



# DOSSIER PROFESSIONNEL (DP)

<i>Nom de naissance</i>	➤ BOUGHEDIRI
<i>Nom d'usage</i>	➤ BOUGHEDIRI
<i>Prénom</i>	➤ RIDHA
<i>Adresse</i>	➤ 22 RUE DU MUSÉE 13001

## Titre professionnel visé

Concepteur développeur d'application

### 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 <sup>(DP)</sup>

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

<b>Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité 1</b>	<b>5</b>
- Maquettage de l'application mobile <i>Chat_us</i>	p. 6
- Maquettage de l'application <i>la coop-du-digital</i>	p. 13
- Développer une interface utilisateur de type desktop	p. 18
<b>Concevoir et développer la persistance des données en intégrant les recommandations de sécurité 2</b>	<b>p.</b>
- Concevoir une base de données <i>Chat_us</i>	p. 23
- Concevoir une base de données <i>Chat_us</i>	p. 35
- Développer des composants dans le langage d'une base de données	P. 38
<b>Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité 3</b>	
- Construire une application organisée en couches n° 1	p. 31
- Préparer et exécuter les plans de tests d'une application n° 2	p. 61
- Préparer et exécuter le déploiement d'une application n° 3	p. 64
<b>Titres, diplômes, CQP, attestations de formation (facultatif)</b>	<b>p.</b>
<b>Déclaration sur l'honneur</b>	<b>p. 70</b>
<b>Documents illustrant la pratique professionnelle (facultatif)</b>	<b>p.</b>
<b>Annexes (Si le RC le prévoit)</b>	<b>p.</b>

# **EXEMPLES DE PRATIQUE PROFESSIONNELLE**

## Activité-type 1

### Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

Exemple n° 1 - *Chat\_sport : Maquetter une application*

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

Pendant ma formation au sein de La Plateforme, j'ai travaillé en équipe pour développer une application mobile de chat en React Native. Notre objectif principal était de créer une plateforme conviviale et facile à utiliser, permettant aux utilisateurs de se connecter, de communiquer et de partager leurs pronostics et statistiques pour chaque match. Nous avons volontairement opté pour une approche ouverte, visant à rendre l'application accessible à tous et à offrir un espace de détente pour les conversations.

Dans le cadre de ce projet, nous avons développé une application mobile ainsi qu'un panneau d'administration. Après avoir rédigé le cahier des charges, j'ai commencé par concevoir des maquettes à faible fidélité, puis à haute fidélité à l'aide de l'outil Figma.

Avant de me lancer dans la création des maquettes, nous avons établi une charte graphique pour guider notre travail. L'utilisation de Figma nous a permis de collaborer de manière efficace, en facilitant l'échange d'idées entre les membres de l'équipe.

Parallèlement, j'ai utilisé Trello pour collaborer à la gestion et à l'organisation du projet. Cette plateforme nous a permis de diviser les tâches en fonction de leur priorité et de leur complexité, ce qui a favorisé une meilleure organisation et offert une visibilité au client sur l'avancement du projet.

Cette expérience m'a permis d'acquérir des compétences en matière de conception d'interfaces d'application, de gestion de projets informatiques et d'organisation de l'environnement de développement.

Ce projet valide les compétences suivante :

- Maquetter une application
- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement

#### 2. Précisez les moyens utilisés :

Figma : maquettes

Trello : gestion de projet

# DOSSIER PROFESSIONNEL <sup>(DP)</sup>

## 3. Avec qui avez-vous travaillé ?

Pour ce projet j'ai travaillé avec Aurélien Adjami, [Yonathan DARMON](#)

## 4. Contexte

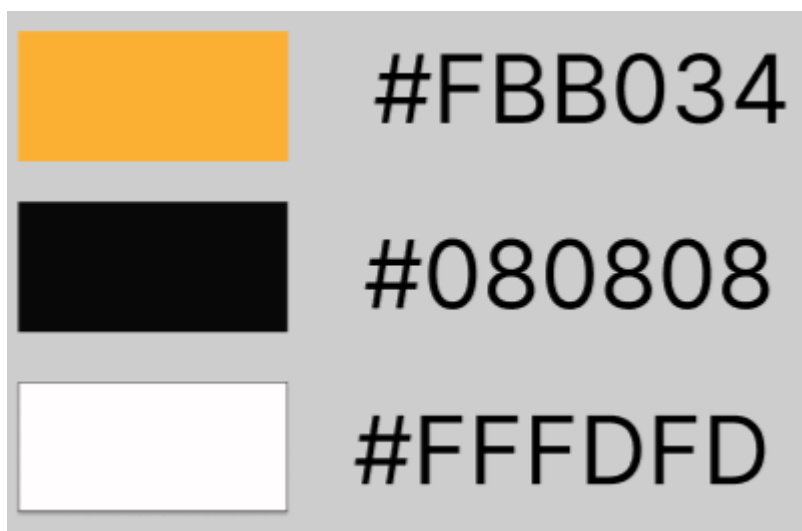
Nom de l'entreprise, organisme ou association ▶ *LAPLATEFORME*

Chantier, atelier, service ▶ *Chat\_sport\_us*

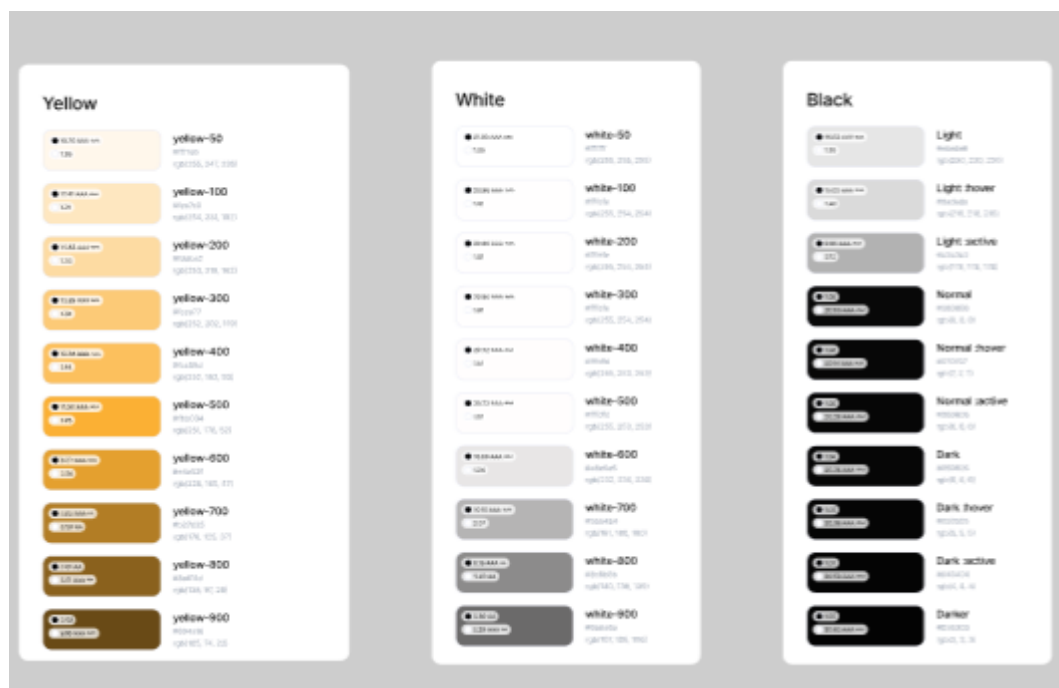
Période d'exercice ▶ Du : *02/01/2023* au : *11/01/2023*

## 5. Informations complémentaires (facultatif)

charte graphique:



# DOSSIER PROFESSIONNEL (DP)



## Typographies

### Metropolis

Texte, titre, titre section, composant, label, contenu des pages statiques

#### Texte mobile

Ceci est un titre H1 - 24px

Ceci est un titre H 2- 20px

Ceci est un titre H 3 - 18px

Ceci est un titre H 4- 16px

Ceci est un titre H 5- 14px

Texte 16 px : Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Maecenas faucibus mollis interdum. Maecenas faucibus. Cras mattis consectetur purus sit amet.

Texte 14 px : Donec sed odio dui. Cras justo odio, dapibus ac facilisis in, egestas eget quam. Maecenas faucibus mollis interdum. Maecenas faucibus. Cras mattis consectetur purus sit amet.

# DOSSIER PROFESSIONNEL <sup>(DP)</sup>



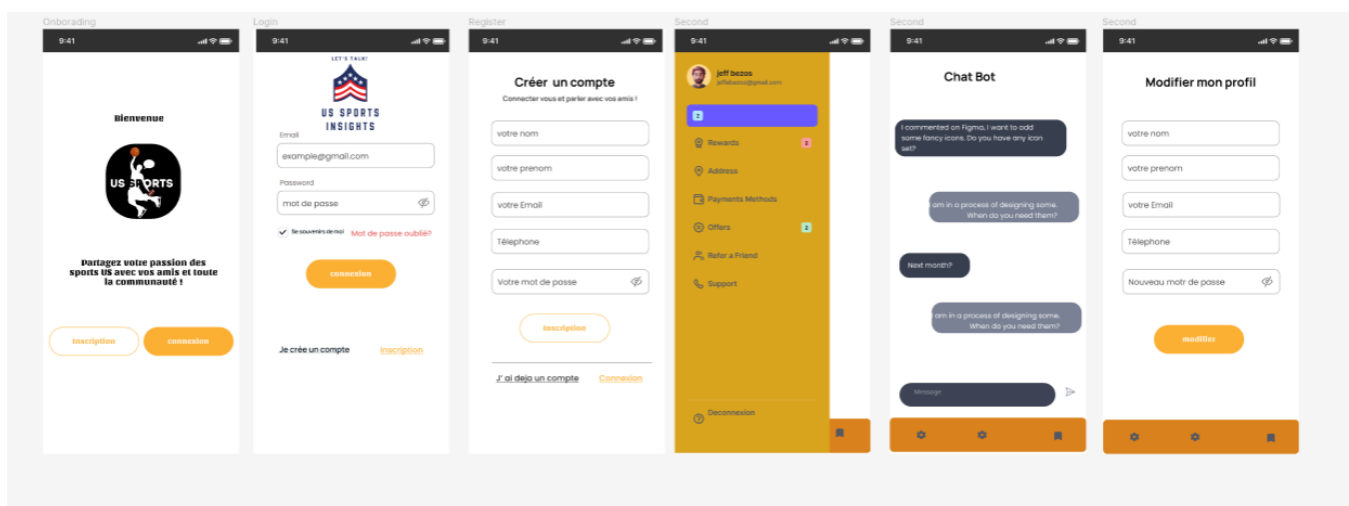
**Maquette basse fidélité :**



# DOSSIER PROFESSIONNEL (DP)



Maquette haute fidélité :



## Activité-type 1 Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

Exemple n°2 - **IODA Consulting**

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

C'est un projet que j'ai fait en groupe pendant mon alternance. Le demandeur est le Cabinet D'expert-comptable Ioda consulting à Aix-en-Provence. Il s'agit d'un cabinet d'expert comptable agissant comme plateforme de centralisation des horaires de formations pour les entreprises client du cabinet en Provence Alpes Côte d'Azur. Le **IODA Consulting** souhaite faire évoluer son modèle économique. En effet, le cabinet souhaite passer d'un modèle de comptable excel horaire à un modèle où il serait plus autonome et plus adéquat avec la demande client pour les frais de formation et le remboursement. L'outil ici développé, serait donc un service que proposerait les services IODA consulting sous forme de plateforme numérique à tous les clients et également leurs employés.

Pour cela nous avons donc réalisé des maquettes (annexe 1) puis une designeuse a réalisé la version finale du design. Puis nous avons commencé à faire un MPD pour enfin commencer le projet. Nous avons ensuite commencé à développer la partie front end avec Next Js et l'api avec strapi(un CMS headless qui permet de faire des API facilement).

### 2. Précisez les moyens utilisés :

NextJS pour le front-end, Strapi pour l'API, Git/Github et Jira pour la collaboration

### 3. Avec qui avez-vous travaillé ?

max machin

### 4. Contexte

Nom de l'entreprise, organisme ou association - **IODA CONSULTING**

Chantier, atelier, service - *Projet dans le cadre de mon alternance*

Période d'exercice - Du : 01/03/23 au : en cours

### 5. Informations complémentaires (facultatif)

## ANNEXES

# DOSSIER PROFESSIONNEL (DP)

(Si le RC le prévoit)



<

## Vos projets

Nom	Date début	Date fin	Nombre tâches	Statut	Actions
	2023-04-04	2024-04-05	3	En cours	
Harum error harum pr	2018-01-01	2024-01-01	2	En cours	



<

## Créer un nouveau projet

Nom du projet \*

Date de début  
jj/mm/aaaa

Date de fin  
jj/mm/aaaa

Phases du projet

[Ajouter une tâche personnalisée](#)

AJOUTER LE PROJET

<

- Mon compte
- Mes salariés
- Mes projets
- Validation
- Récapitulatif

SE DÉCONNECTER

MES EMPLOYÉS

## Vos projets

Nom	Date début	Date fin	Nombre tâches	Statut	Actions
	2023-04-04	2024-04-05	3	En cours	
Harum error harum pr	2018-01-01	2024-01-01	2	En cours	

## Activité-type 1 Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

Exemple n° 3 - Plateforme la coop-du-digital

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

En début d'année, nous avons pour mission de développer une plateforme de gestion de publicité pour notre client, Crédit Agricole, en utilisant le framework Symfony pour le back-end et React.js pour le front-end. Symfony est un framework qui permet de créer des applications fullstack en utilisant le pattern MVC (Modèle-Vue-Contrôleur). Il repose sur des bibliothèques externes pour gérer la création des modèles, les interactions avec la base de données, ainsi que les vues.

Pour la gestion des modèles, Symfony utilise l'ORM Doctrine, qui nous permet de créer des entités (modèles) et des migrations pour générer les tables dans notre base de données. Nous pouvons également valider les données et persister une entité dans la base de données grâce à l'entity manager.

Concernant les vues, Symfony utilise Twig, un moteur de templates qui facilite l'écriture du HTML et permet d'utiliser des variables provenant du contrôleur.

Notre application se compose de deux parties : une partie back office destinée à l'administrateur et une partie client destinée aux marketeurs. J'ai également créé des fixtures pour remplir la base de données avec de fausses données, telles que des articles, des commentaires, etc., afin de peupler le site.

