



MINISTÈRE CHARGÉ
DE L'EMPLOI

DOSSIER PROFESSIONNEL (DP)

<i>Nom de naissance</i>	► BOUASRIA
<i>Nom d'usage</i>	► BOUASRIA
<i>Prénom</i>	► ISMAÏL
<i>Adresse</i>	► 9 bis chemin de Paradis 13500 Martigues

Titre professionnel visé

Concepteur Développeur Informatique

MODALITE D'ACCES :

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

DOSSIER PROFESSIONNEL (DP)

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.

DOSSIER PROFESSIONNEL (DP)

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

Sommaire

Exemples de pratique professionnelle

Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité	p. 5
▶ Maquetter une application - SimpleChat	p. 5
▶ Développer des composants d'accès aux données - SimpleChat	p. 9
▶ Développer la partie front-end d'une interface utilisateur web - SimpleChat	p. 12
▶ Développer la partie back-end d'une interface utilisateur web - SimpleChat	p. 14
▶ Développer une interface utilisateur de type desktop - Shooter Game	p. 17
Concevoir et développer la persistance des données en intégrant les recommandations de sécurité	p. 19
▶ Concevoir une base de données - SimpleChat	p. 19
▶ Mettre en place une base de données - SimpleChat	p. 21
▶ Développer des composants dans le langage d'une base de données - SimpleChat	p. 25
Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité	p. 28
▶ Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement - SimpleChat	p. 29
▶ Concevoir une application - SimpleChat	p. 32
▶ Développer des composants métier - SimpleChat	p. 35
▶ Construire une application organisée en couches - SimpleChat	p. 36
▶ Développer une application mobile - SimpleChat	p. 38
▶ Préparer et exécuter les plans de tests d'une application - SimpleChat	p. 42
▶ Préparer et exécuter le déploiement d'une application - Portfolio	p. 45
Titres, diplômes, CQP, attestations de formation (facultatif)	p. 49
Déclaration sur l'honneur	p. 50

DOSSIER PROFESSIONNEL (DP)

Documents illustrant la pratique professionnelle (*facultatif*)

p. 51

Annexes (*Si le RC le prévoit*)

p. 52

DOSSIER PROFESSIONNEL (DP)

EXEMPLES DE PRATIQUE PROFESSIONNELLE

DOSSIER PROFESSIONNEL (DP)

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

Maquetter une Application ▶ Application mobile SimpleChat

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

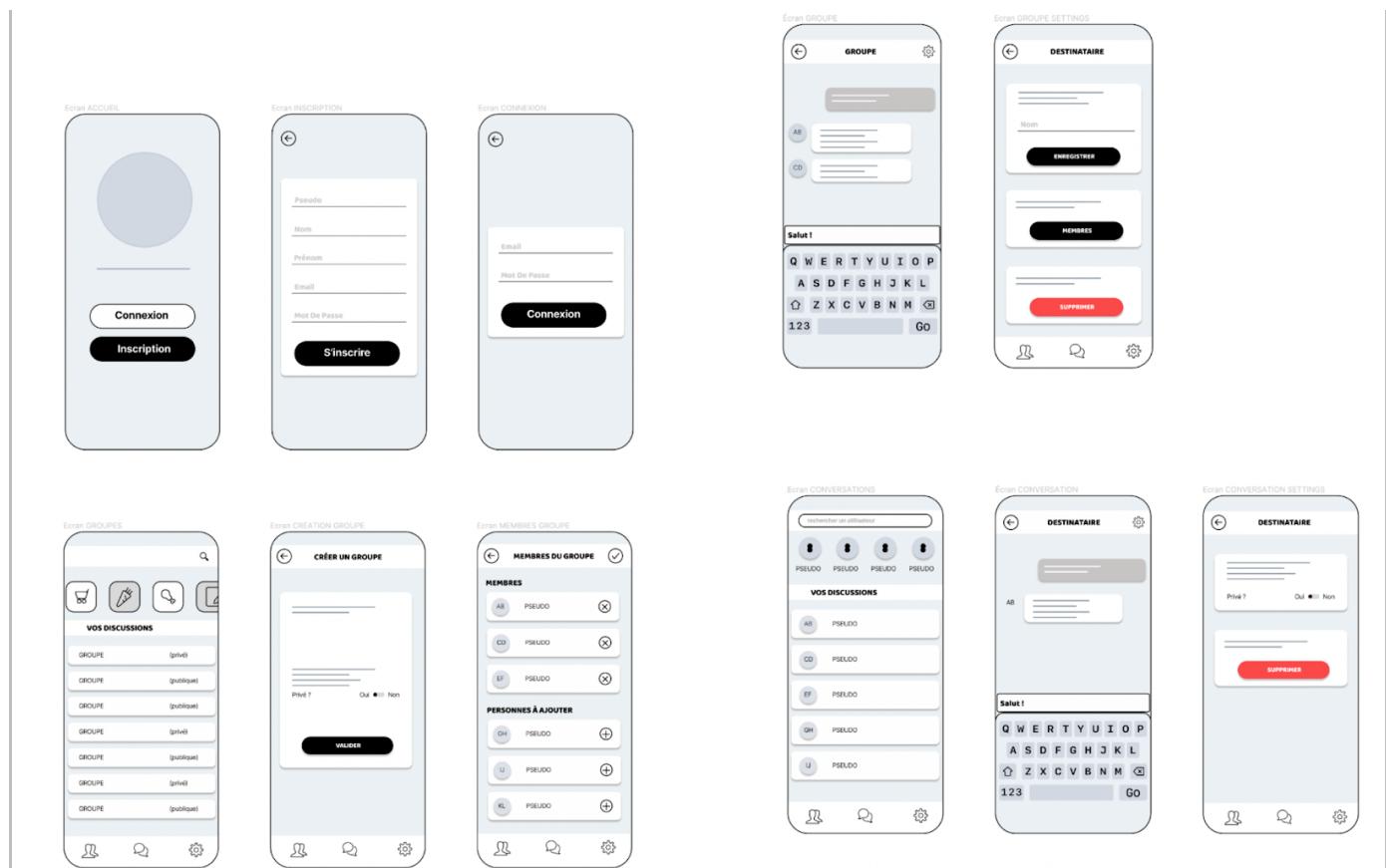
Dans le cadre de notre projet d'application mobil j'ai été amené à réaliser plusieurs maquettes :

Le maquettage front-end se concentre sur l'apparence et l'interaction de ces éléments, en veillant à ce qu'ils soient intuitifs, attrayants et conviviaux pour les utilisateurs. Cela peut inclure des choix de couleurs, des icônes, des polices, des mises en page et des transitions.

Les éléments clés du **maquettage front-end de SimpleChat** sont :

- 1. Page de connexion** : Une interface où les utilisateurs peuvent se connecter à l'application à l'aide de leurs identifiants.
- 2. Liste des conversations** : Une vue qui affiche toutes les conversations disponibles pour l'utilisateur, peut-être sous forme de liste ou de vignettes.
- 3. Interface de création de conversation** : Une page où les utilisateurs peuvent créer de nouvelles conversations en ajoutant des contacts ou en les recherchant.
- 4. Conversation individuelle** : Une vue qui montre les messages échangés entre les utilisateurs d'une conversation spécifique, avec la possibilité d'envoyer de nouveaux messages.
- 5. Profil utilisateur** : Une interface qui permet aux utilisateurs de consulter et de modifier leur profil, y compris leurs informations personnelles et leurs préférences.

DOSSIER PROFESSIONNEL (DP)



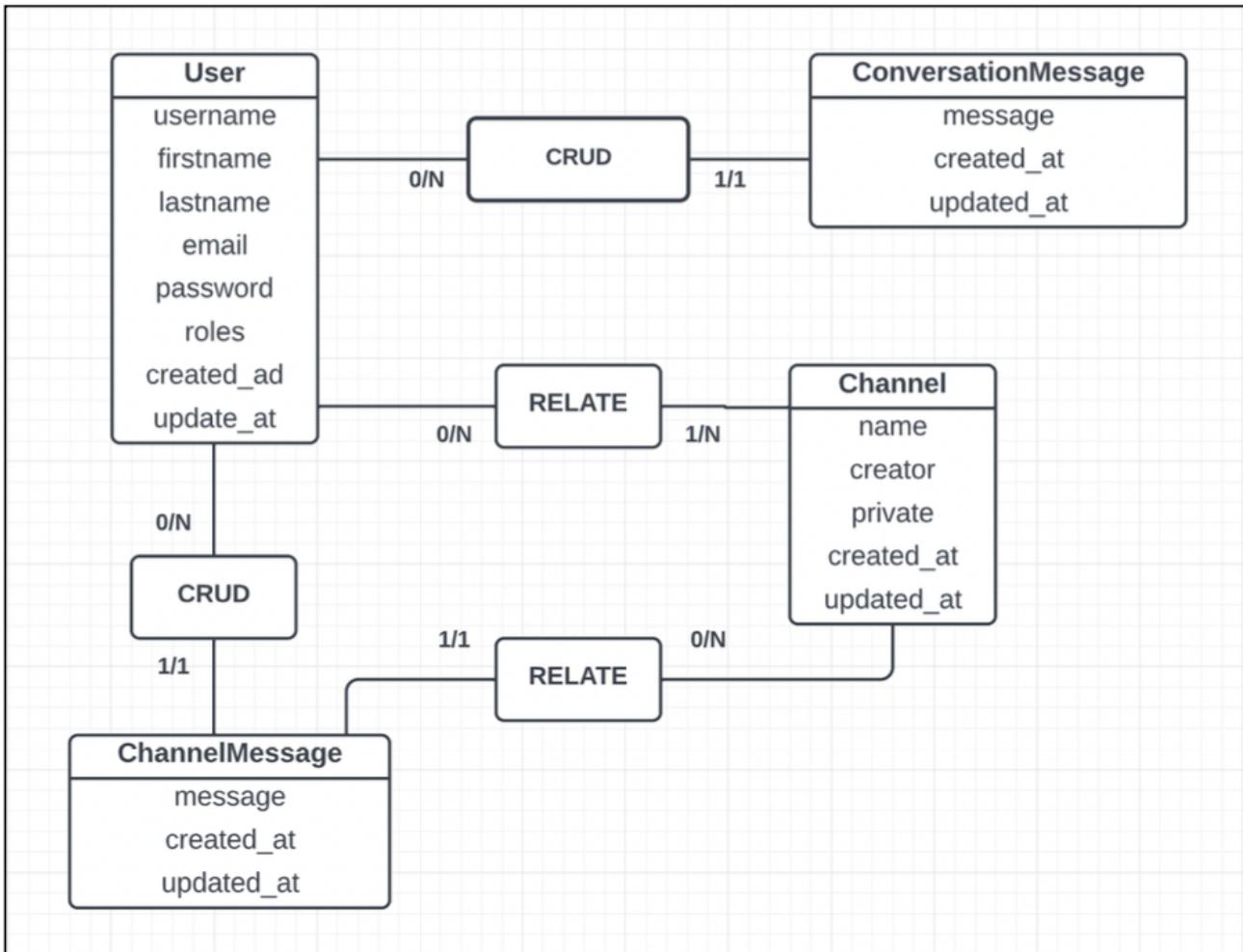
Le maquettage back-end de l'application SimpleChat concerne l'architecture et la logique sous-jacente de l'application. Il s'agit de concevoir la structure des données, les fonctionnalités et les interactions nécessaires pour prendre en charge les fonctionnalités front-end.

