Remerciements

Je tiens à remercier **El-Habib SOULAIMANA**, pour ton rôle en tant que formateur développeur web & mobile à l'AFPA de Brest, ton accompagnement tout au long de la formation et de mon stage.
Un grand merci pour ta pédagogie, tes conseils et ton engagement.

Je tiens à exprimer ma gratitude à **Yohann SERPAULT**, fondateur et dirigeant de Gemeny Software, pour la confiance que tu m'as accordé en me donnant l'opportunité de participer à ce beau projet, ainsi que tes nombreux conseils et ta disponibilité.
Pour tous ces enseignements, ces échanges que nous avons pu avoir autour du projet, encore merci Yohann.

Merci à tous mes camarades de formation, pour cette super ambiance et le soutien que ce collectif a représenté pour moi, en espérant avoir pu vous apporter autant que vous m'avez apporté.
Mention spéciale à mon binôme du duo des Nouilles.

À ma compagne Émilie qui m'a apporté son soutien dans ma démarche de reconversion, pendant mes moments de doutes et de réussites.

Sommaire

Liste des compétences

Ce projet couvre l'ensemble des compétences professionnelles requis pour l'obtention du titre professionnel "Développeur Web & Web Mobile".

Voici la liste des compétences par activité type :

1. **Développer la partie front-end d'une application web ou web mobile sécurisée**

    - Mettre en place mon environnement de travail
    - Maquetter des interfaces utilisateur web ou web mobile
    - Réaliser des interfaces utilisateur statiques web ou web mobile
    - Développer la partie dynamique des interfaces utilisateur web ou web mobile

2. **Développer la partie back-end d'une application web ou web mobile sécurisée**

    - Mettre en place une base de données relationnelle
    - Développer des composants d'accès aux données SQL et NoSQL
    - Développer des composants métier côté serveur

Contexte du projet

Présentation de l'entreprise Gemeny Software

Gemeny Software est une entreprise française innovante spécialisée dans la cybersécurité, plus précisément dans les solutions d'obfuscation de données sensibles et confidentielles. Fondée en 2023, la société est basée à Paris avec des racines opérationnelles à Brest, et s'inscrit dans la dynamique de la French Tech.

Son produit phare, le logiciel *Gemeny*, est un outil reconnu pour l'obfuscation avancée de fichiers, permettant de sécuriser efficacement les informations sensibles, aussi bien dans des contextes militaires, industriels que civils.

L'entreprise propose une solution 100% offline, ce qui garantit qu'aucune donnée n'est transmise à l'extérieur lors du traitement, répondant ainsi aux exigences les plus strictes en matière de confidentialité et de souveraineté numérique. Le logiciel s'adresse à tous types d'organisations (défense, santé, finance, etc.) qui souhaitent protéger leurs données contre les risques de fuite ou d'espionnage industriel.

L'activité est en phase de lancement avec un logiciel fonctionnel qui bénéficie de mises à jour régulières.

L'équipe Gemeny est composée de

- son fondateur Yohann Serpault qui travaille sur ce projet en parallèle de son activité d'ingénieur DevOps chez Thalès à Brest,
- de Tim Molivier, stagiaire en marketing digitale, qui a développé la stratégie commerciale de l'entreprise,
- de moi-même, stagiaire développeur web & web mobile.

Le produit Gemeny

Gemeny est une application monolithe qui permet l'obfuscation de données à partir de fichier type JSON en appliquant des règles, puis de récupérer le fichier de données obfusquées sur le répertoire choisi.

Il convient de déterminer ce qui constitue une règle d'obfuscation avant d'aller plus loin, car ces éléments sont essentiels.

Une règle comporte :

1. la source de données (data source) qui détermine quelle valeur sur le jeu de données sera modifié par cette règle (clé json par exemple)
2. l'expression personnalisée (custom regex) est un outil avancé mais simplifié qui vous permet de faire correspondre le résultat final à un schéma définit
3. des options d'aléatoires, pour remplacer la valeur par une autre depuis des données réalistes ou avec de l'aléatoire complet
4. la correspondance, si le jeu de données entrants comportent des éléments similaires, Gemeny respectera le schéma rencontré (exemple : le tracé d'un plan de vol de drone)

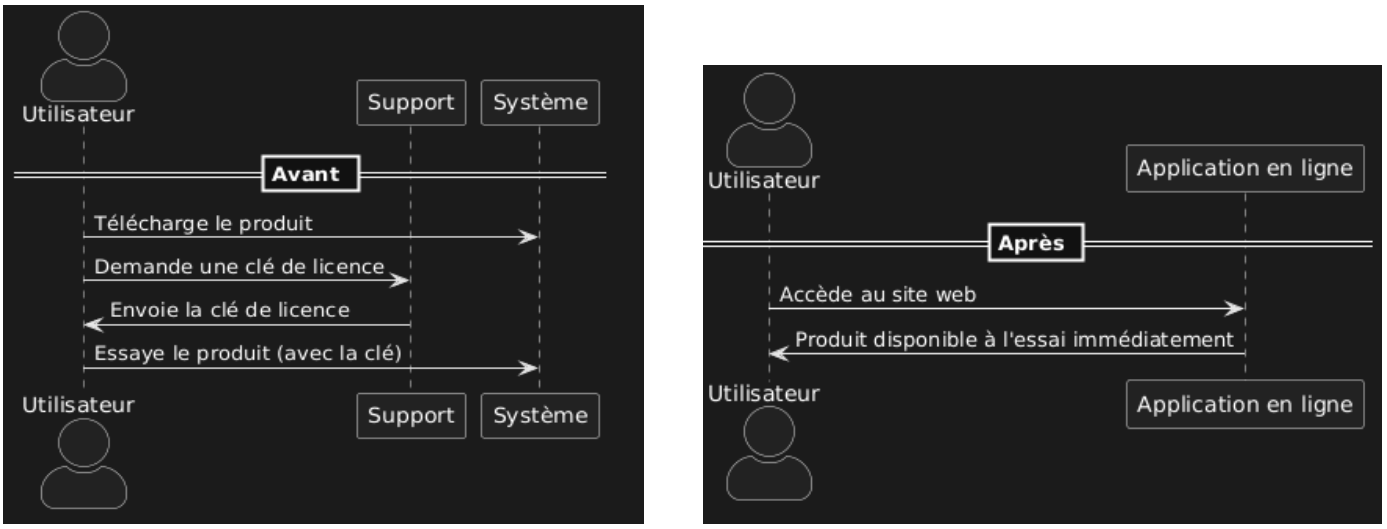
➔ **[Annexe p.1] Exemples d'obfuscation de données**

L'application est téléchargeable et utilisable hors ligne grâce à une clé de licence obtenable via un formulaire en ligne pour une version gratuite de 7 jours, ou via une souscription payante pour étendre la durée de la licence.

La clé de licence est propre à une seule adresse MAC, pour utiliser le produit sur plusieurs supports, des solutions sont disponibles.

Objectif du Projet

Objectif Général : Construire une plateforme web d'obfuscation de données, permettant aux utilisateurs d'avoir un accès au produit GEMENY, pour tester la version directement en ligne, pouvoir accéder à des fonctionnalités complémentaires. La plateforme doit être fonctionnelle, intuitive et répondre aux besoins des utilisateurs.

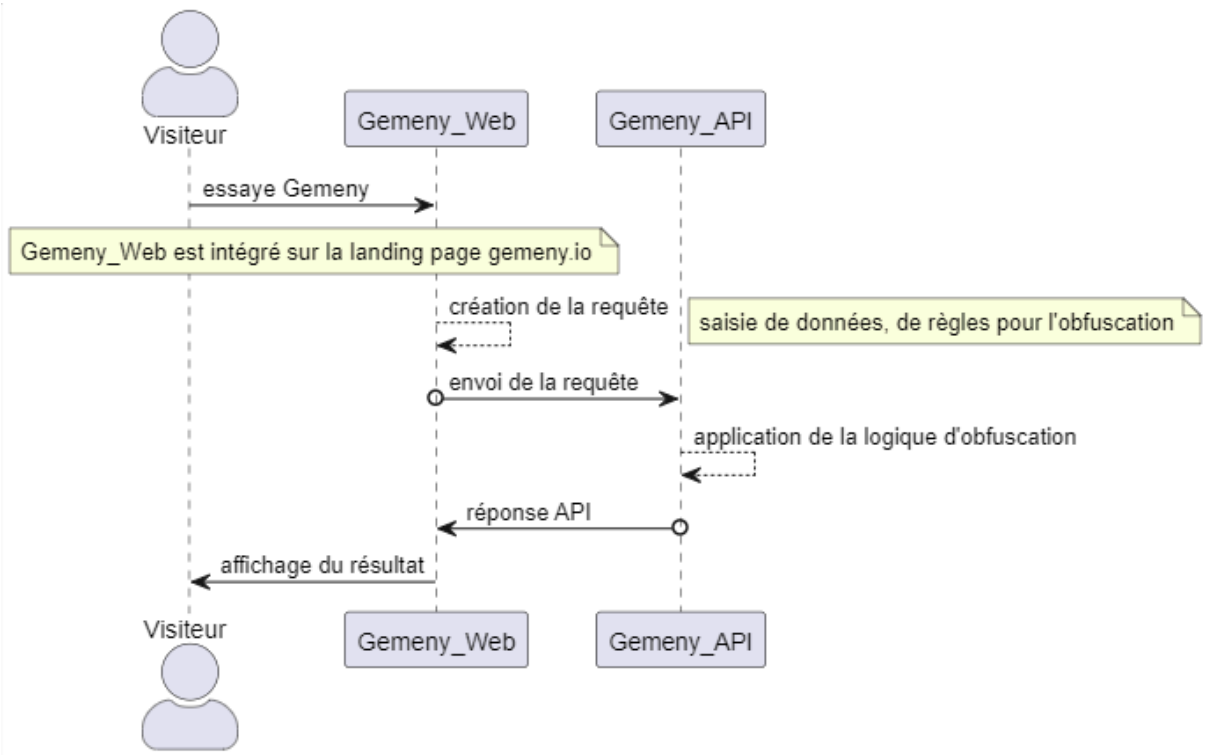


Séquence d'utilisateur pour essayer le logiciel Gemeny

L'ancien processus pour permettre aux potentiels clients d'essayer le produit Gemeny était plus laborieux, risquant une potentielle fuite au vu du nombre d'étapes à remplir, conformément à la règle des 3 clics.

Selon ce concept, un utilisateur doit pouvoir accéder au contenu recherché en trois clics maximum depuis la page d'accueil, sous peine de quitter le site par frustration ou lassitude.

Mise en place : Le site de l'entreprise **gemeny.io** est construit et hébergé avec WordPress. Afin de pouvoir être employé sur d'autres plateformes par la suite, je propose d'héberger l'interface web de Gemeny sur une page distincte, et de l'intégrer ensuite avec un iframe sur le site d'accueil des visiteurs du site. C'est la solution qui est retenue après concertation avec mon maître de stage lors de daily en début de projet.



Séquence d'utilisation pour l'essai de Gemeny en ligne

Fonctionnalités de Base (Minimum Viable Product - MVP) :

1. Tester le produit en ligne (fonctionnalité prioritaire) :

- **Objectif :** Permettre aux utilisateurs d'essayer le produit sans avoir recours à un téléchargement, à l'acquisition d'une licence temporaire, en passant par une interface en ligne.
- **Détails :**
  - **Choix de jeux de données :** Possibilité pour l'utilisateur d'utiliser des jeux de données prédéfinis comme point de départ, libre de modifier le document qui sera traité.
  - **Insertion de règles:** L'ajout de règles pour l'obfuscation est faite par l'utilisateur, de façon intuitive avec des aides à la complétion.
  - **Validation :** L'utilisateur envoie une première requête validant les données et règles renseignées, avant l'envoi de la requête d'obfuscation.
  - **Résultat :** Une fois la requête d'obfuscation traité, le résultat s'affiche pour comparer avec les données renseignées, permet de constater l'efficacité du produit, et une partie de ses capacités.

2. Gestion des Utilisateurs (pour potentiel usage futur):

- **Objectif :** Permettre aux utilisateurs de s'inscrire, de se connecter et de gérer leur profil.
- **Détails :**
  - **Inscription :** Formulaire d'inscription avec les champs obligatoires (nom, adresse, email, mot de passe). Validation des données côté client et serveur. Stockage sécurisé des mots de passe (hashage).
  - **Connexion :** Formulaire de connexion avec les champs email et mot de passe. Authentification des utilisateurs.
  - **Dashboard:** Interface permettant aux admin de modifier les permissions des utilisateurs.

Cadre de travail

Gemeny Software est une entreprise récente, et ne dispose pas encore de locaux, je travaille donc en télétravail depuis mon domicile et en open space depuis l'AFPA avec des collègues de formation.

Nous adoptons la méthode Agile avec mon maître de stage, avec un daily chaque matin pour échanger sur les avancées de chacun, les difficultés rencontrées et des propositions sur nos projets respectifs (design, format de données).

Les attendus du stage

L'attendu de l'entreprise pour mon stage est la conception, le développement et le déploiement de cette interface web, et ce sur un délai de 6 semaines correspondant à la date de présentation du produit lors du salon SOFINS 2025 auquel participe Gemeny Software pour présenter son produit **[Annexe p.23]**.

Le temps restant sur mon stage me permettra de continuer à développer des fonctionnalités sur cette interface, résoudre les éventuels bugs, et travailler sur une application Back-End de type API (*Application Programming Interface*) Rest pour la gestion des utilisateurs, avec une interface d'inscription, de connexion, et une page dashboard pour que les administrateurs puissent gérer les permissions des utilisateurs.