### 2. Précisez les moyens utilisés :

React js pour le front-end, Symfony et Api platform pour l'API, Git/Github et Jira pour la collaboration

### 3. Avec qui avez-vous travaillé ?

[Louise DÉCOMBE](#), Fatima El Mohin

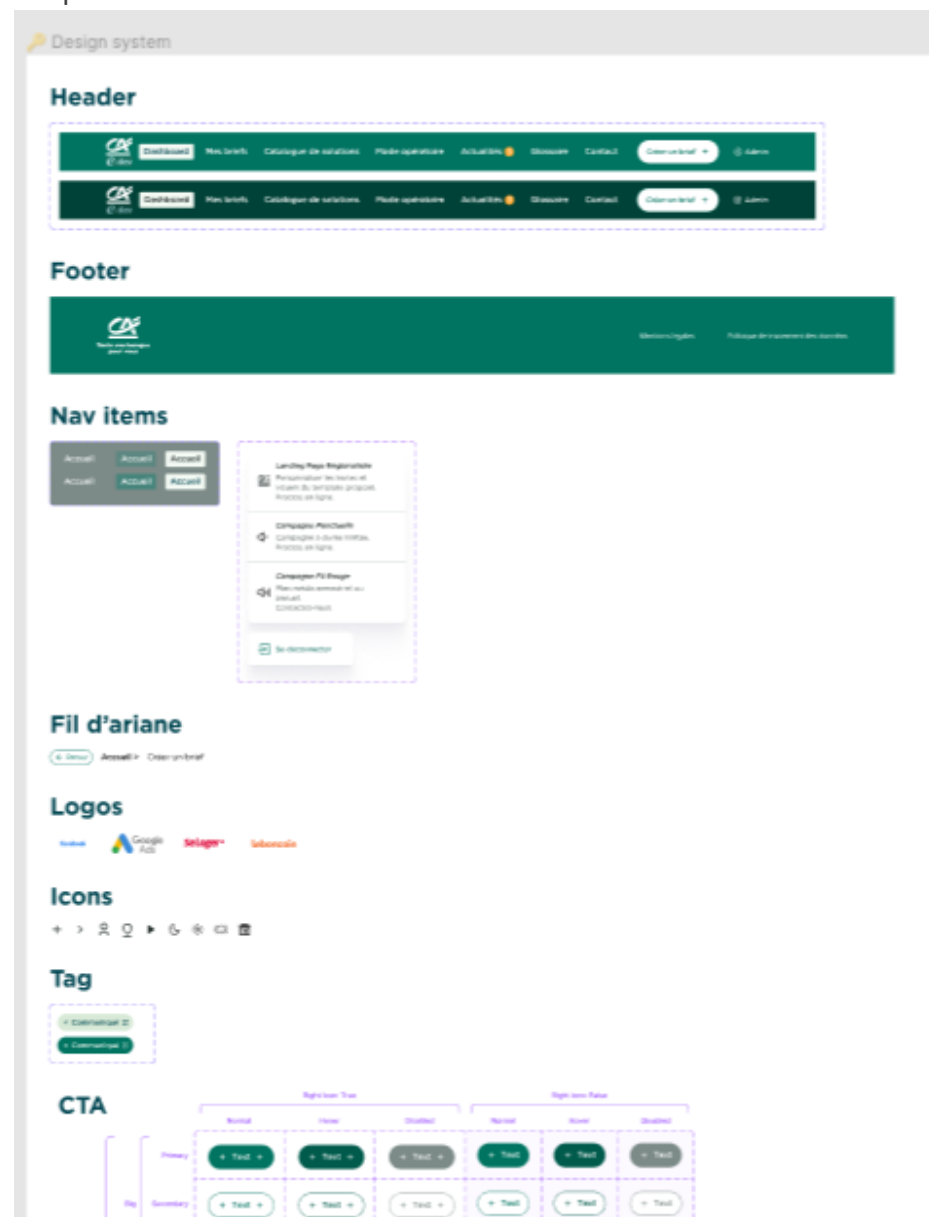
### 4. Contexte

Nom de l'entreprise, organisme ou association - *l'atelier de plateforme (Crédit agricole)*

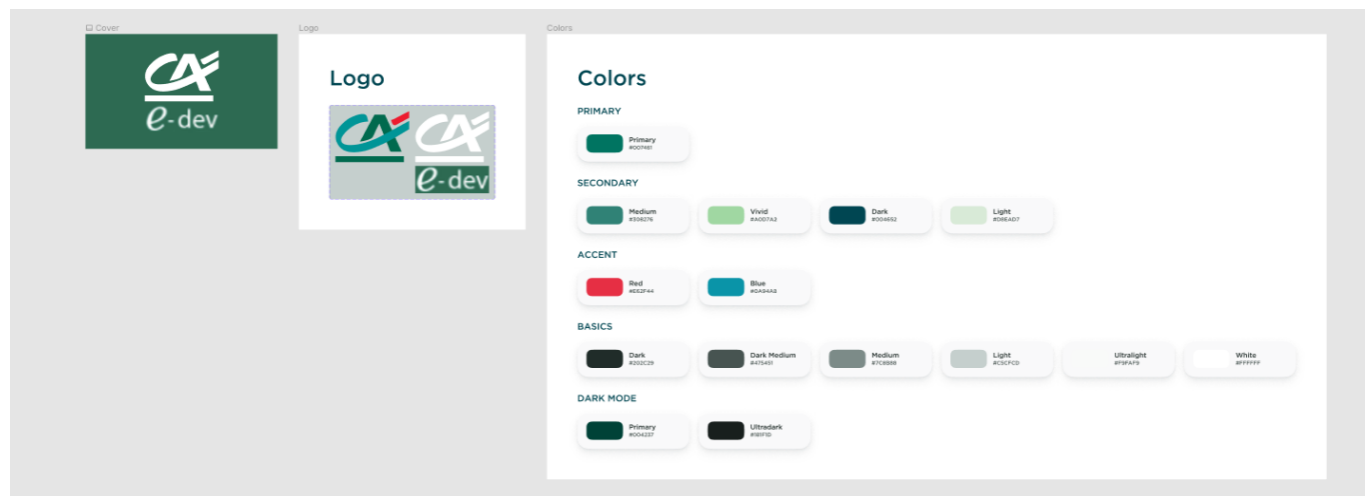
Période d'exercice - Du : 20/01/2023 au : 30/06/2023

### 5. Informations complémentaires (facultatif)

Maquette haute fidélité :

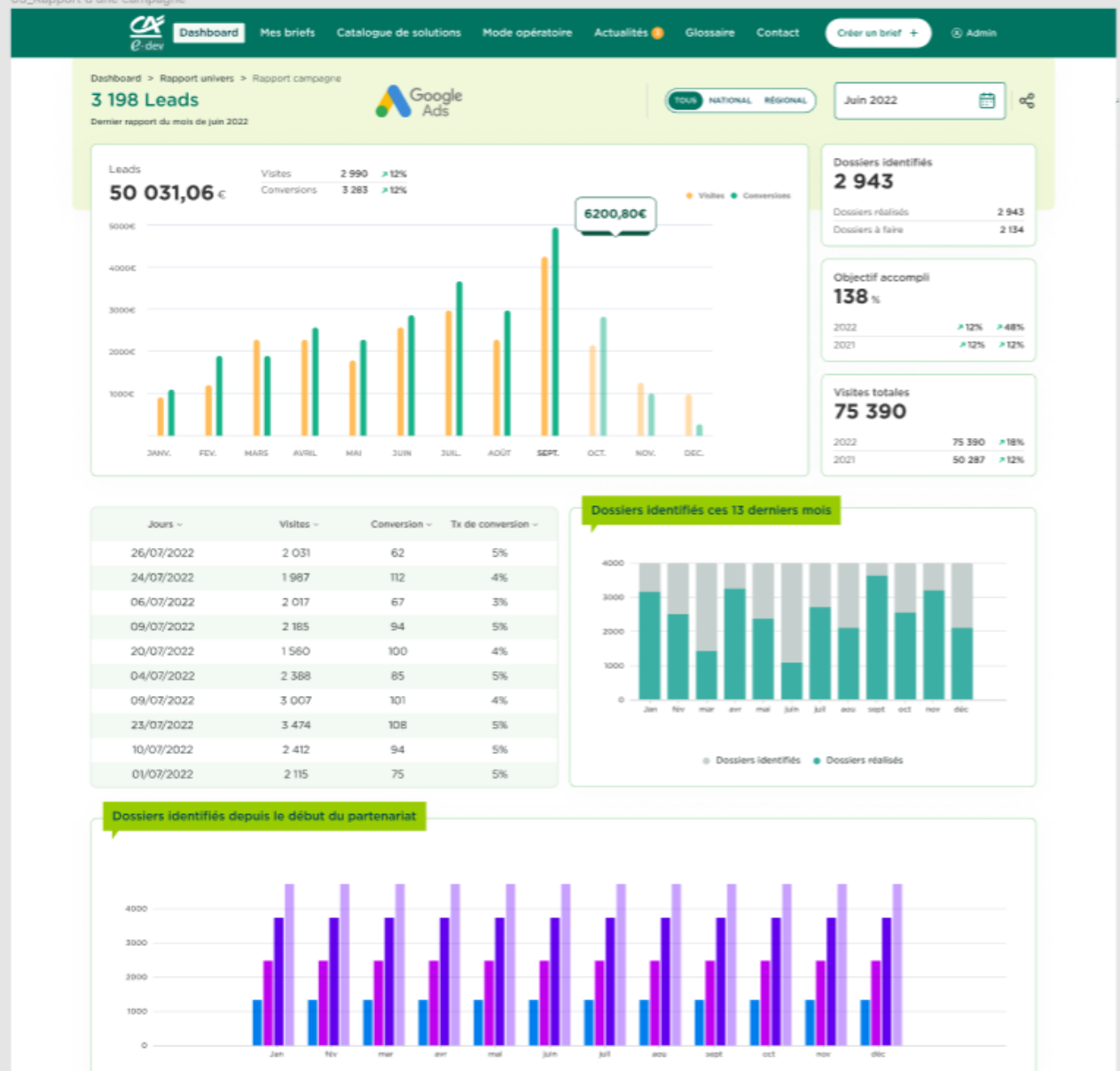


# DOSSIER PROFESSIONNEL (DP)



# DOSSIER PROFESSIONNEL (DP)

03\_Rapport d'une campagne



# DOSSIER PROFESSIONNEL (DP)

The screenshot displays the API Platform interface in a web browser. The top bar shows the URL `https://127.0.0.1:8000/api` and a status indicator "Non sécurisé". The interface is divided into two main sections: a list of endpoints on the left and a detailed view of the selected endpoint on the right.

**Endpoint List:**

- Brief**
  - GET `/api/briefs` Retrieves the collection of Brief resources.
  - POST `/api/briefs` Creates a Brief resource.
  - GET `/api/briefs/{id}` Retrieves a Brief resource.
  - PUT `/api/briefs/{id}` Replaces the Brief resource.
  - DELETE `/api/briefs/{id}` Removes the Brief resource.
  - GET `/api/user/briefs` Retrieves the collection of Brief resources.
- CampaignBudget**
  - GET `/api/campaign_budgets` Retrieves the collection of CampaignBudget resources.
  - POST `/api/campaign_budgets` Creates a CampaignBudget resource.
  - GET `/api/campaign_budgets/{id}` Retrieves a CampaignBudget resource.
  - PUT `/api/campaign_budgets/{id}` Replaces the CampaignBudget resource.
  - DELETE `/api/campaign_budgets/{id}` Removes the CampaignBudget resource.
  - PATCH `/api/campaign_budgets/{id}` Updates the CampaignBudget resource.
- CampaignContext**
  - GET `/api/campaign_contexts` Retrieves the collection of CampaignContext resources.
  - POST `/api/campaign_contexts` Creates a CampaignContext resource.
  - GET `/api/campaign_contexts/{id}` Retrieves a CampaignContext resource.

**Endpoint Detail View (Selected):**

- GET `/api/landing_pages/{id}`** Retrieves a LandingPage resource.
- PUT `/api/landing_pages/{id}`** Replaces the LandingPage resource.
- Login Check**
  - POST `/api/login` Creates a user token.

**Login Check Details:**

- Parameters:** No parameters.
- Request body:** Required. Media type: `application/json`.
- The login data:** Example Value: 

```
{  "username": "string",  "password": "string"}
```
- Responses:**

Code	Description	Links
200	User token created	No links
- Example Value:**

```
{  "token": "string"}
```

**User**

- GET `/api/me` Retrieves the collection of User resources.
- GET `/api/users/{id}` Retrieves a User resource.



### Activité-type 2 Concevoir et développer des composant d'interface utilisateur en intégrant les recommandations de sécurité

*Exemple n° 1 - Le jeu de Sokoban - Développer une interface utilisateur de type desktop*

---

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

Ce projet valide les compétences :

- Développer une interface utilisateur de type desktop

Dans le cadre de ma formation, j'ai développé le jeu du sokoban en python.

Pour cela, j'ai utilisé la librairie Pygames, qui me permet de créer des éléments graphiques ainsi que d'écouter sur des événements bien précis comme sur celui des touches du clavier. Ce projet a été codé Orienté Objet pour pouvoir au mieux faire interagir mes éléments et une plus grande lisibilité du code. J'ai commencé par initier le jeu avec la fonction init de pygame et définir la surface sur laquelle on va jouer. J'ai ajouté une fonction drawMap et drawPlayer pour dessiner le fond de mon interface afin d'être sûr de la position de mes murs et de mon joueur.

```
print("Bienvenue !")
print("[1] : Lancer le jeu")
print("[2] : Lancer l'éditeur de map")
choix = input("Réponse (1 ou 2) : ")
editor = Editor()
if choix == "1":
    map = input("Entrez le nom du fichier de la map : ")
    board = open("boards/{}.txt".format(map), "rb")
    board = pickle.load(board)
    game = Game(board)
    game.init()
elif choix == "2":
    editor.init()
else:
    print("Pas un choix")
```

Ensuite, j'ai créé une classe Player Pour gérer les différents événements de mon player.

You, 1 minute ago | 1 author (You)

```
class Player:

    def __init__(self, pos):
        self.pos = pos
        self.move_right()

    def move_right(self):
        return (self.pos[0], self.pos[1] + 1)
    def move_left(self):
        return (self.pos[0], self.pos[1] - 1)
    def move_up(self):
        return (self.pos[0] - 1, self.pos[1])
    def move_down(self):
        return (self.pos[0] + 1, self.pos[1])
```

You, 1

Il ne me reste plus qu'à lancer une boucle qui mettra le jeu à jour avec les nouvelles données à chaque action possible dans le jeu selon la touche pressée que j'aurai au préalable configurée.

