**Titre Professionnel de**

**Concepteur Développeur d’Applications 2024**

**DOSSIER PROJET**

**SHILKOV Konstantin**

**PictoPicto  
\_\_\_**

**Application pour les enfants autistes Spring Boot - Next.js**

Изображение выглядит как Шрифт, Графика, логотип, символ

Автоматически созданное описание



Sommaire

[I. Liste des compétences du référentiel couvertes par le projet 2](#_Toc169877373)

[1. Développer une application sécurisée 2](#_Toc169877374)

[2. Concevoir et développer une application sécurisée organisée en couches 2](#_Toc169877375)

[3. Préparer le déploiement d’une application sécurisée 2](#_Toc169877376)

[II. Résumé du projet en anglais 3](#_Toc169877377)

[III. Cahier des charges 4](#_Toc169877378)

[1. Contexte 4](#_Toc169877379)

[2. Objectif 4](#_Toc169877380)

[IV. Gestion de projet 6](#_Toc169877381)

[1. Planning et méthodologie de suivi 6](#_Toc169877382)

[2. Diagramme de Gantt 6](#_Toc169877383)

[V. Spécifications fonctionnelles 7](#_Toc169877384)

[1. Diagramme de cas d’utilisation 7](#_Toc169877385)

[2. Descriptions détaillées des cas d’utilisation 8](#_Toc169877386)

[3. Modèle conceptuel de données 11](#_Toc169877387)

[4. Diagramme de classes 13](#_Toc169877388)

[5. Diagrammes d’activité 14](#_Toc169877389)

[6. Diagrammes de séquence 16](#_Toc169877390)

[7. Diagramme de composant 16](#_Toc169877391)

[8. Diagramme de déploiement 17](#_Toc169877392)

[9. Maquettes du projet 17](#_Toc169877393)

[VI. Spécifications techniques. 20](#_Toc169877394)

[1. Techologies choisies 20](#_Toc169877395)

[2. Architecture orientée services de l’application côté serveur 21](#_Toc169877396)

[3. Processus de travail 23](#_Toc169877397)

[4. Sécurité 25](#_Toc169877398)

[VII. Réalisations 27](#_Toc169877399)

[1. Partie front-end (côte client) 27](#_Toc169877400)

[2. Partie back-end (côte serveur) – API Gateway 29](#_Toc169877401)

[3. Partie back-end (côte serveur) – Discovery server 32](#_Toc169877402)

[4. Partie back-end (côte serveur) – REST API services de la logique métier 34](#_Toc169877403)

[VIII. Jeu d’essai 39](#_Toc169877404)

[1. Tests unitaires 40](#_Toc169877405)

[2. Tests d’integration 42](#_Toc169877406)

[3. Conclusion 45](#_Toc169877407)

[IX. Veille sur les vulnérabilités de sécurité 45](#_Toc169877408)

[1. Méthodologie 45](#_Toc169877409)

[2. Exemple de mise en place 45](#_Toc169877410)

[X. Démarches DevOps 45](#_Toc169877411)

[XI. Situation de travail ayant nécessité une recherche 45](#_Toc169877412)

[1. Problème Rencontré 45](#_Toc169877413)

[2. Étapes de Résolution 46](#_Toc169877414)

[3. Résumé 47](#_Toc169877415)

[XII. Conclusion 47](#_Toc169877416)

# I. Liste des compétences du référentiel couvertes par le projet

|  |  |  |
| --- | --- | --- |
| **Activités types** | **N° Fiche CP** | **Compétences professionnelles** |
| 1. Développer une application sécurisée | **1** | Installer et configurer son environnement de travail en fonction du projet |
| **2** | Développer des interfaces utilisateur |
| **3** | Développer des composants métier |
| **4** | Contribuer à la gestion d’un projet informatique |
| 2. Concevoir et développer une application sécurisée organisée en couches | **5** | Analyser les besoins et maquetter une application |
| **6** | Définir l’architecture logicielle d’une application |
| **7** | Concevoir et mettre en place une base de données relationnelle |
| **8** | Développer des composants d’accès aux données SQL et NoSQL |
| 3. Préparer le déploiement d’une application sécurisée | **9** | Préparer et exécuter les plans de tests d’une application |
| **10** | Préparer et documenter le déploiement d’une application |
| **11** | Contribuer à la mise en production dans une démarche DevOps |

# II. Résumé du projet en anglais

The project concerns children with autism. Indeed, these children have a difficult development of their communication. They have the words, but often use them without understanding their exact meaning. This is the case, for example, with echolalia, where the child repeats word for word, like a parrot, what he has heard without knowing the meaning.

A partnership agreement has been signed between **ECAM-EPMI** (engineering school) and **EPEA** (Ecoute Parents Enfants Autistes) to develop digital tools to help autistic children. Thanks to the work of **Mr. Moncef BENKHERRAT** (research professor at EPMI), an application has been designed as a digital tool to assist children communicate better, find their bearings in time and socialize.

**“PictoPicto”** helps them to increase, maintain or develop their abilities. The work carried out consists in exploiting both technological and social aspects. It is financed by the **Terre Plurielle** foundation thanks to the help and support of **Mr Pascal LEGENDRE** (Technical Director at Bouygues Energies et Services), who sponsored the project with Terre Plurielle.

**“PictoPicto”** can provide autistic children, their parents, and therapists with simple and user-friendly digital learning tools for communication. The general aim of this tool is to stimulate linguistic expression of internal mental states, using a graphic support and a verbal explanation strategy. This translates into:

● Associating an image with its textual meaning

● Constructing a sentence, communicating using pictograms

● Reading and listening to constructed sentences

The first version of “**PictoPicto**” was launched in 2021. However, due to the rapid progress and development of technologies, it becomes difficult to maintain.

As part of our internship, we were tasked with developing a new version of this application using current technologies, ensuring that it meets all modern requirements for performance and security.

# III. Cahier des charges

## 1. Contexte

Le problème majeur des enfants atteints d'autisme n'est donc pas l'absence de parole mais le développement difficile de leur communication. Ils ont les mots, mais les utilisent souvent sans en comprendre la signification exacte. Ce qui est, par exemple, le cas dans l'écholalie où l'enfant répète mot pour mot, comme un perroquet, ce qu'il a entendu sans en connaître la signification.

Une convention de partenariat a été signée entre ECAM-EPMI (Ecole d’ingénieurs) et EPEA (Ecoute Parents Enfants Autistes) dans le but de développer des outils numériques d’aides aux enfants autistes. Grâce au travail de Monsieur Moncef BENKHERRAT (Enseignant chercheur à l’ECAM-EPMI), une application permettant aux enfants de mieux communiquer, se repérer dans le temps et se sociabiliser a été conçue.

Ce projet consiste à utiliser les nouvelles technologies comme un outil. Il permet d’augmenter, de maintenir ou de développer leurs capacités. Le travail effectué consiste à exploiter les aspects à la fois technologiques et sociaux. Il est financé par la fondation Terre Plurielle grâce à l'aide et au soutien de Monsieur Pascal LEGENDRE (directeur technique à Bouygues Energies et Services) qui a parrainé le projet auprès de Terre Plurielle.

L’EPMI peut ainsi mettre à la disposition des enfants autistes, de leurs parents et de leurs thérapeutes, des outils numériques d'apprentissage de la communication simples et ergonomiques.

Deux versions de l’application ont déjà été développées et sont accessibles en ligne.

## 2. Objectif

L’application permet de répondre à des problématiques primordiales. C’est une aide aux enfants autistes pour comprendre l’expression des vécus et les expressions d’autrui. L’objectif général de cet outil est la stimulation de l’expression linguistique des états mentaux internes à l’aide d’un support graphique et à travers une stratégie d’explication verbale. Cela se traduit par :

● Associer une image à sa signification textuelle

● Construire une phrase, communiquer en utilisant des pictogrammes

● Lire et écouter les phrases construites

Pour le moment il n’existe qu’une application web que ce soit pour les familles ou les professionnels de santé. L’objectif est de séparer l’application en 3 parties :

● Une application web destinée aux public (enfants avec leurs parents)

● Une application web destinée aux professionnels de structures spécialisées

● Une application mobile

Durant ma période de stage, du 18 mars au 31 mai 2024, en binôme avec un autre stagiaire, on nous a confié la conception et le développement de l’application pour les professionnels. Voilà les demandes qui ont été formulées pour ce projet :

* Back-end en Spring Boot et le front-end en Next.js
* S’inspirer de l’application existante et l’améliorer
* Restructurer la base de données
* Garder les fonctionnalités existantes pour la construction des phrases
* Améliorer ou créer les algorithmes pour la grammaire et la conjugaison afin de construire des phrases correctes
* Permettre aux professionnels de gérer leurs patients et le contenu pédagogique (gestion des catégories et des pictogrammes)

Cette application sera ensuite testée dans des structures spécialisées.

# IV. Gestion de projet

## 1. Planning et méthodologie de suivi

Etant donné les moyens humains (quatre personnes) et un temps très court (11 semaines) consacré au développement de l’application, nous avons adopté la méthode agile pour la gestion du projet. Nous avons privilégié une approche itérative par la réalisation de versions fonctionnelles plus petites à des intervalles réguliers plutôt que de développer l’application dans son intégralité. Les démonstrations fréquentes des fonctionnalités achevées permettent de s'assurer que l'application développée répond réellement aux besoins et aux attentes des utilisateurs.

