Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE

MINISTÈRE CHARGÉ DE L'EMPLOI

DOSSIER PROFESSIONNEL (DP)

Nom de naissance	‣	DUBOIS
Nom d'usage	‣	DUBOIS
Prénom	‣	Léa
Adresse	‣	22 Boulevard Jean Moulin, 13005, Marseille

Titre professionnel visé

Concepteur Développeur d'Applications

MODALITÉ D'ACCÈS :

☒ Parcours de formation
☐ Validation des Acquis de l'Expérience (VAE)

Présentation du dossier

Le dossier professionnel (DP) constitue un élément du système de validation du titre professionnel. **Ce titre est délivré par le Ministère chargé de l'emploi.**

Le DP appartient au candidat. Il le conserve, l'actualise durant son parcours et le présente **obligatoirement à chaque session d'examen**.

Pour rédiger le DP, le candidat peut être aidé par un formateur ou par un accompagnateur VAE.

Il est consulté par le jury au moment de la session d'examen.

Pour prendre sa décision, le jury dispose :

1. des résultats de la mise en situation professionnelle complétés, éventuellement, du questionnaire professionnel ou de l'entretien professionnel ou de l'entretien technique ou du questionnement à partir de productions.

2. du **Dossier Professionnel** (DP) dans lequel le candidat a consigné les preuves de sa pratique professionnelle.

3. des résultats des évaluations passées en cours de formation lorsque le candidat évalué est issu d'un parcours de formation

4. de l'entretien final (dans le cadre de la session titre).

[Arrêté du 22 décembre 2015, relatif aux conditions de délivrance des titres professionnels du ministère chargé de l'Emploi]

Ce dossier comporte :

- pour chaque activité-type du titre visé, un à trois exemples de pratique professionnelle ;
- un tableau à renseigner si le candidat souhaite porter à la connaissance du jury la détention d'un titre, d'un diplôme, d'un certificat de qualification professionnelle (CQP) ou des attestations de formation ;
- une déclaration sur l'honneur à compléter et à signer ;
- des documents illustrant la pratique professionnelle du candidat (facultatif)
- des annexes, si nécessaire.

DOSSIER PROFESSIONNEL (DP)

Pour compléter ce dossier, le candidat dispose d'un site web en accès libre sur le site.

http://travail-emploi.gouv.fr/titres-professionnels

Sommaire

Exemples de pratique professionnelle

EXEMPLES DE PRATIQUE PROFESSIONNELLE

Activité-type 1 Développer une application sécurisée

Exemple n°1 ▸ KMS

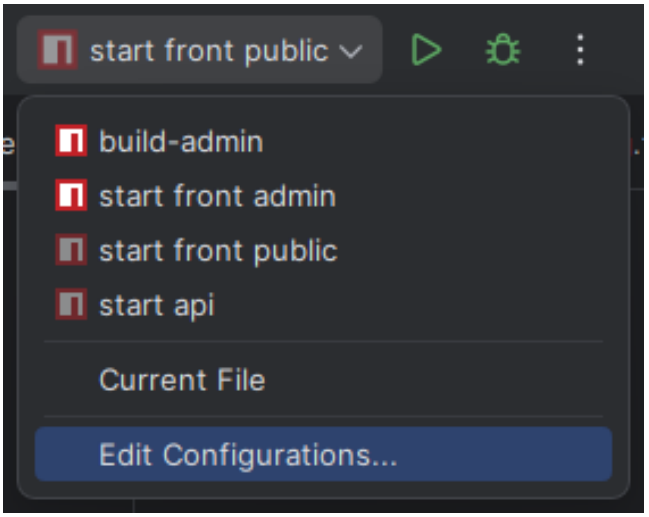
1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

'KMS' est un projet pour refaire entièrement le site web du club de krav maga 'Krav Maga Spirit' ainsi que leur panel admin.

✔ **Installer et configurer son environnement de travail en fonction du projet.**

Les prérequis pour pouvoir développer cette application sont les suivants :

J'ai aussi installé IntelliJ IDEA pour développer en Angular et en Node.js, que j'ai configuré pour pouvoir lancer rapidement les différentes parties de l'application :

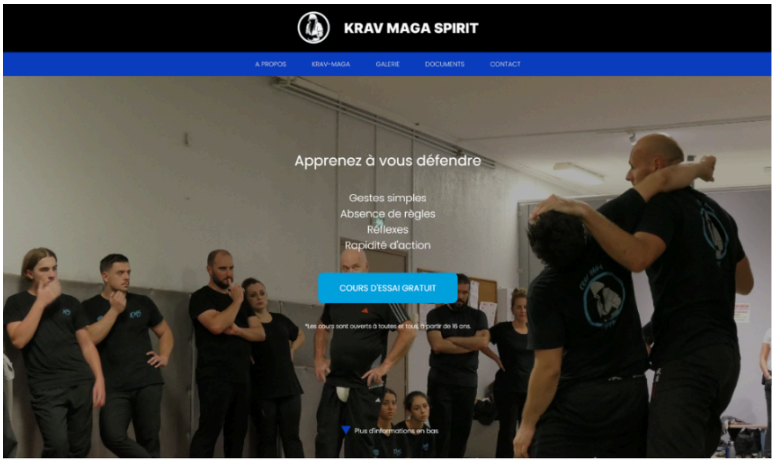


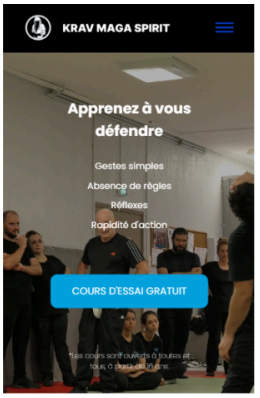
✔ **Développer des interfaces utilisateur**

L'interface utilisateur a été développée avec Angular avec TypeScript. Elle est donc développée en composants. Par exemple, on importe les composants HeaderComponent, FooterComponent, CardTarifComponent et Carrousel, qui seront affichés dans la page.

```typescript
import {
  Component,
  ViewChild,
  OnInit,
  ComponentRef, ViewContainerRef, Injector
} from '@angular/core';

import {HeaderComponent} from "../../components/header/header.component";
import {FooterComponent} from "../../components/footer/footer.component";
import {CardTarifComponent} from "../../components/card-tarif/card-tarif.component";
import {Carrousel} from "../../components/carrousel/carrousel.component";

@Component({ Show usages  fabien-ricca
  selector: 'home',
  standalone: true,
  imports: [
    HeaderComponent,
    FooterComponent,
    CardTarifComponent,
    Carrousel
  ],
  templateUrl: './home.component.html',
  styleUrl: './home.component.css'
})
```

Voici à quoi la page home la page Home sur un ordinateur et un portable, comparée à la maquette prévue :