Nous nous sommes appuyés sur la méthode “**Merise**” pour concevoir notre base de données SQL. Grâce à l'outil **LucidChart** nous avons donc élaboré :

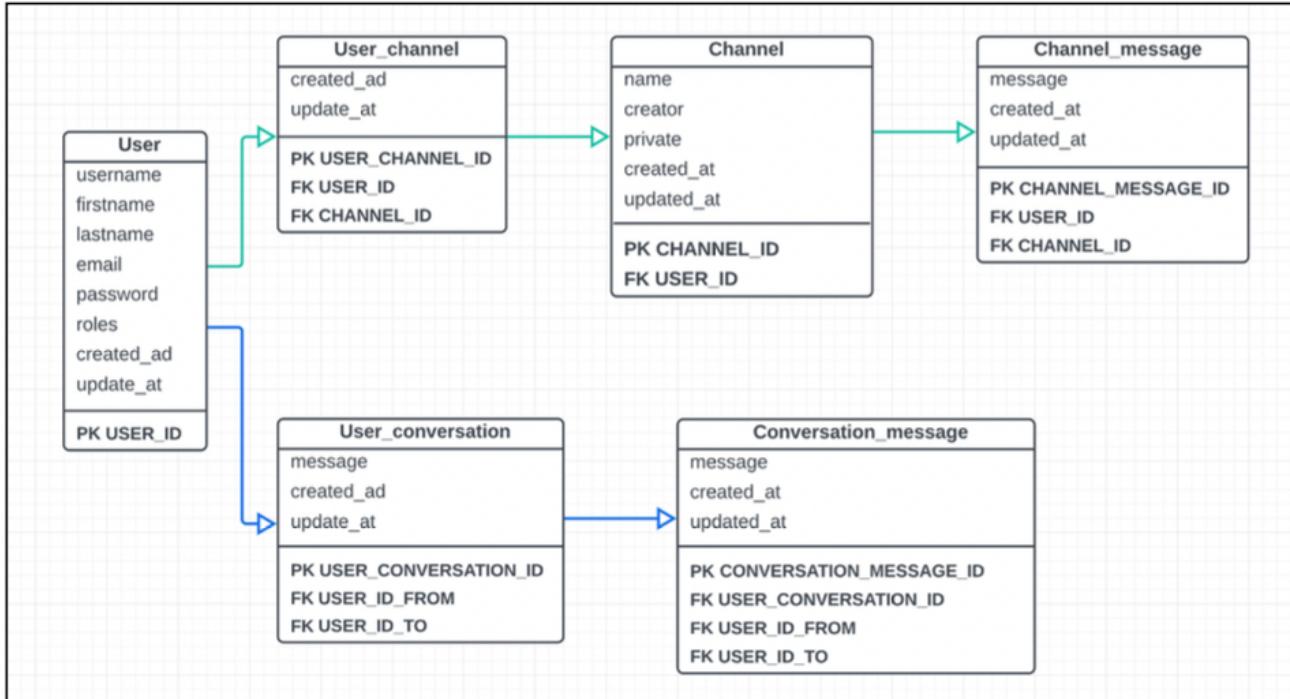
- **Modèle Conceptuel de Données (MCD)**
- **Modèle Logique de Données (MLD)**

DOSSIER PROFESSIONNEL (DP)

MCD



MLD



du

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

- ▶ Figma
- ▶ Lucidchart

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail seul

4. Contexte

Période d'exercice ▶ Du : 02/01/2023 au : 07/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Maquetter une application »

DOSSIER PROFESSIONNEL (DP)

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

Développer des composants d'accès aux données ► Application mobile simpleChat

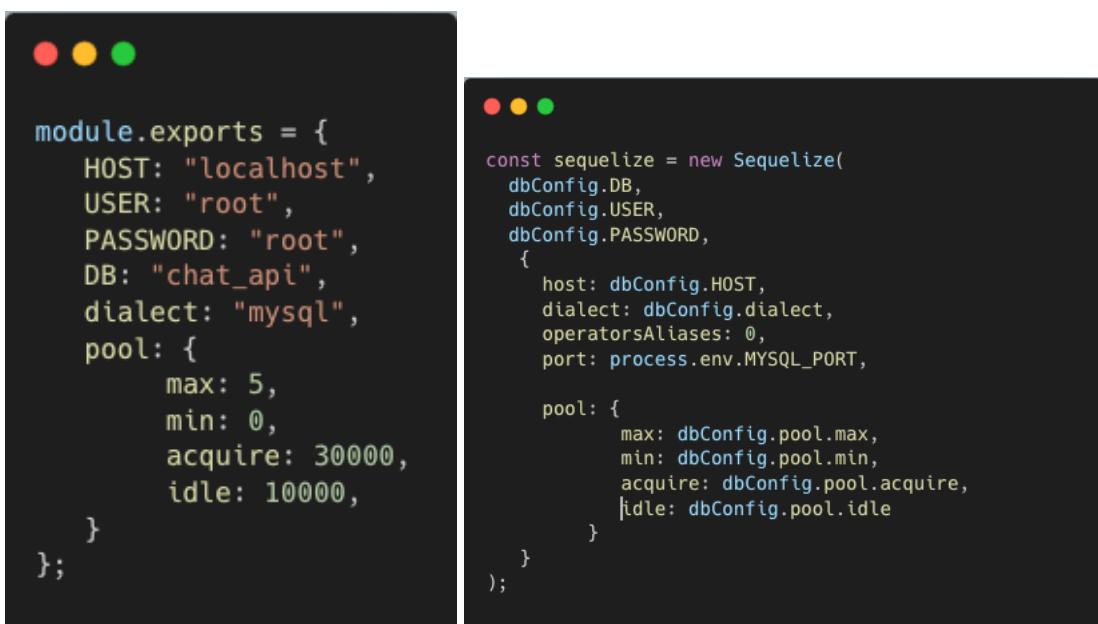
1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

En équipe de 3 personnes, notre objectif était de réaliser une application mobile de chat connectée à des Sockets. Le parcours client se déroule comme suit :

- ▶ Un utilisateur s'inscrit, puis se connecte pour accéder à son tableau de bord personnel.
- ▶ À partir de ce tableau de bord, il peut créer des groupes publics ou privés, rejoindre des groupes publics, inviter des membres dans son groupe et échanger des messages au sein de celui-ci.
- ▶ Il peut également créer des conversations privées avec d'autres utilisateurs pour échanger des messages.
- ▶ Que ce soit pour un groupe ou une conversation privée, les informations sont modifiables, et seuls les auteurs des messages peuvent les modifier ou les supprimer.

Pour développer l'API consommée par l'application mobile, j'ai tout d'abord réalisé un Modèle Conceptuel de Données pour définir les entités nécessaires à la réalisation de l'API.

Ensuite, j'ai configuré et développé la connexion à la base de données en utilisant **Sequelize** et **Express** afin de pouvoir interagir avec celle-ci.

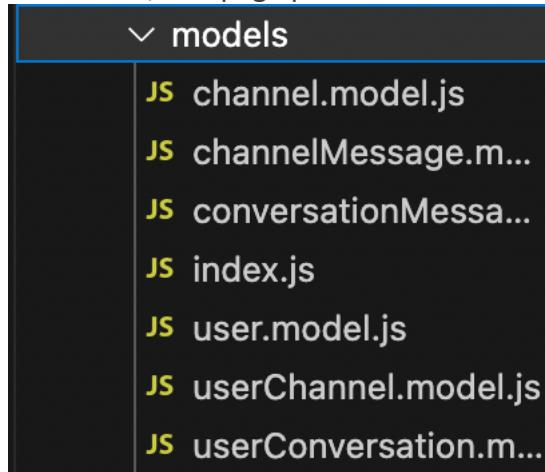


```
module.exports = {
  HOST: "localhost",
  USER: "root",
  PASSWORD: "root",
  DB: "chat_api",
  dialect: "mysql",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000,
  }
};

const sequelize = new Sequelize(
  dbConfig.DB,
  dbConfig.USER,
  dbConfig.PASSWORD,
  {
    host: dbConfig.HOST,
    dialect: dbConfig.dialect,
    operatorsAliases: 0,
    port: process.env.MYSQL_PORT,
    pool: {
      max: dbConfig.pool.max,
      min: dbConfig.pool.min,
      acquire: dbConfig.pool.acquire,
      idle: dbConfig.pool.idle
    }
);
;
```

DOSSIER PROFESSIONNEL (DP)

Une fois le Modèle Conceptuel de Données conceptualisé et la connexion à la base de données opérationnelle, j'ai commencé à développer les différentes entités. Pour cela, j'ai créé un dossier appelé "Model", et à l'intérieur de celui-ci, une page par entité.



Chaque entité a ensuite été typée et associée en tant que clé étrangère selon les besoins.

Une fois les entités créées, j'ai pu développer les composants permettant l'accès aux données dans un dossier nommé "Service". Ce dossier contient également un fichier par entité.

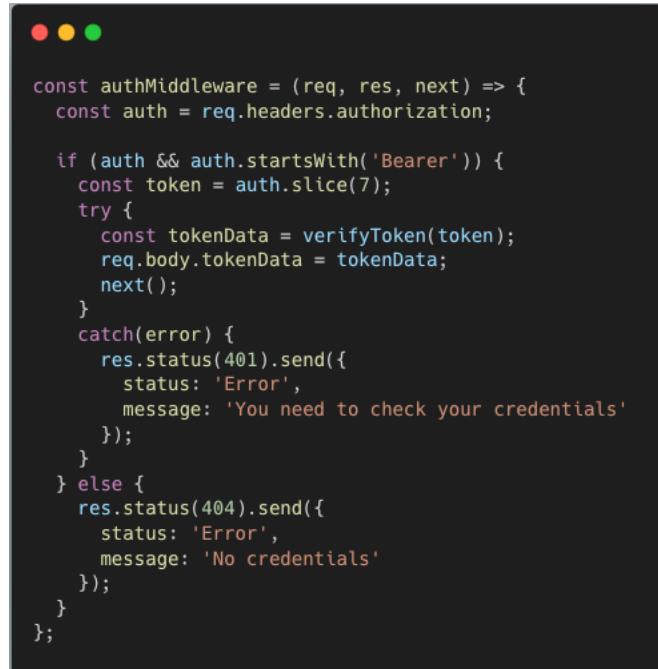
Chaque service présente un ensemble de méthodes asynchrones afin de ne pas gêner le déroulement d'autres fonctions, permettant ainsi différentes actions.

En résumé, les services proposent des méthodes (appelées dans des routeurs) permettant de réaliser des opérations de type **CRUD**, ou des **GET** plus ou moins spécifiques selon divers paramètres tels que l'administrateur, l'utilisateur connecté ou le créateur d'un groupe.