La première semaine a été consacrée à l’élaboration du cahier des charges et à la définition des besoins du client. Les deux semaines suivantes ont été consacrées à la conception de l’application. Pendant la troisième semaine on a commence la création de la base de données. Six prochaines semaines auront été nécessaires au développement des parties front-end et back-end de l’application. Les deux dernières semaines ont été consacrées au déploiement de l’application et de la base de données, aux tests et débogage.

Le suivi du projet a été réalisé à l’aide de 3 outils:

● Utilisation de la méthode Kanban avec la mise en place d’un Slack

● Mise en ligne du code sur un dépôt Git hébergé sur Gitlab

● Utilisation de Discord et Slack: échanges réguliers avec notre tuteur de stage, Moncef BENKHERRAT et notre coordinateur de stage, Mustapha YOUNSI, afin de vérifier l’état d’avancement du projet et pouvoir en faire des démonstrations.

## 2. Diagramme de Gantt

Изображение выглядит как текст, линия, Шрифт, снимок экрана

Автоматически созданное описание

# V. Spécifications fonctionnelles

## 1. Diagramme de cas d’utilisation

Tout d’abord, on a créé une diagramme de cas d’utilisation montrant comment les fonctionnalités et acteurs sont interconnectés.

Il y a trois acteurs principales : Patient, Therapeute, SuperAdmin et un acteur supplemantaire : Invité .

Изображение выглядит как диаграмма, текст, карта

Автоматически созданное описание

## 2. Descriptions détaillées des cas d’utilisation

J’ai ensuite rédigé une explication textuelle des cas d’utilisation suivant le même schéma: titre, résumé, acteurs, préconditions, scénario nominal, enchaînements d’erreurs, postconditions. Voici des plusieurs explications :

**Se connecter**

|  |  |
| --- | --- |
| **Sommaire d’identification** | |
| Titre du cas d’utilisation | Se connecter |
| Résumé | Permet de s'authentifier dans le système et d'accéder à des  fonctionnalités spécifiques |
| Acteurs | Therapeute, SuperAdmin |
| **Description des scénarios** | |
| Préconditions | Application accessible |
| Scénario nominal | 1. L’acteur accède à la page de connexion  2. L’acteur saisit son login et son mot de passe.  3. Le système vérifie l'existence du compte.  4. Le système renvoie l'interface correspondante. |
| Enchaînements d’erreur | 3a. Aucun compte correspondant au couple login/mot de passe indiqué :  - le système lève une exception ;  - le cas d'utilisation se termine en échec ;  - retour à l'étape 1 |
| Post-conditions | L’acteur est authentifié et accède aux fonctionnalités qui lui sont dédiées |

**Gérer les ressourse pédagogiques**

|  |  |
| --- | --- |
| **Sommaire d’identification** | |
| Titre du cas d’utilisation | Gérer les ressourse pédagogiques |
| Résumé | Permet d'ajouter ou modifier des questions et des pictogrammes |
| Acteurs | Therapeute, SuperAdmin |
| **Description des scénarios** | |
| Préconditions | Application accessible, acteur est connecté |
| Scénario nominal | 1. L’acteur accède la page de gestion des ressourse pédagogiques 2. L’acteur choisit un ressourse 3. L’acteur saisit les données 4. L’acteur met à jour |
| Enchaînements d’erreur | 3a. Données invalide. Affichage d'un message d'erreur |
| Post-conditions |  |

**Gérer les dossiers des patients**

|  |  |
| --- | --- |
| **Sommaire d’identification** | |
| Titre du cas d’utilisation | Gérer les dossiers des patients |
| Résumé | Permet de modifier, ajouter ou désactiver les dossiers des patients |
| Acteurs | Therapeute, SuperAdmin |
| **Description des scénarios** | |
| Préconditions | Application accessible, acteur est connecté |
| Scénario nominal | 1a. Le Therapeute accède à la page de consultation du dossier de ses patients  1b. Le SuperAdmin accède à la page de consultation du dossier de tous les patients  2. L’acteur ajoute, modifie ou désactive un dossier du patient  3. Le système vérifie si les données sont valides  4. Le système renvoie l'interface correspondante. |
| Enchaînements d’erreur | 3a. Les données sont invalides :  - le système lève une exception ;  - le cas d'utilisation se termine en échec  - retour à l'étape 2 |
| Post-conditions | Le dossier médical est créé, modifié ou désactivé |

**Choisir une question**

|  |  |
| --- | --- |
| **Sommaire d’identification** | |
| Titre du cas d’utilisation | Choisir une question |
| Résumé | Permet au Therapeute de poser une question à l'enfant |
| Acteurs | Therapeute |
| **Description des scénarios** | |
| Préconditions | Application accessible, séssion est lancé |
| Scénario nominal | 1. L’acteur lance une séssion en mode d’echange 2. L’acteur choisit une question |
| Enchaînements d’erreur |  |
| Post-conditions | Lecture orale de question par le service de synthèse vocale  Affichage des categories et des pictogrammes associés avec cette question |

**Construire une phrase**

|  |  |
| --- | --- |
| **Sommaire d’identification** | |
| Titre du cas d’utilisation | Construire une phrase |
| Résumé | Permet au Patient de repondre à la question du Therapeute ou s’exprimer |
| Acteurs | Patient |
| **Description des scénarios** | |
| Préconditions | Application accessible, séssion est lancé, mode de communication est choisi |
| Scénario nominal | 1. L’acteur glisse-dépose des pictogrammes sur une forme de construction de phrase |
| Enchaînements d’erreur |  |
| Post-conditions | Correction et affichage de la phrase |

**Lancer une synthèse vocale**

|  |  |
| --- | --- |
| **Sommaire d’identification** | |
| Titre du cas d’utilisation | Lancer une synthèse vocale |
| Résumé | Permet à le Patient d'écouter la phrase |
| Acteurs | Therapeute, Patient |
| **Description des scénarios** | |
| Préconditions | Application accessible, séssion est lancé, phrase est construite |
| Scénario nominal | 1. L’acteur appuie sur le bouton « Ecouter la phrase » / « Ecouter la phrase mot à mot » |
| Enchaînements d’erreur |  |
| Post-conditions | Lecture orale de la phrase par le service de synthèse vocale |

**Conjuguer la phrase**

|  |  |
| --- | --- |
| **Sommaire d’identification** | |
| Titre du cas d’utilisation | Conjuguer la phrase |
| Résumé | Permet à lacteur de conjuguer la phrase aux temps différents |
| Acteurs | Therapeute, Patient |
| **Description des scénarios** | |
| Préconditions | Application accessible, séssion est lancé, Phrase est construite |
| Scénario nominal | 1. L’acteur appuie sur les boutons de conjugaison de phrase |
| Enchaînements d’erreur |  |
| Post-conditions | La phrase est conjuguée.  Lecture orale de la phrase par le service de synthèse vocale |

## 3. Modèle conceptuel de données

Pour la conception de la base de données, j’ai commencé par le modèle conceptuel de données (**MCD**) afin d’avoir une représentation de mes entités, de leurs attributs et des relations entre elles.

Изображение выглядит как текст, диаграмма, План, линия

Автоматически созданное описание

Suite à celui-ci, j’ai pu réaliser mon modèle logique de données (**MLD**) :

**Institution** (institution\_id, institution\_title, institution\_contactName, institution\_email, institution\_phoneNumber, institution\_code, institution\_createdAt, institution\_updatedAt)

**User** (user\_id, user\_firstName, user\_lastName, user\_email, user\_password, user\_phoneNumber, user\_job, user\_isActive, user\_isVerified, user\_createdAt, user\_updatedAt, #institution\_id)

**Patient** (patient\_id, patient\_firstName, patient\_lastName, patient\_grade, patient\_sex, patient\_birthDate, patient\_isActive, patient\_code, patient\_createdAt, patient\_updatedAt, #user\_id)

**Note** (note\_id, note\_estimation, note\_comment, note\_createdAt, note\_updatedAt, #user\_id, #patient\_id)

**Phrase** (phrase\_id, phrase\_text, phrase\_createdAt, phrase\_updatedAt, #patient\_id, #question\_question\_id)

**AudioPhrase** (audioPhrase\_id, audioPhrase\_score, audioPhrase\_createdAt, audioPhrase\_updatedAt, #phrase\_id, #audiomedia\_amedia\_id)

**Question** (question\_id, question\_text, question\_createdAt, question\_updatedAt)

**Category** (category\_id, category\_title, category\_superCategoryId, category\_createdAt, category\_updatedAt, #categorymedia\_cmedia\_id)

**Pictogram** (pictogram\_id, pictogram\_title, pictogram\_type, pictogram\_createdAt, pictogram\_updatedAt, #category\_id, #irregular\_irregular\_id, #pictogrammedia\_pmedia\_id)

**Tag** (tag\_id, tag\_title)

**Irregular** (irregular\_id, irregular\_feminin, irregular\_plurial, irregular\_pastParticiple, #pictogram\_pictogram\_id)

**Conjugation** (conjugation\_id, conjugation\_tense, conjugation\_firstPersonSingular, conjugation\_firstPersonPlurial, conjugation\_secondPersonSingular, conjugation\_secondPersonPlurial, conjugation\_thirdPersonSingular, conjugation\_thirdPersonPlurial, #irregular\_id)

**AudioMedia** (aMedia\_id, aMedia\_audioName, #audiophrase\_audiophrase\_id)

**CategoryMedia** (cMedia\_id, cMedia\_imageName, #category\_category\_id)

**PictogramMedia** (pMedia\_id, pMedia\_imageName, #pictogram\_pictogram\_id)

**Role** (role\_id, role\_title)

**Word** (word\_id, word\_place\_num, #phrase\_id, #pictogram\_id)

**is\_assotiated\_with** (category\_id, question\_id)

**includes** (tag\_id, pictogram\_id)

**user\_has\_role** (user\_id, role\_id)

## 4. Diagramme de classes

J’ai ensuite réalisé un diagramme de classes afin de modéliser les objets de mon système et les relations entre eux.