Horaires

LUNDI	VENDREDI
19h30 - 21h Tous niveaux	20h - 21h30 Tous niveaux

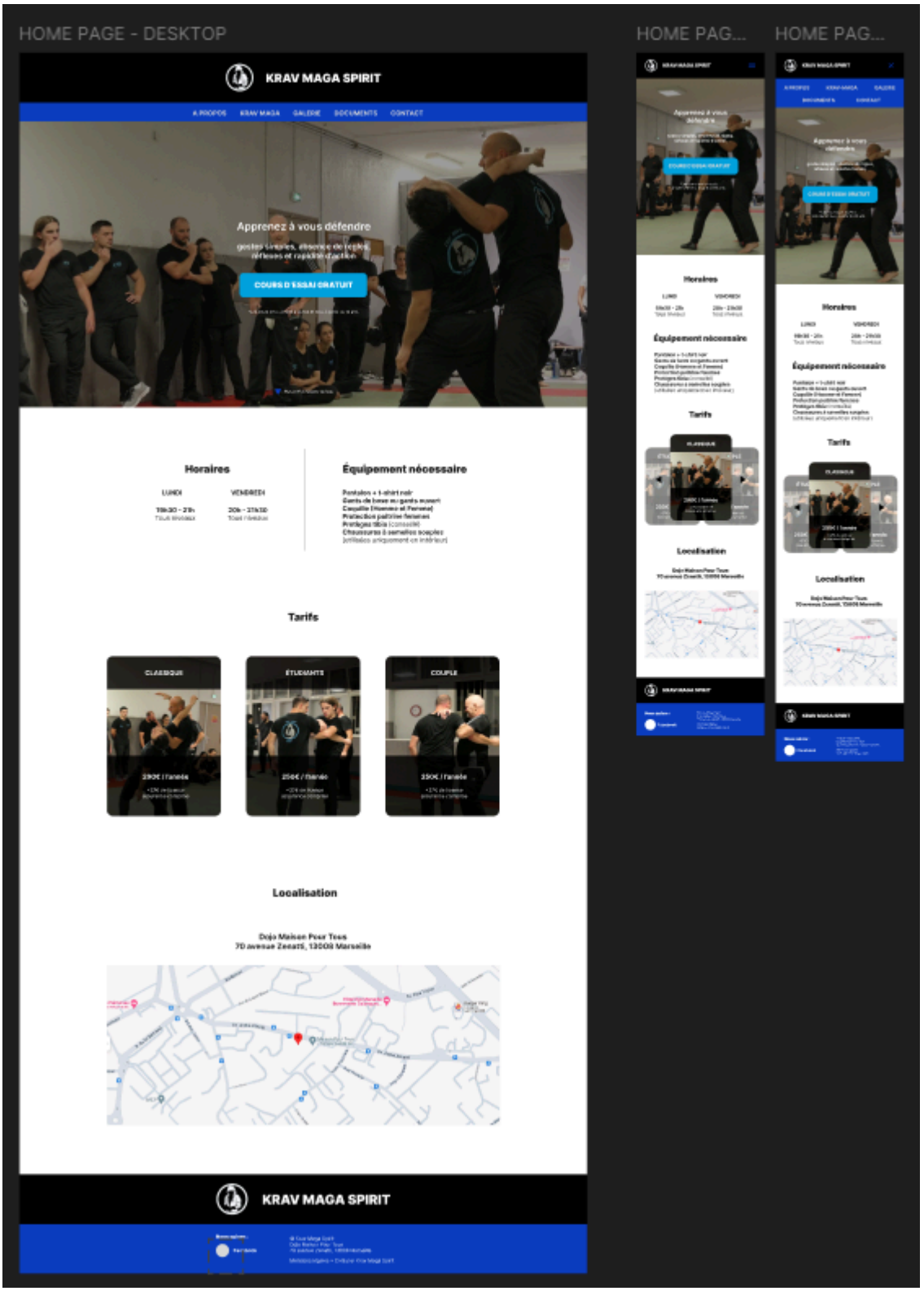
Équipement nécessaire

Pantalon + t-shirt noir
Protection poitrine femmes
Gants de boxe ou gants ouverts
Coquille
Protèges tibias
Chaussures à semelles souples

Tarifs

Localisation

Dojo Maison Pour Tous
70 avenue André Zenatti, 13008 Marseille


HOME PAGE - DESKTOP
HOME PAG...
HOME PAG...
KRAV MAGA SPIRIT
Apprenez à vous défendre
COURS D'ESSAI GRATUIT
Horaires
Équipement nécessaire
Tarifs
Localisation
KRAV MAGA SPIRIT

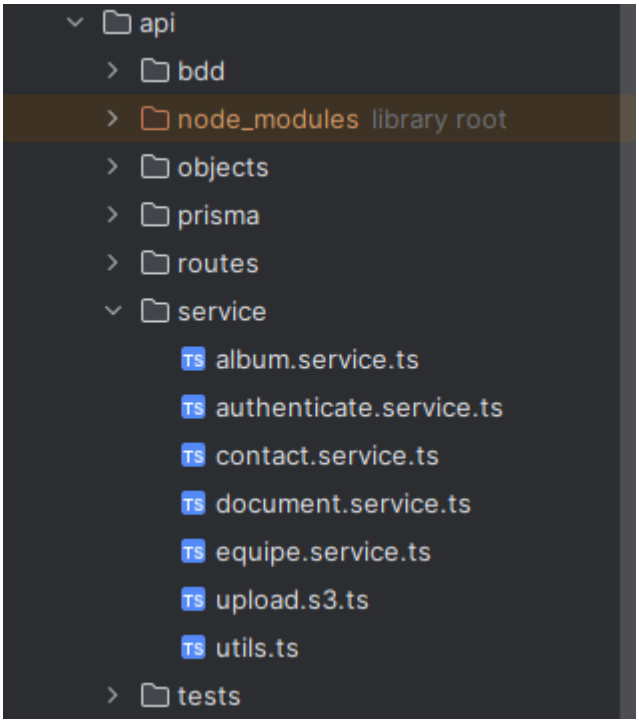
L'interface est responsive grâce aux media queries implémentées dans le CSS de chaque composant. Par exemple, voici une media query du CSS de la page Home :

```css
@media (max-width: 960px){

  #hero{
    background-image: url("/assets/photos/hero-img.png");
    background-position: center;
    background-position-y: center;
    background-repeat: no-repeat;
    background-size: cover;
    height: 60vh;

    color: white;
    margin: 0;

    display: flex;
    flex-direction: column;
    justify-content: center;
  }
```

Des tests unitaires ont été effectués sur les composants.

✔ Développer des composants métier

Le backend a été développé en Node.js avec TypeScript en suivant une architecture en microservices. Chaque service se concentre sur une partie précise de l'application. Il y a par exemple un service pour les albums, un service pour l'authentification, un service pour les documents, etc…



La sécurité est assurée de plusieurs façons sur cette partie de l'application :

- **Système d'authentification JWT** : un token avec une durée de vie de 2 heures est généré après une authentification réussie. Ce token contient l'identifiant utilisateur, le rôle de l'utilisateur et la date d'expiration du token. Comme le token ne contient pas d'information sensible, il n'y a aucun risque de sécurité, même si quelqu'un arrive à le déchiffrer.
- **Sécurisation des mots de passe** : les mots de passe choisis par les utilisateurs doivent adhérer à des critères de complexité (longueur minimale et caractères spéciaux). A la création du compte, les mots de passe sont hashés avec bcrypt avec un facteur de 12 rounds minimum.
- **Sécurisation des accès à la base de données** : Prisma prépare les requêtes automatiquement, ce qui évite les failles de sécurité venant d'un oubli, qui pourrait créer une faille XSS. La pool de connexion sécurisé évite les attaques par déni de service en limitant le nombre de connexions simultanées et mettant en place un timeout automatique. De plus, le logging automatique des opérations en base de données permet de pouvoir comprendre d'où vient le problème et de trouver une solution si jamais il y a un problème en base.

Des tests ont été mis en place pour les composants backend.

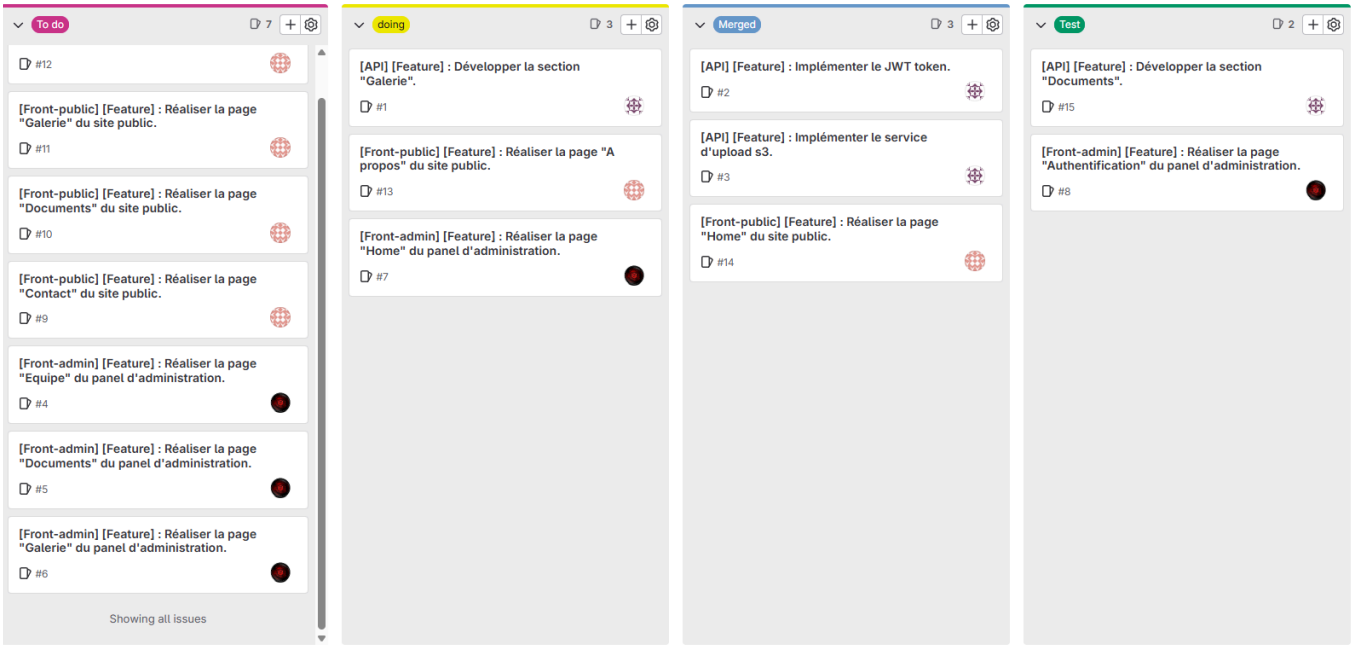
✔ Contribuer à la gestion d'un projet informatique

