Dossier Professionnel (DP)

Nom de naissance ▸ LOPEZ

Nom d'usage ▸ LOPEZ

Prénom ▸ Rémi

Adresse ▸ 84 Boulevard de la libération, 13004 MARSEILLE

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 ministère chargé de l'Emploi]

Ce dossier comporte :

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

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

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

Dossier Professionnel (DP)

Sommaire

Exemples de pratique professionnelle

EXEMPLES DE PRATIQUE PROFESSIONNELLE

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

Maquetter une application ▸ *SCEV*

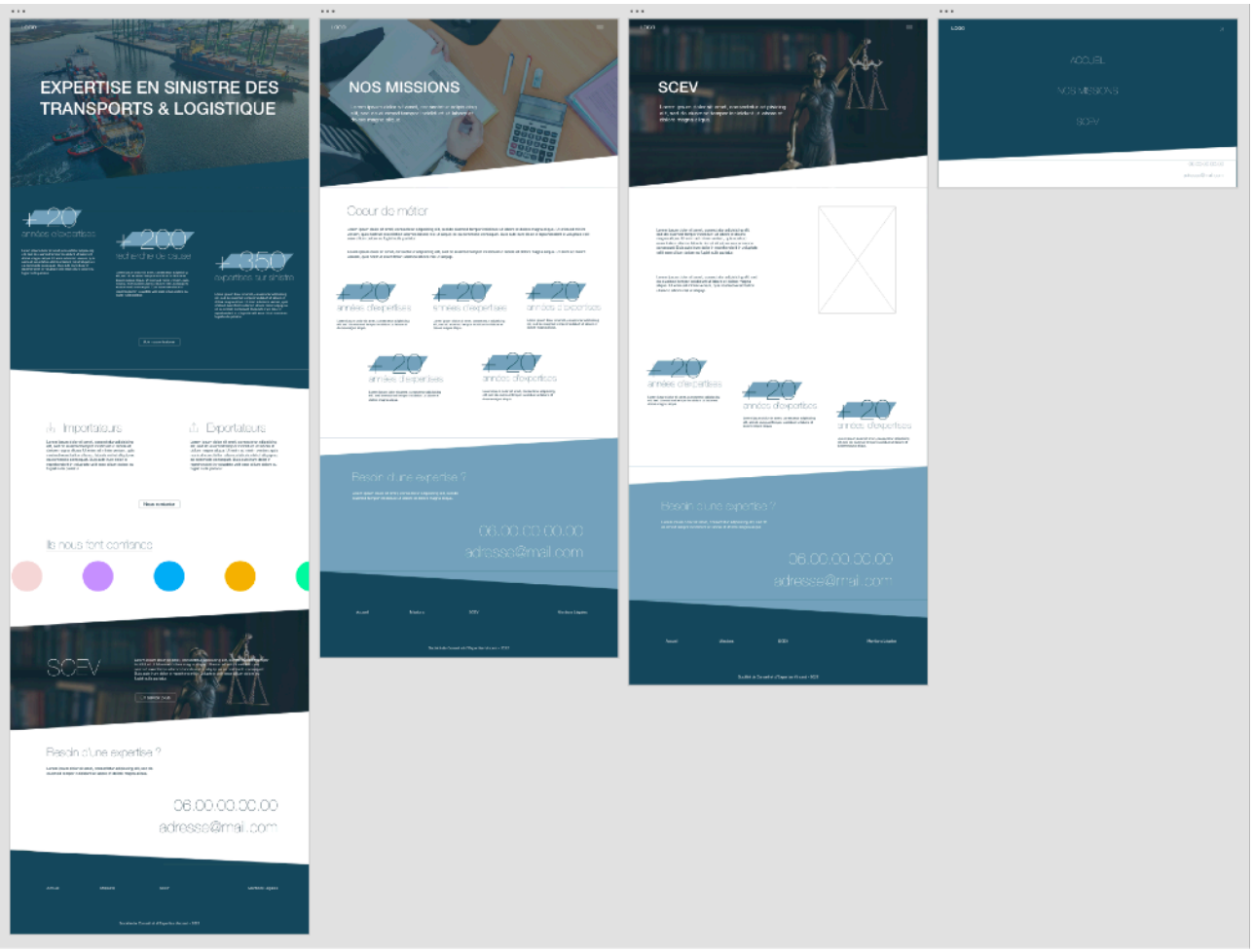
1. Décrivez les tâches ou opérations que vous avez effectuées, et dans quelles conditions :