Изображение выглядит как текст, диаграмма, План, Параллельный

Автоматически созданное описание

## 5. Diagrammes d’activité

Afin de modéliser les scénarios et les étapes de décisions de mes fonctionnalités, j’ai réalisé les diagrammes d’activités. En voici 3 exemples :

**Activité Inscription**

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

**Activité Connection**

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание

**Activité Session**

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

## 6. Diagrammes de séquence

Pour représenter comment les objets du système interagissent pour un scénario particulier, et les messages échangés, j’ai réalisé des diagrammes de séquence. En voici 2 exemples :

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, диаграмма, снимок экрана, число

Автоматически созданное описание

## 7. Diagramme de composant

J’ai réalisé le diagramme de composants afin de représenter les relations entre les différents composants d'un système logiciel..

Изображение выглядит как диаграмма, текст, линия, План

Автоматически созданное описание

## 8. Diagramme de déploiement

J’ai réalisé le diagramme de déploiement afin de représenter les infrastructures physiques et logicielles du système.

Изображение выглядит как диаграмма, линия, План, Параллельный

Автоматически созданное описание

## 9. Maquettes du projet

Voici les maquettes que nous avons produites en conservant la charte graphique.

**Design Desktop :**

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Операционная система, программное обеспечение

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

**Design Tablette :**

Изображение выглядит как текст, снимок экрана, Шрифт, веб-страница

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

# VI. Spécifications techniques.

## 1. Techologies choisies

Pour le projet, voici les technologies qui ont été retenues :

● Spring Boot pour le backend en architecture orientée services (SOA)

● Spring Security pour la sécurité

● Next.js pour le frontend

● MySQL et NoSQL (MongoDB) pour les bases de données

● Spring Data JPA et Spring Data MongoDB pour la gestion de persistance des données

**Spring Boot** est un framework open-source pour le développement d'applications Java qui permet de créer facilement des applications en simplifiant le processus de configuration et de déploiement. En effet, il fournit une plate-forme préconfigurée (Spring Initializr) avec une configuration minimale à laquelle on peut ajouter des dépendances spécifiques. Il intègre le framework Spring qui fournit une architecture complète pour la création d'applications d'entreprise robustes et évolutives.

**Spring Security** est un module de sécurité puissant et hautement configurable pour les applications Java. Il offre des fonctionnalités pour protéger les applications des attaques courantes, sécuriser les processus d’authentification, définir des rôles et autorisations …

**Next.js** est un framework de développement web basé sur **React**, conçu pour permettre la création d'applications web modernes et performantes. Il propose des fonctionnalités avancées telles que le rendu côté serveur (SSR), la génération de sites statiques (SSG), le routage dynamique, et l'optimisation automatique des performances. Next.js facilite également la gestion des données avec des API intégrées et offre une expérience de développement améliorée grâce à des outils de débogage et des configurations simplifiées.

**MySQL** est un système de gestion de base de données relationnelle (**SGBDR**) open-source pour gérer et stocker des données structurées. Les données sont organisées sous forme de tables relationnelles en utilisant le langage SQL.

**Spring Data JPA** est un module de Spring Data qui simplifie l'accès aux bases de données relationnelles dans les applications Java basées sur le framework Spring. Il facilite la création de requêtes, l'interaction avec la base de données et la gestion des entités persistantes.

**MongoDB** est une base de données NoSQL orientée documents qui utilise des documents de type JSON avec des schémas dynamiques. Cela signifie que vous pouvez stocker vos données sous une forme flexible, sans avoir besoin de les structurer au préalable. MongoDB est conçu pour la scalabilité horizontale, les performances élevées et la facilité de développement, ce qui en fait un choix populaire pour les applications modernes nécessitant une gestion efficace de grandes quantités de données non structurées.

**Spring Data MongoDB** est un module du projet Spring Data, qui fournit une intégration facile et puissante entre l'écosystème Spring et MongoDB. Il simplifie l'accès aux données en offrant des abstractions de haut niveau pour la manipulation des documents MongoDB, tout en tirant parti des fonctionnalités de Spring telles que l'injection de dépendances et la gestion des transactions. Spring Data MongoDB permet aux développeurs de travailler avec MongoDB de manière intuitive et efficace, en utilisant des concepts familiers de Spring et des annotations spécifiques pour la gestion des documents.

## 2. Architecture orientée services de l’application côté serveur

**Architecture orientée services (SOA)** - est un style architectural dans le développement de logiciels qui repose sur l'utilisation de services indépendants. Chaque service est une unité fonctionnelle distincte qui interagit avec d'autres services via des interfaces et des protocoles spécifiques. La SOA permet de créer un système flexible et évolutif où les composants peuvent être facilement remplacés ou mis à jour sans affecter l'ensemble du système.

Dans notre fichier ***pom.xml*** racine, nous définissons tous les paramètres globaux du projet:

* projet **parent** dont les paramètres sont hérités la version du framework utilisée ;
* l’ **identifiant du groupe**, de l’artefact de projet, la version, la description du projet ;
* la liste des **modules** du projet ;
* la gestion des **dépendances** pour le projet ;
* les **propriétés** du projet tels que les versions du langage et/ou des modules Spring Boot utilisés ;

etc ...

Изображение выглядит как текст, снимок экрана, документ, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

En plus des modules standard décrivant la logique métier de l'application (dans notre cas, il y en a trois : le module **Category-Question-Pictogram-service**, le module **User-Institution--Patient-Note-service** et le module **Phrase-service**), cette architecture utilise des modules de service supplémentaires qui assurent l'interaction entre les modules de logique métier et offrent au projet évolutivité et haute stabilité. Dans notre cas, nous avons utilisé deux de ces modules: Discovery-server et Api-gateway.

**Discovery-server** : Un service utilisé pour enregistrer et découvrir dynamiquement les services au sein du système. Il permet aux services de trouver et de communiquer avec d'autres services sans configuration statique.

**Api-gateway** : Un service qui agit comme un point d'entrée unique pour les clients, gérant les demandes et les acheminant vers les services appropriés. Il peut également fournir des fonctionnalités supplémentaires telles que l'authentification, la surveillance et la gestion des versions des API.

Il est également considéré comme une bonne pratique d'externaliser la configuration du projet, où presque toute la configuration des modules est stockée dans un dépôt GitHub séparé et est accessible via un autre module de service supplémentaire, Config-server.

***Config-server*** : Un service qui fournit une gestion centralisée de la configuration pour les applications distribuées. Il permet de stocker les configurations dans un dépôt Git (ou un autre dépôt) et de les distribuer dynamiquement aux différents modules de l'application.

Cependant, cette approche n'est pas utilisée dans notre projet.

## 3. Processus de travail

Изображение выглядит как диаграмма, зарисовка, Технический чертеж, План

Автоматически созданное описание

**Requête du client** : Le client envoie une requête HTTP à ***API-Gateway*** pour accéder aux fonctionnalités de l'application.

**Routage de la requête** : ***API-Gateway*** reçoit la requête et l'analyse pour déterminer vers quel service de logique métier API-REST il doit être dirigé. Cela peut être basé sur l'URL de la requête, les en-têtes, les paramètres de requête ou d'autres métadonnées.

**Recherche du service** : Ensuite, ***API-Gateway*** interroge ***Discovery-Server*** pour déterminer l'adresse et le port du service qui traitera cette requête. ***Discovery-Server*** maintient généralement un registre des services enregistrés et de leurs métadonnées, telles que l'URL et la version.

**Redirection de la requête** : Une fois que ***Discovery-Server*** a renvoyé les informations sur l'emplacement du service, ***API-Gateway*** redirige la requête vers le service correspondant en utilisant l'adresse obtenue.

**Traitement de la requête** : Le ***service de logique métier API-REST*** reçoit la requête et la traite en fonction de la logique métier de l'application. Il peut extraire des données de la base de données, effectuer des opérations métier et générer une réponse.

**Formation de la réponse** : Après avoir effectué les opérations nécessaires, le service crée une réponse HTTP contenant les résultats de l'opération ou les données demandées.

**Retour de la réponse au client** : Le service renvoie la réponse via ***API-Gateway***, qui la transmet ensuite au client.