A screenshot of a terminal window. The code shown is a Node.js router configuration. It includes two routes: a GET route for listing channels and a POST route for creating a new channel. Both routes use the 'channel' module to handle the requests. The code uses ES6 syntax, including arrow functions and async/await.

Pour des raisons de sécurité, j'ai mis en place les JSON Web Tokens, dont les méthodes de génération et de vérification permettent de s'assurer que l'utilisateur souhaitant accéder aux données possède les droits requis.

DOSSIER PROFESSIONNEL (DP)



```
const authMiddleware = (req, res, next) => {
  const auth = req.headers.authorization;

  if (auth && auth.startsWith('Bearer ')) {
    const token = auth.slice(7);
    try {
      const tokenData = verifyToken(token);
      req.body.tokenData = tokenData;
      next();
    }
    catch(error) {
      res.status(401).send({
        status: 'Error',
        message: 'You need to check your credentials'
      });
    }
  } else {
    res.status(404).send({
      status: 'Error',
      message: 'No credentials'
    });
  }
};
```

2. Précisez les moyens utilisés :

- ▶ L'éditeur de code **Visual Studio Code** pour la création des méthodes.
- ▶ La **documentation Express**.
- ▶ La **documentations Sequelize**.
- ▶ La **documentation JSON Web Token**.
- ▶ **Postman** pour tester les méthodes des différents services.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail avec mes deux camarades de projets.

4. Contexte

Période d'exercice ▶ Du : 10/01/2023 au : 14/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Développer des composants d'accès aux données »

DOSSIER PROFESSIONNEL (DP)

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

Développer la partie front-end Application mobile SimpleChat
d'une interface utilisateur web ▶

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

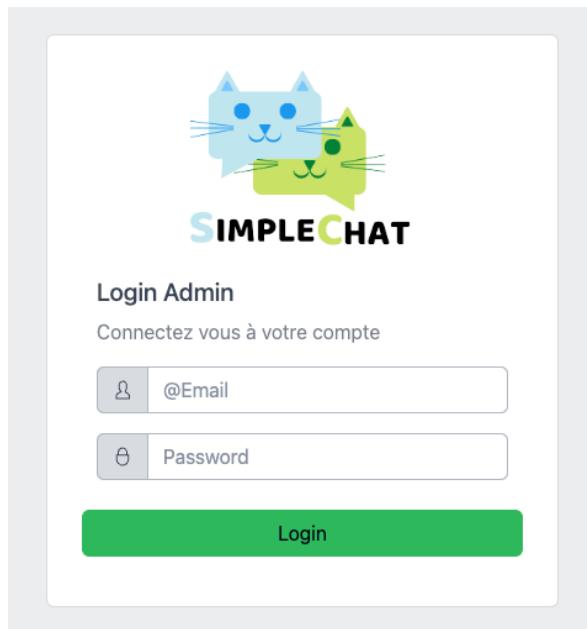
En résumé, le projet consistait à développer une interface web permettant à un administrateur de gérer facilement les données envoyées par les utilisateurs, en complément de l'application mobile développée en **React Native**. Pour cela, la bibliothèque React.js a été choisie, car elle est étroitement liée à la version Native, facilitant ainsi le développement de cette interface web.

Le « **dashboard** » développé offre à l'administrateur une vue sur les utilisateurs, les groupes et les messages échangés au sein de ces groupes. Pour accéder au dashboard, l'administrateur doit se connecter en saisissant son adresse e-mail et son mot de passe, et doit posséder **le rôle ADMIN** dans la base de données pour des raisons de sécurité.

Le dashboard permet à l'administrateur de consulter et de supprimer facilement des groupes publics et privés, des messages échangés par les utilisateurs, ainsi que les utilisateurs eux-mêmes.

Pour réaliser cette application web :

- La commande "**npx create-react-app**" a été utilisée pour créer un projet React.js.
- La commande "**npm start**" est utilisée pour lancer l'application sur un serveur local (localhost :3000).
- Le package "**react-router-dom**" a été installé et utilisé pour créer un routeur et générer différents URL.



DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

- ▶ L'éditeur de code **Visual Studio Code** pour la création des méthodes.
- ▶ La documentation **React**

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail en seul

4. Contexte

Période d'exercice ▶ Du : 25/01/2023 au : 27/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Développer le front-end d'une interface utilisateur web »

DOSSIER PROFESSIONNEL (DP)

Activité-type 1

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

Développer la partie back-end *Application mobile SimpleChat*
d'une interface utilisateur web ▶

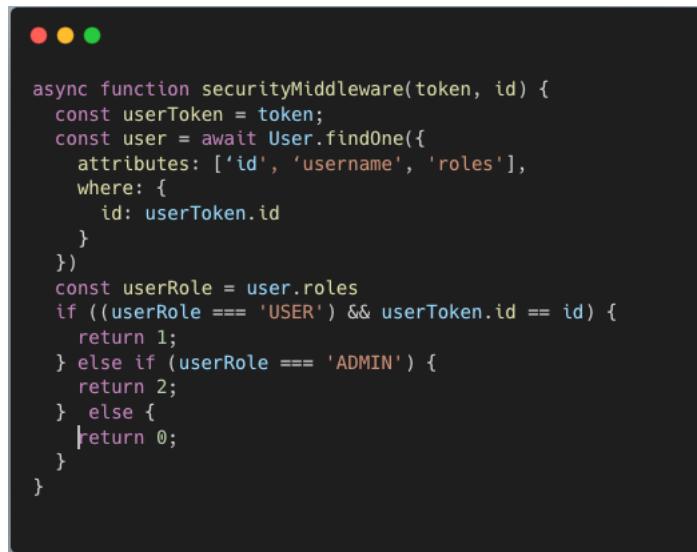
1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Étant donné que le backend de l'interface web est également utilisé par l'application mobile, j'ai dû revoir certains services car l'API était opérationnelle depuis plusieurs jours lorsque j'ai commencé à développer l'interface utilisateur web.

J'ai donc créé de nouveaux services spécifiques à l'administrateur. Pour améliorer l'efficacité, j'ai développé une méthode qui récupère le rôle de l'utilisateur à partir du token envoyé dans l'en-tête grâce à une requête.

Si le rôle de l'utilisateur existe dans la base de données, je le récupère et vérifie immédiatement sa valeur pour retourner l'un des trois résultats suivants :

- 0 s'il n'est pas authentifié,
- 1 s'il possède le rôle USER,
- 2 s'il possède le rôle ADMIN.



```
async function securityMiddleware(token, id) {
  const userToken = token;
  const user = await User.findOne({
    attributes: ['id', 'username', 'roles'],
    where: {
      id: userToken.id
    }
  })
  const userRole = user.roles
  if ((userRole === 'USER') && userToken.id === id) {
    return 1;
  } else if (userRole === 'ADMIN') {
    return 2;
  } else {
    return 0;
  }
}
```

Cette méthode est appelée en tant que middleware depuis le routeur. Cela permet de vérifier rapidement si l'utilisateur qui tente d'accéder à certains endpoints est bien un administrateur. Par exemple, depuis son dashboard, l'administrateur peut consulter la liste de tous les types de groupes, qu'ils soient privés ou publics. Pour cela, j'ai développé une nouvelle méthode ainsi qu'un nouvel endpoint.

DOSSIER PROFESSIONNEL (DP)

```
● ● ●

router.get( '/api/admin/:id/channels', authMiddleware,
async (req, res) => {
channel.adminGetAllChannel(req, res)
});
```

Dans l'exemple ci-dessus, le routeur est généré à l'aide d'Express en utilisant la méthode **Router()** et est ensuite stocké dans la variable "router". La méthode **get()** indique que cet endpoint effectue une requête de **type GET**.

L'URL est spécifiée du côté client, et elle doit être accompagnée d'un ID pour effectuer la requête. Cela permet de vérifier rapidement s'il s'agit bien d'un administrateur. Un premier middleware est appelé pour vérifier que le token est présent dans l'en-tête, puis le valider et extraire les données qu'il contient.

Chaque service est nommé en fonction de sa fonctionnalité, et il prend systématiquement l'ID de l'utilisateur pour les requêtes privées.

```
● ● ●

async function adminGetAllChannel(req, res){
  const id = req.params.id;
  const userToken = req.body.tokenData;
  const userControl = await securityMiddleware(userToken, id)
    if(userControl === 2) {
      await Channel.findAll({
        order: [
          ['created_at', 'DESC'],
        ],
        attributes: [
          'id',
          'name',
          'creator',
          'private',
          'created_at',
        ],
      })
      .then(channels => { res.status(200).send({
        status: 'Success',
        data: channels,
      })
    })
    .catch(err => {
      res.status(500).send({ status: 'Error', message: err.message });
    });
    } else {
      res.status(500).send({
        status: 'Error',
        message: 'You're not authorized',
      })
    }
}
```

DOSSIER PROFESSIONNEL (DP)

Chaque méthode est asynchrone et contient une structure conditionnelle basée sur le rôle de l'utilisateur. Dans le premier cas, seule l'administrateur peut accéder à cette requête. Dans le cas suivant, le service est accessible à la fois à un utilisateur dont l'ID et le token correspondent, ainsi qu'à un administrateur.



```
if(userControl === 1 || userControl === 2) {  
... }
```

L'administrateur dispose ainsi des requêtes sécurisées, permettant d'effectuer des opérations de type CRUD en base de données.

2. Précisez les moyens utilisés :

- ▶ L'éditeur de code **Visual Studio Code**.
- ▶ La **documentation Express**.
- ▶ La **documentation JSON Web Token**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail avec mon groupe de projet

4. Contexte

Période d'exercice ▶ Du : 17/01/2023 au : 20/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Développer le back-end d'une interface utilisateur web »

DOSSIER PROFESSIONNEL (DP)

Activité-type 1 Cliquez ici pour entrer l'intitulé de l'activité

Développer une interface *Shooter Game – jeu en python utilisateur de type desktop*

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

Le projet "Shooter Game" consiste en la création d'un jeu de tir en Python où le joueur contrôle un personnage se déplaçant horizontalement et doit éliminer des ennemis. L'objectif est de démontrer la capacité du développeur à créer une interface graphique de type desktop.

Le jeu comprendra les éléments suivants :

1. Interface utilisateur : Une interface graphique attrayante et conviviale sera développée à l'aide d'une bibliothèque Python : **Pygame**. L'interface permettra au joueur de voir son personnage, les ennemis et d'autres éléments du jeu.

```
# importer Pygame et l'initialiser
import pygame
import math
from classes.game import Game

pygame.init()

# Générer la fenêtre du jeu
pygame.display.set_caption("Shooter Game")
screen = pygame.display.set_mode((1080, 720))

# Importer charger à l'arrière plan
background = pygame.image.load('./assets/bg.jpg')

# Charger notre jeu
game = Game()

running = True
```



2. Contrôle du personnage : Le joueur pourra contrôler le personnage en utilisant les touches du clavier pour se déplacer horizontalement et tirer sur les ennemis.