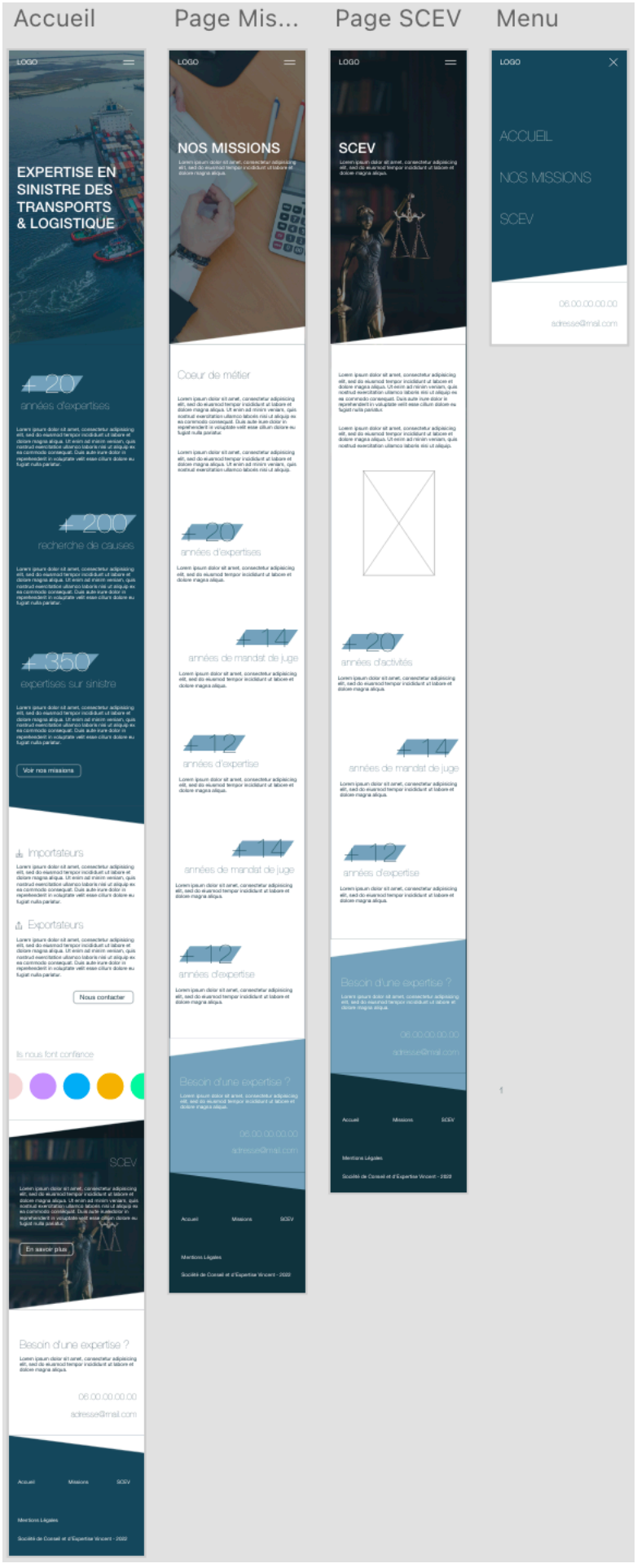
Pour répondre aux attentes d'un client, en suivant un cahier des charges et en effectuant une veille concurrentielle sur les sites concurrents, j'ai proposé une maquette en version mobile et desktop afin de présenter l'aspect responsive de mon design.

Pour cela, j'ai utilisé Adobe XD, un logiciel d'UI/UX design performant qui permet la création de différentes vues (pages) ainsi que la création d'un prototype, contenant une navigation entre les pages, pour offrir une représentation concrète du design et de l'interactivité au client.

Les seules contraintes au niveau du design étaient que le bleu devait utilisé et il fallait un bandeau comprenant les logos des partenaires de mon client.

Le client souhaitant trois pages et un menu, j'ai réalisé quatre maquettes en version mobile et quatre autres en version desktop :




Accueil
Page Mis...
Page SCEV
Menu
LOGO
EXPERTISE EN SINISTRE DES TRANSPORTS & LOGISTIQUE
+20
années d'expertises
+200
recherche de causes
+350
expertises sur sinistre
Voir nos missions
Importateurs
Exportateurs
Nous contacter
Ils nous font confiance
SCEV
En savoir plus
Besoin d'une expertise ?
06.00.00.00.00
adresse@mail.com
Accueil
Missions
SCEV
Mentions Légales
Société de Conseil et d'Expertise Vincent - 2022
NOS MISSIONS
Coeur de métier
+14
années de mandat de juge
+12
années d'expertise
ACCUEIL
NOS MISSIONS
SCEV
+20
années d'activités

2. Précisez les moyens utilisés :

- **Adobe XD** (logiciel UX - UI).
- Veille concurrentielle.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail seul.

4. Contexte

Période d'exercice ▸ Du 01/05/2022 au 09/05/2022

5. Informations complémentaires (facultatif)

- **Ce projet m'a permis de valider la compétence « Maquetter une application »**
  - **La maquette prend en compte les spécificités fonctionnelles décrites dans les cas d'utilisation ou les scénarios utilisateur**
  - **L'enchaînement des écrans est formalisé par un schéma**
  - **La maquette respecte les principes de sécurisation d'une interface utilisateur**
  - **La communication écrite, en français ou en anglais, est rédigée de façon adaptée à l'interlocuteur et sans faute**

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

Au cours de la formation, nous devions réaliser un jeu en python, en utilisant la bibliothèque PyGame, pour ensuite, l'utiliser sur notre ordinateur.