Ainsi, ***API-Gateway*** joue le rôle de point d'entrée pour les requêtes des clients, tandis que ***Discovery-Server*** assure la découverte et le routage dynamiques vers les services qui traitent ces requêtes. Ces composants interagissent pour fournir un accès transparent aux clients aux fonctionnalités de l'application, en masquant la complexité de sa structure interne et en assurant la scalabilité et la flexibilité.

Chaque service de la logique métier respecte l’architecture n-tiers :

**Couche de présentation** (package ***controller***) **:** Les contrôleurs API REST sont responsables du traitement des demandes des clients et de la fourniture des réponses appropriées. Ils servent de points d'entrée pour les clients, en recevant les demandes HTTP, en traitant les entrées et en renvoyant les réponses HTTP correspondantes.

**Couche métier** (packages ***service*** et ***dto***) **:** Cette couche est responsable de la logique métier de l'application. Elle est composée des différentes classes “**service**” qui implémentent la logique métier de l'application. Ces services contiennent les règles métier et les processus principaux de l'application, en orchestrant les opérations nécessaires pour satisfaire les requêtes des clients. Les **DTO** (Data Transfer Objects) sont utilisés pour transférer des données entre différentes composantes à l'intérieur de l'application, telles que les services et les référentiels. Ils peuvent être utilisés pour transmettre des données entre différents modules de logique métier, pour décrire les requêtes et les réponses entre les couches, ainsi que pour harmoniser les données entre différentes composantes.

**Couche d’accès aux données** (packages ***repository*** et ***model***) **:** Cette couche est responsable de l'accès aux données de l'application. Elle est composée des classes “repository” qui fournissent des interfaces pour accéder aux données stockées dans des bases de données en manipulant les objets (“entity”) qui correspondent aux tables de notre base de données. Les repositories permettent de récupérer, sauvegarder, mettre à jour et supprimer les données de manière abstraite.

**Couche de configuration** **:** Cette couche comprend les fichiers de configuration de l'application Spring. Il s'agit notamment du fichier “***pom.xml***”, des fichiers de propriétés “***application.properties(yml)***” et “messages.properties”, des ***annotations*** et des classes Java de configuration. Ces fichiers contiennent des informations sur la configuration de l'application, telles que les beans Spring ou les sources de données.

**Couche d’infrastructure** **:** Cette couche comprend les composants qui fournissent des fonctionnalités transversales à l'ensemble de l'application. Il s'agit notamment des composants de sécurité dans le package “security”, ainsi que des autres aspects transversaux comme la gestion des logs, la surveillance et la gestion des transactions.

## 4. Sécurité

La sécurité est restée une de nos principales préoccupations dans le développement. En effet, nous avons mis en place plusieurs choses pour sécuriser au maximum l’application :

**Injection SQL :**

L’injection SQL est une attaque très courante qui est largement répandue sur le Web. En injectant du code SQL dans les inputs, l’attaquant peut ainsi manipuler la base de données sous-jacente de l'application, récupérer des informations sensibles, effectuer des opérations non autorisées ou contourner l’authentification avec par exemple ‘OR 1=1–;.

Afin de se protéger des injections SQL nous avons utilisé Hibernate qui est un framework pour java, qui implémente la spécification Spring Data JPA. Hibernate fournit une couche d'abstraction qui gère les requêtes paramétrées et l’échappement automatique. Cela protège l’application contre les injections SQL.

**Injection XSS (Cross-Site Scripting) :**

L’injection XSS permet d’injecter du code HTML (et donc aussi d'exécuter des scripts) afin de modifier le contenu de la page. Cela se fait généralement dans les inputs de type “commentaires” pour que les modifications apportées par l’attaquant soient disponibles pour l’ensemble des autres utilisateurs. Cela permet par exemple d’insérer des liens malveillants pour de l’hameçonnage ou encore récupérer une session via les cookies.

Afin de se protéger des injections XSS nous avons utilisé les contraintes de validation sur les objets correspondant aux inputs avec un “regex pattern” pour interdire certains caractères spéciaux (les chevrons notamment) .

**Attaques CSRF (Cross-Site Request Forgery) :**

L'attaque CSRF est une technique utilisée par les pirates informatiques pour tromper les utilisateurs d'un site web afin d'effectuer des actions non désirées à leur insu. L'attaque se produit lorsque le pirate parvient à faire envoyer une requête malveillante depuis le navigateur de la victime vers un site web légitime auquel elle est connectée, exploitant ainsi la confiance accordée au site. Cette attaque permet au pirate d'exécuter des actions au nom de la victime, telles que le changement de mot de passe, l'envoi de messages ou même l'achat de produits, compromettant ainsi la sécurité et l'intégrité des données de la victime.

Next.js, utilisé en conjonction avec Spring Boot, permet de se prémunir contre les attaques CSRF. En effet, Spring Boot, peut générer automatiquement des jetons CSRF et les fournir via une API. Next.js peut ensuite récupérer ces jetons et les inclure dans les formulaires et les requêtes API. Spring Boot fournit également des filtres de validation CSRF qui vérifient la présence et la validité des jetons CSRF lors de la réception des requêtes.

**Contrôle des données saisies :**

Toujours avec les contraintes de validation, on s’est assuré que l'utilisateur ne puisse pas saisir n’importe quelles valeurs, exemples : un numéro de téléphone ne peut pas contenir de lettres, une adresse mail doit comporter un “@”, etc.

**Création de compte :**

Lors de la création de compte, plusieurs choses ont été mises en place pour la sécurité :

● Vérification du pseudonyme et de l’adresse mail pour s’assurer qu’ils n’existent pas déjà

● Activation du compte à l’aide d’un lien envoyé par email

● Hachage du mot de passe avant de le stocker et contraintes de validation pour imposer sa complexité

**Autorisations**:

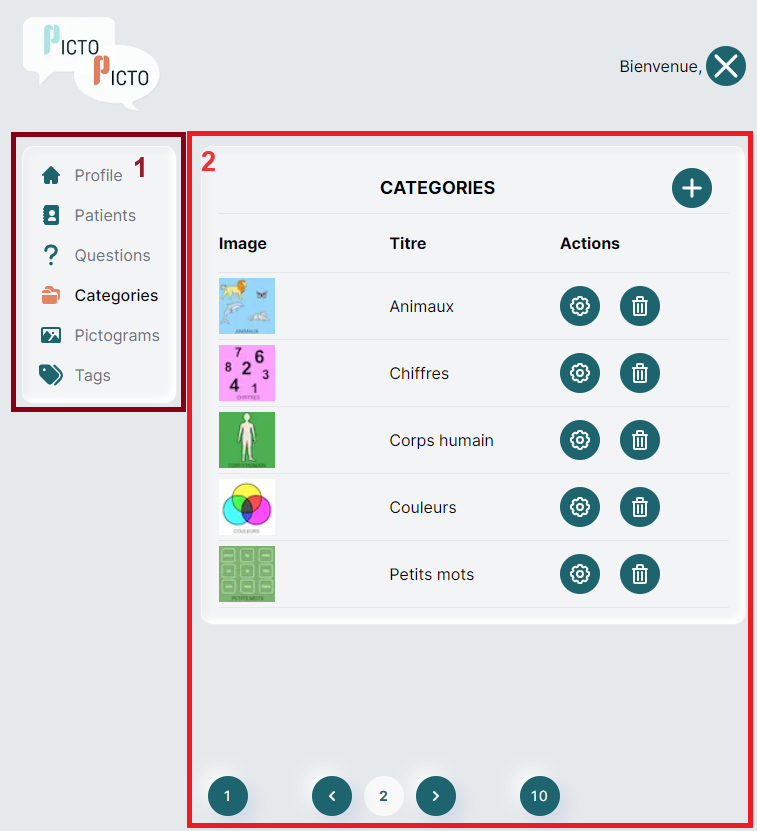
Un rôle est attribué à chaque utilisateur. Par défaut il a le rôle “User” et est limité dans les actions qu’il peut exécuter. Le rôle “Superadmin” quant à lui, donne des autorisations spécifiques supplémentaires comme la gestion aux autres professionnels de santé.

# VII. Réalisations

## 1. Partie front-end (côte client)

Pour la réalisation de l'interface utilisateur, j'ai utilisé le framework **Next.js** basé sur React. Ci-dessous, le principe de fonctionnement de la partie cliente de notre application est illustré à travers l'exemple du traitement de la liste des catégories.