3. Gestion des ennemis : Des ennemis apparaîtront à l'écran, se déplaceront et le joueur devra les éliminer en les touchant avec ses tirs.

4. Gestion des collisions : Le jeu générera les collisions entre les tirs du joueur et les ennemis, ainsi que la détection de collisions entre le joueur, les ennemis et les limites de l'écran.

DOSSIER PROFESSIONNEL (DP)



En développant ce projet, j'ai pu démontrer mes compétences en matière de développement d'interfaces graphiques en Python, de gestion des entrées utilisateur, de mouvement d'objets à l'écran, de gestion des collisions et de suivi des scores sur une application de jeu interactive et attrayante de type desktop.

2. Précisez les moyens utilisés :

- ▶ L'éditeur de code **PyCharm CE**
- ▶ La documentation **Python**
- ▶ La documentation **Pygame**

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail seul

4. Contexte

Période d'exercice ▶ Du : 12/06/2023 au : 16/06/2023

5. Informations complémentaires (facultatif)

Ce projet m'a permis de valider la compétence « Développer une interface type desktop »

DOSSIER PROFESSIONNEL (DP)

Activité-type 2

Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

Concevoir une base de données ▶ Application mobile – Simple Chat

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

Avant de développer une API ou le backend d'une application, il est crucial de définir avec précision les entités que nous souhaitons stocker dans la base de données, ainsi que les champs qui les composent. Dans notre cas, pour l'application de messagerie mobile, nous avons identifié quatre tables (Modèle Conceptuel de Données - MCD) :

1. **USER** : correspond à l'utilisateur.
2. **CHANNEL** : correspond au groupe, pouvant être public ou privé.
3. **CHANNEL MESSAGE** : correspond au message échangé au sein d'un groupe.
4. **CONVERSATION MESSAGE** : correspond au message échangé entre deux utilisateurs.

Pour faciliter la recherche de l'utilisateur associé à un groupe ou à une conversation privée entre deux utilisateurs, nous avons créé deux nouvelles tables. Celles-ci contiennent principalement les clés étrangères nécessaires pour lier les utilisateurs aux groupes et aux conversations :

1. **USER CHANNEL** : relie l'entité USER à l'entité CHANNEL.
2. **USER CONVERSATION** : relie l'entité USER (**from**) à l'entité USER (**to**).

Une fois la conception validée, nous pouvons passer à la réalisation et à la mise en place de la base de données. Il reste toujours possible de modifier ultérieurement les colonnes de ces tables en fonction des évolutions nécessaires de l'application.

2. Précisez les moyens utilisés :

- ▶ Conception **Lucidchart**.
- ▶ **MAMP**.
- ▶ **PhpMyAdmin**

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de conception avec les membres de mon groupe.

DOSSIER PROFESSIONNEL (DP)

4. Contexte

Période d'exercice ► Du : 02/01/2023 au : 06/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Concevoir une base de données »

DOSSIER PROFESSIONNEL (DP)

Activité-type 2

Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

Mettre en place une base de données ▶ Application mobile SimpleChat

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

Dans le cadre du projet de messagerie en temps réel, une fois que la conception de la base de données a été validée, j'ai participé à sa mise en place pour stocker et gérer les données de l'application.

Nous avons choisi **MySQL** comme système de gestion de base de données, car c'est une base de données relationnelle qui répond parfaitement aux besoins de notre application et dont l'implémentation est relativement simple.

Pour cela, j'ai décidé d'utiliser **Sequelize**, un puissant **ORM** (Object-Relational Mapping) pour **Node.js**, qui est compatible avec différents moteurs de base de données tels que MySQL, Postgres, etc. Dans notre cas, nous l'avons configuré pour utiliser le langage SQL.

La première étape de la mise en place consistait à créer un fichier config.js contenant un module exportable, qui contient les informations nécessaires pour effectuer la connexion à la base de données :



```
module.exports = {
  HOST: "localhost",
  USER: "root",
  PASSWORD: "root",
  DB: "chat_api",
  dialect: "mysql",
  pool: {
    max: 5,
    min: 0,
    acquire: 30000,
    idle: 10000,
  }
};
```

Ci-dessus, nous retrouvons les paramètres essentiels tels que le nom de la base de données, l'utilisateur et son mot de passe, ainsi que l'hôte. De plus, nous spécifions le dialecte, qui correspond au langage utilisé pour notre application, le langage SQL.

Pour des raisons de sécurité, ce module est déclaré de manière indépendante et utilisé lors de la configuration via une instance de **Sequelize** dans un fichier index.js situé dans le dossier "models". Il est également important de noter que le port est défini dans un fichier .env.

DOSSIER PROFESSIONNEL (DP)

```
const Sequelize = new Sequelize(  
    dbConfig.DB,  
    dbConfig.USER,  
    dbConfig.PASSWORD,  
    {  
        host: dbConfig.HOST,  
        dialect: dbConfig.dialect,  
        operatorsAliases: 0,  
        port: process.env.MYSQL_PORT,  
  
        pool: {  
            max: dbConfig.pool.max,  
            min: dbConfig.pool.min,  
            acquire: dbConfig.pool.acquire,  
            idle: dbConfig.pool.idle  
        }  
    }  
)
```

Dans le dossier "models", j'ai contribué à la création des modèles pour chaque entité, qui sont ensuite appelés à la suite de l'instance de Sequelize pour être accessibles lors des requêtes effectuées dans les différents services. Ces modèles peuvent être importés dans ces services pour interagir avec la base de données.

```
const db = {};  
db.Sequelize = Sequelize; db.sequelize = sequelize;  
db.user = require("./user.model.js")(sequelize, Sequelize);  
db.channel = require("./channel.model.js")(sequelize, Sequelize);  
db.userChannel = require("./userChannel.model.js")(sequelize, Sequelize);  
db.channelMessage = require("./channelMessage.model.js")(sequelize, Sequelize);  
db.userConversation = require("./userConversation.model")(sequelize, Sequelize);  
db.conversationMessage = require("./conversationMessage.model.js")(sequelize, Sequelize);  
  
module.exports = db;
```

Enfin, la connexion à la base de données est établie depuis le fichier app.js situé à la racine, où nous appelons la configuration de la base de données. Pour cela, j'ai utilisé la méthode sync(), qui est une promesse. Cette méthode peut renvoyer une erreur en cas de rejet de la promesse initiale.

```
const db = require("./model")  
db.sequelize.sync()  
.then(() => {  
    console.log('Synced db.')  
})  
.catch((err) => {  
    console.log('Error : ' + err.message)  });
```

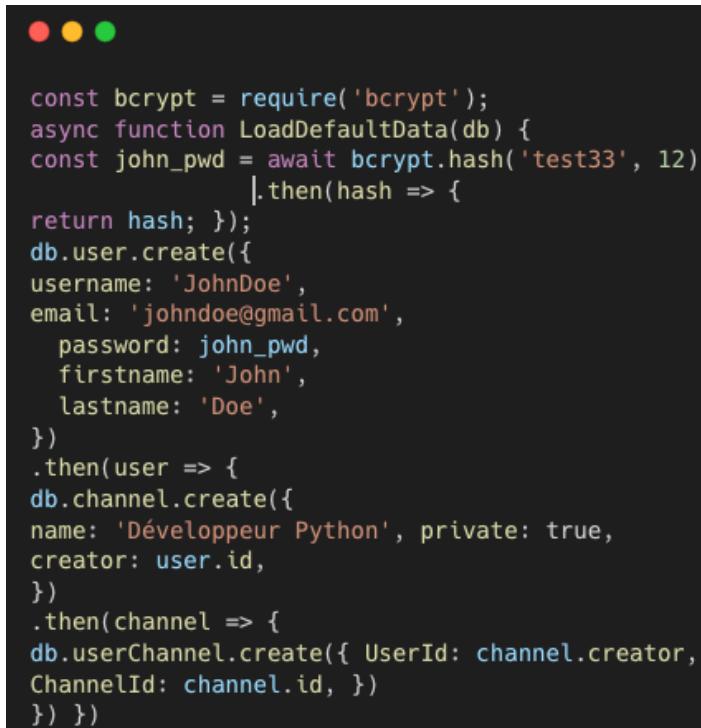
DOSSIER PROFESSIONNEL (DP)

Grâce à cette méthode, la base de données est configurée et connectée. La répartition en différents fichiers n'affecte pas cette connexion, ce qui signifie que nous pouvons modifier une entité sans altérer la connexion ou la configuration.

Afin d'avoir un jeu de données commun et à jour pour notre groupe, je me suis occupé de créer un fichier nommé defaultData.js. Dans ce fichier, j'ai ajouté deux utilisateurs, un groupe, deux messages au sein de ce groupe, ainsi qu'une conversation entre les deux utilisateurs représentés par deux messages.

Ce jeu de données nous permet de vérifier rapidement si les valeurs attendues correspondent et sont bien enregistrées dans la base de données.

Par exemple, pour la création d'un utilisateur et d'un groupe :



```
const bcrypt = require('bcrypt');
async function LoadDefaultData(db) {
  const john_pwd = await bcrypt.hash('test33', 12)
    .then(hash => {
      return hash;
    });
  db.user.create({
    username: 'JohnDoe',
    email: 'johndoe@gmail.com',
    password: john_pwd,
    firstname: 'John',
    lastname: 'Doe',
  })
    .then(user => {
      db.channel.create({
        name: 'Développeur Python', private: true,
        creator: user.id,
      })
        .then(channel => {
          db.userChannel.create({ UserId: channel.creator,
            ChannelId: channel.id, })
        })
    })
}

LoadDefaultData(db);
```

Ci-dessus, on remarque la présence d'une constante "**john_pwd**" qui utilise la méthode hash() pour encrypter le mot de passe, de la même manière qu'elle sera utilisée dans l'endpoint lors de l'inscription d'un utilisateur.

2. Précisez les moyens utilisés :

- ▶ L'éditeur de code **Visual Studio Code**.
- ▶ La documentation **Express**.
- ▶ La documentation **Sequelize**

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

J'ai effectué une partie de ce travail de seul et en groupe.

4. Contexte

Période d'exercice ► Du : 03/01/2023 au : 05/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Développer des composants d'accès aux données »

DOSSIER PROFESSIONNEL (DP)

Activité-type 2

Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

Développer des composants dans le langage d'une base de données

Application mobile – Simple Chat

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

Après avoir configuré et connecté la base de données, et défini les entités utilisées dans la configuration, j'ai commencé à développer les composants d'accès aux données, permettant d'effectuer des opérations courantes telles que la lecture, l'écriture, la mise à jour et la suppression de données.

Ces composants ont été placés dans un dossier appelé "services", avec un fichier service dédié à chaque entité. Cette approche facilite la lecture, l'écriture et la maintenance de ces composants.

Chaque méthode est auto-commentée et récupère divers paramètres de l'URL selon les besoins, utilisant les méthodes prédéfinies par **Sequelize**, notamment **findById**, **findAll**, **create**, **update** et **destroy**.

