



DOSSIER PROFESSIONNEL (DP)

Nom de naissance ROUSSEL
Nom d'usage ROUSSEL
Prénom Corentin
Adresse 39 Quater allées de craponne

Titre professionnel visé

Concepteur Développeur d'Ap&plication

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.

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

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

DOSSIER PROFESSIONNEL (DP)

SOMMAIRE

Exemples de pratique professionnelle

| | | |
|---|----|-----------|
| Développer une application sécurisée | p. | 5 |
| - KMS | p. | 5 |
| - Smart Locker | p. | 12 |
| Concevoir et développer une application sécurisée organisée en couches | p. | 17 |
| - KMS | p. | 17 |
| - Smart Locker | p. | 25 |
| Préparer le déploiement d'une application sécurisée | p. | 33 |
| - KMS | p. | 33 |
| Titres, diplômes, CQP, attestations de formation | p. | 37 |
| Déclaration sur l'honneur | p. | 38 |
| Documents illustrant la pratique professionnelle (facultatif) | p. | |
| Annexes (Si le RC le prévoit) | p. | |

EXEMPLES DE PRATIQUE

PROFESSIONNELLE

DOSSIER PROFESSIONNEL (DP)

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 :

Installation et développement

Prérequis

- Angular version 17 minimum
- Node.js version 20 minimum
- Docker et Docker Compose (obligatoire, même en local)
- Git

⚠️ Important : Un environnement Docker est nécessaire même en développement local pour lancer au minimum la base de données.

Pour ce projet, j'ai installé et configuré un environnement de développement basé sur Docker.

Chaque service (frontend, backend, base de données) était conteneurisé. Nous avons utilisé Traefik comme reverse proxy pour gérer les routes et les accès. L'éditeur NeoVim a été personnalisé pour le développement (plugins, syntaxe, linters).

```
● nvim-lint 0.38ms 17 BufReadPre    █ already up to date
● nvim-lspconfig 51.11ms 🚀 start    █ already up to date
● nvim-nio 0.13ms ✨ nvim-dap      █ already up to date
● nvim-treesitter 3.85ms 🚀 start    █ already up to date
```

GitLab a été utilisé pour la gestion de versions, la collaboration (issues, merge requests) et les pipelines CI/CD. L'environnement de travail a été documenté en interne pour faciliter la prise en main par les autres membres de l'équipe.

Un **Readme** a été établi au début du projet afin de suivre l'avancé de celui-ci et modifier tout au long du projet pour convenir au changement apportés vous trouverez des screens de celui-ci plus bas qui explique les différentes parties du référentiel a validé

L'installation de **Git** a été une étape indispensable pour permettre le **clonage du dépôt distant** contenant le projet, ainsi que pour assurer le **suivi des versions** du code tout au long du

développement. Git a facilité la gestion collaborative grâce à des branches dédiées pour chaque fonctionnalité, le suivi des modifications, et la résolution des conflits lors des fusions.

L'**interface utilisateur** a été conçue et développée à l'aide du **framework Angular**, qui offre une architecture modulaire, une gestion efficace des composants, et des outils puissants pour le **data-binding** et la **navigation entre vues**. Ce choix a permis de produire une application **réactive, maintenable et évolutive**, tout en assurant une bonne séparation des responsabilités côté frontend.

```
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({
  selector: 'home',
  standalone: true,
  imports: [
    HeaderComponent,
    FooterComponent,
    CardTarifComponent,
    Carrousel
  ],
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})
```

Voici un import de plusieurs classes définis afin de pouvoir réutiliser les différentes pages pour peupler la page HOME

Le design a été réalisé grâce au maquette Figma réalisés au préalable. La charte graphique définie au début du projet a été respectée (polices, couleurs, iconographie).

DOSSIER PROFESSIONNEL (DP)

The screenshot displays the Krav Maga Spirit website's responsive design. On the left, the desktop view shows a group of people in a training session, with a central banner for a free trial class. In the middle, the tablet view shows a similar scene with a different banner. On the right, the mobile view shows a smaller version of the scene with a banner for a free trial class.

Horaires

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

Équipement nécessaire

Pantalon + t-shirt noir
Gants de boxe ou gants ouvert
Coquille (Homme et Femme)
Protection poitrine femmes
Protèges tibia (conseillé)
Chaussures à semelles souples
(utilisées uniquement en intérieur)

Tarifs

| CLASSIQUE | PLE |
|-----------|----------------|
| ETUDE | 290€ / l'année |
| 250€ | 12€/mois |
| 12€/mois | 12€/mois |
| 12€/mois | 12€/mois |

L'affichage est responsive et s'adapte à toutes les tailles d'écran grâce aux media queries implémentés dans les fichiers css nous avons un site web responsive aussi bien sur portable que sur ordinateur.

```
@media (max-width: 1020px){  
    .card{  
        border-radius: 22px;  
        width: 100px;  
        height: 143px;  
        display: flex;  
        flex-direction: column;  
        justify-content: space-between;  
        margin: 3.5vh 0;  
    }  
}
```

Des **tests unitaires** ont été réalisés sur les **composants Angular** afin de garantir la fiabilité et la stabilité du code. Ces tests visaient à valider le bon fonctionnement de chaque composant isolément, en simulant des entrées et en vérifiant les sorties attendues. L'objectif était de détecter rapidement les erreurs lors des modifications et d'assurer la non-régression du comportement des interfaces.

Le **backend** de l'application a été développé en **TypeScript**, en s'appuyant sur une **architecture orientée objet**. Cette approche a permis de structurer le code de manière claire et modulaire. Des **services spécifiques** ont été mis en place pour encapsuler la **logique métier** : gestion de l'authentification avec vérification des identifiants et génération de JWT, manipulation des documents (ajout, suppression, consultation), et administration des équipes (création, attribution des membres, droits d'accès). Cette organisation facilite la maintenance, les tests et l'évolutivité du projet.

```
/**  
 * Méthode appelée pour la création d'un nouvel album.  
 * @param data  
 */  
async createAlbum(data: any): Promise<Response> {  
    try {  
        // On vérifie la nullité des champs.  
        const fieldsToVerify = ['dir_s3', 'nom', 'miniature'];  
        if (!this.utils.checkRequiredFields(data, fieldsToVerify) ||  
            (data == null || !data.listPhotos || data.listPhotos.length < 1)) {  
            return new Response(400, "Les données reçues ne sont pas correctes.");  
        }  
  
        // On convertit les données reçues en DTO.  
        const dateCreation = Date.now(); // Utiliser timestamp number  
        const album = new AlbumDTO(0, data.dir_s3, data.nom, data.miniature, dateCreation);  
        album.description = data.description || null;  
    }  
}
```

La sécurité est assurée via JWT et l'échappement des entrées.

Le code est structuré, commenté et testé avec Prisma. Des tests de sécurité ont été effectués (injection, contournement d'authentification).

Le projet a été géré en méthode agile avec des sprints hebdomadaires.

J'ai participé à la planification des tâches via GitLab Issues et au suivi via un tableau de bord.