L'interface réalisée doit être :

- simple d'accès,
- intuitive,
- formuler des requêtes saines sur l'API,
- réalisée avec une approche mobile-first.

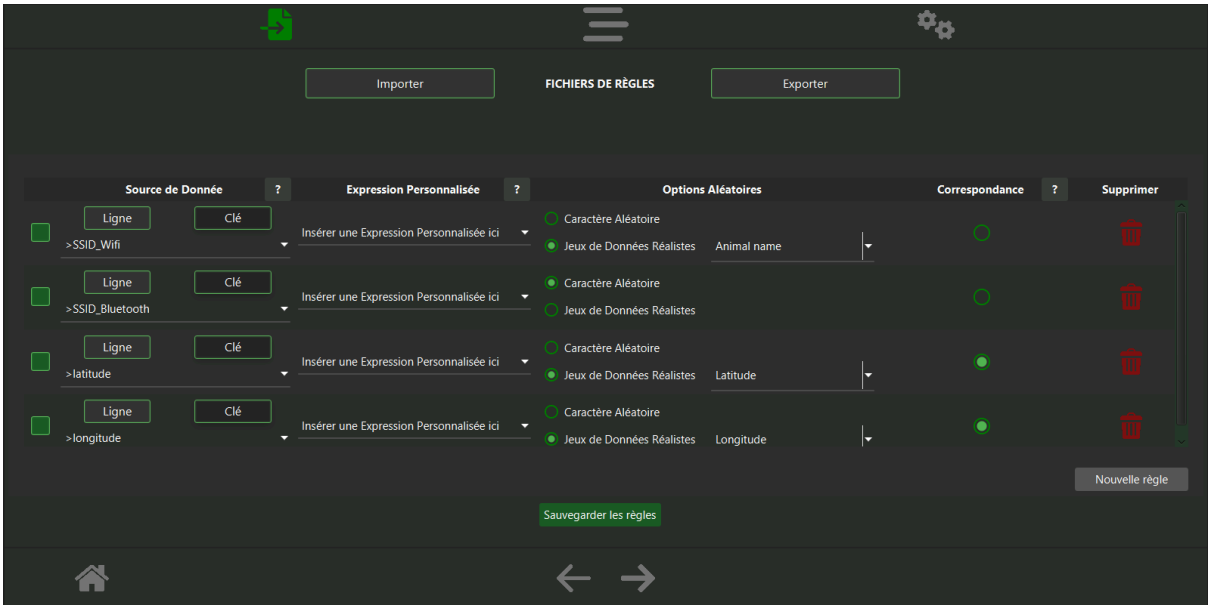
L'API Gemeny sera développé conjointement par Yohann Serpault, et représente la partie métier du produit.

Mes réalisations Front-end

Maquettage de l'application

Afin de réaliser les maquettes de l'interface web de Gemeny j'utilise Figma, et je décide de partir de l'application Gemeny existante, dans le but de :

- conserver un maximum de cohérence dans l'expérience utilisateur,
- respecter la charte graphique,
- proposer des améliorations.

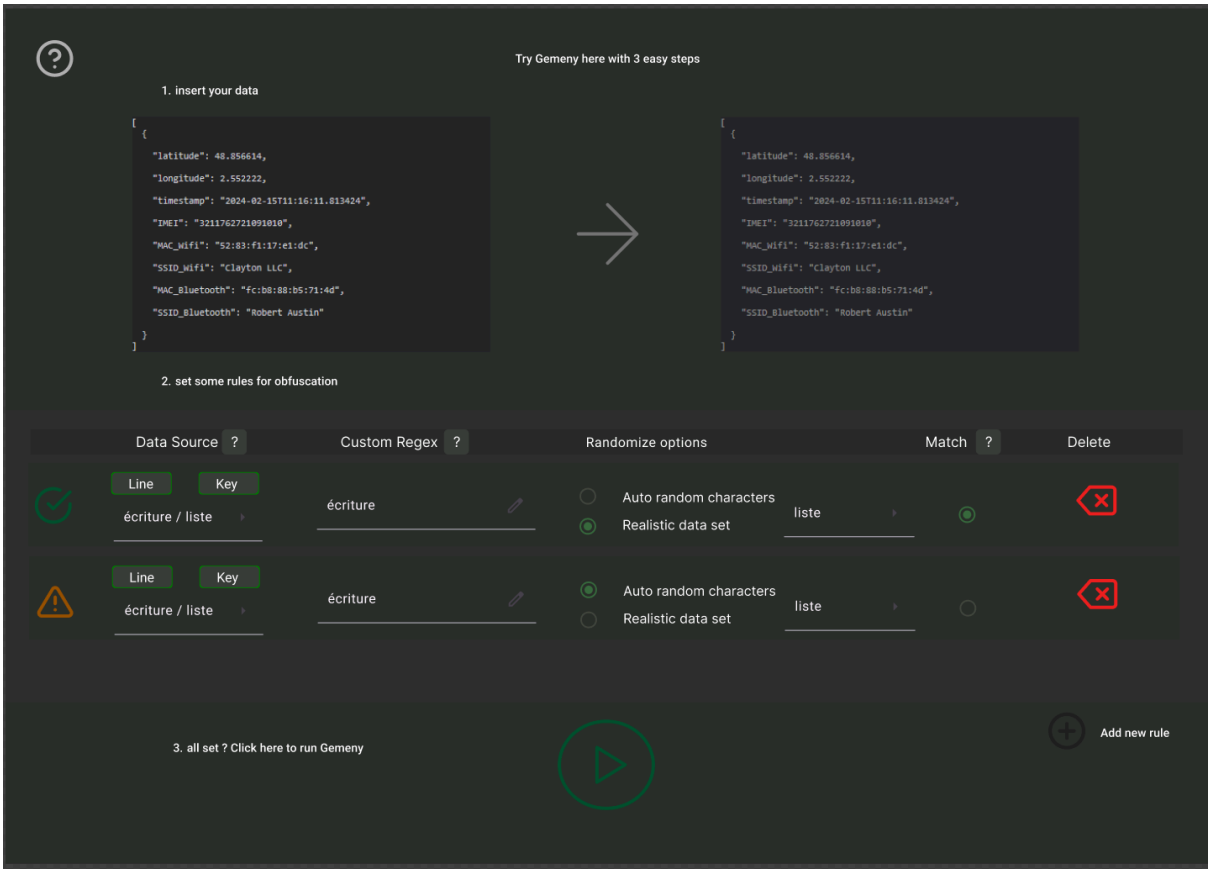


Screen de l'application lourde Gemeny sur desktop

L'application Gemeny est intuitive mais ne permet pas d'afficher les changements sur les données après traitement,
il est décidé de pouvoir les comparer facilement sur l'interface web.

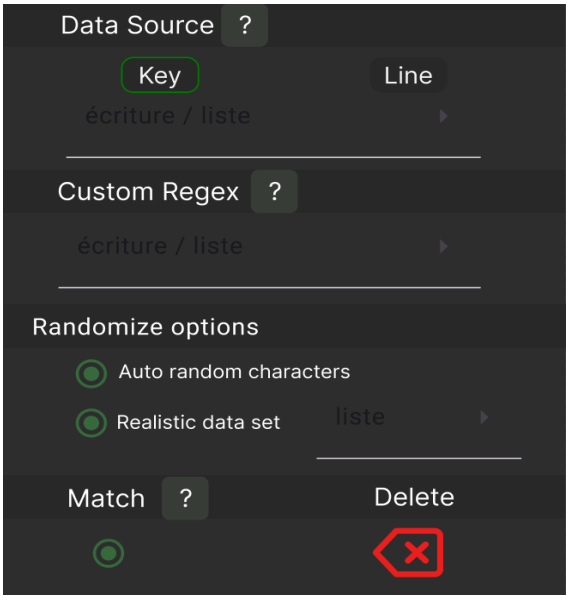
J'ai donc besoin d'afficher :

- ☑ les données entrantes
- ☑ une liste de règles applicables sur ces données
- ☑ le statut (valide ou non) de chaque règle
- ☑ le résultat = les données traitées par Gemeny API



Maquette desktop réalisé avec Figma

L'interface actuelle n'est pas pensée pour un affichage sur mobile, notamment à cause de l'édition des règles qui a été pensée pour tenir sur une seule ligne, rendant le visuel très large.
Je me rends rapidement compte qu'il faut revoir totalement la disposition des éléments d'une règle pour l'affichage mobile.



J'opte pour la possibilité d'afficher ou de cacher chaque règle dans la liste, afin de ne pas rendre le défilement vertical trop important.

Maquette d'une règle sur mobile

Mise en place de mon environnement de travail

Pour structurer les différentes actions sur le projet, assurer une traçabilité et un contrôle sur mes différentes versions, je mets en place un projet sur GitLab avec mon maître de stage, afin de découper en tâches (issues) les opérations à effectuer, par priorité, avec un nommage évocateur.

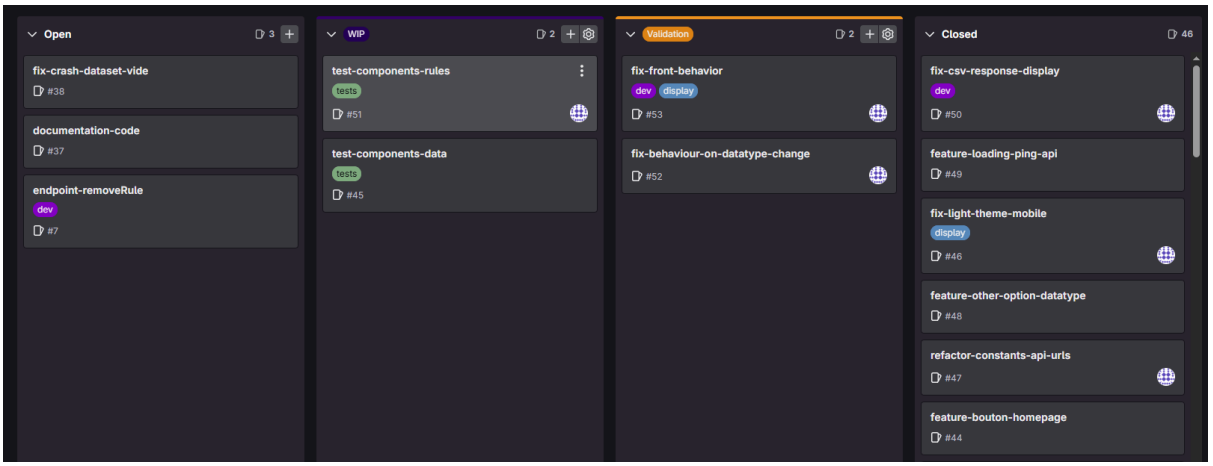
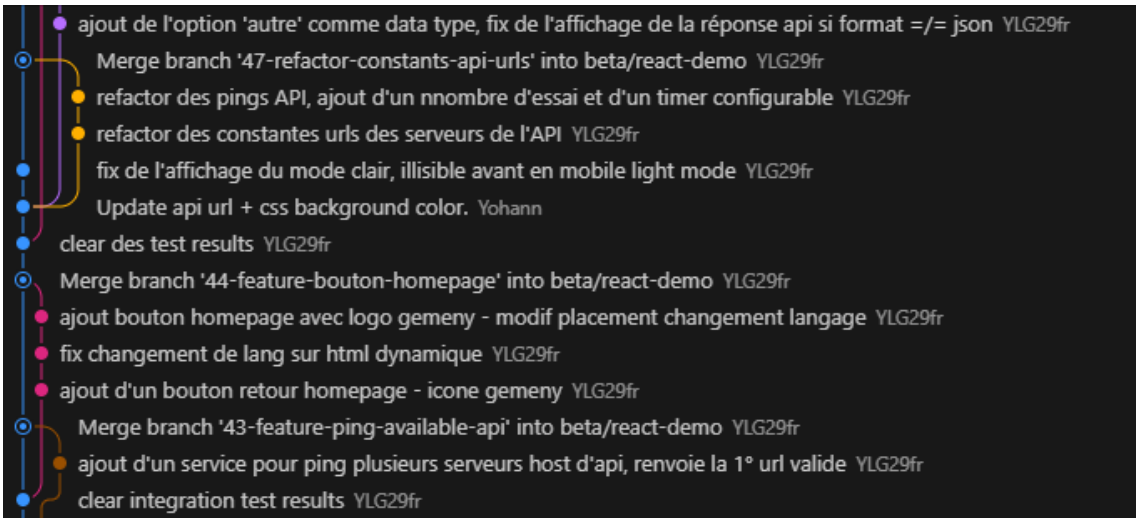


Tableau Kanban du projet front : backlog / in progress / review / closed

Chaque tâche sera associé à une nouvelle branche git, portant le même nom, qui seront ensuite merge sur la branche de dev : 'beta/react-demo' une fois la tâche effectuée.