Le principe du jeu est celui du Sokoban, ou gardien d'entrepôt. le joueur doit ranger des caisses sur des cases cibles. Il peut se déplacer dans les quatre directions, et pousser (mais pas tirer) une seule caisse à la fois. Une fois toutes les caisses rangées, le niveau est réussi et le joueur passe au niveau suivant, généralement plus difficile.

Le projet contient 50 niveaux, ainsi qu'une bande son et plusieurs bruitages selon les actions effectuées par l'utilisateur.

Des classes ont été définies pour une meilleure maintenance et compréhension du code. Par exemple, la classe File va permettre de sauvegarder et récupérer le dernier niveau réalisé avec succès.

La classe Menu permet d'effectuer différentes actions comme la relance du niveau, la configuration de la bande son ou encore pour quitter le jeu.

La classe Game est utilisée comme manager est utilise les différentes classes pour générer l'écran, le plateau, le niveau, les actions possibles par l'utilisateurs, … C'est cette classe qui est appelée dans le fichier principal, qui permet à un utilisateur de lancer le jeu sur son ordinateur.

```
def main( ):
    game = Game( )
    game.start( )

    pygame.init( )
    pygame.display.init( )
    pygame.display.set_caption('My Sokoban')

    screen =  pygame.display.set_mode(game.screen.size)
    bg_img = pygame.transform.scale(
        pygame.image.load('themes/space.png'),
        game.screen.size
    )

    mixer.init( )
    Sound( ).intro( )

    while True:
        screen.blit(bg_img, (0, 0))
        game.play(screen)


if __name__ == '__main__':
    main( )
```

2. Précisez les moyens utilisés :

- J'ai utilisé l'éditeur de code **Visual Studio Code**.
- J'ai utilisé de la **documentation** de pygame.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de développement d'une interface utilisateur de type desktop seul.

4. Contexte

Période d'exercice ▸ Du 17/04/2023 au 23/04/2023

5. Informations complémentaires (facultatif)

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

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

Pour décrire brièvement le projet, nous étions en groupe de 3 personnes et nous devions réaliser une application mobile de chat, connectée à des Sockets. Le parcours client est le suivant :

Un utilisateur s'inscrit, puis se connecte pour accéder à son dashboard personnel. Depuis cette page, il peut créer un groupe publique ou privé, rejoindre un groupe publique, inviter des membres dans son groupe et échanger des messages au sein de celui-ci.

Il peut également créer une conversation privée entre lui et un autre utilisateur, pour échanger des messages.

Qu'il s'agisse d'un groupe ou d'une conversation privée, les informations sont modifiables, les messages échangées peuvent être modifiés et supprimés seulement par l'auteur de ces derniers.

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

Pour pouvoir interagir avec une base de données, j'ai commencé à configurer et développer la connexion à la base de données, en utilisant Sequelize et Express.

Une fois le MCD conceptualisé et la connexion à la base de données effective, j'ai pu m'atteler au développement des différentes entités. J'ai pour cela commencé par définir un dossier nommé Model et à l'intérieur de celui-ci, une page par entité.

Chaque entité a par la suite été typée, puis associée en clé étrangère selon les besoins.

Une fois les entités créées, je pouvais développer les composants permettant l'accès au données dans un dossier nommé Service et contenant lui aussi, un fichier par entité.

Chaque service présente un ensemble de méthodes asynchrones, afin de ne pas gêner le déroulement d'autres fonctions, qui permettent diverses actions.

Pour reprendre le descriptif du projet et ses fonctionnalités, 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 (administrateur, utilisateur connecté, créateur d'un groupe, …)

Pour des raisons de sécurité, j'ai mit en place les JSON Web Token, dont les méthodes de génération et vérification permette de s'assurer que l'utilisateur souhaitant accéder aux données possède les droits demandés.

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 seul.

4. Contexte

Période d'exercice ▸ Du 01/01/2023 au 05/01/2023

5. Informations complémentaires (facultatif)

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

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

Pour reprendre brièvement le projet précédent, nous devions développer une interface web permettant à un administrateur de gérer facilement les données envoyées par les utilisateurs.

Comme nous avions déjà développé l'application mobile en React Native, j'ai opté pour la bibliothèque React.js. En effet, elle est très proche de la version Native ce qui facilite grandement le développement de cette interface web.

J'ai donc imaginé et développé un dashboard permettant de voir les différents utilisateurs, les groupes et les messages échangés au sein de ces derniers.

Pour accéder à ce dashboard, l'administrateur doit se connecter en saisissant son adresse email et son mot de passe, et pour des raisons de sécurité, posséder le rôle ADMIN en base de données.

Une fois connecté, un firewall prend le relais pour faire appel au endpoint de régénération de Token, permettant à l'administrateur de rester connecté autant de temps que nécessaire.

Depuis ce dashboard, l'administrateur peut aisément consulter et supprimer des groupes publique et privé, des messages échangés par les utilisateurs ainsi que les utilisateurs eux-même.

Dans une version plus poussée, on peut imaginer qu'il puisse également créer et modifier des groupes et messages, cependant je ne me suis pas attardé sur cette partie là puisqu'elle ne correspond pas totalement à la vision que j'ai d'un administrateur.

