mistral
From Aptean

Hassou Rayhan
IUT informatique
Clermont-Ferrand
2022-2023

Rapport de stage

Encadrante de stage : Sylvie Guillaume
(Professeur de l'IUT)
Tuteur de stage : Cédric Gouttebroze
(Directeur développement)

UNIVERSITÉ
Clermont Auvergne

J'autorise la diffusion de mon rapport sur l'intranet de l'IUT

REMERCIEMENT

Je tiens à dédier cette section pour remercier toutes les personnes ayant contribué au bon déroulement de mon stage et qui m'ont aidé et soutenu.

Je voudrais remercier tout d'abord l'entièreté de l'équipe de Mistral informatique pour m'avoir accueilli au sein de cette entreprise dynamique, chaleureuse et très professionnelle.

J'aimerai aussi adresser des remerciements plus particulier à Cédric Gouttebroze (Directeur Développement), mon tuteur au sein de l'entreprise, pour m'avoir conseillé lors de ce stage mais aussi Christophe Firpo (Tech lead), Romain Benejam, Allan Jarry, Benoît Damblon, Sacha Jacquet Dachard et toute l'équipe de Développement pour leur bienveillance, leur aide et conseils très précieux.

Enfin, je souhaite remercier ma professeure référente, Mme Sylvie GUILLAUME pour m'avoir suivi et conseillé durant le stage et l'IUT informatique d'Aubière pour les connaissances m'ayant permis le bon déroulement du stage.

INTRODUCTION

Du 9 mai au 10 juillet 2023, j'ai réalisé mon stage de deuxième année de BUT Informatique chez Mistral informatique situé à Clermont-Ferrand au 39 Rue Georges Besse. J'ai pu rencontrer une première fois cette entreprise lors du forum des stages se tenant à l'IUT puis j'ai eu la chance par la suite de faire un entretien et d'obtenir mon stage.

L'entreprise dispose d'une application à usage interne nommée PreMis servant aux différents pôles de l'entreprise. Elle est aussi utilisée pour tester de nouvelles fonctionnalités. C'est dans cette optique que Cédric Gouttebroze m'a confié la tâche de créer un tableau de bord personnalisable offrant la capacité aux utilisateurs d'avoir un outil correspondant à leurs besoins.

Tout d'abord, je commencerai par présenter l'établissement et son organisation, puis je détaillerai les objectifs du stage. Suite à cela, je présenterai une partie analyse dans laquelle j'exposerai l'étude et la conception préalable à la réalisation pour ensuite présenter le développement de l'application en lui-même. Enfin, je dresserai un bilan technique et personnel de ce stage.

PRESENTATION DE L'ENTREPRISE

I. Présentation générale

Le groupe mistral est un éditeur de progiciels de gestion intégré (ERP) crée en 1980 à Clermont-Ferrand, s'adressant à des concessionnaires, distributeurs, loueurs, importateurs et réparateurs de matériel et équipement pour les domaines agricole, manutentionnaire, industriel ou dans le bâtiment.

Fort d'une expérience de plus de quarante ans dans ce domaine, le Groupe Mistral emploie aujourd'hui plus de 80 personnes et accompagne plus de 550 clients, représentant près de 15 000 utilisateurs en France et à l'étranger. Ses solutions ERP et sa gamme complète d'applications mobiles sont devenues des références pour l'ensemble du secteur et sont utilisées par des entreprises de toutes tailles, des TPE/PME jusqu'aux groupes nationaux et internationaux.

Figure 1 : Locaux de mistral

II. Présentation de l’environnement Mistral

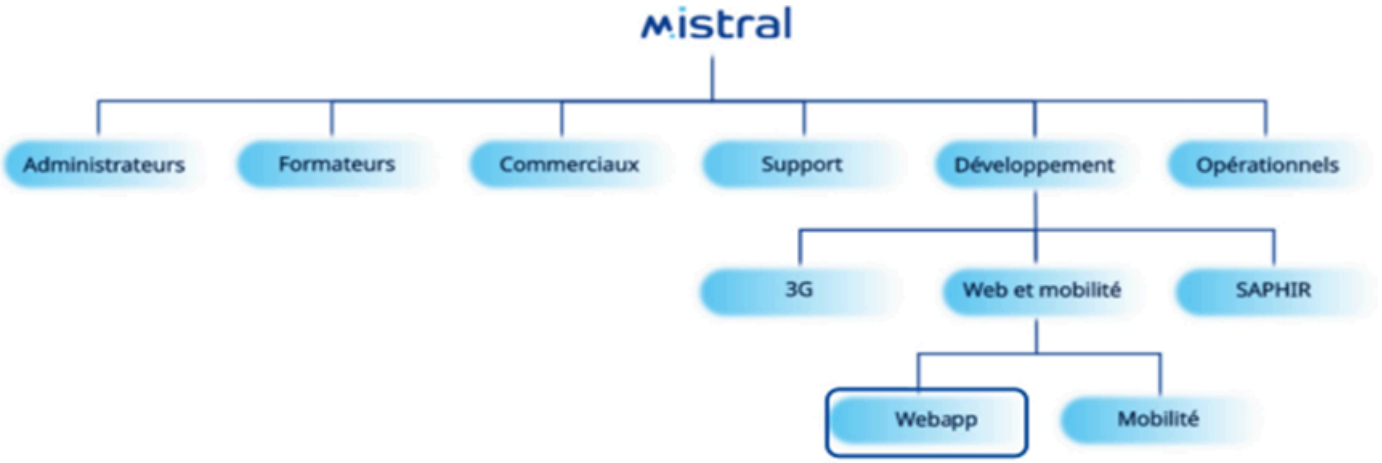


Figure 2 : Schéma de l’organisation de MISTRAL INFORMATIQUE

L'équipe de développement de la société se compose en deux pôles (le pôle web/mobilité et ERP) qui fonctionnent en tandem au sein de l'organisation, avec Christophe FIRPO en tant que Tech lead et Cédric Gouttebroze en tant que directeur Développement.

PreMis étant une application Web, j’ai rejoint le pôle Web/mobilité. Par conséquent, les activités de mon équipe diffèrent grandement de celles de l'équipe de Développement mobile. Néanmoins, nous nous engageons dans une communication quotidienne lors des "daily meeting*" Cette méthode, nommée LEAN & AGILE*, encourage chaque membre de l'équipe à partager son état d'esprit actuel, appelé "team mood", ainsi que ses activités de la veille et ses plans pour la journée à venir pendant 20 minutes. Cette méthode permet un historique continu de l'avancement des projets à l'ensemble de l'équipe et offre aux membres de l'équipe la possibilité de s'entraider si nécessaire.

Ces réunions ne sont pas la seule forme de communication utilisée. Une fois par semaine, des "points Misframes" sont organisés où de petites équipes peuvent discuter de points techniques importants pour faciliter le partage, l'expansion et la réutilisation des connaissances. J’ai eu l’occasion de participer à l’une d’entre elles.

Ces points offrent également une perspective à plus long terme sur les actions, les projets et les évolutions significatives à mettre en œuvre, avec des tâches priorisées et des contributeurs assignés pour chaque sujet.

Lean et agile* La méthode Lean et Agile sont deux approches de gestion de projet et de développement logiciel qui visent à améliorer l'efficacité, la flexibilité et la collaboration dans le processus de travail.
Daily meeting* : Réunion quotidienne d’une durée de 15 minutes environ permettant d’informer à ses coéquipiers son avancement, son humeur, les difficultés rencontrées et ce que l’on a prévu aujourd’hui.

III. Présentation de PreMis

PreMis est une application web interne utilisée pour gérer la formation client et les différentes tâches assignées aux employés. Ces tâches peuvent être regroupées en trois types : les anomalies, les évolutions et les bons de préparation. Tout au long de ce rapport, nous utiliserons le terme "tâches" pour les désigner. Tout au long de ce rapport, le terme "tâches" sera utilisé pour englober ces trois catégories, car elles représentent les principales responsabilités et activités au sein de PreMis

PreMis permet aux employés d'accéder et de gérer efficacement les tâches qui leur sont assignées. L'application propose des fonctionnalités spécifiques conçues pour traiter ces trois types de tâches.

1. Anomalies : Cette catégorie concerne les problèmes ou les dysfonctionnements inattendus rencontrés dans le système. Elle inclut l'identification et la résolution de bugs, d'erreurs ou de problèmes logiciels. Les tâches liées aux anomalies visent à résoudre ces problèmes et à assurer le bon fonctionnement de l'application.

2. Évolutions : Ces tâches consistent à apporter des améliorations ou des modifications aux fonctionnalités existantes des applications au sein de Mistral. Elles peuvent inclure l'ajout de nouvelles fonctionnalités, l'amélioration de l'expérience utilisateur, l'optimisation des performances ou la mise en œuvre d'améliorations demandées par le clients. L'objectif des évolutions est d'améliorer en continu l'application en fonction des retours des utilisateurs et des besoins évolutifs.

3. Bons de préparation : Ensemble des actions à faire pour satisfaire une commande client (livraison du nodule logiciel, du hardware ou les développements spécifique)

Figure 3 : Page d'accueil de PreMis

PRESENTATION SYNTHETIQUE DU STAGE

I. Existant et objectif du projet

Dans le tableau ci-dessous, nous présentons par exemple les anomalies sous forme de regroupements, incluant toutes les anomalies affectées. Pour accéder aux informations qui nous intéressent, il est nécessaire d'appliquer des filtres spécifiques à chaque type de tâche, que ce soit les bons de préparation, les anomalies ou les évolutions.

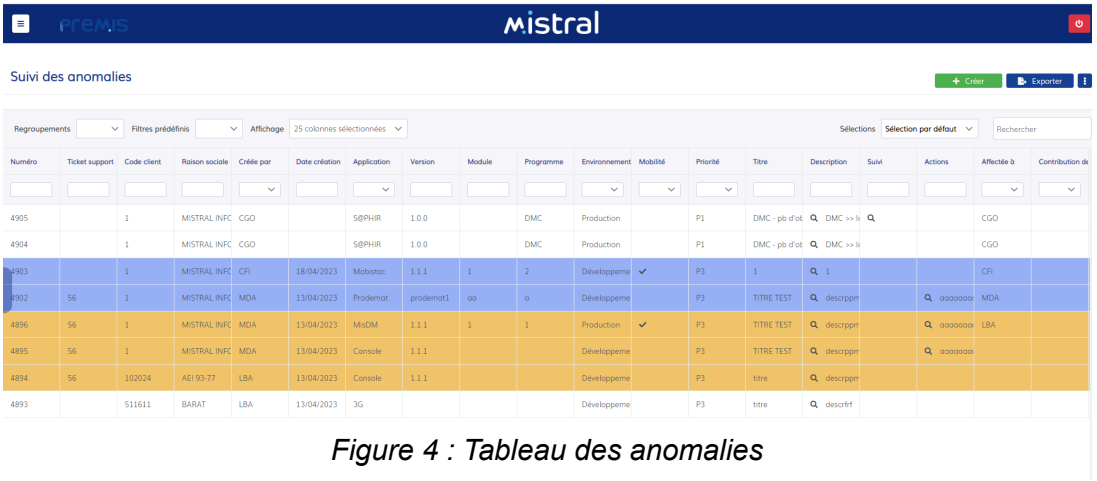


Figure 4 : Tableau des anomalies

(voir le tableau en plus grand dans l'annexe numéro 4 page 43)

L'objectif principal est de fournir aux utilisateurs un accès direct à l'ensemble de leurs tâches, regroupées au même endroit, afin de faciliter la lisibilité et d'améliorer l'efficacité du personnel de Mistral. Cette approche vise à éliminer la nécessité de naviguer fréquemment entre différents onglets et d'appliquer des filtres à un tableau pour visualiser leurs tâches.

En centralisant toutes les tâches dans un seul emplacement, nous offrons aux utilisateurs une vue d'ensemble pratique et complète de leurs responsabilités. Nous facilitons également la gestion et le suivi des différentes activités. Les utilisateurs peuvent rapidement repérer les tâches en attente, les tâches en cours ou celles qui nécessitent une attention particulière, sans avoir à passer d'un onglet à l'autre ou à appliquer des filtres complexes.

Cette solution contribue à optimiser l'expérience utilisateur en simplifiant l'interface et en réduisant le temps nécessaire pour accéder aux informations pertinentes. Les utilisateurs de Mistral peuvent ainsi gérer efficacement leurs tâches, améliorer leur productivité et se concentrer sur les priorités, sans être encombrés par des étapes

supplémentaires et des manipulations fastidieuses.Cela favorise une meilleure gestion des tâches, une productivité accrue et une expérience utilisateur optimisée.