You, 6 minutes ago | 1 author (You)

```
import pygame
from Player import Player
```

You, 6 minutes ago | 1 author (You)

```
class Game:
    def __init__(self, board):
        self.RUNNING = True
        self.board = board
        self.screen = 0
        self.HEIGHT = self.WIDTH = 800
        self.ROWS = self.COLS = 10
        self.ROW_WIDTH = self.WIDTH // self.COLS
        self.COL_WIDTH = self.HEIGHT // self.ROWS
        self.sprites = []
        self.player_sprite = []
        self.player_pos = (0, 0)
        self.music = 0
        self.player = 0
        self.collide = 0
        self.grunt = 0
```

You, 1 minute ago | 1 author (You)

```
class Player:
```

```
    def __init__(self, pos):
        self.pos = pos
        self.move_right()
```

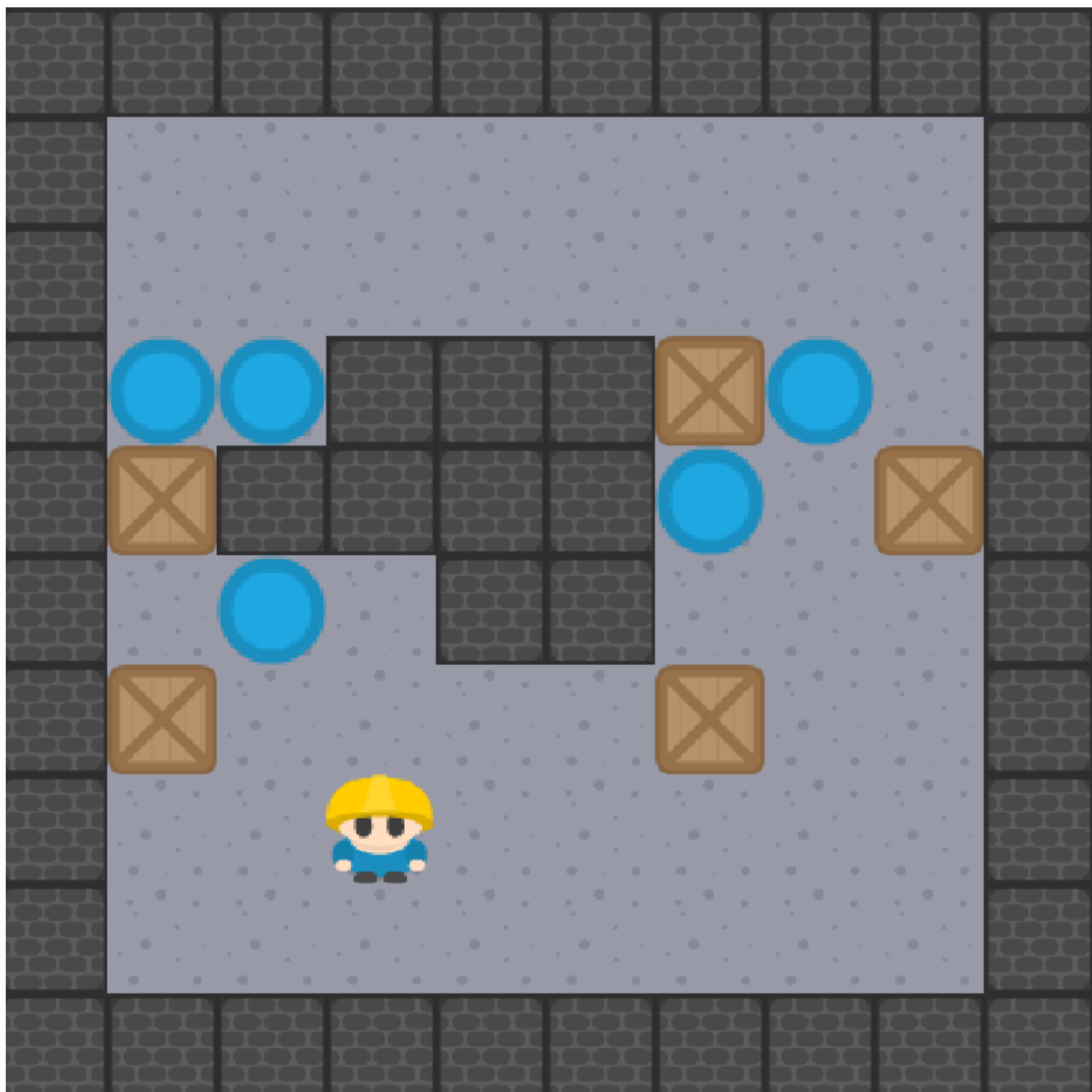
```
    def move_right(self):
        return (self.pos[0], self.pos[1] + 1)
```

```
    def move_left(self):
        return (self.pos[0], self.pos[1] - 1)
```

```
    def move_up(self):
        return (self.pos[0] - 1, self.pos[1])
```

```
    def move_down(self):
        return (self.pos[0] + 1, self.pos[1])
```

You, 1 minute ago • r



## 2. Précisez les moyens utilisés :

- Python
- Pip comme gestionnaire de package
- librairies : pygame, random et sys



Pour mener à bien ce projet j'ai utilisé le langage de programmation Python.  
Ce langage de programmation s'est hissé parmi les plus utilisés dans le domaine du développement de logiciels, de gestion d'infrastructure et d'analyse de données.

C'est d'ailleurs le langage de programmation le plus utilisé dans le domaine du Machine Learning, du Big Data et de la Data Science.

J'ai beaucoup utilisé figma pour créer, découper, redimensionner mes tiles (images de mon jeu, personnage...). Enfin j'ai utilisé Visual Studio Code comme IDE et Git, github comme outils de versionning.

### 3. Avec qui avez-vous travaillé ?

Clément CAILLAT

### 4. Contexte

Nom de l'entreprise, organisme ou association ► La plateforme

Chantier, atelier, service ► MySokoban

Période d'exercice ► Du : 20/01/2023 au : 30/06/2023

### 5. Informations complémentaires *(facultatif)*

## Activité-type 2

### Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

*Exemple n° 1 - Développer des composants d'accès aux données*

---

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

Dans le cadre de la conception du projet application mobile "Chat\_us", nous avons décidé d'utiliser Node.js avec Express et Sequelize parce que **Sequelize** est un ORM pour node.js compatible avec différents moteurs de base de données comme Mysql, Sqlite .

Express est un framework web pour Node.js qui facilite la création d'applications web et d'API REST. Il offre une structure légère et flexible, permettant de gérer les routes, les requêtes HTTP et les réponses de manière efficace.

Pour la persistance des données, nous avons utilisé Sequelize mysql2 , une base de données mysql2 . Sequelize offre une flexibilité importante dans la modélisation des données, ce qui est adapté aux besoins évolutifs de notre application.

L'ORM (Object-Relational Mapping) utilisé avec Node.js et Sequelize . Ce dernier permet de définir des schémas de données, de valider les entrées et d'interagir avec la base de données de manière simple et intuitive.

Ainsi, nous avons pu mettre en place notre API REST en utilisant Express et MongoDB avec l'aide de Mongoose pour la gestion de la persistance des données. Cette combinaison nous a permis de développer rapidement et efficacement notre application mobile "Neko" en exploitant les avantages de Node.js et Sequelize.

Ligne de commande permettant de créer un fichier package :

`npm init`

Ligne de commande pour démarrer un projet express :

`npm install express`

Ligne de commande pour initialiser sequelize :

`npm install -g sequelize-cli`

```
/***/
/** Import des modules nécessaires */
const { Sequelize } = require("sequelize");

/***/
/** Connexion à la base de données */
let sequelize = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASS,
  {
    host: process.env.DB_HOST,
    port: process.env.DB_PORT,
    dialect: "mysql",
    logging: false,
  }
);

/***/
/** Mise en place des relations */
```

Code gérant la connexion à la base de données Sequelize.

```
/** Import des modules nécessaires */
const { DataTypes } = require("sequelize");
/***/
/** Définition du modèle Message */
module.exports = (sequelize) => {
  const Message = sequelize.define(
    "message",
    {
      id: {
        type: DataTypes.INTEGER(10),
        primaryKey: true,
        autoIncrement: true,
      },
      content: {
        type: DataTypes.STRING,
        allowNull: false,
      },
    },
    { paranoid: true }
  ); // Ici pour faire du softDelete

  return Message;
};

/***/
/** Ancienne Synchronisation du modèle */
// Message.sync()
```

Définition du schéma de modèle utilisé par le module sequelize pour l'entité « message ». La création de ce modèle permet de réaliser des opérations CRUD sur la collection « message » dans la base de



## DOSSIER PROFESSIONNEL <sup>(DP)</sup>

données générer grâce a sequelize . En utilisant Sequelize migrate , l'approche de ma base de données est plutôt "code first" (ou "schema first"), cela signifie que je défini initialement les schémas des des collections formant la base de données dans mon code à l'aide de Sequelize.

### 2. Précisez les moyens utilisés :

- Un IDE Visual Studio
- Code Git pour le versionning
- Github comme plateforme de contrôle de version de git afin de pouvoir les utiliser en collaboration. Trello pour la gestion de projet
- Documentation Sequelize, express,Sequelize .

### 3. Avec qui avez-vous travaillé ?

Aurélien ADJIMI, Yonathan DARMON

### 4. Contexte

Nom de l'entreprise, organisme ou association ▶ *LAPLATEFORME*

Chantier, atelier, service ► Dans le cadre de ma formation concepteur/développeur d'application

Période d'exercice ▶ Du : *02/01/2023* au : *31/03/2023*

### 5. Informations complémentaires (facultatif)

---

## DOSSIER PROFESSIONNEL <sup>(DP)</sup>

---

---

### Activité-type 3

### Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

*Exemple n° 1 - Développer La partie front-end d'une interface utilisateur web*

---

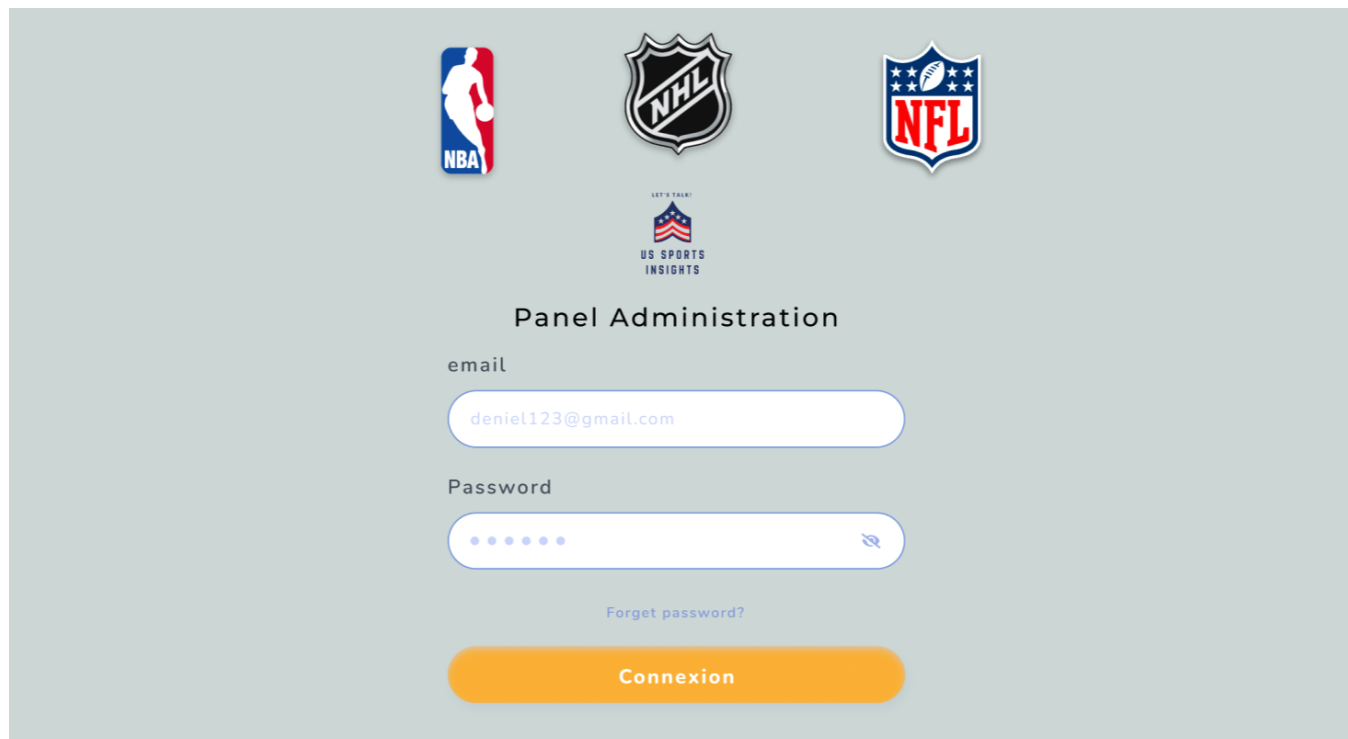
#### 1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

L'un des projets les plus importants de cette année est le développement d'une interface d'administration web spécialement conçue pour gérer et contrôler mon application mobile "chat\_us\_sport".

Pour le développement de la partie front-end de cette interface, mes collaborateurs et moi avons choisi d'utiliser Next JS, un framework open-source JavaScript qui permet de développer des applications web et mobiles avec JavaScript ou TypeScript.

Personnellement, j'ai choisi d'utiliser Next.JS pour deux raisons principales. Tout d'abord, ce framework est souvent considéré comme un choix idéal pour les petites à moyennes applications.