Pour réaliser cette application web :

- J'utilise la commande *npx create-react-app* pour créer un projet React.js
- J'utilise la commande *npm start* pour lancer l'application sur un serveur local (localhost:3000)
- J'installe et utilise le package *react-router-dom* permettant de créer un routeur et de générer différents URL

2. Précisez les moyens utilisés :

▸ J'ai utilisé l'éditeur de code **Visual Studio Code**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail seul.

4. Contexte

Période d'exercice ▸ Du 13/02/2023 au 17/02/2023

5. Informations complémentaires (facultatif)

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

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

Le back-end de l'interface web étant également utilisé par l'application mobile, il m'a fallut revoir certains services puisque l'API était opérationnelles depuis plusieurs jours lorsque j'ai commencé le développement de l'interface utilisateur web.

Il a fallut que je crée de nouveaux services spécifique à l'administrateur et pour une efficacité, j'ai développé une méthode qui récupère le rôle de l'utilisateur via le token envoyé dans le header grâce à une requête.

S'il existe en base de données, je récupère le rôle et vérifie immédiatement sa valeur pour retourner trois type de résultats :

- **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',
          '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 depuis le routeur en qualité de middleware. Cela permet de s'assurer rapidement que l'utilisateur qui tente d'accéder à certains endpoints est bien un administrateur. Par exemple, depuis son dashboard, il peut consulter la liste de tout type de groupe, qu'ils soient privés ou publiques. Pour cela, j'ai développé une nouvelle méthode ainsi qu'un nouvel endpoint :

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

Dans l'exemple ci-dessus, le router est généré par express et sa méthode Router( ) puis stocké dans la variable router. La méthode get( ) signifie que ce endpoint effectue une requête de type GET.

On retrouve l'URL à renseigné côté client, ainsi qu'un ID à fournir dans ce dernier. Cela permettra de vérifier rapidement qu'il s'agit bien d'un administrateur. Un premier middleware est appelé pour vérifier que le token est bien présent dans le header, puis le vérifier et extraire les données qu'il comporte.

Chaque service est nommé selon sa fonctionnalité et il prend systématiquement l'ID de l'utilisateur pour des requêtes privées. Voici un exemple des service spécifique à l'

```
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',
    })
  }
}
```

Chaque méthode est asynchrone, et comporte une structure conditionnelle selon le rôle de l'utilisateur, dans ce cas, seul l'administrateur peut accéder à cette requête, mais dans le cas suivant, le service est autant accessible à un utilisateur dont l'ID et le token correspondent qu'à un administrateur.

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

L'administrateur dispose ainsi des requête 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 seul.

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 « Développer le back-end d'une interface utilisateur web »**

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

Concevoir une base de données ▸ *Simple Chat (cf. Annexe p.1)*

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

Avant de développer une API et/ou le back-end d'une application, il est important de bien définir les entités que l'on souhaite stocker en base de données ainsi que les champs qui les composent. (cf. Annexe p.1)

Dans le cas de notre application de messagerie mobile, nous avons définit quatre tables (MCD) :

- ***USER*** : correspond à l'utilisateur
- ***CHANNEL*** : correspond au groupe, qui peut être publique ou privé
- ***CHANNEL MESSAGE*** : correspond au message échangé au sein d'un groupe
- ***CONVERSATION MESSAGE*** : correspond au message échangé entre deux utilisateurs

Afin de retrouver facilement l'utilisateur associé à un groupe ou une conversation privé entre deux utilisateurs, deux nouvelles tables sont créées. Elles contiennent principalement les clés étrangères nécessaires pour retrouver les groupes et conversation associées à l'utilisateur.

- ***USER CHANNEL*** : lien entre l'entité USER et l'entité CHANNEL
- ***USER CONVERSATION*** : lien entre l'entité USER (from) et l'entité USER (to)

Lorsque la conception est validée, nous pouvons passer à sa réalisation et la mise en place de la base de données. Il est toujours possible de modifier ultérieurement les colonnes de ces tables selon les besoins évolutifs de l'application.

2. Précisez les moyens utilisés :

- J'ai utilisé l'éditeur de code **Visual Studio Code**.
- J'ai utilisé la documentation officielle **Express** et **Sequelize**.
- J'ai utilisé **MAMP** et **PhpMyAdmin**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de conception de base de données en groupe.

4. Contexte

Période d'exercice ▸ Du 02/01/2023 au 03/01/2023

5. Informations complémentaires (facultatif)

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

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

Toujours en lien avec le projet de messagerie en temps réel, une fois la conception de base de données validée, j'ai participé à la mise en place de celle-ci afin de stocker et gérer les données de l'application.

Nous avons choisit comme système de gestion de base de données MySQL, une base de données relationnelle qui correspond pleinement aux besoins de notre application et dont l'implémentation est relativement simple.