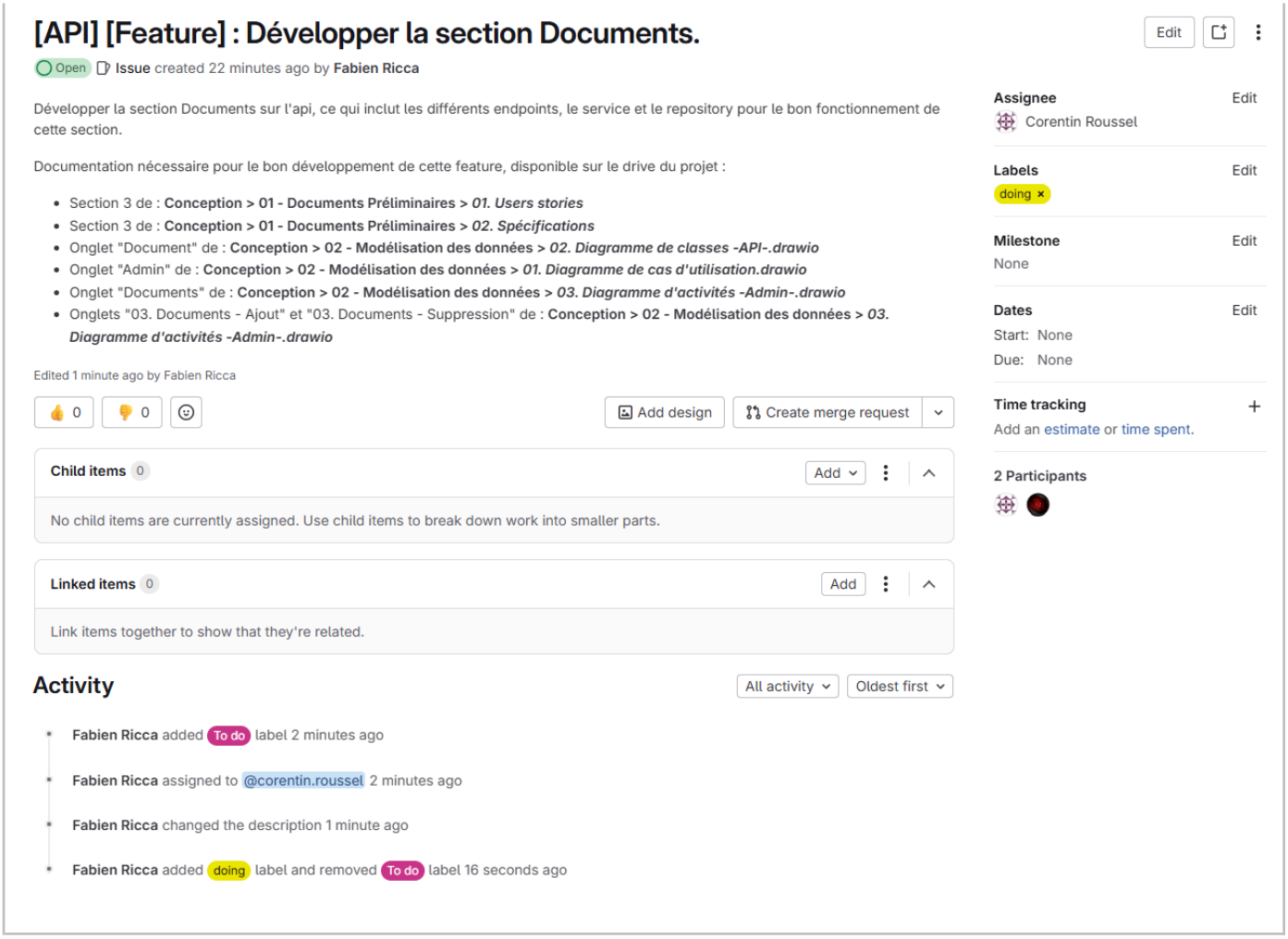
Nous avons choisi d'utiliser la méthodologie Agile pour ce projet. Nous avons donc fait des sprints d'une semaine et avons fait plusieurs réunions avec les clients, surtout pendant la phase de conception.

Nous avons fixé des rôles et des tâches principales pour chaque personne de l'équipe, les voici :

- **Fabien** : coordinateur du projet – il réalise les maquettes de la partie administrateur, planifie et organise les tâches de l'équipe et met en place la CI-CD
- **Corentin** : responsable du développement back-end – il réalise l'API et participe à la rédaction des spécifications fonctionnelles et techniques
- **Moi** : responsable du développement front-end – je réalise les maquettes de la partie publique et développe ensuite le front de la partie publique, participe à la rédaction des spécifications fonctionnelles et techniques et supervise l'avancement et le respect des règles mises en place des tickets GitLab

Nous avons aussi fait des tickets GitLab pour chaque tâche à faire, comme suit :





2. Précisez les moyens utilisés :

- **Node.js**, **Express et Go** pour développer la partie backend de l'application
- **Angular** pour développer la partie frontend de l'application
- **Prisma** pour générer automatiquement les requêtes
- **Docker** pour conteneurisé et isoler chaque composant de l'application
- **Docker Compose** pour connecter les conteneurs ensemble
- **Traefik** pour gérer les certificats SSL et les routes dynamique, ce qui garantit un accès sécurisé à l'application
- **Git** pour le versionning
- **GitLab** pour héberger notre code source, créer et gérer des tickets et mettre en place nos pipelines CI/CD
- **IntelliJ IDEA** et **Visual Studio Code** pour éditer le code

3. Avec qui avez-vous travaillé ?

Fabien RICCA et Corentin ROUSSEL

4. Contexte

Nom de l'entreprise, organisme ou association ▸ La Plateforme

Chantier, atelier, service ▸ Dans le cadre de la formation Concepteur Développeur d'Applications

Période d'exercice ▸ Du : 01/2025 au : 06/2025

5. Informations complémentaires *(facultatif)*

DOSSIER PROFESSIONNEL (DP)

Activité-type 2

Concevoir et développer une application sécurisée organisée en couches

Exemple n° 1 – KMS

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

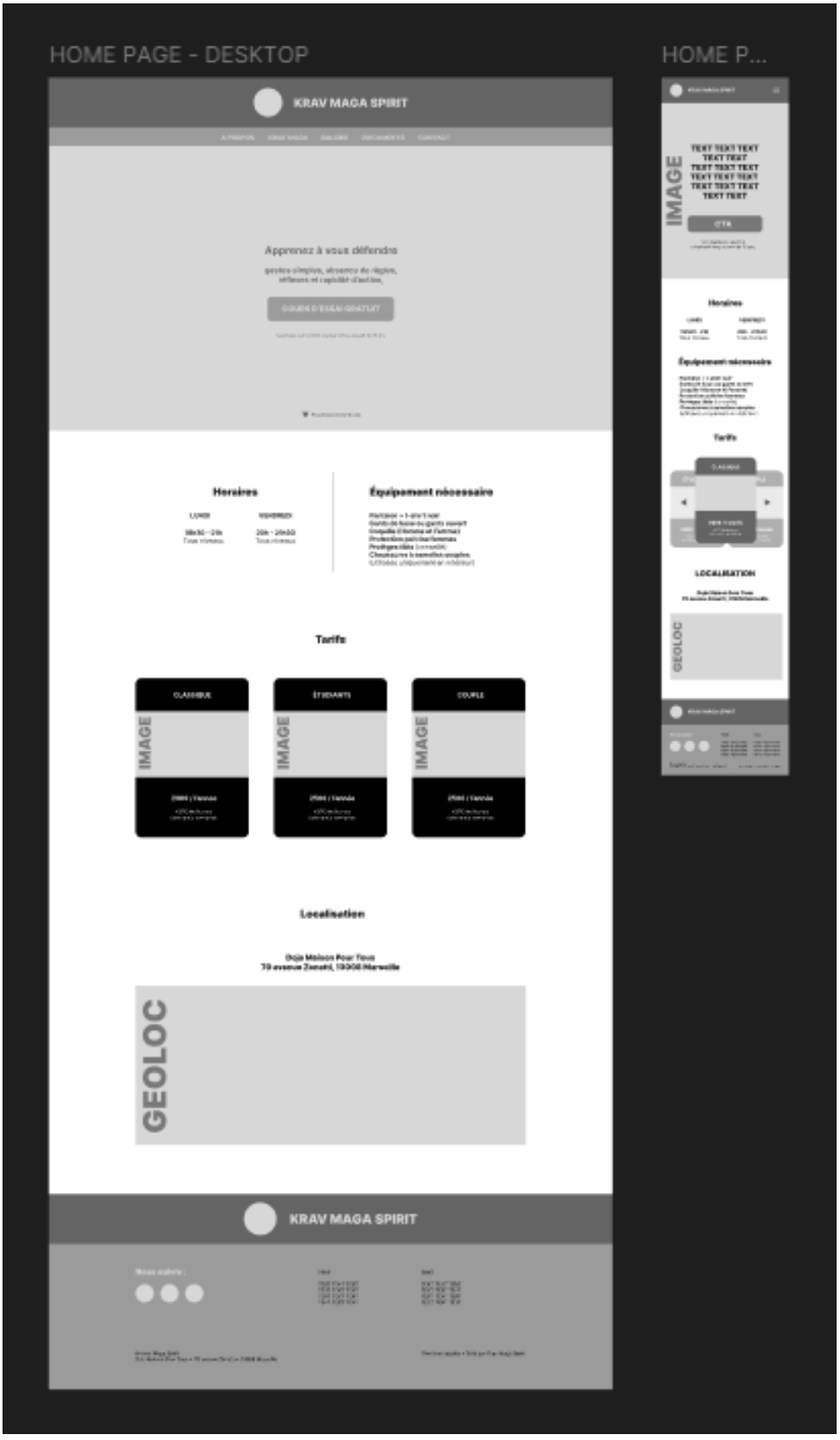
✔ **Analyser les besoins et maquetter une application**