GitGraph : exemples de branches de travail merge sur la branch dev et leurs commits

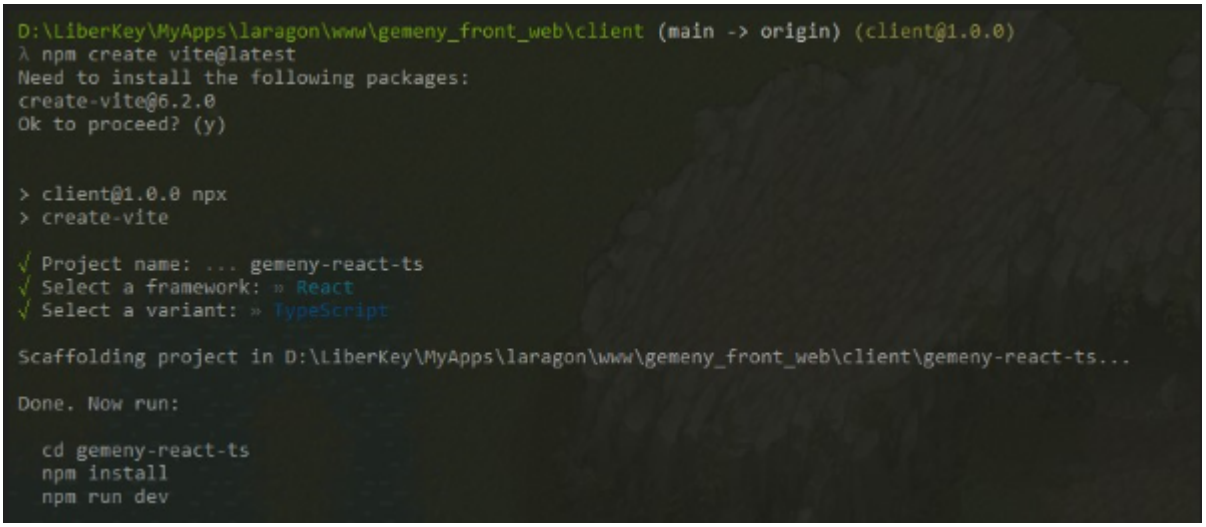
➔ **[Annexe p.2] GitFlow**

Pour la réalisation de l'interface web, avec l'accord du maître de stage, j'opte pour l'utilisation de React en TypeScript, en utilisant le builder Vite.

➔ [Annexe p.3] Exemple de mon composant 'Bouton' avec React

React est un framework puissant pour l'affichage front end dans un contexte web, basé sur la notion de composant, avec un typage (interface) si on utilise TypeScript.

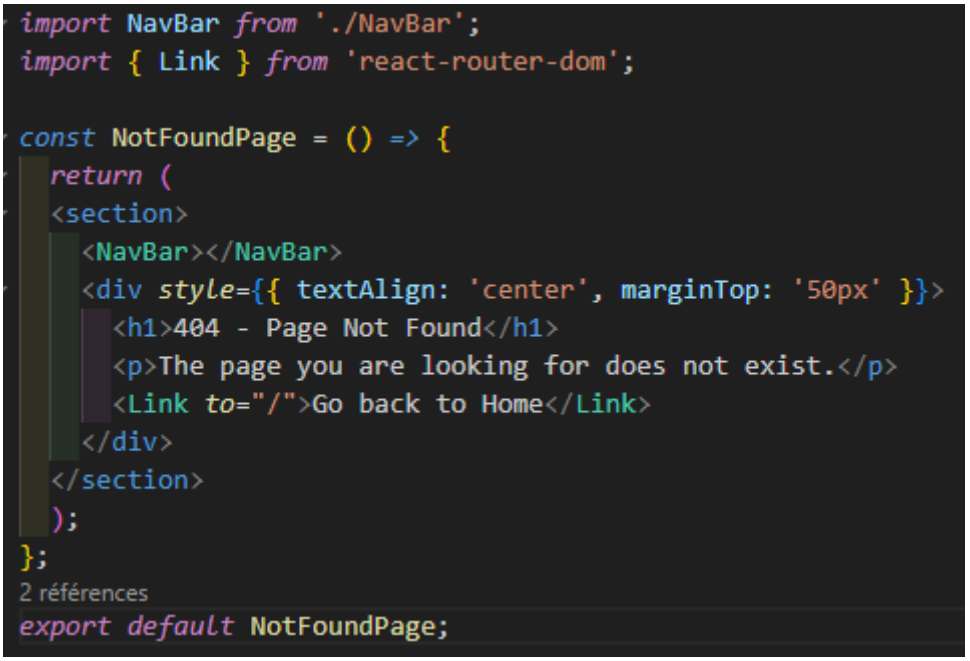
- React (v.19) : afin de pouvoir créer des composants réutilisables et maintenables en HTML et CSS
- TypeScript(v.5.7) : me permet d'ajouter un système de typage afin de rendre plus fiable mon code dès son écriture (je m'assure de passer des valeurs attendues et/ou correctes)
- Vite(v.6.1) est le builder que j'utilise pour mettre en place une structure basique pour le projet
- NPM pour gérer mes dépendances, les installer et les mettre à jour sur le projet, lancer les scripts d'exécution, de production, de tests…
- Axios (v.1.8+) est une librairie JavaScript permettant de gérer les requêtes HTTP de manière simple, efficace et configurable



Initialisation du projet, mise en place de la structure basique

➔ [Annexe p.4] Scripts et environnement de développement

Dans mon projet, un composant React peut prendre des paramètres en entrée, appliquer de la logique puis renvoie (return) un élément type HTML.



```jsx
import NavBar from './NavBar';
import { Link } from 'react-router-dom';

const NotFoundPage = () => {
  return (
  <section>
    <NavBar></NavBar>
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h1>404 - Page Not Found</h1>
      <p>The page you are looking for does not exist.</p>
      <Link to="/">Go back to Home</Link>
    </div>
  </section>
  );
};
2 références
export default NotFoundPage;
```

Exemple pour la page 404 Not Found, qui utilise le composant NavBar pour la navigation

J'utilise VSCode comme IDE pour le projet front Gemeny Web, pour bénéficier de fonctionnalités comme un explorateur intégré, pour parcourir le projet dans son ensemble.

Structure du projet

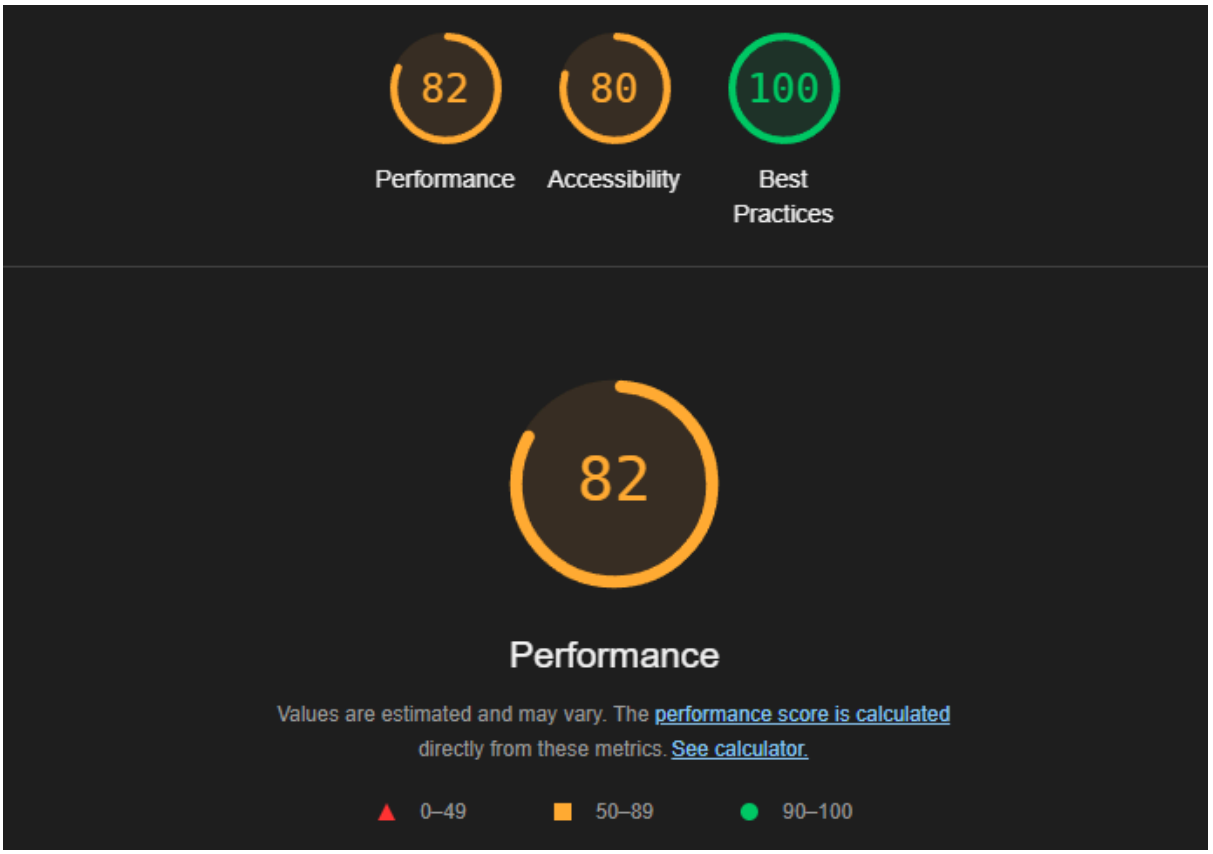
Je complète la structure de mon projet en y ajoutant les composants et en assurant une répartition des fonctions, des différents composants React qui correspondent à des éléments HTML dans lesquels je peux appliquer de la logique. Je distingue un dossier **'locales/'** qui me servira à renseigner tous les éléments de type texte, par langue (anglais et français pour le moment).

➔ **[Annexe p.5] Structure projet Gemeny_Web**

Utilisation d'outils d'audit

L'application web a bénéficié de plusieurs ajustements visuels au cours de sa réalisation, avec le concours d'utilisateurs test pour l'accessibilité, le rendu visuel, et rendre son utilisation le plus intuitive possible.

J'utilise l'outil LightHouse présent sur le navigateur Chrome pour effectuer des audits réguliers et améliorer les points les plus pertinents de l'application en termes de bonne pratique, performances et accessibilité.



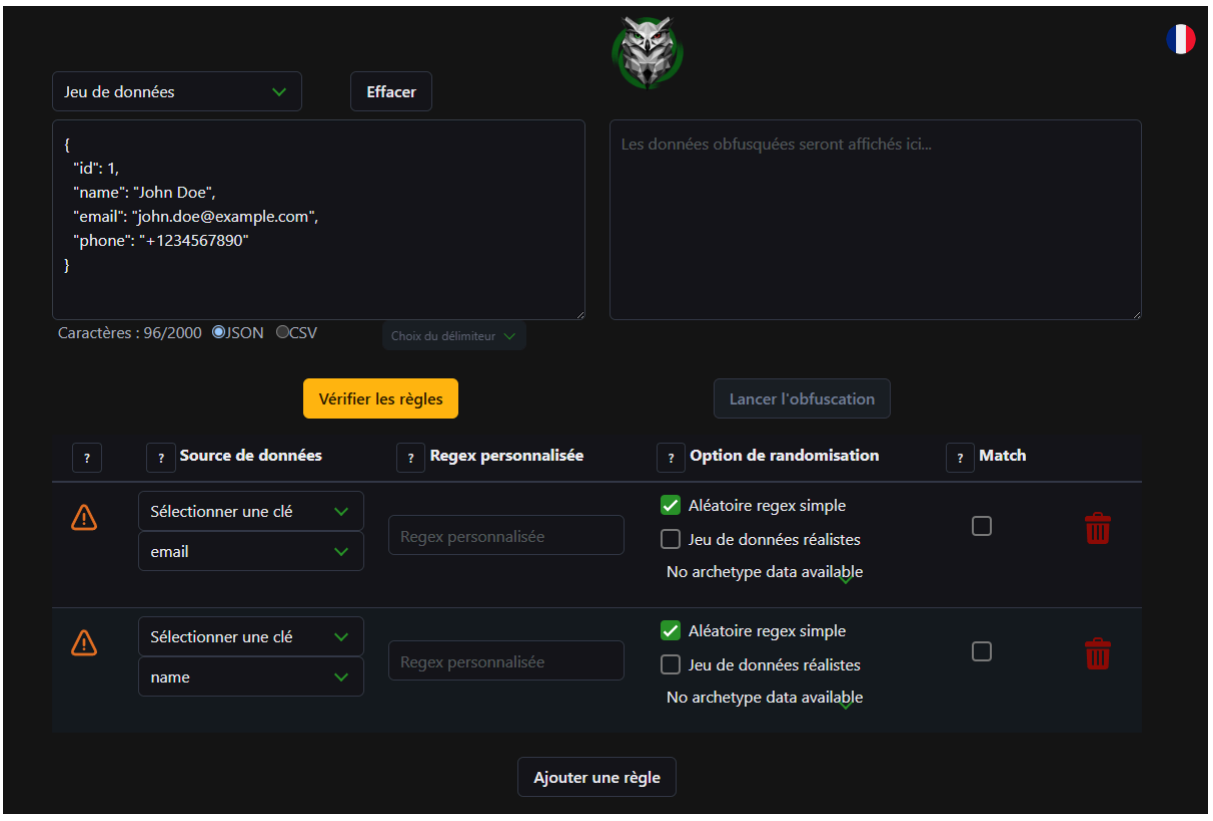
Rapport LightHouse (navigateur Chrome)

Réalisation de l'interface statique

La priorité du projet étant la création de l'interface communicante avec l'API Gemeny, je concentre mes efforts sur sa réalisation.

