Luc SARRAZIN

BTS SIO 2ème année

Rapport de stage

Janin Consulting

Développeur Full-Stack :

Étapes de réalisation durant le stage pour créer un bot connecté à l'IA capable de simuler des entretiens professionnels pour s'entraîner

20/01/2025 au 14/03/2025

Sommaire

Introduction

Dans le cadre de ma formation en BTS Services Informatiques aux Organisations, spécialité Solutions Logicielles et Applications Métiers (SLAM), j'ai effectué un stage du 20 février au 14 mars 2025 au sein de l'entreprise Janin Consulting, afin de contribuer à la poursuite du développement du projet Ask.

Janin Consulting est une structure dirigée par l'auto-entrepreneur Anthony Janin, spécialisée dans la conception de logiciels informatiques personnalisés selon les besoins de ses clients. L'équipe était composée de quatre stagiaires, chacun en charge d'un module différent du projet. Deux d'entre eux travaillaient sur le Bot 1, une intelligence artificielle destinée à aider les utilisateurs à créer un CV de qualité. Ce bot analyse un document (PDF, Word, etc.) transmis par l'utilisateur, en extrait les informations clés, puis propose des améliorations concernant la structure, le contenu ou encore la formulation, afin d'obtenir un CV clair, cohérent et adapté aux attentes du marché.

Pour ma part, j'avais la charge du Bot 2, conçu pour offrir une simulation d'entretien interactive et personnalisée, en s'appuyant sur le CV validé par le Bot 1 et le contenu d'un appel d'offre. Mon rôle était de développer cette fonctionnalité de bout en bout, en prenant en charge à la fois le front-end avec Angular et le back-end avec Java Spring Boot.

Le bot génère des questions dynamiques réparties en trois catégories : techniques, comportementales, et expériences spécifiques. Il s'adapte aux réponses du candidat, propose des aides contextuelles en cas de blocage, et analyse automatiquement les réponses pour fournir un feedback structuré. Un rapport PDF final est ensuite généré, incluant un score, des recommandations personnalisées et des axes d'amélioration.

Ce travail s'intègre dans l'application Ask, une solution SaaS visant à améliorer la préparation des entretiens professionnels grâce à l'IA. L'objectif est de proposer aux candidats une expérience immersive, leur permettant de s'entraîner dans des conditions proches du réel. Ce projet m'a permis de renforcer mes compétences en développement full-stack, architecture logicielle, gestion de données et en intégration d'outils d'intelligence artificielle dans un contexte professionnel.

Développement de l’application

1. Développement Web (Front-End)

1.1 Conception du bot IA

Le Bot 2 a pour objectif de simuler un entretien interactif à partir du CV généré et validé par le Bot 1, ainsi que des exigences d’un appel d’offre. Il a été conçu pour offrir au candidat un environnement d’entraînement réaliste, lui permettant de se préparer efficacement aux attentes d’un recruteur.

Les questions générées par le bot sont catégorisées selon trois catégories :

Techniques : évaluation des compétences métiers.

Comportementales : analyse du savoir-être professionnel.

Expériences spécifiques : approfondissement des missions réalisées.

Le commercial pourra configurer différents types de questions pour le bot :

Choix multiple (checkbox) : plusieurs réponses possibles.

Choix unique (radio button) : une seule réponse correcte.