Deuxièmement, après avoir étudié React lors de ma formation et l'avoir utilisé en entreprise, je souhaitais également réaliser un projet en NextJS. J'ai donc saisi cette opportunité pour améliorer mes compétences dans cette technologie et compléter mes connaissances sur les frameworks front-end les plus populaires du marché actuel. Voici la page de connexion au panel administrateur :



NBA NHL NFL

LET'S TALK!  
US SPORTS  
INSIGHTS

**Panel Administration**

email

deniel123@gmail.com

Password

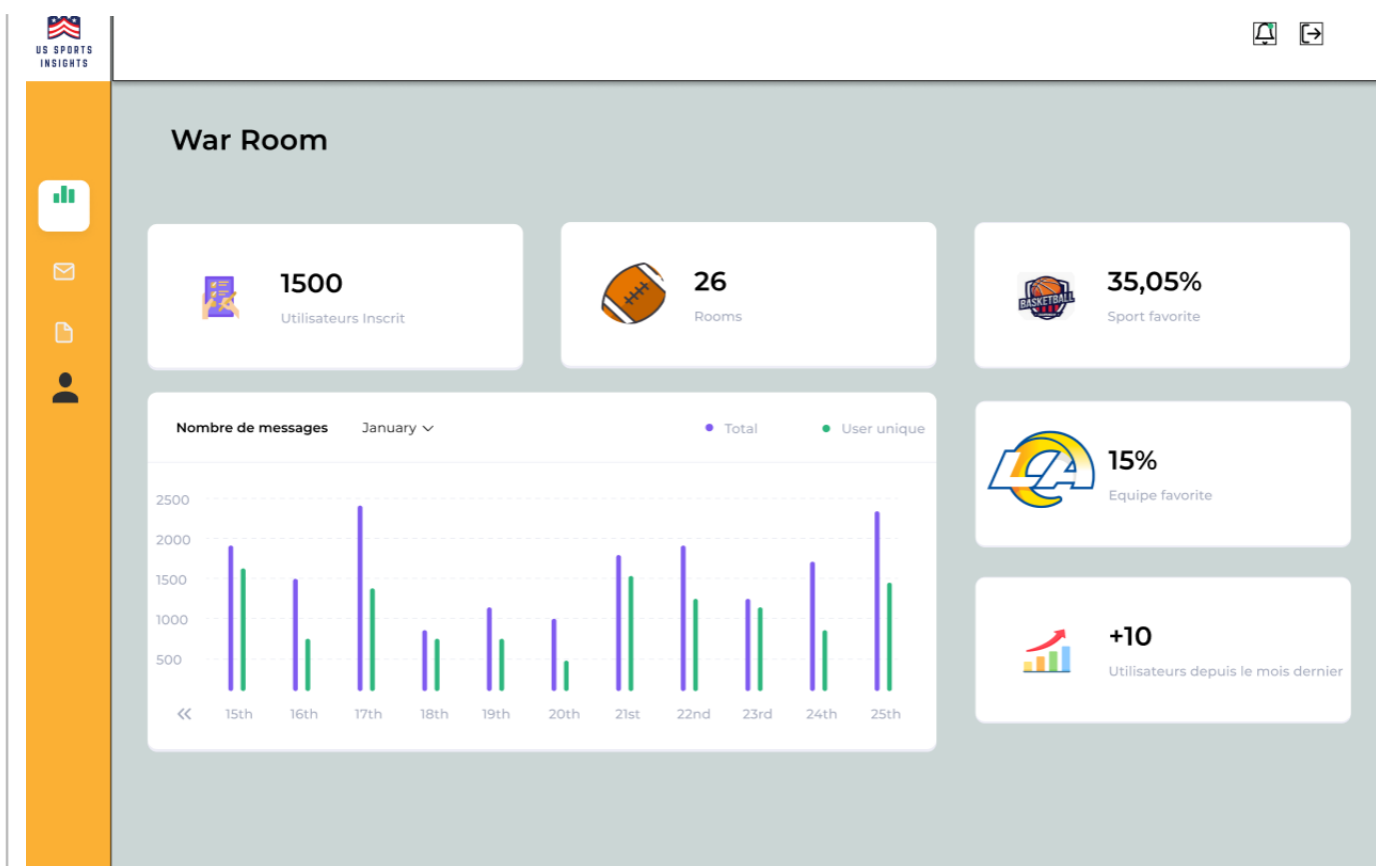
.....

[Forgot password?](#)

**Connexion**

Au niveau des fonctionnalités de cette interface, un utilisateur possédant les droits administrateurs sur mon application pourra gérer les utilisateurs inscrits sur l'application (gestion des droits de ceux-ci, bannissement d'utilisateurs malveillants, voir le nombre d'utilisateurs inscrits et connectés), il aura aussi une main sur la gestion de contenu de l'application. Page d'accueil :

# DOSSIER PROFESSIONNEL (DP)



Pour récupérer des données depuis la base de données, la communication entre le front et le back de mon application, qui est géré par une seule et même api, se fait en utilisant axios. C'est une bibliothèque javascript fonctionnant comme un client http, cela permet de créer des requêtes avec des méthodes (GET, POST, PUT et DELETE).

## 2. Précisez les moyens utilisés :

- Un IDE Visual Studio Code
- Git pour le versionning
- Github comme plateforme de contrôle de version de git afin de pouvoir les utiliser en collaboration. Trello pour la gestion de projet
- React js

## 3. Avec qui avez-vous travaillé ?

# DOSSIER PROFESSIONNEL <sup>(DP)</sup>

Aurélien ADJIMI, Yonathan DARMON

## 4. Contexte

Nom de l'entreprise, organisme ou association ➤ *Laplateforme*

Chantier, atelier, service ➤ *Dans le cadre de ma formation concepteur/développeur d'application*

Période d'exercice ➤ Du : *02/01/2023* au : *31/03/2023*

## 5. Informations complémentaires (facultatif)

## Activité-type 3

### Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

Exemple n° 1 - Développer La partie back-end d'une interface utilisateur web

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

Dans le périmètre du projet d'application mobile, j'avais à développer avec d'autres étudiants de ma promotion une API REST qui peut être consommée par mon application mobile et également par l'interface administrateur web.

Nous avons le choix entre deux stack, php et symfony ou bien nodeJs et express, notre projet s'est tourné vers l'utilisation du framework javascript car après des recherches poussées nous avons trouvé que nodeJs/Express est souvent utilisé pour des cas d'utilisation et une manipulation intensive de données en temps réel, et nous voulions également être libre d'organiser notre code comme on le voulait et express offre une grande flexibilité et modularité au niveau de l'architecture du projet.

Une fois le choix effectué nous pouvions commencer le développement, premièrement il y a l'initialisation du projet : Créer un nouveau répertoire pour votre projet et initialisez-le en utilisant npm (Node Package Manager) avec la commande npm init.

Cela créera un fichier package.json qui contiendra les dépendances et les informations de configuration de votre projet nécessaires pour le projet. Ensuite il faut installer express : En utilisant la commande npm install express cela permettra de l'installer et de l'ajouter aux dépendances de mon projet.

Une fois ces deux étapes réalisées il faut créer le fichier qui servira de point d'entrée de mon application, dans mon cas ce sera server.js

```
const express = require("express");
const app = require("express")();
const http = require("http");
const server = http.createServer(app);
```

Maintenant je peux commencer à définir mes routes qui seront appelées par mon application mobile et mon panel admin :

```
app.use("/roles", routeRole);
app.use("/auth", routeLogin);
app.use("/users", routeUsers);
```

Sur l'exemple ci-dessus, j'ai au préalable créé une nouvelle instance du router grâce à celui-ci je peux définir des endpoints (URI) qui peuvent accepter différentes requêtes http et ensuite permettent d'effectuer pour chacun

une exécution d'une fonction d'un de mes controllers qui contient la logique métier et qui est responsable des échanges avec ma BDD (certaines de ces URI/routes sont protégées par un middleware d'authentification. Par exemple, la route qui termine par « /login » accepte uniquement une requête http POST contrairement à la route racine terminant par « / » qui accepte les requêtes POST et GET (cette dernière est protégée par un middleware), une fois appelée par le client cette route appelle la fonction login de mon controller qui gère les utilisateurs.

**Middleware :** Un middleware est une fonction ou un ensemble de fonctions dans une application web qui s'exécute entre la réception d'une requête du client et l'envoi d'une réponse par le serveur. Il agit comme une couche intermédiaire entre le client et le serveur, traitant et manipulant les requêtes avant qu'elles n'atteignent leur destination finale.

```
router.post('/login', async (req, res) => {
  const {email, password} = req.body
  // je valide les données reçues
  if(!email || !password){
    return res.status(400).json({message: "email ou mot de passe incorrect"})
  }
  // trouver l user si il existe dans la database
  db.user.findOne({where: {email: email}, raw: true})
    .then(user => {
      // si l'utilisateur existe
      if(user === null){
        return res.status(401).json({message: "le compte lié à ce mail n'existe pas !"})
      }
      // check le mot de passe
      bcrypt.compare(password, user.password)
        .then(test => {
          if(!test){
            return res.status(401).json({message: "mot de passe incorrect"})
          }
          // generer le jwt token
          const putaindetoken = jwt.sign({
            // les infos que je veux envoyés
            id: user.id,
            // ...
          }, process.env.JWT_SECRET, {expiresIn: '1h'})
          return res.status(200).json({token: putaindetoken})
        })
    })
})
```

Exemple d'une fonction de mon User controller qui permet de récupérer les informations d'un utilisateur avec son id. Une fois que les routes, les controller et les middleware pour le router sont définis, j'ai pu monter le router sur mon application Express principale en utilisant la méthode `app.use()`



```
router.patch("/:id", async (req, res) => {
  const userId = req.params.id;
  const { lastname, firstname, login, email, password } = req.body;

  // Validation des données reçues
  if (!lastname || !firstname || !login || !email || !password) {
    return res.status(400).json({ message: "Missing Data" });
  }

  try {
    // Vérification si l'utilisateur existe
    const user = await db.user.findByPk(userId);
    if (!user) {
      return res.status(404).json({ message: "User not found" });
    }

    // Mise à jour des données de l'utilisateur
    user.lastname = lastname;
    user.firstname = firstname;
    user.login = login;
    user.email = email;

    // Hashage du nouveau mot de passe
    const hash = await bcrypt.hash(
      password
```

A partir de là on peut démarrer l'api en local qui écoute sur le port choisi en amont en utilisant la commande:  
npm run dev

```
const port = 8888;
http.listen(port, () => {
  console.log(`Example app listening on port ${port}`);
});
```

### 2. Précisez les moyens utilisés :

- Un IDE Visual Studio Code

## DOSSIER PROFESSIONNEL <sup>(DP)</sup>

- Git pour le versionning
- Github comme plateforme de contrôle de version de git afin de pouvoir les utiliser en collaboration. Trello pour la gestion de projet
- Un IDE NPM Express
- Postman pour tester les appels à mon API

### 3. Avec qui avez-vous travaillé ?

Aurélien ADJIMI, Yonathan DARMON

### 4. Contexte

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

Chantier, atelier, service ▶ *Dans le cadre de ma formation concepteur/développeur d application*

Période d'exercice ▶ Du : *02/01/2023* au : *31/03/2023*

### 5. Informations complémentaires (facultatif)

## Activité-type 2 *Concevoir et développer la persistance des données en intégrant les recommandations de sécurité*

*Exemple n° 1 - Conception et mise en place de la base de donnée de chat\_us\_sport*

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

Pour concevoir et développer la persistance des données, j'ai suivi les étapes suivantes :

Identifier les besoins : Tout d'abord, j'ai identifié les besoins de l'application pour laquelle la base de données sera utilisée. J'ai cherché à comprendre les types de données qui doivent être stockés, la fréquence à laquelle ces données seront utilisées et comment elles seront liées les unes aux autres.

Créer un schéma conceptuel : Ensuite, j'ai créé un schéma conceptuel qui décrit les entités principales de la base de données, les relations entre ces entités et les contraintes de l'application. Cela m'a aidé à clarifier la structure globale de la base de données avant de commencer à travailler sur les détails techniques. Le schéma de la base de données est la structure qui définit la façon dont les données sont stockées dans la base de données.

Créer un schéma logique : Après avoir créé le schéma conceptuel, j'ai créé le schéma logique qui décrit les tables spécifiques de la base de données, les champs de chaque table et les clés primaires et étrangères qui établissent des relations entre les tables.

Normaliser les données : La normalisation des données est un processus qui permet d'organiser les données de manière à éliminer les redondances et à améliorer l'efficacité de la base de données. Cela implique de séparer les données en tables distinctes et de minimiser les champs répétitifs.

Vérifier la qualité des données : Il est important de vérifier la qualité des données pour s'assurer que les données stockées dans la base de données sont fiables et cohérentes. Cela implique de définir des règles d'intégrité des données, de valider les entrées utilisateur et de nettoyer les données avant de les stocker. Il existe plusieurs outils et méthodes utilisés pour concevoir une base de données relationnelle efficace. Dans mon cas, les méthodes utilisées sont les suivantes :

Le MCD (Modèle Conceptuel de Données) est un outil important dans la conception de bases de données relationnelles. Il permet de modéliser les entités et les relations entre ces entités dans la base de données. Il est utilisé pour définir la structure globale de la base de données sans entrer dans les détails techniques.

Le MLD (Modèle Logique de Données) est la représentation concrète du MCD dans un langage adapté à la gestion des bases de données relationnelles. Il définit les tables, les champs, les clés primaires et

étrangères, ainsi que les contraintes d'intégrité.

La conception physique consiste à traduire le MLD en un modèle physique de la base de données, en définissant les types de données spécifiques pour chaque champ, les index et autres optimisations pour améliorer les performances.

La mise en œuvre de la base de données implique la création des tables, des vues, des procédures stockées et des autres éléments nécessaires dans le système de gestion de base de données choisi.