Ma mission est symbolisé par ce diagramme :

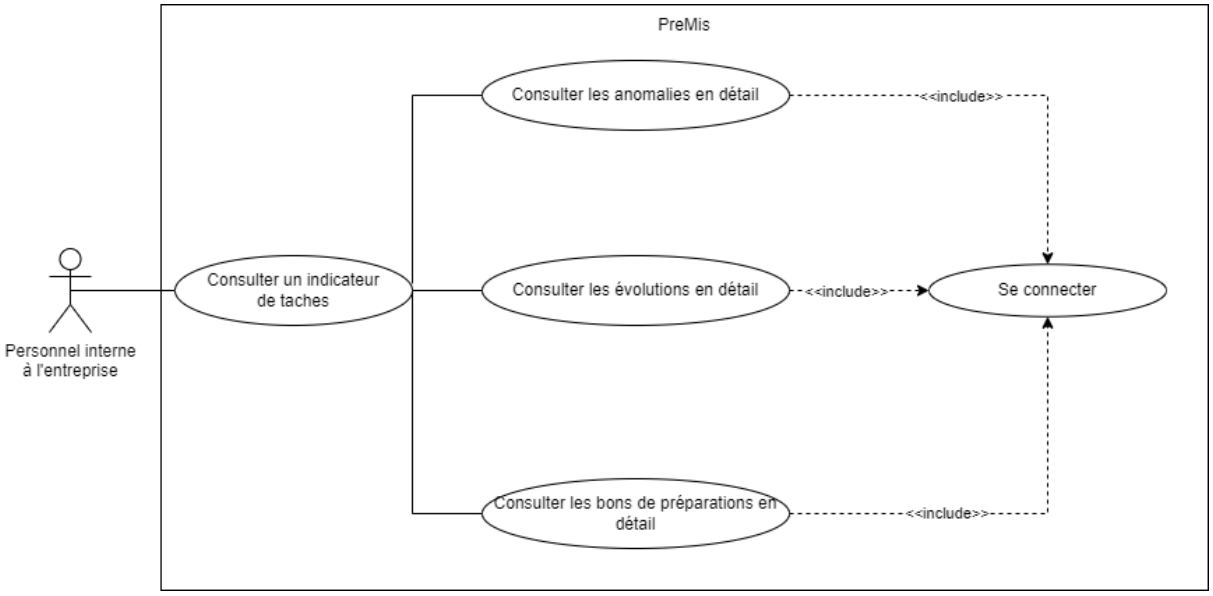


Figure 5 : Diagramme de cas d'utilisations de la fonctionnalité

Nom	Consulter ses tâches
Objectif d'un acteur	Avoir toutes les informations sur les différentes tâches que l'acteur doit réaliser.
Acteur principal	Personnel interne à l'entreprise.
Conditions initiales	Se connecter.
Scénario d'utilisation	Un employé souhaite voir rapidement les différentes tâches qui lui incombent.
Condition de fin	Lorsque l'utilisateur a terminé, il ferme la page.

II. Environnement matériel et logiciels

Pour ce projet j'ai utilisé des technologies spécifiques que je vais vous présenter ci-dessous :

A) Front*

Angular : Angular est un framework open source basé sur TypeScript crée en 2016 par Google et une communauté de particuliers. Il permet de faire du front avec du HTML/CSS et du back avec du JavaScript/TypeScript. Ici, nous l'utilisons seulement pour le front.

PrimeNG : Pour les UI nous utilisons une librairies open-source qui permet donc d'avoir une identité visuelle identique sur toutes les applications de l'environnement Mistral grâce à la surcouche ajoutée par les équipes de Développement. Cette librairie à été choisi pour son large choix d'UI et pour sa facilité d'utilisation

Visual Studio Code : Visual studio code est un IDE* développé par Microsoft pour Windows, Linux et macOS. Les fonctionnalités incluent la prise en charge du débogage, la mise en évidence de la syntaxe, la complétion intelligente du code, les snippets, la refactorisation du code et Git intégré. Il inclut directement le TypeScript ce qui est très pratique lorsque l'on utilise Angular.

B) Back*

Java : Java est un langage orienté objet de haut niveau largement répandu de nos jours car très performant et facile à utiliser. Il est utilisé pour le back car très performant.

Spring Boot : Spring Boot est un outil permettant de simplifier le développement d'applications Java en fournissant une configuration par défaut intelligente, un serveur web embarqué et des fonctionnalités de gestion de la configuration. Il est populaire pour le développement de microservices et d'applications autonomes en Java. Nous l'utilisons car il nous facilite la communication avec le serveur

Idea : IntelliJ IDEA est un environnement de Développement intégré (en anglais Integrated Development Environment - IDE) destiné au développement de logiciels informatiques reposant sur la technologie Java. Il est développé par JetBrains et disponible en deux versions, l'une communautaire, open source, sous licence Apache 2 et l'autre propriétaire, protégée par une licence commerciale.

Back* : Correspond à la partie technique d'une application, d'un site etc. C'est la partie où nous allons créer les fonctionnalités

Front* : Partie visuelle de l'application ou du site.

(Voir annexe 9 page 48)

III. Gantt prévisionnel

J'ai réalisé un diagramme de Gantt à partir des tâches que j'ai déterminé ce qui me facilitera l'organisation de mes journées et de mon temps (cf annexe 2 page 41)

Les tâches sont séparées en 2 parties, la partie Front et Back, cette séparation me permettra de me familiariser avec Angular, de comprendre comment fonctionnent les graphes PrimeNg et prendre en main l'environnement de travail.

Conception : Conception et documentation pour préparer au mieux la phase de Développement

Front - Tâche : Développement de la partie de graphique des tâches sans fonctionnalités

Front - Graphiques : Développement de la partie graphique des différents graphes, correspond au graphique affiché lorsque l'utilisateur clique sur les Tâches.

Front - Paramètres : Développement de la partie graphique des paramètres qui permettront de changer la couleur des graphes ou leur type.

Front - Tableau : Développement de la partie graphique des tableaux qui seront affichés lorsque l'utilisateur clique sur les graphiques.

Front - Page déroulante : Développement de la partie page déroulante, la partie la plus dure du front car n'étant pas implémenté directement sur PrimeNg. Elle apparaît quand plusieurs tableaux et graphiques sont affichés.

Back - Requête SQL : Développement des requêtes sql permettant de récupérer les informations depuis la base de données. Ces requêtes seront utilisées par les services.

Back - Service : Développement des services qui permettent de récupérer les informations brut données par les requêtes sql, de les traiter et de les ranger dans les classes qui seront envoyées au contrôleur.

Back - Contrôleur : Développement des contrôleurs permettant d'envoyer une réponse HTTP en envoyant les instances des données gérées par les services.

Liaison Back-Front : La partie la plus longue qui consiste à traiter les données reçues pour les afficher dans le front mais cela consiste aussi à traiter les informations lors de changement dû à l'activité de l'utilisateur.

IV. Gantt réel

Au terme du stage, j'ai fait le diagramme de Gantt réel pour représenter les tâches réalisées (cf annexe 3 page 42). Nous pouvons constater des différences entre le planning prévisionnel et le planning réel sur les tâches suivantes :

Liaison Back-Front : La différence s'explique par un changement d'organisation au cours du projet. A la base je voulais coder toute la partie graphique d'un coup puis m'attaquer au back mais m'étant rendu compte que cela me poserait énormément de problèmes et de réécritures de codes. C'est donc pour cela qu'au lieu de faire cette partie tout à la fin, je les fais au fur et à mesure.

Back : Le retard et la séparation en 2 de cette partie est causé par l'ajout de l'enregistrement des paramètres de l'utilisateur. A la base, je voulais que ces informations soient enregistrés dans le local storage* ce qui consiste un travail de développement négligeable. Cependant, après discussion avec Christophe Firpo (Tech lead), qui m'a expliqué que les employés de Mistral ne travaille pas forcément sur leur ordinateur fourni par l'entreprise, j'ai du modifier mon code pour enregistrer les paramètres en base de données ce qui m'a ajouté une certaine couche de complexité supplémentaire. La programmation du Back à aussi été impactée par le changement d'organisation, j'ai donc codé au fur et à mesure en parallèle du front et de la liaison entre les deux.

Front : Cette partie à pris du retard pour 2 raisons. La première est ,comme indiqué dans la partie liaison Back-Front, un changement d'organisation, le front n'a donc pas été codé d'un coup mais au fur et à mesure de l'avancement du projet. La deuxième est le besoin de réécrire de nombreux composants que je devais utiliser complexifiant la tâche et augmentant la durée de celle-ci.

En conclusion, la différence majeure est que le travail n'a pas été fait partie par partie mais tout en parallèle.

Avant de commencer le développement de l'application, une phase d'analyse a été réalisée afin d'identifier les besoins des utilisateurs, de comprendre les différents types de données à manipuler et de définir les vues de l'interface.

I. Détails des données existantes

Les différentes tâches ayant des structures différentes, cela peut poser problème pour la maintenabilité du code et si par la suite nous venions à ajouter un nouveau type de tâche. Pour régler ce problème, il est impératif de généraliser les données que nous recevons dans une classe commune.

Les différentes tâches se présentent ainsi :

Anomalies :

Id	Titre	Commentaire	Statut développement	Statut support

Evolutions :

Id	Version	Titre	Statut développement	Statut support	Statut Comité

Bons de préparation :

Code Agent	Associé	Statut	Titre	Attribué à

Les tâches présentent des similitudes mais sont néanmoins très différentes, il est donc impossible de les généraliser en l'état. Cependant, nous ne voulons pas afficher toutes les données, le but étant simplement de permettre à l'utilisateur de rapidement savoir ce qu'ils ont à faire. Il y a donc de nombreuses informations que l'on peut enlever.

Les données que l'on veut afficher sont seulement le titre de la tâche, la liste des statuts et le nombre de tâche par statuts tout ceci sous le nom de "Graphe" :

Titre	Liste de Statut	Nombre de tâches

Nous avons donc 3 sources de données qui vont converger vers une seule classe puis être envoyées à la partie front afin d'être affichées de la même manière. Si, par la suite, nous venions à ajouter un nouveau type de donnée,il serait possible de le faire sans modifier le code côté front-end offrant un gain de temps énorme et une maintenabilité du code assez grande.

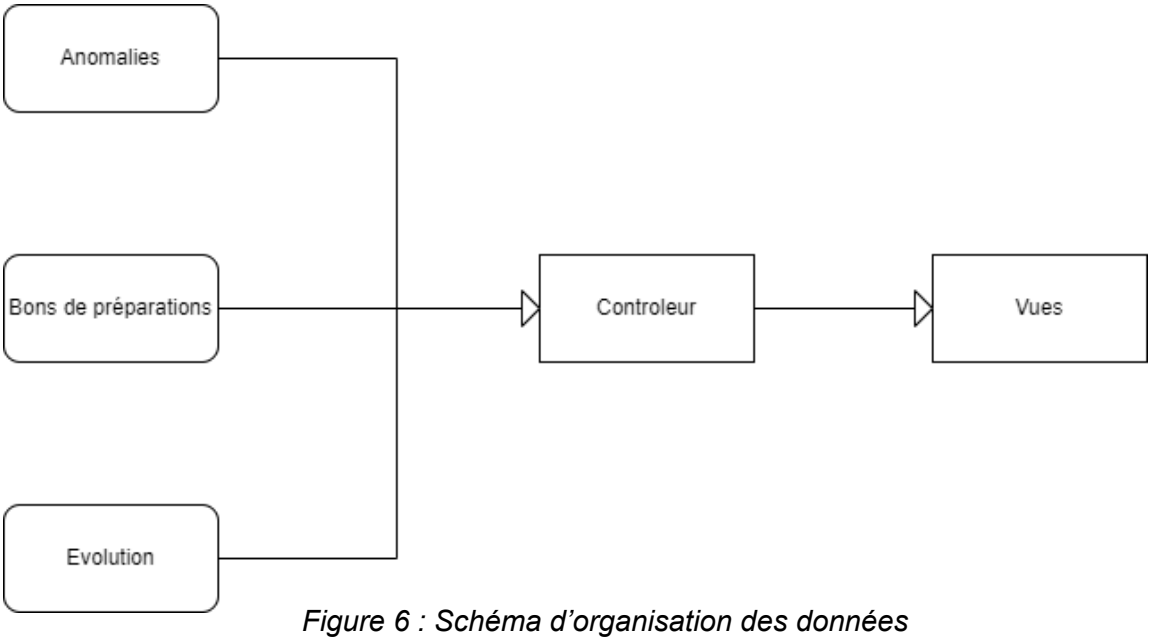


Figure 6 : Schéma d'organisation des données

Cette solution permettra également de créer un code plus propre en front car cela permettra de créer un seul composant au lieu de plusieurs. Cela offre l'occasion donc de créer un composant réutilisable dans d'autres applications, s'intégrant parfaitement à la méthodologie lean et le maintien de l'écosystème Mistral mise en place.

II. Définition des vues de l'interface

Le besoin étant nouveau, je me suis lancé dans une phase de conception de maquettes afin de mettre à plat les différentes idées que nous pourrions avoir, me permettant ainsi d'en tirer un premier aperçu (maquette entière en annexe numéro 1, page 40). La page est constituée de trois graphes permettant à l'utilisateur de voir le nombre de tâches qu'il a à faire. S'il clique sur l'une d'entre elles, une nouvelle page s'ajoute en dessous, lui permettant de voir plus de détails. Comme évoqué précédemment, si un nouveau type de tâches venait à s'ajouter, la page sera automatiquement mise à jour et un nouveau graphe s'ajoutera sans avoir besoin d'effectuer des développements complémentaires.