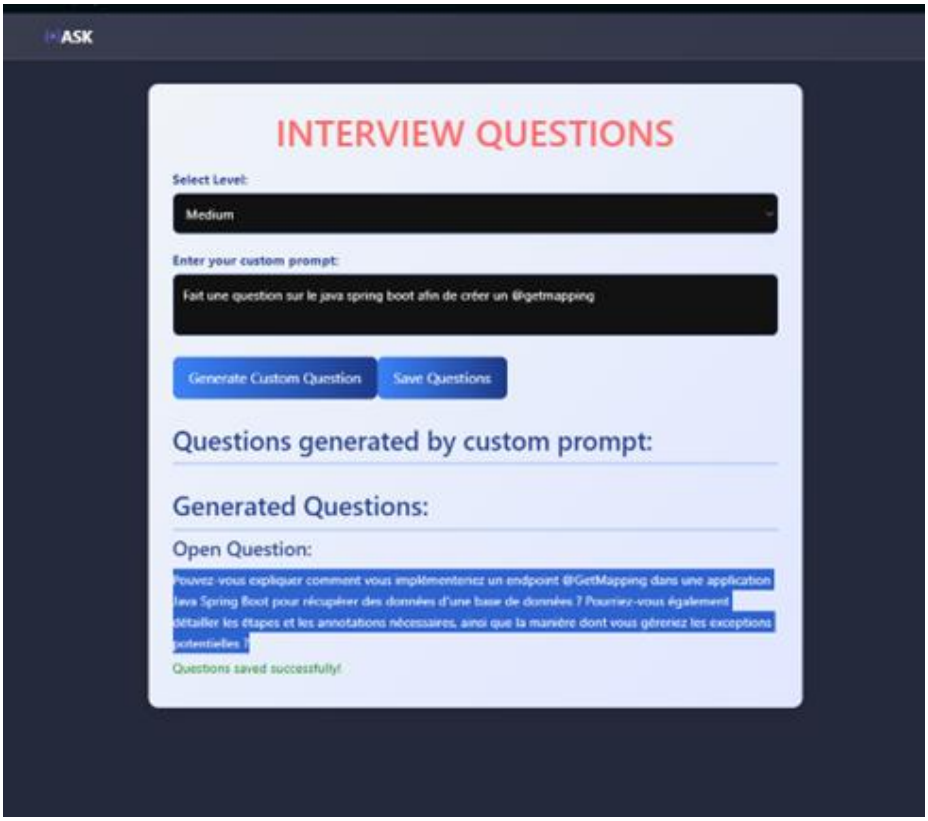
Questions ouvertes (textarea) : réponse libre du candidat.

Le bot ajustera les questions en fonction des réponses et proposera des aides en cas de blocage. Chaque réponse sera analysée pour aider le candidat à structurer son discours, avec un score et un rapport PDF final contenant des recommandations.

Voici une maquette de la page :

À la suite de notre maquette initiale, j’ai dû créer deux pages essentielles pour le bon fonctionnement du bot : Question Prompt et Job Interview.

La page Question Prompt : Sur cette page, le recruteur peut ajouter ou modifier des questions pour l'entretien, en fonction des exigences de l'appel d'offre. Il peut également choisir la difficulté de chaque question (facile, moyenne, difficile) afin d'adapter l'entretien au niveau du candidat.



La page Job Interview : Sur cette page, le candidat doit répondre aux questions posées par le recruteur ainsi qu'aux questions générées automatiquement par l'IA. À la fin de l'entretien, le candidat reçoit un rapport détaillé indiquant la qualité de ses réponses et un feedback sur sa performance, afin de pouvoir s'entraîner et s'améliorer.

1.2 Programmation des pages (Angular et HTML/CSS)

Pour créer les différentes pages, j'ai utilisé Visual Studio Code, ce qui m'a permis de travailler sur le HTML, le CSS et Angular (qui utilise TypeScript). Angular a été choisi par mon tuteur pour sa structure modulaire et ses fonctionnalités avancées.

Sur l'image ci-dessus, on retrouve la fonction qui permet d'évaluer les réponses de l'utilisateur à une question donnée.

```
evaluateResponses(): Promise<void> {
  return new Promise((resolve) => {
    const evaluationPromises: Promise<void>[] = this.Reponses.map((reponse) => {
      return new Promise<void>((innerResolve) => {
        this.apiService.evaluateResponse(reponse.question, reponse.reponse, reponse.type, reponse.choice, reponse.cv, reponse.offre).subscribe({
          next: (evaluation) => {
            console.log('Évaluation de la réponse...');
            console.log(evaluation);

            this.questions.push(evaluation);
            this.answers.push({
              question: reponse.question,
              response: reponse.reponse,
              score: evaluation.pass ? 1 : 0,
              feedback: evaluation.explanation,
              choice: reponse.choice
            });

            if (!evaluation.pass) {
              this.incorrectQuestions.push(reponse);
            }

            innerResolve();
          },
          error: (err) => {
            console.error('Erreur lors de l\'évaluation de la réponse:', err);
            innerResolve();
          }
        });
      });
    });

