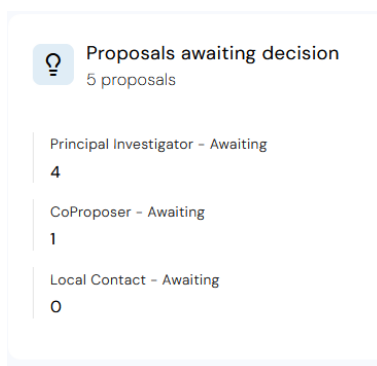




DÉVELOPPEMENT D'UNE TUILE DANS L'APPLICATION DASHBOARD

STAGE EFFECTUÉ DU 28/04/2025 AU 13/06/2025

RAPPORT DE STAGE



Maître de stage :

HARDING Richard

hardingr@ill.fr

Stagiaire :

GELINEAU Arthur

arthur.gelineau@etu.toulouse-inp.fr

Organisme d'accueil :

Institut Laue-Langevin

Grenoble, 38000

Table des matières

I	Préambule	3
A	Remerciements	3
B	Résumés	3
1	Abstract	3
2	Résumé	3
C	Introduction	4
II	Contexte, missions et réflexions sur le stage	5
A	Découverte de l'ILL	5
B	Contexte, missions et résultats du stage	6
1	Contexte nécessaire pour comprendre	6
2	La mission	11
3	Les résultats	22
C	Responsabilités environnementales et sociétales dans le cadre du stage	23
1	Impact énergétique	23
2	Approche humaine et sociétale du travail à l'ILL	24
D	Analyse réflexive du stage	24
III	Conclusion	25
IV	Annexes	26
1	Liste des tâches effectuées	26
2	Détail des domaines les plus étudiés à l'ILL	26

3	Les pays membres de l'ILL	27
4	Soumettre une proposal via le User Club	27
5	Le National Balance	28
6	Les rôles du User Office et des Secrétaires de collège	29
7	Le Dev-proxy	29
8	Illustration de la BDD du microservice proposal-service	30
9	Liste de tous les logiciels installés	30
10	Git	31
11	TypeScript et Angular	32
12	Les critères définissant une API REST	33
13	Les différentes requêtes HTTP	34
14	Hibernate	35
15	Les annotations Java	35
16	Lancer l'application Dashboard en local	38
17	L'utilisation de l'IA éthique durant le stage	39

I Préambule

A Remerciements

Je remercie chaleureusement mon tuteur de stage Richard HARDING ainsi que Cédric ORTIZ, qui m’ont donné cette opportunité de stage, de multiples conseils et connaissances tout du long de mon stage.

Je remercie également toute l’équipe développement du service informatique de l’Institut Laue-Langevin (ILL), notamment Ludovic LEROUX, chef de groupe, qui m’a accueilli, intégré et suivi durant la totalité de mon stage.

Enfin je remercie toutes les personnes travaillant à l’ILL avec lesquelles j’ai pu collaborer, discuter et qui m’ont permis d’avoir cette expérience professionnelle enrichissante.

B Résumés

1 Abstract

This internship, carried out at the Institut Laue-Langevin (ILL) in Grenoble from April 28 to June 13, 2025, was part of my academic course at Toulouse INP – Prépa T². The ILL is a scientific research center which welcomes many users each year for experiments. The main objective of this project was to design and implement an interactive tile within an inhouse developed dashboard application for visiting scientists. Throughout the internship, I worked with modern technologies such as Angular, TypeScript, Java, Quarkus, Docker, and Kubernetes, gaining insight into real-world software development workflows. This experience helped me strengthen my technical and collaborative skills, and gave me a deeper understanding of software engineering in a research environment, even though I may pursue a different path within the field of computer science.

2 Résumé

Ce stage, réalisé à l’Institut Laue-Langevin (ILL) à Grenoble du 28 avril au 13 juin 2025, s’inscrit dans le cadre de ma formation à Toulouse INP – Prépa T². L’ILL est un institut de recherche qui

accueil des utilisateurs afin qu'ils puissent y mener leurs expériences scientifiques. Ma mission principale consistait à développer une tuile interactive dans l'application Dashboard, destinée à l'affichage synthétique de propositions scientifiques. Ce travail m'a amené à utiliser des technologies telles qu'Angular, TypeScript, Java, Quarkus, Docker et Kubernetes, dans un environnement de développement professionnel et collaboratif. Ce stage m'a permis de renforcer mes compétences techniques, de mieux comprendre les méthodologies de développement en entreprise, et de confirmer mon intérêt pour le secteur de l'informatique, même si je ne m'oriente pas spécifiquement vers le métier de développeur.

C Introduction

Dans le cadre de ma formation, j'ai réalisé un stage à l'Institut Laue-Langevin (ILL), centre de recherche international situé à Grenoble, spécialisé dans les sciences et technologies neutroniques. Chaque année, l'ILL accueille des équipes scientifiques du monde entier pour y mener des expériences. Celles-ci nécessitent du personnel qualifié (scientifiques, techniciens, etc.) et des instruments adaptés aux besoins des utilisateurs. Tout commence par la soumission d'un projet scientifique, qui sera accepté ou non par l'ILL. Dans la suite du rapport, ces soumissions seront appelées des proposals.

Mon stage s'est inscrit dans ce contexte : il a porté sur l'amélioration de la plateforme numérique, nommée Dashboard, en particulier sur le suivi des proposals en attente de filtrage. En tant que développeur IT, j'ai été chargé de développer une tuile permettant de visualiser ces proposals dans l'interface.

L'objectif du stage était donc de faciliter la visualisation de l'état des proposals pour l'utilisateur, en automatisant leur affichage et leur état dans le cycle de vie du processus.

Ce rapport présente dans un premier temps l'ILL, puis il pose un contexte plus global pour le stage, avant de détailler les différentes missions qui m'ont été confiées ainsi que les résultats obtenus. Nous finaliserons avec les différentes analyses environnementales, sociétales et réflexives axés sur le stage avant de conclure.

II Contexte, missions et réflexions sur le stage

A Découverte de l'ILL

L'ILL est un institut de recherche international, qui est un des leaders dans les sciences et les technologies neutroniques. Plus exactement, c'est ce qui s'appelle un institut de service, c'est-à-dire que l'ILL met à disposition ses infrastructures, son personnel et ses instruments de pointe pour offrir à plus de 1400 chercheurs et chercheuses, des faisceaux de neutrons de très grande qualité. L'ILL dénombre ainsi environ 1000 expériences réalisées chaque année.

L'ILL réalise des expériences dans un large panel de domaines. En effet, 60% de la recherche est dédiée aux sciences fondamentales, dans des domaines tels que la physique de la matière molle, la chimie, la biologie, la physique nucléaire et la science des matériaux. Les 40% restants sont dédiés aux enjeux sociétaux, dans des domaines tels que la santé, l'énergie, l'environnement et les technologies de l'information. Nous pouvons voir un détail de tous les domaines, avec la figure en annexes. En plus de travailler avec des scientifiques de plus de 40 pays différents, l'ILL collabore aussi avec des entreprises industrielles [19].

Fondé à Grenoble en 1967, sous l'impulsion des gouvernements français et allemands, l'ILL tire son nom des scientifiques Max von Laue et Paul Langevin, respectivement allemand et français, dans le but de les honorer pour leurs contributions scientifiques mais aussi sociétales [14]. Ensuite rejoints par le Royaume-Uni, ces trois pays forment les Associés de l'ILL et contribuent à hauteur de 75% du budget annuel. Dix autres pays contribuent actuellement aussi au budget de l'ILL à hauteur de 20%. Enfin, les 5% restants sont assurés par des fonds propres [13].

À l'heure actuelle, l'ILL restera actif jusqu'en 2033. Bien qu'une nouvelle source de neutrons soit en construction en Suède, l'ILL se dit prêt à rester compétitif sur la scène européenne et internationale [16].

B Contexte, missions et résultats du stage

1 Contexte nécessaire pour comprendre

1.1 Comprendre les proposals

Comme expliqué en introduction, dans le cadre de mon stage, j'ai dû implémenter une tuile dans le Dashboard, reprenant les proposals qui n'ont pas encore subi un processus de sélection. Cette notion de proposals est un point central du stage et il est important pour bien comprendre la suite du rapport que nous nous penchions un peu plus sur les différents types de proposals, leur cycle de vie ainsi que les différents acteurs impliqués.

Pour toutes les proposals, nous pouvons retrouver trois acteurs majeurs. Ces rôles permettent de classer correctement les différentes proposals dans plusieurs tuiles du Dashboard, ce qui est primordial pour les utilisateurs. Il est donc intéressant de comprendre à quoi ils correspondent. Nous retrouvons un « principal investigator », également appelé « main Proposer », qui est la personne en charge du projet scientifique. Nous retrouvons ensuite plusieurs « coproposers » qui sont donc les scientifiques faisant partie du projet. Enfin, il y a les « local contacts », scientifiques travaillant à l'ILL qui sont chargés du bon déroulement de l'expérience à laquelle ils sont affectés. Voici ci-après l'exemple d'une tuile, développée par l'équipe avant mon arrivée, qui liste les proposals en fonction du rôle de l'utilisateur.