Figure 7 : Partie de maquette

J'ai donc pu produire un prototype qui à été validé (cf Annexes numéro 1 page 40). Voici un storyboard résumant le fonctionnement :

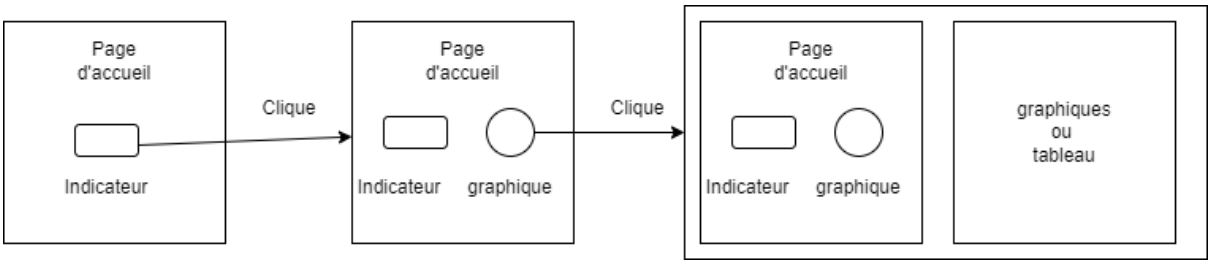


Figure 8 : Storyboard simplifié du fonctionnement du tableau de bord

Lorsqu'on clique sur un indicateur, un graphe apparaît en dessous des indicateurs déjà visibles, cependant si l'utilisateur sélectionne différentes formes de graphe une nouvelle page se créera en dessous de la précédente.
Un menu apparaît sur le bord droit de l'écran permettant à l'utilisateur de naviguer entre ce qu'on appellera désormais les "slides". Pour finir, si l'utilisateur clique sur un graphe, le tableau correspondant s'ajoute à une nouvelle slide lui permettant de voir encore plus en détail. (pour mieux comprendre le concept de slide, se référer à l'annexe 8 page 47)

III. Diagramme de séquence

Premis étant une application plutôt complexe, j'ai eu du mal à me repérer dans les centaines de classes, de modèles et de contrôleurs la composant. J'ai donc décidé d'élaborer un diagramme de séquence avec l'aide du Tech lead Christophe Firpo afin de mieux comprendre l'architecture de l'application et pouvoir par la suite organiser mes classes correctement. Il est basé sur le chemin parcouru pour afficher les anomalies dans la page "Anomalies".

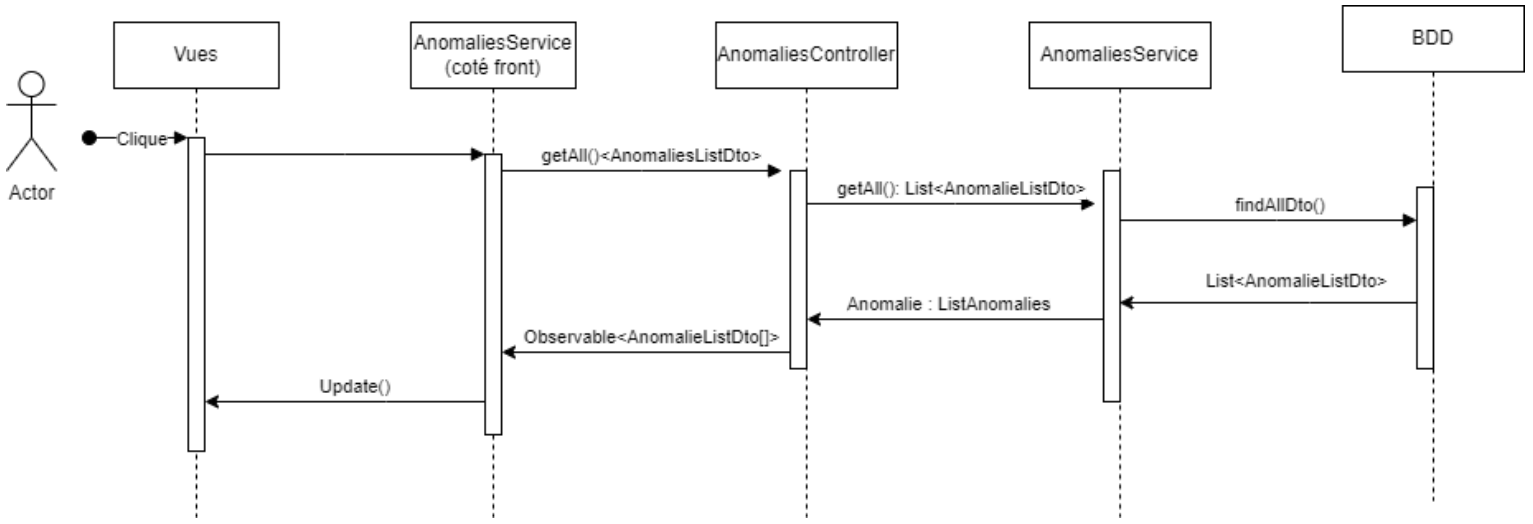


Figure 9 : Diagramme de séquence des anomalies

Le diagramme comporte plusieurs acteurs que nous allons détailler ici :

Vues : Correspond à l'application, à ce que voit l'utilisateur. Lorsque l'utilisateur clique sur la page des anomalies, l'entité ***"AnomaliesService"*** (côté front) se lance.

AnomaliesService (côté front): Ce service se trouve dans la partie front, c'est donc une classe Angular. Elle sert à récupérer la liste d'anomalie et la donner aux vues pour lui permettre de l'afficher. Elle va appeler sa fonction getAll() qui retourne une liste d'AnomalieListDTO (une classe qui permet de stocker les entités que l'on reçoit de la base de données). Dans cette fonction on retrouve une requête HTTP get vers la classe ***"AnomaliesContrôleur"*** permettant de récupérer une liste d'anomalies. Le chemin vers cette fonction est précisé avec l'interpolation ('${this.baseUrl}')

```
getAll(): Observable<AnomalieListDto[]> {
  return this.http.get<AnomalieListDto[]>(`${this.baseUrl}`);
}
```

Figure 10 : Fonction getAll de la classe AnomaliesService

AnomaliesContrôleur : Située dans la partie back de l'application, elle va permettre d'encapsuler les données reçues dans une responseEntity* et de renvoyer une réponse HTTP (qui vont être récupérées par la classe ***"AnomaliesService"***). Pour récupérer ces données, la classe va faire appel à la fonction findAllDto() de la classe ***"AnomaliesService"*** (côté back).

```
@GetMapping
public ResponseEntity<List<AnomalieListDTO>> getAll() {
    return new ResponseEntity<>(this.anomalieService.findAllDto(), new HttpHeaders(), HttpStatus.OK);
}
```

Figure 11 : Fonction getAll de la classe AnomaliesContrôleur

AnomaliesService (côté back) : Cette méthode est celle qui va récupérer depuis la base de données les anomalies et qui va les stocker dans une liste et les renvoyer.

Cette analyse m'a permis de créer un diagramme de séquence mais en utilisant les classes que j'allais devoir crée et utiliser. Le voici :

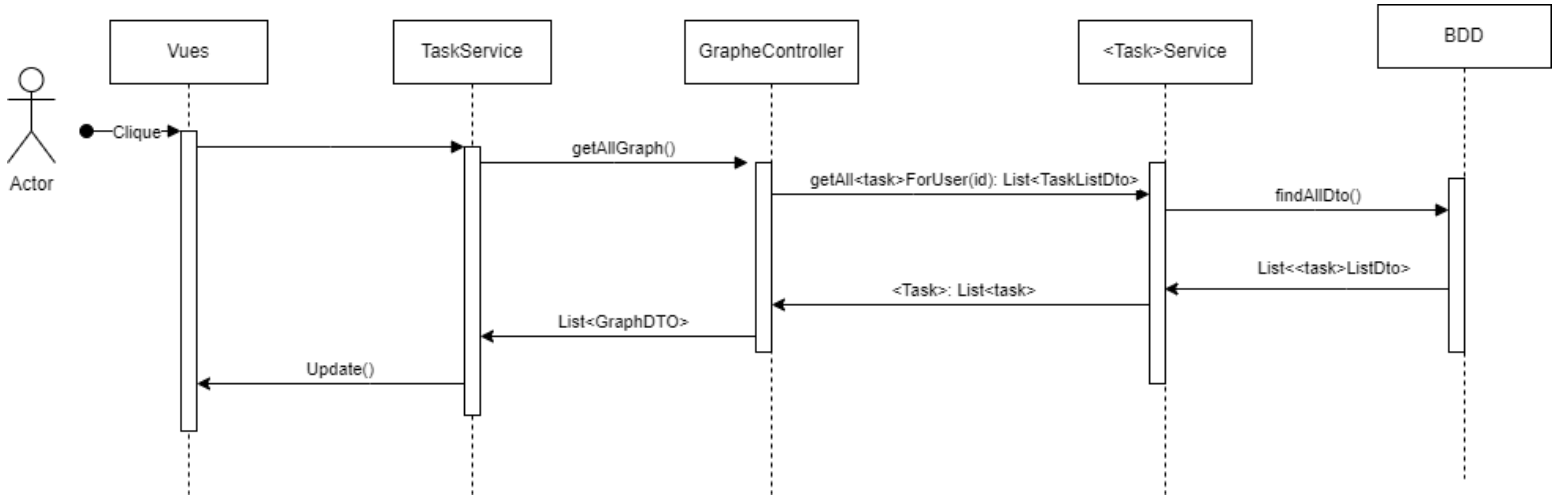


Figure 12 : Diagramme de séquence des graphes

C'est le même fonctionnement que précisé ci-dessous à la différence que le grapheContrôleur retournera une liste de GraphDTO.
Notons que le <Task>Service indique qu'il y aura autant de service que de type de tâche à récupérer.

Response Entity* : Objet utilisé pour stocker des données qui seront envoyé par une réponse HTTP

I. Installation

Avant de commencer le développement, j'ai dû configurer mon environnement de travail en installant plusieurs outils et plugins. Pour utiliser Angular, par exemple, j'ai installé Node.js, qui me permet d'utiliser la commande npm pour installer les modules JavaScript/TypeScript requis. J'ai également ajouté des plugins spécifiques dans IntelliJ et Visual Studio Code, tels que Peacock et Project-Color, qui colorent les environnements de développement en fonction du projet sur lequel je travaille (par exemple, Azure blue pour PreMis).

Ensuite, j'ai procédé à la configuration du registre npm en exécutant la commande suivante : npm config set registry http://nexus.mistral.fr:8081/repository/MAngular/. Cette étape était nécessaire pour installer les bibliothèques Mistral nécessaires au Développement. Cette bibliothèque contient les différents composants permettant de respecter la charte graphique Mistral.

Ces étapes de configuration étaient indispensables pour mettre en place un environnement de Développement fonctionnel et prêt à être utilisé pour la réalisation de la fonctionnalité.

II. Conventions de développement Mistral

J’avais quelques règles à respecter lorsque je codais que voici :

1) Langue

Les noms des répertoires, composants, services, modèles, variables et méthodes doivent être anglais.

Les commentaires et la documentation sont écrits en français.

2) Nommage des composants/classes/entités

Les composants/classes/entités doivent être nommés en suivant la règle : thème – sujet – type – action.

Exemple :

Angular	Java
event-card-create	EventCardCreate

3) Conventions d'écritures

Le nommage doit respecter certains principes :

- Un nom permettant de l'identifier simplement.
- Un type permettant de connaître son utilité sans forcément devoir "éplucher" son contenu.
- Éventuellement être placé dans un dossier qui le regroupe avec ces "confrères" (les services ensembles, les modèles ensemble, etc).
- Le nom du fichier sera en kekab-case. (first-name)
- Le nom de l'objet sera lui en PascalCase. (FirstName)
- Les attributs de l'objet seront en camelCase. (firstName)
- Les énumérations et constantes utilisent la convention snake_case (All Caps).

4) Écriture des méthodes

Le nom de la méthode doit être explicite sur l'action qu'effectue celle-ci. Pour l’écriture du code des méthodes, celui devra respecter la règle de la single responsability : la méthode ne doit faire qu’un seul traitement. Si un traitement se fait en plusieurs étapes, il faut donc décomposer la méthode en sous-méthodes. Cette décomposition permettra de faciliter la relecture, la maintenance et les tests unitaires du code.

III. Front

1) Décomposition en composants

Angular permet de créer des composants et de les appeler dans d'autres. J'ai donc utilisé cette fonctionnalité pour avoir du code HTML/CSS moins lourd et plus spécifique.

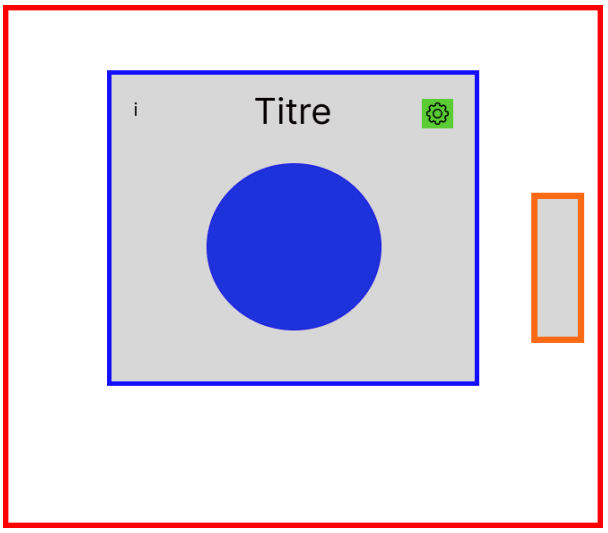


Figure 13 : Décomposition de la page principale du tableau de bord

Dahsboard.component :
C'est le composant principal qui servira de réceptacle pour tous les autres composants. C'est sur celui-ci que l'utilisateur arrive. Il appelle donc les composants dashboard.task.component et Dashboard.menu.