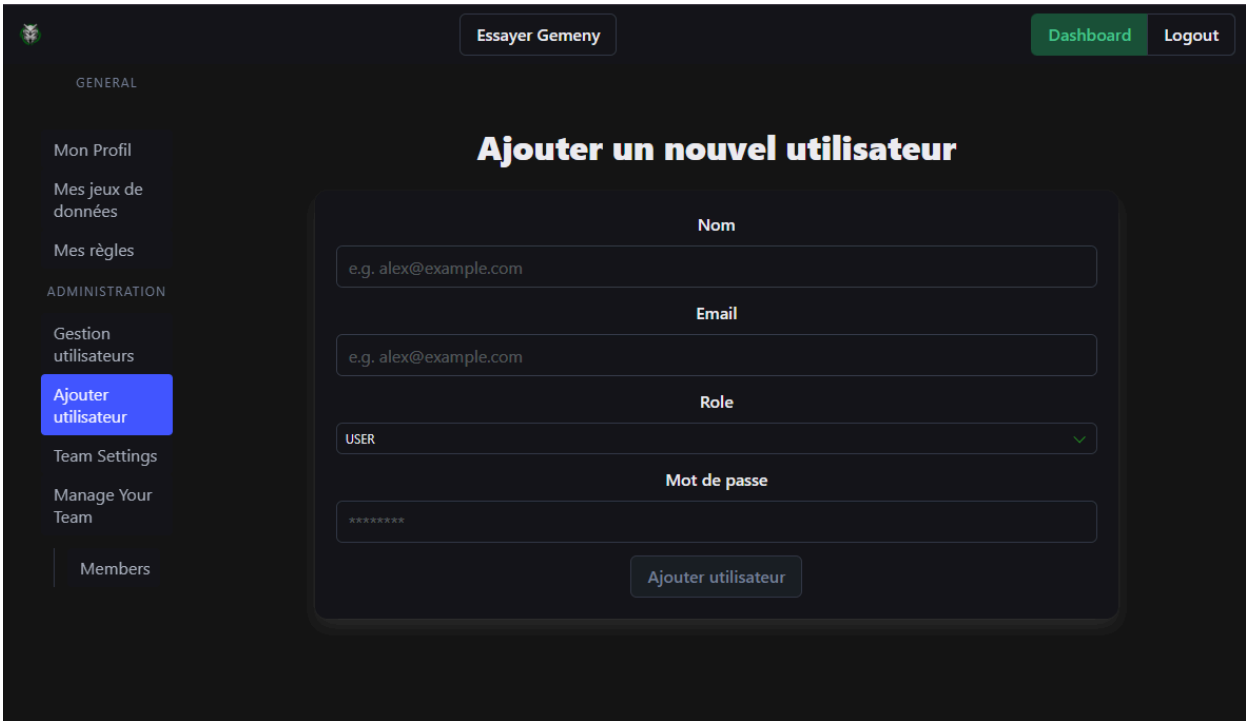
J'utilise les balises HTML sémantiques pour créer mes éléments, sur lesquelles j'applique des règles de style CSS pour la mise en forme, ainsi que des attributs spécifiques pour chaque élément, comme aria-label.

L'attribut aria-label est un outil puissant pour améliorer l'accessibilité des applications web en fournissant des descriptions textuelles aux éléments interactifs qui manquent de texte visible et facilitent l'utilisation de liseuse.



Application Gemeny Web - interface d'essai en ligne

Après la présentation de Gemeny au salon du SOFINS, je réalise un dashboard en vue d'une gestion d'utilisateurs, réservé aux administrateurs.



Ajout d'un nouvel utilisateur depuis le dashboard admin

```
<div className='column dashboard-content'>
<p className='title'>Ajouter un nouvel utilisateur</p>
<form className="box" onSubmit={handleAddNewUserClick} aria-label="Formulaire d'ajout d'utilisateur">
  <div className="field">
    <label className="label" htmlFor="nom-input">Nom</label>
    <div className="control">
      <input className="input" id="nom-input" type="text" placeholder="e.g. alex@example.com" required
        aria-label="Nom" value={identifier}
        onChange={(e) => setIdentifier(e.target.value)}
      />
    </div>
  </div>
  <div className="field">
    <label className="label" htmlFor="email-input">Email</label>
    <div className="control">
      <input className="input" id="email-input" type="email" placeholder="e.g. alex@example.com" required
        aria-label="Adresse email" value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
    </div>
  </div>
```

Extrait de code HTML du formulaire de création utilisateur

Réalisation de l'interface dynamique

L'interface de l'application Gemeny Web est essentiellement dynamique, que ce soit pour :

- la gestion du texte visible, avec une gestion multilingue,
- des actions réalisés par l'utilisateur pour saisir des données en entrées (vérifiés à chaque changement pour être utilisés comme référence pour les règles d'obfuscation),
- règles d'obfuscation sur lesquelles l'utilisateur peut interagir pour en ajouter, supprimer et surtout modifier ses paramètres.

Une règle est définit comme suit :

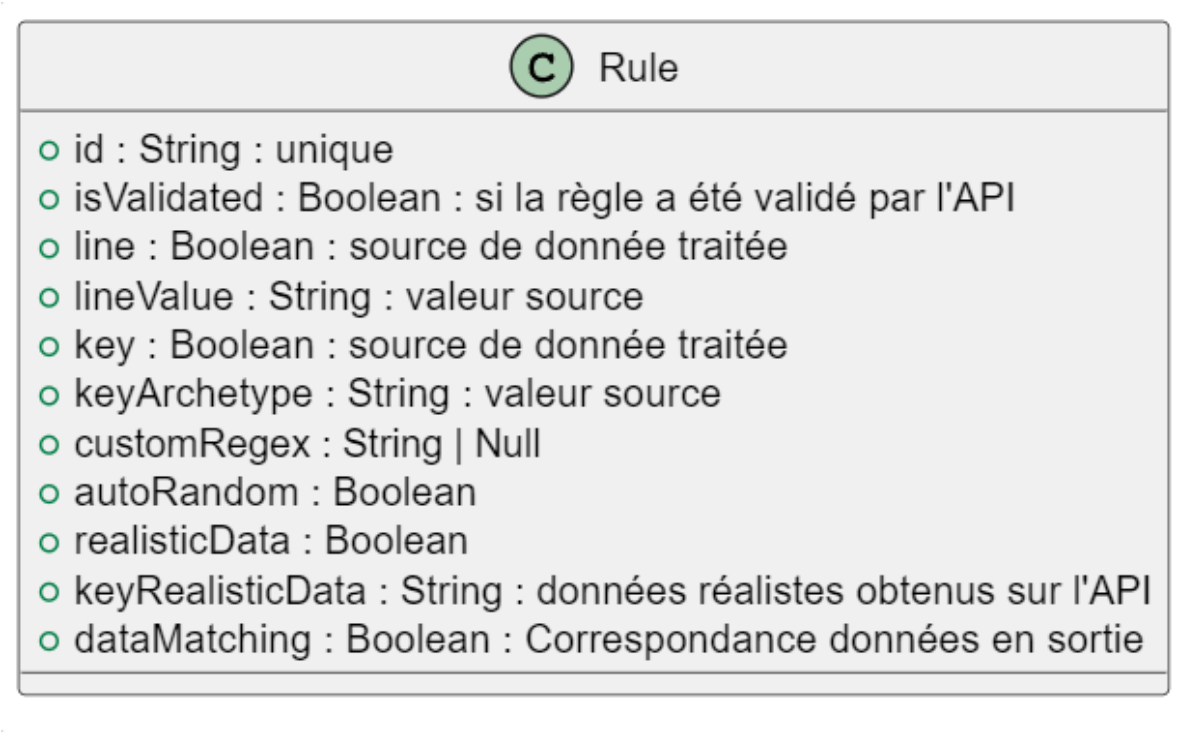


Diagramme UML représentant une règle

De nombreuses contraintes s'appliquent sur chaque règle comme par exemple :

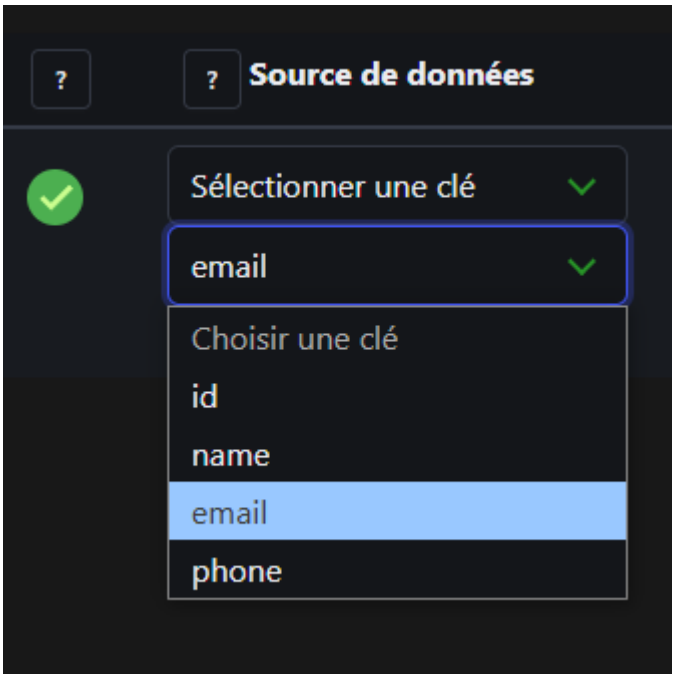
- *line* et *key* ne peuvent être *true* ensemble, si *false* leur valeur associée est un string vide "".
- si *isValidated* est false, la règle n'est pas utilisée dans la requête d'obfuscation (message "seules les règles valides sont appliquées").
- et d'autres, assurant des requêtes saines à l'API et un retour immédiat pour l'utilisateur en cas de mauvaise saisie.

Pour modifier chaque paramètres de chaque règle, l'utilisateur a accès à des éléments d'interface dynamique adaptés, sous la forme de liste déroulantes, de boutons checkbox, de saisie de texte.

Le premier paramètre de chaque règle est le choix de la source de données, qui permet à l'utilisateur de choisir quelle donnée sera traitée par les autres paramètres de la règle.

➔ **[Annexe p.6] Composant SelectDataSource.tsx**

Par définition si l'utilisateur renseigne des données sous format JSON, le choix de la source de données pourra être une clé de ce JSON.



Sélection d'une clé pour choisir la valeur à obfusquer sur un jeu de données saisie par l'utilisateur.

Une fois tous les paramètres renseignés, l'utilisateur envoie une requête de vérification des règles à l'API, qui indique les règles valides ou invalides, ensuite affichés à l'utilisateur sur l'interface.

➔ **[Annexe p.7] Affichage dynamique validation de règles**

J'ai rencontré à ce moment ma première grande difficultée sur le projet :
Comment gérer des valeurs (paramètres de règles) dynamiquement :

1. avec plusieurs instances de règles, dans une liste qui donne un tableau sur l'affichage utilisateur,
2. sur plusieurs composants d'interface pour chacune de ces règles,
3. faire correspondre ces données entre l'affichage mobile et desktop puisque j'utilise des composants différents.

La première solution que j'ai pu mettre en place consistait à stocker ces informations dans des objets de type **Rule** et de faire "suivre" en paramètres de mes composants les valeurs nécessaires à son fonctionnement.

Mais cela me demande de passer beaucoup de valeurs en paramètres de chaque composant - Exemple : pour le composant de sélection de la source de données, je dois transmettre à minima :

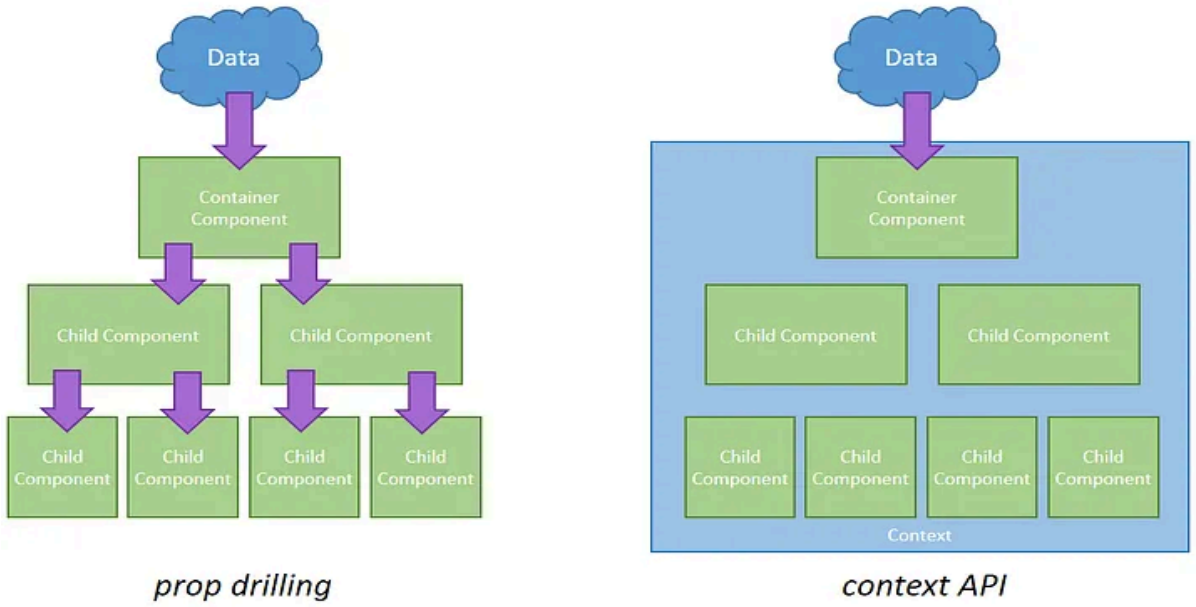
1. l'identifiant de la règle concerné
2. les clés identifiés sur la saisie de l'utilisateur (si JSON ou CSV)

→ puis intégrer un mécanisme pour renvoyer le résultat de la logique du composant en plus de l'élément HTML (objectif principal du composant).
Et ce pour chaque composant de mon application front.