Maintenant que nous avons étudié les différents acteurs, étudions quelques types de proposals :

- Les proposals EPS : Ce sont les proposals les plus courantes. Elles suivent un processus de sélection bien établi, détaillé dans les lignes ci-dessous.
- Les proposals EASY : Celles-ci nécessitent un temps de faisceau assez court et, comme leur nom l'indique, elles sont facilement réalisables. Ainsi, le local contact peut très souvent effectuer seul l'expérience, sans présence d'utilisateurs extérieurs.
- Les proposals DDT : Types de proposals où la Direction de l'institut alloue de manière discrétionnaire du temps de faisceau pour un projet scientifique. Elles ont donc la spécificité de ne passer aucun filtre contrairement aux proposals EPS. Ce sont en grande majorité des

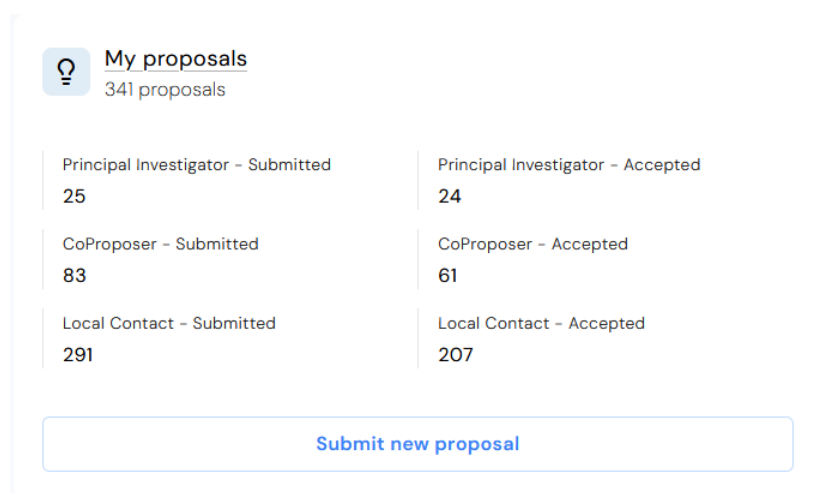


FIGURE II.1 – Capture d’écran d’une tuile du Dashboard, démontrant l’importance des rôles.

projets à très fort enjeux scientifique, qui ont par le même occasion la possibilité d’apporter une grande renommée à l’ILL. Par exemple, lors de la période du COVID-19, ce type de proposals a régulièrement été mis en place.

- Les proposals CRG : Les pays membres financent certains instruments. Il leur est donc accordé du temps de faisceau proportionnel à la hauteur du financement, qu’ils pourront organiser comme ils le souhaitent, sur ces instruments.
- Les proposals INDU : Des industriels peuvent payer pour avoir du temps de faisceau.

S’attarder sur le cycle de vie des proposals de type EPS est important. Pendant l’année, les scientifiques peuvent soumettre des proposals avant une date limite (une en février et l’autre en septembre). Pour soumettre une proposal, ils passent par une application nommée User Club, dans lequel ils renseignent un certain nombre d’informations tels que le collège, une sous-catégorie à ce collège, les instruments sur lesquels ils veulent faire leurs expériences, l’ensemble des membres de la proposal et leurs rôles respectifs, un abstract et bien d’autres encore. Le lecteur peut trouver plus de détails en annexes.

Une fois l’échéance passée, l’ensemble des proposals recueillies sont filtrées une première fois par le User Office et les secrétaires de collège. Ce premier filtrage a pour but de supprimer notamment les doublons ou des « bad proposals », donc des proposals ne répondant pas aux critères de bases pour être prises en considération. Ce sont des proposals qui sont aberrantes ou parfois des proposals

effectuées dans le but de tester le service. Si la proposal passe cette première sélection, elle obtient le statut de proposal submitted. Cela ne signifie pas pour autant que la proposal soit acceptée. Si vous avez été attentifs, sur la capture d'écran ci-dessus, nous pouvons apercevoir l'état de la proposal à la droite du rôle.

Toutes les proposals ainsi conservées subissent alors un processus de sélection appelé « review ». Celui-ci se déroule en deux phases effectuées en parallèle. La première phase, interne à l'ILL, étudie la faisabilité technique de l'expérience. Durant la seconde phase, un comité de chercheurs internes et externes à l'ILL examine la qualité de la proposition scientifique. C'est la Peer Review. À l'issue de la Peer Review, la proposal reçoit une note (grading), soit A, B ou C. Les proposals classées en A sont presque assurées d'obtenir du temps de faisceau, tandis que celles en C sont presque assurées de ne pas en recevoir. La peer review va permettre également d'allouer du temps de faisceau aux meilleures proposals. Cependant, à ce stade-là, une proposal avec du temps de faisceau alloué n'est pas encore considérée comme acceptée. En effet, une dernière étape appelée « National Balance », peut venir corriger le temps de faisceau alloué précédemment.

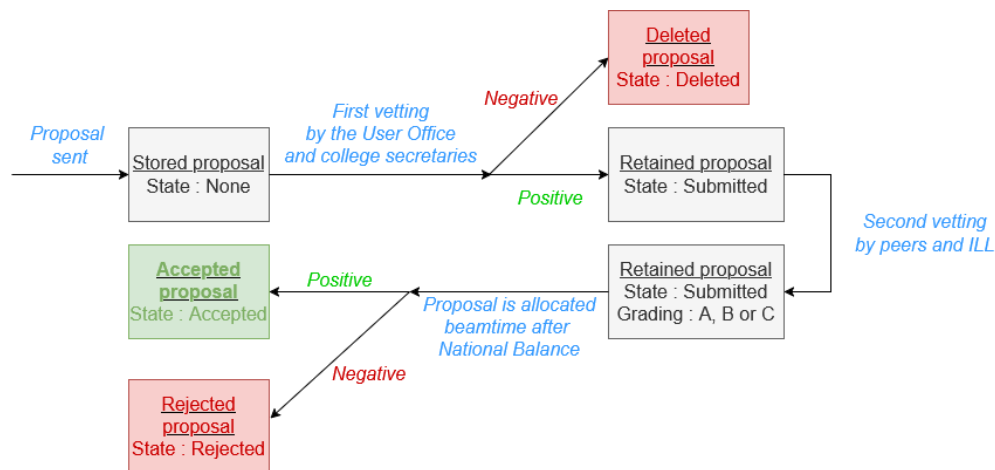


FIGURE II.2 – Schéma du cycle de vie d'une proposal EPS.
[1]

À l'issue de l'expérience, le groupe de recherche doit soumettre un rapport ou une publication. Le lien qui existe entre une publication parue dans une revue scientifique et une proposal permet à l'ILL d'obtenir de la crédibilité et de la renommée.

1.2 Comprendre l'architecture du projet

Maintenant que nous avons compris les objets sur lesquels nous travaillons, il nous faut comprendre l'architecture du projet. Le projet est défini selon plusieurs couches, chacune d'entre elles ayant une responsabilité bien définie. Nous pouvons le diviser en 2 ou 3 parties : une partie applicative, des Microservices et un proxy (présent uniquement si nous travaillons en local).

La partie applicative est divisée en deux sous-catégories : le frontend et le backend. La partie frontend gère toute la partie interface utilisateur (UI), c'est-à-dire la partie visible avec laquelle l'utilisateur interagit. Il gère donc l'expérience utilisateur en affichant les données, en capturant les actions des utilisateurs et en appelant le backend. Le backend quant à lui gère toute la logique métier, c'est-à-dire l'ensemble des règles et processus qui régissent les opérations de l'application. Il gère donc l'accès aux données, la sécurité, etc...

Quand un utilisateur sollicite une page au travers d'une requête HTTP, cette dernière est transmise à la partie backend, donc au Dashboard-server. Celui-ci va traiter la demande soit en interne soit en appelant des services. Il peut faire appel à une base de données (BDD). Celle-ci gère toutes les requêtes associées aux préférences utilisateurs. Mais il peut aussi faire appel à un autre service, présent dans les Microservices.

Le Dev-proxy est un système utilisé au sein de l'ILL lors du développement local. Il s'agit d'un proxy, c'est-à-dire un composant intermédiaire entre deux hôtes, qui a pour rôle ici de faciliter les échanges. Si nous travaillons en local, alors il y a une possibilité qu'il soit utilisé, tandis que si nous ne travaillons pas en local, il n'intervient tout simplement pas. Le Dev-proxy est détaillé en annexes, il est cependant conseillé de lire l'entièreté du rapport avant pour une meilleure compréhension. Si aucun problème ne survient, le Dashboard-server transmet la requête vers les Microservices, qu'elle passe par le proxy ou non.

Les Microservices sont des services autonomes et découplés de l'application principale qui traitent un ensemble de fonctions spécifiques. L'utilisation des microservices offrent de nombreux avantages. Ils permettent notamment de mieux organiser le code. Le deuxième avantage est d'empêcher l'application, notamment la partie cliente, d'accéder directement à la BDD. Cela augmente la

sécurité de l'application. Enfin, le dernier avantage est de pouvoir être réutilisable par d'autres applications, sans avoir à réimplémenter à chaque nouvelle application des codes fastidieux et très similaires. L'utilisation des microservices s'oppose directement à l'architecture monolithique, c'est-à-dire où toute la logique métier est inscrite dans un unique bloc, associé à une unique application.

Dans le projet, les Microservices sont rangés par thèmes : Proposal (gère tout ce qui est lié à une proposal), Identity (gère tout ce qui touche aux utilisateur), Stay (gère tout ce qui est lié au séjour d'un utilisateur) et Publications (gère tout ce qui est lié aux publications d'un utilisateur). Comme vous pouvez vous en douter, le microservice sur lequel je vais travailler est Proposal, mais nous en reparlerons dans la partie plus détaillée des missions de stage. Enfin, chaque microservice possède sa propre BDD.

Le schéma ci-dessous offre un visuel pour mieux se représenter l'architecture du projet en local. Elle sera par la suite étoffée.

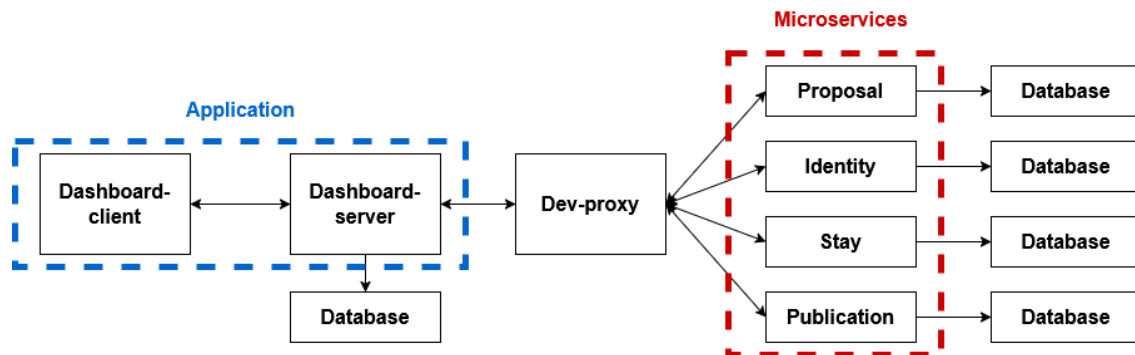


FIGURE II.3 – Schéma de l'architecture simplifié du projet Dashboard en local.
[2]

1.3 Le lien entre les deux

Comme expliqué dans la partie précédente, il nous faut détailler un peu la BDD du microservice Proposal. La BDD possède en tout 25 tables. Quand l'utilisateur soumet un projet scientifique, nous avons vu qu'il soumet un nombre important d'informations, tels que les membres, des environnements spécifiques ou les échantillons utilisés par exemple. Toutes ces informations sont stockées dans des tables temporaires commençant par le préfixe temp (Voir figure IV.4). D'après le processus décrit ci-dessus, quand la proposal obtient le statut de submitted, toutes les informations associés à

celle-ci sont transférées dans les tables portant le même nom sans le préfixe temp. Par la suite, le processus décrit plus tôt est appliqué.