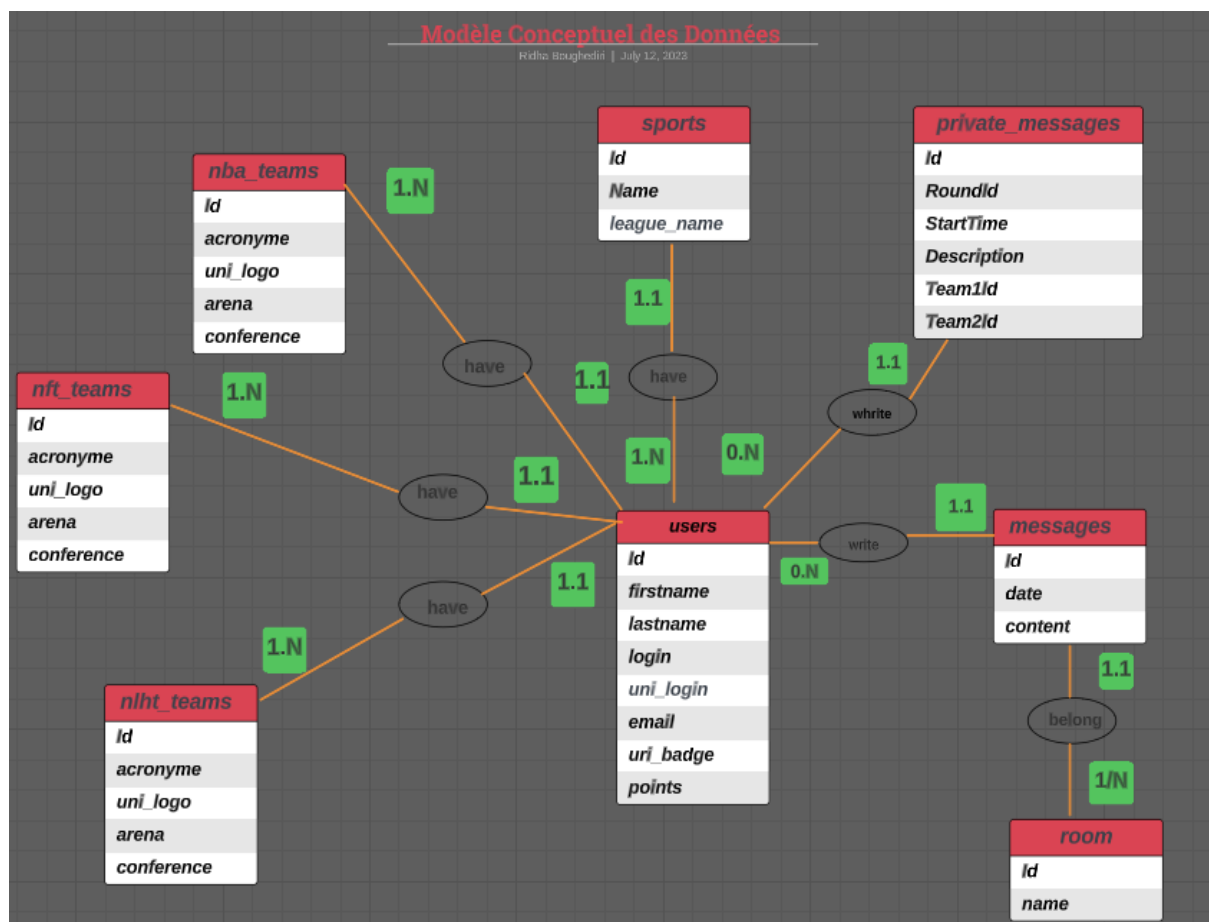
En suivant ces étapes, j'ai pu concevoir et développer la persistance des données de manière efficace et cohérente pour l'application Chat\_us.

I - Conception Pour la base de données, j'ai décidé d'utiliser Sequelize qui est une bibliothèque ORM (Object-Relational Mapping) pour diverses raisons. Ce fût premièrement une initiative personnelle car n'ayant pas eu l'occasion de l'aborder en formation, je me suis dit que c'est une bonne occasion d'apprendre. De plus, c'est une compétence très demandée. Ayant pour objectif d'avoir une bonne maîtrise de la stack MERN (MySQL, ExpressJS, React, NodeJS), ce choix me parut évident. Sequelize fonctionne en interagissant avec différentes bases de données relationnelles, telles que MySQL, PostgreSQL, SQLite, etc.

Afin de faciliter l'accès et les différentes opérations liées à la base de données, j'ai utilisé Sequelize. Cette dernière permet de définir des modèles avec des données fortement typées. Une fois qu'un modèle est défini, Sequelize permet de créer une instance basée sur ce modèle spécifique. De nombreuses fonctions permettant de sélectionner, enregistrer, supprimer des données ainsi que d'autres fonctionnalités telles que des hooks, les associations, les validations sont mises à disposition par Sequelize.

Après avoir réfléchi à la structure de la base de données, j'ai réalisé un schéma de cette dernière via le logiciel Lucidchart.

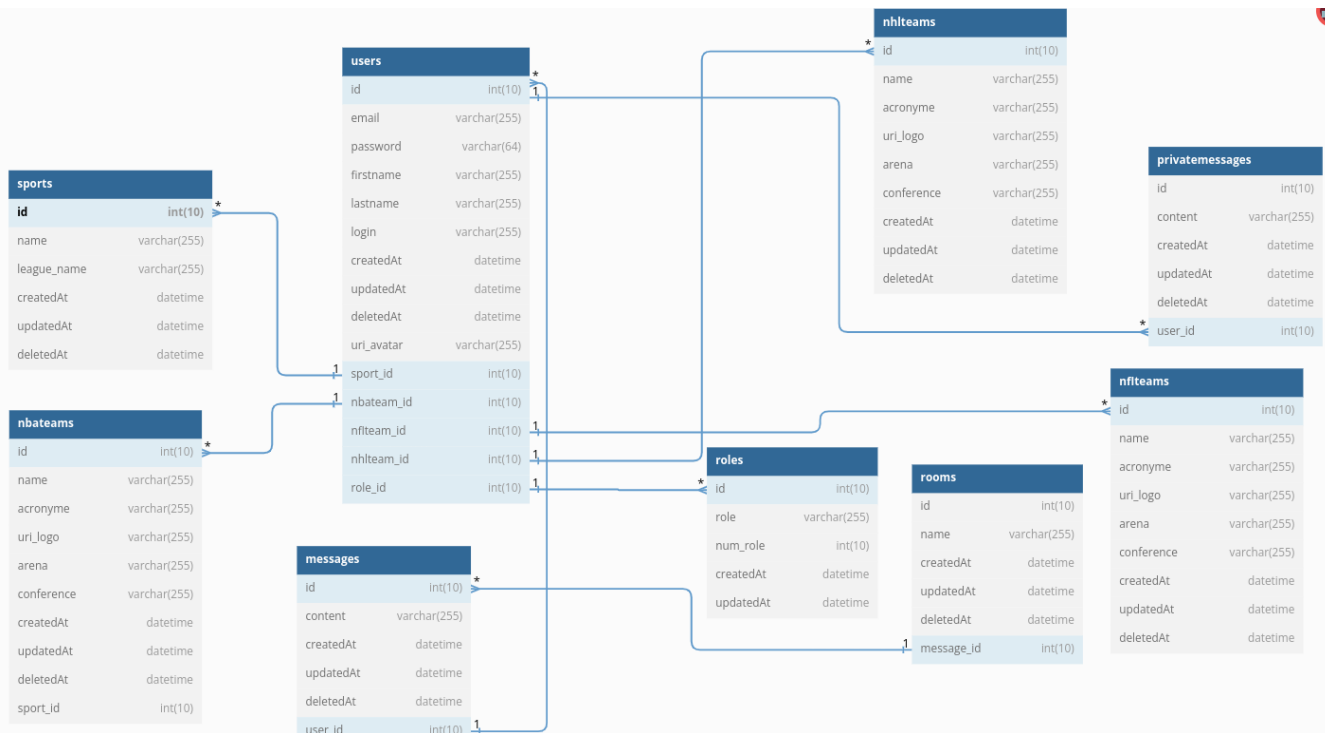
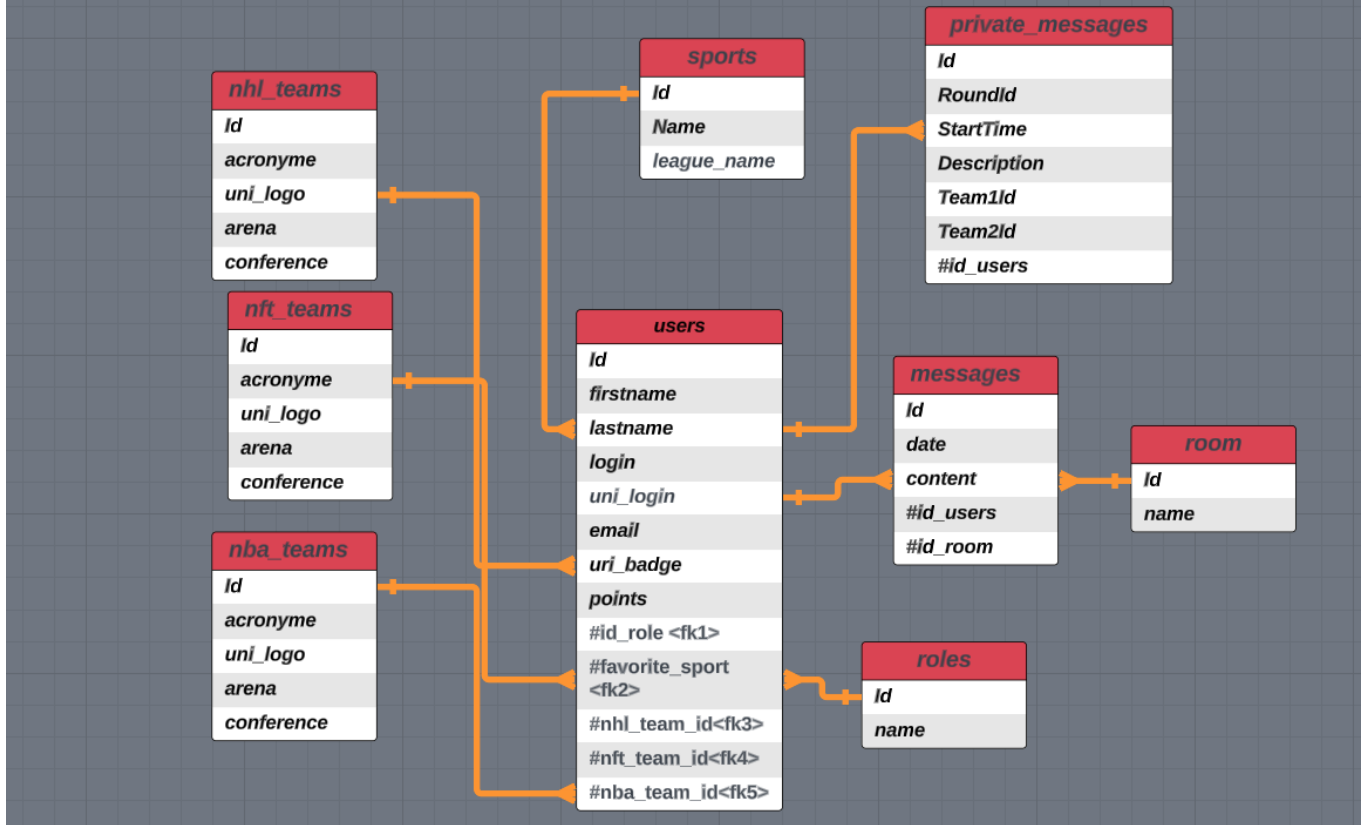
# DOSSIER PROFESSIONNEL (DP)



# DOSSIER PROFESSIONNEL (DP)

## Modèle Logique de Données (Relationnelles)

Ridha Boughediri | July 12, 2023



# DOSSIER PROFESSIONNEL <sup>(DP)</sup>

## 2. Précisez les moyens utilisés :

Lucidchart pour concevoir le MCD et le MLD  
DB diagramm pour concevoir le MPD

## 3. Avec qui avez-vous travaillé ?

Aurélien ADJIMI, Yonathan DARMON

## 4. Contexte

Nom de l'entreprise, organisme ou association ▶ *Laplateforme*

Chantier, atelier, service ▶ *Dans le cadre de ma formation concepteur/développeur d application*

Période d'exercice ▶ Du : *02/01/2023* au : *31/03/2023*

## 5. Informations complémentaires (facultatif)

## Activité-type 3

### Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

Exemple n° 1 - Réalisation du site web Serv-top (API et client)

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

Lors de ma formation à La Plateforme\_, j'ai eu l'occasion de créer Serv-top, une application mobile de référencement de serveurs privés de jeux-vidéos dont je vous ai présenté la maquette de l'application plus haut dans ce dossier. Pour mener ce projet à bout, j'ai décidé de réaliser un site web en lien avec cette application mobile et donc utilisant la même API à côté de ma formation. Je vais donc vous présenter dans cette section la création de ce site web et de l'API.

Sur beaucoup de jeux-vidéos, les joueurs ont la possibilité de créer leur propre serveur afin de jouer à plusieurs. Dès lors, de nombreux serveurs privés naissent chaque jour. Serv-top est un site web et une application mobile ayant pour but de référencer ces serveurs privés dans un classement. Ainsi, notre application s'adresse à deux types d'utilisateurs : les joueurs, qui recherchent un serveur sur lequel jouer, et les créateurs, qui cherchent à faire connaître leur serveur. Serv-top propose de référencer les serveurs de plusieurs jeux-vidéos dont, par exemple, Minecraft, GTA RP, Ark et Hytale. Un système de vote est mis en place afin de créer un classement. Plus un serveur a de votes, mieux il est référencé et donc a de la visibilité. Un utilisateur ne peut voter pour un serveur qu'une fois toutes les deux heures. Les votes sont remis à zéro chaque premier du mois afin de laisser une chance aux nouveaux serveurs de se démarquer et ainsi de gagner en popularité. Un utilisateur a également la possibilité de laisser un avis et une note à un serveur un peu à l'image de Google My Business.

#### 1.1 - Développement de l'API

Pour développer cette API j'ai utilisé l'environnement d'exécution **NodeJS** ainsi que le framework **ExpressJS**.

#### Arborescence

J'ai suivi l'architecture MVC (sans les views puisque c'est une API) pour développer notre API. Cette dernière est constituée d'un dossier routes dans lequel sont rangés les routeurs de l'API. Mes controllers et mes models, sont rangés dans des dossiers nommés respectivement controllers et models. Un dossier

utils contient des fonctions qui se répètent dans mon code pour éviter de surcharger mes controllers. Enfin un dossier public contient les images stockées dans l'API. Tout le monde peut y accéder.

Fonctionnement de l'API

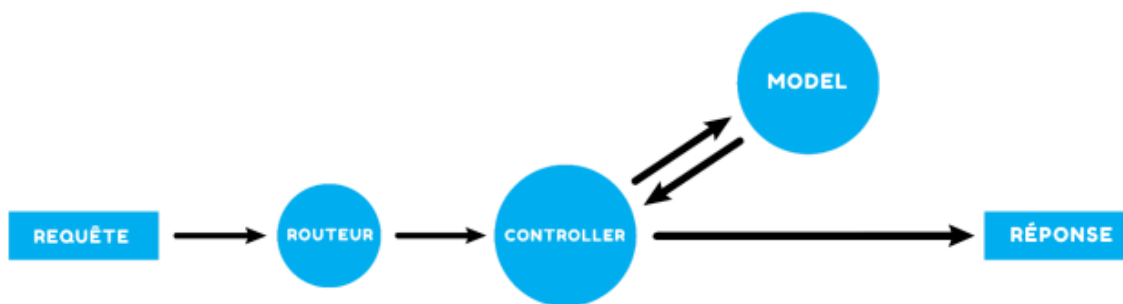


Lorsque un utilisateur fait une requête à mon API, cette dernière passe, en fonction de la route, par un de mes routeurs.