Cette solution ne me satisfait pas, en ne permettant pas de centraliser la logique de modification d'une règle, en multipliant le risque d'erreur pour chaque élément.
(j'apprends plus tard que cette approche est décrite par le terme prop-drilling)

Je recherche alors une solution en m'appuyant sur la documentation officielle, des exemples de projets sur GitHub, et je trouve une solution sur la documentation de React : les contextes.

J'utilise les contexts de React pour gérer les valeurs sur lesquelles l'utilisateur interagit, ainsi je peux appliquer des vérifications avant d'utiliser ces dernières, et surtout je peux récupérer, modifier ces valeurs depuis plusieurs composants si besoin :



Utilisation de prop drilling < context

En suivant ce principe, je peux faire interagir un de mes composants pour accéder à, ou modifier une donnée, afin que celle-ci soit récupérée par un autre composant pour l'afficher par exemple.

➔ **[Annexe p.8] Schémas principe Context API**

J'ai mis ce principe à l'exécution notamment pour la gestion des règles d'obfuscation que l'utilisateur met en place avec un typage spécifique pour être conforme avec l'API Gemeny.

Lien avec les APIs Gemeny

Contexte :

Pour connecter l'interface web Gemeny à ses APIs (métier et utilisateur), j'ai utilisé **Axios**, une librairie JavaScript permettant de gérer les requêtes HTTP de manière simple et efficace, grâce à la création d'instances configurables.

Mise en œuvre :

Deux instances Axios distinctes sont créées :

- `businessApi` pour les appels à l'API métier (obfuscation, validation de règles, récupération des archétypes de données réalistes).
- `authApi` pour la gestion des utilisateurs (inscription, connexion).

```
import axios, { AxiosError, AxiosInstance , InternalAxiosRequestConfig} from 'axios';
import { GEM_API_URL, AUTH_API_URL } from '../constants/apiURL';

// Configuration de base pour l'API métier
const businessApi: AxiosInstance = axios.create({
    baseURL: GEM_API_URL,
    headers: {
        'Content-Type': 'application/json',
    }
});

  // Configuration pour l'API utilisateur
const authApi: AxiosInstance = axios.create({
    baseURL: AUTH_API_URL || 'http://localhost:8080/api/users',
    headers: {
        'Content-Type': 'application/json',
    }
});
```

Avantages :

- Séparation claire des responsabilités (métier vs. authentification),
- Centralisation et personnalisation de la configuration (baseURL, headers, etc.),
- Possibilités d'ajouter des intercepteurs pour automatiser la gestion des tokens et des erreurs.

Utilisation :

Les appels aux APIs se font ensuite via des services dédiés, selon leur usage :

```
export const authAPI = {
    /**
     * Authentifie l'utilisateur et retourne les tokens JWT
     * @param credentials - Email et mot de passe
     * @returns Les tokens d'accès et de rafraîchissement
     */
    login: async (credentials: LoginRequest): Promise<AuthResponse> => {
        const response = await authApi.post<AuthResponse>('/auth/login', credentials);
        return response.data;
    },
```

Extrait d'un service Api, pour l'authentification : api.auth.service.ts authAPI

```
// Login function
const login = async (email: string, password: string) => {
  const { user_info, access_token } = await authAPI.login({
    login: email,
    password
  }).catch(error => {
    // Transformer les erreurs Axios en erreurs métier
    const message = axios.isAxiosError(error)
      ? error.response?.data?.message || error.message
      : 'Erreur technique';
    throw new Error(message); // Recréer une Error propre
  });
```

Extrait d'un composant utilisant le service authAPI, pour connexion utilisateur

Spécificité pour Gemeny :

Pour les besoins de l'entreprise j'ai mis en place une logique pour ping plusieurs serveurs hébergeant l'API métier, afin d'ignorer un serveur s'il n'est pas disponible (down-time, maintenance …).

Résumé :

L'intégration Axios structurée facilite la communication entre le front-end React et les différentes APIs de Gemeny, tout en garantissant sécurité, évolutivité et clarté dans le code.

➔ **[Annexe p.9 - 10] Extraits code lien APIs**

Tests et gestion d'erreurs

Durant le développement de l'application front, je teste les interactions entre les composants en local directement sur navigateur avec les outils de développement intégrés, puis je teste l'interaction avec l'API dès qu'une première version est déployée, afin de faire les tests des premières fonctionnalités rapidement, avec des accès privés qui me sont fournis.

J'utilise des logs sur la console pour identifier et localiser les erreurs, ainsi que des messages d'erreurs personnalisés.

```
export default {
    UNKNOWN_ERROR : "Erreur inconnue",
    NETWORK_ERROR : "Erreur de connexion",
    UNAUTHORIZED : "Accès non autorisé",
    NOT_FOUND : "Ressource non trouvée",
    SERVER_ERROR : "Erreur interne serveur",
    RULES_NOT_VALIDATED : "Règles non validées : ",
    RULES_ZERO_VALIDATED : "Aucune règle validée : ",
    RULES_EMPTY : "Ajoutez au moins une règle",

    INVALID_INPUT : "Saisie non valide",
    INVALID_INPUT_JSON : "Saisie JSON non valide",
    INVALID_INPUT_CSV : "Saisie CSV non valide, pensez à vérifiez le séparateur ",
    CSV_SEPARATOR_REQUIRED : "Séparateur CSV requis",

}       You, le mois dernier • Affichage des règles valides revue, si les règl…
```

Constantes de textes d'erreurs en français

Je mets en place des tests unitaires sur les composants dans un objectif futur de CI (intégration continue) avec la librairie Jest (v.29+).
Les tests produisent un fichier .xml que je traduis en fichier Json car je préfère travailler avec ce format, il convient bien aussi dans la logique CI de l'entreprise.

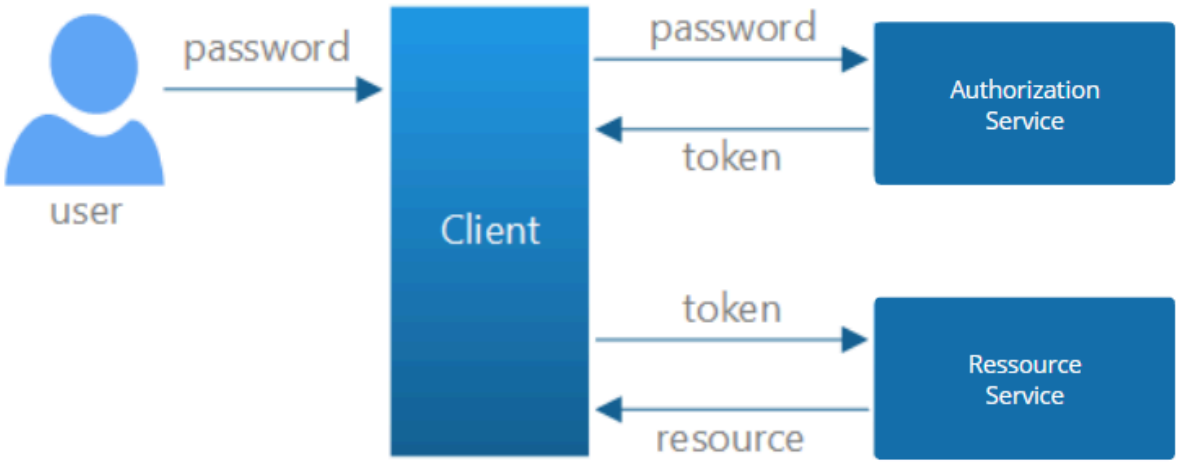
➔ **[Annexe p.11] Test unitaires sur composants front**

Mes réalisation Back-end

En vue d'un potentiel développement de Gemeny pour de la gestion d'utilisateur, je commence la conception puis la mise en place de fonctionnalités en ce sens via une application Back-End que je nomme Gemeny_User_API.

J'opte pour un micro-service API Rest Stateless :

- le serveur ne conserve aucune information de session entre les requêtes (chaque requête doit contenir toutes les informations nécessaires à son traitement),
- ce modèle facilite la scalabilité et la répartition de charge : n'importe quel serveur peut traiter n'importe quelle requête, sans notion de session utilisateur stockée côté back-end.



Séquence de principe API Stateless

➔ **[Annexe p.12] Comparaison Stateless vs Stateful**

Mise en place de mon environnement de travail

Pour la réalisation de l'application, je définis et configure mes outils :

- Laragon (v.6.0) comme serveur d'environnement de développement (je le considère comme un HUB pour mes autres outils)
- Git (v.2.4+), pour le contrôle de version - sauvegarde du code
- MariaDB (v.11) comme gestionnaire de base de données relationnelle
- Java (v.21) avec Maven comme builder, gestion de dépendances
- Spring Boot (v.3.4.3) comme framework Java pour les outils web
- Postman (v.11.27) pour effectuer les tests manuels sur l'application

J'utilise principalement IntelliJ IDEA (v.2024.3.5 - Community Edition), avec peu de plugins (esthétique surtout), l'IDE étant déjà très complet.

Pour réaliser mes diagrammes, j'utilise :

- VsCode avec l'extension PlantUML
- la documentation officiel disponible sur plantuml.com

J'utilise le logiciel Looping pour la conception de ma base de données avec la méthode Merise.

Conception de l'API

Je commence par donner du contexte aux utilisateurs qui utiliseraient une version future de Gemeny Web.