Dashboard.task.component :
C'est le composant montrant le graphique et le nombre de tâches. Il est appelé par le dashboard.component. Il est appelé grâce à une boucle donc il y'en aura autant que de données que le Dashboard.component reçoit mais nous détaillerons cela dans la partie Back. Il appelle un autre composant qui est le Dashboard-task-parameter.component
Ce composant sera appelé grâce à une boucle, il y'en aura donc autant que le front reçoit de graphe.

Dashboard.taskparameter.component : Button ouvrant en modal un composant faisant office de paramètre. Sur celui-ci, l'utilisateur à la possibilité de changer la couleur du graphe sélectionné.

Dashboard.menu.component :
Ce composant est le menu qui permettra à l'utilisateur de naviguer entre les différentes "slides". Il sera toujours placé sur la droite.

2) Hiérarchisation des composants

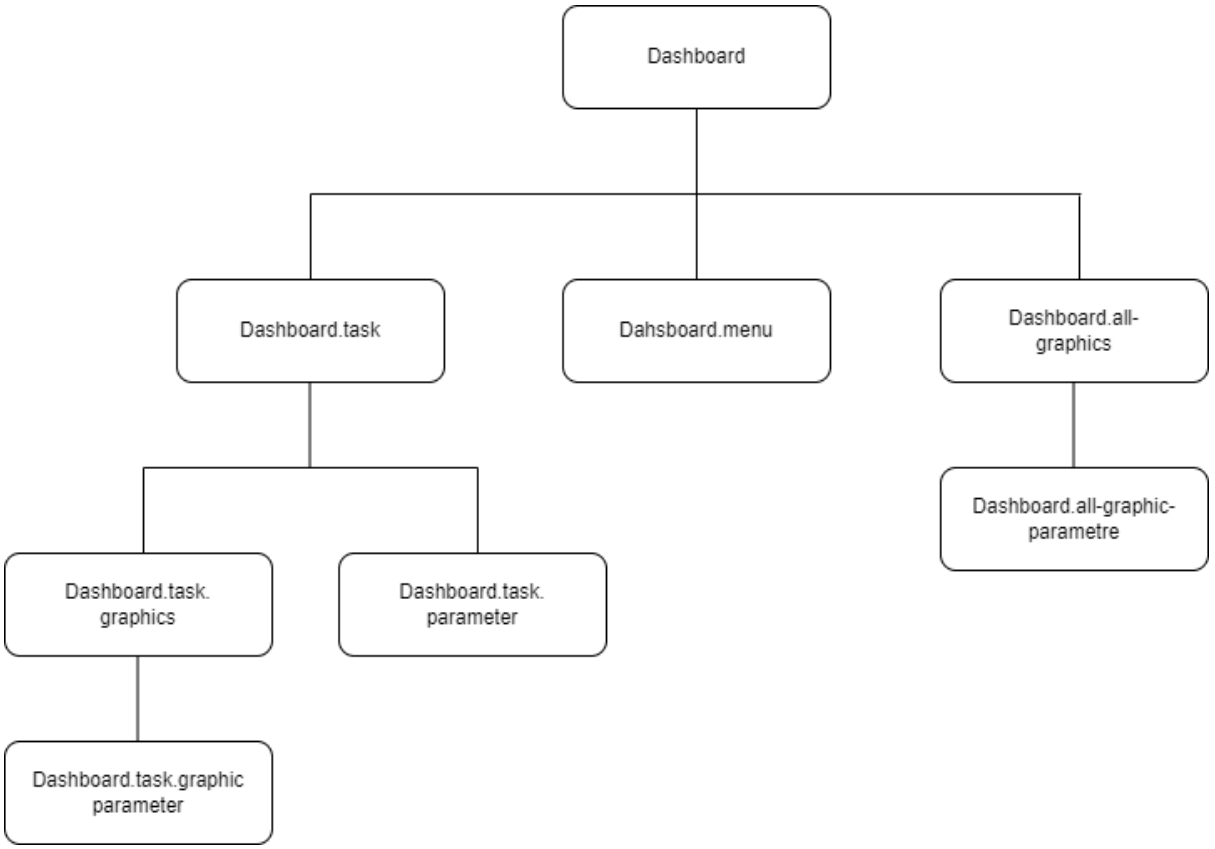


Figure 14 : Lien entre les composants

Voici un schéma simplifié montrant la hiérarchisation globale de tous les composants utilisés pour la fonctionnalité. Chaque composant à un composant parent sauf le composant principal Dashboard. Les composants parents appellent leurs composants enfants mais ils restent pour la plupart indépendants et peuvent être utilisés. Pour preuve, pour les tableaux, je réutilise un composant déjà existant et les composants de graphes pourront être intégrés à d'autre application sans problème. Certains communiquent avec les autres en recevant et/ou en envoyant des données. Cependant je n'ai pas pu réutiliser les composants de paramètres plusieurs fois car chaque composant à des critères de modifications différentes. Vous retrouverez la version complète avec les données transférées en annexe (cf annexe numéro 5 page 44)

IV. Back

1) Repository

Pour récupérer mes données j'utilise des requêtes SQL décrites dans des repository uniques à chaque classe.

Un repository est un concept couramment utilisé en Développement logiciel pour stocker, récupérer et manipuler des données. Il sert de couche intermédiaire entre l'application et la source de données (comme une base de données) en encapsulant* la logique d'accès aux données.

Le but principal d'un repository est de fournir une abstraction entre le code métier de l'application et les détails spécifiques de stockage et d'accès aux données. Il facilite le travail avec les données en fournissant des méthodes et des opérations prédéfinies pour effectuer des opérations courantes telles que la création, la lecture, la mise à jour et la suppression (CRUD)*.

Une requête SQL est découpée de la sorte :

- La requête avec la balise @Query.

```
@Query("SELECT DISTINCT a.statutDev " +
        "FROM Anomalie a " +
        "WHERE a.affecte.initiales = :trigramme")
```

Figure 15 : Requête SQL utilisant la balise @Query

- Le nom de la fonction permettant de l'appeler et précisant le type de retour et les attributs que demande la requête.

```
List<String> findDistinctStatutSupportForUser(@Param("trigramme") String trigramme);
```

Figure 16 : Déclaration du nom de la fonction permettant l'utilisation de la requête SQL

Ces requêtes seront appelées dans les services. A noter que Spring Boot offre beaucoup de possibilités de personnalisation mais au vu des données que j'ai eu à récupérer, de simples requêtes SQL m'ont suffi.
J'ai dû créer une classe repository pour chacune de mes classes (Anomalies, Évolutions, Bons de préparation et GrapheParam). La plupart du temps je récupère les différents statuts en fonction de l'utilisateur passé en paramètre puis je récupère les différentes tâches en fonction du statut passé en paramètre ce qui me permet d'avoir des listes de tâches triées par statut pour l'utilisateur connecté.

Les tables avec lesquels je communique sont les suivantes :

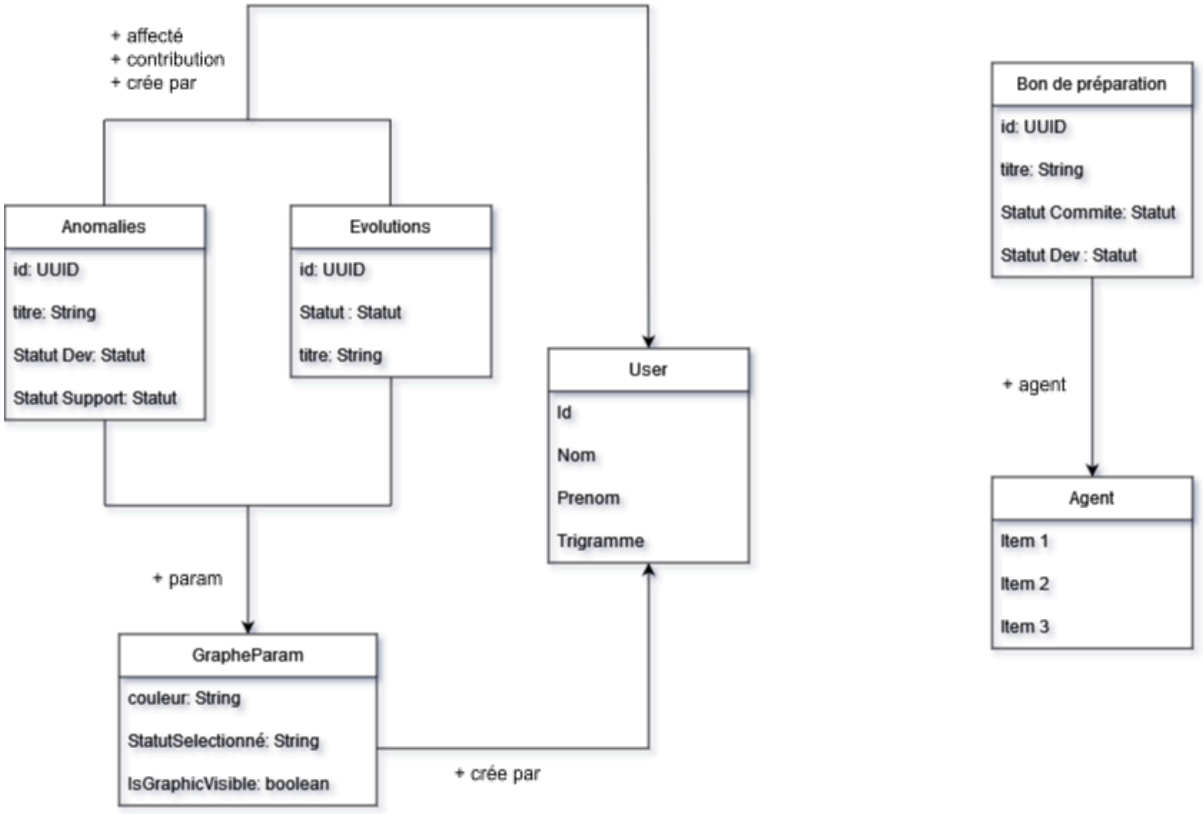


Figure 17 : Diagramme de classe (seulement les classes utilisées)

Les classes Anomalies, Évolutions et GrapheParam (une classe crée pour enregistrer les paramètres utilisateur cf page 34) possèdent toutes un attribut "affecté" qui représente un utilisateur. De plus, les classes Évolutions et GrapheParam sont liées par un attribut nommé "paramètre", qui est de type GrapheParam. En revanche, la classe Bon de préparations n'est pas directement liée à la classe Utilisateur, mais elle est associée à la classe Agent.

J'ai travaillé avec ces six classes pour recueillir toutes les informations nécessaires afin de garantir le bon fonctionnement du tableau de bord.

Les différentes requêtes SQL utilisés se présentent de la sorte :

GrapheParamService : Utilisation de la table GrapheParam

Nom	findByUserTrigram
Objet retourné	GrapheParam
Description	Récupère le grapheParam lorsque le titre correspond au graphe passé en paramètre et que l'utilisateur correspond à l'utilisateur connecté.

AnomalieService et EvolutionService : Utilisation de la table Anomalie/Evolution
Ces 2 services possèdent quasiment les mêmes requêtes Sql ayant un mode de fonctionnement très proche a la différence que les Évolutions ont une liste de Statut en plus, les statuts de comité.

Nom	getTaskForUserAndStatut
Objet retourné	Liste de tâches (Anomalies, Évolutions ou bons de préparation).
Description	Prends en paramètres le trigramme de l'utilisateur connecté et le titre d'un statut pour récupérer toutes les tâches qui ont pour statut celui passé en paramètres.

Nom	findDistinctStatut
Objet retourné	Liste de String.
Description	Prends en paramètre le trigramme de l'utilisateur connecté et récupère tous les statuts des tâches qui lui sont affectées.

Bons de préparation Service : Utilisation de la table Bons de préparation, Agent et User.

Pour récupérer ces différents graphes, lors de l'appel des différentes requêtes nécessaire à la récupération des données, le trigramme de l'utilisateur connecté est passé en paramètre (l'ID qui permet de différencier les utilisateurs), permettant ainsi lors des requêtes SQL de comparer ce trigramme à celui de l'utilisateur affecté aux anomalies/evolution. Le problème étant que la classe Bons de préparation ne possède pas d'utilisateur affecté mais un code agent pour permettre de reconnaître l'utilisateur.
Pour cette classe j'ai dû ajouter une surcouche de vérification permettant de récupérer le code agent à partir du trigramme. Ici ne seront présentées que les requêtes SQL qui diffèrent de celles que l'on peut retrouver dans EvolutionService ou AnomalieService, le trigram étant simplement remplacé par le code agent.

Nom	getNameWithTrigram
Objet retourné	Un String.
Description	Récupére le nom de la personne connectée avec le trigramme dans la table User.

Nom	getCodeAgentWithName
Objet retourné	Un String.
Description	Récupère le code agent de la personne connecté grâce à son nom dans la table Agent.

CRUD* : acronyme qui représente les opérations de base effectuées sur des données persistantes dans un système informatique. Il est largement utilisé dans le développement d'applications et fait référence aux quatre principales opérations : Create (Créer), Read (Lire), Update (Mettre à jour) et Delete (Supprimer). Ces opérations sont généralement associées à des fonctionnalités de manipulation des données dans une base de données ou un système de stockage.
Système de persistance des données* : Permet de stocker et de récupérer des données de manière durable.