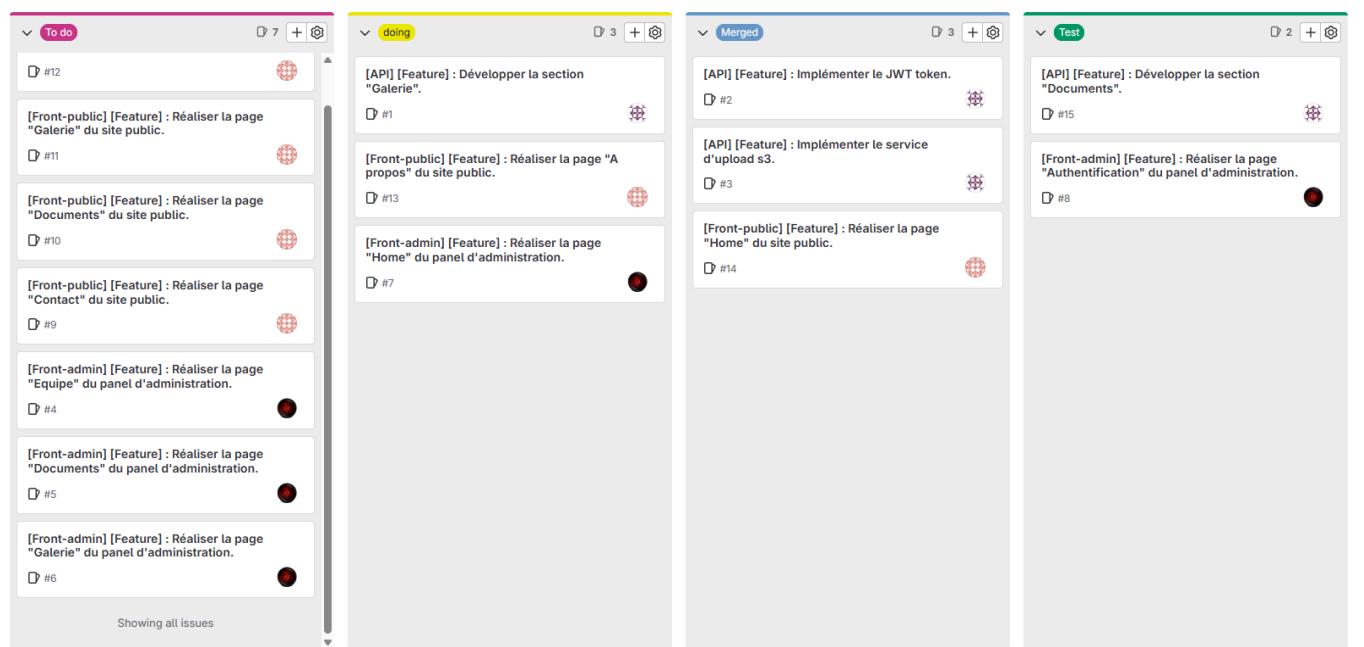
Des points réguliers ont permis d'ajuster les priorités et d'identifier les blocages.

DOSSIER PROFESSIONNEL (DP)

Diagramme de Gantt

| Principal | Secondaires | Janvier | Fevrier | Mars | Avril | Mai | Juin | Juillet |
|---------------|---------------------------|---|---------|-------------------------|---|-----|------|--|
| Conception | Documents Préliminaire | Cahier des charges, Analyse des besoins, Etude de la concurrence | | | | | | |
| | Fonctionnel | Spécifications Fonctionnelles Détailées, Maquettes wireframe et Haute qualité | | | | | | |
| | Modelisation des données | MCD, MLD et MPD, Diagramme de cas d'utilisation, Diagramme de classes, Diagramme de séquence, Diagramme d'activités | | | | | | |
| | Architecture technique | Architecture 'logicielle', Architecture 'Réseau', Documentation technique | | | | | | |
| Développement | Mise en place du projet | | | Mise en place du projet | | | | |
| | Frontend | | | | Développer les fonctionnalites frontend: Mise en page pour plusieurs types d'appareils, creation de modal, creation des pages en accord avec la charte graphique etc... | | | |
| | Backend | | | | Développer les fonctionnalites backend: Authentification, upload et download de fichier, Ajout et suppression d'instructeur, etc... | | | |
| Deploiement | Mise en place de la CI/CD | | | | Mise en place de la CI/CD fonctionnel et fini pour un déploiement propre | | | |
| | Finalisation | | | | | | | Finition du projet pour la présentation du diplome |
| Dossiers | Rédaction des dossiers | | | | | | | Rédaction du dossier de projet et dossier rose. |

L'environnement de travail et les outils (Google Chat, GitLab, Drive) ont été choisis en fonction de la collaboration en équipe.



Voici l'exemple d'un ticket défini dans gitlab

[API] [Feature] : Développer la section Documents.

[Open](#) [Issue created 22 minutes ago by Fabien Ricca](#)

Développer la section Documents sur l'api, ce qui inclut les différents endpoints, le service et le repository pour le bon fonctionnement de cette section.

Documentation nécessaire pour le bon développement de cette feature, disponible sur le drive du projet :

- Section 3 de : [Conception > 01 - Documents Préliminaires > 01. Users stories](#)
- Section 3 de : [Conception > 01 - Documents Préliminaires > 02. Spécifications](#)
- Onglet "Document" de : [Conception > 02 - Modélisation des données > 02. Diagramme de classes -API-.drawio](#)
- Onglet "Admin" de : [Conception > 02 - Modélisation des données > 01. Diagramme de cas d'utilisation.drawio](#)
- Onglet "Documents" de : [Conception > 02 - Modélisation des données > 03. Diagramme d'activités -Admin-.drawio](#)
- Onglets "03. Documents - Ajout" et "03. Documents - Suppression" de : [Conception > 02 - Modélisation des données > 03. Diagramme d'activités -Admin-.drawio](#)

Edited 1 minute ago by Fabien Ricca



[Add design](#) [Create merge request](#)

Child items 0

[Add](#) [⋮](#)

No child items are currently assigned. Use child items to break down work into smaller parts.

Linked items 0

[Add](#) [⋮](#)

Link items together to show that they're related.

Activity

[All activity](#) [Oldest first](#)

- * Fabien Ricca added [To do](#) label 2 minutes ago
- * Fabien Ricca assigned to [@corentin.roussel](#) 2 minutes ago
- * Fabien Ricca changed the description 1 minute ago
- * Fabien Ricca added [doing](#) label and removed [To do](#) label 16 seconds ago

Et aussi une représentation graphique de l'arborescence git

DOSSIER PROFESSIONNEL (DP)

| | | |
|--|----------------------------|------------------|
| [Clean] : - Correction de l'erreur 'script init-pri | ./nettoyage | fabien-ricca |
| create middleware to facilitate jwt use |/feature_ecole/auth... | corentin-roussel |
| add controller logic to register and loggin | | corentin-roussel |
| configure route for login and register | | corentin-roussel |
| add connection to db and function to open collection | | corentin-roussel |
| add model for the login and register request and user object pu | corentin-roussel | |
| file modified or deleted | | corentin-roussel |
| file modified or deleted | | corentin-roussel |
| add utils function for password and jwt generation | | corentin-roussel |
| add go.mod and go.sum to install dependencies necessary | | corentin-roussel |
| [Test] : - Test ci-cd pour suivi projet. | origin & develop | fabien-ricca |
| [Test] : - Test ci-cd pour suivi projet. | | fabien-ricca |
| [Test] : - Test ci-cd pour suivi projet. | | fabien-ricca |
| Merge branch 'clean/professionnalisation_du_projet' into 'develop' | Fabien Ricca | |
| [Clean] : - Déplacement de nginx.conf dans les web. - Mise à j | fabien-ricca | |
| [Clean] : - Mise en place de la creation et de l'utilisation de du | fabien-ricca | |
| [Clean] : - Rajout de la migration prisma. | | fabien-ricca |
| [Clean] : - Suppression des scripts sql utilisés lors de l'initialis | fabien-ricca | |
| [Clean] : - Renpùùage script sql. - Déplacement du Dockerfile- | fabien-ricca | |
| [develop] : - Nettoyage des photos. | | fabien-ricca |
| [develop] : - Test de la ci-cd apres merge. | | fabien-ricca |
| Merge branch 'env/ci-cd' into 'develop' | Fabien Ricca | |
| [ci - cd] : - Test gitlab-ci.yml #11. | | fabien-ricca |

2. Précisez les moyens utilisés :

Dans le cadre du projet **KMS**, plusieurs outils et technologies ont été mobilisés pour garantir la sécurité, la maintenabilité et la fiabilité de l'application tout au long de son développement.

- Le **back-end** de l'application a été développé avec **Node.js et Express**, offrant un environnement stable et performant.
- Le **front-end** a été conçu avec **Angular**, qui intègre des mécanismes de protection contre les failles courantes comme le XSS.
- Comme le front et le back s'appuie sur le même langage socle (JavaScript) cela simplifie une mise en œuvre de type Full Stack.
- L'ensemble des services de l'application a été **conteneurisé avec Docker**, permettant un isolement clair des environnements et une meilleure maîtrise des dépendances.
- **Docker Compose** a été utilisé pour orchestrer les différents conteneurs, notamment la base de données et le serveur applicatif.
- Le **code source** a été versionné via **Git**, avec un hébergement sur **GitLab**, ce qui a permis la mise en place de pipelines CI/CD intégrant des outils de vérification de code.
- **Traefik** a été utilisé comme reverse proxy pour gérer les routes dynamiques et les certificats SSL, garantissant un accès sécurisé à l'application.
- Enfin, l'environnement de développement a été configuré avec **NeoVim**, permettant une personnalisation avancée de l'éditeur, avec des plugins pour le linting, la vérification de type et le formatage automatique.