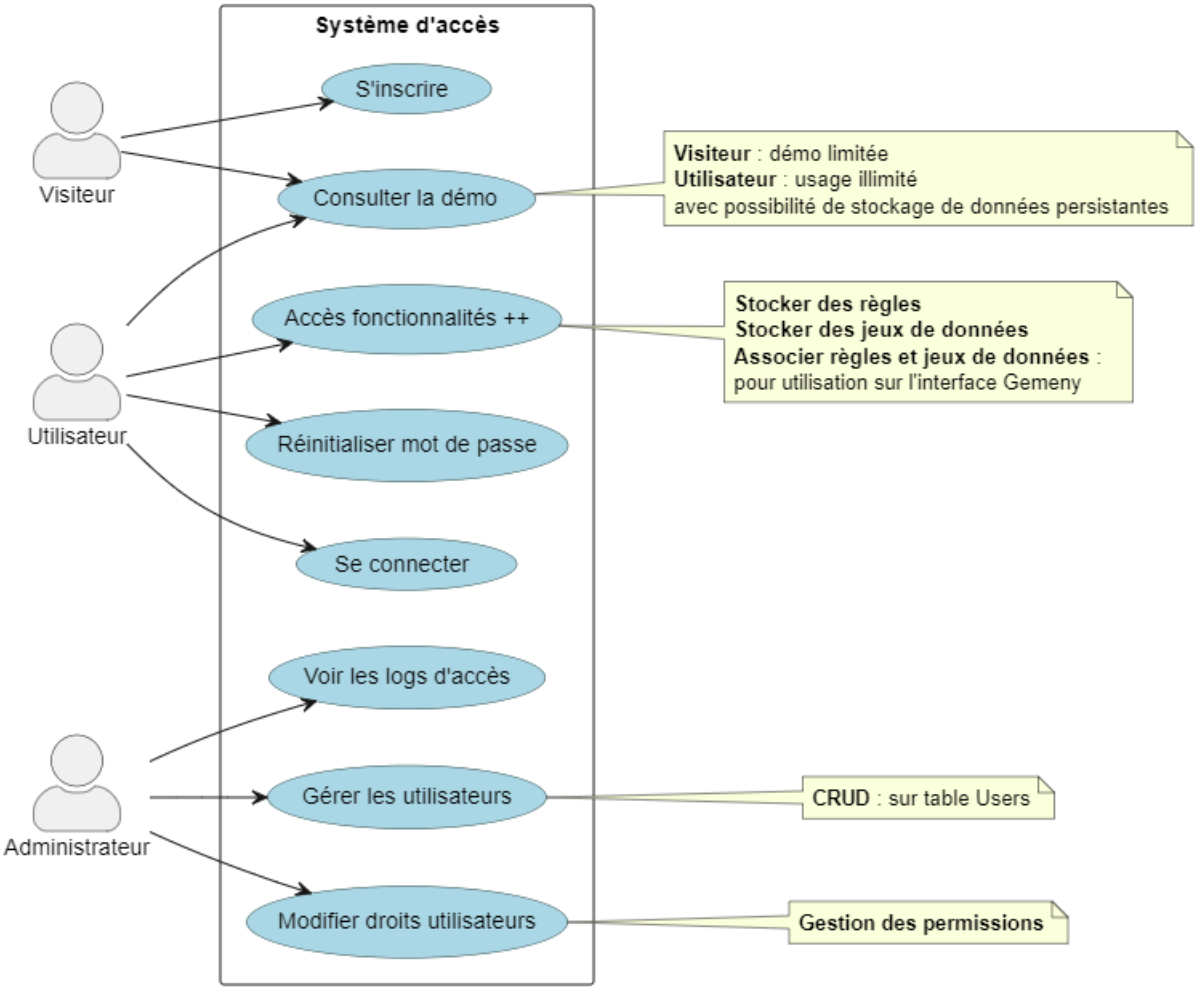


Diagramme useCase des utilisateurs Gemeny

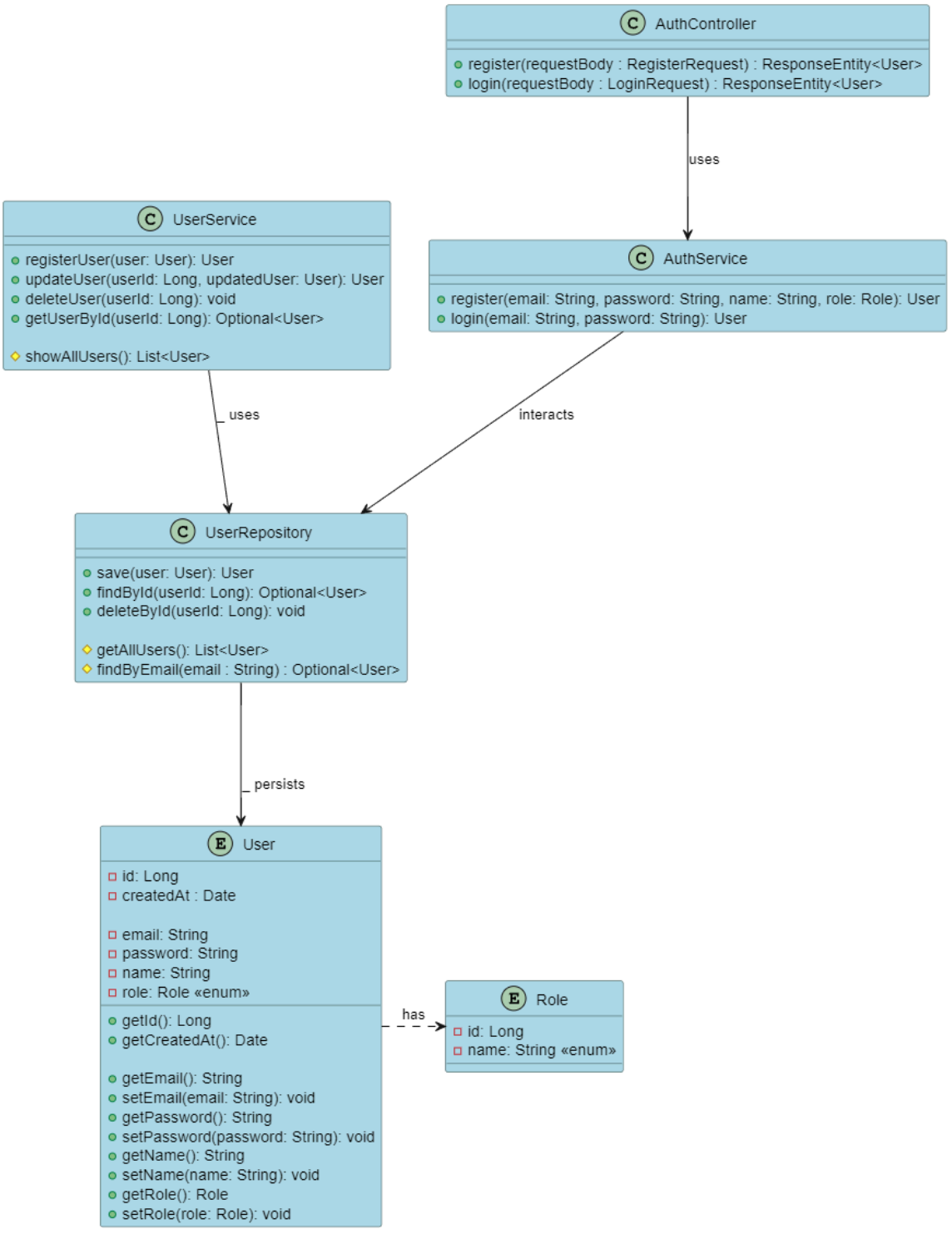
(Actuellement les utilisateurs de Gemeny Web le font depuis le site web de Gemeny, on peut les considérer comme Visiteur Consultant la démo)

Dictionnaire de données

Le dictionnaire de données servira de référence pour la création des classes, models pour harmoniser les données entre la base de données et l'application back-end.

➔ **[Annexe p.13 - 14] Dictionnaire de données**

Pour m'aider dans la conception (puis guider la réalisation) je réalise des diagrammes UMLs (useCase, sequence, classe).



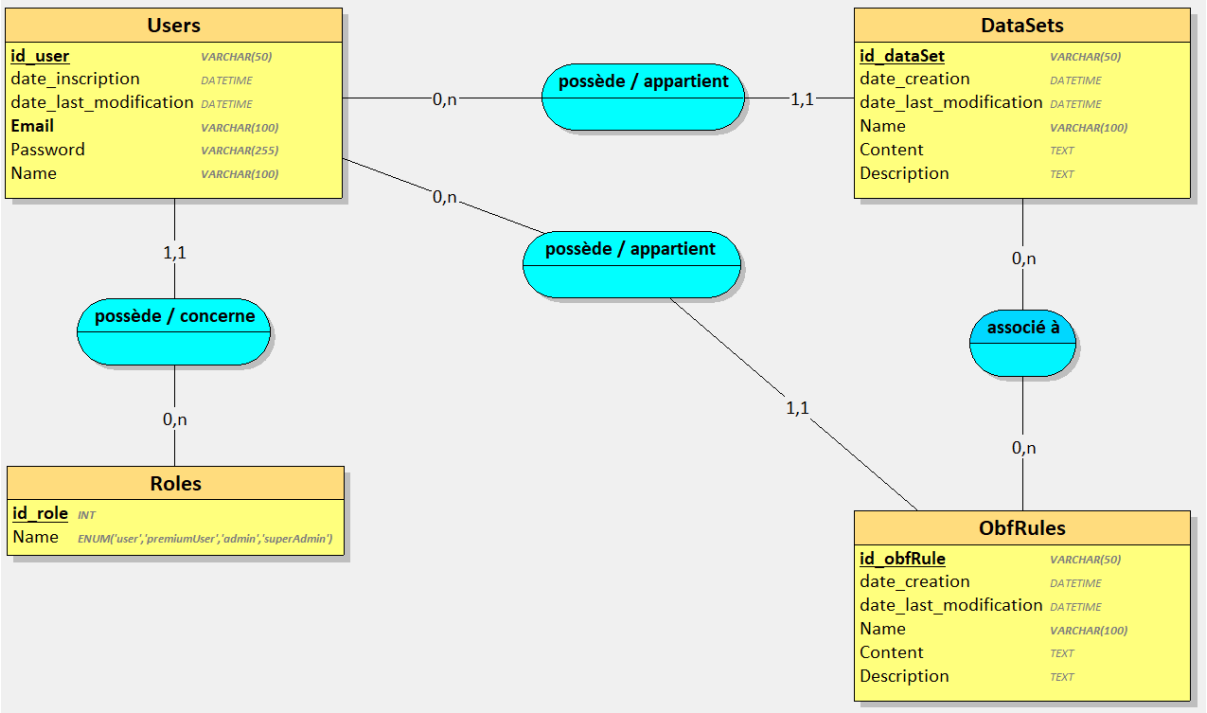
Extrait diagramme de classe pour l'authentification, les classes User et Role

➔ **[Annexe p.15] Diagrammes UML**

Je choisis d'utiliser une base de données relationnelles pour pouvoir croiser les données grâce aux relations entre tables.

L'outil Looping facilite la création de ce MCD en suivant les principes de **Merise**, ce qui permet de représenter de manière claire les dépendances fonctionnelles et les cardinalités entre les tables. Une fois le **MCD** validé, il est transformé en Modèle Logique des Données (**MLD**), qui prépare la structure pour l'implémentation dans un Système de Gestion de Base de Données(**SGBD**) relationnel.

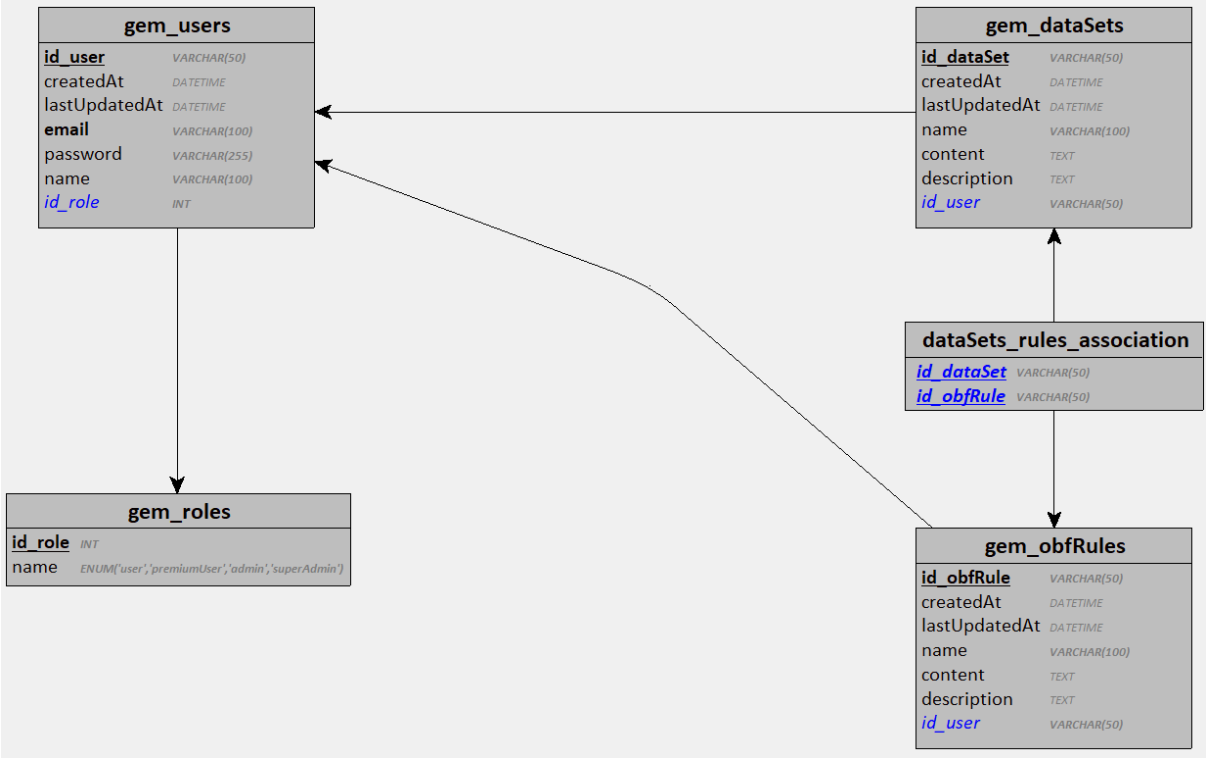
Ce processus garantit la cohérence, la robustesse et la capacité à croiser efficacement les données grâce aux relations définies entre les tables.



Conception avec la méthode Merise : MCD de la base de données, relation entre tables

La méthode Merise permet de mettre en évidence les relations entre mes tables.
J'identifie une relation one-to-many entre les tables gem_users et gem_roles :

- chaque utilisateur possède 1 seul rôle
- chaque rôle concerne 0 ou plusieurs utilisateurs
- une clé étrangère (foreign key) servira de lien entre ces 2 tables



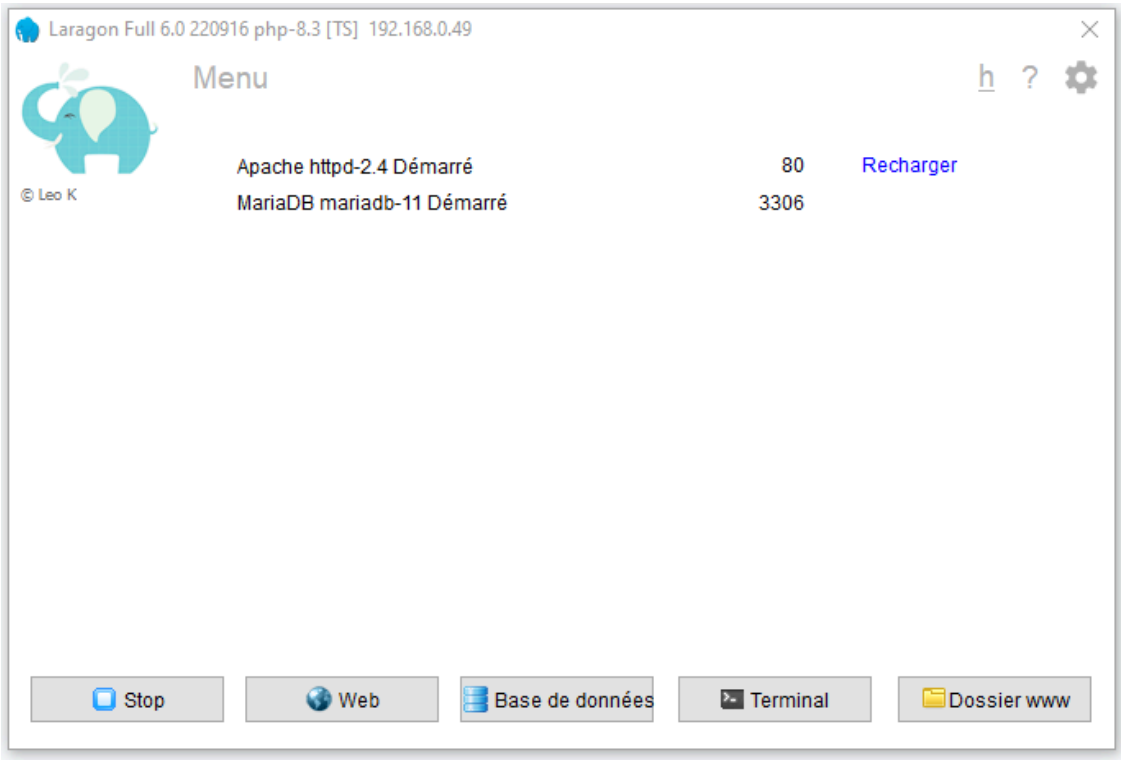
MLD : Mise en évidence des relations entre tables via clé étrangères et table d'association