Pour cela, j'ai choisit d'utiliser Sequelize, un puissant ORM pour Node.js, compatible avec différent moteurs de base de données comme 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 :

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

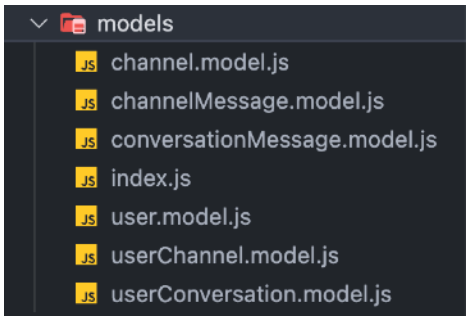
On retrouve ci-dessus les paramètres incontournables tels que le nom de la base de données, l'utilisateur et son mot de passe ou encore l'hôte. De plus, on retrouve la spécification 'dialect' qui correspond au langage utilisé pour notre application, le langage SQL.

Pour des raisons de sécurité, ce module est déclaré indépendamment et utilisé lors de la configuration via une instance de Sequelize dans un fichier index.js présent dans le dossier models. On note également que le port est définit dans un fichier .env.

```
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
        }
    }
);
```

Toujours dans le dossier models, j'ai participé à la création des modèles pour chacune des entités qui sont appelées à la suite de l'instance de sequelize, pour être accessible lors des requêtes effectuées dans les différents services. Elles peuvent être importées dans ces derniers 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 s'effectue depuis le fichier app.js situé à la racine, dans lequel on appelle la configuration de la base de données. J'ai pour cela utilisé la méthode sync( ) qui est une promesse. Elle peut retourner une erreur dans le cas où la promesse initiale est rejetée

```
const db = require("./model")

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

A partir de cette méthode, la base de données et configurée et connectée. La répartition en différents fichiers n'impacte pas celle-ci, on peut par exemple modifier une entité sans pour autant altérer la connexion ou la configuration.

Afin de posséder une jeu de données commun dans notre groupe et à jour par rapport à la base de données, j'ai réalisé un fichier nommé defaultData.js dans lequel je crée deux utilisateurs, un groupe, deux messages au sein de celui-ci ainsi qu'une conversation entre les deux utilisateurs représentée par deux messages.

Ce jeu de données permet de voir rapidement si les valeurs attendues correspondent et sont bien enregistrées en 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,
      })
    })
  })
```

On note ci-dessus la présence d'une constante 'john_pwd' qui utilise la méthode hash( ) pour encroûter le mot de passe, comme il sera le cas dans le endpoint lors de l'inscription d'un utilisateur.

2. Précisez les moyens utilisés :

- J'ai utilisé l'éditeur de code **Visual Studio Code**.
- J'ai utilisé la documentation officielle **Express** et **Sequelize**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de création des composants d'accès aux données seul.

4. Contexte

Période d'exercice ▸ Du 03/01/2023 au 04/01/2023

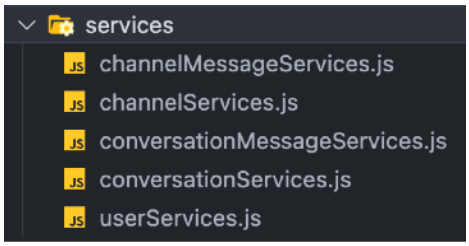
5. Informations complémentaires (facultatif)

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

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

Une fois la base de données configurée et connectée, les entités définies et appelés lors de la configuration, j'ai pu 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.

Pour cela, j'ai choisit de placer ces composants dans un dossier nommé 'services' et comme pour les modèles, de mettre en place un fichier service par entité, ce qui facilite grandement la lecture, l'écriture et la maintenance de ces composants.



Chacune des méthode est auto-commentée, récupère divers paramètres de l'URL selon les besoins et utilise les méthodes prédéfinies par Sequelize : *findByPk*, *findAll*, *create*, *update* et *destroy*.

Chaque service est développé de façon asynchrone pour n'entraver en rien le fonctionnement de l'API. Selon le bon déroulement ou non de la requête, le service retourne uniformément un statut et des données. Cela permet ensuite côté client d'obtenir une réponse pour la requête demandée, qu'elle soit passée ou qu'elle ait échouée.

Ainsi, les services proposés pour l'entité **USER**, à l'image d'un CRUD, permettent de :

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

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

- Récupérer tout type de groupe uniquement pour l'administrateur
- Récupérer tous les groupes publique
- Récupérer la liste des utilisateurs membre d'un groupe
- Récupérer la liste des utilisateurs non membre 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 proposés 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 proposés 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 en a bloqué un autre
- Créer une conversation privée
- Modifier la valeur 'blocked'
- Supprimer une conversation privée

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

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

2. Précisez les moyens utilisés :

- J'ai utilisé l'éditeur de code **Visual Studio Code**.
- J'ai utilisé la documentation officielle **Express** et **Sequelize**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de développement de composants dans le langage d'une base de données seul.