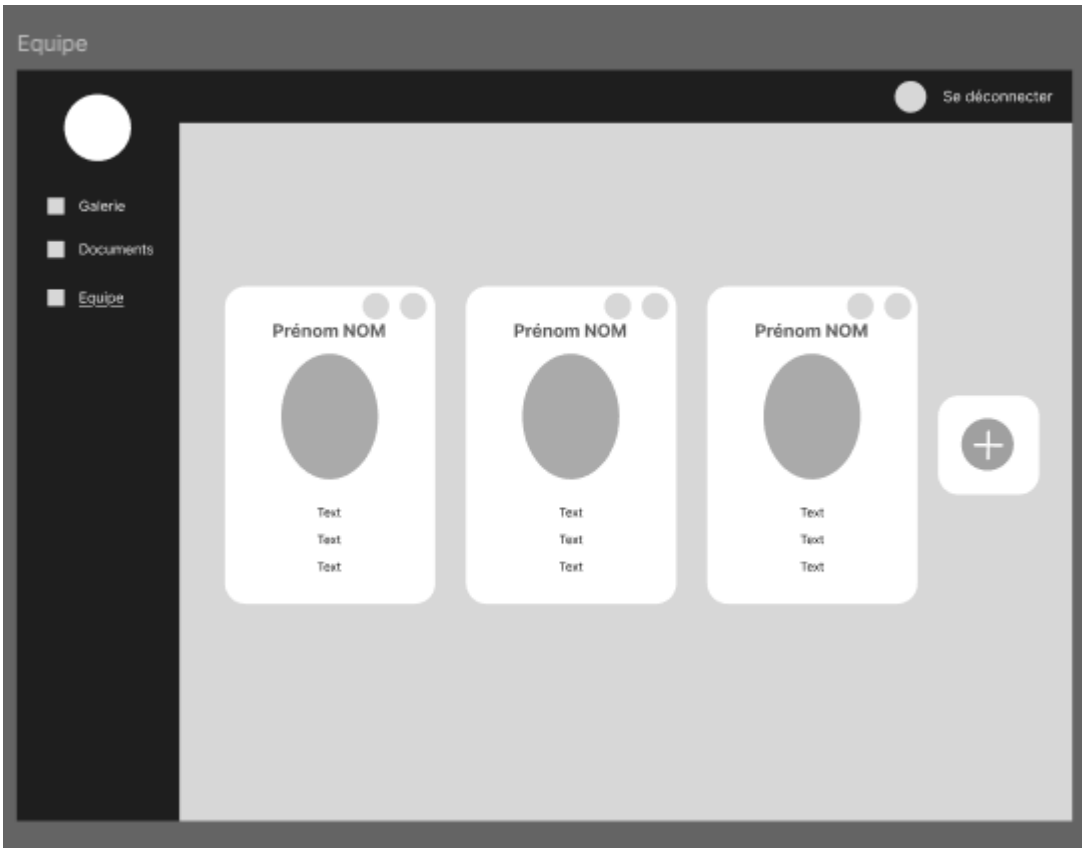
Pour connaître les besoins du client, nous avons discuté avec lui tout au long de la conception du site et nous avons créé plusieurs documents qui montrent en détail les besoins exprimés par le client (vous pouvez retrouver tous les documents dans les annexes de mon dossier de projet) :

- **Cahier des charges** : Décrit les besoins du client et les objectifs du projet.
- **Spécifications techniques** : Décrit les fonctionnalités et les contenus attendus.
- **User stories** : Décrit les critères d'acceptation des fonctionnalités en se mettant dans la peau des utilisateurs.

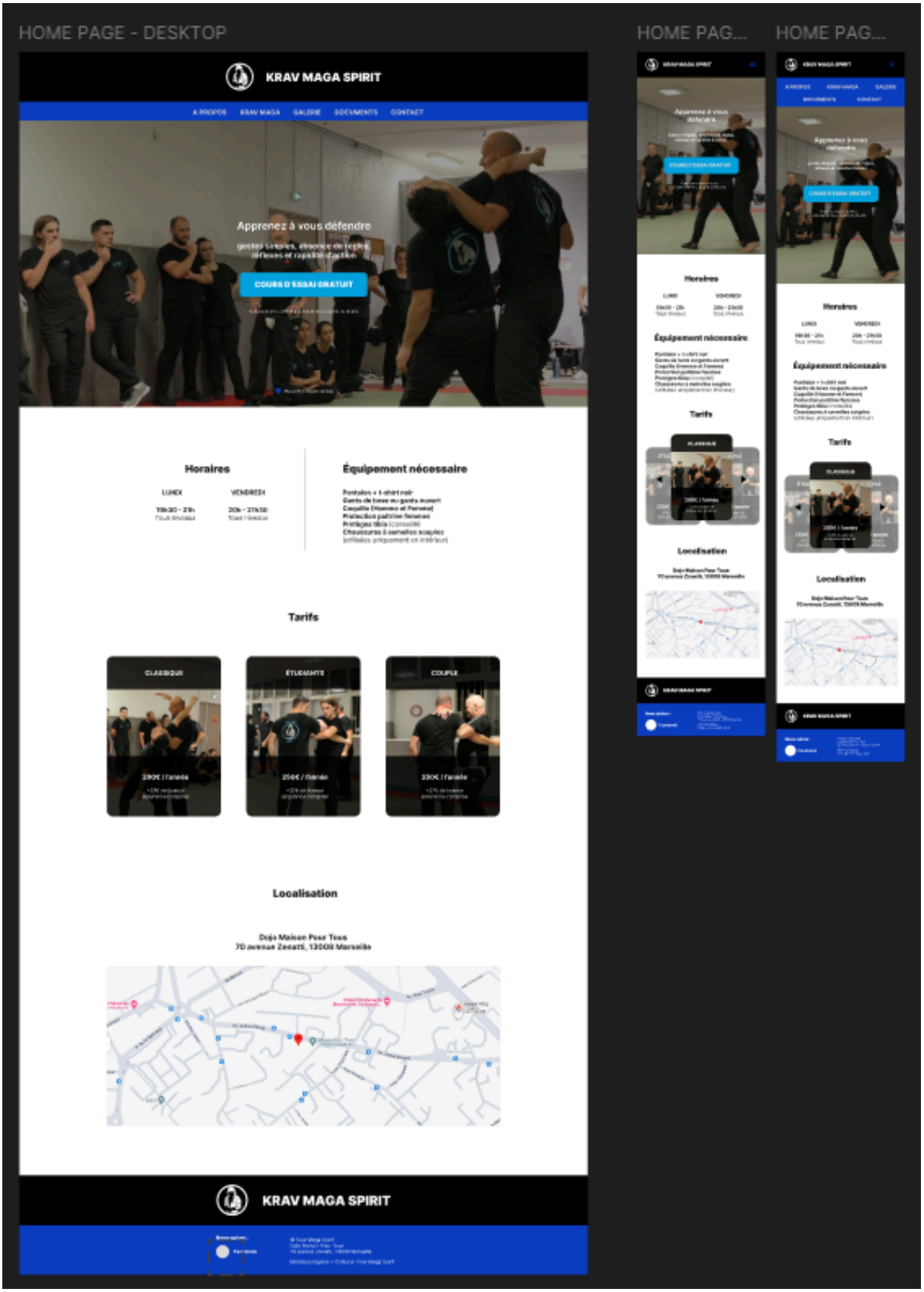
J'ai ensuite maquetté l'application, en veillant à intégrer tous les contenus et les fonctionnalités demandés par le client.

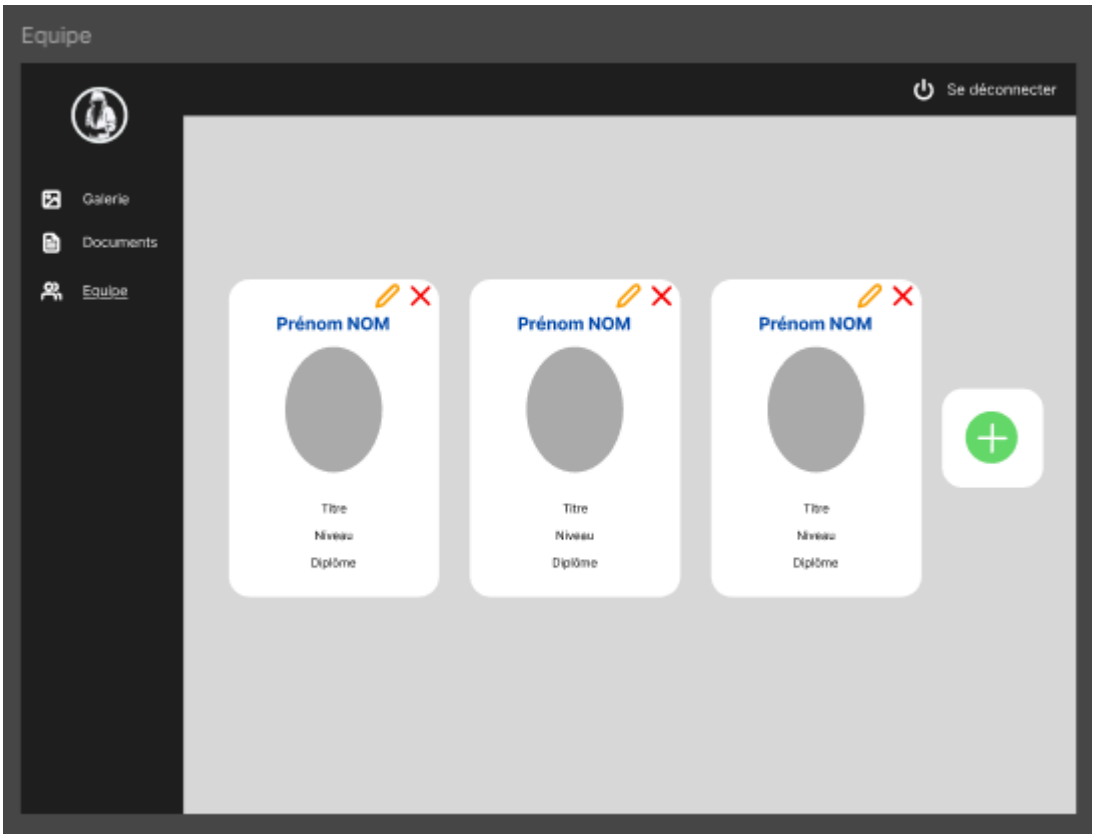
Pour cela, j'ai commencé par créer un wireframe, sans couleur, images ou icônes :