MLD

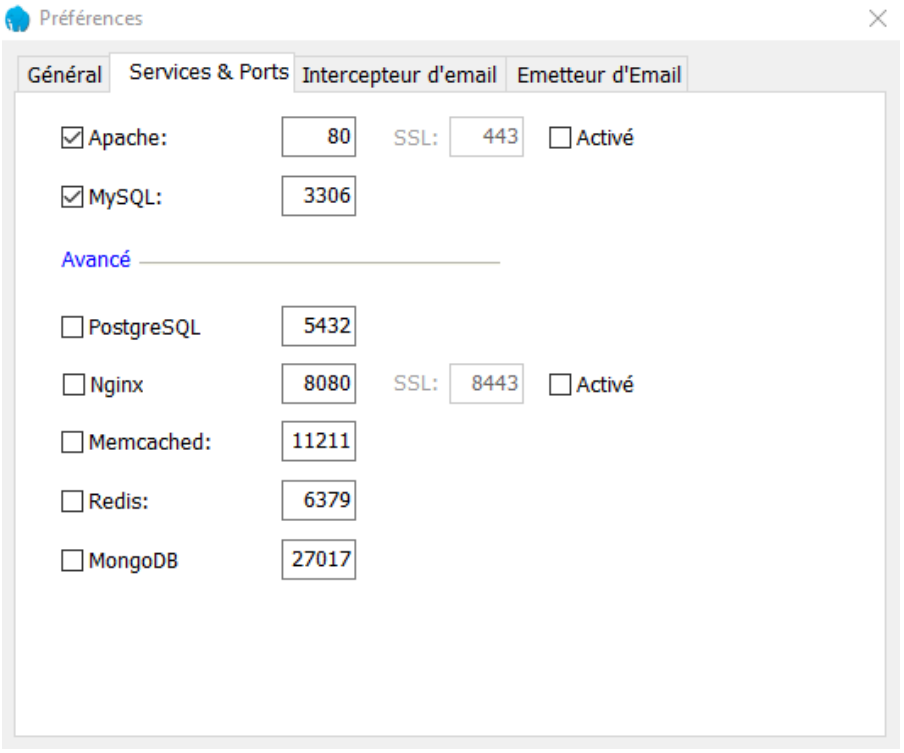
gem_roles = (**id_role** *INT*, name *ENUM('user','premiumUser','admin','superAdmin')*);
gem_users = (**id_user** *VARCHAR(50)*, createdAt *DATETIME*, lastUpdatedAt *DATETIME*, **email** *VARCHAR(100)*, password *VARCHAR(255)*, name *VARCHAR(100)*, *#id_role*);
gem_dataSets = (**id_dataSet** *VARCHAR(50)*, createdAt *DATETIME*, lastUpdatedAt *DATETIME*, name *VARCHAR(100)*, content *TEXT*, description *TEXT*, *#id_user*);
gem_obfRules = (**id_obfRule** *VARCHAR(50)*, createdAt *DATETIME*, lastUpdatedAt *DATETIME*, name *VARCHAR(100)*, content *TEXT*, description *TEXT*, *#id_user*);
dataSets_rules_association = (*#id_dataSet*, *#id_obfRule*);

Mise en place de la base de données

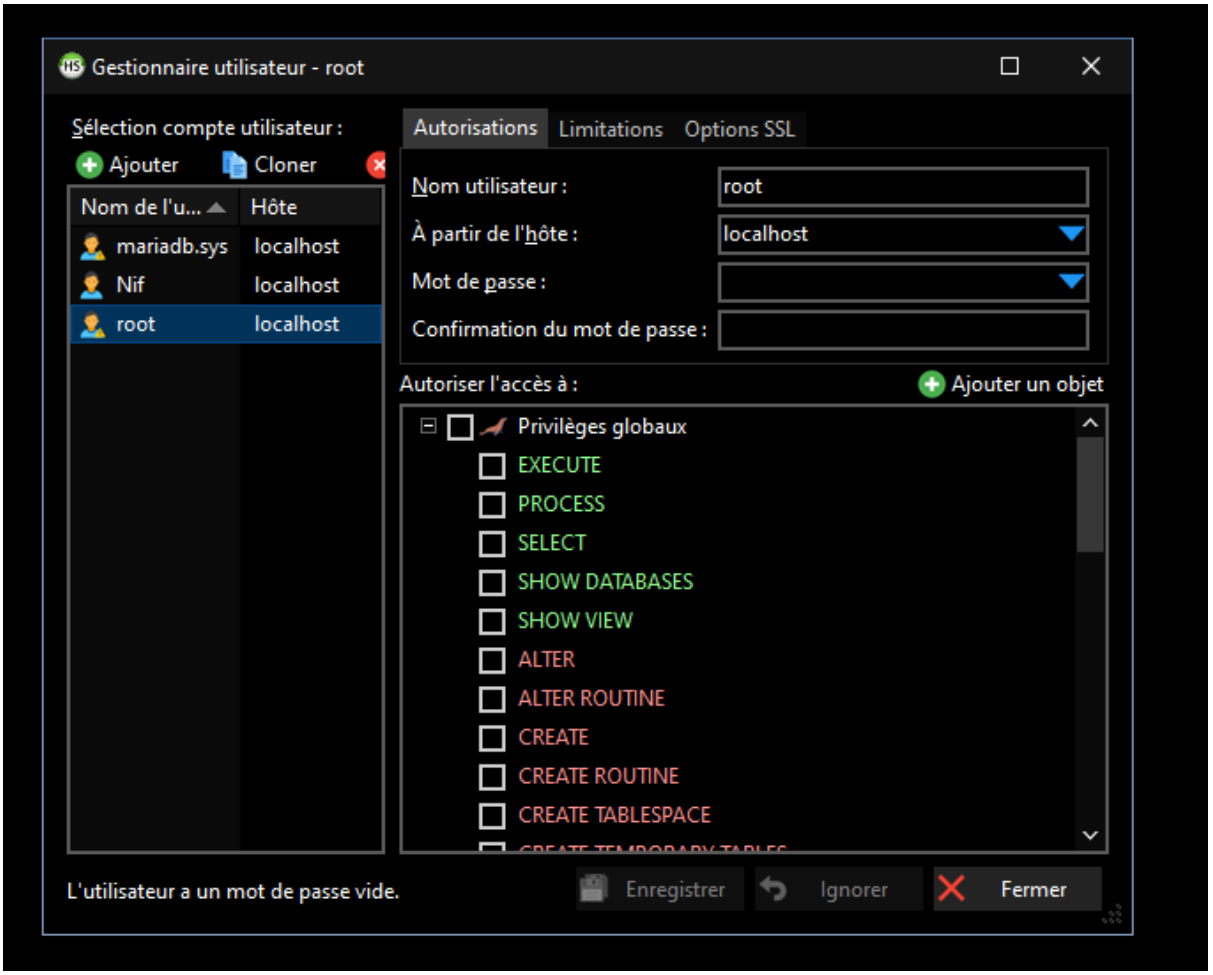
Pour initier ma base de données, je lance mon serveur Laragon, qui gère l'initialisation de MariaDB selon la configuration mise en place.



Interface d'utilisation et de configuration Laragon



Je gère les accès en fonction des utilisateurs avec l'interface HeidiSQL, ainsi je peux restreindre ou accorder des privilèges sur la base de données.



Interface gestion de MariaDB avec HeidiSQL

Une fois les configurations faites, je peux interagir avec la base de données directement sur l'interface, via des scripts SQL, ou des commandes sur un terminal de commande.

J'opte pour l'utilisation de scripts, afin de pouvoir stocker les instructions, les réutiliser plus tard (intégration sur le repository-DAO de mon API).

Je commence par créer une base de données SQL, et l'utilise :

```
CREATE DATABASE IF NOT EXISTS gemeny_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
USE gemeny_db;
```

Je crée ensuite les tables gem_users et gem_roles pour la gestion des utilisateurs et les rôles disponibles, en les associant avec une relation one-to-many entre elles :

```sql
CREATE TABLE gem_roles(
   id_role INT AUTO_INCREMENT,
   name ENUM('user','premiumUser','admin','superAdmin') NOT NULL DEFAULT 'user',
   PRIMARY KEY(id_role)
);

CREATE TABLE gem_users(
   id_user VARCHAR(50),
   createdAt DATETIME DEFAULT CURRENT_TIMESTAMP,
   lastUpdatedAt DATETIME ON UPDATE CURRENT_TIMESTAMP,
   email VARCHAR(100) NOT NULL,
   password VARCHAR(255) NOT NULL,
   name VARCHAR(100) NOT NULL,
   id_role INT NOT NULL,
   PRIMARY KEY(id_user),
   UNIQUE(email),
   FOREIGN KEY(id_role) REFERENCES gem_roles(id_role)
);
```

Enfin j'ajoute les rôles et des données utilisateurs sur ces tables :

```sql
INSERT INTO gem_roles (name)
    VALUES
    ('user'),
    ('premiumUser'),
    ('admin'),
    ('superAdmin')
    ;


INSERT INTO gem_users (id_user ,name, email, password, id_role)
    VALUES
    ("1",'Bob', 'mailBob@truc.much', 'passw0rdSol1d', '4'),
    ("2",'Rébecca', 'Armand@truc.much', 'Saint-Didier-des-Bois', '2'),
    ("3",'Aimée', 'Hebert@truc.much', 'Marigny-le-Châtel', '3'),
    ("4",'Marielle', 'Ribeiro@truc.much', 'Maillères', '1'),
    ("5",'Hilaire', 'Savary@truc.much', 'Conie-Molitard', '1')
    ;
```

Pour récupérer des information sur les utilisateurs ayant le rôle “Admin”, je fais une requête SQL avec jointure interne (**INNER JOIN**) entre gem_users et gem_role, avec une condition spécifique (**WHERE**) :

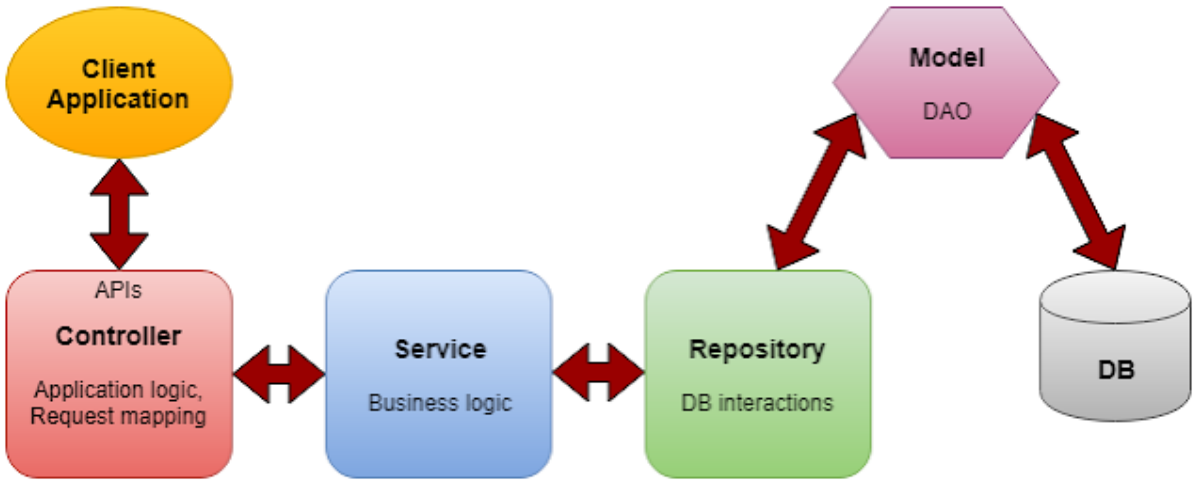
```sql
SELECT
    u.id_user,
    u.email,
    u.name,
    r.name AS role,
    u.createdAt,
FROM
    gem_users u
JOIN
    gem_roles r ON u.id_role = r.id_role
WHERE
    r.name = 'admin';
```

Requête d’accès SQL avec jointure de table
Utilisation d’alias u pour users, r pour roles

Je mets en place plusieurs requêtes préparées, l’interaction sur la base de données se fera uniquement via ces requêtes, pour limiter le risque d’injection SQL.

Réalisation de la structure MVC

Je mets en place une structure de type **MVC** (Model Vue Controller) pour mon projet d'API, en séparant les préoccupations, afin de rendre le code plus maintenable, plus modulaire.



Pattern design MVC qui a servi de référence sur le projet

1. **Model** : classe donnant la structure d'une entitée, exemple pour User
2. **Repository** : gère l'accès à la base de données
3. **Service** : définit la logique métier de l'application
4. **Controller** : orchestre les services, fait la passerelle entre les requêtes reçues en entrée de l'application via les endpoints, gère les permissions et les restrictions d'accès selon les rôles de l'utilisateur
5. **DTO** : Data Transfer Object, est un objet utilisé pour transférer des données entre différentes couches de l'application, il encapsule uniquement les données nécessaires pour un cas d'utilisation donné.

➔ **[Annexe p.16] Structure projet Gemeny_Auth_API**

Je confie la gestion de la Vue à une application front que je développe en parallèle pour la gestion des utilisateurs, en faisant une distinction forte entre la fonction du back et du front.