Tous ces outils ont contribué à renforcer la sécurité de l'application, tant au niveau du code que de son exécution, tout en assurant une collaboration fluide entre les membres de l'équipe.

3. Avec qui avez-vous travaillé ?

Fabien Ricca, Léa Dubois

4. Contexte

Nom de l'entreprise, organisme ou association ➔ *La Plateforme.*

Chantier, atelier, service ➔ *Projets d'écoles.*

Période d'exercice ➔ Du : *06/01/2025* au : *01/08/2025*

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

Activité-type 1 Développer une application sécurisé

Exemple n° 1 - Smart Locker

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

L'environnement de travail a été configuré dès le lancement du projet, en fonction des besoins techniques, des profils dans l'équipe et des outils collaboratifs choisis:

- Le développement Android a été réalisé avec Android Studio, configuré individuellement selon les postes de travail.
- La gestion de projet a été centralisée sur Jira, avec le suivi de tâches, les user stories et les sprints.
- La documentation technique et fonctionnelle a été tenue à jour sur Confluence, avec l'historique des décisions et livrables.
- Les réunions et brainstormings ont été facilités grâce à Mural et Lucidchart (maquettes, diagrammes d'architecture).
- Le travail collaboratif s'est appuyé sur Confluence pour l'échange des documents.
- Figma a servi à la création des maquettes UI, partagées avec l'équipe et modifiables en temps réel.
- ChatGPT a été utilisé ponctuellement pour de l'assistance technique ou pour suggérer des optimisations de code.
- Pour les tests de sécurité sur la base de données, nous avons utilisé sqlmap, outil d'analyse automatique des injections SQL.

Enfin, plusieurs **technologies et bibliothèques** ont été intégrées dans les environnements locaux de chaque développeur :

- **Vue.js** pour le développement de l'interface utilisateur, **Vuex** pour la gestion d'état, **NestJS** comme framework backend structuré, **Axios** pour les appels HTTP côté client, et **Leaflet** pour l'affichage des cartes interactives.

L'ensemble de ces outils a été documenté et partagé avec l'équipe pour garantir la reproductibilité de l'environnement.

L'interface utilisateur de l'application a été conçue en suivant les maquettes réalisées sur Figma. Ces maquettes ont été validées en amont avec l'équipe projet, puis traduites en vues XML. Les interfaces ont été pensées pour être intuitives, responsives (compatibles mobiles/tablettes) et

accessibles.

Des composants réutilisables ont été développés pour afficher les listes de contenus, les fiches détaillées et les actions utilisateurs.

Les animations et transitions ont été optimisées pour améliorer l'expérience utilisateur.

Des tests manuels ont été réalisés sur différents formats d'écran pour vérifier le bon affichage.

Les composants métier de l'application Smart Locker ont été développés avec NestJS côté serveur, en respectant les principes de la programmation orientée objet.

La logique métier couvrait notamment la gestion des casiers (disponibilité, statut, localisation), le système de réservation (création, expiration automatique, annulation), ainsi que le lien entre les utilisateurs, les casiers et leur emplacement sur une carte du campus.

Des services ont été conçus pour assurer la cohérence métier : vérification des créneaux disponibles, attribution exclusive, limitation d'accès selon les rôles (utilisateur/admin).

Les entrées utilisateur ont été systématiquement validées côté serveur pour éviter les incohérences et sécuriser les données.

Le code a été commenté, versionné.

```
async create(createLockerDto: CreateLockerDto) {
    // Mise en place d'un querybuilder pour vérifier si le group locker existe
    const result = await this.LockeRepository.createQueryBuilder('locker')
        .innerJoin('group_locker', 'grp_locker', 'grp_locker.id = locker.id_group_locker')
        .where('grp_locker.id=:id', { id: createLockerDto.id_group_locker })
        .getExists();
    if(result){
        return this.LockeRepository.insert(createLockerDto);
    }
    else{
        throw new BadRequestException('Group locker not found');
    }
}
```

DOSSIER PROFESSIONNEL (DP)

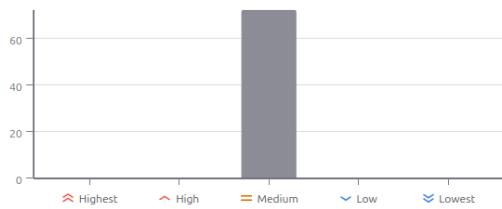
Vue d'ensemble de l'état

Obtenez un instantané de l'état de vos tickets. [Afficher tous les tickets](#)



Aucune activité pour le moment

Créez quelques tickets et invitez des coéquipiers à rejoindre votre projet pour en suivre l'activité.



Activité récente

Tenez-vous au courant de ce qui se passe tout au long du projet.

mardi 22 juillet 2023

Guangquan YE a modifié la personne assignée en Guangquan YE pour le ticket: [OVER-94: Front - Web - Réservations - Réserveations create](#) REVUE EN COURS

il y a 7 jours

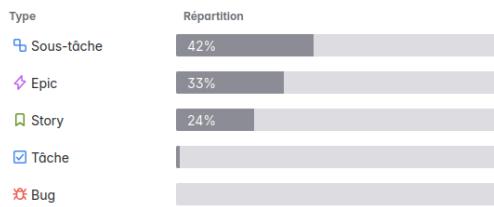
Guangquan YE a mis à jour « status » sur [OVER-94: Front - Web - Réserveations - Réserveations create](#) REVUE EN COURS

il y a 7 jours

Guangquan YE a supprimé la personne assignée de [OVER-94: Front](#)

Types de ticket

Obtenez une répartition des tickets par type. [Voir tous les tickets](#)



La gestion du projet a été assurée via l'outil Jira, où toutes les tâches ont été planifiées, suivies et réparties entre les membres de l'équipe.

Des Epics ont été créés pour structurer les grandes fonctionnalités (réservation, gestion des casiers, cartographie, authentification, etc.), puis déclinés en user stories et sous-tâches selon la méthode agile.

Chaque membre disposait de ses tickets, ce qui permettait un suivi précis de l'avancement via le tableau Kanban.

Les délais ont été définis en fonction des sprints hebdomadaires, et les éventuels retards ou blocages étaient remontés durant les réunions d'équipe.

J'ai participé activement à la mise à jour de l'état des tâches, à l'évaluation des charges et à la priorisation selon les dépendances techniques.

L'environnement de développement (NestJS, Vue.js, PostgreSQL) était aligné avec l'architecture logicielle prévue, et les outils collaboratifs (Confluence, Figma, Google Chat) ont soutenu la coordination entre les membres.

2. Précisez les moyens utilisés :

Pour développer l'application Smart Locker, l'environnement de travail a été configuré en local sur chaque poste avec une stack moderne :

- Frontend: Vue.js, installé via Node.js et géré avec Vue CLI.
- Backend: NestJS, framework modulaire basé sur Node.js et TypeScript, installé avec npm.
- Base de données: PostgreSQL, installée localement et dans des conteneurs Docker pour les environnements isolés.

L'ensemble du projet est versionné avec Git et hébergé sur Git Hub.
Le ticketing et le suivi des tâches ont été assurés avec Jira, et la documentation projet centralisée sur Confluence.
Les maquettes fonctionnelles ont été créées et partagées sur Figma pour assurer la cohérence graphique et ergonomique.
Chaque développeur disposait d'un environnement complet.

3. Avec qui avez-vous travaillé ?

Fabien Ricca, Léa Dubois

4. Contexte

Nom de l'entreprise, organisme ou association ▶ **La Plateforme**

Chantier, atelier, service ▶ **Projets d'écoles.**

Période d'exercice ▶ Du : **01/07/2024** au : **26/07/2024**

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 :

1 Analyser les besoins et maquetter une application

J'ai participé à l'analyse du cahier des charges pour identifier les fonctionnalités clés :

- Gérer les équipes
- Gérer la galerie
- Gérer les documents.

Les fonctionnalités cité au dessus ont était établi au préalable avec l'aide du client et sont non flexibles pour la mise en production du projet

Cette mis en forme du cahier des charges nous a permis de dérouler et de préciser plusieurs aspects de la conception du projet telle que:

- Spécifications fonctionnelles: Décrit **en détail les fonctionnalités** attendues sans entrer dans la technique.
- User Stories: Petits récits exprimant un besoin utilisateur (« En tant que..., je veux..., afin de... »)
- Diagrammes UML: **Schémas de modélisation** du système (cas d'usage, classes, etc.) pour mieux comprendre et concevoir.
- Maquettes(Wireframe): **Esquisses de l'interface** utilisateur pour valider l'ergonomie et l'apparence.
- Conception de la base de données: **MCD** : Modèle métier avec **entités, attributs et relations**, **MLD** : Version structurée avec **clés primaires et étrangères**, **MPD** : Version technique avec **tables, types, contraintes**, prête à être implantée.