Le lecteur a maintenant une vision plus globale du projet, ce qui va nous permettre de plonger un peu plus dans l'aspect technique.

2 La mission

2.1 Les logiciels indispensables

Chronologiquement, comprendre ce qui a été expliqué dans la dernière section n'a pas été ma première tâche assignée. Quand je suis arrivé le 28 avril 2025, ma première mission a été de configurer mon environnement de travail. Cela signifie à la fois d'installer tous les logiciels indispensables comme Git, Docker et Quarkus mais aussi de récupérer le code source du projet. Le lecteur peut trouver la liste complète de tous les logiciels installés en annexes. De plus, récupérer le code source sur sa machine est important car elle permet de travailler en local sur ses propres branches, sans prendre le risque de « casser » le code déjà existant. Pour cela, il faut utiliser le logiciel Git qui discute avec le dépôt distant GitLab. Git est un VCS, c'est-à-dire un logiciel permettant de faire de la gestion de versions de fichiers. La principale force de Git est de pouvoir travailler sur différentes branches en même temps et de pouvoir les fusionner plus tard [10]. Quand nous travaillons comme cela, nos fichiers peuvent avoir différents états. Le lecteur peut en apprendre plus sur les états et les commandes Git en annexes.

Docker est un logiciel de conteneurisation, c'est-à-dire qu'il permet de gérer des applications dans un environnement isolé qui se nomme un conteneur. Un conteneur Docker est un environnement d'exécution contenant tous les composants nécessaires, tels que le code, les dépendances et les bibliothèques nécessaires pour exécuter le code de l'application sans utiliser les dépendances de la machine hôte. De plus, ils permettent aux développeurs d'empaqueter des logiciels à exécuter sur n'importe quel système d'exploitation cible. Vous comprenez donc la force de Docker. Enfin, Docker est très flexible et aide grandement les développeurs pour la gestion de code [6]. En effet, c'est un excellent outil pour assurer l'intégration continue (CI) et le déploiement/livraison continu (CD) d'une application. Pour comprendre le CI et le CD, il faut comprendre les différents

environnements de développement d'une application telle que le Dashboard. Le développeur, par exemple moi-même, travaille sur une branche personnelle. Il fusionne, via les commandes git, ces changements dans la branche nommée develop. À ce moment-là, l'application est compilée, testée et son image docker est générée (c'est ce qui permet d'exécuter l'application dans un conteneur). Cette étape est l'intégration continue. L'étape du déploiement continu consiste à déployer le code précédemment construit lors de la CI, vers l'environnement de développement. Le code est ensuite déployé en preprod et prod quand la branche develop est fusionnée dans celle de master. C'est le CD.

Quarkus est un framework Java, conçu pour le développement d'applications cloud-native, notamment les Microservices. Il permet de construire des Microservices rapides au démarrage, légers en mémoire et bien intégrés à Kubernetes.

Kubernetes, dont le raccourci est K8s, est l'outil qui déploie, gère et met à l'échelle l'ensemble des conteneurs Docker. K8s assure la communication entre les différents services [12]. Ce n'est pas un logiciel que j'ai moi-même installé, mais il reste important pour la structure globale du projet.

Nous reviendrons dans la sous-section « Les résultats », sur la manière dont tous ces outils co-existent pour ne former qu'une unité fonctionnelle, soit l'application Dashboard.

2.2 Dashboard-client

Après avoir installé l'environnement de travail et après avoir légèrement pris connaissance de l'utilité de chaque logiciel, je me suis lancé dans la compréhension du frontend. J'avais auparavant déjà quelques connaissances en HTML, CSS et JavaScript, cela m'a permis de comprendre plus facilement les codes récupérés. La partie client se base sur Angular, qui est un framework javascript permettant de développer des SPA. Il s'appuie sur les langages de programmation HTML, CSS et TypeScript. L'une des forces d'Angular est la réutilisabilité de ces composants. Un component peut être vu comme le principal objet permettant la création d'applications Angular et il se divise en trois unités : un fichier HTML qui gère l'affichage du component dans le navigateur, un fichier CSS qui gère son style et enfin un fichier TypeScript qui gère son comportement. J'ai donc dû passer par une première étape qui a été de me mettre à niveau sur TypeScript et sur l'environnement Angular. Le lecteur peut en apprendre plus sur TypeScript ainsi que sur Angular en annexes.

Dans le dossier `components/` (celui qui gère les composants de l'application), via la commande **ng generate component 'user-temporary-proposals'**, j'ai créé mon composant Angular sur lequel j'ai travaillé, nommé `user-temporary-proposals`. Comme expliqué précédemment, cela a donc automatiquement créé trois fichiers : `user-temporary-proposals.component.html`, `user-temporary-proposals.component.scss` et `user-temporary-proposals.component.ts`. En m'inspirant du code de cette tuile, j'ai développé ma propre tuile. Dans la suite, nous nous référerons beaucoup à la tuile finalisée.

Dans le but de vous immerger dans la partie pratique de mon stage, je vous montrerai différentes lignes de code que j'ai développées. Il est bien entendu impossible que je vous présente l'ensemble des fichiers que j'ai codés. Je ne me concentrerai donc que sur les parties les plus intéressantes. Commençons par étudier le fichier HTML :

```
1 <app-menu-card>
2
3 <app-card-header
4   header
5   title="{{ 'user-temporary-proposals.title' | translate }}"
6   icon="icon-lightbulb"
7   [subtitle]="
8     allProposalsCount() !== undefined
9     ? ('user-proposals.subtitle' | translate: { count: allProposalsCount() })
10    : undefined
11 "
12 />
13
14 <div content class="user-temporary-proposals-wrapper">
15   <div class="user-temporary-proposals-content">
16     <app-clickable-labeled-value
17       (clicked)="navigateToProposals(ProposalMemberRole.MAIN_PROPOSER, null)"
18       [value]="mainProposerCount()?.toString() || undefined"
19       label="{{ 'user-temporary-proposals.pi-suggested' | translate }}"
20     />
```

Listing II.1 – Partie de code du fichier `user-temporary-proposals.component.html`

Le sélecteur `< app - menu - card >` permet d'injecter dans mon composant (`user-temporary-proposals`) le contenu HTML du composant possédant ce sélecteur, avec son style et ses comportements spécifiques. Les lignes 3 à 12, permettent de créer l'en-tête de ma tuile. Sur la tuile finale, elle possède un titre (`Proposals awaiting decision`), une icône et un nombre de total de proposals

(5) pour cet utilisateur. Les lignes 14 à 19 permettent de créer la première ligne du corps de la tuile, celle qui récupère toutes les propositions avec le statut Principal Investigator. Celle-ci possède une valeur, qui est un nombre. Dans le cas de la tuile finale, cet utilisateur en possède 4. De plus, quand nous cliquons sur cette ligne, cela nous renvoie vers une table contenant toutes les propositions submitted ou acceptées. J'ai construit de la même manière les deux autres lignes, en adaptant les libellés.

Le fichier SCSS n'est qu'une extension d'un fichier CSS.

Le fichier TypeScript est quant à lui bien plus intéressant, décortiquons le :

```
1 // Create a component with its properties
2 @Component({
3   selector: 'app-user-temporary-proposals',
4   imports: [
5     [Button, Message, PersonalInfoBlockComponent, ToggleSwitch, TranslatePipe, TypographyComponent]
6       as const,
7     AgCharts,
8     CardHeaderComponent,
9     ClickableLabeledValueComponent,
10    MenuCardComponent,
11  ],
12   templateUrl: './user-temporary-proposals.component.html', // Link both TypeScript and HTML files
13   styleUrls: ['./user-temporary-proposals.component.scss'], // Link both TypeScript and SCSS files
14   changeDetection: ChangeDetectionStrategy.OnPush,
15 })
16 export class UserTemporaryProposalsComponent {
17
18   protected readonly ProposalMemberRole = ProposalMemberRole; // Initialize user's role
19
20   protected readonly mainProposerCount: WritableSignal<number> = signal<number>(undefined); //
21     Initialize the value for mainProposerCount type proposals
22   protected readonly proposerCount: WritableSignal<number> = signal<number>(undefined);
23   protected readonly localContactCount: WritableSignal<number> = signal<number>(undefined);
24   protected readonly allProposalsCount: WritableSignal<number> = signal<number>(undefined);
25
26   // The constructor indicates the dependencies
27   constructor(
28     private readonly _router: Router,
29     private readonly _accountTempProposalService: AccountTempProposalService,
30   ) {}
31
32   // Initialize the fetchProposals method
```

```

32 ngOnInit(): void {
33     this.fetchProposals();
34 }

```

Listing II.2 – Partie de code du fichier user-temporary-proposals.component.ts

Dans la partie **class UserTemporaryProposalComponent**, nous y initialisons différentes variables que nous utiliserons dans la méthode nommée `fetchProposals`. C’est cette méthode qui permet de réellement initialiser les valeurs clés de notre component Angular.

```

1  fetchProposals(): void {
2  // Initialize variables, used as filters
3  const mainProposerFilter = {
4      memberRoles: [ProposalMemberRole.MAIN_PROPOSER],
5  };
6  const proposerFilter = {
7      memberRoles: [ProposalMemberRole.PROPOSER],
8  };
9  const localContactFilter = {
10     memberRoles: [ProposalMemberRole.LOCAL_CONTACT],
11 };
12
13 // Code that collects the values
14 forkJoin([
15     this._accountTempProposalService.countAccountTempProposals(mainProposerFilter),
16     this._accountTempProposalService.countAccountTempProposals(proposerFilter),
17     this._accountTempProposalService.countAccountTempProposals(localContactFilter),
18     this._accountTempProposalService.countAccountTempProposals({}),
19 ])
20 .pipe(take(1))
21 .subscribe(counts => {
22     this.mainProposerCount.set(counts[0]); // Assign the Observable's value (here, array's first
23         value) to mainProposerCount, which update the variable
24     this.proposerCount.set(counts[1]);
25     this.localContactCount.set(counts[2]);
26     this.allProposalsCount.set(counts[3]);
27 });
28 }