2) Service

A) Explication du fonctionnement

Un service, dans le contexte du Développement logiciel, est une composante d'une application qui effectue des tâches spécifiques et offre des fonctionnalités ou des opérations bien définies. Les services sont généralement conçus pour isoler et regrouper la logique métier ou les fonctionnalités communes afin de les rendre réutilisables, modulaires et facilement maintenables.

Les services ici vont en l'occurrence permettre de recevoir les données en appelant les repository dont il a besoin puis de les traiter en les ordonnant en graphe pour les envoyer au contrôleur. Il y en a 3, un par tâches.

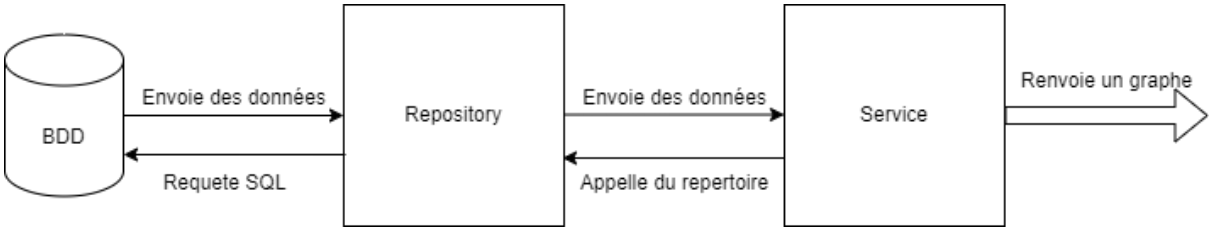


Figure 18 : Transition des données

Pour chaque service, nous aurons des fonctions dédiées pour récupérer tous les statuts associés à une tâche en fonction de l'utilisateur. Chaque fonction appellera la requête SQL correspondante, par exemple findAllAnoStatutForUser. Ensuite, nous aurons une fonction qui utilisera ces résultats pour créer un graphe spécifique à cette tâche et le renvoyer.

Un graphe est composé d'un titre et d'une liste de statuts. Chaque statut comprend un titre et une liste de travaux. Pour chaque type de tâche, nous avons une fonction dédiée qui renverra le graphe correspondant. Ces fonctions seront appelées dans le contrôleur pour obtenir tous les graphes nécessaires.

A noter que je ne pouvais pas utiliser un service commun nommé par exemple “GrapheService” car chaque type de tâches présentent des traitements de données différents pour créer leur graphe.

Cette approche nous permettra d'organiser et de structurer les données de manière claire et cohérente, en utilisant des fonctions spécifiques pour chaque étape du processus. Cela facilitera la maintenance du code et améliorera la lisibilité et la compréhension du fonctionnement de l'application.

B) Patron de conception DTO

Il est important de rappeler que les données transmises au contrôleur prennent la forme d'un objet appelé "graphe". Cet objet est composé d'un titre représentant son nom et d'une liste associée. Par conséquent, sa structure diffère totalement de celle des objets que nous recevons habituellement de la base de données, tels que les objets Anomalie, Evolution et Bons de préparation.

Afin de gérer cette différence de structure et d'assurer une meilleure gestion des données, nous avons opté pour l'utilisation d'un patron de conception appelé DTO. Ce patron de conception permet de créer des objets spécifiques, appelés classes DTO (Data Transfer Object), qui ne sont pas destinés à être enregistrés directement en base de données, mais qui servent de passerelle pour le transfert des données entre différentes parties de notre application.

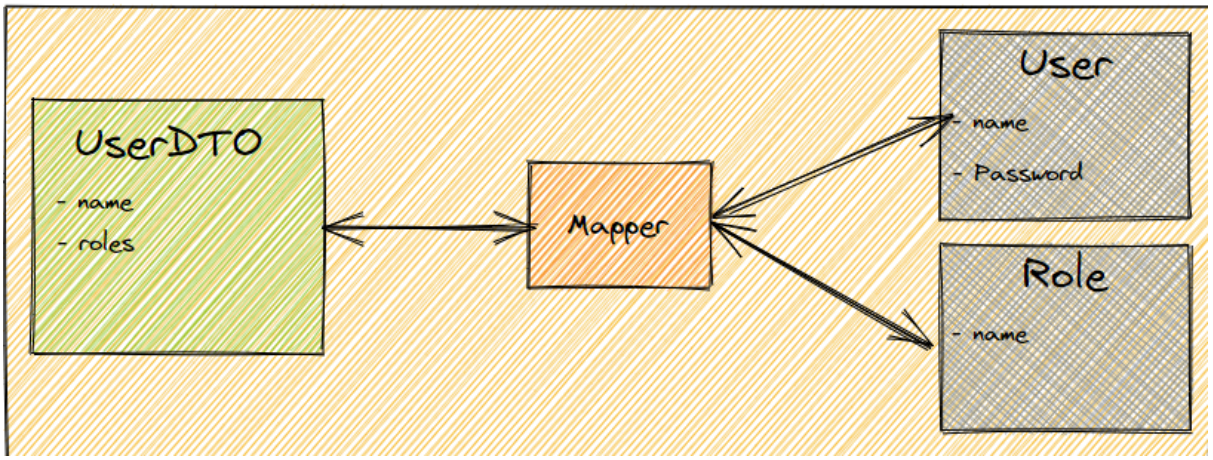


Figure 19 : Fonctionnement du patron de conception DTO

Dans notre cas :

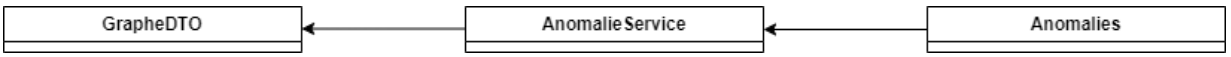


Figure 20 : Utilisation du patron de conception DTO

Lorsque les objets sont reçus depuis la base de données, nous les traitons en utilisant le patron de conception afin de les transformer en instances de classes DTO. Ici, L'anomalie service joue le rôle de l'assembleur. Ces classes DTO jouent un rôle crucial dans la communication des données, en facilitant leur passage d'un composant à un autre, tout en garantissant une structure cohérente et adaptée aux besoins spécifiques du graphe.

Patron de conception* : Solution optimale pour un problème récurrent connu.

3) Contrôleur

Le contrôleur a pour but de recevoir des requêtes et d'en renvoyer une réponse. Lorsqu'une requête est reçue, le contrôleur examine les données de la requête, détermine l'action à entreprendre et invoque les services appropriés pour effectuer le traitement nécessaire. Une fois ce traitement terminé, une réponse est générée et envoyée.

Mon contrôleur va appeler les différents services, créer une liste de grapheDto puis renvoyer une responseEntity (une réponse HTTP) contenant cette fameuse liste.
Cette approche permettra de ne pas avoir à toucher le front si jamais un nouveau type de tâche venait à être ajouté car il suffirait de renvoyer au contrôleur un nouveau type de graphe.

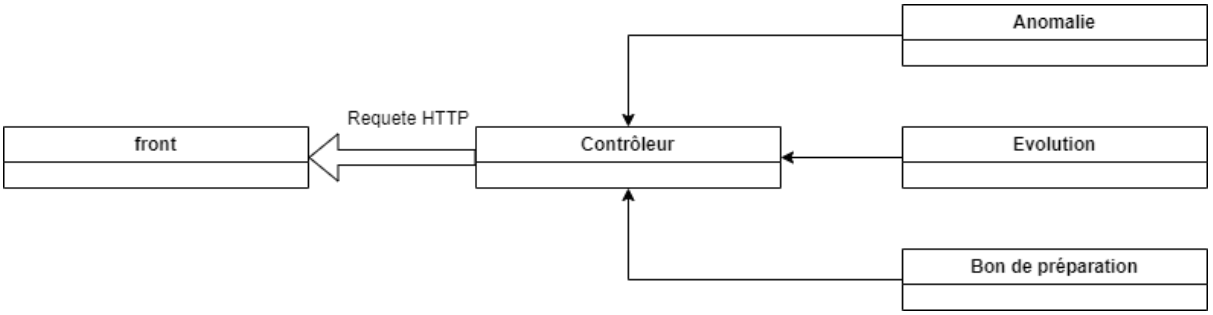


Figure 21 : Fonctionnement du contrôleur

Ce traitement est fait ici et non dans le service pour séparer les responsabilités entre la création des graphes et leur envoie par requête HTTP. De plus, cela est nécessaire pour créer la liste de GrapheDTO, cela ne peut pas se faire dans une grapheService.

V. Liaison Front-Back

Une fois les parties logiques et graphiques terminées, elles doivent être reliées entre elles afin d'afficher les données.

1) Récupération des données

Pour récupérer les données provenant du backend et les utiliser dans l'application frontend, un service a été mis en place. Ce service envoie une requête HTTP au backend pour récupérer les données souhaitées. Les composants du frontend s'abonnent ensuite à ce service pour recevoir les données.

Dans notre cas, seuls deux composants, dashboard.component et dashboard.all.graphic, ont besoin de ces données. Une fois que ces composants reçoivent les données, ils peuvent les communiquer aux autres composants de l'application.

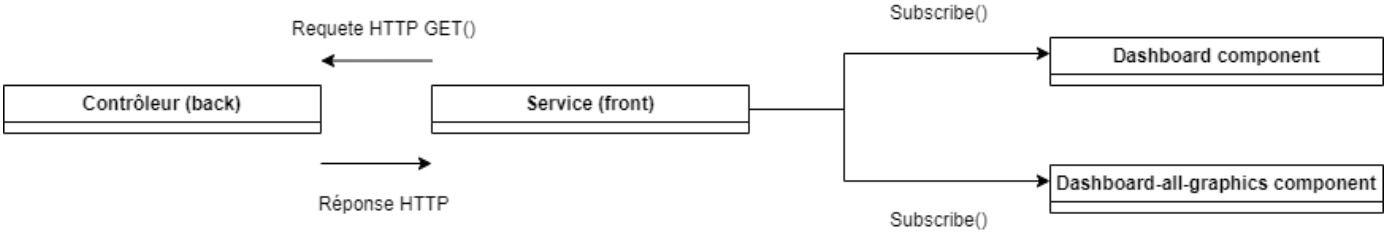


Figure 22 : Communication des données entre le front et le back

Cette approche permet une séparation claire des responsabilités. Cela permet aussi de respecter les conventions de code de Mistral. Le service gère les requêtes HTTP et la récupération des données, tandis que les composants spécifiques se concentrent sur l'affichage et la manipulation des données une fois qu'elles sont disponibles.

Cela favorise également la réutilisabilité, car d'autres composants pourraient également s'abonner au service pour récupérer les mêmes données si nécessaire. Ainsi, la logique de récupération des données est centralisée et les composants peuvent se concentrer sur leur propre fonctionnalité spécifique.

La récupération des données dans le service se fait via la fonction "get". Le chemin vers la réponse HTTP est déterminé en utilisant un attribut appelé "baseUrl", qui récupère l'URL de l'application plus le nom du contrôleur en backend. Ce nom de contrôleur est spécifié à l'aide d'une balise Spring Boot dans la partie frontend du code.

Ensuite, on appelle la fonction get du contrôleur dans les pages souhaitées grâces à la fonction subscribe.

2) Structure du projet

A) Explication du modèle MVC

Premis est organisé avec le patron d'architecture MVC (model - view - contrôleur) que je vais expliquer ci-dessous :

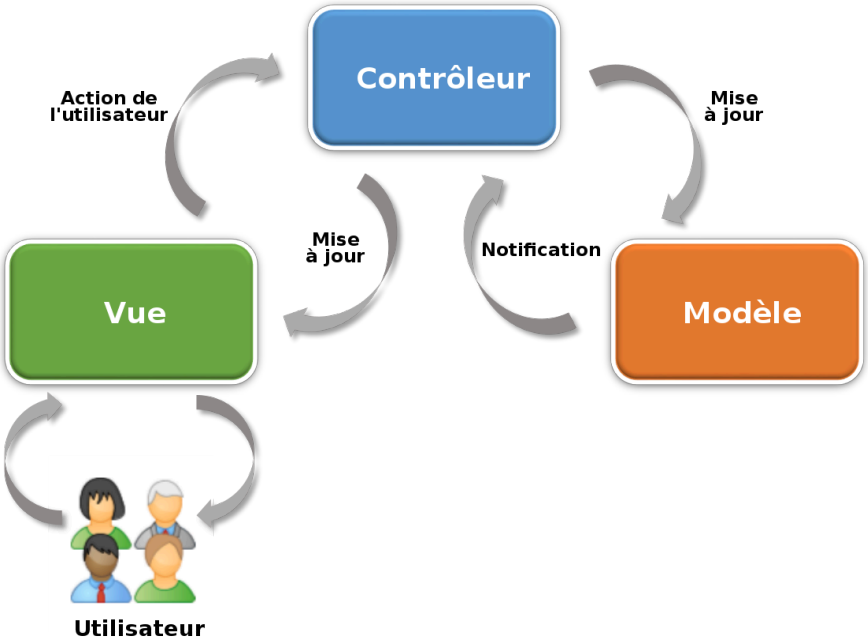


Figure 23 : Schéma explicatif du modèle MVC

Le modèle MVC (Modèle-Vue-Contrôleur) est un motif architectural couramment utilisé dans le développement logiciel pour organiser et structurer les applications. Il divise une application en trois composants principaux : le modèle, la vue et le contrôleur. Chaque composant a un rôle spécifique et est responsable d'une partie spécifique de la logique de l'application.