4. Contexte

Période d'exercice ▸ Du 04/01/2023 au 06/01/2023

5. Informations complémentaires (facultatif)

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

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

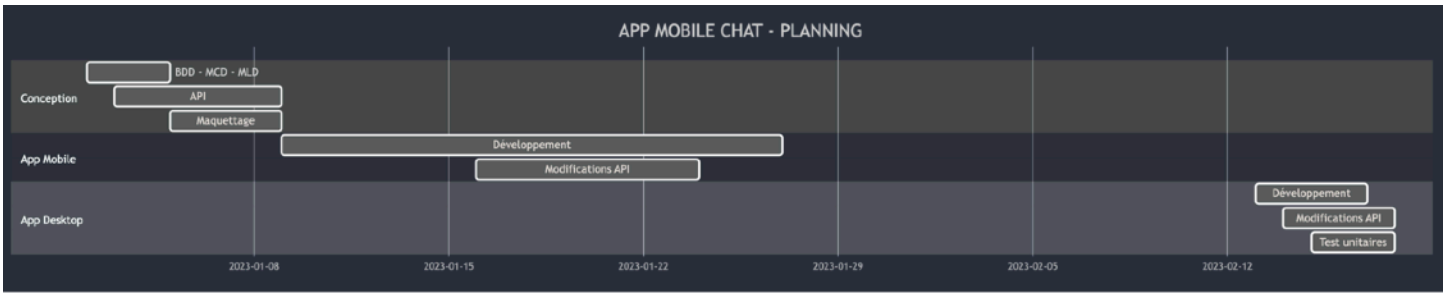
Afin de répondre au mieux aux attentes que nous avions autour de notre projet, nous avons décidé de gérer la réalisation de l'application de chat en temps réel de façon **agile**, et en plusieurs phases.

Tout d'abord, nous avons réfléchit aux besoins et la viabilité de notre application. En d'autres termes, est-elle utile ? Qu'apporte-elle de nouveau parmi celles déjà existantes ? A-t-on les compétences suffisante dans l'équipe pour réaliser un projet d'une telle envergure dans un temps donné ?

Autant de questions auxquelles nous avons répondu en échangeant nos points de vues. Une fois nos discours accordés, nous sommes passé à la phase suivante et nous avons commencé la planification de nos objectifs.

Pour cela, nous avons décidé de répartir nos tâches en **sprint** d'une semaine, permettant de rapidement s'adapter en cas d'objectifs trop complexes, peu réalisables. Pour cela, nous faisions un point en début de semaine suivante pour observer notre avancement dans ce projet.

Afin de visualiser plus facilement nos tâches, nous avons modélisé un **diagramme de Gantt** pour représenter nos tâches principales qui ont par la suite, été divisées en petites tâches.



Nous sommes ensuite passé à la phase d'exécution durant laquelle nous avons utilisé divers **logiciels de collaboration** d'équipe tels que **Trello** pour que chacun s'entende sur les tâches à réaliser et leurs échéances, ou encore **Google Chat** pour communiquer efficacement et rapidement. De plus, ce logiciel de messagerie permet de s'échanger tout type de documents et de texte. Une véritable aubaine pour communiquer rapidement des bouts de code ou autre.

Lorsque toutes les tâches quotidiennes étaient rédigées et attribuées, nous passions en phase de suivi et de contrôle pour surveiller en temps réel la faisabilité du projet selon la durée allouée.

Pour cela, nous disposions de divers 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 existence d'un point bloquant
- **Terminé** : tâches attribuées, réalisées et testées

Nous nous assurions qu'aucun effet de bord ne viendrait entraver notre travail en effectuant des **Pull Request** sur le Repository Github, en demandant à un membre de l'équipe d'effectuer une revue de code.

Cette pratique permet d'impliquer tous les acteurs du projet et améliore la compréhension de ce dernier sur toutes ses parties.

2. Précisez les moyens utilisés :

- J'ai utilisé l'éditeur de code **Visual Studio Code** pour la réalisation du diagramme de Gantt
- J'ai utilisé **Trello**.
- J'ai utilisé **Google Doc** et **Google Chat**.
- J'ai utilisé **Figma**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de gestion de projet et d'organisation de l'environnement de développement en groupe.

4. Contexte

Période d'exercice ▸ Du 02/01/2023 au 02/01/2023

5. Informations complémentaires (facultatif)

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

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

Pour répondre aux besoins du projet, nous avons dans un premier temps rédigé via Google Doc un cahier des charges dans lequel nous avons détaillé l'objectif, les fonctionnalités attendues, les entités nécessaires, les contraintes techniques et les langages que nous allions utiliser.

Côté **fonctionnalités**, nous avons opté pour une application de messagerie en temps réel, ce qui signifie que l'utilisateur doit s'inscrire et se connecter pour accéder à ses conversions, qu'elles soient en groupe ou privées.

Un utilisateur peut ensuite créer un groupe, privé ou publique et rédiger des messages au sein de celui-ci. Il peut bien évidemment inviter des membres à rejoindre son groupe s'il en est le créateur. De plus, il peut modifier la confidentialité de son groupe, son nom ainsi que les messages qu'il a rédigé. Enfin, il peut supprimer ses messages et son groupe s'il en est le créateur.

Un utilisateur peut créer une conversation privée avec un autre utilisateur et envoyer des messages. S'il en est l'auteur, il peut modifier et supprimer ses messages. Il dispose également d'une fonctionnalité permettant de bloquer l'autre utilisateur s'il juge, par exemple, ses propos insultants, déplacés, … et il peut supprimer la conversation, qui entraine la suppression de tous les messages.