Tous les services sont développés de manière asynchrone pour ne pas entraver le fonctionnement de l'API. Ils retournent uniformément un statut et des données, permettant au client d'obtenir une réponse claire pour chaque requête, qu'elle soit réussie ou non.

Les services pour l'entité **USER**, similaire à un **CRUD**, permettent de :

- ⇒ Récupérer une liste des utilisateurs et certaines informations.
- ⇒ Créer un utilisateur (inscription).
- ⇒ Obtenir un JWT (lors de la connexion).
- ⇒ Récupérer les informations d'un utilisateur.
- ⇒ Rejoindre un groupe public.
- ⇒ Modifier ses informations personnelles.
- ⇒ Supprimer son compte.
- ⇒ Obtenir un nouveau JWT.

Les services pour l'entité **CHANNEL** permettent de :

- ⇒ Récupérer tous les groupes pour l'administrateur uniquement.
- ⇒ Récupérer tous les groupes publics.

DOSSIER PROFESSIONNEL (DP)

- ⇒ Récupérer la liste des utilisateurs membres d'un groupe.
- ⇒ Récupérer la liste des utilisateurs non membres d'un groupe.
- ⇒ Récupérer la liste des groupes d'un utilisateur.
- ⇒ Créer un groupe.
- ⇒ Modifier un groupe.
- ⇒ Ajouter un utilisateur à son groupe.
- ⇒ Supprimer un utilisateur de son groupe.
- ⇒ Supprimer un groupe.

Les services pour l'entité **CHANNEL MESSAGE** permettent de :

- ⇒ Récupérer tous les messages d'un groupe.
- ⇒ Créer un message.
- ⇒ Modifier un message.
- ⇒ Supprimer un message.

Les services pour l'entité **CONVERSATION** permettent de :

- ⇒ Récupérer toutes les conversations privées pour un utilisateur.
- ⇒ Récupérer une conversation privée.
- ⇒ Récupérer la valeur 'blocked' pour savoir si un utilisateur a bloqué un autre.
- ⇒ Créer une conversation privée.
- ⇒ Modifier la valeur 'blocked'.
- ⇒ Supprimer une conversation privée.

Les services pour l'entité **CONVERSATION MESSAGE** permettent de :

- ⇒ Récupérer tous les messages d'une conversation privée.
- ⇒ Créer un message.
- ⇒ Modifier un message.
- ⇒ Supprimer un message.

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

- ▶ L'éditeur de code **Visual Studio Code**.
- ▶ La **documentation Express**.
- ▶ La **documentation Sequelize**

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail en partie seul et en groupe

4. Contexte

Période d'exercice ▶ Du : 10/01/2023 au : 12/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Développer des composants dans le langage d'une base de données »

DOSSIER PROFESSIONNEL (DP)

Activité-type 3

Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement ▶ Application mobile SimpleChat

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

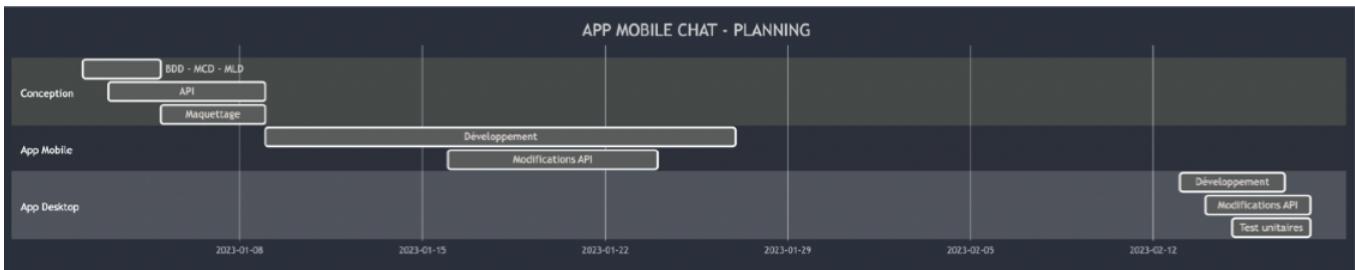
1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

Pour répondre aux attentes de notre projet de chat en temps réel, nous avons adopté une approche agile en le divisant en plusieurs phases.

Dans un premier temps, **nous avons réfléchi aux besoins** et à **la viabilité de notre application**. Nous nous sommes demandé si elle était utile, ce qu'elle apporterait de nouveau parmi les applications déjà existantes, et si notre équipe possédait les compétences nécessaires pour réaliser un projet de cette envergure dans le délai imparti. Nous avons échangé nos points de vue pour répondre à ces questions.

Une fois que nous avons été d'accord sur la direction à prendre, nous avons entamé la phase de planification de nos objectifs. Nous avons choisi de répartir **nos tâches en sprints d'une semaine**, permettant une adaptation rapide en cas d'objectifs trop complexes ou difficilement réalisables. Nous **faisions un point en début de chaque semaine pour évaluer notre progression dans le projet**.

Pour mieux visualiser nos tâches, nous avons utilisé **un diagramme de Gantt** pour représenter nos principales étapes, qui ont ensuite été divisées en petites tâches. Cela nous a aidés à avoir une vue d'ensemble et à mieux gérer notre avancement tout au long du projet.



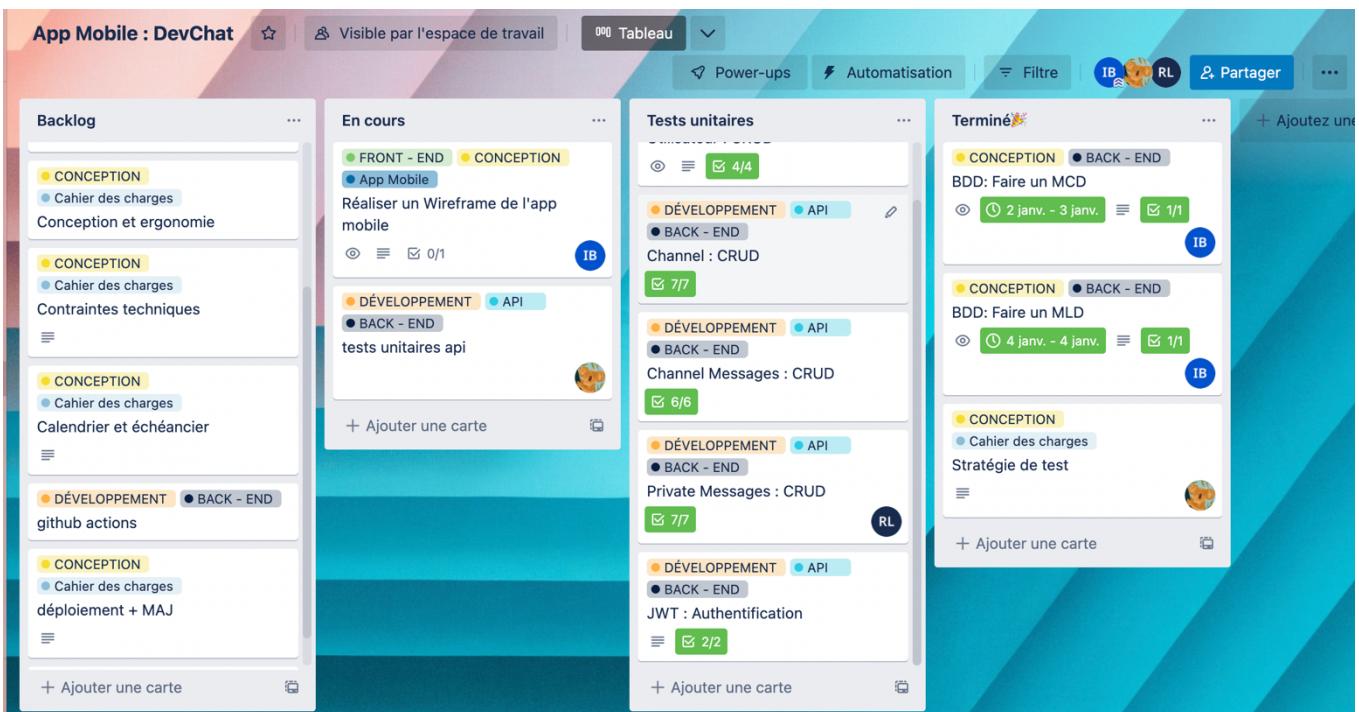
Ensuite, nous sommes passés à la phase d'exécution, au cours de laquelle nous avons utilisé divers logiciels de collaboration d'équipe tels que **Trello**. Cela nous a permis de nous mettre d'accord sur les tâches à réaliser et leurs échéances. De plus, nous avons utilisé **Google Chat** pour communiquer efficacement et rapidement. Ce logiciel de messagerie nous a également permis d'échanger des documents et du texte, ce qui était très pratique, notamment pour partager des bouts de code.

Une fois que toutes les tâches quotidiennes étaient rédigées et attribuées, nous sommes passés en phase de suivi et de contrôle pour surveiller en temps réel la faisabilité du projet dans le délai imparti.

DOSSIER PROFESSIONNEL (DP)

Pour cela, nous utilisions différentes colonnes sur Trello :

- "**A faire**" : tâches non attribuées
 - "**En cours**" : tâches attribuées et en cours de réalisation
 - "**En test**" : tâches attribuées, réalisées et en cours de tests
 - "**Bloqué**" : tâches attribuées, mais avec des problèmes bloquants
 - "**Terminé**" : tâches attribuées, réalisées et testées



Nous nous assurions qu'aucun effet indésirable ne viendrait entraver notre travail en effectuant des **Pull Requests** sur le **Repository Github**. Nous demandions à un membre de l'équipe d'effectuer une revue de code, ce qui permettait d'impliquer tous les acteurs du projet et d'améliorer la compréhension de celui-ci dans toutes ses parties.

2. Précisez les moyens utilisés :

- ▶ L'éditeur de code **Visual Studio Code**.
 - ▶ **Trello**
 - ▶ **Google, Google Doc et Google Chat**
 - ▶ **Figma**
 - ▶ **LucidChart**

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail avec les membres du groupe

DOSSIER PROFESSIONNEL (DP)

4. Contexte

Période d'exercice ► Du : 02/01/2023 au : 12/02/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Gestion d'un projet informatique et organisation de l'environnement de travail »

DOSSIER PROFESSIONNEL (DP)

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

Concevoir une application ▶ Application mobile SimpleChat

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

Dans un premier temps, nous avons rédigé **un cahier des charges via Google Doc** pour définir les objectifs, les fonctionnalités attendues, les entités nécessaires, les contraintes techniques et les langages que nous utiliserions pour notre application de messagerie en temps réel.

Concernant les fonctionnalités, nous avons opté pour **une application de messagerie en temps réel**, où les utilisateurs doivent s'inscrire et se connecter pour accéder à leurs conversations, qu'elles soient en groupe ou privées. Ils peuvent créer des groupes privés ou publics et échanger des messages à l'intérieur.

L'auteur d'un groupe peut inviter des membres, modifier la confidentialité, le nom du groupe, ainsi que les messages qu'il a écrits. De plus, il peut supprimer ses messages et son groupe s'il en est le créateur. Les utilisateurs peuvent également entamer des conversations privées et envoyer des messages. Dans les conversations privées, l'auteur peut modifier et supprimer ses messages. Il peut également bloquer un autre utilisateur et supprimer la conversation, ce qui entraîne la suppression de tous les messages.