Modèle : Le modèle représente les données de l'application et définit la logique métier. Il encapsule les données, les règles de traitement et les opérations effectuées sur ces données. Le modèle est indépendant de l'interface utilisateur et ne connaît pas l'existence de la vue ou du contrôleur. Il est responsable de l'accès aux données, de leur manipulation et de leur persistance.

Vue : La vue est responsable de l'interface utilisateur de l'application. Elle affiche les données du modèle et permet à l'utilisateur d'interagir avec l'application. La vue récupère les données du modèle et les présente de manière appropriée à l'utilisateur. Elle peut également envoyer des événements au contrôleur lorsqu'un utilisateur interagit avec l'interface.

Contrôleur : Le contrôleur agit comme un intermédiaire entre le modèle et la vue. Il reçoit les événements de la vue et traite les actions correspondantes. Il met à jour le modèle en fonction des actions de l'utilisateur et actualise la vue pour refléter les changements. Le contrôleur orchestre l'interaction entre le modèle et la vue, en veillant à ce qu'ils restent indépendants l'un de l'autre.

En suivant le modèle MVC, les interactions de l'utilisateur sont capturées par la vue et transmises au contrôleur. Le contrôleur traite ces interactions, met à jour le modèle en conséquence et actualise la vue. Cela permet une séparation claire des responsabilités et facilite la maintenance, l'évolutivité et la réutilisabilité du code.

B) Application du modèle MVC

En faisant la conception de ma fonctionnalité, j'en suis arrivé naturellement au modèle MVC :

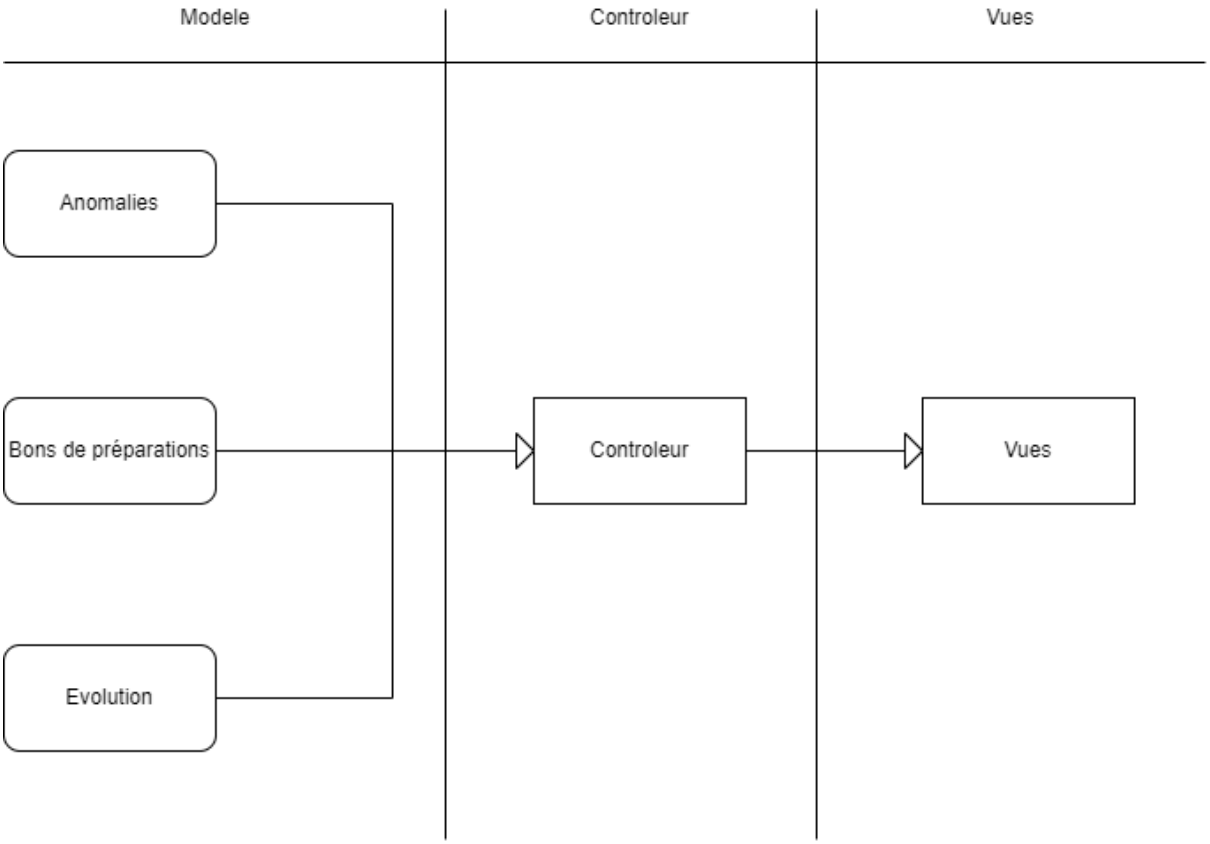


Figure 24 : Apparition du modèle MVC lors de la conception

Les sources de données seront les modèles car ce sont elles qui définissent la logique métier. Elles convergent vers le contrôleur qui va traiter ses données et les envoyées aux vues qui elles vont les afficher.

Voici comment j'ai utilisé les différentes classes pour appliquer ce modèle :

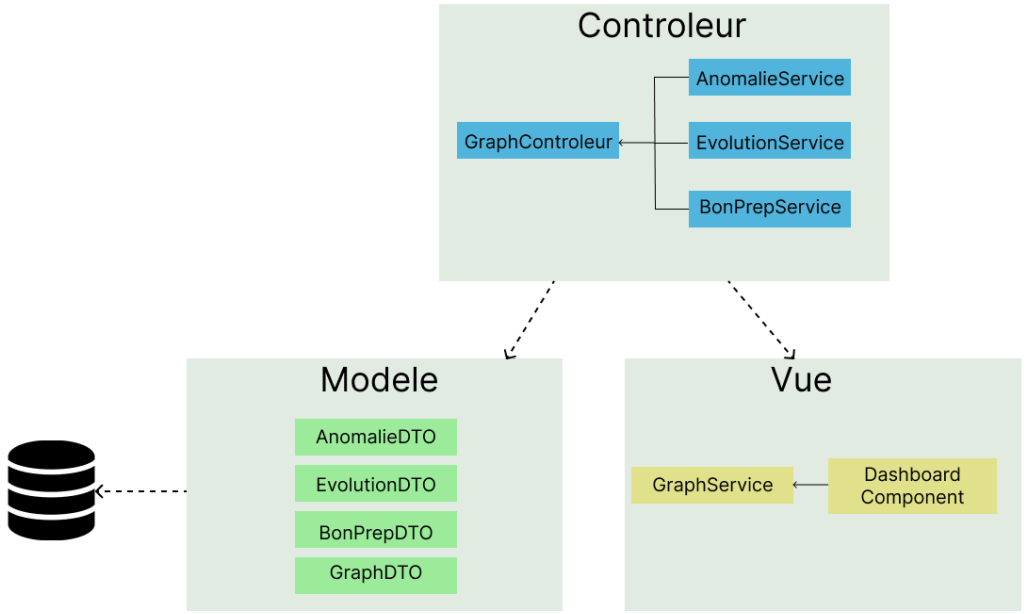


Figure 25 : Application du MVC pour la fonctionnalité tableau de bord

Modèle : Dans cette partie, j'ai créé des classes qui représentent les données et la logique métier de l'application. Par exemple, j'ai créé une classe GraphDTO et j'ai utilisé des classes déjà présente tel que AnomalieDTO pour représenter les entités principales de mon domaine.

Vue : La vue permet l'affichage des données, cela est possible grâce au Graphe Service qui récupère une réponse HTTP et les différents composants angular qui vont les traiter et afficher.

Contrôleur : Le contrôleur, qui permet de traiter et envoyer une réponse, est ici séparé en 2 parties : les services qui vont traiter les données et le contrôleur qui va renvoyer une réponse.

En utilisant ce modèle MVC, j'ai pu séparer efficacement la logique métier, la présentation des données et la gestion des interactions. Cela a permis une meilleure organisation du code, une réutilisabilité accrue et une maintenance plus facile de l'application.

3) Sauvegarde des paramètres utilisateurs

Pour récupérer et enregistrer les différents paramètres utilisateur tels que la couleur du graphe et la position des graphes, j'ai opté pour l'enregistrement de ces paramètres en base de données. Pour cela, j'ai créé une classe GrapheParam qui agit comme une entité permettant de stocker les paramètres dans la base de données. Parallèlement, en suivant le patron de conception DTO (Data Transfer Object), j'ai également développé une classe GrapheParamDTO qui est utilisée pour transférer ces données.

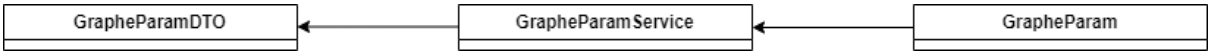


Figure 26 : Application du patron de conception DTO

La mise en œuvre des DTOs suit un processus similaire à celui des graphes. Ils sont créés et manipulés dans le service GrapheParamService avant d'être transmis au contrôleur pour les opérations ultérieures. Du côté du front-end, les paramètres sont récupérés en effectuant une requête HTTP GET lors de la création de la page. Lorsque la page est fermée ou lors d'une action spécifique, une requête HTTP PUT est utilisée pour envoyer les paramètres modifiés au back-end. Ces paramètres sont ensuite enregistrés dans la base de données pour une utilisation future.

En enregistrant les paramètres utilisateur en base de données, j'assure une persistance et une récupération fiables de ces informations lors des sessions ultérieures de l'utilisateur.

BILAN TECHNIQUE

La fonctionnalité que j'ai développée est implémentée sur PreMis pour une première phase de tests réels. L'objectif est de recueillir des retours afin d'améliorer l'ergonomie. Cette fonctionnalité permet aux employés de Mistral de visualiser rapidement les différents travaux qu'ils doivent réaliser. Comme mentionné précédemment, l'idée est encore à un stade préliminaire et peut évoluer ou être complètement transformée à tout moment. Il est donc possible que dans quelques semaines ou mois, mon code soit modifié, amélioré, voire complètement remplacé.

Mon code sert donc de base pour les développements futurs, et c'est précisément l'objectif de mon stage. J'ai travaillé dès le départ pour rendre mon code aussi maintenable que possible et le plus générique possible, afin qu'il puisse être facilement repris par d'autres développeurs.

L'accent a été mis sur la qualité et la lisibilité du code, en suivant les meilleures pratiques de programmation. J'ai utilisé des conventions de nommage claires et j'ai structuré le code de manière logique et modulaire.

En prévision des évolutions et des modifications potentielles, j'ai veillé à rendre mon code flexible et extensible.

Dans l'ensemble, j'ai mis tout en œuvre pour fournir une base solide et bien conçue pour la fonctionnalité, afin de faciliter les évolutions et les améliorations ultérieures. Je suis fier d'avoir contribué à ce projet et j'espère que mon travail pourra servir de fondement solide pour les développements à venir.

BILAN FONCTIONNEL

Le travail que j'ai fourni à la fin de mon stage est très similaires aux maquettes (cf annexes 6 page 45) que j'ai pu faire dans la partie conception mais présente néanmoins des différences :

I) Les tâches

Les tâches sont présenté de la sorte :

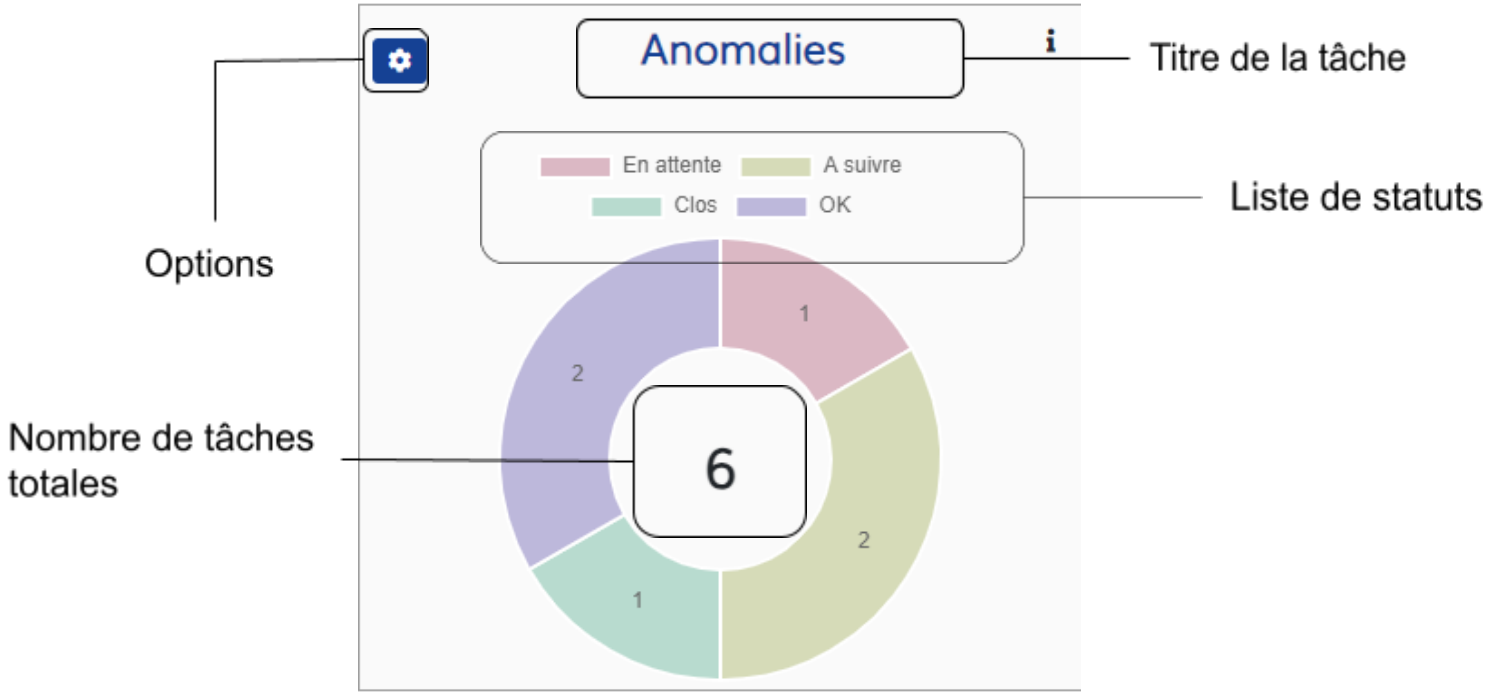


Figure 27 : Visuel réel des tâches dans le tableau de bord

Ces tâches sont personnalisables, on peut changer la couleur du graphique et le type de statut affiché. En cliquant sur le bouton des options cette page s'ouvre :