Toutes ces fonctionnalités sont connectés à des sockets pour le temps réel, ce qui permet d'afficher par exemple un message reçu sur l'écran d'une conversation privée sans avoir à rafraichir la page.

En ce qui concerne les **caractéristiques technique** de notre application, nous avons opté :

- Pour l'API, nous avons opté pour Node.js pour son côté asynchrone, Express pour le serveur et le routage, les Sockets pour le temps réel ainsi que Sequelize pour l'accès aux données.

- Pour l'application mobile, nous avons opté pour le framework React Native qui permet un gain de temps considérable puisqu'il permet de développer une application mobile hybride (IOS, Android). Nous avons également utilisé Expo pour son émulation d'application mobile, une chose très précieuse lors du développement de notre application.

- Pour l'application desktop, nous avons opté pour la bibliothèque React.js pour ses capacités de performances et sa proximité avec React Native, qui représente un gain de temps considérable.

Pour l'**interface utilisateur**, nous avons conçu une maquette et un prototype de l'application via Figma. Cela nous a permis d'avoir une direction concrète à suivre pour l'expérience utilisateur.

La **sécurité de l'application** étant un point primordial, nous avons imaginé un système d'authentification par JWT, configurés du côté de l'API pour s'assurer que l'utilisateur est bien connecté et possède bien les droits demandés pour effectuer des opérations en base de données.

Bien entendu, nous hachons le mot de passe de l'utilisateur avant de l'envoyer en base de données grâce à la dépendance bcrypt, qui permet également de comparer le mot de passe envoyé par l'utilisateur lorsqu'il souhaite se connecter.

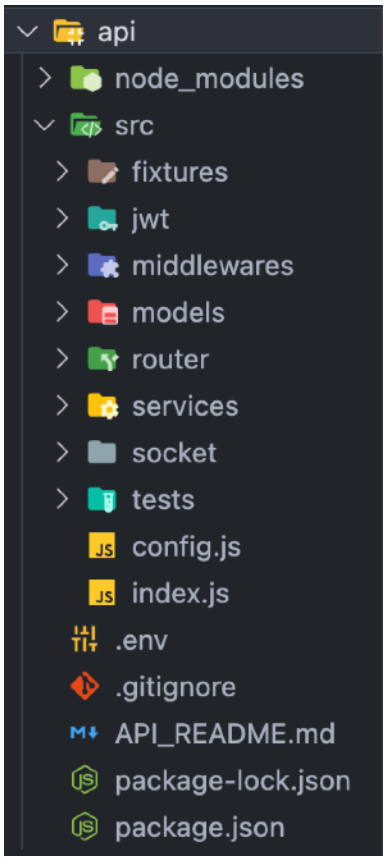
Ainsi, lorsque la connexion est effective, on reçoit côté front-end un access token et un refresh token à la durée de vie plus longue et qui permet de régénérer les deux token lorsque l'access arrive à expiration. Cela à pour effet de laisser connecter l'utilisateur, c'est-à-dire qu'il n'a pas a saisir à nouveau ses identifiants pour profiter des fonctionnalités de l'application.

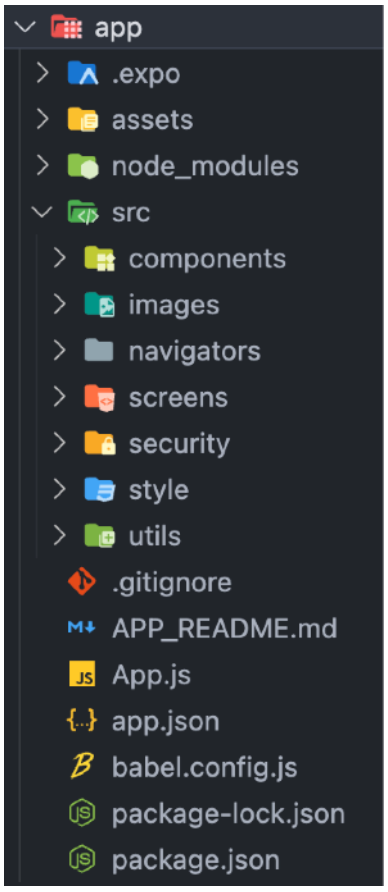
Ces tokens sont stockés dans des storages, accessibles depuis un hook context de React Native et React.js.