Après s'être connecté à son compte, l'utilisateur-thérapeute accède à la page de tableau de bord (dashboard) **(1)** contenant une liste de toutes les données disponibles, telles que les catégories, les questions, les pictogrammes et les patients. Lorsqu'il sélectionne les informations souhaitées dans cette liste (dans notre cas, nous voulons obtenir la liste des catégories), une requête est envoyée à la partie serveur de l'application avant le rendu de la page. Après traitement de la requête, le serveur renvoie la liste des catégories au client **(2).**

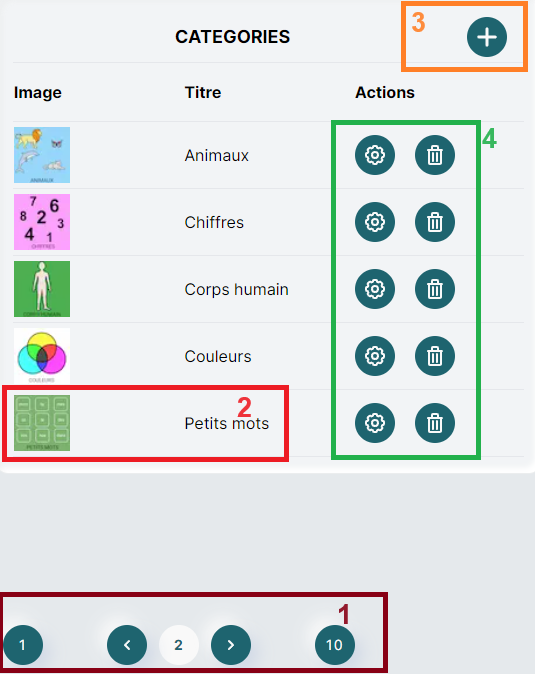
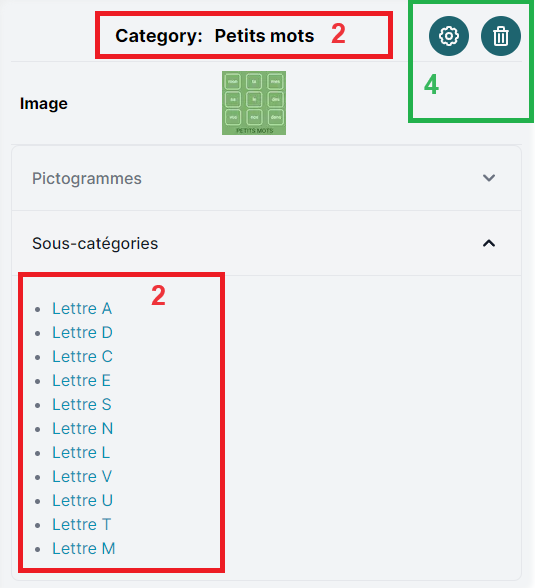


Par default, tous les pages sont les composants serveur. Le **Server Side Rendering (SSR)** est une technique utilisée pour générer le HTML de l'application sur le serveur plutôt que sur le client. Cela permet d'améliorer les performances et le SEO (optimisation pour les moteurs de recherche). Avec SSR, lorsque l'utilisateur demande une page, le serveur traite la requête, génère le HTML complet et l'envoie au navigateur. Next.js facilite l'implémentation de SSR en React en permettant aux développeurs de choisir quelles pages doivent être pré-rendues sur le serveur et lesquelles doivent être rendues côté client.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Автоматически созданное описание

Ensuite, sur la page de la liste des catégories, plusieurs autres opérations sont disponibles. Comme nous demandons au serveur la liste des catégories sous forme de pages paginées pour le tableau de bord, nous avons la possibilité de parcourir notre liste de catégories **(1)**. En plus de cela, nous pouvons obtenir des informations plus détaillées sur chaque catégorie **(2)**. Pour ce faire, il suffit de cliquer sur la catégorie souhaitée, et selon le même processus décrit précédemment, le serveur enverra au client des informations détaillées sur la catégorie spécifique.

**(2) =>** 

Nous pouvons également créer une nouvelle catégorie **(3)**, mettre à jour une catégorie existante ou la supprimer **(4)**.

## 2. Partie back-end (côte serveur) – API Gateway

Lorsqu'une requête provenant du client atteint la partie serveur de l'application, elle est traitée uniquement et exclusivement par le **API Gateway**.

Fichier de configurtation ***application.yml***

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Fichier ***CategoryController***

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

***@CrossOrigin(origins = "http://localhost:3000")*** - Cette annotation est utilisée pour permettre les requêtes Cross-Origin Resource Sharing (CORS) provenant du domaine spécifié (dans ce cas, http://localhost:3000). Cela permet à un navigateur d'accéder aux ressources depuis un domaine différent de celui où l'application web est hébergée.

***@RestController*** - Cette annotation indique que la classe est un contrôleur Spring MVC où chaque méthode renvoie directement une réponse de type JSON ou XML. Cela combine les annotations @Controller et @ResponseBody.

**@RequestMapping("/api/gateway/categories")** - Cette annotation est utilisée pour mapper les requêtes HTTP vers les méthodes du contrôleur. Ici, elle spécifie que toutes les requêtes avec l'URI commençant par /api/gateway/categories seront dirigées vers ce contrôleur.

***@GetMapping*** - Cette annotation est une version spécialisée de @RequestMapping pour les requêtes HTTP GET. Elle mappe les requêtes HTTP GET vers des méthodes de gestionnaires spécifiques.

***@RequestParam*** - Cette annotation est utilisée pour extraire les paramètres de requête des URL et les associer aux paramètres de méthode. Dans ce cas, elle extrait les paramètres page, size et isSeance de la requête.

***@PathVariable*** - Cette annotation est utilisée pour extraire des valeurs de variable d'URI et les associer aux paramètres de méthode. Dans ce cas, elle extrait la valeur de la variable superId de l'URI.

***@PostMapping*** - Cette annotation est une version spécialisée de @RequestMapping pour les requêtes HTTP POST. Elle mappe les requêtes HTTP POST vers des méthodes de gestionnaires spécifiques.

***@ResponseStatus(HttpStatus.CREATED)*** - Cette annotation permet de définir le statut de réponse HTTP pour une méthode de gestionnaire spécifique. Ici, elle indique que la méthode renverra un statut HTTP 201 (Created) lorsque l'entité sera créée avec succès.

***@RequestBody*** - Cette annotation est utilisée pour associer le corps de la requête HTTP au paramètre de méthode. Dans ce cas, elle associe le contenu JSON ou XML du corps de la requête à l'objet CategoryDetails.

La requête est dirigée vers un contrôleur et, en fonction de l'URI de la requête, elle est redirigée vers la méthode appropriée du service api-gateway, qui l'envoie ensuite vers l'instance la moins chargée du service de logique métier correspondant. Cette tâche est assurée par le **Discovery-server**. Après l'exécution de toutes les étapes dans les services de logique métier concernés (qui seront examinés un peu plus loin), la réponse est renvoyée au api-gateway, où elle est éventuellement transformée (si nécessaire), puis envoyée au client.

Изображение выглядит как текст, снимок экрана, документ, Шрифт

Автоматически созданное описание

***@Component*** - Cette annotation indique que la classe annotée est un composant Spring. Elle sert à marquer une classe comme étant un candidat pour la détection automatique de composants lors de la configuration basée sur les annotations et le balayage de classpath.

On injecte une dépendance ***webClientBuilder*** dans une classe CategoryService de API Gateway.

private final WebClient.Builder webClientBuilder;

public CategoryService(WebClient.Builder webClientBuilder) {

this.webClientBuilder = webClientBuilder;

}

Pourquoi est-ce utile?

***Injection de dépendances:*** Utiliser un constructeur pour initialiser webClientBuilder permet l'injection de dépendances, une technique courante en programmation orientée objet et en particulier dans les frameworks comme Spring. Cela rend le code plus modulaire et testable.

***Requêtes HTTP:*** WebClient est utilisé pour envoyer des requêtes HTTP de manière asynchrone et non bloquante. WebClient.Builder permet de configurer et de créer des instances de WebClient avec les paramètres nécessaires (comme les URI de base, les en-têtes, etc.).

En résumé, ces lignes de code sont essentielles pour initialiser le client Web utilisé par CategoryService pour effectuer des appels aux autres services de l'application.

Pour récupérer les categories, on execute cette méthode.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Voici une explication étape par étape :

* **Création d'un WebClient**

On utilise webClientBuilder pour construire un nouveau client Web (WebClient). Ce client servira à envoyer des requêtes HTTP.

* **Envoi de la requête GET**

On envoie une requête GET à l'URI spécifiée. Les paramètres number et size sont injectés dans l'URI pour définir la pagination.

* **Récupération de la réponse HTTP**

La réponse de la requête est récupérée et transformée en un Mono<CategoryPageResponse>. Cela signifie que nous attendons une réponse contenant une seule instance de CategoryPageResponse.

* **Transformation de la réponse avec flatMap**

Une fois la CategoryPageResponse obtenue, flatMap est utilisé pour transformer cette réponse de manière asynchrone.

* **Conversion de chaque catégorie en Mono<CategoryDetails>**

Le contenu de CategoryPageResponse (qui est une liste de catégories) est converti en une liste de Mono<CategoryDetails> en utilisant la méthode convertCategoryDetails. Cette méthode envoie des requêtes supplémentaires pour récupérer des détails sur chaque catégorie.

* **Combinaison des Monos avec Mono.zip**

Mono.zip est utilisé pour combiner les Mono<CategoryDetails> en un seul Mono.

La fonction fournie à Mono.zip transforme le tableau de résultats en une liste de CategoryDetails.

Le contenu de CategoryPageResponse est mis à jour avec les CategoryDetails enrichis.

La méthode retourne enfin le CategoryPageResponse mis à jour.

Cette méthode envoie une requête paginée pour récupérer une page de catégories. Pour chaque catégorie, elle récupère ses détails complets en utilisant convertCategoryDetails. Les résultats asynchrones sont ensuite combinés et la réponse de la page de catégories est mise à jour avec les détails complets de chaque catégorie, avant d'être renvoyée. Cette approche asynchrone et non bloquante permet une meilleure performance et une utilisation optimale des ressources.