Mon API n'enverra en sortie que des données de type JSON, mon application front y accédera via les endpoints que j'ai définis, et suivant des restrictions d'accès selon les permissions de l'utilisateur.

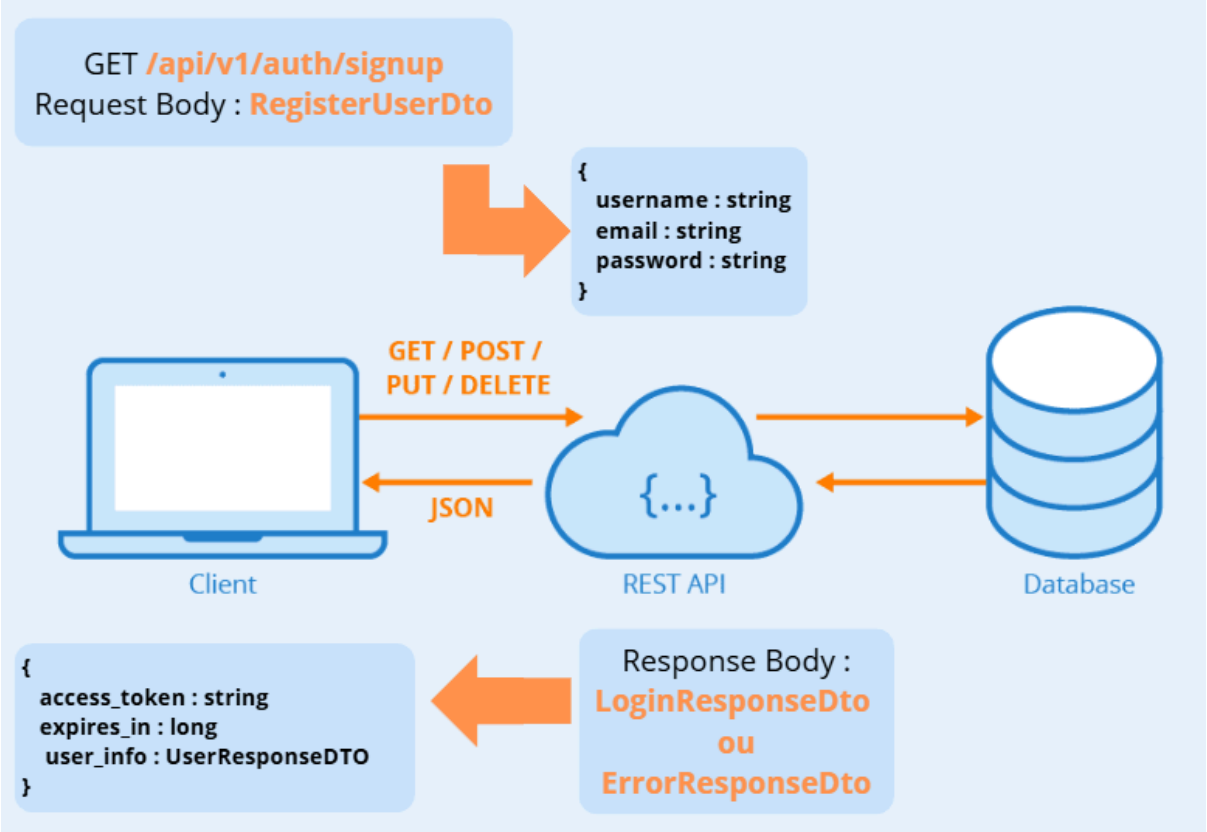


Schéma illustrant la communication client / API Rest

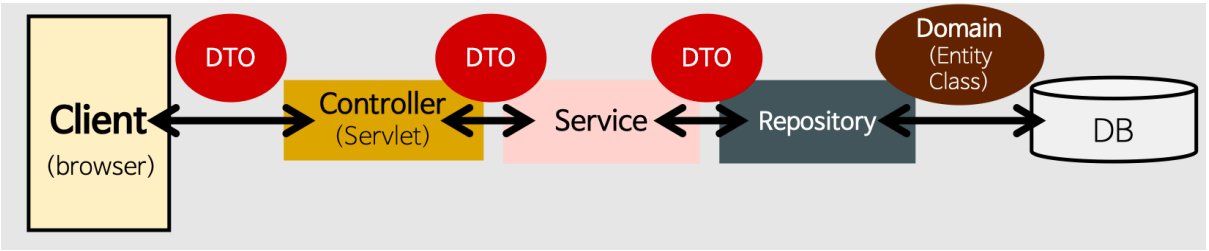
Exemple pour une inscription d'un utilisateur

Développement du cycle de vie d'une donnée

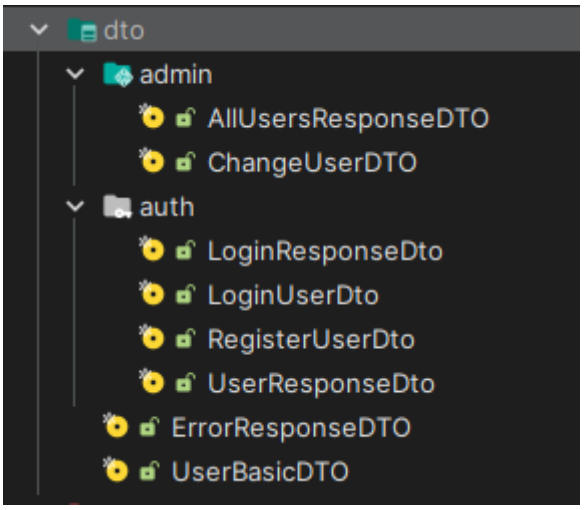
Je commence par la réalisation du flux des données de Users, avec la création de la classe **User**, qui servira de modèle pour la base de données. J'utilise l'ORM JPA Hibernate pour faciliter le lien avec la base de données avec les annotations @Table, @Column (**mappage**) .

➔ **[Annexe p.17] Extrait de code pour Model User**

Je mets en place des **DTO** pour gérer le transfert d'information entre les différents composants, pour m'assurer que l'accès à certaines données soit restreintes dès leur utilisation dans le back, et pour formatter les sorties de l'API avec des DTO spécifiques pour les réponses (utilisation sur le front ou Postman pour tester les requêtes sur l'API)



MVC avec utilisation de DTO entre les composants



Exemples de DTOs mis en place dans mon projet, nommés selon leur usage

➔ **[Annexe p.18] Exemples de DTOs**

Je crée le repository pour User, pour accéder à la base de données, notamment à la table User, pour écrire mes méthodes SQL pour établir les requêtes préparées (CRUD - Create Read Update Delete) : **UserRepository**

Je définis les méthodes applicables à **User** avec un dictionnaire de méthodes, puis les ajoute sur mon fichier de services pour les utilisateurs : **UserService**.

➔ **[Annexe p.19] Dictionnaire de méthodes**
➔ **[Annexe p.20] Extrait de code de service**

Je détermine ensuite un endpoint sur mon controller spécifique à l'authentification pour la création d'un utilisateur (inscription) :
AuthController

➔ **[Annexe p.21] Endpoints de l'API**
➔ **[Annexe p.22] Extraits de code de controller**

Pour l'inscription d'un utilisateur, ce dernier renseigne :

1. une adresse mail, un nom, et un mot de passe, via une requête sur l'endpoint " `/signup` "
2. le controller d'authentification (**AuthController**) capte la requête et orchestre les services à appliquer dessus (logique et sécurité)
3. ces informations sont transférés via le DTO (**RegisterUserDTO**)
4. la méthode createUser() de **UserService** effectue des vérifications (mail unique, id disponible, mot de passe conforme), renvoie une erreur ou valide l'inscription
5. le repository utilisateurs (**UserRepository**) reçoit l'instruction de persister le nouvel utilisateur sur la base de données, renvoie une erreur ou une confirmation
   (**UserResponseDTO** ou **ErrorResponseDTO**)

Gestion des routes protégées

La sécurité des accès à l'API Gemeny est assurée grâce à la gestion des routes protégées, basée sur les rôles des utilisateurs et l'authentification JWT.

Principe général

Toutes les routes sensibles (administration, gestion des utilisateurs, modification de données) sont protégées côté back-end via Spring Security.
L'accès à chaque route est conditionné par le rôle de l'utilisateur
(ex : USER, ADMIN, SUPER_ADMIN).
Les rôles sont attribués lors de la création de l'utilisateur et stockés en base de données.

Exemple d'implémentation (AdminController)
Dans le contrôleur AdminController, les annotations **@PreAuthorize** permettent de restreindre l'accès selon le rôle :

```java
@RestController  YLG29fr
@PreAuthorize("hasAnyRole('ADMIN','SUPER_ADMIN')")
@RequestMapping("/api/v1/admin")
public class AdminController {

    private final UserServiceImpl userServiceImpl;

    public AdminController(UserServiceImpl userServiceImpl) { this.userServiceImpl = userServiceImpl; }


    @GetMapping("/allUsers")  YLG29fr
    public ResponseEntity<List<UserBasicDTO>> allUsers() { return ResponseEntity.ok(userServiceImpl.allUsers()); }
```

@PreAuthorize("hasAnyRole('ADMIN','SUPER_ADMIN')") sur la classe : toutes les routes sont accessibles uniquement aux administrateurs.

Certaines routes critiques (création ou suppression d'admin/super admin) sont encore plus restreintes avec
@PreAuthorize("hasRole('SUPER_ADMIN')").

Sécurité

L'application Gemeny Web a été conçue avec une attention particulière portée à la sécurité, afin de protéger les données des utilisateurs et de limiter les risques liés aux vulnérabilités courantes du web.

Voici les principales menaces identifiées et les mesures mises en place ou prévues pour y répondre :

☑ Injection SQL/NoSQL

- **Risque** :

Un utilisateur malveillant pourrait tenter d'injecter du code dans les requêtes vers la base de données pour accéder, modifier ou supprimer des données.

- **Mesures** :

1. Utilisation de requêtes préparées et de l'ORM Hibernate pour éviter toute concaténation de chaînes dans les requêtes SQL.
2. Validation et assainissement systématique des entrées utilisateur côté front

☑ XSS (Cross-Site Scripting)

- **Risque** :

Un attaquant pourrait injecter du JavaScript malveillant dans l'interface, compromettant les données ou la session de l'utilisateur.

- **Mesures** :

1. Échappement systématique des données affichées dans le DOM.
2. Utilisation du framework React qui protège nativement contre la plupart des attaques XSS en échappant les saisies utilisateurs.
3. Validation côté serveur des champs susceptibles de contenir du HTML.

☑ CSRF (Cross-Site Request Forgery)

- **Risque** :

Un utilisateur authentifié pourrait être piégé pour exécuter une action à son insu.

- **Mesures** :

1. L'API étant stateless et utilisant des tokens JWT en Authorization header, le risque de CSRF est fortement limité.
2. Pour les actions sensibles, possibilité d'ajouter un double contrôle (confirmation par mot de passe, captcha, etc.).

☐ Brute-force (à implémenter)

- **Risque** :

Tentatives répétées d'accès par mot de passe pour deviner les identifiants.

- **Mesures** :

1. Limitation du nombre de tentatives de connexion (rate limiting).
2. Blocage temporaire du compte après plusieurs échecs.
3. Stockage des mots de passe uniquement sous forme hashée (déjà mis en place avec Spring Security).

☑ Exposition de données sensibles

- **Risque** :

Fuite d'informations confidentielles dans les logs, messages d'erreur ou réponses API.

- **Mesures** :

1. Suppression des messages d'erreur détaillés en production, affichage de messages génériques.
2. Journalisation sécurisée, sans jamais logger de mots de passe ou de données personnelles.
3. Utilisation de variables d'environnement pour les clés secrètes et l'accès à la base de données.

☑ Utilisation d’un iframe (transmis à mon maître de stage)

- **Risque :**

Clickjacking, exécution de scripts malveillants ou fuite de données via une intégration non contrôlée.

- **Mesures :**

1. Restriction de l’intégration via la directive CSP frame-ancestors ou l’en-tête HTTP X-Frame-Options (valeur SAMEORIGIN).
2. Activation de l’attribut sandbox sur les iframes, avec permissions limitées au strict nécessaire.
3. Filtrage strict des échanges via postMessage pour empêcher toute fuite ou interception de données.

Conclusion

Ce stage chez Gemeny Software a été une expérience particulièrement enrichissante, tant sur le plan technique que personnel. J'ai eu l'opportunité de participer activement à la conception et au développement d'un micro-service de plateforme web innovante, répondant à des enjeux réels de cybersécurité et de protection des données sensibles.

Ce projet m'a permis de mettre en pratique l'ensemble des compétences acquises durant ma formation, tout en découvrant les exigences et la rigueur du travail en méthode Agile.

L'accompagnement de mon maître de stage, son implication, ainsi que l'ambiance collaborative au sein de l'AFPA ont été des atouts majeurs dans la réussite de cette mission.
J'ai pu approfondir mes connaissances en React, TypeScript, gestion de projet avec GitLab, Java pour la création d'une API, SQL, et j'ai pris conscience de l'importance de l'ergonomie et de l'accessibilité dans la conception d'interfaces utilisateurs.

Au-delà des compétences techniques, ce stage m'a conforté dans mon choix de carrière et m'a donné envie de poursuivre dans le domaine du développement web, notamment dans le cadre d'une alternance pour le titre Concepteur Développeur d'Applications (CDA).

Je ressors de cette expérience motivé, avec l'envie de continuer à apprendre et à relever de nouveaux défis.