HOME PAGE - DESKTOP
KRAV MAGA SPIRIT
Apprenez à vous défendre
Horaires
Équipement nécessaire
Tarifs
CLASSIQUE
ÉTUDIANTS
COUPLE
IMAGE
Localisation
GEOLOC
KRAV MAGA SPIRIT
HOME P...
TEXT TEXT TEXT
TEXT TEXT
TEXT TEXT TEXT
TEXT TEXT TEXT
TEXT TEXT TEXT
TEXT TEXT
CTA
Horaires
Équipement nécessaire
Tarifs
LOCALISATION
GEOLOC



Puis j'ai ajouté toutes les couleurs, les images, les icônes et les textes pour créer la maquette finale, celle que nous avons utilisé pour créer le front de l'application :


HOME PAGE - DESKTOP
HOME PAG...
HOME PAG...
KRAV MAGA SPIRIT
Apprenez à vous défendre
COURS D ESSAI GRATUIT
Horaires
Équipement nécessaire
Tarifs
Localisation
KRAV MAGA SPIRIT

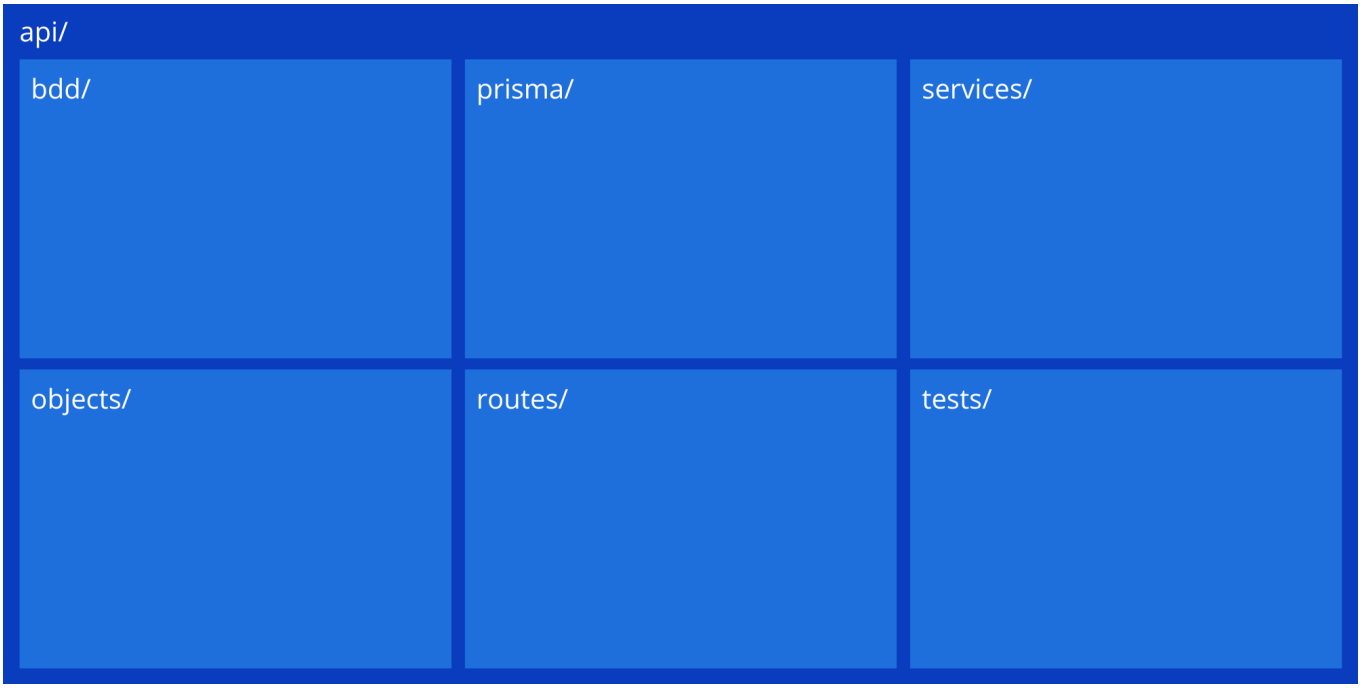


✔ **Définir l'architecture logicielle d'une application**

Back

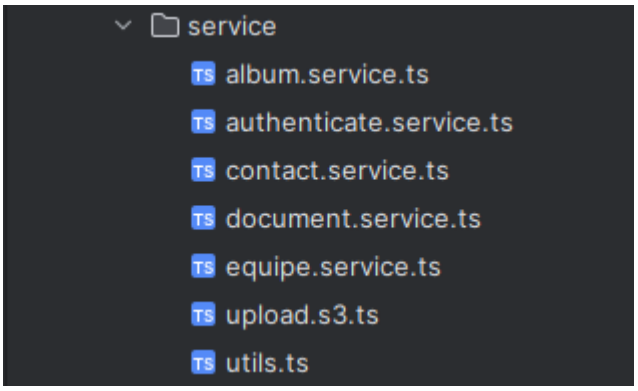
Pour développer la partie backend de l'application est une API REST, que nous avons choisi de développer en Node.js avec Express.

Voici un schéma qui représente l'architecture de l'API :



- ➔ bdd/ contient les modèles des objets présents dans la base de données, comme album, document, equipe, etc…
- ➔ objects/ contient les DTOs.
- ➔ prisma/ contient tous les fichiers pour prisma, comme les backups, la configuration, les migrations, etc…
- ➔ routes/ contient les routes exposées par l'API.
- ➔ services/ contient les services utilisés par les routes.
- ➔ tests/ contient les tests unitaires et d'intégration.

Comme une architecture en microservices a été choisie pour cette API, chaque service s'occupe d'une partie précise de l'application :



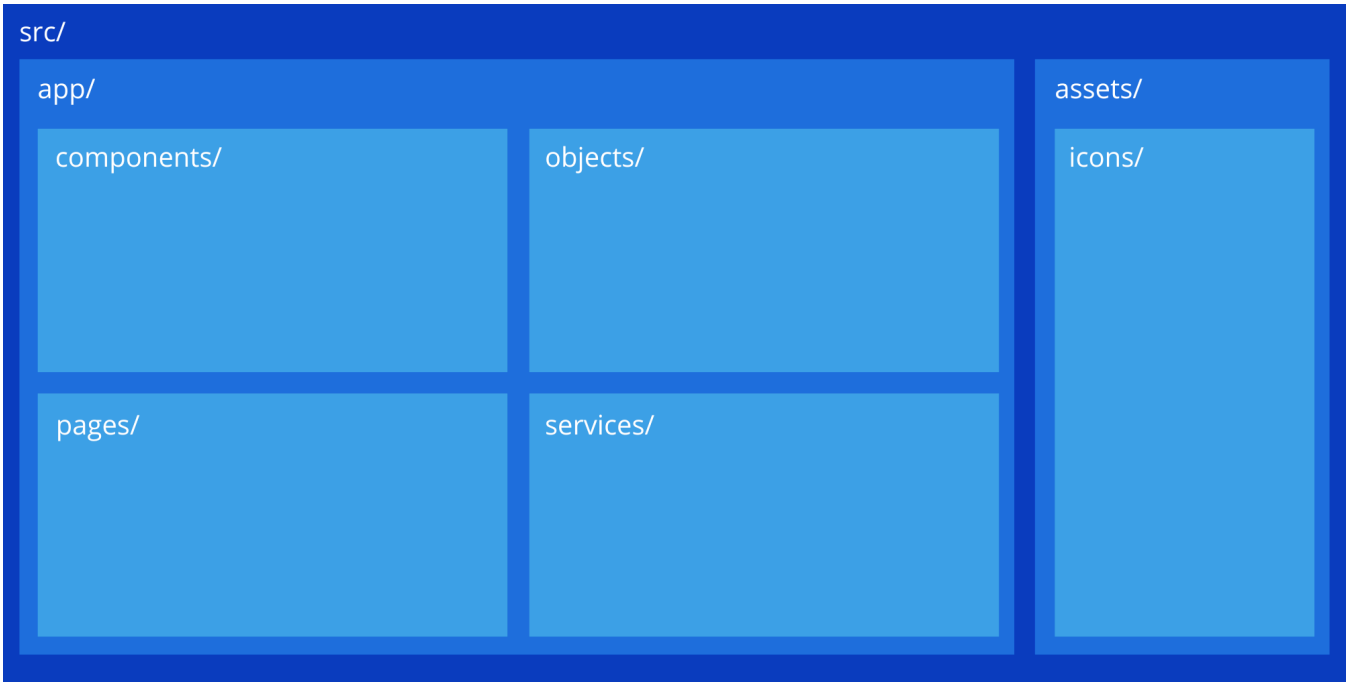
Front

La partie frontend de l'application a été développée en Angular. Angular a été choisi pour sa légèreté et

pour le fait que beaucoup d'entreprises l'utilisent.