## 3. Partie back-end (côte serveur) – Discovery server

**Discovery server** est un composant clé à la fois dans l'architecture orientée services (SOA) et dans l'architecture microservices. Sa principale fonction est de permettre l'enregistrement automatique et la découverte des microservices, facilitant ainsi aux autres services et aux clients de trouver et d'utiliser ces microservices sans avoir besoin de connaître leurs adresses et ports exacts.

Изображение выглядит как текст, Шрифт, линия, снимок экрана

Автоматически созданное описание

Ce code Java démarre une application Spring Boot en tant que Discovery Server avec la configuration nécessaire pour fonctionner en tant que serveur Eureka.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Ces configurations permettent au Discovery Server (Eureka Server) de fonctionner correctement dans votre application Spring Boot, facilitant ainsi l'enregistrement et la découverte des microservices au sein de votre architecture SOA.

**Rôle du Discovery server** **dans l'écosystème SOA :**

* Enregistrement des services : Chaque microservice, lors de son démarrage, s'enregistre auprès du serveur de découverte en lui fournissant son adresse réseau, son port et d'autres métadonnées comme sa version et son chemin contextuel.
* Découverte des services : Les applications clientes et les autres microservices peuvent découvrir dynamiquement les services disponibles en interrogeant le serveur de découverte. Cela élimine le besoin de stocker des configurations statiques dans les applications clientes et simplifie la mise à l'échelle et la maintenance du système.
* Gestion du cycle de vie : Le serveur de découverte surveille l'état des services enregistrés. Il peut détecter quand les services deviennent disponibles ou non disponibles, et les retirer ou les ajouter au registre en fonction de leur état.

**Communication avec API Gateway**

**API Gateway** joue le rôle de point d'entrée central dans le système, gérant le contrôle et la gestion du trafic et des requêtes externes. Dans le contexte de SOA, **API Gateway** utilise le **Discovery server** pour la dynamique de la découverte et du routage des requêtes vers les microservices internes :

1. **Routage dynamique :**

API Gateway utilise les informations fournies par Discovery server pour déterminer quels microservices sont disponibles et comment y accéder à tout moment. Cela permet à API Gateway de s'adapter automatiquement aux changements dans la structure et la configuration des microservices sans avoir besoin de modifier manuellement les paramètres de routage.

2. **Équilibrage de charge :**

Discovery server peut également fournir des informations sur les instances disponibles de chaque microservice, permettant ainsi à API Gateway d'effectuer l'équilibrage de charge et de sélectionner l'instance la plus appropriée pour traiter une requête spécifique.

Ensemble, ces fonctionnalités assurent la flexibilité, la scalabilité et la fiabilité du système basé sur SOA, simplifiant ainsi le développement et la maintenance des applications basées sur microservices.

## 4. Partie back-end (côte serveur) – REST API services de la logique métier

Nous allons envisager l'implémentation de la logique métier à travers l'exemple d'un service API-REST appelé "Category-Question-Pictogram-service".

Une requête HTTP arrive sur le service depuis ***API-Gateway*** et est dirigée vers la classe-contrôleur. Ci-dessous, nous présentons la classe-contrôleur pour la catégorie.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

***@RestController*** – Сette annotation indique que la classe est un contrôleur REST. Elle combine les annotations ***@Controller*** et ***@ResponseBody***, ce qui signifie qu'elle traite les requêtes HTTP REST et renvoie directement le résultat sous forme de réponse HTTP sans passer par une vue.

***@RequestMapping("/api/v1/categories")*** - Cette annotation mappe les requêtes HTTP vers des méthodes spécifiques de la classe contrôleur. Dans ce cas, toutes les requêtes vers **l'URI** "/api/v1/categories" seront dirigées vers les méthodes de cette classe.

***@Autowired*** – Cette annotation est utilisée pour injecter des dépendances dans une classe. Dans ce cas, elle injecte une instance de CategoryServiceImpl dans le contrôleur.

***@GetMapping*** – Cette annotation indique qu'il s'agit d'une méthode traitant les requêtes HTTP GET vers l'URI spécifié dans ***@RequestMapping***. Elle répondra aux requêtes GET vers "/api/v1/categories".

***@RequestParam*** – Ces annotations récupèrent les paramètres de requête de l'URL et les associent aux variables locales de la méthode. @RequestParam récupère les valeurs des paramètres de requête dans l'URL. defaultValue spécifie une valeur par défaut si le paramètre n'est pas présent dans l'URL.

***@PathVariable*** – Cette annotation récupère les variables de chemin de l'URL et les associe aux variables locales de la méthode. @PathVariable récupère les valeurs des variables de chemin dans l'URL.

**@PostMapping** – indique qu'il s'agit d'une méthode traitant les requêtes HTTP POST.

**@ResponseStatus(HttpStatus.CREATED)** – spécifie que le statut de la réponse HTTP sera 201 (Créé) si la création est réussie.

**@RequestBody** – Cette annotation lie le corps de la requête HTTP à un objet Java, dans ce cas, un objet de type ***CategoryDTO***. L'objet JSON reçu dans le corps de la requête sera automatiquement converti en un objet CategoryDTO.

**@PutMapping** – indique qu'il s'agit d'une méthode traitant les requêtes HTTP PUT

**@DeleteMapping** – indique qu'il s'agit d'une méthode traitant les requêtes HTTP DELETE. Ces annotations définissent les URI vers lesquelles ces méthodes répondront.

Ensuite, le contrôleur transmet la demande à la classe-service correspondant, où les données de la demande sont structurées dans une classe-DTO approprié et traitées conformément à la logique métier concernant ce module de service. Voici la classe-DTO de catégorie et un exemple de quelques méthodes de la classe-service de catégorie.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Pour les ***DTO***, on utilise un enregistrement (***record***) au lieu d'une classe parce que l'enregistrement (record) en Java est conçu pour représenter des données et porte en lui la sémantique de **l'immutabilité**. Cela signifie qu'une fois l'objet enregistrement (record) créé, son état ne peut pas être modifié. Cette approche est appropriée pour les DTO, qui sont généralement utilisés pour transférer des données entre différentes composantes de l'application et ne doivent pas être modifiés pendant le transfert.

***@NotBlank*** – Cette annotation indique que le champ "imageName" ne doit pas être null et ne doit pas être une chaîne vide.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

***@Service*** – Cette annotation est utilisée pour marquer une classe comme un composant du service dans le contexte de Spring. Les classes annotées avec @Service sont automatiquement détectées lors du balayage des composants et instanciées en tant que beans dans le contexte de l'application.

***@Autowired*** – Cette annotation est utilisée pour injecter une dépendance dans une classe. Dans ce cas, les instances de ***CategoryRepository***, ***CategoryMediaRepository*** et ***QuestionRepository*** seront automatiquement injectées dans la classe CategoryServiceImpl.

***@Override*** – Cette annotation indique qu'une méthode est une redéfinition (ou une implémentation) d'une méthode de la classe parente ou de l'interface. Dans ce cas, les méthodes suivantes redéfinissent des méthodes de l'interface CategoryService.

Ensuite, après l'application de la logique métier concernant la catégorie, la demande est envoyée à la classe-repository (***DAO***) de catégorie, qui renvoie une réponse de la base de données sous forme de classe d'entité Category, laquelle est transmise en retour à travers la chaîne de retour via le service de classe vers le contrôleur de classe, puis à travers la passerelle API vers la partie client de l'application.

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Cette interface ***CategoryRepository*** étend l'interface ***JpaRepository***. JpaRepository est une interface de Spring Data JPA qui fournit des méthodes prédéfinies pour effectuer des opérations CRUD (Create, Read, Update, Delete) sur une entité donnée.

L'entité associée à cette interface est ***CategoryEntity***, qui est spécifiée comme premier paramètre générique de JpaRepository. Cela signifie que CategoryEntity est l'entité avec laquelle le repository va travailler et que l'identifiant de l'entité est de type Long.

Les méthodes déclarées dans cette interface fournissent différentes fonctionnalités pour interagir avec la base de données :

***findAll(Pageable pageable):*** Cette méthode récupère toutes les catégories de manière paginée en fonction des paramètres de pagination spécifiés dans l'objet Pageable. ***Pageable*** est un objet qui contient des informations sur la pagination, telles que le numéro de page, la taille de la page et les critères de tri.

***findAllBySupercategory(CategoryEntity cat):*** Cette méthode récupère toutes les sous-catégories pour une catégorie supérieure donnée spécifiée en paramètre.

***findAllBySupercategoryIsNull():*** Cette méthode récupère toutes les super-catégories, c'est-à-dire les catégories qui n'ont pas de catégories supérieures.

***findAllByQuestionsIsNotNull():*** Cette méthode récupère toutes les catégories qui sont attachées à des questions, c'est-à-dire les catégories qui ont au moins une question associée.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

***@Entity:*** L'annotation @Entity indique que la classe est une entité qui sera cartographiée sur une table dans la base de données. Dans ce cas, la classe CategoryEntity représente la table categories.