```

J'ai utilisé le framework Bootstrap, qui a facilité la création des pages et de l'esthétique, notamment grâce à ses composants prêts à l'emploi comme les icônes. De plus, il a permis une implémentation simplifiée pour afficher les résultats directement sur la page HTML.

```
<!-- Écran des questions -->
<div *ngIf="isInterviewStarted && !isInterviewEnded" id="question-screen" class="text-center">
  <div id="conversation" class="mb-3">
    <div *ngFor="let msg of conversation" [ngClass]="{'bot-message': msg.sender === 'bot', 'candidate-message': msg.sender === 'user'}">
      <div *ngIf="msg.sender === 'bot'" class="bot-message">
        <img src="https://media.istockphoto.com/id/1445426863/fr/vectoriel/concept-de-conception-de-logo-vectoriel-chat-bot.jpg?s=612x612&w=0&k=20&c=HMIHtjqCnZ0ajOCUrJnHlaAh8Fj1_ubPieZ1P
        <strong>Bot: </strong> {{ msg.question }}{{ msg.text }}
      </div>
      <div *ngIf="msg.sender === 'user'" class="candidate-message">
        <strong>Vous: </strong> {{ msg.text }}
        <img src="https://cdn-icons-png.flaticon.com/512/3135/3135715.png" alt="Vous" />
      </div>
    </div>
  </div>

  <div *ngIf="currentQuestion">
    <h3>{{ currentQuestion.text }}</h3>
```

L'implémentation a été compliquée, et j'ai dû chercher de l'aide dans les documentations, notamment en raison des difficultés rencontrées avec l'imbrication des directives NgIf et NgFor. Ces directives ne captaient pas correctement certaines données provenant du back-end, ce qui a nécessité un travail supplémentaire pour résoudre ces problèmes.

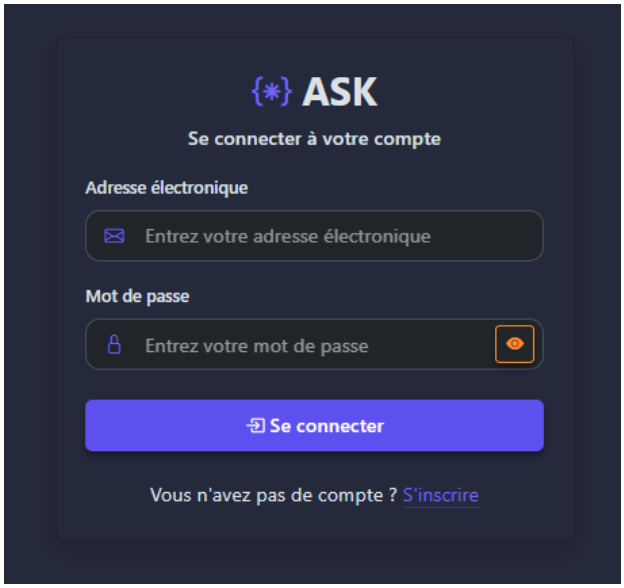
Ceci est la fonction dans l'API service permettant d'envoyer nos données vers le back-end afin de les recevoir et pouvoir renvoyer la réponse d'OpenAI :

```
public evaluateResponse(question: string, response: string, type: string, choice : string, resumeData: any, callOfTenders:any): Observable<any> {
  console.log('resumeData', resumeData, 'callOfTenders', callOfTenders);
  const params = new HttpParams()
    .set('question', question)
    .set('response', response)
    .set('type', type.toLowerCase())
    .set('cv', resumeData)
    .set('offre', callOfTenders)
    .set('choice', choice);
  const body ={
    resumeData,
    callOfTenders
  }

  return this.httpClient.post<any>(`${this.baseUrl}/evaluate`, body, { params },);
}
```

On retrouve donc dans cette méthode tous les paramètres nécessaires à l'évaluation d'une réponse. Une fois cette fonction implémentée, elle m'a servi de base pour les autres appels API. Ainsi, j'ai pu réutiliser la logique de construction des paramètres et l'envoi des requêtes HTTP pour d'autres fonctionnalités similaires.

J'ai également mis en place un design responsive en utilisant du CSS grâce aux classes et IDs, ce qui permet d'ajuster les éléments de la page pour le format téléphone. Les cours sur le CSS durant l'année m'ont permis d'acquérir des connaissances sur le responsive, ce qui m'a permis de les mettre en œuvre durant le stage.