```

Listing II.3 – Méthode `fetchProposals` du fichier user-temporary-proposals.component.ts

La méthode `fetchProposals` initialise d’abord des variables, qui seront utilisées en tant que filtres. Elle appelle ensuite l’opérateur `forkJoin`. Cette méthode prend un tableau d’Observables en entrée

et renvoie un Observable de tableau de type nombre. Les Observables sont des objets qui, en programmation « réactive », permettent d'émettre des valeurs sur une période donnée, de manière synchrone ou asynchrone. Les requêtes HTTP gérées par Angular, comme celle-ci, sont des Observables asynchrones, permettant à l'application d'effectuer d'autres tâches en attente de la réponse émise par les Observables. La méthode pipe s'applique au résultat de la méthode forkJoin. Elle est utilisée pour chaîner plusieurs opérateurs sur les Observables. Ici le premier opérateur « pipé » est take(1). Enfin, la méthode passée en paramètre de « subscribe » permet d'accéder au tableau de valeurs émis par la chaîne d'observables et de fixer les valeurs des variables de UserTemporaryProposalComponent. Ce sont ces valeurs qui seront affichées dans l'écran de l'utilisateur par l'intermédiaire de la partie HTML du composant vue précédemment.

Dans la section suivante, nous allons voir comment sont alimentées ces valeurs affichées sur la tuile.

2.3 Endpoints et partie backend

Dans la dernière section, je vous ai parlé à plusieurs reprises de requêtes HTTP, sans pour autant expliciter leur signification. Le frontend de l'application communique avec le backend grâce à une API REST. Une API REST est une interface qui respecte les principes de conception du style architectural REST (voir en annexes).

Pour chaque demande provenant du client, le frontend envoie une requête HTTP au backend qui traite cette demande et retourne une réponse au frontend. Ainsi, si nous reprenons la Figure II.3, chaque flèche représente une requête HTTP. La question est donc : comment ces requêtes sont-elles traitées par les services backend ? Par exemple, comment le bon nombre de proposals est-il renvoyé au niveau de la tuile ?

Comme expliqué précédemment, la requête envoyée par le client est transmise au Dashboard-server vers ce que l'on appelle des endpoints. Les endpoints sont des méthodes côté backend qui vont « intercepter » une requête sur une URL spécifique. Ils définissent les opérations disponibles pour interagir avec les ressources du serveur [4]. Il existe de multiples requêtes HTTP que le lecteur peut retrouver en annexes.

Le Dashboard-server peut gérer les requêtes HTTP grâce à un objet que l'on nomme le controller. Son but est de réceptionner la requête et de communiquer après avec un objet Service du backend, qui lui doit gérer la logique métier.

Le Service contient la logique métier, c'est-à-dire que c'est à ce moment-là que sont appliqués les différentes règles, les calculs et les transformations avant de manipuler les données. Il appelle ensuite le RestClient pour transmettre la requête vers le microservice approprié.

Le RestClient joue le rôle du client vers les Microservices dans la partie backend de l'application.

Depuis le Microservice, nous pouvons observer un comportement similaire. Un controller intercepte la requête et l'envoie au Service. Cependant, le Service n'appelle pas un RestClient mais il appelle un Repository hibernate (ORM) pour accéder aux données de la base.

Le Repository interagit alors avec la BDD, en effectuant des requêtes de type HQL ou Criteria. Les requêtes HQL sont similaires à des requêtes SQL, seulement elles sont axées vers de l'orienté objet. Les requêtes Criteria utilisent des objets hibernates dédiés pour construire la requête. Elles sont notamment mieux adaptées lorsque nous devons construire des requêtes de manière dynamique.

Le Repository utilise ensuite les Entities hibernate. Une Entity permet d'associer une table de la BDD avec une classe du modèle objet (bean).

Le schéma ci-dessous offre un visuel, pour mieux comprendre.

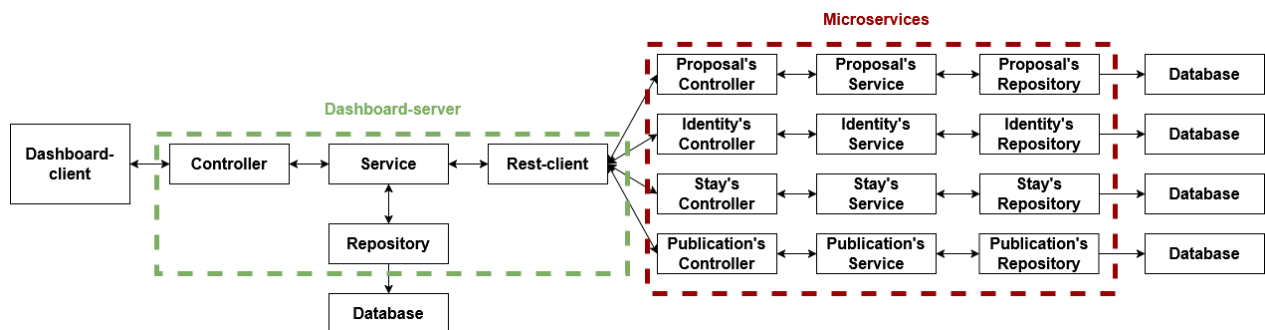


FIGURE II.4 – Architecture détaillée du projet Dashboard.

[3]

Chaque microservice/backend est organisé en plusieurs couches, chacune ayant une responsabilité bien définie : une telle architecture est appelée architecture N-tiers. Nous pouvons la résumer ainsi : une couche cliente (controllers), une couche métier (services et modèles de données) et une couche

de persistance (repositories).

Maintenant que nous avons compris le déroulé des opérations, comment cela s'implémente-t-il ?

Commençons par le controller du Dashboard-server.

```
1  @Path("/account/temporary-proposals") //
2  @ApplicationScoped
3  @Consumes(MediaType.APPLICATION_JSON)
4  @Produces(MediaType.APPLICATION_JSON)
5  @Authenticated
6  public class AccountTemporaryProposalController {
7      // Initialize variables and method
8      private final TemporaryProposalService temporaryProposalService;
9      public AccountTemporaryProposalController(final TemporaryProposalService
10         temporaryProposalService) {
11         this.temporaryProposalService = temporaryProposalService;
12     }
13     @GET
14     @Path("/count")
15     public Long countAccountTemporaryProposals(@Context final SecurityContext securityContext,
16         TemporaryProposalFilter temporaryProposalFilter){
17         AccountToken accountToken = (AccountToken) securityContext.getUserPrincipal();
18
19         if (temporaryProposalFilter == null) {
20             temporaryProposalFilter = new TemporaryProposalFilter();
21         }
22         temporaryProposalFilter.setIdentityIds(List.of(Long.toString(accountToken.user().getId())
23             ));
24         return this.temporaryProposalService.countAll(temporaryProposalFilter);
25     }
26 }
```

Listing II.4 – Controller de Dashboard-server nommé AccountTemporaryProposalController.java

Les éléments importants de ce fichier sont les annotations `@Path`. Le lecteur peut en apprendre plus sur la signification des différentes annotations en annexes. La première annotation, à ligne 1, signifie que l'URL de base du controller est **/account/temporary-proposals**. La seconde annotation `@Path`, signifie que si l'URL de la requête correspond à **/account/temporary-proposals/count**, conjuguée à une méthode HTTP GET (annotation précédente, ligne 13) alors la méthode `countAccountTemporaryProposals` du controller sera exécutée. Elle renvoie la valeur retournée par l'appel à la méthode `countAll` du service `temporaryProposalService`. Cette méthode est définie dans le

Service, ainsi continuons donc avec le Service :

```
1  @ApplicationScoped
2  public class TemporaryProposalService {
3
4      @RestClient
5      TemporaryProposalRestClient temporaryProposalRestClient;
6      public Long countAll(final TemporaryProposalFilter filter) {
7
8          return temporaryProposalRestClient.countAll(filter);
9      }
10 }
```

Listing II.5 – Service de Dashboard nommé TemporaryProposalService.java

L'annotation `@RestClient` signifie que nous créons un objet de type `RestClient`. La méthode `countAll` précédemment appelée, appelle elle aussi une méthode `countAll`, mais qui s'applique à un objet de type `RestClient`, défini dans la classe `TemporaryProposalRestClient.java`.

```
1  @RegisterRestClient(configKey = "proposal-api")
2  @RegisterProvider(RestClientResponseExceptionMapper.class)
3  @Path("/api")
4  @Produces(MediaType.APPLICATION_JSON)
5  @ApplicationScoped
6  public interface TemporaryProposalRestClient {
7      @GET
8      @Path("/temporary-proposals/count")
9      Long countAll(@BeanParam final TemporaryProposalFilter params);
10 }
```

Listing II.6 – Code issu du fichier TemporaryRestClient.java

L'annotation `@RegisterRestClient` permet de transmettre la requête, définie de la ligne 7 à 9, vers le microservice `proposal-service`.