***@Table(name = "categories"):*** L'annotation @Table est utilisée pour spécifier des informations supplémentaires sur la table de la base de données, telles que son nom. Dans ce cas, @Table(name = "categories") indique que la table dans la base de données s'appellera "categories".

***@Id:*** L'annotation indique que le champ est un identifiant de l'entité. Dans ce cas, le champ id est l'identifiant de la table.

***@GeneratedValue(strategy = GenerationType.IDENTITY):*** L'annotation spécifie la stratégie de génération des valeurs pour l'identifiant. Dans ce cas, la stratégie GenerationType.IDENTITY est utilisée, ce qui permet à la base de données de générer automatiquement des identifiants uniques.

***@ManyToOne:*** L'annotation indique une relation "plusieurs à un" entre les entités. Dans ce cas, cela signifie qu'une catégorie peut avoir plusieurs sous-catégories.

***@JoinColumn(name = "supercategory\_id"):*** L'annotation est utilisée pour configurer la jointure entre les tables de la base de données. Dans ce cas, le nom de la colonne dans la table des catégories qui fait référence à l'identifiant de la catégorie parent est spécifié.

***@ManyToMany(mappedBy = "categories"):*** L'annotation indique une relation "plusieurs à plusieurs" entre les entités. Dans ce cas, cela signifie qu'une catégorie peut être liée à plusieurs questions, et qu'une question peut être liée à plusieurs catégories. Le paramètre mappedBy spécifie le nom du champ dans l'entité liée qui gère cette relation.

***@JsonSerialize(using = CategorySerializer.class):*** L'annotation est utilisée pour configurer le processus de sérialisation de l'objet en JSON. Dans ce cas, la classe CategorySerializer est spécifiée, ce qui définit la manière dont la catégorie est sérialisée.

***@OneToMany(mappedBy = "category"):*** L'annotation indique une relation "un à plusieurs" entre les entités. Dans ce cas, une catégorie peut avoir plusieurs pictogrammes. Le paramètre mappedBy spécifie le nom du champ dans l'entité liée qui gère cette relation.

J'utilise la bibliothèque **Lombok** pour le langage de programmation Java, qui génère automatiquement des méthodes d'accès et d'autres méthodes standard pour les classes Java en utilisant des annotations. Cela permet de réduire la quantité de code répétitif dans le projet et d'améliorer la productivité du développement. Ci-dessous sont les annotations Lombok utilisées dans le code fourni:

***@Getter:*** Génère des méthodes d'accès (getters) pour tous les champs de la classe.

***@Setter:*** Génère des méthodes de configuration (setters) pour tous les champs de la classe.

***@NoArgsConstructor:*** Génère un constructeur sans paramètres.

***@ToString:*** Génère une méthode toString(), qui renvoie une représentation sous forme de chaîne de caractères de l'objet, incluant les valeurs de tous ses champs.

# VIII. Jeu d’essai

Pour garantir la robustesse et la fiabilité de mes services, j’ai implémenté des tests unitaires et des tests d’intégration. Cette section décrit les étapes de réalisation de ces tests, leur but, et les aspects spécifiques qu'ils vérifient.

**Fichier de configuration des tests *application.yml***

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

## 1. Tests unitaires

Les tests unitaires visent à valider le bon fonctionnement des différentes parties isolées du code, telles que les méthodes des classes. Ils permettent de s’assurer que chaque unité de code fonctionne comme prévu indépendamment des autres composants du système.

**Outils Utilisés :**

* JUnit - un framework de test unitaire pour Java. Il permet d'écrire et d'exécuter des tests unitaires afin de valider le comportement attendu des différentes unités de code (méthodes, classes).
* Mockito - un framework de moquage pour Java. Il permet de créer des objets fictifs (mock objects) afin de simuler le comportement des dépendances dans les tests unitaires. Cela permet de tester une unité de code en isolation sans dépendre des implémentations réelles de ses dépendances.

J’ai structuré les tests unitaires en créant une classe de test pour chaque classe du service. Par exemple, pour la classe CategoryService, j’ai créé CategoryServiceTest.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

C’est une configuration des tests unitaires pour le CategoryService en utilisant les mocks pour les dépendances, permettant ainsi de tester le service en isolation sans dépendre des implémentations réelles de CategoryRepository et CategoryMediaService.

***@SpringBootTest*** :

**Définition** : Cette annotation permet de configurer un contexte d'application Spring pour les tests. Elle charge toutes les configurations de l'application et permet de tester l'application dans un environnement similaire à la production.

**Utilisation** : Dans le contexte des tests unitaires, @SpringBootTest est souvent utilisée pour s'assurer que toutes les configurations nécessaires sont chargées, bien qu'elle soit plus couramment utilisée pour les tests d'intégration.

***@ExtendWith(MockitoExtension.class)*** :

**Définition** : Cette annotation permet d'intégrer le support de Mockito dans les tests JUnit 5. Elle configure automatiquement les objets ***@Mock*** et ***@InjectMocks*** avant l'exécution des tests.

**Utilisation** : Elle est essentielle pour utiliser les annotations de Mockito comme @Mock et @InjectMocks.

***@Mock :***

**Définition** : Cette annotation est utilisée pour créer des objets factices (mocks) des dépendances nécessaires pour les tests. Elle permet de simuler le comportement des objets de manière contrôlée.

**Utilisation** :

*@Mock private CategoryRepository categoryRepository*; : Crée un mock pour le CategoryRepository afin de simuler l'accès à la base de données pour les catégories.

*@Mock private CategoryMediaService categoryMediaService*; : Crée un mock pour le CategoryMediaService pour simuler les opérations de gestion des médias associés aux catégories.

***@InjectMocks :***

**Définition** : Cette annotation permet d'injecter les mocks créés avec @Mock dans l'objet à tester. Elle initialise l'objet et injecte les mocks dans ses dépendances.

**Utilisation** :

*@InjectMocks private CategoryServiceImpl categoryService*; : Crée une instance de CategoryServiceImpl et injecte les mocks categoryRepository et categoryMediaService dans celle-ci.

**Test de la Récupération d’une Catégorie par ID :**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Ce test vérifie que la méthode *findOneById* du service récupère correctement une catégorie par son ID. Il simule la réponse du categoryRepository pour un ID donné et vérifie que le service retourne la catégorie attendue.

**Test de la Création d’une Catégorie :**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Ce test vérifie que la méthode *createOne* du service crée correctement une nouvelle catégorie. Il simule le processus de sauvegarde de la catégorie et la création d'un document média associé. Ensuite, il vérifie que les attributs de la catégorie créée correspondent aux attentes.

**Le résultat d’execution des tests unitaires :**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Ce que Vérifient les Tests Unitaires**

* La fonctionnalité attendue : Les tests vérifient que chaque unité de code produit les résultats attendus lorsque des données d'entrée spécifiques lui sont fournies.
* Les cas limites et les conditions spéciales : Ils examinent souvent le comportement des unités de code dans des situations extrêmes ou des cas spéciaux pour s'assurer que le code gère correctement ces conditions.
* La maintenabilité et la lisibilité : Bien que plus subjectifs, les tests unitaires peuvent indirectement évaluer la qualité du code en termes de maintenabilité et de lisibilité.
* La gestion des erreurs : Les tests vérifient si les unités de code génèrent des erreurs ou des exceptions lorsque cela est approprié, et s'ils les gèrent correctement.

## 2. Tests d’integration

Les tests d’intégration visent à vérifier le bon fonctionnement des différents composants du système lorsqu'ils sont combinés. Contrairement aux tests unitaires, ils testent le système de manière plus globale, incluant la base de données réelle et les interactions entre les composants.

**Outils Utilisés :**

* Spring Boot Test : Spring Boot Test fournit des annotations et des fonctionnalités pour écrire des tests d'intégration dans des applications Spring Boot. Il permet de configurer le contexte de l'application et de tester les interactions entre les composants.
* H2 Database : H2 est une base de données en mémoire qui peut être utilisée pour les tests. Elle permet de simuler une base de données réelle sans affecter la base de données de production.
* MockMvc : MockMvc est une classe de Spring qui permet de tester les contrôleurs REST de manière simple et rapide. Elle permet de simuler des requêtes HTTP et de vérifier les réponses.

J’ai structuré les tests d’intégration pour tester les endpoints REST du contrôleur des catégories. Par exemple, pour le contrôleur CategoryController, j’ai créé CategoryControllerIntegrationTest.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

C’est une configuration des tests d'intégration pour l'application en utilisant MockMvc pour tester les endpoints REST et en encapsulant chaque test dans une transaction pour garantir l'isolation des tests. Le CategoryMediaService est mocké pour contrôler son comportement lors des tests.

***@AutoConfigureMockMvc :***

**Définition :** Cette annotation permet de configurer automatiquement MockMvc, un outil de Spring pour tester les contrôleurs web. Elle configure l'environnement nécessaire pour les tests des endpoints REST.

**Utilisation :** Elle permet d'injecter MockMvc dans les tests pour simuler et tester les requêtes HTTP.

***@Transactional :***

**Définition :** Cette annotation permet d'encapsuler chaque test dans une transaction. La transaction est automatiquement annulée à la fin du test, garantissant que les modifications de la base de données effectuées pendant le test ne persistent pas.