Pour la réalisation technique de notre application, nous avons choisi Node.js pour l'API en raison de son côté asynchrone, ainsi que Express pour le serveur et le routage. **Les Sockets ont été utilisés pour le temps réel, et Sequelize pour l'accès aux données.**

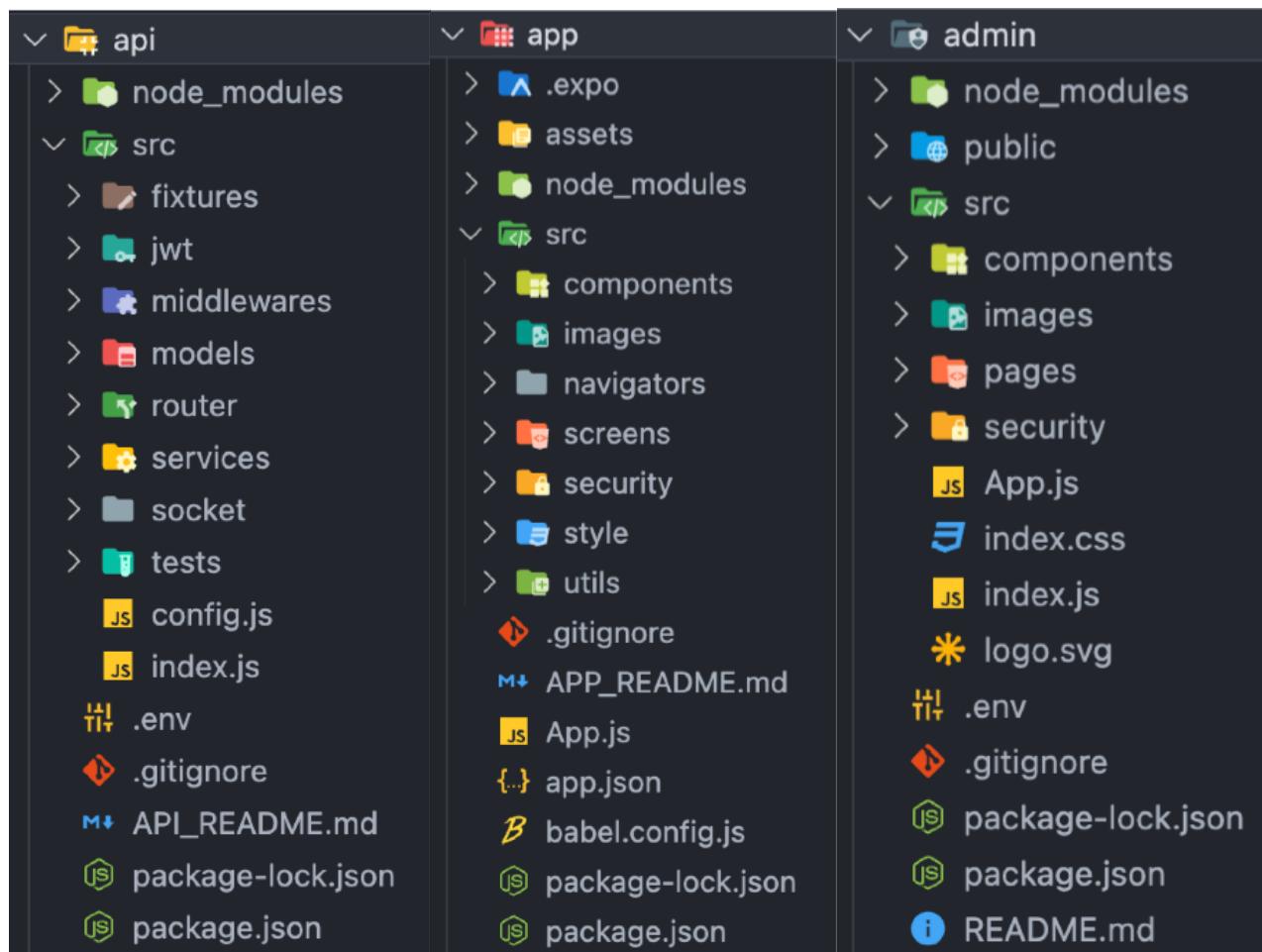
Pour l'application mobile, nous avons opté pour le framework React Native, permettant une approche hybride (iOS, Android). **Expo a été utilisé pour l'émulation de l'application mobile**, ce qui s'est avéré très utile pendant le développement. Quant à l'application de bureau, **nous avons choisi la bibliothèque React.js en raison de ses performances et de sa similarité avec React Native**, ce qui a permis un gain de temps considérable.

Nous avons conçu l'interface utilisateur à l'aide **de maquettes et de prototypes créés via Figma et Lucidchart**, afin d'avoir une direction concrète pour l'expérience utilisateur.

La sécurité de l'application est un aspect essentiel, et nous avons mis en place **un système d'authentification basé sur JSON Web Tokens (JWT)**. Les mots de passe des utilisateurs sont hachés avant d'être stockés dans la base de données grâce à la dépendance bcrypt. **Les tokens sont stockés dans des storages et gérés par un context hook de React Native et React.js**. Une structure conditionnelle vérifie les autorisations pour chaque requête privée, en s'assurant que l'utilisateur soit connecté et possède les droits nécessaires.

DOSSIER PROFESSIONNEL (DP)

Concernant l'architecture de l'application, nous avons choisi de diviser le projet en trois dossiers distincts : l'API, l'application mobile et l'interface administrateur.



L'organisation en différentes couches de l'application permet une meilleure maintenabilité du code et facilite son évolution. Cette approche permet d'ajouter de nouvelles fonctionnalités sans altérer la qualité du code existant. Par exemple, lors de **l'ajout des sockets à l'API, j'ai simplement créé un dossier "socket" contenant un seul fichier qui gère les différents appels en temps réel**. Cette modularité facilite l'extension de l'application sans compromettre sa stabilité.

2. Précisez les moyens utilisés :

- ▶ J'ai utilisé l'éditeur de code **Visual Studio Code**
- ▶ La documentation **Express**
- ▶ La documentation **Sequelize**
- ▶ La documentation **Socket.io**

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

J'ai effectué une partie de ce travail seul et en groupe

4. Contexte

Période d'exercice ► Du : 18/01/2023 au : 25/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Concevoir une application »

DOSSIER PROFESSIONNEL (DP)

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

Développer des composants métier ► Application mobile Simple Chat

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

Lors de la réalisation de notre application de Chat en temps réel nous avons identifié plusieurs classes métiers :

- ⇒ Modèles : USER, CHANNEL, CHANNEL MESSAGE, CONVERSATION MESSAGE, USER CHANNEL, USER CONVERSATION.
- ⇒ Services : CRUD pour utilisateurs, groupes, messages, conversations privées.
- ⇒ Sockets : Transmission en temps réel pour créations, modifications et suppressions de groupes, conversations et messages.

2. Précisez les moyens utilisés :

- J'ai utilisé l'éditeur de code **Visual Studio Code**
- La **documentation Express**
- La **documentation Sequelize**
- La **documentation Socket.io**

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail en groupe et seul.

4. Contexte

Période d'exercice ► Du : 04/01/2023 au : 11/01/2023

5. Informations complémentaires (facultatif)

Ce projet m'a permis de valider la compétence « Développer des composants métiers »

DOSSIER PROFESSIONNEL (DP)

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

Construire une application organisée en couches ► Application mobile SimpleChat – API/Sequelize

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

Le choix d'une architecture organisée en couches présente plusieurs avantages pour notre application :

1. **Séparation des responsabilités en couches logiques**, facilitant ainsi la compréhension et la maintenance de l'application. Les couches comprennent la persistance des données, la sécurité, le temps réel, etc., préservant ainsi la logique de notre application et de notre API.
2. **Réutilisation aisée du code entre différentes parties de l'application**, comme les middlewares de l'API utilisés dans chaque routeur ou les méthodes pour vérifier les droits des utilisateurs.
3. **Facilité d'évolution de l'application en ajoutant, modifiant ou supprimant des fonctionnalités sans causer d'effets indésirables**. Les tests unitaires garantissent cette évolution en identifiant tout problème lors d'une modification.

Pour préserver l'intégrité de notre application, nous avons choisi d'utiliser une architecture multicouche, comme expliqué dans la partie "Concevoir une application".

Dans l'organisation de l'API, les entités se trouvent dans le dossier "models", les méthodes d'accès aux données sont dans le dossier "services", les points d'extrémité (endpoints) sont déclarés dans le dossier "router", et les middlewares sont répertoriés dans le dossier portant le même nom, etc.

Par exemple, si je souhaite ajouter une entité, un service et un endpoint :

1. Je crée un fichier avec le nom de l'entité dans le dossier "models", où je déclare ses champs.
2. J'importe l'entité dans le fichier commun aux entités qui contient la configuration pour la base de données.
3. Je crée un fichier portant le nom de l'entité dans le dossier "services", où je déclare les méthodes possibles pour l'entité, comme les opérations CRUD (Création, Lecture, Mise à jour, Suppression).
4. Je crée un fichier portant le nom de l'entité dans le dossier "router", où je déclare le ou les endpoints correspondant à la ou les méthodes précédemment créées.
5. Je crée un fichier portant le nom de l'entité dans le dossier "tests", où j'écris les tests unitaires associés au(x) endpoint(s) créé(s).

Ainsi, il est facile de procéder en sens inverse lorsque l'on souhaite supprimer ou modifier une entité. Les logiques sont bien réparties à travers l'application, et on ne devrait jamais voir une seule requête SQL dans une page Web.

DOSSIER PROFESSIONNEL (DP)

Chaque composant possède un périmètre de responsabilités précis, reposant le moins possible sur d'autres classes. Chaque méthode exécute une série d'opérations aussi simples que possible, et les données circulent sur une chaîne, changeant d'état au fur et à mesure.

2. Précisez les moyens utilisés :

- ▶ J'ai utilisé l'éditeur de code **Visual Studio Code**
- ▶ La **documentation Express**
- ▶ La **documentation Sequelize**

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail seul et en groupe

4. Contexte

Période d'exercice ▶ Du : 05/01/2023 au : 13/01/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Construire une application organisée en couches »

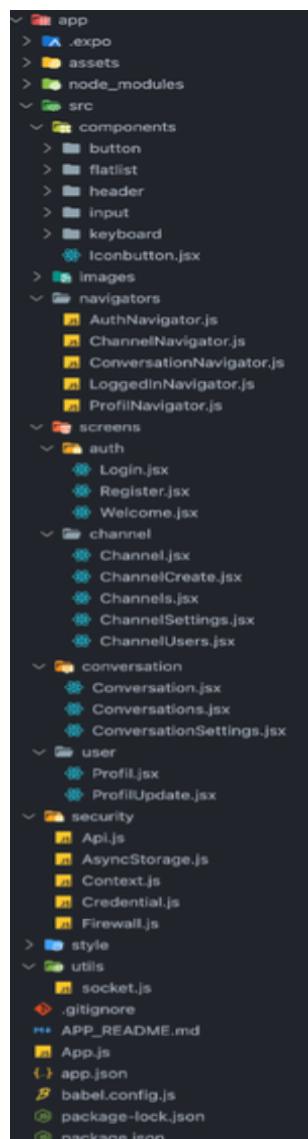
DOSSIER PROFESSIONNEL (DP)

Activité-type 3

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

Développer une application mobile Application mobile - SimpleChat

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



Après avoir développé l'API de notre application de messagerie en temps réel, nous nous sommes concentrés sur le développement de l'application mobile.