Ce dernier, en fonction de la route et de la méthode (get, post, ...), appelle ensuite un de mes controllers (qui contient la logique).

Le controller utilise un des mes models pour faire des manipulations avec les données.

Enfin le controller renvoie une réponse sous format JSON, prête à être exploitée par l'application mobile et le site web.



*Architecture de l'API*

### Middlewares :

Il me paraît important de définir ce qu'est un middleware car je serai amené à utiliser ce terme beaucoup de fois dans ce dossier. Les middlewares sont tout simplement des fonctions qui peuvent accéder à l'objet Request (req), l'objet response (res) et à la fonction middleware suivant dans le cycle demande-réponse de l'application. La fonction middleware suivante est couramment désignée par une variable nommée next.

Les fonctions middleware peuvent effectuer les tâches suivantes :

- Exécuter tout type de code
- Apporter des modifications aux objets de demande et de réponse
- Terminer le cycle de demande-réponse
- Appeler la fonction middleware suivante dans la pile

### Routage

Dans mon fichier app.js, j'utilise des fonctions middlewares d'ExpressJS pour faire le routage de mon application. Ces dernières me permettent d'appeler un routeur en fonction de la route demandée par

l'utilisateur. Dans l'exemple ci-dessous, si l'url demandée se finit par **/api**, mon routeur

**RouterMessages** est utilisé.

```
app.use(helmet());

app.use(bodyParser.json());

app.use("/rooms", routeRooms);
app.use("/sports", routeSports);
app.use("/messages", routeMessages);
app.use("/nbaTeams", routeNba);
app.use("/nflTeams", routeNFL);
app.use("/nhlTeams", routeNHL);

app.use("/roles", routeRole);
app.use("/auth", routeLogin);
```

Dans mes routeurs, la classe `express.Router` me permet de créer un système de routage complet. En effet, grâce à cette dernière je peux utiliser ce que l'on appelle des "middlewares niveau routeur". Ces middlewares fonctionnent de la manière suivante :

- Dans un premier temps, une méthode `route()` me permet de définir un schéma de route
- Ensuite, je peux utiliser les méthodes `get`, `post`, `patch` et `delete` pour faire différentes actions en fonction de la méthode de la requête. Par exemple,
  - voici ci-dessous un extrait de mon routeur qui gère les routes en rapport avec les messages . Lorsque la route `/api/messages /` est appelée avec la méthode `GET`, la fonction **findALL()** de mon contrôleur `messageController` est utilisée

```
router.get("/", (req, res) => {
  db.message
    .findAll()
    .then(messages => res.json({ data: messages }))
    .catch(err =>
      res.status(500).json({ message: "Database Error", error: err })
    );
});
```

Autre exemple, lorsque la route appelée est /api/messages /1 avec la méthode PATCH, les fonctions protect, *Checktokenexiste* et de mes controllers sont utilisées.

```
router.patch("/:id", checkTokenexist, async (req, res) => {
  let messageId = parseInt(req.params.id);

  if (!messageId) {
    return res.status(400).json({ message: "Missing parameter" });
  }

  try {
    // Recherche de l'message et vérification
    let message = await db.message.findOne({
      where: { id: messageId },
      raw: true,
    });
    if (message === null) {
      return res.status(404).json({ message: "This message does not exist" });
    }

    // Mise à jour de l'message
    await db.message.update(req.body, { where: { id: messageId } });
    return res.json({ message: "message Updated" }, res.status(200));
  } catch (err) {
    return res.status(500).json({ message: "Database Error", error: err });
  }
});
```

### Controllers

Les controllers contiennent la logique de mon application. Ceux sont eux par exemple qui gère les

requêtes et les réponses. La plupart de mes controllers de mon application ont cinq mêmes fonctions. Ce sont celles qui me permettent de faire mon CRUD (Create, Request, Update, Delete).

Bien évidemment, certains de mes controllers ont d'autres fonctions propres à leur fonctionnalités.

Voici

ci-dessous une de mes fonctions qui sont propres à certains controllers :

- authController.js

```
const bcrypt = require("bcrypt");
const jwt = require("jsonwebtoken");
const DB = require("../db.config");
const User = DB.User;

exports.login = async (req, res) => {
  try {
    const { email, password } = req.body;

    if (!email || !password) {
      return res.status(400).json({ message: "Mauvais email ou mot de passe" });
    }

    const user = await User.findOne({ where: { email: email }, raw: true });

    if (!user) {
      return res.status(401).json({ message: "Ce compte n'existe pas !" });
    }

    const isValid = await User.checkPassword(password, user.password);

    if (!isValid) {
      return res.status(401).json({ message: "Mot de passe incorrect" });
    }

    const token = jwt.sign(
      {
        id: user.id,
        nom: user.nom,
        prenom: user.prenom,
        email: user.email,
      },
      process.env.JWT_SECRET,
      { expiresIn: process.env.JWT_DURATION }
    );

    return res.json({ access_token: token });
  } catch (err) {
    if (err.name === "SequelizeDatabaseError") {
      res
        .status(500)
        .json({ message: "Erreur de base de données", error: err });
    } else {
      res
        .status(500)
        .json({ message: "Échec du processus de connexion", error: err });
    }
  }
}
```

## Models :

Dans mes controllers je fais appel à mes models. Ces derniers contiennent la “business logic” de mon application, c’est-à-dire que c’est eux qui s'occupent de ce qui est en rapport avec les données.

Pour gérer les interactions avec la base de données, j'utilise l'ORM Sequelize qui est une bibliothèque JavaScript qui permet de définir des schémas avec des données fortement typées. Une fois qu'un schéma est défini, Sequelize permet de générer une table basée sur un schéma spécifique.

J'ai donc défini un schéma Sequelize dans chacun de mes models. Chaque schéma correspond à une collection Sequelize et définit la forme de l'entité au sein de cette collection.

```
/** Import des modules nécessaires */
const { DataTypes } = require("sequelize");

/** Définition du modèle User */
module.exports = (sequelize) => {
  const User = sequelize.define(
    "user",
    {
      id: {
        type: DataTypes.INTEGER(10),
        primaryKey: true,
        autoIncrement: true,
      },
      email: {
        type: DataTypes.STRING,
        validate: {
          isEmail: true, //une validation de
        },
        allowNull: false,
      },
      password: {
        type: DataTypes.STRING(64),
        is: /^[0-9a-f]{64}$/i, //ici une con
        allowNull: false,
      },
      firstname: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      lastname: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      login: {
        type: DataTypes.STRING,
        allowNull: false,
      },
      uri_avatar: {
        type: DataTypes.STRING,
        allowNull: true,
      },
      uri_avatar: {
        type: DataTypes.STRING
    },
      uri_avatar: {
        type: DataTypes.STRING,
        allowNull: true,
      },
      uri_avatar: {
        type: DataTypes.STRING,
        allowNull: true,
      },
      points: {
        type: DataTypes.INTEGER,
        allowNull: true,
      },
      uri_badge: {
        type: DataTypes.STRING,
        allowNull: true,
      },
    },
    { paranoid: true }
  ); // Ici pour faire du softDelete

  return User;
};

/** Ancienne Synchronisation du modèle */
// User.sync();
// User.sync({force: true})
//User.sync({alter: true})
```

Extrait du model userModel.js

Dans Sequelize, les schémas (ou modèles) définissent la structure de la table de base de données et permettent à Sequelize de comprendre comment interagir avec cette table.

Un schéma est une représentation JavaScript d'une table de base de données, qui inclut des informations sur les colonnes, les types de données, les clés primaires, les associations et les contraintes.

Comme vous pouvez le voir dans l'exemple ci-dessus, on peut définir avec plus ou moins de précision le format de données que l'on attend pour chaque « champ », cela permet de sécuriser facilement les données que l'on ajoute dans la base de données. Dans mon cas, le champ email doit : être de type String (type), être obligatoirement renseigné (required), unique (unique) et valide (validate).

La propriété lowercase transforme les majuscules (si l'utilisateur en a utilisé) en minuscules. Il est possible d'implémenter nos propres règles de validation ou utiliser celle d'autres packages grâce à la propriété validate. Dans l'exemple ci-dessus je vérifie si l'email de l'utilisateur est valide grâce à la fonction isEmail du package validator en l'utilisant dans la propriété validate. Beaucoup d'autres propriétés que je n'ai pas cité existent pour définir nos schémas.

Une fois le modèle défini, il est possible d'utiliser des hooks dessus. Sequelize fournit des hooks similaires aux middlewares de Mongoose. Pour ce projet, j'ai utilisé trois types de hooks :

1. les "beforeCreate" hooks
2. les "beforeUpdate" hooks
3. et les "beforeBulkCreate" hooks

Enfin, j'ai également créé des méthodes qui travaillent avec mes modèles. Par exemple, dans mon modèle User, j'ai ajouté une méthode pour vérifier le mot de passe saisi par l'utilisateur lors de la connexion avec le mot de passe stocké dans la base de données. L'avantage de ces méthodes est qu'elles sont utilisables sur les instances de nos modèles, dans les contrôleurs. De plus, comme elles sont créées sur nos modèles, elles ont accès à l'objet this.

Voici un exemple de définition de modèle avec des hooks et une méthode utilisant Sequelize :



```
hooks: {
  beforeCreate: async (user) => {
    const hashedPassword = await bcrypt.hash(user.password, 10);
    user.password = hashedPassword;
  },
  beforeUpdate: async (user) => {
    if (user.changed('password')) {
      const hashedPassword = await bcrypt.hash(user.password, 10);
      user.password = hashedPassword;
    }
  },
  beforeBulkCreate: async (users) => {
    for (const user of users) {
      const hashedPassword = await bcrypt.hash(user.password, 10);
      user.password = hashedPassword;
    }
  },
},
});

User.prototype.checkPassword = async function (password) {
  return await bcrypt.compare(password, this.password);
};

module.exports = User;
```

## 1.2 - Développement du site web

Pour développer ce site web j'ai utilisé la bibliothèque ReactJS ainsi que le framework NextJS. Arborescence

La structure du projet que j'ai utilisé correspond à celle que NextJS propose lorsque l'on crée un projet avec la commande create-next-app.

Mes composants réutilisables (boutons, inputs etc...) dans un dossier components. Mes pages sont dans un dossier pages. Un dossier public me sert à ranger les images et polices que l'application utilise. Un dossier styles contient les différents fichiers CSS de mes pages. Enfin j'ai un dossier store qui contient mon Context ainsi que mes hookcustoms.

### Routage

NextJS rend le routage de l'application super simple. En effet, il suffit de créer une page dans le dossier pages de l'application pour que cette dernière soit accessible comme route. Si je crée un fichier `pages/test.js`, il sera accessible à l'adresse : `adressedusite.com/test`. Un fichier nommé index.js est accessible à l'adresse `adressedusite.com/`.

Il est également possible de créer des sous-dossiers dans le dossier pages et les pages créées dans ces sous-dossiers seront accessibles à l'adresse : `adressedusite.com/nomDuSousDossier/nomDeLaPage`.

C'est d'ailleurs ce que l'on appelle des "nested routes".

Enfin, il est possible de créer des routes dynamiques. Pour ce faire, il faut tout simplement nommer un fichier entre crochet []. Par exemple, dans mon dossier pages/serveurs/, j'utilise une page nommée [dashboard].js. Ainsi lorsque l'on tape l'url "nomdemonsite.com/admin/dashboard", la page [dashbord].js .

### Construction des pages

Avec NextJS, étant un framework de ReactJS, une page est tout simplement un composant React qui return du HTML.

Il existe deux principaux types de composant avec React : les Functional Components qui sont en fait des fonctions qui return un élément React (JSX) et les Class Components qui sont des classes qui return du JSX. Avant la version 16.8 de React il était préférable d'utiliser les Class Components car les Functional Components ne permettaient pas d'utiliser des "states" (états), qui sont un gros point fort de React. Depuis la 16.8 et l'introduction des React Hooks, il est possible d'utiliser des states (avec le hooks useState notamment).

De par sa simplicité d'écriture, les Functional Components sont donc devenus de plus en plus utilisés et c'est ce type de composant que j'ai choisi d'utiliser pour les pages et composants de mon application web Chat\_sport\_us.

```
<>
<Head>
<title>Dashborad</title>
<link rel="icon" href="/favicon.ico" />
</Head>
<Box sx={{ display: 'flex' }}>
  <CssBaseline />
  <AppBar position="fixed" open={open}>
    <Toolbar>
      <IconButton
        color="inherit"
        aria-label="open drawer"
        onClick={handleDrawerOpen}
        edge="start"
        sx={{ mr: 2, ... (open && { display: 'none' }) }}
      />
      <MenuIcon />
    </IconButton>
    <Typography variant="h6" noWrap component="div">
      Dashboard admin
    </Typography>
  </Toolbar>
</AppBar>
<Drawer
  sx={{
    width: drawerWidth,
    flexShrink: 0,
    '& .MuiDrawer-paper': {
      width: drawerWidth,
      boxSizing: 'border-box',
    },
  }}
  variant="persistent"
  anchor="left"
  open={open}
>
  <DrawerHeader>
    <IconButton onClick={handleDrawerClose}>
      {theme.direction === 'ltr' ? <ChevronLeftIcon /> : <ChevronRightIcon />}
    </IconButton>
  </DrawerHeader>
  <Divider />
  <List>
    {[ 'Stats', 'Utilisateurs', 'Messages', 'Groupes' ].map((text, index) => (
      <ListItem key={text} disablePadding>
        <ListItemButton onClick={handleButton}>
          <ListItemIcon>
            {index == 0 ? <BarChartTwoToneIcon /> : index == 1 ? <GroupTwoToneIcon /> : index == 2 ? <MailIcon /> : index == 3 ? <GroupsTwoToneIcon /> : null}
          </ListItemIcon>
          <ListItemText primary={text} />
        </ListItemButton>
      </ListItem>
    ))}
  </List>
  <Divider />
</List>
```

Le HTML est utilisé pour construire nos pages. Cependant NextJS fournit également certains composants. Dans l'extrait ci-dessus vous pouvez voir une balise `<Head>`. C'est un composant proposé par Next permettant d'ajouter des éléments dans la balise head de notre page.