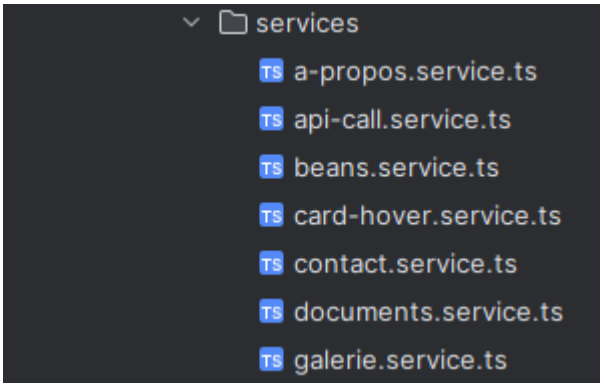
Le front de l'application est divisé en deux parties : la partie publique et la partie admin. Les 2 parties ont beau être divisées, elles gardent tout de même la même architecture.

Voici un schéma de l'architecture des fronts de l'application :



- ➔ components/ contient les composants UI réutilisables comme le header, le footer, les cards, etc…
- ➔ pages/ contient les composants de chaque page du site, comme home, documents, galerie, etc…
- ➔ objects/ contient les classes utilisées pour créer les objets utilisés dans les pages ou les composants.
- ➔ services/ contient les appels aux APIs et la logique métier.

Du côté front aussi les services ne s'occupent que d'une partie de l'application :

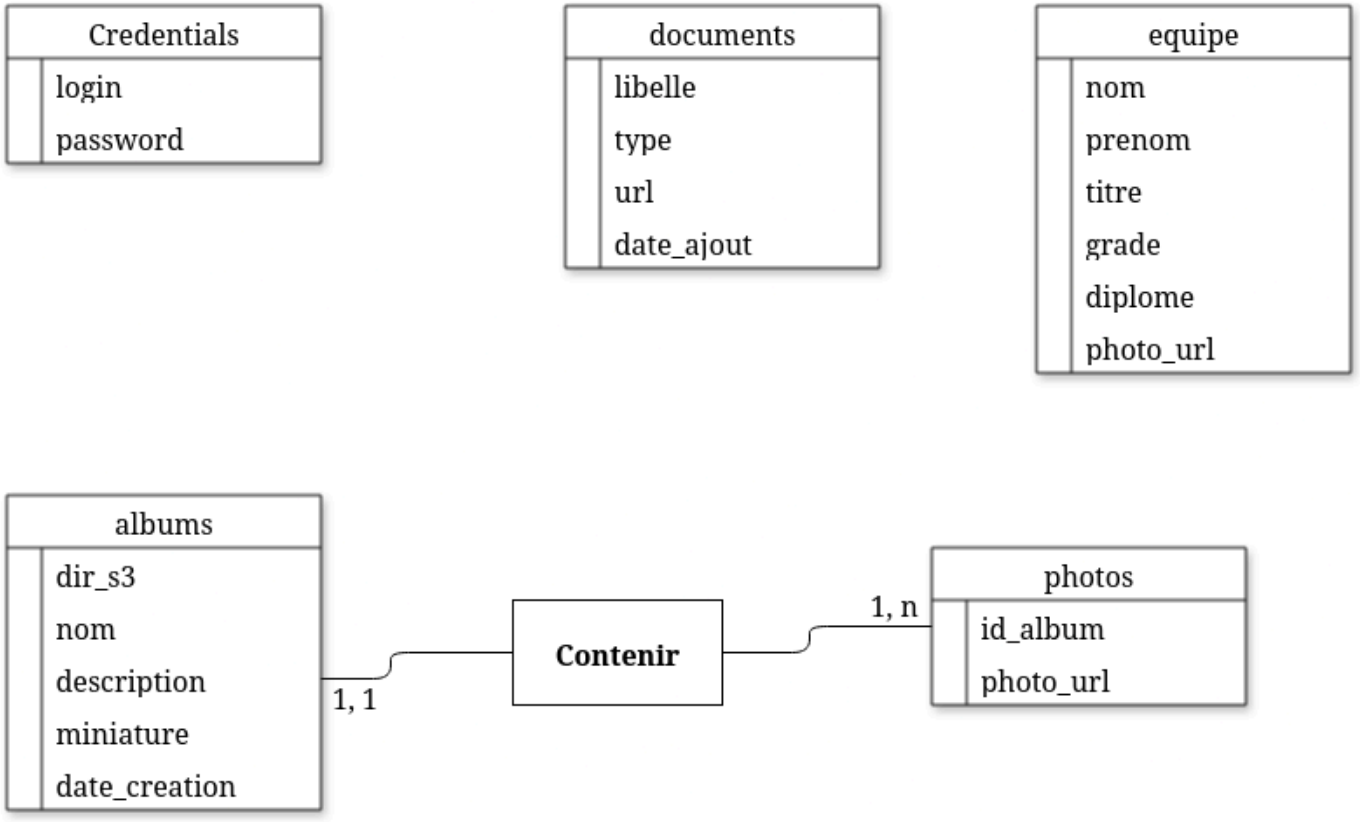


✔ **Concevoir et mettre en place une base de données relationnelle**

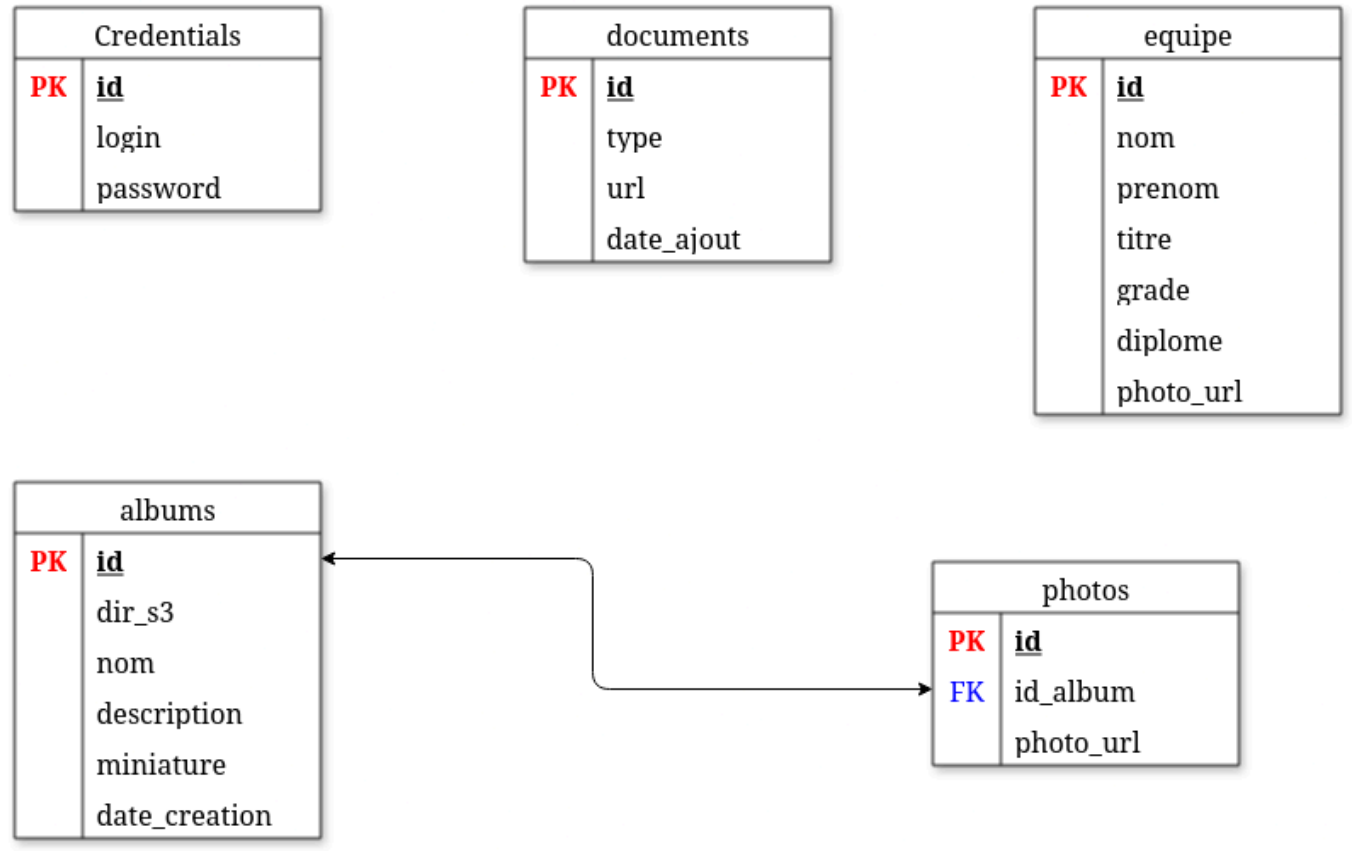
J'ai participé à la conception des bases de données. Nous avons 2 bases de données : une en PostgreSQL que nous appelons avec la partie du code que nous avons développé en Node.js, et une en MongoDB qui est appelée par la partie du code développée en Go.