Pour la page Modèle, j'avais prévu, avec le tuteur et le client, de rajouter un système de géolocalisation qui permettrait de proposer aux modèles des offres proches de leur localisation. Cependant, je n'ai pas pu continuer cette fonctionnalité, car elle n'était finalement pas prioritaire. J'ai quand même pu aborder cette option avec Angular/TypeScript en ajoutant un code qui récupère la localisation de l'utilisateur en demandant l'accès via une popup.Voici le système de d'authentification :



Ceci est le système qui me permet de me connecter à mon compte et d'accéder à mes pages, qui sont accessibles uniquement selon mon rôle. Par exemple, avec ce compte qui est un compte recruteur, j'ai accès aux pages RECRUITER.

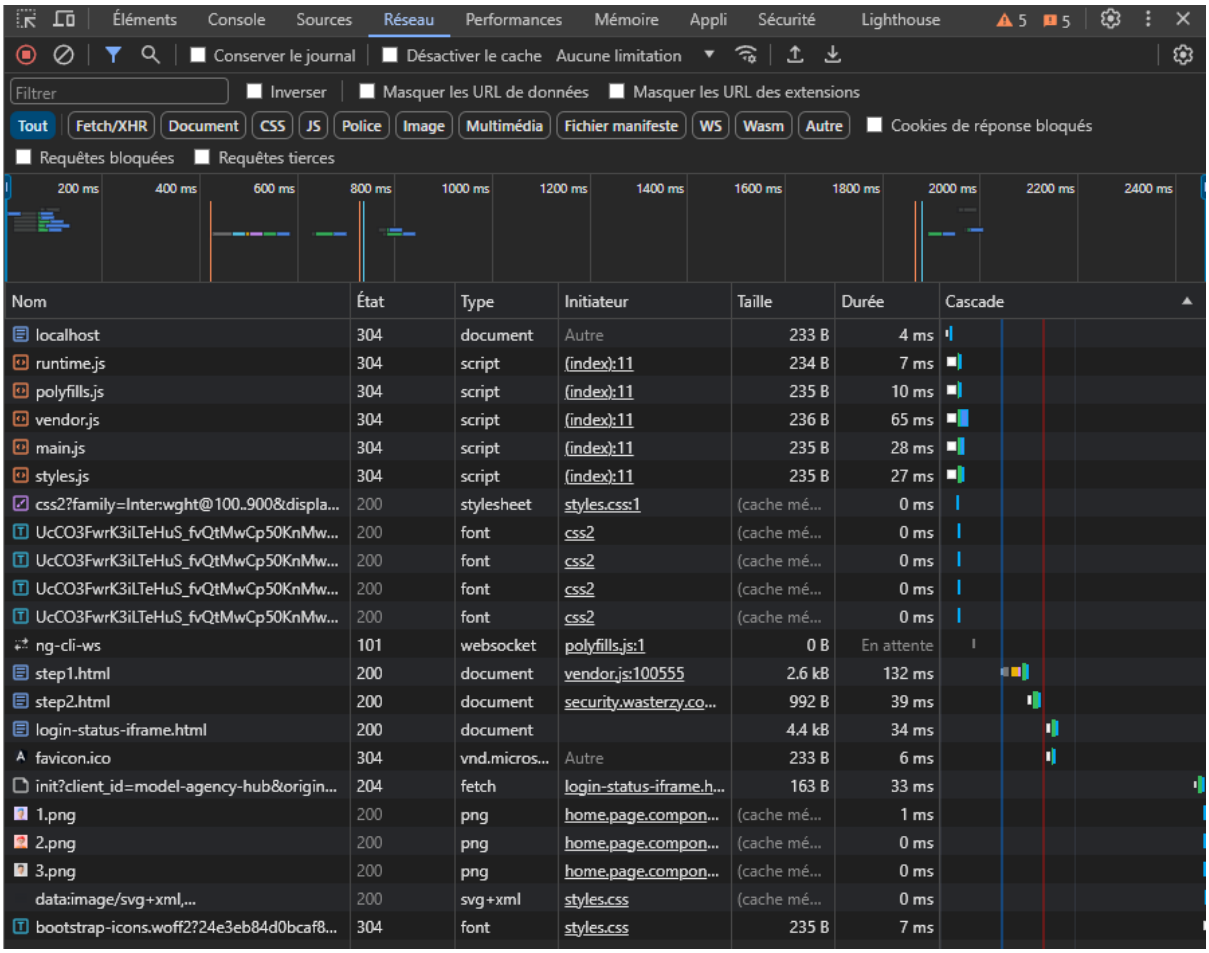
J'ai mis en place tout un système d'authentification et de redirection de pages intégré dans le programme.

```
const routes: Routes = [
  { path: '', component: HomePageComponent },
  { path: 'sign-in', component: SignInPageComponent },
  { path: 'sign-up', component: SignUpPageComponent },
  { path: 'interview', component: JobInterviewComponent, canActivate: [AuthGuard], data : {roles:  ['CANDIDATE', 'RECRUITER', 'ADMIN'] }},
  { path: 'PromptQuestion', component: QuestionPromptComponent, canActivate: [AuthGuard], data : {roles:  ['CANDIDATE', 'RECRUITER', 'ADMIN'] }},
  { path: 'interview2', component: InterviewComponent, canActivate: [AuthGuard], data : {roles:  ['CANDIDATE', 'RECRUITER', 'ADMIN'] }},

  { path: 'resumes', component: ResumesPageComponent, canActivate: [AuthGuard], data: { roles: ['CANDIDATE', 'RECRUITER', 'ADMIN'] } },
  { path: 'call-of-tenders', component: CallOfTendersPageComponent, canActivate: [AuthGuard], data: { roles: ['CANDIDATE', 'RECRUITER', 'ADMIN'] } },
  { path: 'reports', component: ReportsPageComponent, canActivate: [AuthGuard], data: { roles: ['CANDIDATE', 'RECRUITER', 'ADMIN'] } },
  { path: 'resume-template', component: ResumeTemplatePageComponent, canActivate: [AuthGuard], data: { roles: ['CANDIDATE', 'RECRUITER', 'ADMIN'] } },
  { path: 'profile', component: ProfilePageComponent, canActivate: [AuthGuard], data: { roles: ['CANDIDATE', 'RECRUITER', 'ADMIN'] } },
  { path: 'bot', component: BotPageComponent, canActivate: [AuthGuard], data: { roles: ['CANDIDATE', 'RECRUITER', 'ADMIN'] } },

  { path: 'vs-card', component: VsCardCandidatePageComponent, canActivate: [AuthGuard], data: { roles: ['CANDIDATE'] } },

  // otherwise redirect to landing
  { path: '**', redirectTo: '' }
];
```