Figure 28 : Paramétrage des tâches

II) Les graphiques et tableaux

Lorsque l'on clique sur une tâche, une page s'ouvre sous celle déjà existante. Cette page permet d'afficher différents graphiques en lien avec la tâche active. (cf annexe 7 page 46). Cette page est aussi paramétrable, on peut y choisir le type de graphe affichés. Ceci est donc totalement différent de la conception de base ou le fait de cliquer pour afficher un nouveau graphique. Ceci à été modifié car après test, les retours indiquent que le graphique en plus de la nouvelle page était redondant, j'ai donc décidé de garder la page avec les différents graphiques.

Figure 29 : Paramètres de sélection des graphiques affichés

Mais il peut arriver que l'utilisateur veuille voir les taches en détail. Pour cela, il suffit de cliquer sur ces graphiques pour afficher un tableau avec la liste des tâches détaillées affectée à l'utilisateur. (cf annexe 4 page 43).

III) Le menu

Etant donné que les composants s'ajoutent les unes en dessous des autres, la page peut vite devenir chargée et il peut être contraignant de se naviguer. C'est pour cela que j'ai mis au point un menu facilitant la navigation. Il est constamment placé au milieu à droite de la page.

Figure 30 : Menu du tableau de bord

CONCLUSION

Ce stage a été une expérience très enrichissante pour moi dans le domaine de l'informatique.

Tout d'abord, il m'a permis d'avoir un aperçu de l'organisation d'une entreprise informatique et de comprendre les exigences et le niveau requis en Développement. Pendant mes études à l'IUT, je me demandais souvent si mon niveau était suffisant pour travailler, mais ce stage m'a montré que je pouvais m'adapter et répondre aux attentes professionnelles.

Ensuite, c'était la première fois que je travaillais avec des technologies inconnues pour moi, ce qui a rendu ma tâche initialement complexe. Cependant, cela m'a également permis d'acquérir de nouvelles compétences et d'apprendre à m'adapter rapidement à de nouveaux environnements. J'ai découvert une nouvelle façon d'apprendre et de relever des défis.

Enfin, ce stage m'a offert l'opportunité de développer des compétences en conception et en architecture d'applications de manière autonome. C'était un exercice totalement différent de ce que j'avais pu faire auparavant. Le fait que d'autres développeurs reprennent mon travail m'a incité à être rigoureux et attentif aux exigences de conception. Travailler sur une application déjà existante m'a également permis de progresser, car cela représentait une première pour moi. Analyser et comprendre le code de développeurs plus expérimentés a été extrêmement bénéfique pour mon stage. C'était une expérience très valorisante qui a renforcé ma confiance en mes compétences et ma capacité à évoluer dans le monde professionnel de l'informatique.

ENGLISH SUMMARY

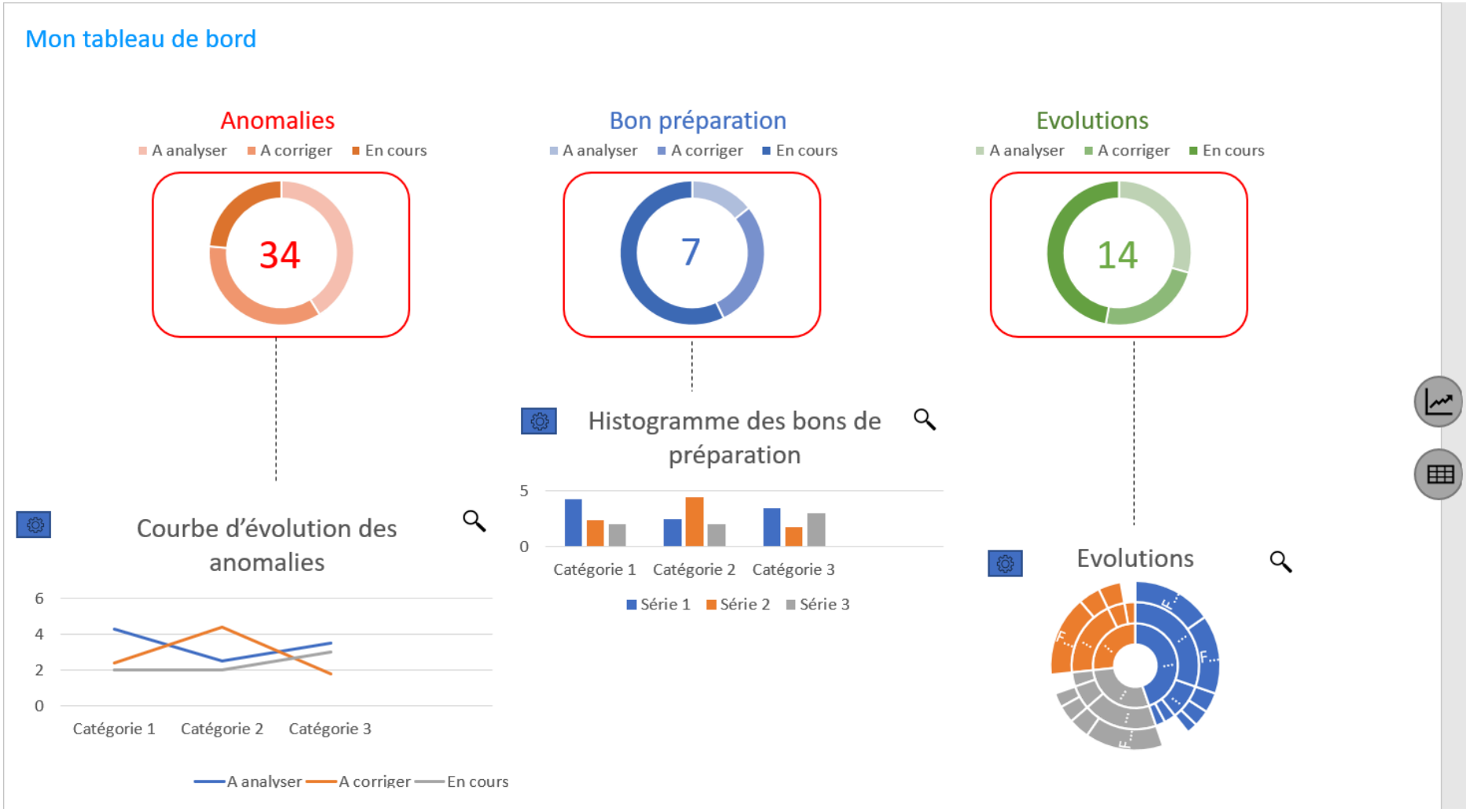
During my computer science studies at the IUT, I had the opportunity to complete an internship at MISTRAL INFORMATIQUE, a computer company based in Clermont-Ferrand. This company specializes in developing integrated solutions and management packages (ERP) for various industries, including agriculture and manufacturing.

During my internship, I was involved in designing and developing a new feature for their internal application, PreMis. To begin with, I undertook the task of designing the feature by creating several diagrams and models. This included sequence diagrams to identify the required classes and their behavior during interactions, as well as use case diagrams to capture all the functionalities. Additionally, I created a model to guide the feature's design.

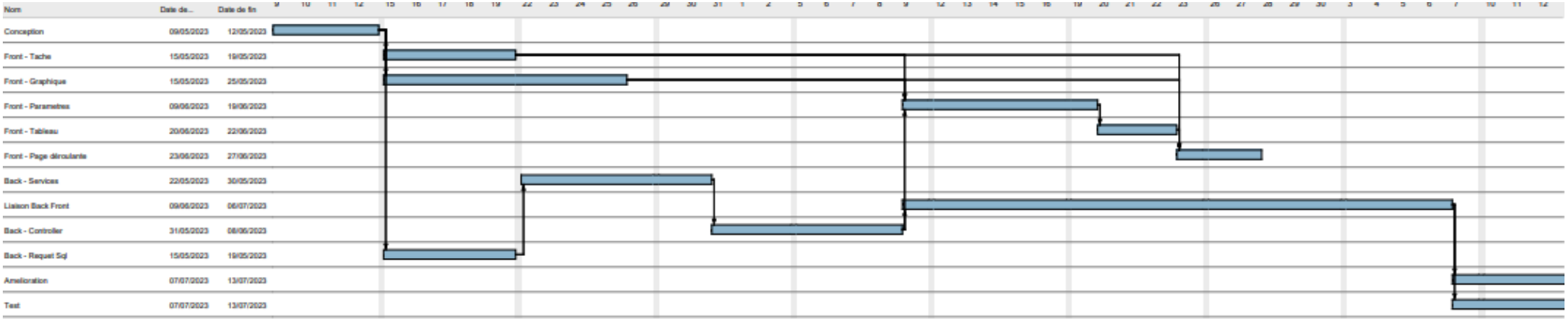
After completing the design phase, I proceeded with the implementation of the feature. I started by developing the application's frontend, focusing on creating reusable components to ensure flexibility and ease of maintenance. Following that, I began working on the backend by writing SQL queries to retrieve the necessary data from the database. Once the data was retrieved, I processed it to generate graphs and prepared the HTTP responses accordingly. This involved encoding controllers and services to handle the data manipulation and communication between different parts of the application.

Overall, my internship at MISTRAL INFORMATIQUE provided me with valuable experience in designing and developing a new functionality for their internal application. I had the opportunity to apply my skills in software engineering, database management, and frontend development to contribute to the success of the project.

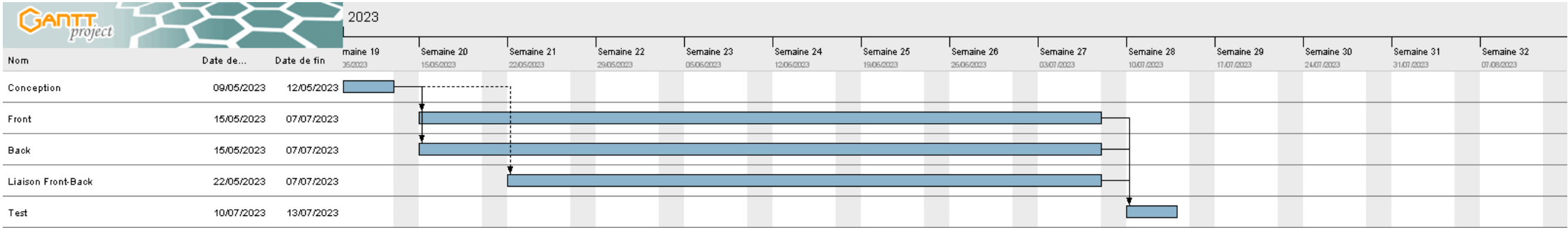
Annexe 1 : Maquette de la page Tableau de bord