Continuons avec le controller associé au microservice `proposal-service`. Le controller de `proposal-service` est construit de manière similaire à celui du `Dashboard-server`.

```
1  @Path("/temporary-proposals") // Annotation used to indicate the URL where the endpoint will be
    available in order to respond HTTP requests
2  @Consumes(MediaType.APPLICATION_JSON)
3  @Produces(MediaType.APPLICATION_JSON)
4  @ApplicationScoped
5  public class TemporaryProposalController {
6      private final TemporaryProposalService temporaryProposalService;
```

```

7
8     public TemporaryProposalController(final TemporaryProposalService temporaryProposalService)
9     {
10         this.temporaryProposalService = temporaryProposalService;
11     }
12
13     @GET
14     @Path("/count")
15     public Long countAll(@BeanParam final TemporaryProposalFilter proposalFilter) {
16         return this.temporaryProposalService.countAll(proposalFilter);
17     }

```

Listing II.7 – Code de proposal-service nommé TemporaryProposalController.java

Le controller expose une requête HTTP GET via le chemin **/temporary-proposals/count**, permettant de compter les proposals temporaires correspondant aux filtres fournis en paramètres. Le controller renvoie un objet défini dans le Service, sur lequel nous appelons la méthode countAll.

```

1  @ApplicationScoped // Annotation used to define the scope of a bean in the app
2  @Transactional // Annotation used to handle transactions
3  public class TemporaryProposalService {
4      private final TemporaryProposalRepository repository; // Definition of an attribute named
5                      repository, which has the type TemporaryProposalRepository (a class that handles
6                      database access)
7
8      @Inject // Annotation used to automatically inject the dependencies needed so that the class
9              can work properly.
10     public TemporaryProposalService(TemporaryProposalRepository repository) {
11         this.repository = repository;
12     }
13
14     // Definition of a method named countAll that is used in TemporaryProposalController file.
15     // It takes a filter defined in the TemporaryProposalFilter
16     public Long countAll(TemporaryProposalFilter filter) {
17         return this.repository.countAll(filter); // Calls the countAll method defined in the
18             TemporaryProposalRepository file
19     }
20 }

```

Listing II.8 – Service de proposal-service nommé TemporaryProposalService.java

L'objet sur lequel nous avons appelé la méthode countAll renvoie une valeur depuis le Repository. TemporaryProposalFilter est une classe Java que j'ai codée, qui définit des filtres, utilisés en paramètres.

```

1  @ApplicationScoped
2  public class TemporaryProposalRepository {
3      private static final Logger logger = LoggerFactory.getLogger(TemporaryProposalRepository.
4          class);
5
6      private final EntityManager entityManager;
7
8      @Inject
9      TemporaryProposalRepository(EntityManager entityManager) {
10         this.entityManager = entityManager;
11     }
12
13     // Method used in the TemporaryProposalService file, enabling to count the number of
14     // proposals corresponding to filters
15     public Long countAll(final TemporaryProposalFilter filter) {
16         final CriteriaBuilder cb = entityManager.getCriteriaBuilder(); // CriteriaBuilder is a
17         // class used to construct criteria queries
18         final CriteriaQuery<Long> cbQuery = cb.createQuery(Long.class); // Create a CriteriaQuery
19         // object.
20         final Root<TemporaryProposal> root = cbQuery.from(TemporaryProposal.class); // Create and
21         // add a query root corresponding to the given entity, forming a cartesian product with
22         // any existing roots.
23
24         TemporaryProposalRequestContext context = new TemporaryProposalRequestContext(root); //
25         // Creation of a new object context
26         final List<Predicate> predicates = this.convertFilterToPredicates(filter, cbQuery, cb,
27             context);
28
29         cbQuery.where(cb.and(predicates.toArray(new Predicate[0])));
30
31         cbQuery.select(cb.countDistinct(root));
32         TypedQuery<Long> query = entityManager.createQuery(cbQuery);
33         return query.getSingleResult();
34     }

```

Listing II.9 – Repository de proposal-service nommé TemporaryProposalRepository.java

Le fichier est bien plus long ; il y a notamment toutes les méthodes permettant de transformer les filtres en prédicats pour les requêtes criteria. Le Repository utilise les différentes Entities (représentation en hibernate des tables de la BDD du Microservices proposal-service). J'ai donc créé le même nombre d'Entities que de tables temporaires dans la BDD de la figure II.4, soit 6 fichiers Entity.

```

1  @Entity
2  @Table(name = "temp_measure")

```

```

3 public class TemporaryMeasure {
4     @Id
5     @Column(name = "id", nullable = false)
6     private Long id;
7
8     @JsonIgnore
9     @ManyToOne
10    @JoinColumn(name = "temp_proposal_id", nullable = false, foreignKey = @ForeignKey(name = "
        temp_measure_temp_proposal_id_fk"))
11    private TemporaryProposal temporaryProposal;

```

Listing II.10 – Partie de code de l’Entity nommé TemporaryProposal

Voici la création d’une Entity hibernate grâce à l’annotation `@Entity`. Celle-ci est celle de la table `temp_measure`. Nous lui ajoutons ensuite tous ses propriétés « mappées » sur les colonnes de la table via plusieurs annotations. De plus, certaines tables de la BDD peuvent être liées à d’autres. Nous pouvons voir que les tables (et donc les Entities) sont liées via les annotations `@ForeignKey` et leur nom. Dans le but de compter le nombre de proposals, cela n’est pas particulièrement important, cependant, quand il faudra afficher le détail des proposals, cela sera un point majeur.

3 Les résultats

Maintenant que j’ai présenté les différents composants techniques de manière isolée et expliqué leur rôle respectif à travers des extraits de code et des schémas, il est temps de les réunir pour en comprendre le fonctionnement global. Le but est donc de présenter le Dashboard dans son ensemble : non plus comme un empilement de Microservices ou de conteneurs, mais comme une unique unité. Je vous ai précédemment expliqué les différents outils que sont Docker, Quarkus et K8s. Si nous essayons de visualiser ces outils sous forme de boîtes, K8s est la boîte extérieure. Il orchestre, supervise et déploie tout. K8s est composé d’un « Control Plane », qui est le cerveau du cluster. Il gère toutes les machines qui lui sont associées, nommées des « Nodes ». À l’intérieur de ceux-ci, nous trouvons ces principaux objets :

- Les Pods : Ce sont des unités minimales d’exécution et chaque Pod peut contenir un ou plusieurs conteneurs Docker.
- Les Deployments : Ce sont des contrôleurs K8s qui gèrent en particulier le nombre de Pods, via des Replicas.

- Les Services Kubernetes : Ils permettent d'établir la communication entre les différents Replicas d'un même Pod.

K8s est un sujet extrêmement complexe et n'étant pas le sujet principal du stage, je ne rentrerai pas plus en détail. Le lecteur peut néanmoins en apprendre plus grâce à la documentation officielle de Kubernetes [11].

Enfin, pour déployer et exécuter les éléments de l'application, j'ai utilisé différentes commandes dans les terminaux intégrés des IDE. Le lecteur pourra trouver les instructions détaillées pour exécuter l'application Dashboard en local en annexe. Une fois ces opérations effectuées, nous pouvons accéder, en local, à l'application Dashboard depuis notre navigateur favori où nous pouvons voir le rendu final de l'application. Voici une capture d'écran de la tuile à la fin du stage :



FIGURE II.5 – Tuile finalisée.

C Responsabilités environnementales et sociétales dans le cadre du stage

1 Impact énergétique

Comme tout projet informatique, celui mené durant mon stage a généré une consommation énergétique liée à l'utilisation d'ordinateurs, de serveurs et d'outils de déploiement comme Docker et K8s, dans un environnement numérique naturellement énergivore. Bien que le développement web ait un impact environnemental relativement indirect et offre peu de leviers d'action concrets, l'ILL adopte des pratiques favorisant la durabilité, telles qu'un Plan de Déplacement Entreprises (PDE) avantageux, un site bien desservi par les transports en commun, et la promotion du télétravail. Pour ma part, j'ai contribué à cette démarche en pratiquant le covoiturage avec deux collègues.

2 Approche humaine et sociétale du travail à l'ILL

Le projet sur lequel j'ai travaillé se veut inclusif et accessible, ne ciblant aucun groupe particulier et ne nécessitant pas de compétences techniques spécifiques de la part des utilisateurs. De plus, les enjeux de sécurité sont primordiaux, via des outils comme K8s, mais je ne peux que les mentionner, n'ayant pas directement travaillé dessus. Cependant, mon expérience à l'ILL m'a toutefois permis de découvrir une culture forte de la sécurité et du bien-être au travail, illustrée par un séminaire obligatoire de prévention des risques professionnels et par les mesures rigoureuses en place dans un environnement sensible, notamment lors de visites d'instruments en fonctionnement. Cette immersion m'a permis de comprendre que, dans un cadre scientifique à haut risque, la dimension humaine est une priorité absolue.

D Analyse réflexive du stage

Ce stage m'a permis de passer d'une approche théorique à une expérience concrète du développement web en entreprise. J'ai découvert l'importance de la rigueur, de la communication et de la documentation dans un projet collaboratif. De plus, l'utilisation d'outils comme Docker et Kubernetes, que je ne maîtrisais pas au départ, m'a poussé à sortir de ma zone de confort, mais grâce à l'accompagnement de mon tuteur et à ma capacité à apprendre en autonomie, j'ai réussi à m'adapter et à progresser rapidement. J'ai également mieux cerné mes préférences professionnelles, notamment mon intérêt pour la compréhension globale des systèmes, au-delà du développement pur.

Par ailleurs, j'ai pu expérimenter un mode de travail structuré autour de la méthode agile, avec des sprints, des démonstrations, des réunions quotidiennes (daily) et des séances de planification. Dès mon arrivée, j'ai assisté à une démonstration du Dashboard, qui m'a permis de mieux comprendre les enjeux du projet. Lors de la deuxième démo, j'ai moi-même présenté mon travail, un exercice valorisant et formateur. Ce cadre m'a permis de développer mes compétences en organisation, en communication et en intégration de retours, tout en découvrant les réalités de la collaboration en entreprise.

III Conclusion

Ce stage avait pour objectif principal l'implémentation d'une tuile de l'application Dashboard, destinée à afficher des informations clés à l'utilisateur sur les proposals avant intégration. Grâce à un travail rigoureux et une bonne organisation, j'ai pu achever cette mission dans les temps.

Ce stage s'est révélé extrêmement formateur et stimulant. Il m'a permis de mieux comprendre le déroulement concret d'un projet informatique, le développement bien entendu mais aussi en partie le déploiement. J'ai découvert et manipulé de nombreux outils modernes utilisés dans l'industrie, tels que Docker, Kubernetes, Angular et Quarkus, ainsi que des langages que je maîtrisais peu ou pas, comme Java et TypeScript.

Au-delà des compétences techniques, cette expérience m'a permis de développer des savoir-faire professionnels essentiels : la collaboration avec une équipe, le travail en autonomie, la gestion des priorités et l'adaptation à un environnement technique complexe.