Des maquettes Figma ont été produites, respectant les exigences utilisateur, avec un parcours fluide. Un schéma d'enchaînement des écrans a été réalisé pour représenter la navigation. L'ensemble a servi de base pour la conception technique.

DOSSIER D'INSCRIPTION

[Demande de licence FFKDA](#)

[Fiche d'inscription KMS](#)

PROGRAMME TECHNIQUE des ceintures FFKDA krav maga :

[Ceinture Jaune](#)

[Ceinture Orange](#)

[Ceinture Verte](#)

[Ceinture Bleu](#)

[Ceinture Marron](#)

[1er DAN \(version 2018\)](#)

DOSSIER D'INSCRIPTION

[Demande de licence FFKDA](#)

[Fiche d'inscription KMS](#)

PROGRAMME TECHNIQUE des ceintures FFKDA krav maga :

[Ceinture Jaune](#)

[Ceinture Orange](#)

[Ceinture Verte](#)

[Ceinture Bleu](#)

[Ceinture Marron](#)

[Ceinture Bleu](#)



KRAV MAGA SPIRIT

Nous suivre :



TEXT

TEXT TEXT TEXT
TEXT TEXT TEXT
TEXT TEXT TEXT
TEXT TEXT TEXT

TEXT

TEXT TEXT TEXT
TEXT TEXT TEXT
TEXT TEXT TEXT
TEXT TEXT TEXT

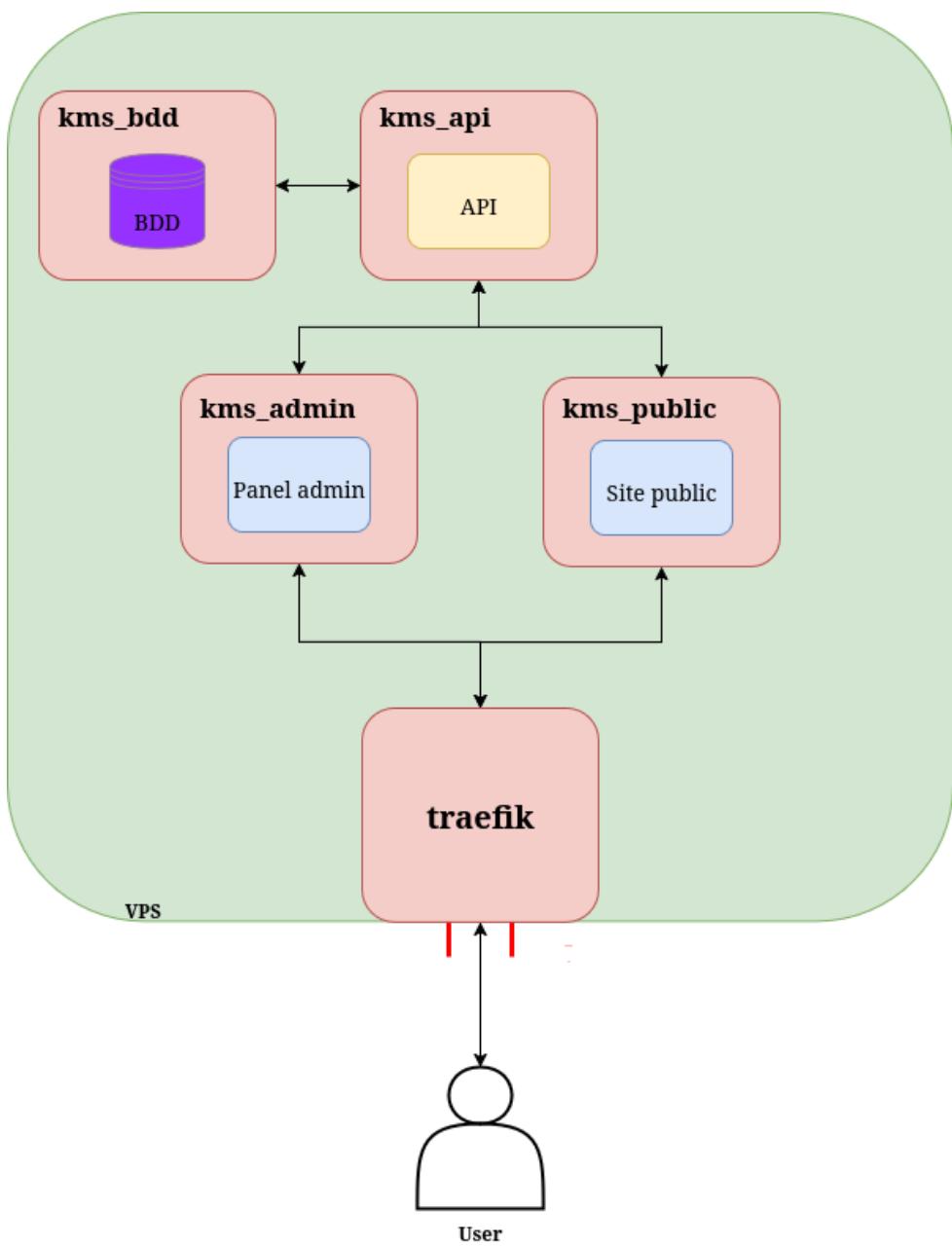
© Krav Maga Spirit
Dojo Maison Pour Tous • 70 avenue Zoratti • 13008 Marseille

Mentions légales • Crée par Krav Maga Spirit

DOSSIER PROFESSIONNEL (DP)

The screenshot shows the Krav Maga Spirit website with a navigation bar at the top. Below the navigation, there are two main sections: "DOSSIER D'INSCRIPTION" on the left and "PROGRAMME TECHNIQUE" on the right. The "DOSSIER D'INSCRIPTION" section contains links for "Demande de licence FFKDA" and "Fiche d'inscription KMS". The "PROGRAMME TECHNIQUE" section lists various belt colors: Ceinture Jaune, Ceinture Orange, Ceinture Verte, Ceinture Bleue, Ceinture Marron, and 1er DAN (version 2018). To the right of these lists is another "PROGRAMME TECHNIQUE" section with the same color-coded boxes. At the bottom of the page is a footer with social media links (Facebook) and address information.

L'architecture est de type multicouche répartie : Angular en frontend, Express basé sur Node.js et go en backend, PostgreSQL, MongoDB en base de données.
Chaque couche est isolée et a un rôle défini. La sécurité est présente à tous les niveaux (JWT, validations).



La base de données relationnelle PostgreSQL a été modélisée à partir du MCD puis implémentée via Prisma ORM.

Les règles de nommage ont été respectées et l'intégrité référentielle assurée.

Les données sensibles sont protégées.

Une procédure de restauration a été documentée.

DOSSIER PROFESSIONNEL (DP)

J'ai implémenté des accès aux données via Prisma ORM. Les traitements couvrent la création, lecture, modification, suppression (CRUD).

```
/**  
 * Méthode appelée pour la création d'un nouvel album.  
 * @param album  
 */  
async createAlbum(album: AlbumDTO) {  
  
/**  
 * Méthode appelée pour récupérer la liste de tous les albums existants.  
 */  
async getAlbums() {  
  
/**  
 * Méthode appelée pour la modification d'un album.  
 * @param album  
 */  
async updateAlbum(album: AlbumDTO): Promise<boolean> {  
  
/**  
 * Méthode appelée pour la suppression d'un album.  
 * @param id  
 */  
async deleteAlbum(id: number) {
```

Les exceptions sont gérées (erreurs de validation, droits insuffisants, incohérences).

```
async getAlbumById(id: number) {  
try {  
  const album = await this.prisma.album.findUnique({  
    where: { id: id },  
    include: {  
      photos: {  
        orderBy: { id: 'asc' }  
      }  
    }  
  });  
  
  if (album) {  
    console.log(`[INFO] ${this.currentDate()} --> Récupération de l'album n°${id}.`);  
  }  
  return album;  
} catch (error) {  
  console.log(`[ERROR] ${this.currentDate()} --> Une erreur a été rencontrée lors de la récupération de l'album n°${id} :\\n ${error}`);  
  throw error;  
}
```

Nous utilisons des blocs **try/catch** pour **intercepter les erreurs potentielles** lors de l'exécution du code. Cela permet de **journaliser les erreurs** (logs) pour faciliter le diagnostic, puis de **envoyer une réponse d'erreur adaptée** à l'utilisateur ou à l'appelant de l'API, tout en évitant de divulguer des informations sensibles.

