    if (!user) {
      return res.status(400).json({ status: 400, message: "Cet
utilisateur n'existe pas" });
    }
    return res.status(200).json({ status: 200, result: user });
  } catch (error) {
    res.sendStatus(500);
    console.log(error);
  }
});
```

GET /user_by_id/:id : Récupère un utilisateur par son identifiant.
=> Renvoie l'utilisateur avec son email et son rôle.

La route pour mettre à jour un utilisateur :

```
router.put('/user_update/:id', async (req, res) => {
  try {
    const updatedUserData = req.body;
    const user = await User.findById(req.params.id);
    if (!user) {
      return res.status(404).json({ status: 404, message:
"Utilisateur non trouvé" });
    }
    Object.assign(user, updatedUserData);
    const updatedUser = await user.save();
    return res.status(200).json({ status: 200, result:
updatedUser });
  } catch (error) {
```



```
return res.status(500).json({ status: 500, message: "Erreur lors  
de la mise à jour de l'utilisateur" });  
}  
});
```

ici on met à jour l'utilisateur et on l'enregistre.

La méthode **user.save** permet de sauvegarder un utilisateur dans la base de données avec persistance.

Par exemple pour la route user_update/:id:

-La route reçoit une requête HTTP de type put. Cette requête contient des données utilisateur au format JSON.

-Le contrôleur extrait les données utilisateur du corps de la requête (req.body) et les déséréalise (transformer un objet JSON en données exploitables par mon programme, ici en un objet Javascript).

Recherche de l'utilisateur : Le contrôleur utilise User.findById(req.params.id) pour rechercher l'utilisateur dans la base de données MongoDB par son identifiant. Si l'utilisateur est trouvé, les nouvelles données seront assignées à l'objet utilisateur à l'aide de la méthode Object.assign(user, updateUserData).

La méthode user.save est appelée pour sauvegarder les modifications dans la base de données MongoDB. Cela persiste (enregistre dans la couche physique) les changements et met à jour l'objet utilisateur dans la bdd.