Pour rappel, l'application offre les fonctionnalités suivantes : un utilisateur doit s'inscrire et se connecter pour accéder à ses conversations de groupe ou privées. Il peut créer des groupes publics ou privés, gérer leur confidentialité et les membres présents. Il peut également rédiger, éditer et supprimer des messages au sein de son groupe. De plus, il peut initier des conversations privées avec d'autres utilisateurs, les bloquer si nécessaire, et envoyer, éditer et supprimer des messages dans ces conversations.

Ces fonctionnalités sont reliées à des sockets via l'API, permettant de mettre à jour en temps réel la création, l'édition et la suppression de groupes, de conversations privées et de messages, sans nécessiter de rafraîchissement de la page.

L'architecture de l'application mobile est présentée dans l'image ci-dessous. Le dossier "src" contient plusieurs couches de dossiers pour organiser la logique et faciliter la maintenance.

Le dossier "components" contient principalement des composants réutilisables pour l'affichage de données, l'accessibilité ou la navigation. **Le dossier "navigators"** contient divers routeurs de React Navigation pour la navigation inter-entités. Par exemple, le fichier "AuthNavigator.js" contient les écrans "Welcome.jsx", "Login.jsx" et "Register.jsx". **Le dossier "screens"** contient les écrans nécessaires à l'application, correspondant aux fonctionnalités décrites dans le cahier des charges. On retrouve un dossier par entité, ainsi que des fichiers contenant les opérations de type CRUD.

Le dossier "security" contient diverses méthodes utilisées pour les appels à l'API, le stockage des tokens, l'utilisation d'un contexte pour la transmission de données entre les composants parents et enfants, ainsi que la régénération du token une fois que l'utilisateur est connecté.

DOSSIER PROFESSIONNEL (DP)

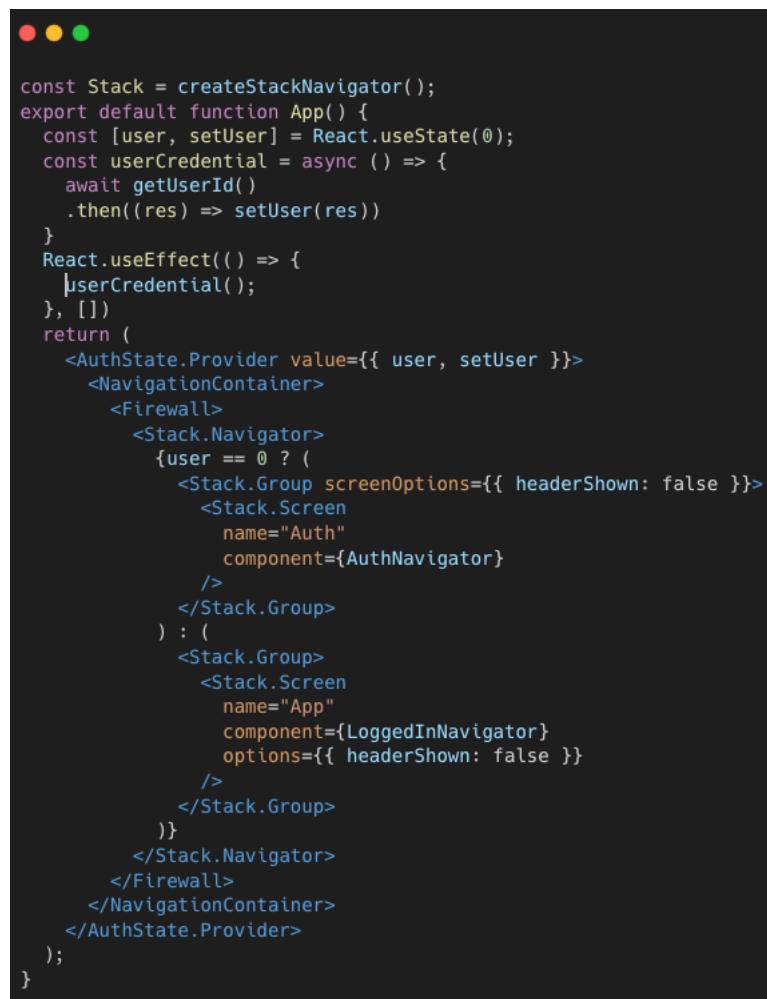
Le dossier "utils" contient la configuration et la connexion des sockets côté client.

Une documentation de l'application mobile est disponible dans le fichier "APP_README.md", où sont mentionnés les langages utilisés, les instructions pour lancer l'application depuis le terminal, ainsi que l'architecture du projet.

Le développement de l'application mobile s'est déroulé en plusieurs étapes :

- 1. Création du projet React Native.**
- 2. Création des divers dossiers pour séparer la logique.**
- 3. Développement des écrans et simultanément des navigateurs par fonctionnalité ou entité.**
- 4. Développement des composants et intégration du style.**
- 5. Développement des fonctions pour l'appel de l'API et la sécurité de l'utilisateur.**
- 6. Configuration et connexion aux sockets.**

Le cœur de l'application se trouve dans le fichier "App.js" :



```
const Stack = createStackNavigator();
export default function App() {
  const [user, setUser] = React.useState();
  const userCredential = async () => {
    await getUserId()
      .then((res) => setUser(res))
  }
  React.useEffect(() => {
    [userCredential];
  }, [])
  return (
    <AuthState.Provider value={{ user, setUser }}>
      <NavigationContainer>
        <Firewall>
          <Stack.Navigator>
            {user == 0 ? (
              <Stack.Group screenOptions={{ headerShown: false }}>
                <Stack.Screen
                  name="Auth"
                  component={AuthNavigator}
                />
              </Stack.Group>
            ) : (
              <Stack.Group>
                <Stack.Screen
                  name="App"
                  component={LoggedInNavigator}
                  options={{ headerShown: false }}
                />
              </Stack.Group>
            )}
          </Stack.Navigator>
        </Firewall>
      <NavigationContainer>
        </AuthState.Provider>
      );
}
```

DOSSIER PROFESSIONNEL (DP)

Dans le code extrait ci-dessus, plusieurs contrôles de sécurité sont mis en place pour l'application. En utilisant la méthode **getUserId()**, nous récupérons l'ID de l'utilisateur lorsqu'il est connecté. Ensuite, le résultat est stocké dans la variable "user" à l'aide du **hook useState**.

```
/**  
 * USER ID GETTER  
 * @returns user_id  
 */  
const getUserId = async () => {  
    try {  
        const jsonValue = await AsyncStorage.getItem('user_id')  
        return jsonValue != null ? JSON.parse(jsonValue) : null;  
    } catch(e) {  
        console.log(e);  
    }  
}
```

Ce mécanisme permet de vérifier rapidement si l'utilisateur est connecté, s'il a saisi ses identifiants correctement, et s'ils correspondent aux informations enregistrées en base de données. Dans ce cas, il peut pleinement profiter de l'application et utiliser les navigateurs présents dans le groupe appelé "App".

Ce mécanisme de contrôle permet de vérifier rapidement si l'utilisateur est correctement connecté et si ses identifiants correspondent à ceux stockés en base de données. Si c'est le cas, il peut alors pleinement profiter de l'application et utiliser les navigateurs présents dans le groupe appelé "App".

De plus, l'application hérite d'un fournisseur (provider) dans lequel on passe la valeur de l'ID de l'utilisateur. Cette valeur sera ensuite accessible et modifiable depuis n'importe quel composant enfant, sans avoir besoin de le transmettre via les props.

```
const AuthState = React.createContext({  
    user: 0,  
    setUser: (u) => {} });
```

Et pour finir, les **navigateurs** sont encapsulés par un "**firewall**" qui vérifie si l'ID de l'utilisateur est différent de 0 (valeur initiale) et active la connexion au socket. Ce firewall effectue également une régénération des tokens de l'utilisateur pour qu'il reste connecté une fois ses identifiants saisis et vérifiés.

Au niveau des requêtes vers l'API, nous avons adopté une approche consistante à développer des méthodes réutilisables et génériques, capables de prendre différents paramètres en fonction des besoins de l'API. Ainsi, nous avons mis en place des requêtes qui peuvent envoyer du contenu ou non, dont **on spécifie lors de leur utilisation la méthode (GET, POST, PUT, DELETE)** et l'endpoint requis.

DOSSIER PROFESSIONNEL (DP)

Pour les méthodes nécessitant l'utilisation du token de l'utilisateur, nous avons opté pour l'utilisation de l'asyncStorage pour récupérer ce dernier et l'envoyer dans les headers de la requête. Le token est ensuite accessible et exploitable au sein de l'API afin de vérifier les droits de l'utilisateur pour effectuer des opérations en base de données depuis un endpoint spécifique.

Chaque méthode est documentée de manière automatique, facilitant ainsi la recherche des paramètres requis lors de son utilisation.



```
/**  
 * Request without access token  
 * @param {*} path endpoint URL  
 * @param {*} method GET, POST, PUT or DELETE  
 */  
const simpleRequest = async (path, method) => {  
  let result = null;  
  await fetch("http://127.0.0.1:3001/api/" + path, {  
    method: method,  
    headers: {  
      'Content-Type': 'application/json',  
    }  
  })  
  .then((response) => response.json())  
  .then((data) => { result = data });  
  return result;  
};  
/**  
 * Request without access token  
 * @param {*} path endpoint URL  
 * @param {*} method GET, POST, PUT or DELETE  
 * @param {*} content object value  
 */  
const secureRequestContent = async (path, method, content) => {  
  const access = await getAccessToken()  
  let result = null;  
  await fetch("http://127.0.0.1:3001/api/" + path, {  
    method: method,  
    headers: {  
      'Content-Type': 'application/json',  
      'Authorization': `Bearer ${access}`,  
    },  
    body: JSON.stringify(content),  
  })  
  .then((response) => response.json())  
  .then((data) => { result = data });  
  return result;  
};
```

2. Précisez les moyens utilisés :

- ▶ J'ai utilisé l'éditeur de code **Visual Studio Code**
- ▶ La **documentation Express**
- ▶ La **documentation React Native**
- ▶ La **documentation Expo**
- ▶ La **documentation AsyncStorage**
- ▶ La **documentation Socket**

DOSSIER PROFESSIONNEL (DP)

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail en groupe

4. Contexte

Période d'exercice ► Du : 10/01/2023 au : 27/01/2023

5. Informations complémentaires (facultatif)

Ce projet m'a permis de valider la compétence « Développer une application mobile »

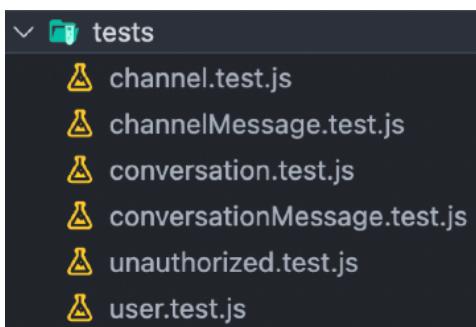
Activité-type 3 Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

Préparer et exécuter les plans de test d'une application ► Application mobile SimpleChat

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

Les tests unitaires sont répondus en entreprise car ils permettent de vérifier que les résultats attendus sont cohérents avec les résultats obtenus. Par exemple, si nous voulons qu'une fonction renvoie une valeur de type string, nous pouvons effectuer des tests unitaires pour nous assurer qu'elle renvoie effectivement une valeur de type string et non un integer, un boolean ou tout autre type.