Nous avons utilisé la méthode Merise pour concevoir nos bases, voici donc les schémas qui en ont résulté :

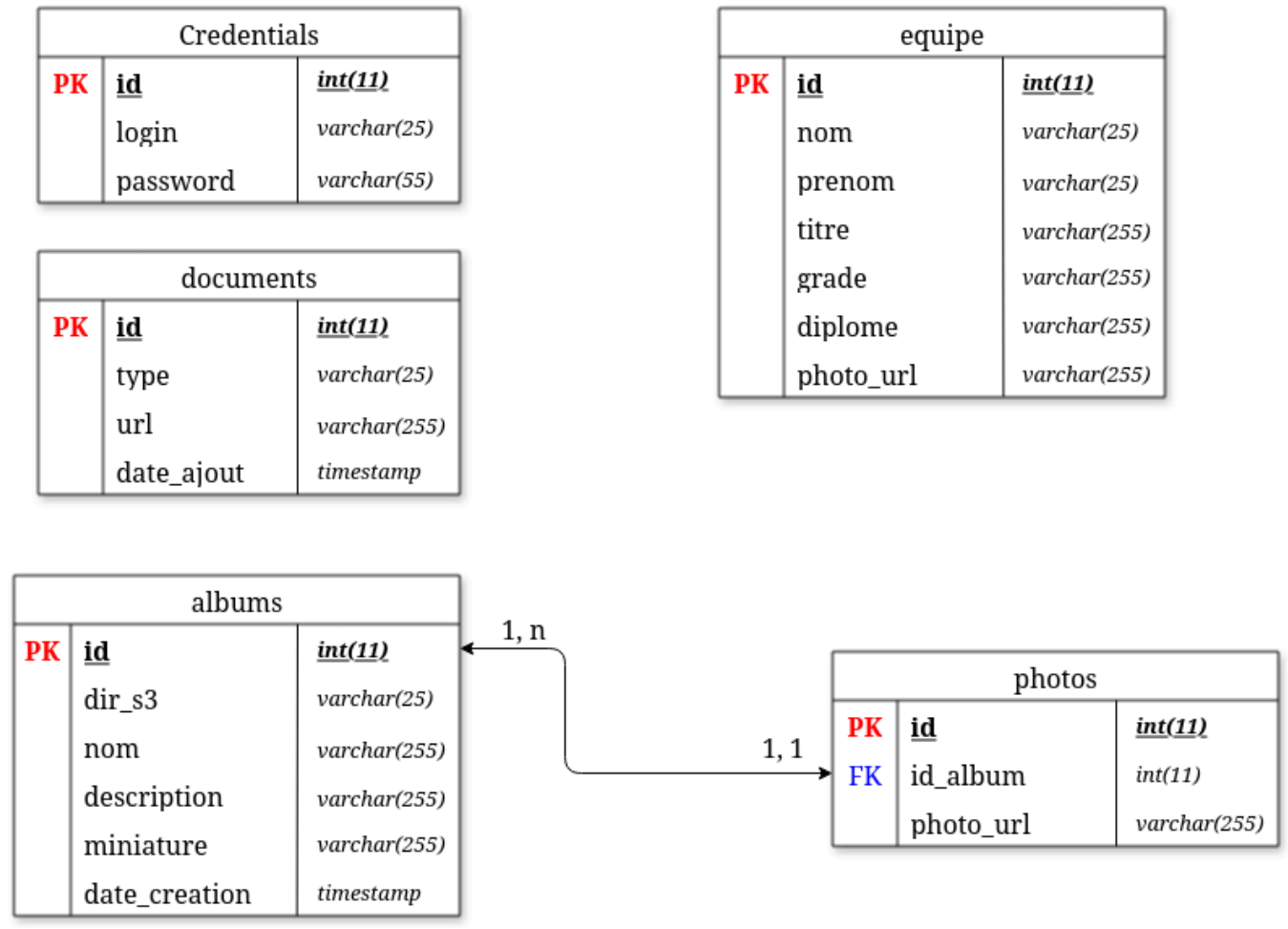
MCD :



MLD :


Credentials
PK id
login
password
documents
PK id
type
url
date_ajout
equipe
PK id
nom
prenom
titre
grade
diplome
photo_url
albums
PK id
dir_s3
nom
description
miniature
date_creation
photos
PK id
FK id_album
photo_url

MPD :


Credentials
PK id int(11)
login varchar(25)
password varchar(55)
equipe
PK id int(11)
nom varchar(25)
prenom varchar(25)
titre varchar(255)
grade varchar(255)
diplome varchar(255)
photo_url varchar(255)
documents
PK id int(11)
type varchar(25)
url varchar(255)
date_ajout timestamp
albums
PK id int(11)
dir_s3 varchar(25)
nom varchar(255)
description varchar(255)
miniature varchar(255)
date_creation timestamp
1, n
1, 1
photos
PK id int(11)
FK id_album int(11)
photo_url varchar(255)

Base MongoDB :

Credentials		
PK	**UniqueID**	***ObjectID***
	username	*String*
	password	*String*
	jwt_refresh_token	*String*
	create_at	*Date*
	updated_at	*Date*

Quand nous avons fini de concevoir les bases de données, j'ai créé les models pour Prisma puis générer le fichier SQL pour créer la base de données.

✔ **Développer des composants d'accès aux données SQL et NoSQL**

Dans la partie de l'application développée en Node et qui utilise la BDD PostgreSql, j'ai implémenté les accès aux données grâce à Prisma ORM.

Les accès aux données incluent la création, la lecture, la modification et la suppression (CRUD) des entités en base de données et d'autres actions, selon les besoins que nous avons eu pendant le développement.

Création d'un album :

```
/** Méthode appelée pour la création d'un nouvel album. ...*/
async createAlbum(album: AlbumDTO) : Promise<{ id: number; dirS3: string; nom:... { Show usages  fabien-ricca
  try {
    const result : { id: number; dirS3: string; nom: string;... = await this.prisma.album.create({
      data: {
        dirS3: album.dir_s3,
        nom: album.nom,
        description: album.description || null,
        miniature: album.miniature,
        dateCreation: new Date(album.date_creation) // Convertir le timestamp en Date
      }
    });

    console.log(`[INFO][${this.currentDate()}] --> Création de l'album n°${result.id} : "${result.nom}".`);
    return result;
  } catch (error) {
    console.log(`[ERROR][${this.currentDate()}] --> Une erreur a été rencontrée lors de la création de l'album "${album.nom}" :\n ${error}`);
    console.log(error);
    throw error;
  }
}
```

Récupération d'un album :

```
/** Méthode appelée pour récupérer la liste de tous les albums existants. ...*/
async getAlbums() : Promise<{ id: number; dirS3: string; nom:... { Show usages  fabien-ricca
  try {
    const albums : { id: number; dirS3: string; nom: string;... = await this.prisma.album.findMany({
      orderBy: {
        dateCreation: 'desc'
      }
    });

    console.log(`[INFO][${this.currentDate()}] --> Récupération des album.`);
    return albums;
  } catch (error) {
    console.log(`[ERROR][${this.currentDate()}] --> Une erreur a été rencontrée lors de la récupération des albums :\n ${error}`);
    console.log(error);
    throw error;
  }
}
```

Modification d'un album :

```
/** Méthode appelée pour la modification d'un album. ...*/
async updateAlbum(album: AlbumDTO): Promise<boolean> {  Show usages  fabien-ricca
  try {
    if (!album.id) {
      throw new Error("L'ID de l'album est requis pour la modification");
    }

    const result : { id: number; dirS3: string; nom: string;... } = await this.prisma.album.update({
      where: {
        id: album.id
      },
      data: {
        nom: album.nom,
        description: album.description || null,
        dateCreation: new Date(album.date_creation) // Convertir le timestamp en Date
      }
    });

    console.log(`[INFO][${this.currentDate()}] --> Modification de l'album n°${album.id} : ${album.nom}`);
    return !!result;
  } catch (error) {
    console.log(`[ERROR][${this.currentDate()}] --> Une erreur a été rencontrée lors de la modification de l'album n°${album.id} : ${album.nom} :\n ${error}`);
    console.log(error);
    throw error;
  }
}
```

Suppression d’un album :

```
125   /** Méthode appelée pour la suppression d'un album. ...*/
129   async deleteAlbum(id: number) : Promise<{ dir_s3: string }[]> {  Show usages  fabien-ricca
130     try {
131       // Récupérer les infos de l'album avant suppression (pour le dir_s3)
132       const albumToDelete : { id: number; dirS3: string; nom: string;... = await this.prisma.album.findUnique({
133         where: { id: id }
134       });
135
136       if (!albumToDelete) {
137         return null;
138       }
139
140       // Supprimer l'album (les photos sont supprimées automatiquement par CASCADE)
141       const result : { id: number; dirS3: string; nom: string;... = await this.prisma.album.delete({
142         where: { id: id }
143       });
144
145       console.log(`[INFO][${this.currentDate()}] --> Suppression de l'album n° ${id}`);
146
147       // Retourner dans le même format que l'ancien code pour compatibilité
148       return [{ dir_s3: albumToDelete.dirS3 }];
149     } catch (error) {
150       console.log(`[ERROR][${this.currentDate()}] --> Une erreur a été rencontrée lors de la suppression de l'album n°${id} :\n ${error}`);
151       console.log(error);
152       throw error;
153     }
154   }
```

On peut aussi voir que les exceptions sont gérées grâce à des blocs try/catch et des logs ont été mis en place sur la console pour savoir s'il y a eu une erreur durant l'accès aux données, ce qui peut aider à déboguer.