**Utilisation :** Cela garantit que chaque test d'intégration démarre avec un état propre et n'affecte pas les autres tests en cours.

***@Autowired :***

**Définition :** Cette annotation permet d'injecter automatiquement les dépendances configurées par Spring dans les tests.

**Utilisation :**

@Autowired private MockMvc mockMvc; : Injecte l'instance de MockMvc pour permettre l'exécution de requêtes HTTP simulées.

@Autowired private CategoryRepository repository; : Injecte le repository réel utilisé par l'application pour interagir avec la base de données.

***@MockBean :***

**Définition :** Cette annotation est utilisée pour créer et injecter un mock dans le contexte de Spring. Contrairement à @Mock, @MockBean remplace le bean dans le contexte de Spring par un mock.

**Utilisation :** @MockBean CategoryMediaService categoryMediaService; : Crée un mock pour CategoryMediaService et le remplace dans le contexte de Spring. Cela permet de simuler les comportements du service de gestion des médias lors des tests d'intégration.

**Test de la Récupération de Toutes les Catégories :**

**Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание**

Ce test vérifie que l'endpoint GET /api/v1/categories retourne correctement une liste de catégories. Il crée une catégorie de test dans la base de données H2, effectue une requête GET, et vérifie que la réponse contient la catégorie attendue.

**Test de la Création d’une Catégorie via Multipart :**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Ce test vérifie que l'endpoint POST /api/v1/categories permet de créer correctement une nouvelle catégorie avec un fichier multipart. Il simule le processus de création d'un document média et vérifie que la réponse de l'API contient les attributs attendus de la nouvelle catégorie.

**Le résultat d’execution des tests d’integration :**

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Ce que Vérifient les Tests d'Intégration**

* Communication entre composants : Ils testent comment les différents composants ou modules d'un système interagissent les uns avec les autres, en s'assurant que les données et les messages sont transmis correctement.
* Flux de données : Les tests vérifient la manière dont les données circulent à travers le système, depuis leur entrée jusqu'à leur sortie, en s'assurant que toutes les transformations et manipulations de données se déroulent correctement.
* Intégration avec des systèmes externes : Ils testent l'intégration avec des systèmes externes, tels que des bases de données, des services web, des API externes, etc., pour s'assurer que l'intégration se fait correctement et que les données sont échangées de manière fiable.
* Scénarios réels : Ils simulent des scénarios réels d'utilisation du système pour vérifier son comportement dans des conditions proches de celles rencontrées en production.
* Gestion des erreurs et des exceptions : Les tests vérifient comment le système gère les erreurs et les exceptions, que ce soit au niveau des communications entre composants ou lors d'intégration avec des systèmes externes.

## 3. Conclusion

En conclusion, les tests unitaires et les tests d’intégration sont essentiels pour garantir la qualité et la fiabilité de l’application. Les tests unitaires permettent de valider chaque composant isolé, tandis que les tests d’intégration vérifient l’interaction entre ces composants. En utilisant ces deux types de tests, j’ai pu m’assurer que le service REST des catégories fonctionne correctement à la fois individuellement et en tant que partie intégrante du système complet.

# IX. Veille sur les vulnérabilités de sécurité

## 1. Méthodologie

## 2. Exemple de mise en place

# X. Démarches DevOps

# XI. Situation de travail ayant nécessité une recherche

## 1. Problème Rencontré

Lors du développement de notre application, j’ai rencontré un problème lié à la transmission de fichiers entre notre API Gateway et notre service REST. Le problème était dû à l'utilisation de deux stacks différents pour gérer les fichiers :

API Gateway utilise le stack réactif de Spring WebFlux, qui manipule les fichiers sous le format FilePart.

Service REST utilise le stack servlette de Spring MVC, qui manipule les fichiers sous le format MultipartFile.

Cette différence de format a causé des incompatibilités lors de la transmission des fichiers, rendant impossible le bon fonctionnement de notre application.

## 2. Étapes de Résolution

* **Identification du Problème :**

La première étape a été d'identifier clairement le problème. J’ai constaté que les fichiers envoyés depuis l'API Gateway (utilisant FilePart) n'étaient pas correctement reçus par le service REST (attendant des MultipartFile). Cette incompatibilité a été la cause de nombreux échecs dans nos tests d'intégration.

* **Étude de la Documentation Spring :**

Pour trouver une solution, j’ai d'abord consulté la documentation officielle de Spring. Bien que cette dernière fournisse des informations détaillées sur les deux stacks (réactif et servlette), elle ne proposait pas de solution "prête à l'emploi" pour convertir FilePart en MultipartFile.

* **Recherche en Ligne :**

J’ai ensuite utilisé des moteurs de recherche pour trouver des solutions à notre problème. J’ai porté une attention particulière aux forums de développeurs comme StackOverflow. Voici comment nous avons procédé :

* Formulation des Requêtes : J’ai formulé mes requêtes de recherche en utilisant des mots-clés précis, tels que "Spring WebFlux FilePart to MultipartFile" et "Spring reactive servlet file upload".
* Affinement des Requêtes : J’ai ajusté mes requêtes en fonction des résultats obtenus, en ajoutant ou modifiant des termes pour obtenir des réponses plus pertinentes.
* Examen des Résultats : J’ai passé en revue plusieurs solutions proposées par la communauté, en analysant celles qui semblaient les plus applicables à notre situation.
* **Solution Trouvée :**

Après une recherche approfondie, j’ai trouvé une discussion pertinente sur StackOverflow qui abordait un problème similaire : [How to stream file from multipart/form-data in Spring WebFlux](https://stackoverflow.com/questions/70408075/how-to-stream-file-from-multipart-form-data-in-spring-webflux).

Une partie essentielle de la solution impliquait l'utilisation de **MultipartBodyBuilder** pour composer les données dans un format lisible par le service REST.

**MultipartBodyBuilder** est une classe utilitaire fournie par Spring WebFlux qui permet de construire des requêtes multipart/form-data. Elle facilite l'assemblage des différentes parties d'une requête multipart, comme des champs de formulaire et des fichiers, dans un format compatible avec le serveur cible. Cela est particulièrement utile lorsque l'on doit envoyer des données complexes, comme des fichiers, dans une application réactive.

* **Adaptation et Implémentation de la Solution :**

J’ai adapté cette solution à notre projet pour formater les données de manière compatible avec notre service REST.

Voici comment j’ai implémenté cette solution :

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

**Résumé du Code :**

* Créer un MultipartBodyBuilder : Cette classe est utilisée pour construire une requête multipart/form-data en ajoutant les différentes parties (titre, supercatégorie, fichier image, et type de contenu).
* Ajouter des Parties à la Requête : Les informations comme le titre et la supercatégorie sont ajoutées en tant que parties simples, tandis que le fichier image est ajouté en utilisant asyncPart pour gérer le flux de données de manière réactive.
* Détermination du Type de Contenu : Une méthode determineContentType est utilisée pour définir le type de contenu du fichier basé sur son extension.
* Envoi de la Requête : La requête multipart est ensuite envoyée au service REST via WebClient, en spécifiant l'URI du service cible. Les données multipart sont construites à l'aide de BodyInserters.fromMultipartData.
* Traitement de la Réponse : La réponse du service est ensuite transformée en un objet CategoryDetailsResponse et traitée par la méthode convertCategoryDetails.
* **Tests et Validation :**

Après avoir implémenté cette solution, j’ai procédé à des tests rigoureux pour nous assurer que la conversion fonctionnait correctement et que les fichiers étaient transmis sans problème entre l'API Gateway et le service REST. Les tests ont confirmé que ma solution fonctionnait comme prévu, résolvant ainsi le problème de compatibilité.

## 3. Résumé

Grâce à une recherche méthodique et à l'utilisation des ressources en ligne, j’ai pu résoudre un problème complexe de compatibilité entre les stacks réactif et servlette de Spring. Cette expérience a renforcé ma capacité à trouver des solutions efficaces en utilisant les ressources disponibles sur internet et à adapter ces solutions à notre projet spécifique.

# XII. Conclusion

C'est avec un vif intérêt que j'ai participé à ce projet, y prenant énormément de plaisir. Il m'a permis de consolider mes compétences, d'approfondir mes connaissances et de progresser en tant que concepteur et développeur. De plus, j'ai pleinement réalisé l'importance cruciale de la phase de conception, en particulier pour la tâche qui m'a été confiée : transformer un projet en l’architecture MVC sur un projet en l’architecture orientée services (SOA). En effet, il s'est avéré essentiel de mener une réflexion approfondie lors de la conception afin d'anticiper la logique, de simplifier le code et de faciliter l'évolution future de l'application.

J’ai également acquis une expérience précieuse dans la gestion de projet. J'ai appris à m'adapter aux exigences changeantes, à respecter les délais et à collaborer efficacement avec des collègues aux compétences variées. J'ai également été exposé à la réalité du monde professionnel, en faisant face à des défis techniques et en apprenant à trouver des solutions créatives pour les surmonter.

Je suis reconnaissant d'avoir eu l'opportunité de contribuer à un projet qui a un impact réel et positif, et je suis prêt à relever de nouveaux défis dans le futur en tant que professionnel du développement d'applications.

Konstantin SHILKOV

le 11 juin 2024