Dans le cadre de notre projet de messagerie en temps réel, il était essentiel d'utiliser des tests unitaires sur l'API, en particulier sur chacun des endpoints. Étant donné que la quasi-totalité des endpoints nécessitent un token, il était crucial de tester la sécurité de ces endpoints pour empêcher toute tentative d'accès sans un token valide.



DOSSIER PROFESSIONNEL (DP)

Pour répondre à ce besoin, j'ai créé un dossier appelé 'tests', dans lequel j'ai ajouté un fichier de test pour chaque entité. Ensuite, j'ai installé les dépendances Jest et Supertest, qui permettent de créer des tests unitaires pour tester des API développées en Node.js.

Voici un exemple d'écriture de tests unitaires pour l'entité USER et les services qui lui sont associés : [Le code d'exemple n'a pas été fourni. Si vous le souhaitez, vous pouvez le fournir, et je serai heureux de vous aider à le reformuler ou à le commenter.

DOSSIER PROFESSIONNEL (DP)

```
● ● ●

const request = require("supertest");
const {server, app} = require("../index");
const db = require('../models/index');
require("dotenv").config();
/* Connecting to the database before each test. */
beforeEach(async () => {
  await db.sequelize.sync()
  .then(() => {
    console.log('Synced db.')
  })
  .catch((err) => {
    console.log('Error : ' + err.message)
  });
}); /* Closing database connection after each test. */
afterEach(async () => {
  await server.close();
});
  37 sur 41
/**
 * Public endpoint : GET all users
 */
describe('GET /api/users', () => {
  it('Should return all users', async () => {
    const res = await request(app)
      .get('/api/users');
    expect(res.statusCode).toBe(200);
    expect(res.body.status).toBe('Success');
    expect(res.body.data.length).toBeGreaterThan(0);
  });
}); /**
 * Private endpoints : user CRUD
 */
describe('User logged in with JWT', () => {
  let token = '';
  let user_id = 0;
  beforeEach(async () => {
    await request(app)
      .post('/api/register')
      .send({
        username: 'username__test',
        email: 'username@test.com',
        password: 'test',
        firstname: 'first__test',
        lastname: 'last__test',
      });
  });
  beforeEach(async () => {
    const res = await request(app)
      .post('/api/login')
      .send({
        email: 'username@test.com',
        password: 'test'
      });
    token = res.body.data.access_token;
    user_id = res.body.data.user_id
  });
  it('Should update user informations', async () => {
    const res = await request(app)
      .put(`/api/user/${user_id}`)
      .set('Authorization', `Bearer ${token}`)
      .send({
        firstname: 'update__first__test',
      });
    expect(res.statusCode).toBe(200);
    expect(res.body.status).toBe('Success');
    expect(res.body.data.updated.firstname).toBe('update__first__test');
  });
  it('Should delete user', async () => {
    const res = await request(app)
      .delete(`/api/user/${user_id}`)
      .set('Authorization', `Bearer ${token}`)
    expect(res.statusCode).toBe(200);
    expect(res.body.status).toBe('Success');
  });
});
```

DOSSIER PROFESSIONNEL (DP)

Pour expliquer brièvement l'écriture des tests unitaires, qui sont répliqués pour chaque entité selon leurs endpoints, une connexion et une déconnexion à la base de données sont effectuées entre chaque test. Ensuite, pour les endpoints publics, une requête asynchrone est réalisée, et on s'assure que le résultat renvoyé correspond à celui attendu lorsque l'opération se déroule correctement (code statut égal à 200).

En revanche, pour les endpoints privés nécessitant un token, il est nécessaire de créer un utilisateur et de le connecter pour obtenir son ID et son token. Cela permet également de tester si ce endpoint fonctionne correctement. Enfin, les tests unitaires correspondants aux endpoints à tester sont écrits, généralement pour une modification, une lecture de données ou une suppression.

En lançant la commande "**npm run test**" dans le terminal, on peut rapidement observer si chaque endpoint, et donc service, retourne la valeur attendue.

2. Précisez les moyens utilisés :

- ▶ J'ai utilisé l'éditeur de code **Visual Studio Code**
- ▶ Le **Terminal de commande** de l'IDE
- ▶ La **documentation Jest**

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail seul et en groupe

4. Contexte

Période d'exercice ▶ Du : 03/02/2023 au : 07/02/2023

5. Informations complémentaires (*facultatif*)

Ce projet m'a permis de valider la compétence « Préparer et exécuter les plans de tests d'une application »

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

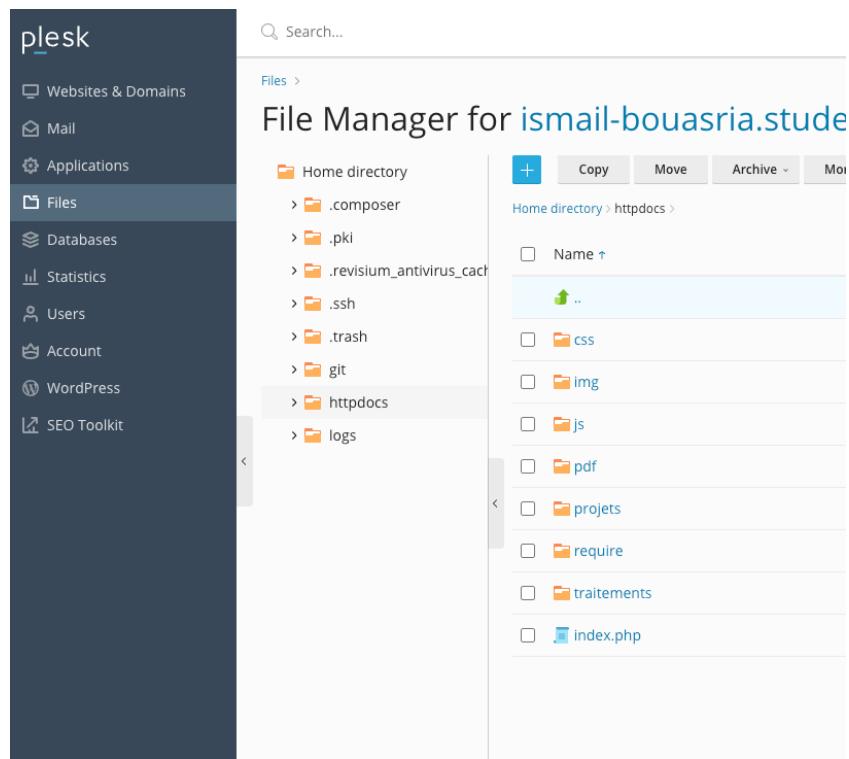
Préparer et exécuter le déploiement d'une application ▶ Application mobile Simple Chat

DOSSIER PROFESSIONNEL (DP)

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

Dans ma recherche d'alternance, j'ai réalisé un portfolio que j'ai ensuite déployé sur un hébergeur Plesk.

Pour cela, j'ai d'abord transféré **mon dossier "portfolio"** directement sur le serveur, dans le dossier "**htdocs**".



Après cela, j'ai ajouté ma **base de données MySQL** en utilisant l'onglet **Database**. J'ai créé un utilisateur pour établir la connexion. Ensuite, j'ai mis à jour les informations de connexion dans mes variables d'environnement.

DOSSIER PROFESSIONNEL (DP)

Screenshot of the Plesk interface showing the 'Add a Database' dialog.

The left sidebar shows the Plesk navigation menu with 'Databases' selected. The main area shows the 'Add a Database' form:

- General** section:
 - Database name: ismail-bouasria_portefolio
 - Database server: localhost:3306 (default for MariaDB, v5.5.68)
 - Related site: ismail-bouasria.students-laplateforme.io
- Users** section:
 - Create a database user: checked
 - Database user name: ismail-bouasria
 - Password: masked
 - Confirm password: masked
 - User has access to all databases within the selected subscription: unchecked
- Buttons: OK (blue), Cancel (grey)

J'indique le point d'entrée de mon domaine, mon site est maintenant en ligne.

Screenshot of the Plesk interface showing the domain management screen.

The left sidebar shows the Plesk navigation menu. The main area shows the domain ismail-bouasria.students-laplateforme.io:

- Domain name: ismail-bouasria.students-laplateforme.io
- Status: Active
- Type: Website
- Dashboard tab is active, showing a preview of the website.
- Files & Databases section:
 - Connection Info (for FTP, Database)
 - File Manager
 - Databases
 - FTP Access
 - Backup & Restore
 - Website Copying
- Security section:
 - SSL/TLS Certificates (Domain not secured)
 - Password-Protected Directories
 - Advisor
- Dev Tools sidebar:
 - PHP Settings (Version 8.0.26)
 - Logs
 - Applications
 - Git (Enabled)
 - PHP Composer
 - Install WordPress
 - Node.js
 - SEO Toolkit
 - Website Importing

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

- ▶ J'ai utilisé Plesk

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail seul.

4. Contexte

Période d'exercice ▶ Du : 20/05/2022 au : 30/05/2022

5. Informations complémentaires (*facultatif*)

Ce projet me permet de valider la compétence de « préparer et d'exécuter du déploiement d'une application ».

DOSSIER PROFESSIONNEL (DP)

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.

DOSSIER PROFESSIONNEL (DP)

Déclaration sur l'honneur

Je soussigné(e) Ismaïl BOUASRIA ,

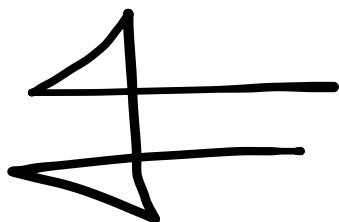
déclare sur l'honneur que les renseignements fournis dans ce dossier sont exacts et que je suis
l'auteur(e) des réalisations jointes.

Fait à Marseille

le 30/05/2023

pour faire valoir ce que de droit.

Signature :



DOSSIER PROFESSIONNEL (DP)

Documents illustrant la pratique professionnelle

(facultatif)

DOSSIER PROFESSIONNEL (DP)

ANNEXES

(Si le RC le prévoit)