Dans mon exemple, je

l'utilise pour définir le titre de ma page et le lien vers un favicon. Beaucoup d'autres composants très utiles sont proposés comme :

- `<Image />` pour insérer une image et l'optimiser
- `<Link />` pour faire des liens entre les pages

Pour styliser mes pages et mes composants, j'utilise des fichiers CSS nommés en suivant la convention suivante : ``[name].module.css``. J'importe ces derniers dans mes pages pour pouvoir les utiliser. Les couleurs de certains éléments de mon site changent en fonction de mon state global redux (qui est passé en props à mes composants).

Pour ce faire, j'ai utilisé du CSS-in-JS directement à l'intérieur des balise grâce à l'attribut style. Ce qui m'a donc permis d'utiliser mes props.

## Pre-rendement des pages:

Le principal avantage de NextJS est qu'il permet pre-render les pages. Ce que cela signifie c'est que Next.js génère du HTML pour chaque page à l'avance, au lieu de tout faire par JavaScript côté client. Le pré-rendu peut entraîner de meilleures performances et un meilleur référencement.

NextJS propose deux méthode de pré-rendement :

- Static Generation : le HTML est généré au moment de la construction de la page et est réutilisé à chaque demandes
- Server-side Rendering : le HTML est généré à chaque requête

## **2. Précisez les moyens utilisés :**

Pour réaliser ce projet j'ai utilisé une stack full Javascript. **JS**

### Technologies utilisées pour le back :



J'ai également utilisé Node JS comme environnement d'exécution.



Pour développer l'API j'ai utilisé Express JS qui est un framework de NodeJS. Très léger, il apporte peu de surcouches pour garder des performances optimales et une exécution rapide.



Pour la base de données, j'ai décidé d'utiliser Sequelize, qui est un ORM (Object-Relational Mapping) pour les bases de données relationnelles. Sequelize permet de représenter les données sous forme d'objets JavaScript, et de les mapper directement sur les tables de la base de données. Il prend en charge plusieurs dialectes de base de données relationnelles tels que MySQL, PostgreSQL, SQLite, etc.

## DOSSIER PROFESSIONNEL <sup>(DP)</sup>

### Technologies utilisées pour le front (site web) :



J'ai utilisé du HTML pour structurer mon site et du CSS pour le styliser.



React est une librairie qui permet de créer des Single Page App, en créant un DOM virtuel. Avec, il est possible de décomposer nos pages en petits composants indépendants, intelligents et réutilisables, et tout cela avec du simple JavaScript.



NextJS m'a permis d'utiliser React tout en gardant un référencement optimal. En effet, ce dernier propose diverses fonctionnalités permettant d'optimiser au maximum le rendu de mes pages, favorisant grandement le référencement SEO.

### 3. Avec qui avez-vous travaillé ?

J'ai travaillé avec [Aurélien ADJIMI](#), [Yonathan DARMON](#)

### 4. Contexte

Nom de l'entreprise, organisme ou association ▶

*Laplateforme*

Chantier, atelier, service ▶

***Le dashboard du Api-chat\_sport***

Période d'exercice ▶ Du : 02/04/2023 au : 02/05/2023

### 5. Informations complémentaires (facultatif)

## Activité-type 3

### Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

#### Exemple n° 1 - Développement de l'application mobile Chat\_sport

---

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

Durant ma formation à La Plateforme\_, j'ai créé Chat\_sport , une application mobile de chat de serveurs axée sur le sport américain . Je vous ai présenté la maquette de l'application ainsi que le site web, plus haut dans ce dossier. Cette application utilise la même API que le site web que je vous ai présenté précédemment. Cette dernière a les mêmes fonctionnalités que le site web.

Elle nous servira principalement à nous démarquer de la concurrence, en proposant un moyen aux utilisateurs de chatter rapidement de n'importe où et sans avoir un ordinateur sous la main. Je vais donc vous présenter dans cette section le développement de l'application mobile.

Arborescence React laisse la liberté sur la façon d'architecturer un projet. Cependant, il présente deux approches populaires : - grouper les fichiers par fonctionnalités - grouper les fichiers par type. La structure du projet que j'ai utilisé correspond à un regroupement par type de fichiers. J'ai rangé tous mes composants réutilisables (boutons, inputs etc...) dans un dossier components.

Mes pages sont dans un dossier pages. Un dossier assets me sert à ranger les images et polices que l'application utilise.

Un dossier de navigation qui contient mes différentes stacks de navigation. Mes appels vers l'api sont rangés dans un dossier services.

Pour éviter d'avoir des répétitions de fonctions dans mon code, j'ai regroupé les fonctions que j'utilise le plus souvent dans un dossier utils.

Enfin j'ai un dossier Context qui contient mon Provider context . React vs React Native Ayant déjà eu l'occasion de travailler avec ReactJS dans d'autres projets personnels, l'univers de React Native ne m'a pas du tout paru étranger.

ReactJS est le cœur de React Native, et il incarne tous les principes et la syntaxe de React. Les seules différences techniques sont dues aux plateformes sur lesquelles ils sont utilisés. Le code du navigateur dans React est généré via Virtual DOM tandis que React Native utilise des API natives pour générer les composants sur mobile. Par exemple : React utilise le HTML alors que React Native a ses propres composants. Une balise en HTML peut être comparée à

une balise dans React Native. Une deuxième différence est le fait que React Native n'utilise pas de CSS non plus. Pour styliser nos composants, on utilise l'API d'animation fournie avec React Native et plus particulièrement la balise StyleSheet.

### Utilisation de API-Context :

Plusieurs fonctionnalités de mon application ont nécessité l'utilisation de Context. Context est une fonctionnalité de React qui permet de créer un ou plusieurs "contexts". Pour citer une fonctionnalité ayant recours à l'utilisation de Context, on peut prendre comme exemple le **Login Screen** de l'application. En effet, la complexité de cette fonctionnalité réside dans le fait que je veux pouvoir définir l'accès au users selon la connexion faite par l'utilisateur et l'utiliser à plusieurs endroits de l'application, donc plusieurs composants (quasiment partout). L'utilisation d'un context global fut donc la solution la plus adaptée.

J'ai donc créé un **AuthContext** pour mon application qui est en quelque sorte le contexte global de mon application. Ce contexte contient les données du **LoginScreen** donc les informations concernant mon users est accessible depuis tous les composants de l'application qui en ont besoin. Ainsi, chaque composant peut accéder aux données **Users** à travers le contexte et mettre à jour l'état et l'authentification en utilisant les méthodes fournies par le contexte.

```
import React, { createContext, useEffect, useState } from "react";
import axios from "axios";
import * as SecureStore from "expo-secure-store";
import { BASE_URL } from "../config";
import jwtDecode from "jwt-decode";

export const AuthContext = createContext();

const AuthProvider = ({ children }) => {
  const [userToken, setUserToken] = useState(null);
  const [isLoading, setIsLoading] = useState(false);
  const [userInfo, setUserInfo] = useState(null);
  const [test, setTest] = useState("je suis un développeur millionnaire");

  const login = async (email, password) => {
    setIsLoading(true);
    try {
      const response = await axios.post(`${BASE_URL}/auth/login`, {
        email,
        password,
      });
      const { access_token } = response.data;
      await SecureStore.setItemAsync("userToken", access_token);
      const decodedToken = jwtDecode(access_token);
      setUserInfo(decodedToken);
      setUserToken(access_token);
    } catch (error) {
      console.error("Erreur de connexion", error);
    } finally {
      setIsLoading(false);
    }
  };
};
```

### Configuration de la fonction AuthContext en important CreateContext

L'utilisation de Context simplifie la gestion de l'état global de mon application, en évitant d'avoir à passer les données **Users** de parent en parent jusqu'aux composants qui en ont besoin. Au lieu de cela, j'ai simplement besoin d'importer le contexte dans les composants concernés et d'utiliser les données du **Userinfo** directement à partir du contexte.



```
return (  
  <AuthContext.Provider  
    value={{  
      test,  
      userToken,  
      userInfo,  
      isLoading,  
      login,  
      logout,  
    }}  
  >  
    {children}  
  </AuthContext.Provider>  
);  
};  
  
export default AuthProvider;
```

Context offre également des avantages en termes de performance, car il évite les rendus inutiles des composants qui n'ont pas besoin d'être mis à jour lorsque je me déconnecte. Les composants peuvent s'abonner uniquement aux données spécifiques dont ils ont besoin à partir du contexte, ce qui réduit les mises à jour inutiles et améliore les performances de l'application.

En utilisant Context, j'ai pu gérer de manière efficace et flexible la persistance des données de mon application, ainsi que d'autres fonctionnalités nécessitant un état global partagé entre les composants.

```
You, 1 second ago | 1 author (You)  
import React from "react";  
import AppNav from "../src/navigation/AppNav";  
import AuthProvider from "../src/Context/AuthContext";  
  
const App = () => {  
  return (  
    <AuthProvider>  
      <AppNav />  
    </AuthProvider>  
  );  
};  
You, 2 months ago • 2nde commit ...  
  
export default App;
```

Exemple simplifié du déroulement des étapes pour le flux d'authentification :

```
You, 1 second ago | 1 author (You)
import { ActivityIndicator, View } from "react-native";
import AppStack from "../AppStack";
import AuthStack from "../AuthStack";
import React, { useContext } from "react";
import { NavigationContainer } from "@react-navigation/native";
import { AuthContext } from "../Context/AuthContext";

function AppNav() {
  const { isLoading, userToken } = useContext(AuthContext);
  You, 1 second ago • Uncommitted changes
  if (isLoading) {
    return (
      <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
        <ActivityIndicator size={"large"} />
      </View>
    );
  }

  return (
    <NavigationContainer>
      {userToken !== null ? <AppStack /> : <AuthStack />}
    </NavigationContainer>
  );
}

export default AppNav;
```

1. L'utilisateur accède à l'application et est présenté avec une page de connexion.
2. L'utilisateur saisit ses identifiants (nom d'utilisateur et mot de passe) dans les champs de connexion.
3. Lorsque l'utilisateur soumet le formulaire de connexion, un événement est déclenché et le client envoie une requête POST au serveur avec les informations d'identification de l'utilisateur.
4. Le serveur reçoit la requête de connexion et vérifie les informations d'identification fournies par l'utilisateur. Cela peut impliquer la vérification des informations dans une base de données, le chiffrement du mot de passe pour comparaison, etc.
5. Si les informations d'identification sont valides, le serveur génère un jeton d'authentification. Le jeton est généralement un jeton JWT (JSON Web Token) qui contient des informations sur l'utilisateur et une signature pour vérifier son authenticité.

```
const isLoggedIn = async () => {
  try {
    setIsLoading(true);
    const userToken = await SecureStore.getItemAsync("userToken");
    if (userToken) {
      setUserToken(userToken);
      const decodedToken = jwtDecode(userToken);
      setUserInfo(decodedToken);
    }
  } catch (error) {
    console.log(`Erreur lors de la vérification de la connexion : ${error}`);
  } finally {
    setIsLoading(false);
  }
};
```

6. Le serveur renvoie le jeton d'authentification en tant que réponse à la requête de connexion.
7. Le client reçoit le jeton d'authentification et le stocke localement, par exemple dans le stockage local (secureStore) .
8. À partir de ce point, le jeton d'authentification est utilisé pour authentifier les requêtes ultérieures de l'utilisateur. Lorsqu'un utilisateur souhaite accéder à une ressource protégée, il envoie le jeton d'authentification avec la requête.

```
useEffect(() => {
  isLoggedIn();
}, []);

return (
  <AuthContext.Provider
    value={{
      test,
      userToken,
      userInfo,
      isLoading,
      login,
      logout,
    }}
  >
    {children}
  </AuthContext.Provider>
);
};
```

9. Le serveur vérifie le jeton d'authentification à chaque requête pour s'assurer qu'il est valide et non expiré. Si le jeton est valide, le serveur autorise la requête et renvoie les données demandées à l'utilisateur. Sinon, le serveur renvoie une erreur d'authentification et demande à l'utilisateur de se reconnecter.
10. Lorsque l'utilisateur se déconnecte ou que le jeton expire, le jeton d'authentification est supprimé du stockage local du client, le déconnectant ainsi de l'application.

Il convient de noter que cet exemple simplifié peut varier en fonction des exigences spécifiques de l'application et des technologies utilisées. Cependant, il fournit une vue d'ensemble générale du flux d'authentification.

### **Exécuter les plans de tests d'une application :**

Pour écrire des tests dans une application Node.js, j'ai utilisé des bibliothèques de test populaires comme Jest et Supertest . Dans cet exemple, j'ai utilisé Jest test pour créer des tests pour une route simple en utilisant le module d'assertions natif de Node.js.

1. J'ai créé un fichier de test pour la route "user". un fichier nommé "users.spec.js".
2. Dans ce fichier de test, j'ai importé les modules nécessaires (par exemple, le fichier de route et le module d'assertions de Node.js)

```
const app = require("../app");
const request = require("supertest");

describe("GET /users", () => {
  it("should return a status 200", async () => {
    const response = await request(app).get("/users");
    expect(response.statusCode).toBe(200);
  });
});

describe("GET /users/firstlast", () => {
  it("should return a status 200", async () => {
    const response = await request(app).get("/users/firstlast");
    expect(response.statusCode).toBe(200);
  });
});

describe("GET /users/:id", () => {
  it("should return a status 200", async () => {
    const response = await request(app).get("/users/123");
    expect(response.statusCode).toBe(200);
  });
});

describe("POST /users", () => {
  it("should return a status 201 and create a user", async () => {
    const response = await request(app).post("/users").send({
      username: "john_doe",
      password: "password",
    });
    expect(response.statusCode).toBe(201);
    expect(response.text).toBe("User Created");
  });
});
```

## 2. Précisez les moyens utilisés :

Figma : maquettes

# DOSSIER PROFESSIONNEL <sup>(DP)</sup>

Trello : gestion de projet  
git  
github  
git kraken

## 3. Avec qui avez-vous travaillé ?

Pour ce projet j'ai travaillé avec Aurélien Adjami, [Yonathan DARMON](#)

## 4. Contexte

Nom de l'entreprise, organisme ou association ▶ *LAPLATEFORME*

Chantier, atelier, service ▶ *Chat\_sport\_us*

Période d'exercice ▶ Du : *02/01/2023* au : *11/01/2023*

## 5. Informations complémentaires (facultatif)

## Activité-type 3 Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

Exemple n° 1 - Cliquez ici pour entrer l'intitulé de l'exemple

---

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

Les étapes que vous avez mentionnées sont correctes et couvrent les bases du déploiement d'une API REST sur PulseHeberg. Cependant, je remarque que certaines étapes spécifiques à PulseHeberg sont absentes. Voici une version mise à jour des étapes en tenant compte de cela :

1. j'ai choisi un plan d'hébergement approprié sur PulseHeberg en fonction de mes besoins et de mon budget. je me suis assuré que le plan choisi offre suffisamment de ressources pour exécuter l'API REST, comme la puissance de traitement, la mémoire et le stockage.
2. le choix d'un nom de domaine et configurer un sous-domaine pour votre API REST.
3. je me suis assuré d'avoir les fichiers de mon application API REST prêts à être déployés. Ces fichiers comprennent généralement mon code source pour mon API, mes fichiers de configuration, mes fichiers de routage, mes contrôleurs, mes modèles de données, ainsi que toutes les dépendances ou bibliothèques nécessaires au bon fonctionnement de mon application.
4. J'utilise le gestionnaire de fichiers intégré à PulseHeberg, une option de transfert de fichiers telle que l'outil Terminus pour télécharger mes fichiers sur le serveur. Cela me permet de transférer mes fichiers d'application, les dépendances, mes bibliothèques et tout autre élément nécessaire sur le serveur d'hébergement.
5. Je configure les paramètres du serveur web (Nginx) pour pointer vers les fichiers de mon API REST. Cela inclut la configuration d'un virtual host, la définition de règles de redirection et d'autres paramètres spécifiques à mon API REST. J'accède généralement à ces paramètres via le tableau de bord PulseHeberg et le dashboard terminus .
6. J'assure également que mon domaine ou mon sous-domaine est correctement configuré pour pointer vers mon API REST hébergée sur PulseHeberg.
7. Une fois tout configuré, je teste mon API REST pour m'assurer qu'elle fonctionne correctement. J'effectue des requêtes à partir d'un client REST, comme Postman, ou directement depuis un navigateur pour vérifier si j'obtiens les réponses attendues.