L'informatique étant un domaine vaste et en constante évolution, ce stage m'a offert une vision plus nuancée et concrète du métier de développeur. J'ai découvert une facette de l'informatique que je ne connaissais qu'en surface, et que j'ai pu explorer en profondeur. Bien que le développement ne soit pas nécessairement le chemin que je choisirai sur le long terme, cette immersion m'a permis d'élargir ma compréhension du secteur et d'orienter plus précisément mes réflexions sur mon projet professionnel.

Enfin, je conclus ce rapport avec une réelle satisfaction : celle d'avoir contribué à un projet utile, dont les résultats auront un impact direct sur l'expérience utilisateur. Cette perspective donne du sens au travail accompli et renforce ma motivation à continuer à apprendre et à progresser dans le domaine de l'informatique.

IV Annexes

1 Liste des tâches effectuées

Voici une liste des tâches effectuées lors de mon stage :

- Comprendre les objets sur lesquels j’ai travaillé.
- Comprendre l’architecture du projet.
- Comprendre, plus ou moins en profondeur, les outils utilisés.
- Implémenter la tuile.
- Participer aux différentes réunions.
- Présenter son travail lors d’une démonstration.
- Participation à un séminaire obligatoire sur la prévention des risques au travail.
- Visite d’instruments.

2 Détail des domaines les plus étudiés à l’ILL

La figure en dessous présente les différents domaines étudiés à l’ILL :

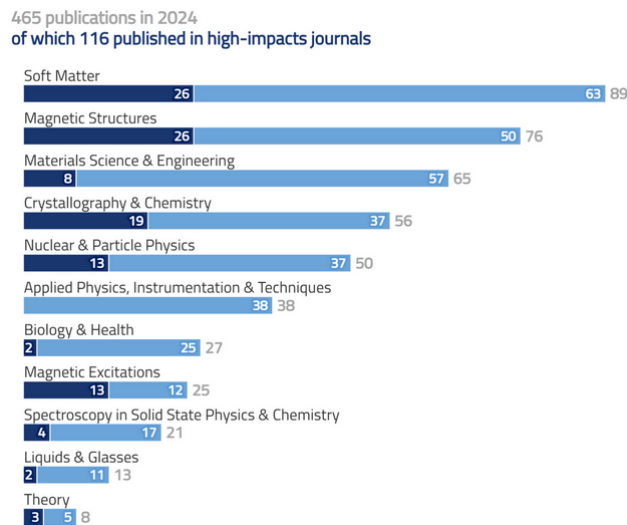


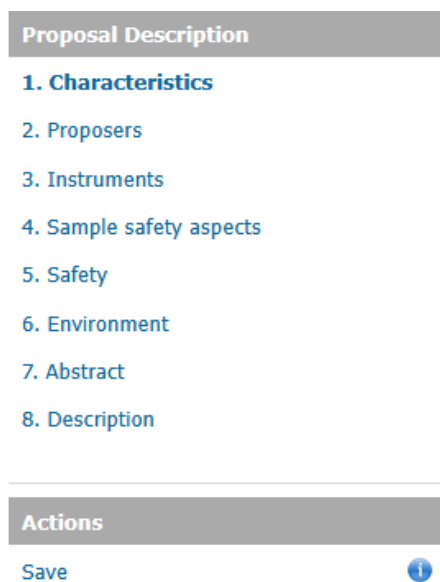
FIGURE IV.1 – Les différents domaines d’expérimentations à l’ILL [18].

3 Les pays membres de l'ILL

L'Espagne est le premier pays à être devenu membre de l'ILL en 1987. Un an plus tard, c'est au tour de la Suisse de le devenir, en 1988. Dans les années 90, trois nouveaux pays deviennent membre, l'Autriche en 1990, l'Italie en 1997 et la République tchèque en 1999. Dans le nouveau millénaire, nous comptons 4 nouveaux membres, la Suède qui les rejoint en 2005, la Pologne en 2006 ainsi que la Slovaquie et le Danemark en 2009. Enfin, très récemment, la Slovénie vient compléter cette liste en devenant membre en 2020.

4 Soumettre une proposal via le User Club

Quand un utilisateur veut soumettre une proposal, il lui faut renseigner un questionnaire. La capture d'écran suivante présente les intitulés auxquels celui-ci doit répondre :



Proposal Description

1. Characteristics
2. Proposers
3. Instruments
4. Sample safety aspects
5. Safety
6. Environment
7. Abstract
8. Description

Actions


Save 

FIGURE IV.2 – Menu du formulaire de soumission d'une proposal
[20]

Dans la section Characteristics, l'utilisateur doit renseigner des informations générales telles que le titre de la proposal, le collège auquel elle est associée, le domaine de recherche général (biologie, chimie, ingénierie, etc...)

Dans la section Proposer, l'utilisateur définit des co-proposers, étant lui-même main proposer. Il

peut aussi ajouter des local contacts.

Dans la section Instruments, l'utilisateur définit les instruments sur lesquels il veut faire leur expérience. Il peut choisir plusieurs instruments pour une même proposal ou choisir des instruments de substitution.

Dans la section Sample safety aspects, l'utilisateur indique les échantillons sur lesquels porteront les expériences. Il doit ensuite renseigner un nombre important d'informations sur celui-ci comme sa masse, son état, etc...

La section Safety est utilisée pour que l'utilisateur renseigne la dangerosité des produits utilisés et les risques inhérents aux expériences.

Dans la section Environment, l'utilisateur renseigne des environnements spéciaux pour son expérience. Cela peut être une température (très haute ou très basse), une pression, un champ électrique ou magnétique spécifique, etc...

Dans la section Abstract, l'utilisateur doit renseigner un résumé sur son expérience, comme le protocole, une description de l'expérience, etc...

Enfin, dans la section Description, l'utilisateur ajoute des documents décrivant l'intérêt scientifique de la proposal et éventuellement la mise en place technique des expériences.

5 Le National Balance

Le National Balance est la dernière étape qui assure du temps de faisceau aux proposals. Il s'agit d'une procédure gérée par le SCO (Scientific Coordination Office), visant à réajuster le temps de faisceau précédemment alloué par les peer review, dans l'objectif de respecter le niveau de financement de chaque pays associé ou membre.

6 Les rôles du User Office et des Secrétaires de collège

Le rôle principal du User Office est d'accompagner les utilisateurs. Par exemple, ce bureau aide les utilisateurs désirant venir sur le site pour une expérimentation, en les assistant pour organiser leur séjour et en leur offrant un accompagnement administratif [15].

Les secrétaires de collège sont des scientifiques de l'ILL responsables d'un domaine de recherche (collège). Le secrétaire de collège nomme les personnes effectuant l'internal review. Il supervise aussi la peer review. L'image suivante présente l'ensemble des collèges.

College	Main subject areas
College 1	Applied Materials Science, Instrumentation and Techniques
College 2	Theory
College 3	Nuclear and Particle Physics
College 4	Magnetic excitations
College 5a	Crystallography
College 5b	Magnetic structures
College 5c	Nanoscale magnetism and superconductivity
College 6	Structure and dynamics of disordered systems
College 7	Spectroscopy in solid state physics & chemistry
College 8	Structure and dynamics of biological systems
College 9	Structure and dynamics of soft-condensed matter

FIGURE IV.3 – Ensemble des collèges présent à l'ILL.
[17]

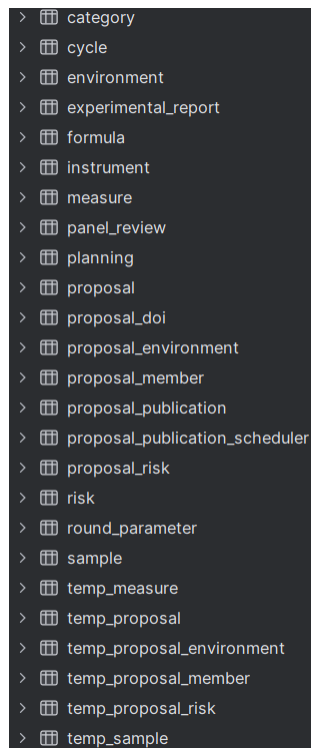
7 Le Dev-proxy

Le Dev-proxy est un système utilisé au sein de l'ILL lors du développement local. Il s'agit d'un proxy, c'est-à-dire un composant intermédiaire entre deux hôtes, qui a pour rôle ici de faciliter les

échanges.

Concrètement, le Dev-proxy est uniquement actif dans un environnement local. Imaginons que je fasse tourner le microservice proposal-service sur ma machine, tandis que les autres Microservices tournent dans K8s. Lorsqu'une requête client vise le microservice proposal-service, le Dev-proxy la redirige automatiquement vers l'instance locale, plutôt que vers celle déployée sur K8s. En revanche, pour tous les autres Microservices non présents en local, le Dev-proxy redirige les requêtes vers leur version sur K8s.

8 Illustration de la BDD du microservice proposal-service



```
> category
> cycle
> environment
> experimental_report
> formula
> instrument
> measure
> panel_review
> planning
> proposal
> proposal_doi
> proposal_environment
> proposal_member
> proposal_publication
> proposal_publication_scheduler
> proposal_risk
> risk
> round_parameter
> sample
> temp_measure
> temp_proposal
> temp_proposal_environment
> temp_proposal_member
> temp_proposal_risk
> temp_sample
```

FIGURE IV.4 – Ensemble des tables de la BDD associée au microservice Proposal.

9 Liste de tous les logiciels installés

Lors de ma première journée, j'ai installé un certain nombre de logiciels, qui m'ont permis de configurer mon espace de travail. Voici la liste de ceux que je n'ai pas pu préciser, avec leurs

fonctions associées :

- JetBrains Toolbox : Gestionnaire d'IDE. Il gère les IDE IntelliJ IDEA, WebStorm et DataGrip.
- IntelliJ IDEA : Installation depuis JetBrains Toolbox. C'est un IDE que j'ai utilisé pour mes développements Java.
- WebStorm : Installation depuis JetBrains Toolbox. C'est un IDE que j'ai utilisé pour mes développements Angular.
- DataGrip : Installation depuis JetBrains Toolbox. C'est un IDE permettant d'interagir avec les BDD postgresql.
- Angular CLI : Outil d'interface de ligne de commande permettant de créer, développer, tester, déployer et maintenir des applications Angular, directement à partir d'un terminal de commandes.
- JBang : Permet d'exécuter des fichiers Java en ligne de commandes.
- Quarkus CLI : Outil d'interface de ligne de commande permettant de créer, développer et gérer des applications Quarkus. Installation via JBang.
- Git
- Docker

10 Git

Les états Git sont les suivants :

- Modified : Cela signifie que le fichier a été modifié localement. Ces modifications ne seront pas prises en compte en cas de commit.
- Staged : Cela signifie que le fichier est préparé pour le commit. Il sera inclus dans le prochain commit.
- Committed : Le fichier a été enregistré dans le dépôt local. Il faut effectuer la commande git push dans un terminal pour qu'il soit envoyé au dépôt distant.