En utilisant des DTOs pour transférer les données, je sécurise un peu plus l'application car ça permet de transférer seulement les données dont j'ai besoin, ce qui permet de limiter la fuite de données sensible si quelqu'un arrive à intercepter les données qui transitent entre la base de données et l'API.

La partie de l'application développée en Go utilise la BDD MongoDB.

La base de données MongoDB facilite l'insertion grâce à ses fonctions intégrées, sans qu'il soit nécessaire d'écrire manuellement des requêtes complexes. Cela limite les risques liés aux erreurs de requêtes personnalisées ou aux injections.

Par exemple, pour inscrire un utilisateur, il faut :

1. Ouvrir une collection

```
20      collection := repository.OpenCollection(repository.Client, "users")
41
42      var collection *mongo.Collection = client.Database("auth-go").Collection(collectionName)
43
44      return collection
45  }
```

2. Compter combien d'utilisateurs ont l'email donné pour vérifier que l'utisateur n'existe pas déjà

```
31      countEmail, _ := collection.CountDocuments(context.TODO(), bson.M{"email": user.Email})
```

3. Pareil pour le nom d'utilisateur

```
36      countUsername, _ := collection.CountDocuments(context.TODO(), bson.M{"username": user.Username})
```

4. Insérer les informations de l'utilisateur dans la base de données

```
46          user.ID = primitive.NewObjectID()
47          user.Password = &hashedPassword
48          user.CreatedAt = time.Now()
49          user.UpdatedAt = time.Now()
50          user.UserID = user.ID.Hex()
51
52          _, err = collection.InsertOne(context.TODO(), user)
```

Nous avons aussi mis en place une gestion rigoureuse des erreurs : chaque fonction renvoie un message d'erreur si le résultat attendu n'est pas obtenu. Ces messages restent généraux, notamment pour les opérations sensibles comme la connexion ou l'inscription, afin de ne pas divulguer d'informations exploitables par un potentiel attaquant.

```
41      hashedPassword, err := utils.HashPassword(*user.Password)
42      if err != nil {
43          return c.JSON(http.StatusConflict, echo.Map{"error": "Un probléme est survenu"})
44      }
```

2. Précisez les moyens utilisés :

- **Angular** pour développer le frontend
- **Node.js** et Express pour le backend de l'API
- **Go** pour l'authentification
- **Prisma ORM** pour simplifier l'accès aux données à partir de l'API
- **Figma** pour les maquettes et wireframes
- **draw.io** pour les schémas de base de données

3. Avec qui avez-vous travaillé ?

DOSSIER PROFESSIONNEL (DP)

Fabien RICCA et Corentin ROUSSEL

4. Contexte

Nom de l'entreprise, organisme ou association ▸ La Plateforme

Chantier, atelier, service ▸ Dans le cadre de la formation Concepteur Développeur d'Applications

Période d'exercice ▸ Du : 01/2025 au : 06/2025

5. Informations complémentaires *(facultatif)*

Activité-type 3 Préparer le déploiement d'une application sécurisée

Exemple n° 1 - KMS

1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

✔ **Préparer et exécuter les plans de tests d'une application**

Des tests unitaires et d'intégration ont été mis en place sur une grande partie de l'application, ce qui nous permet de vérifier si les méthodes fonctionnent bien comme prévu quand on change le code pour ajouter une fonctionnalité par exemple.

Plus précisément, les tests unitaires permettent de s'assurer qu'une méthode isolée se comporte comme prévu et les tests d'intégration permettent de vérifier que des méthodes fonctionnent ensemble comme prévu.

Voici un exemple de tests unitaires :

```
describe('# sendEmail', () :void => {

    test('devrait envoyer un email avec succès', async () :Promise<void> => {
        // Arrange
        const mockData = {
            nom: 'Dupont',
            prenom: 'Jean',
            mail: 'jean.dupont@example.com',
            telephone: '0123456789',
            message: 'Ceci est un message de test'
        };

        mockUtils.checkRequiredFields.mockReturnValue(true);
        mockSendMail.mockResolvedValue({ messageId: 'test-message-id' });

        // Act
        const result :any = await contactService.sendEmail(mockData);

        // Assert
        expect(result).toBeInstanceOf(Response);
        expect(result).toHaveProperty('code', 200);
        expect(result).toHaveProperty('message', 'Votre message a été envoyé avec succès.');
        expect(mockUtils.checkRequiredFields).toHaveBeenCalledWith(
            mockData,
            ['nom', 'prenom', 'mail', 'telephone', 'message']
        );
        expect(mockSendMail).toHaveBeenCalledWith({
            from: 'test@example.com',
            to: 'test@example.com',
            subject: 'Nouveau message de Jean DUPONT',
            html: expect.stringContaining('Nouveau message de contact')
        });
    });
```

Et un exemple de test d'intégration :

```
describe('🧪 Tests de Validation', () : void => {

    test('devrait valider correctement les emails', () : void => {
        const emailTests : { email: string; expected: boolean; descr... = [
            { email: 'test@example.com', expected: true, description: 'Email valide standard' },
            { email: 'user.name@domain.co.uk', expected: true, description: 'Email avec points et domaine complexe' },
            { email: 'test123@test-domain.org', expected: true, description: 'Email avec chiffres et tirets' },
            { email: 'invalid-email', expected: false, description: 'Email sans @' },
            { email: 'test@', expected: false, description: 'Email sans domaine' },
            { email: '@example.com', expected: false, description: 'Email sans utilisateur' },
            { email: 'test@domain', expected: false, description: 'Email sans extension' },
            { email: '', expected: false, description: 'Email vide' },
            { email: 'test spaces@example.com', expected: false, description: 'Email avec espaces' },
            { email: 'test@exam ple.com', expected: false, description: 'Domaine avec espaces' }
        ];

        emailTests.forEach(({ email, expected, description } : { email: string; expected: boolean; descr... ) : void => {
            const result : any = contactService.isEmailValide(email);

            // Assertions Jest
            expect(result).toBe(expected);
        });
    });
```

Nous avons aussi mis en place de tests end-to-end manuel à chaque ajout de fonctionnalité pour vérifier que l'application fonctionne bien dans son ensemble.

✔ Préparer et documenter le déploiement d'une application

Pour le déploiement, j'ai utilisé Docker et Docker Compose afin de lancer tous les services (frontend, backend, base de données) dans un environnement isolé et cohérent.

Voici le docker-compose.yml :

```yaml
services:
  kms_public:
    build:
      context: .
      dockerfile: web-public/Dockerfile-front-public
    container_name: kms_public
    ports:
      - "80:80"
    networks:
      - net-kms

  kms_admin:
    build:
      context: .
      dockerfile: web-admin/Dockerfile-front-admin
    container_name: kms_admin
    ports:
      - "8085:80"
    networks:
      - net-kms

  kms_api:
    build:
      context: .
      dockerfile: api/Dockerfile-back
    container_name: kms_api
    env_file:
      - api/.env
    ports:
      - "8060:8060"
    depends_on:
      - kms_bdd
    networks:
      - net-kms
```

```yaml
  kms_bdd:
    build:
      context: .
      dockerfile: api/Dockerfile-database
    container_name: kms_bdd
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=root
      - POSTGRES_DB=kms
    env_file:
      - api/.env
    ports:
      - "5435:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data
    networks:
      - net-kms

volumes:
  postgres_data:

networks:
  net-kms:
    driver: bridge
```

Le build Angular permet de générer des fichiers statiques optimisés.

Traefik gère le routage des requêtes et la sécurité HTTPS.

Enfin, des outils comme ESLint et Husky ont été mis en place pour garantir un code propre et uniforme avant chaque commit.

✔ Contribuer à la mise en production dans une démarche DevOps

La mise en production s'appuie sur GitLab CI/CD. À chaque push sur la branche principale, le pipeline se déclenche automatiquement : il construit les images Docker, déploie le projet sur le VPS et redémare tous les services dans les containers.

Avant chaque mise en ligne, des tests unitaires, d'intégration et end-to-end manuels sont effectués pour s'assurer du bon fonctionnement global.
Ce processus permettait de déployer rapidement, en limitant les erreurs humaines.

2. Précisez les moyens utilisés :

- Jest pour les tests unitaires et d'intégration
- ESLint et Husky pour standardiser le code

3. Avec qui avez-vous travaillé ?

Fabien RICCA et Corentin ROUSSEL

4. Contexte

Nom de l'entreprise, organisme ou association ▸ La Plateforme

Chantier, atelier, service ▸ Dans le cadre de la formation Concepteur Développeur d'Applications

Période d'exercice ▸ Du : 01/2025 au : 06/2025

5. Informations complémentaires *(facultatif)*

Titres, diplômes, CQP, attestations de formation

(facultatif)

Intitulé	Autorité ou organisme	Date
RNCP niveau 5 : Développeur web et web mobile	**Ministère du travail, du plein emploi et de l'insertion**	**Juillet 2023**

Déclaration sur l'honneur

Je soussigné(e) [prénom et nom] Léa Dubois ,

déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l'auteur(e) des réalisations jointes.

Fait à Marseille le 31/07/2025

pour faire valoir ce que de droit.

Signature :

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé
Cliquez ici pour taper du texte.

ANNEXES

(Si le RC le prévoit)