Ainsi, pour chaque nouvelle page créée, il fallait définir une route en la reliant à AuthGuard. Cela me permet, lorsqu'on se connecte, d'atterrir sur la bonne page correspondant à mon rôle.

1.3 Débogage du Front-End

Le stage m'a appris à inspecter une page dans mon navigateur internet, ce qui m'a grandement aidé à identifier les problèmes et erreurs dans le code.



En allant dans l'onglet Réseau, j'ai accès à toutes les informations entrantes et sortantes du site internet, telles que les logos Bootstrap ou les requêtes de recherche d'utilisateurs.

J'ai également utilisé cet outil pour récupérer le token d'authentification, nécessaire pour obtenir l'identifiant de l'utilisateur connecté. Chaque utilisateur génère automatiquement un token sécurisé de manière aléatoire.

Dans l'onglet Console, je peux voir toutes les erreurs ou avertissements de programmation grâce à des fonctions comme console.log ou console.warning. Cela fonctionne un peu comme un System.out.println en Java, utilisé pour le débogage ou pour afficher des messages informatifs pour l'utilisateur.

Pour résoudre les différentes erreurs rencontrées, cette interface m'a permis de copier les messages d'erreur afin de faire des recherches sur internet ou de demander de l'aide à mon tuteur pour trouver des solutions.

J'ai aussi nettoyé le front pour une meilleure visibilité des erreurs.

2. Développement Java (Back-End)

2.1 Créations des Services et Controllers de l'API

Durant mon stage, j'ai dû créer une API pour gérer diverses fonctionnalités, telles que la génération de questions par l'IA (ouvertes, fermées et à choix multiples), l'évaluation des réponses, la sauvegarde des questions et la génération de questions en fonction de l'offre.