Les principales commandes Git sont les suivantes [5] :

- git add : Ajoute tous les fichiers modifiés.
- git add < *fichier* > : Ajoute un fichier à la zone de staging (zone d'attente).

- `git branch` : Liste les branches.
- `git branch < nom >` : Crée une nouvelle branche.
- `git checkout -b < nom >` : Crée et bascule vers une autre branche.
- `git clone < URL >` : Copie un dépôt distant sur ta machine.
- `git commit -m "Message"` : Valide les changements ajoutés.
- `git fetch` : Récupère les modifications sans les fusionner.
- `git init` : Crée un nouveau dépôt git dans un dossier.
- `git log (-oneline)` : Montre l'historique des commits. Si on y met le paramètre `oneline`, on obtient une version condensée.
- `git merge < branche >` : Fusionne une branche dans l'actuelle.
- `git pull` : Récupère et fusionne les dernières modifications du dépôt distant.
- `git push origin < branche >` : Envoie les commits vers GitLab.
- `git remote -v` : Liste les dépôts distants.
- `git reset < fichier >` : Retire un fichier de la zone de staging.
- `git reset --hard` : Remet tout comme au dernier commit (destructif).
- `git restore < fichier >` : Restaure un fichier à son dernier commit.
- `git show` : Montre les détails du dernier commit sur la branche actuelle.
- `git status` : Montre les fichiers modifiés, ajoutés ou en attente de commit.
- `git switch < nom >` : Change de branche.

11 TypeScript et Angular

11.1 TypeScript

TypeScript est un langage de programmation basé sur celui de JavaScript. Cependant, contrairement à celui-ci, TypeScript est un langage de programmation à typage statique, c'est-à-dire que lorsque l'on initialise une variable, celle-ci se doit d'avoir un type prédéfini. Si ce type n'est pas respecté, alors une erreur est renvoyée. Voici un exemple qui illustre mes propos :

```
1 let message = "Merci d'avoir lu ce rapport";  
2 message = 42;  
3 console.log(message);
```

Listing IV.1 – Code en JavaScript

Dans ce cas là (JavaScript), comme aucun type n'est défini pour la variable message, alors celle-ci, qui initialement était de type string peut être remplacée par un type int. Le code s'exécutera sans encombre et affichera la valeur 42.

```
1 let message : string = "Merci d'avoir lu ce rapport";  
2 message = 42;  
3 console.log(message);
```

Listing IV.2 – Code en TypeScript

Dans ce cas là (TypeScript), comme le type string est initialement défini pour la variable message, alors toute tentative de la remplacer par un autre type échouera, engendrant une erreur. C'est exactement ce qui se passerait dans le cas de ce code.

11.2 Angular

Avec pour objectif de comprendre Angular, j'ai suivi deux cours, permettant de pratiquer un peu, dans le but de créer ma tuile :

- <https://angular.dev/tutorials/first-app/01-hello-world>
- <https://openclassrooms.com/fr/courses/7471261-debutez-avec-angular/7549306-ajoutez-des-proprietes-personnalisees>

Ils m'ont permis d'acquérir mes premières connaissances en Angular, mais je me suis rapidement retrouvé à « jouer » avec le code, dans l'environnement local, dans le but de comprendre à quoi correspondaient chaque ligne, objets, etc...

12 Les critères définissant une API REST

Pour rappel, les critères définissant une API REST sont les suivants :

- Gestion des requêtes web via le protocole HTTP.
- Communications stateless, entre client et serveur.
- Une URI de base qui représente une ou des ressources.

- Utilisation des méthodes HTTP (GET, POST, DELETE, PUT, PATCH) pour la sémantique de l'opération à effectuer sur une ou des ressources.

Le protocole HTTP est le principal protocole de transmission d'informations sur internet. Il se fait à travers des documents hypertextes, d'où son nom.

Les ressources sont des objets manipulables via des requêtes HTTP.

Une communication stateless signifie que les données ne sont pas enregistrées. Certaines données peuvent néanmoins être mises en cache, dans le but d'éviter au client de refaire des actions qu'il a déjà faites.

Une telle sémantique permet d'être comprise par d'autres développeurs.

13 Les différentes requêtes HTTP

Le protocole HTTP définit un ensemble de méthodes de requête qui permettent de préciser l'intention d'une requête envoyée au serveur, ainsi que le comportement attendu en cas de succès. Chaque méthode possède sa propre sémantique et une syntaxe spécifique. Il existe au total neuf méthodes standardisées :

- CONNECT : Cette requête demande à un proxy d'établir une connexion HTTP entre lui et le serveur. Dans le cas où cette requête est acceptée, le proxy et le serveur envoient de manière aveugle un flux de données, jusqu'à ce que ce tunnel soit arrêté.
- DELETE : Cette requête demande au serveur de supprimer une ressource.
- GET : Cette méthode permet la représentation d'une donnée, c'est-à-dire de récupérer des données.
- HEAD : Cette méthode permet d'obtenir les metadata d'une ressource sous la forme de headers, que le serveur aurait envoyé, si la méthode GET avait été utilisée à la place.
- OPTIONS : Cette méthode permet d'avoir des options de communication en fonction de l'URL ou du serveur. Cela signifie interroger le serveur sur les requêtes HTTP autorisées.
- PATCH : Cette méthode applique des changements à une ressource. En comparaison avec PUT, PATCH ne modifie que les champs renseignés de la ressource, le reste de l'objet restant intact.
- POST : Cette méthode permet d'ajouter des données au serveur. La différence entre un PUT

et un POST est qu'appeler plusieurs fois la requête PUT n'aura pas d'effets secondaires, c'est équivalent à l'appeler une unique fois, tandis que si on appelle plusieurs fois la requête POST, cela aura certainement des effets secondaires.

- PUT : Cette requête applique des changements à une ressource. En comparaison avec PATCH, PUT modifie l'entiereté de l'objet. Les champs non renseignés sont alors réinitialisés ou supprimés.
- TRACE : Cette méthode est utilisée pour diagnostiquer le chemin exact qu'une requête suit jusqu'au serveur cible. [7].

Dans le cadre du développement d'une API REST, les méthodes les plus couramment utilisées sont GET, POST, PUT, DELETE et PATCH.

14 Hibernate

Hibernate est un framework open source de type ORM. Il permet de représenter une BDD en objets Java et inversement. Hibernate a besoin de plusieurs éléments pour fonctionner :

- Une classe de type javabean qui encapsule les données d'une occurrence d'une table. Dans notre cas, c'est une classe de type Entity.
- Des annotations qui assurent la correspondance entre une Entity et une table ainsi qu'entre les colonnes d'une table et les propriétés d'une Entity (mapping).
- Des propriétés de configuration notamment des informations concernant la connexion à la base de données [8].

15 Les annotations Java

Les annotations sont des objets permettant d'ajouter des métadonnées à une portion de code. Dans la plupart des cas, elles servent à conférer un comportement spécifique ou particulier à ce code. Ces annotations précèdent toujours la déclaration qu'elles enrichissent et se distinguent facilement grâce à leur syntaxe, débutant systématiquement par le symbole « @ ».

Elles sont couramment utilisées pour diverses finalités telles que la génération automatique de code, la production de documentation, ou encore pour indiquer au compilateur d'ignorer certaines

erreurs [9]. Voici une liste de l'ensemble des annotations que j'ai pu trouver, que ce soit en faisant des recherches ou en les lisant dans le code :

- `@ApplicationScoped` : Définit une portée d'application pour un bean. Le bean ainsi annoté a une seule instance partagée pendant toute la durée de vie de l'application.
- `@Authenticated` : Restreint l'accès à une méthode ou ressource HTTP aux utilisateurs authentifiés. L'accès nécessite une authentification préalable.
- `@BeanParam` : Permet d'agréger plusieurs paramètres HTTP dans un seul objet Java.
- `@ConfigProperty` : Injecte des valeurs de configuration définies dans un fichier de configuration (comme `application.properties`) dans un champ ou une méthode.
- `@Consumes` : Indique les types MIME (Content-Type) qu'une méthode ou ressource peut accepter en entrée.
- `@Context` : Injecte des objets liés au contexte de la requête.
- `@DeleteMapping` : Indique que la méthode doit répondre à une requête HTTP DELETE, typiquement pour supprimer une ressource.
- `@Entity` : Indique qu'une classe est une Entity.
- `@Generated` : Sert à marquer du code généré automatiquement.
- `@GetMapping` : Spécifie qu'une méthode est mappée sur une requête HTTP GET.
- `@Id` : Spécifie la clé primaire d'une Entity.
- `@Inject` : Sert à injecter des dépendances automatiquement, sans avoir à instancier manuellement l'objet.
- `@NamedQueries` : Conteneur regroupant plusieurs annotations `@NamedQuery`.
- `@NamedQuery` : Déclare une requête JPQL nommée, réutilisable via son nom.
- `@NotBlank` : Valide qu'un champ string n'est ni nul, ni vide, ni uniquement composé d'espaces.
- `@OneToMany`, `@ManyToOne`, `@ManyToMany` : Définissent les relations entre les Entities, pour refléter les associations entre tables.
- `@Override` : Indique qu'une méthode redéfinit une méthode d'une classe parente ou implémente une méthode d'une interface.
- `@Path` : Associe une classe ou méthode à un chemin d'URL dans un service REST. Sert à définir un endpoint.