Les entrées sont validées côté backend, les requêtes sécurisées.

```
/**  
 * Méthode appelée pour ajouter des photos à un album.  
 * @param id_album  
 * @param listPhotos  
 */  
async addPhotosToAlbum(id_album: number, listPhotos: string[]): Promise<void> {  
    try {  
        // On convertit la liste de photos en DTO et on l'ajoute en base.  
        for (const photo of listPhotos) {  
            const photoDTO = new PhotoDTO(0, id_album, photo);  
            await this.model.addPhoto(photoDTO);  
        }  
    } catch (error) {  
        console.error("Erreur lors de l'ajout de photos:", error);  
        throw error;  
    }  
}
```

Utilisations de l'instantiation de la classe model pour pouvoir insérer un photos dans un album à l'aide de prisma

```
/*  
 * Méthode appelée pour ajouter des photos en base.  
 * @param photo  
 */  
async addPhoto(photo: PhotoDTO) {  
    try {  
        const result = await this.prisma.photo.create({  
            data: {  
                idAlbum: photo.id_album,  
                photoUrl: photo.photoUrl  
            }  
        );  
  
        console.log(`[INFO][${this.currentDate()}] --> Ajout de la photo ${photo.idAlbum}`);  
        return result;  
    } catch (error) {  
        console.log(`[ERROR][${this.currentDate()}] --> Utilisation de la méthode addPhoto : ${error}`);  
        console.log(error);  
        throw error;  
    }  
}
```

Nous utilisons des **DTO (Data Transfer Objects)** afin de **simplifier et sécuriser le transfert de données** entre les différentes couches de l'application. Cela permet de **contrôler précisément les champs**

DOSSIER PROFESSIONNEL (DP)

envoyés ou reçus, en limitant les données aux seuls attributs nécessaires, ce qui renforce la sécurité et améliore la lisibilité du code.

Côté base de données, une **base MongoDB** a été utilisée dans le cadre du **projet scolaire**. Grâce à sa structure flexible et orientée document, MongoDB s'est avéré particulièrement adapté pour le prototypage rapide, permettant d'enregistrer des données sans schéma rigide. Cette approche a facilité les ajustements fréquents durant les phases de développement et de test.

Cependant, cette base de données n'a été utilisée **que dans le cadre de la version développée pour le projet d'école**. Elle n'est pas déployée en production, où une solution plus robuste ou relationnelle pourrait être privilégiée en fonction des besoins en performance, sécurité ou structure de données.

```
func DbInstance() *mongo.Client {
    err := godotenv.Load(".env")
    if err != nil {
        log.Fatal("Error loading .env file")
    }

   MongoDb := os.Getenv("MONGODB_URL")

    serverApi := options.ServerAPI(options.ServerAPIVersion1)
    opts := options.Client().ApplyURI(MongoDb).SetServerAPIOptions(serverApi)

    ctx, cancel := context.WithTimeout(context.Background(), 10*time.Second)
    defer cancel()

    client, err := mongo.Connect(ctx, opts)
    if err != nil {
        log.Fatal(err)
    }

    fmt.Println("Connected to MongoDB")
    return client
}
```

Pour l'enregistrement et l'authentification des utilisateurs, nous utilisons les **JSON Web Tokens (JWT)** afin de permettre aux utilisateurs connectés d'accéder aux API **sans devoir renvoyer leur identifiant et mot de passe à chaque requête**.

Les **mots de passe** sont **hachés avec Bcrypt**, ce qui permet de les **obfuscuer** et de les protéger contre les attaques, en rendant leur déchiffrement beaucoup plus complexe et long pour un acteur malveillant.

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.

Enfin, nous avons 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.

2. Précisez les moyens utilisés :

L'environnement de développement a été configuré autour d'une stack moderne et adaptée aux besoins du projet. Le code a été édité principalement avec NeoVim, personnalisé pour supporter JavaScript, TypeScript, Prisma et les autres technologies utilisées.

- Frontend: développé avec Angular.
- Backend: construit en Express avec un design modulaire.
- Base de données: PostgreSQL pour les données relationnelles et MongoDB pour les données non structurées.
- ORM: Prisma ORM pour simplifier les interactions entre le backend et les deux bases de données.

La gestion des modèles de données et des relations a été formalisée via Prisma Schema, assurant une documentation claire et une migration fluide.

Les maquettes UI ont été conçues avec Figma pour garantir une interface utilisateur cohérente avec l'expérience attendue.

Les diagrammes d'architecture et schémas techniques ont été réalisés avec Draw.io, facilitant la compréhension des flux de données et des modules de l'application.

La documentation technique, rédigée sur Google Docs, centralise les informations essentielles du projet (installation, fonctionnement, API, modèles de données). .

3. Avec qui avez-vous travaillé ?

Fabien Ricca, Léa Dubois

4. Contexte

Nom de l'entreprise, organisme ou association ▶ **La Plateforme**

Chantier, atelier, service ▶ **Projets d'écoles.**

Période d'exercice ▶ Du : **06/01/2025** au : **01/08/2025**

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

Activité-type 2 Concevoir et développer une application sécurisée organisée en couches

Exemple n° 1 - Smart Locker

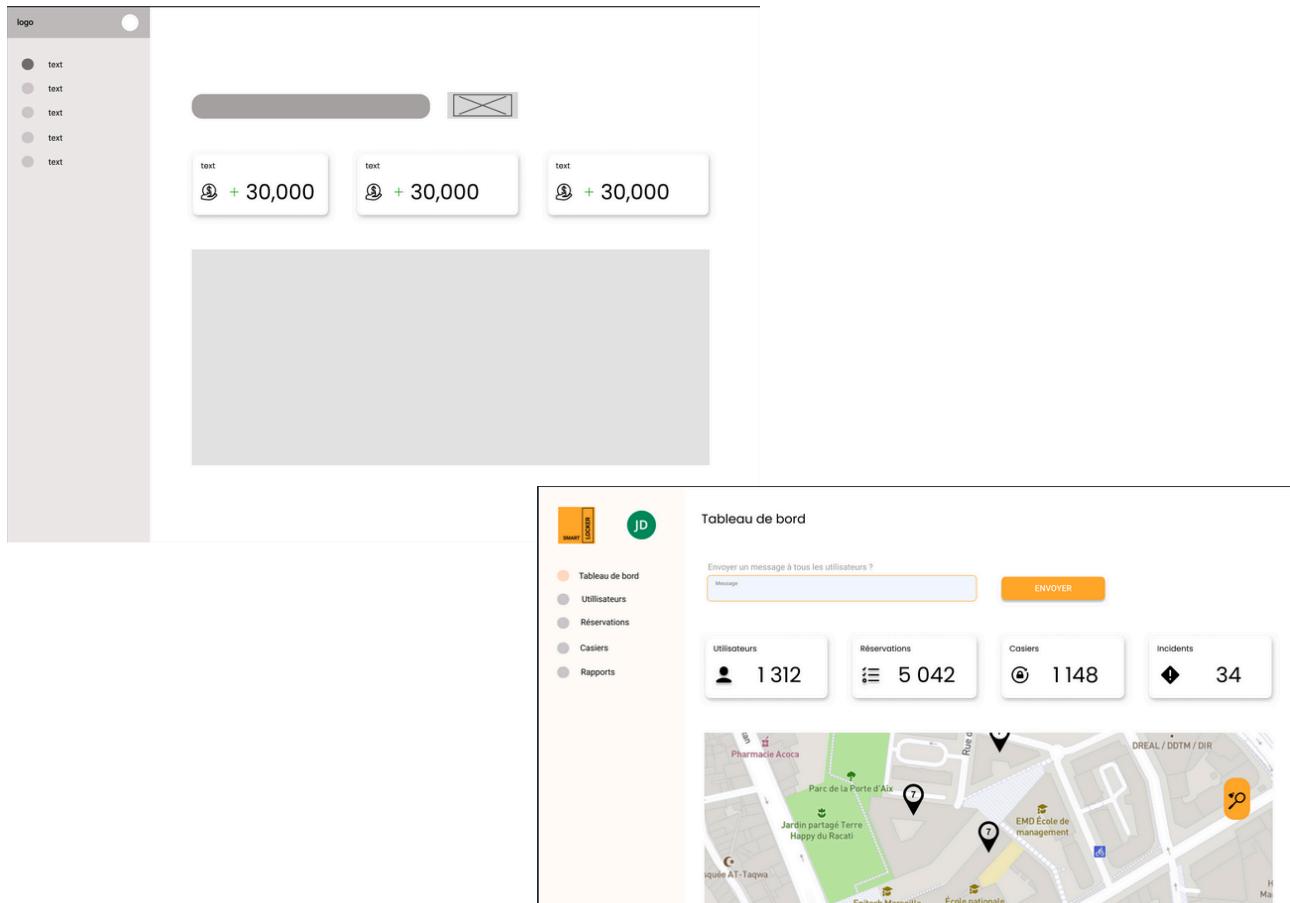
1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