Pour cela, mon tuteur m'a fourni un code GitHub contenant le début d'une API basée sur Java Spring Boot. J'ai ensuite complété cette base en développant des contrôleurs (controllers) et des services afin d'intégrer pleinement ces fonctionnalités.

Exemple de controller :

```java
@PostMapping("question/open")
public OpenQuestionDto generateOpenQuestion(@RequestParam String theme, Level level) {
    OpenQuestion question = generator.generateOpenQuestion(theme, level);
    return questionMapper.toDto(question);
}
```

Exemple de service :

```java
@Override
public OpenQuestion generateOpenQuestion(String theme, Level level) {
    // Vérification des paramètres
    if (theme == null || level == null) {
        throw new IllegalArgumentException("Le thème et le niveau ne peuvent pas être null");
    }

    String systemPrompt = """
    Merci de générer une question ouverte d'entretien en suivant les directives ci-dessous.
    La question doit être pertinente par rapport au thème et au niveau spécifiés.
    Assurez-vous que la question soit claire et engageante pour le candidat sans dépasser un certain nombre de tokens.

    ~~~~~~
    Règles :
    - Ne pas dépasser 500 tokens dans votre réponse.
    - Pour toutes les questions tu dois t'en tenir au CV et à l'offre pour l'évaluer et si il l'a bien répondu ou pas et posé une question pour lui faire réussir la quest
    - Merci de ne pas répondre à ces phrase d'introduction juste à la réponse du candidat par rapport à la question.
    - Merci de répondre seulement par rapport au CV et à l'offre si ça parle d'autre chose tu arrête la conversation en disant que ça sort du concept.
    """;

    String userPrompt = String.format("Thème : %s\nNiveau : %s", theme, level);

    Map<String, Object> params = Map.of(
        k1: "model", v1: "gpt-4o-2024-08-06",  // Utiliser gpt-4 standard
        "messages", List.of(
            Map.of( k1: "role", v1: "system", k2: "content", systemPrompt),
            Map.of( k1: "role", v1: "user", k2: "content", userPrompt)
        ),
        k3: "temperature", v3: 0.7,
        k4: "max_tokens", v4: 500
    );

    // Créer un WebClient pour ajouter les headers nécessaires
    WebClient webClient = WebClient.builder()
        .baseUrl("https://api.openai.com/v1/chat/completions")
        .defaultHeader(HttpHeaders.CONTENT_TYPE, ...values: "application/json")
        .defaultHeader(HttpHeaders.AUTHORIZATION, ...values: "Bearer " + apiKey)
        .build();

    // Effectuer l'appel avec WebClient et récupérer la réponse
    Mono<String> responseMono = webClient.post() RequestBodyUriSpec
```

Ce code ci-dessus permet de générer une question ouverte d'entretien en envoyant une requête à l'API OpenAI. L'objectif est de récupérer une question pertinente basée sur un thème, un niveau, ainsi que les informations du CV et de l'offre d'emploi. La

question générée est ensuite renvoyée au format JSON pour être utilisée sur le front-end.

Vérification des paramètres : On s'assure que le thème et le niveau sont bien renseignés.

Création des instructions : On définit les règles que l'IA doit suivre pour générer une question adaptée au CV et à l'offre d'emploi.

Par exemple :

- La question doit être claire et en rapport avec le CV et l'offre d'emploi.
- Ne pas dépasser 500 tokens (pour gérer la taille et le temps de réaction de l'IA).

Envoi de la requête : On utilise WebClient pour interagir avec OpenAI et récupérer une réponse.

Traitement de la réponse : On extrait la question générée et on la convertit en objet OpenQuestion.

Affichage : La question est renvoyée au format JSON pour être affichée sur le front-end.

Ce processus automatise la création de questions d'entretien personnalisées en fonction du candidat.

Voici l'un des résultats qu'on reçoit dans la console :

Le contrôleur est l'un des éléments qui permet de communiquer entre le Front-End et le Back-End. Le service est la ligne de code qui sera exécutée lorsque le contrôleur sera activé. Par exemple, lorsque dans le contrôleur, generateOpenQuestion sera appelé par le Front-End, le contrôleur exécutera la fonction correspondante dans le service generateOpenQuestion et renverra la réponse au contrôleur, qui la transmettra ensuite au Front-End.