Annexe 2 : Diagramme de gantt prévisionnel



Annexe 3 : Diagramme de gantt réel



Annexe 4 : Tableau des anomalies

Suivi des anomalies

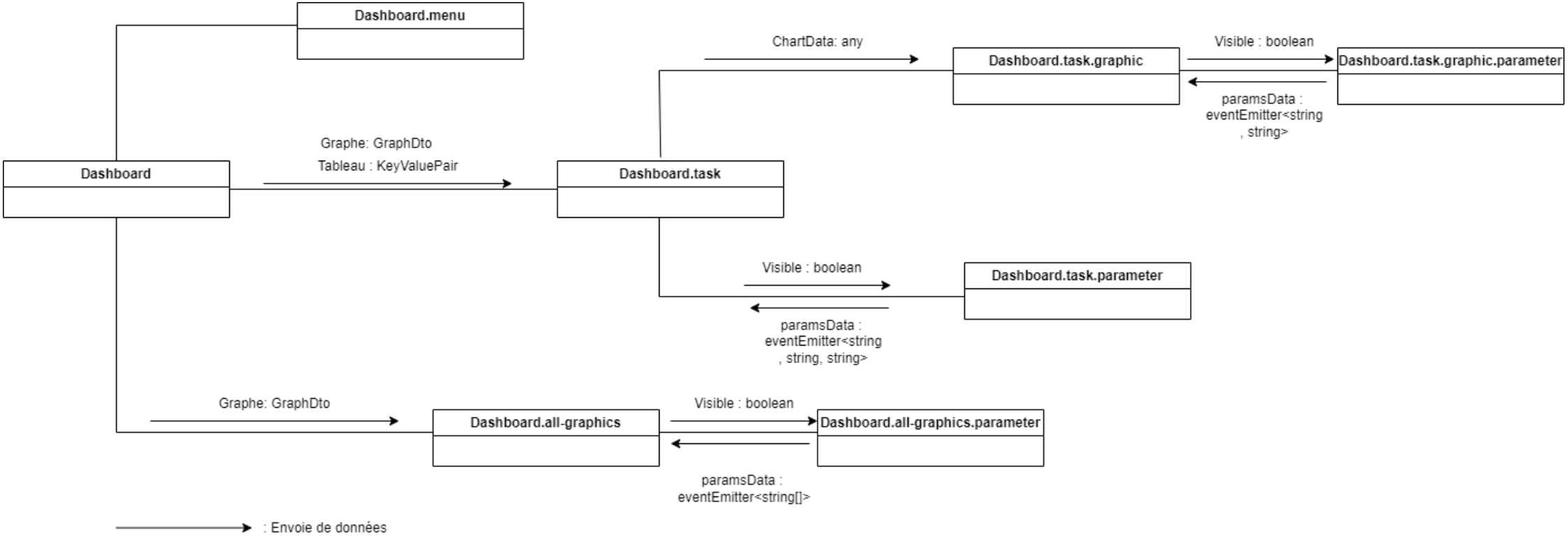
+ Créer | Exporter

Regroupements | Filtres prédéfinis | Affichage: 25 colonnes sélectionnées | Sélections: Sélection par défaut | Rechercher

Numéro	Ticket support	Code client	Raison sociale	Créée par	Date création	Application	Version	Module	Programme	Environnement	Mobilité	Priorité	Titre	Description	Suivi	Actions	Affectée à	Contribution de
4911		140078	DELL 63	RHA	15/06/2023	3G	test	test	test	Développeme		P3	test1	test			RHA	
4910		3175	SO.B.TRAN (V	RHA	15/06/2023	3G	test	test	test	Développeme		P3	test2	test			RHA	
4909		631914	MARVALIN 63	RHA	14/06/2023	3G	test	test	test	Développeme		P3	test3	test			RHA	
4908		291653	ARMORICAINI	RHA	14/06/2023	3G	test	test	test	Développeme		P3	test4	test			RHA	
4907		140078	DELL 63	RHA	14/06/2023	iOT Portail	1.1.1			Développeme		P3	test5	test			RHA	
4905		1	MISTRAL INFC	CGO		S@PHIR	1.0.0		DMC	Production		P1	DMC - pb d'ob	DMC >> I			CGO	
4904		1	MISTRAL INFC	CGO		S@PHIR	1.0.0		DMC	Production		P1	DMC - pb d'ob	DMC >> I			CGO	
4903		1	MISTRAL INFC	CFI	18/04/2023	Mobistac	1.1.1	1	2	Développeme	✓	P3	1	1			CFI	

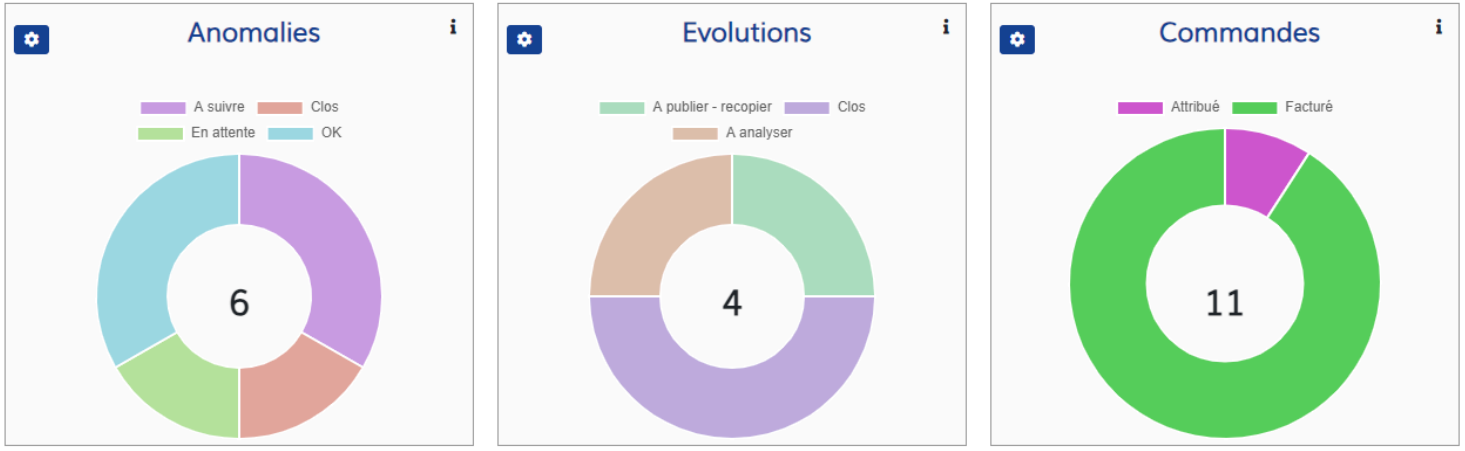
Affichage du 1 au 8 des 4706 éléments « ‹ 1 2 3 4 5 › » 8

Annexe 5 : Schéma complet des communications entre composant

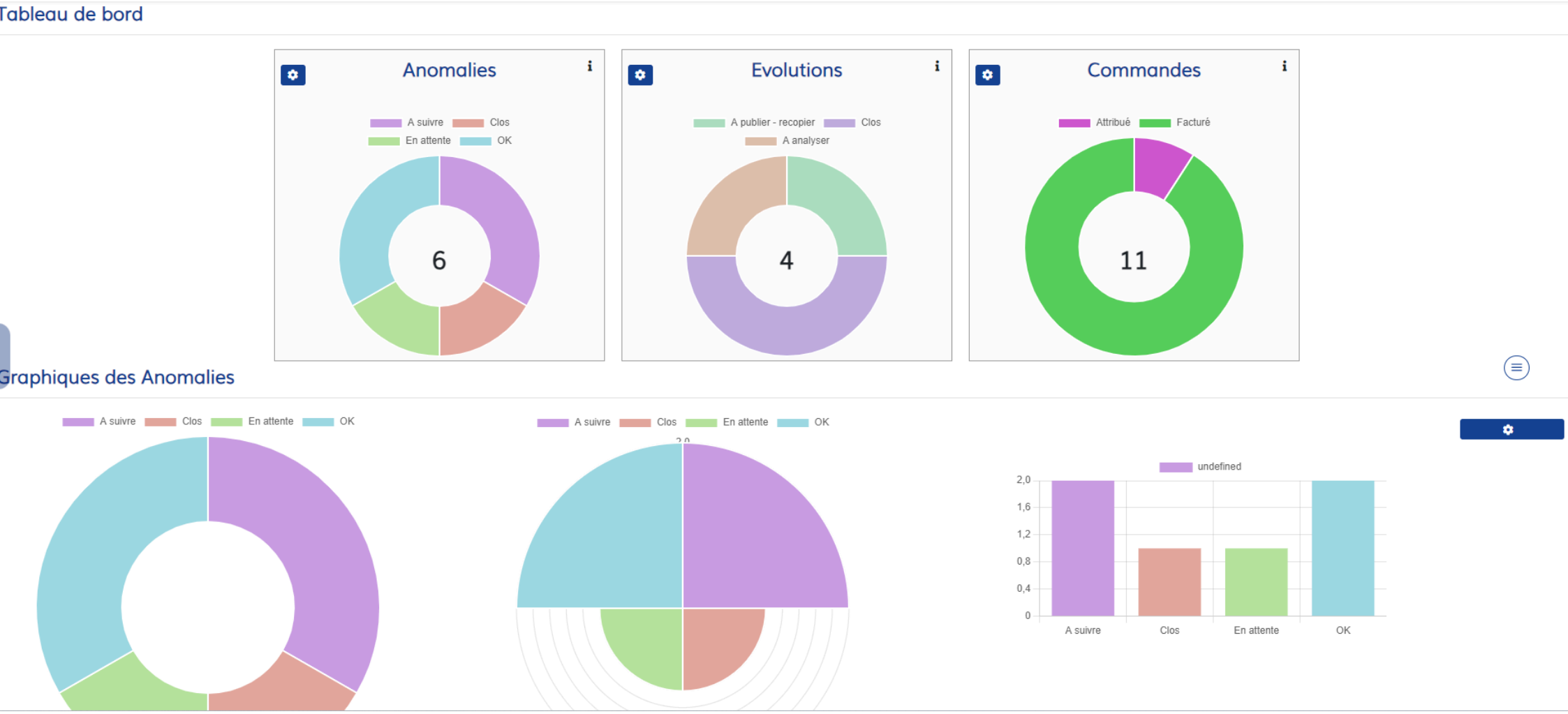


Annexe 6 : Page tableau de bord lorsque les graphiques ne sont pas affichés

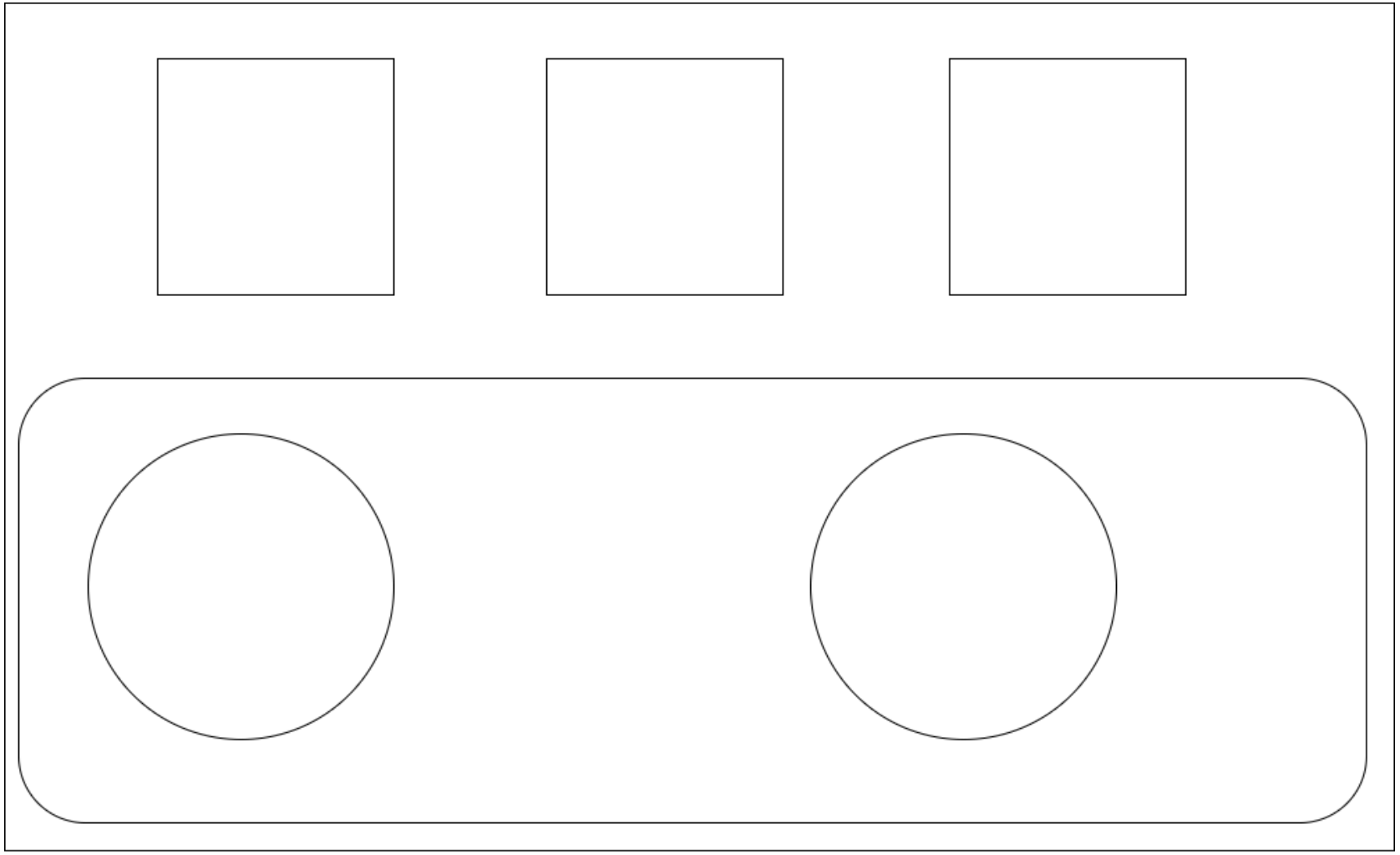
Tableau de bord



Annexe 7 : Page tableau de bord lorsque les graphiques sont affichés



Annexe 8 : Fonctionnement des slides



Slide : contenaire qui s'ajoute sous la page principale

Graphique

Tache

Annexe 9 : Schéma explicatif de la liaison entre le front et le back avec leurs technologies respectives.