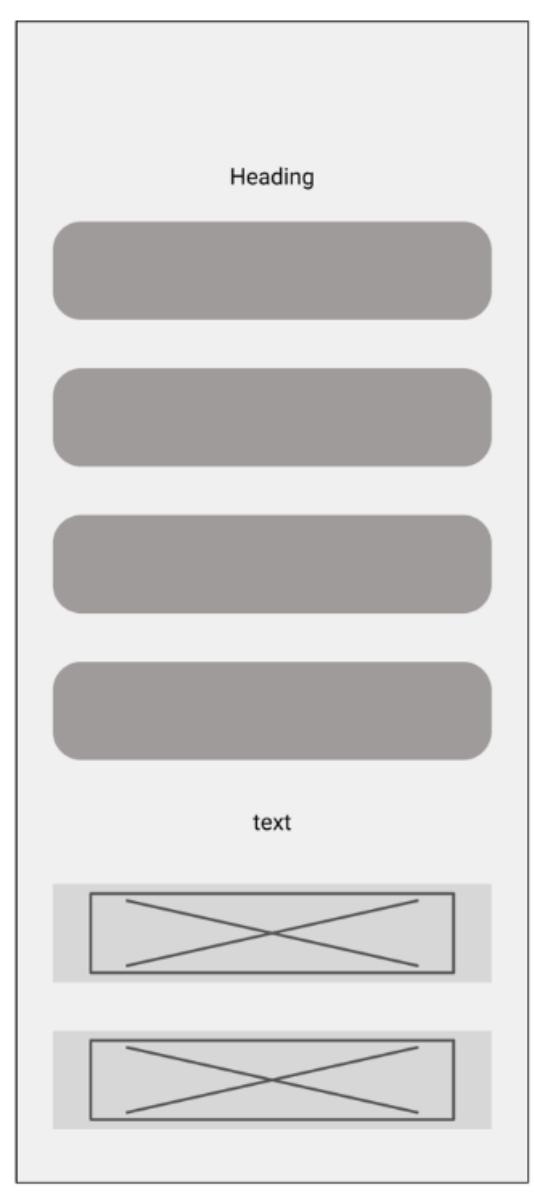
L'analyse des besoins a été menée à partir d'un cahier des charges précisant les attentes des utilisateurs. L'objectif était de concevoir une application de gestion de casiers connectés (Smart Locker) sur un campus.

Les fonctionnalités principales identifiées sont la visualisation des casiers sur une carte, la réservation à distance, et la gestion sécurisée des accès.

Des **maquettes interactives** ont été réalisées avec **Figma**, présentant les principaux écrans : carte interactive, fiche de réservation, tableau de bord utilisateur.

L'enchaînement des vues a été représenté à l'aide d'un **schéma de navigation**. Ces éléments ont permis de valider l'expérience utilisateur et de guider le développement de l'interface.





Créer un compte

Nom
Doe

Prénom
Jane

Email
smart@locker.fr

Mot de passe
fJtughy3#4\$2@1! ✓

Confirmation mot de passe
fJtughy3#

INSCRIPTION

Déjà inscrit ?

DOSSIER PROFESSIONNEL (DP)

Une **charte graphique** a été définie dès les premières phases du projet afin de garantir une cohérence visuelle sur l'ensemble des interfaces de l'application. Elle précise les **codes couleurs**, les **polices typographiques**, les **marges**, ainsi que les **tailles et styles d'éléments UI** (boutons, icônes, menus, formulaires, etc.).

LOGO



icone



COULEUR



Typographie

Principal - ROBOTO

Titres (H1) - 24px
Sous-titres (H2) - 20px
Sous-sous-titres (H3) - 18px
Corps de Texte (P) - 14px
Légendes et Labels - 12px
Boutons et Interactions - 16px

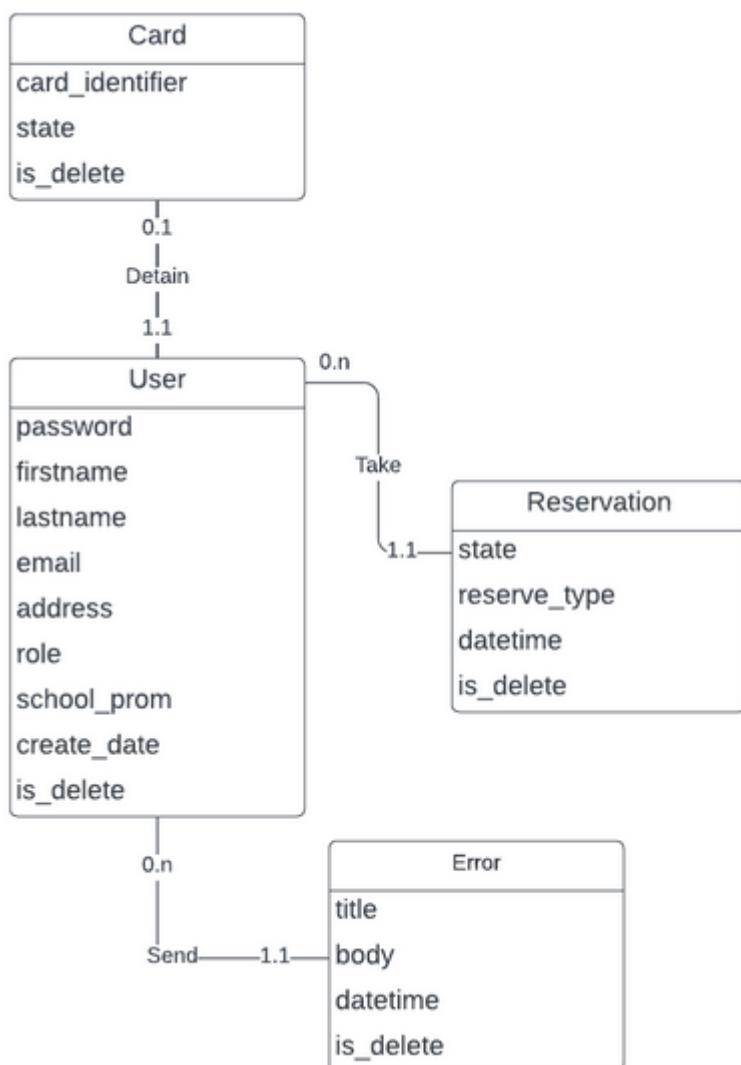
Boutons



L'architecture choisie repose sur un modèle **multicouche sécurisé**, séparant clairement les responsabilités entre l'interface (Vue.js), la logique métier (NestJS) et les données (PostgreSQL via TypeORM).