2.2 Débogage de l'API

Mon tuteur m'a enseigné l'art du débogage avec IntelliJ IDEA, qui dispose d'une fonctionnalité intéressante : les breakpoints. Ces derniers permettent de suspendre l'exécution du programme à des points spécifiques, afin de pouvoir avancer pas à pas dans le code.

La console affiche les erreurs ainsi que les succès, par exemple lors de la recherche d'un utilisateur..

```
Hibernate: select ce1_0.id,ce1_0.creation_date,ce1_0.description,ce1_0.logo_base_64,ce1_0.name,ce1_0.updating_date from user_company uce1_0 join company ce1_0 on ce1_0.id=uce1_0.company_id where uce1_0.user_id=?
Hibernate: select ue1_0.id,ue1_0.creation_date,ue1_0.email,ue1_0.email_verified,ue1_0.first_name,ue1_0.last_name,ue1_0.password,ue1_0.updating_date from `user` ue1_0 where ue1_0.email=?
Hibernate: select r1_0.user_id,r1_0.id,r1_0.creation_date,r1_0.role,r1_0.updating_date from user_role r1_0 where r1_0.user_id=?
```

3. Répartition du Travail

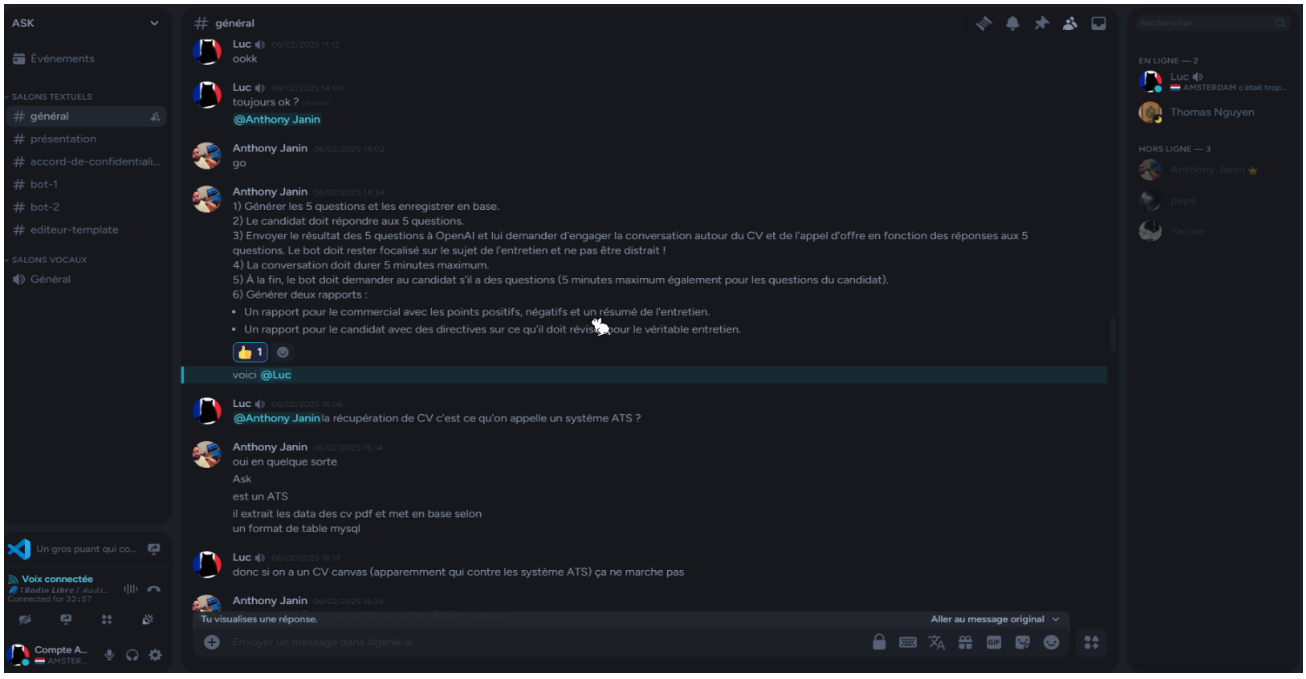
3.1 Tâches Assignées

Pendant mon stage, j'ai travaillé seul, ce qui m'a amené à gérer toutes les tâches de développement de manière autonome. Pour m'organiser, j'ai utilisé Visual Studio Code et IntelliJ IDEA afin de travailler efficacement sur les fichiers et modifier le code en fonction des besoins.