- `@PathParam` : Permet d'extraire des variables dynamiques directement depuis l'URL dans un service REST.
- `@PathVariable` : Dans Spring, permet de lier une partie dynamique de l'URL à un paramètre de méthode.
- `@PostConstruct` et `@PreDestroy` : Méthodes annotées appelées respectivement après l'initialisation et avant la destruction d'un bean.
- `@PostMapping` : Spécifie qu'une méthode doit traiter les requêtes HTTP POST, généralement pour créer une nouvelle ressource.
- `@Produces` : Indique les types MIME qu'une méthode peut renvoyer en réponse à une requête.
- `@PutMapping` : Indique que la méthode doit répondre à une requête HTTP PUT, généralement utilisée pour mettre à jour une ressource.
- `@QueryParam` : Extrait la valeur d'un paramètre de requête.
- `@RequestBody` : Utilisée dans Spring pour indiquer que le corps d'une requête HTTP doit être converti en objet Java.
- `@RequestMapping` : Associe une méthode ou classe à une ou plusieurs routes HTTP, avec ou sans méthode HTTP précisée.
- `@RequestParam` : Extrait les paramètres de requête de l'URL.
- `@ResponseStatus` : Spécifie le code HTTP à retourner pour une méthode ou un contrôleur.
- `@RestClient` : Permet d'injecter automatiquement un client REST typé, facilitant les appels HTTP comme des méthodes Java classiques.
- `@Size` : Valide qu'une collection, une chaîne ou un tableau respecte des bornes de taille minimale et maximale.
- `@SuppressWarnings` : Annotation utilisée pour demander au compilateur d'ignorer certains avertissements spécifiques.
- `@Valid` : Déclenche automatiquement la validation d'un objet Java selon les annotations de validation présentes (par exemple : `@NotBlank`, `@Size`, etc...).

16 Lancer l'application Dashboard en local

Pour lancer la partie cliente de l'application, donc le Dashboard-client, il me faut utiliser la commande `npm start` dans le terminal intégré de l'IDE. Cela permet de lancer l'application en local et de s'y connecter via une URL spécifique.

Pour lancer la BDD associée au Dashboard-server, il me faut utiliser la commande suivante dans l'IDE associé :

```
docker run -e POSTGRES_PASSWORD=... -e POSTGRES_USER=... -p 5433 :5432 -v $PWD/pg-data :/var/lib/postgresql/data postgres :16
```

Le tag `-e` permet de définir les variables d'environnement. Ce sont des valeurs temporairement stockées dans un conteneur, dans le but de modifier le comportement des codes, sans les modifier pour autant. Le tag `-p` permet de lier un port du conteneur à un port de la machine hôte. Ici, 5433 est le port de la machine, tandis que 5432 est le port du container. Enfin, le tag `-v` définit un volume mappé docker (bind mount).

Pour lancer le proxy, j'utilise la commande suivante dans l'IDE associé :

```
docker compose up -d
```

Le tag `-d` permet de dissocier le terminal depuis lesquels ont été lancés les containers docker et leur utilisation. Cela signifie que si l'on ferme le terminal depuis lequel ils ont été lancés, alors les containers continueront d'être exécutés. Sans ce tag, ils auraient été arrêtés à la fermeture du terminal.

Il n'y avait pas de commandes spéciales pour le Dashboard-server, ni le microservice proposal-service. Il me fallait aller dans l'IDE associé à son développement et cliquer sur un bouton run.

Enfin, de manière analogue au lancement de la première BDD, j'ai lancé la BDD du microservice Proposal.

17 L'utilisation de l'IA éthique durant le stage

Lors de mon stage, j'ai pu utiliser l'IA, celle développée par OpenAI, ChatGPT, comme outil d'assistance ponctuelle. Elle a pu m'être utile pour vulgariser des concepts complexes à des fins de compréhension personnelle, en complément des explications fournies par mes collègues ou par les sources trouvées sur internet. Elle m'a aussi été utile lors de problèmes survenus au cours de mon stage, par exemple quand j'ai eu des conflits de ports. Enfin, je l'ai utilisée pour clarifier certaines parties de mon contenu et améliorer la lisibilité de ce rapport.

En somme, l'IA n'a pas été utilisée pour effectuer des analyses critiques, rédiger des sections entières, ni comme source d'information principale. Aucune portion de texte générée par l'IA n'a été copiée directement dans le corps du rapport sans réécriture et validation. Toutes les informations factuelles et citations proviennent de sources vérifiées et référencées.



Institut Laue-Langevin - 71 avenue des Martyrs, Grenoble, 38000

GELINEAU
Arthur

Maths-Info



OBJECTIFS DU STAGE

Le stage a porté sur l'amélioration de la plateforme numérique, nommée Dashboard, en particulier sur le suivi des proposals en attente de filtrage. En tant que développeur IT, j'ai été chargé de développer une tuile permettant de visualiser ces proposals dans l'interface.



MISSIONS EFFECTUÉES ET RÉSULTATS

Missions :

- Mise en place de la tuile dans l'application
- Compréhension des technologies utilisées
- Apprentissage de nouveaux langages de programmation.

Voici une capture d'écran montrant le résultat.

 Proposals awaiting decision
5 proposals

Principal Investigator - Awaiting

4

CoProposer - Awaiting

1

Local Contact - Awaiting

0



COMPÉTENCES DEVELOPPÉES

Cette expérience m'a permis de développer des savoir-faire professionnels essentiels comme la collaboration d'équipe, le travail en autonomie, la gestion des priorités et l'adaptation à un environnement technique complexe. Nous travaillons de manière AGILE.



ENJEUX AUTOUR DES TES

Comme tout projet informatique, celui réalisé lors de mon stage a entraîné une consommation d'énergie liée à l'utilisation d'ordinateurs, de serveurs de développement et d'outils de déploiement. Bien que cette consommation soit difficile à quantifier avec précision, il est crucial de se rappeler qu'un environnement de développement moderne repose sur une infrastructure numérique énergivore. Néanmoins, en dehors de cette consommation, le développement web reste relativement éloigné des enjeux environnementaux, offrant peu de leviers significatifs pour réduire l'empreinte écologique.



Glossaire

- API** Application Programming Interface. 2, 16, 33, 35
- BDD** Base De Données. 2, 9, 10, 17, 21, 22, 30, 31, 35, 38, 39
- CD** Continuous Development. 11, 12
- CI** Continuous Integration. 11, 12
- CRG** Collaborative Research Group. 7
- CSS** Cascade Style Sheet. 12, 14
- DDT** Directors Discretion Time. 6
- EPS** Electronic Proposal Submission. 6–8
- HTML** HyperText Markup Language. 12, 13, 16
- HTTP** HyperText Transfert Protocol. 2, 9, 16–18, 20, 33, 34, 36, 37
- IDE** Integrated Development Environment. 23, 31, 38
- ILL** Institut Laue-Langevin. 1–9, 23, 24, 26, 27, 29
- K8s** Kubernetes. 12, 22–24, 30
- ORM** Object Relational Mapping. 17, 35
- REST** REpresentational State Transfer. 2, 16, 33, 35–37
- SCSS** Sassy Cascade Style Sheet. 14
- SPA** Single Page Application. 12
- UI** User Interface. 9
- URI** Uniform Resource Identifier. 33
- URL** Uniform Resource Locator. 16, 18, 34, 36–38
- VCS** Version Control System. 11

Bibliographie

- [1] Schema made thanks to draw.io, <https://app.diagrams.net/>, accessed May 27, 2025.
- [2] Schema made thanks to draw.io, <https://app.diagrams.net/>, accessed May 28, 2025.
- [3] Schema made thanks to draw.io, <https://app.diagrams.net/>, accessed June 03, 2025.
- [4] Claire D. Les endpoints dans rest. dev.to, 2023. <https://dev.to/kureru/les-endpoints-dans-rest-36ln> - (accessed May 09, 2025).
- [5] Datacamp. Les 20 meilleures commandes git avec des exemples : Guide pratique, 2025. <https://www.datacamp.com/fr/blog/git-commands> - (accessed May 06, 2025).
- [6] Dockerdocs. What is docker?, 2025. <https://docs.docker.com/get-started/docker-overview/> - (accessed May 05, 2025).
- [7] MDN Web Docs. Http request methods. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods> - (accessed May 02, 2025).
- [8] Jean-Michel Doudoux. Hibernate, 1999-2023. <https://www.jmdoudoux.fr/java/dej/chap-hibernate.htm> - accessed May 16, 2025.
- [9] Jean-Michel Doudoux. Les annotations, 1999-2023. <https://www.jmdoudoux.fr/java/dej/chap-annotations.htm> - accessed May 16, 2025.
- [10] Git. What is git? <https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F> - (accessed May 06, 2025).
- [11] Kubernetes. Concepts, 2024. <https://kubernetes.io/fr/docs/concepts/> - accessed June 05, 2025.
- [12] Kubernetes. Overview, 2024. <https://kubernetes.io/docs/concepts/overview/> - (accessed May 07, 2025).

- [13] Institut Laue-Langevin. Associés et membres scientifiques. <https://www.ill.eu/fr/a-propos-de-ill/un-partenariat-international/associes-et-membres-scientifiques> - (accessed May 27, 2025).
- [14] Institut Laue-Langevin. Founders and pioneers. <https://www.ill.eu/fr/about-the-ill/introducing-the-ill/history/founders-and-pioneers> - (accessed May 27, 2025).
- [15] Institut Laue-Langevin. Contacts, 2024. <https://www.ill.eu/fr/for-all-users/contacts> - (accessed June 03, 2025).
- [16] Institut Laue-Langevin. L'ill obtient le feu vert pour prolonger ses activités jusqu'en 2033, 2024. <https://www.ill.eu/fr/news-press-events/news/general-news/lill-obtient-le-feu-vert-pour-prolonger-ses-activites-jusquen-2033> - (accessed May 27, 2025).
- [17] Institut Laue-Langevin. Overview, 2024. <https://www.ill.eu/fr/for-ill-users/colleges/overview> - (accessed June 03, 2025).
- [18] Institut Laue-Langevin. Quelques chiffres, 2024. <https://www.ill.eu/fr/a-propos-de-ill/quest-ce-que-lill/quelques-chiffres> - (accessed June 03, 2025).
- [19] Institut Laue-Langevin. Qu'est ce que l'ill, 2024. <https://www.ill.eu/fr/a-propos-de-ill/quest-ce-que-lill> - (accessed May 27, 2025).
- [20] Institut Laue-Langevin. New proposal, 2025. <https://clubng.ill.eu/userclub/proposals/new/eps> - accessed June 06, 2025.