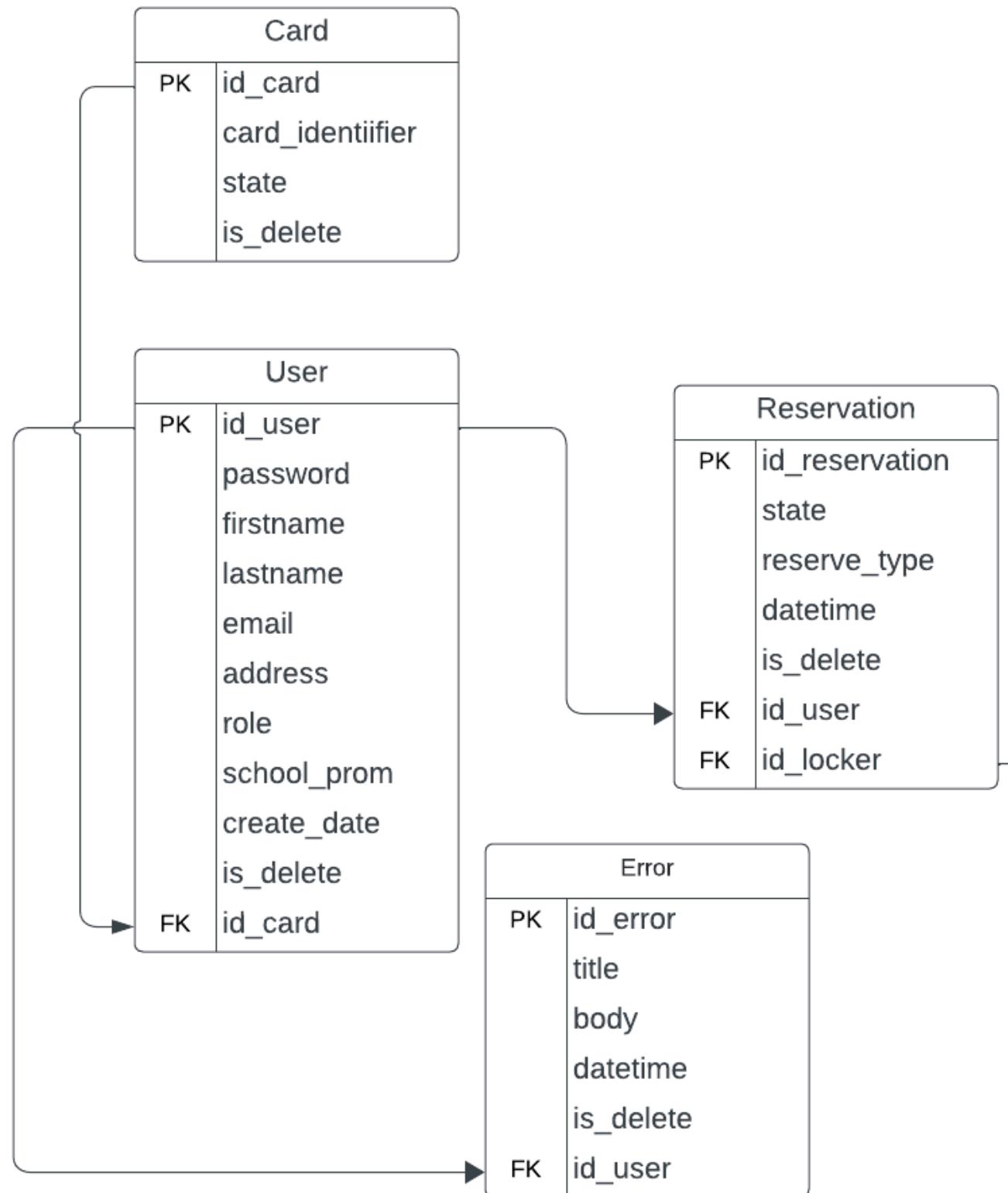
Une base de données relationnelle **PostgreSQL** a été modélisée pour répondre aux besoins fonctionnels du projet, selon les bonnes pratiques du relationnel : normalisation des tables, relations claires, types stricts.

MCD:

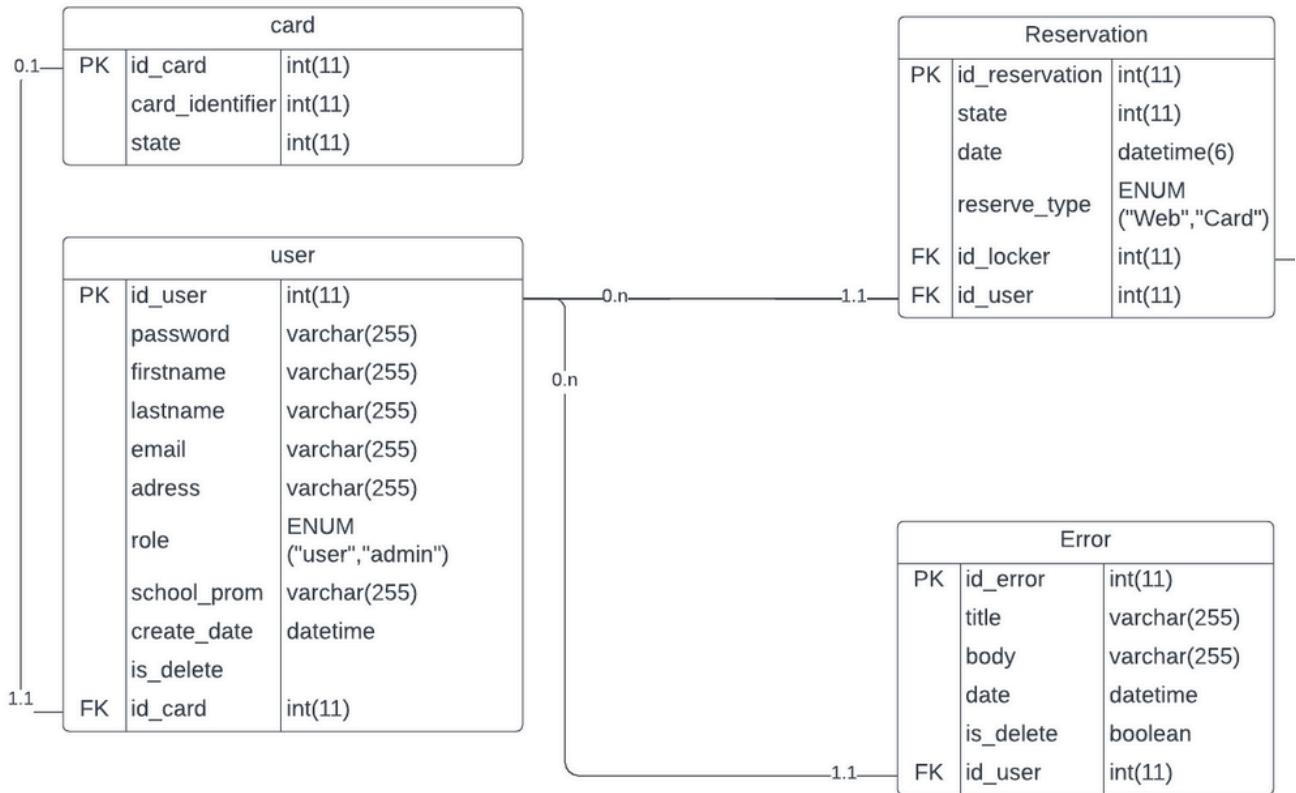


DOSSIER PROFESSIONNEL (DP)

MLD:



MPD:



Les règles de nommage sont homogènes et documentées.

La sécurité est assurée par une gestion fine des droits, des contraintes d'intégrité.

Un **diagramme de classes UML** a été réalisé afin de modéliser de manière claire et structurée les entités principales du projet ainsi que leurs **attributs** (propriétés) et **comportements** (méthodes). Ce diagramme a permis de visualiser l'architecture orientée objet du backend, de repérer les **liens d'héritage**, les **associations** entre les classes (ex. un utilisateur possède plusieurs documents), ainsi que de clarifier les responsabilités de chaque composant.

DOSSIER PROFESSIONNEL (DP)

| User |
|--|
| - id:int - password:string - firstname:string - lastname:string - email:string - address:string - role:string - age:int - school_prom:string - create_date:datetime - id_card:int - is_delete:boolean |
| + findUser(id):return <User> + findUsers():return List<User> + login(params):void + register(params):void + createUser(params):void + updateUser(id_user, params):void + deleteUser(id_user):void |

La documentation de la base et de ses structures a été formalisée en français et intégrée à la documentation technique du projet.

2. Précisez les moyens utilisés :

Pour mener à bien la conception technique du projet **SmartLocker**, plusieurs outils et technologies ont été mobilisés :

- **Maquettage** : Les interfaces utilisateurs ont été conçues à l'aide de **Figma**, outil collaboratif permettant de créer des maquettes interactives et responsives, partagées avec l'ensemble de l'équipe.
- **Schémas fonctionnels et techniques** : Des diagrammes d'architecture, de flux et d'enchaînement des écrans ont été réalisés avec **Draw.io**, afin de formaliser visuellement la structure de l'application.
- **Éditeur de code** : Le développement s'est appuyé sur **NeoVim**, un éditeur léger et personnalisable, configuré pour améliorer la productivité des développeurs.
- **Base de données** : Le projet repose sur une base **PostgreSQL** pour le stockage des données relationnelles.
- **Modélisation et accès aux données** : **TypeORM** a permis de modéliser le schéma des bases de données et de faciliter les opérations de lecture/écriture entre les services et les sources de données.