J'ai dû me concentrer sur plusieurs aspects du projet, notamment le développement de certaines pages et la récupération d'informations essentielles, comme l'ID de l'offre d'emploi et du CV. Travailler seul m'a permis de renforcer ma capacité à résoudre les problèmes de manière indépendante et à structurer mon travail de façon optimale.

Pour faciliter la communication avec mon tuteur et obtenir de l'aide en cas de besoin, j'ai utilisé l'application Discord. Un serveur dédié, mis à disposition par le tuteur, m'a permis d'accéder aux ressources du projet, notamment une présentation du réseau social et de ses différentes fonctionnalités. De plus, le tuteur m'a fourni quelques informations pour m'aider à mieux produire le produit final.

Voici une image du serveur discord :



Pendant mon stage, j'ai développé deux pages et j'ai également dû modifier certaines pages afin de récupérer des informations, comme l'ID de l'offre d'emploi et du CV.

Le seul inconvénient était que les fichiers étaient stockés sur mon ordinateur, ce qui empêchait mon tuteur d'y accéder, notamment pour tester le programme, à moins que je sois présent. Par exemple, la base de données SQL était différente, ce qui m'obligeait à synchroniser constamment mes fichiers. Cela nécessitait une rigueur dans le partage des fichiers, heureusement, j'avais accès à un GitHub partagé par mon tuteur, ce qui m'a permis d'être plus efficace et mieux organisé pour partager les

fichiers, y compris les dossiers SQL, plus facilement. Une solution encore plus efficace aurait été d'utiliser un serveur distant ou une base de données centralisée pour que tous les membres du projet, y compris mon tuteur, aient un accès direct aux fichiers et aux tests sans dépendre d'un stockage local.

Conclusion

Ce stage de deux mois m'a offert une immersion précieuse dans le monde professionnel, notamment dans les aspects variés d'un projet. Travailler dans un environnement discipliné a renforcé ma maturité et ma responsabilité, des traits essentiels pour évoluer dans le domaine du développement de jeux vidéo.

Pendant cette période, j'ai exploré divers langages de programmation tels que HTML, CSS, Angular, Java Spring Boot, MySQL, Bootstrap. J'ai eu l'opportunité de mettre en pratique les connaissances et compétences acquises au cours de mes études en programmation. Ce projet m'a également initié à la création de sites web interactifs et à la gestion de bases de données ainsi que l'utilisation de l'IA dans un projet professionnel, des compétences cruciales dans le développement de jeux.

D'autre part, j'ai acquis de nouvelles compétences techniques comme le débogage avancé, la gestion d'API, la manipulation des tokens, et l'utilisation efficace de GitHub pour le versionnage de code. Cette expérience m'a également appris à rechercher activement des solutions aux problèmes techniques, renforçant ainsi mon autonomie et ma capacité d'adaptation.

Travailler seul sur ce projet m'a permis de mieux structurer mon travail et de gérer l'ensemble du développement de manière indépendante. J'ai dû organiser mes tâches efficacement pour atteindre mes objectifs dans les délais impartis.

Enfin, cette expérience a consolidé mon ambition de devenir programmeur de jeux vidéo. Je suis reconnaissant envers mon tuteur Monsieur Janin pour son soutien et ses conseils précieux tout au long de ce stage enrichissant.

Annexe

Logiciels utilisés

- Visual studio Code
- IntelliJ Idea
- XAMPP

Langage de programmation utilisés

- HTML/CSS
- Java Spring BOOT
- Angular (Typescript)
- MySQL

Framework utilisés

- Bootstrap

Compétences Acquises

- Autonomie
- Recherche de solutions
- Organisation
- Travail d'équipe
- Adaptabilité

Liens utiles (Documentation) :

- Bootstrap Docs :https://getbootstrap.com/docs/5.3/getting-started/introduction/
- Bootstrap Icon : https://icons.getbootstrap.com/
- UUID in Java Spring Boot: https://www.baeldung.com/java-uuid