Il est également essentiel de mettre en place des mesures de sécurité pour mon API REST, telles que



# DOSSIER PROFESSIONNEL <sup>(DP)</sup>

l'authentification, l'autorisation et la protection contre les attaques courantes. Cela garantit la confidentialité et l'intégrité des données échangées avec mon API.

## 2. Précisez les moyens utilisés :

Pour mener à bien mon déploiement j'ai utilisé les outils suivants :

- Pulseheberg
- Debian
- Nginx
- NODE JS
- Sequelize
- MariaDB
- phpMyAdmin
- Terminus

## 3. Avec qui avez-vous travaillé ?

J'ai travaillé seul sur le déploiement.

## 4. Contexte

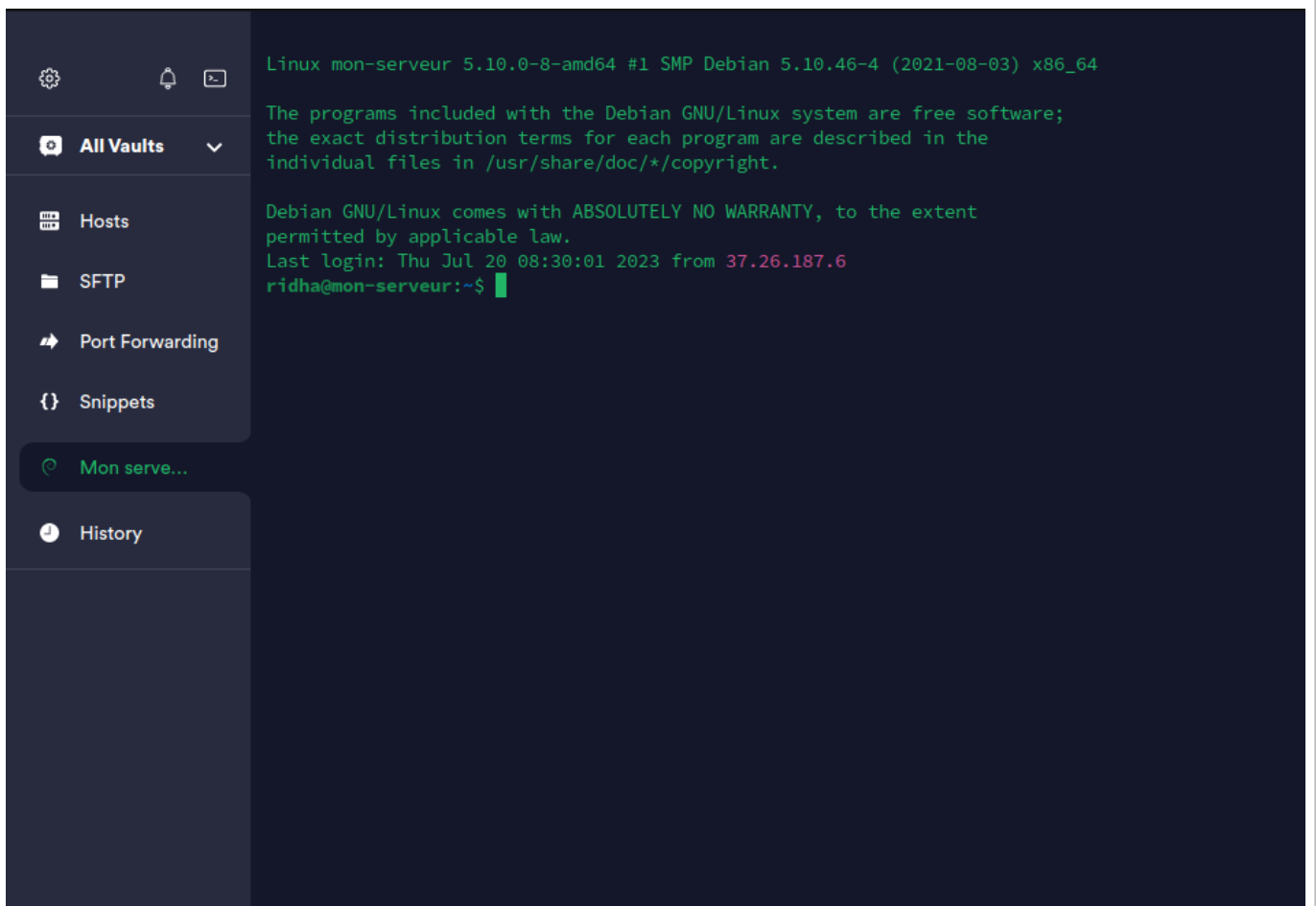
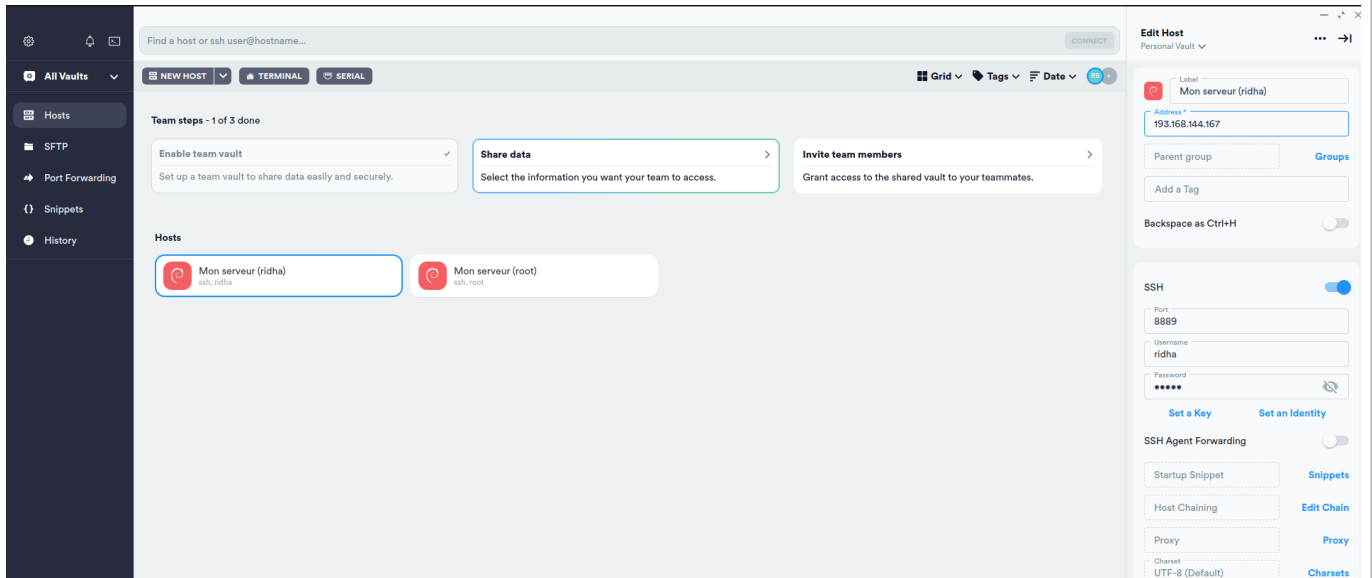
Nom de l'entreprise, organisme ou association ▶ *Laplateforme*

Chantier, atelier, service ▶ ***Le déploiement du Api-chat\_sport***

Période d'exercice ▶ Du : *02/04/2023* au : *02/05/2023*

## 5. Informations complémentaires (facultatif)

# DOSSIER PROFESSIONNEL (DP)



```
0 directories, 0 files
ridha@mon-serveur:~$ tree -d -L 1
.
├── api
```

```
ridha@mon-serveur:~$ tree -d -L 2
.
├── api
│   ├── JWT
│   ├── bdd
│   ├── controllers
│   ├── models
│   ├── node_modules
│   ├── public
│   ├── routes
│   ├── tests
│   └── tmp
└──
```

10 directories  
ridha@mon-serveur:~\$

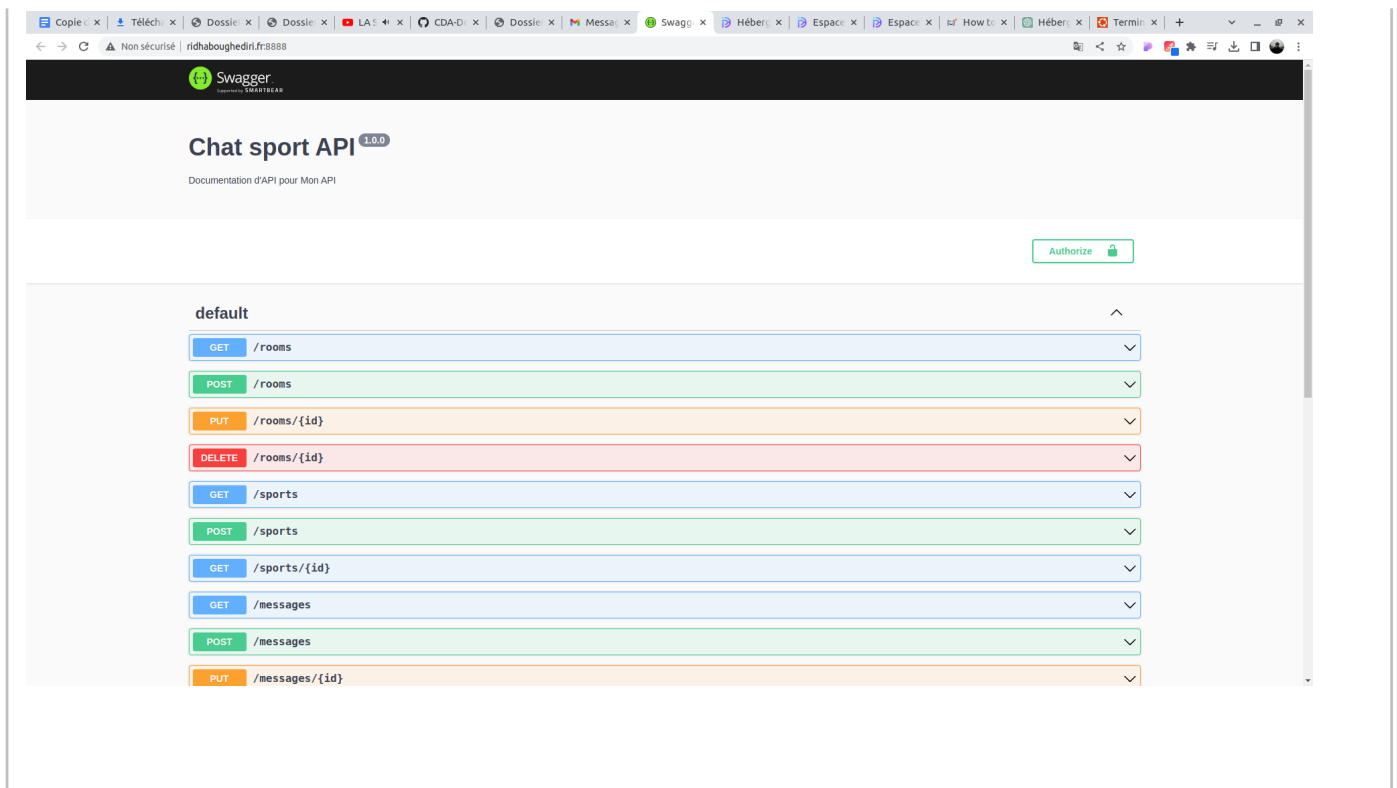
```
ridha@mon-serveur:~/api/tests$ tree
.
├── auth.spec.js
├── messages.spec.js
├── nbaTeams.spec.js
├── nflTeams.spec.js
├── nhlTeams.spec.js
├── privateMessage.spec.js
├── rooms.spec.js
└── users.spec.js
```

0 directories, 8 files  
ridha@mon-serveur:~/api/tests\$

```
ridha@mon-serveur:~/api/routes$ tree
.
├── app-mobile-chat.code-workspace
├── auth.js
├── messages.js
├── nbaTeams.js
├── nflTeams.js
├── nhlTeams.js
├── privateMessage.js
├── roles.js
├── rooms.js
├── sports.js
└── users.js
```

0 directories, 11 files  
ridha@mon-serveur:~/api/routes\$

# DOSSIER PROFESSIONNEL (DP)



---

## DOSSIER PROFESSIONNEL <sup>(DP)</sup>

---

### Titres, diplômes, CQP, attestations de formation

*(facultatif)*

Intitulé	Autorité ou organisme	Date
Cliquez ici.	Cliquez ici pour taper du texte.	Cliquez ici pour sélectionner une date.

### Déclaration sur l'honneur

Je soussigné **RIDHA BOUGHEDIRI** ,  
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 17/07/2023

pour faire valoir ce que de droit.

Signature :



## Documents illustrant la pratique professionnelle

*(facultatif)*

Intitulé
Cliquez ici pour taper du texte.

---

## DOSSIER PROFESSIONNEL <sup>(DP)</sup>

---

---