- **Documentation** : La documentation de conception et les spécifications fonctionnelles ont été rédigées via **Confluence**, partagées dans l'équipe via Confluence pour une centralisation claire des livrables.

Ces moyens ont permis de structurer efficacement la phase de conception, d'assurer une traçabilité des choix techniques et de garantir la cohérence entre les besoins fonctionnels et les solutions mises en œuvre.

3. Avec qui avez-vous travaillé ?

Fabien Ricca, Léa Dubois

4. Contexte

Nom de l'entreprise, organisme ou association ➤ **La Plateforme**

Chantier, atelier, service ➤ **Projets d'écoles.**

Période d'exercice ➤ Du : **01/07/2024** au : **26/07/2024**

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

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 :

Des **tests unitaires** et des **tests d'intégration** ont été mis en place pour vérifier le bon fonctionnement de l'application, notamment pour les fonctionnalités clés telles que l'**authentification**, la gestion des **documents** et de la **galerie**. Ces tests ont été développés à l'aide du framework **Jest**, choisis pour sa rapidité et sa simplicité d'utilisation.

Les **tests unitaires** ont permis de s'assurer que chaque fonction ou service se comportait correctement de manière isolée. Les **tests d'intégration**, quant à eux, ont validé la bonne interaction entre plusieurs composants du système, garantissant une cohérence globale du fonctionnement.

Les **tests d'intégration** Vérifie que plusieurs composants fonctionnent ensemble correctement en testant leurs interactions.

```
test('devrait valider la force des mots de passe', async () => {
  // S'assurer que le service est initialisé
  expect(authService).toBeDefined();

  const passwordTests = [
    { password: 'weak', expected: false, description: 'Mot de passe faible (trop court, pas de maj, chiffre, symbole)' },
    { password: 'StrongPassword123!', expected: true, description: 'Mot de passe fort (tous critères)' },
    { password: 'password', expected: false, description: 'Mot de passe commun (pas de maj, chiffre, symbole)' },
    { password: 'PASSWORD123!', expected: false, description: 'Pas de minuscule' },
    { password: 'StrongPass!', expected: false, description: 'Sans chiffres' },
    { password: 'StrongPass1', expected: false, description: 'Sans caractère spécial' },
    { password: 'strongpass123!', expected: false, description: 'Sans majuscule' },
    { password: 'STRONGPASS123!', expected: false, description: 'Sans minuscule' },
    { password: TEST_OPTIONS.TEST_USER.password, expected: true, description: 'Mot de passe de test (TestPassword123!)' }
  ];
});
```

Les **tests unitaires** vérifient qu'une fonction ou un composant isolé du code se comporte correctement, indépendamment du reste du système..

```

describe('connect', () => {
  test('devrait se connecter avec succès avec des credentials valides', async () => {
    // Arrange
    const mockData = {
      login: 'testuser',
      password: 'plainPassword'
    };

    const mockUser = {
      id: 1,
      login: 'testuser',
      password: 'hashedPassword'
    };

    mockAuthenticateModel.userExists.mockResolvedValue(true);
    mockAuthenticateModel.getUser.mockResolvedValue(mockUser);
    mockBcryptCompare.mockResolvedValue(true as never);

    // Act
    const result = await authenticateService.connect(mockData);

    // Assert
    expect(result).toBeInstanceOf(Response);
    expect(result).toHaveProperty('code', 200);
    expect(result).toHaveProperty('message', 'Connexion réussie');
    expect(mockAuthenticateModel.userExists).toHaveBeenCalledWith('testuser');
    expect(mockAuthenticateModel.getUser).toHaveBeenCalledWith(expect.any(CredentialsDTO));
    expect(mockBcryptCompare).toHaveBeenCalledWith('plainPassword', 'hashedPassword');
  });
});

```

1. Build Angular : Compilation de l'application frontend en fichiers statiques optimisés pour la production (minification, bundling).

2. Lancement des conteneurs Docker : Démarrage de l'infrastructure containerisée avec tous les services (frontend, backend, base de données) via Docker Compose ou Kubernetes.

3. Configuration du reverse proxy : Mise en place du routage des requêtes, gestion SSL et distribution du trafic vers les différents conteneurs via Nginx ou similaire.

DOSSIER PROFESSIONNEL (DP)

Installation

1. Cloner le repository
2. Installer les dépendances

```
# API  
cd api/  
npm install  
npx prisma generate  
cd ..  
  
# Frontend public  
cd web-public/  
npm install  
cd ..  
  
# Frontend admin  
cd web-admin/  
npm install  
cd ..
```

3. Configuration des variables d'environnement

```
# Copier le modèle de configuration  
cp api/.env-modèle api/.env
```

Pour remplir les variables d'environnement, demander les valeurs au coordinateur du projet.

Des scripts ont été écrits pour automatiser ces étapes.

Mode Docker complet

```
# Démarrer tous les services en production  
docker compose up -d  
  
# Arrêter les services  
docker compose down
```

Base de données

```
# Créer un dump de sauvegarde  
cd api/prisma/scripts/  
./create-dump.sh  
  
# Accéder à la base de données  
# Utiliser un gestionnaire de base de données comme DBeaver, les informations de connexion sont à dé
```

Des tests d'intégration, système et validation client ont été effectués avant mise en production.

GitLab CI/CD a été configuré pour automatiser l'intégration continue et les déploiements.
À chaque push et merge/rebase sur la branche develop les conteneurs sont build automatiquement,
déployer sur le vps et il redémarre tout les services dans les containers

Le déploiement se fait automatiquement via GitLab CI sur la branche `develop` :

1. Push et merge/rebase sur `develop`
2. Build automatique des images Docker
3. Déploiement sur le VPS
4. Redémarrage des services dans leurs containers respectifs

Déploiement manuel

```
# Sur le serveur de production
cd /root/projects/kms/
git pull
docker compose down
docker compose up -d --build
```

Pour garantir une qualité de code homogène dans toute l'équipe, les outils **ESLint** et **Husky** ont été configurés. **ESLint** permet de détecter automatiquement les erreurs de syntaxe, les oubli de ponctuation ou les mauvaises pratiques, tandis que **Husky** déclenche ces vérifications automatiquement lors des pré-commits Git, assurant un code propre et cohérent entre tous les membres du projet.

2. Précisez les moyens utilisés :

Pour assurer la qualité logicielle et le bon fonctionnement de l'application **KMS**, plusieurs moyens ont été mis en place à différentes étapes du processus de test et de déploiement.

- Des **tests unitaires** et **tests d'intégration** ont été développés à l'aide du framework **Jest**, permettant de valider individuellement les composants ainsi que leur interaction.
- Les **tests d'intégration**, **tests système** et une **validation utilisateur** ont été menés avant mise en production, afin de garantir le respect des exigences initiales.
- Une **procédure de déploiement** a été rédigée, incluant les étapes suivantes : **build de l'application Angular**, **lancement des conteneurs Docker**, et **configuration du reverse proxy** via **Traefik**. Des **scripts automatisés** ont été écrits pour enchaîner ces étapes sans intervention manuelle.
- Le processus d'**intégration continue et déploiement continu (CI/CD)** a été automatisé via **GitLab CI/CD**. À chaque push ou fusion sur la branche `develop`, les conteneurs sont automatiquement build, déployés sur le **VPS**, et les services sont redémarrés dans leurs conteneurs respectifs.
- Enfin, des outils de contrôle de qualité comme **ESLint** et **Husky** ont été utilisés pour standardiser le code au moment des pré-commits. Ils assurent un formatage homogène (indentation, ponctuation, etc.) et réduisent les erreurs courantes dans l'équipe de développement.

3. Avec qui avez-vous travaillé ?

Fabien Ricca, Léa Dubois

DOSSIER PROFESSIONNEL (DP)

4. Contexte

Nom de l'entreprise, organisme ou association ▶ **La Plateforme**

Chantier, atelier, service ▶ **Projets d'écoles.**

Période d'exercice ▶ Du : **06/01/2025** au : **01/08/2025**

5. Informations complémentaires (facultatif)

Titres, diplômes, CQP, attestations de formation

| 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) **ROUSSEL Corentin**,

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 **01 /08 / 2025**

pour faire valoir ce que de droit.

Signature :

ROUSSEL

Corentin

DOSSIER PROFESSIONNEL ^(DP)

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé

Cliquez ici pour taper du texte.

ANNEXES

(Si le RC le prévoit)