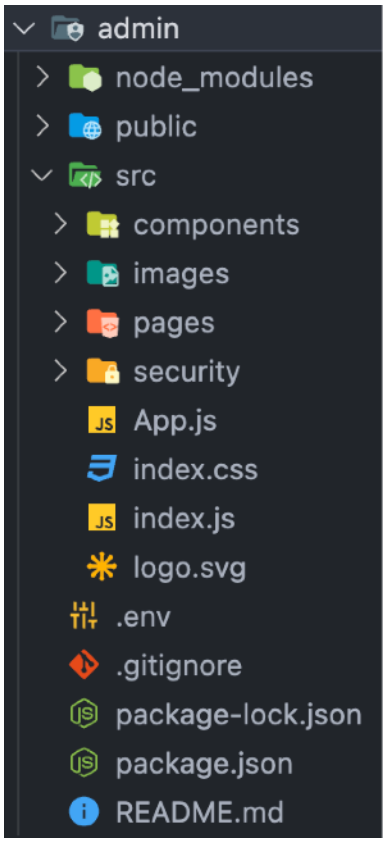
De plus, une structure conditionnelle vérifie pour chaque requête privée que l'utilisateur qui en fait la demande est soit un administrateur, soit un utilisateur connecté et dont l'ID présent dans l'URL correspond à cet utilisateur.

Pour certaines requêtes accessible seulement à l'administrateur, on s'assure que le rôle enregistré en base de données et bien celui d'administrateur puisqu'il possède une vue d'ensemble sur le site.

En ce qui concerne l'**architecture de notre application**, nous avons opté pour une distinction entre l'API, l'application mobile et l'interface administrateur. On se retrouve ainsi avec trois dossiers différents.







Cela permet une meilleure maintenabilité du code ainsi qu'une évolution plus facile de l'application.

En effet, l'organisation en différentes couches permet de rapidement rajouter une fonctionnalité sans pour autant altérer la qualité du code existant. Par exemple, lorsque j'ai rajouté les sockets à l'API, j'ai simplement créé un dossier 'socket' qui contient un seul fichier qui gère les différents appels en temps réel.

2. Précisez les moyens utilisés :

- J'ai utilisé l'éditeur de code **Visual Studio Code**.
- J'ai utilisé la documentation officielle **Express** et **Sequelize**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de conception d'application en groupe et seul pour l'architecture et la sécurité de l'application.

4. Contexte

Période d'exercice ▸ Du 03/01/2023 au 06/01/2023

5. Informations complémentaires (facultatif)

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

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

Lors de la réalisation de notre application de messagerie en temps réel, nous avons identifié diverses classes métier (cf. Annexe p.1)

Au niveau des **modèles** :

- ***USER*** : correspond à l'utilisateur
- ***CHANNEL*** : correspond au groupe, qui peut être publique ou privé
- ***CHANNEL MESSAGE*** : correspond au message échangé au sein d'un groupe
- ***CONVERSATION MESSAGE*** : correspond au message échangé entre deux utilisateurs
- ***USER CHANNEL*** : lien entre l'entité USER et l'entité CHANNEL
- ***USER CONVERSATION*** : lien entre l'entité USER (from) et l'entité USER (to)

Au niveau des **services** (gestionnaires) :

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

- Récupérer tout type de groupe uniquement pour l'administrateur
- Récupérer tous les groupes publique
- Récupérer la liste des utilisateurs membre d'un groupe et non-membre d'un groupe
- Récupérer la liste des groupes d'un utilisateur
- Créer, modifier, supprimer un groupe
- Ajouter, supprimer un utilisateur à son groupe

- Récupérer tous les messages d'un groupe
- Créer, modifier, supprimer un message

- Récupérer toutes les conversations privées pour un utilisateur
- Récupérer une conversation privée
- Créer, supprimer une conversation privée
- Récupérer la valeur 'blocked' pour savoir si un utilisateur en a bloqué un autre
- Modifier la valeur 'blocked'

- Récupérer tous les messages d'une conversation privée
- Créer, modifier, supprimer un message

Au niveau des **sockets** :

- Connexion entre le client et le serveur :
- Transmission en temps réel lors de la création, modification et suppression d'un groupe
- Transmission en temps réel lors de la création, modification et suppression d'une conversation
- Transmission en temps réel lors de la création, modification et suppression d'un message

2. Précisez les moyens utilisés :

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

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de développement de composants métier seul.

4. Contexte

Période d'exercice ▸ Du 04/01/2023 au 06/01/2023

5. Informations complémentaires (facultatif)

- **Ce projet m'a permis de valider la compétence « Développer des composants métier »**

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

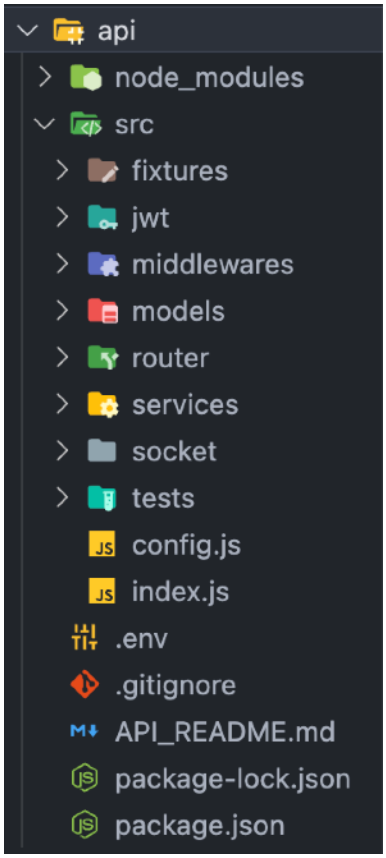
Le choix d'une architecture organisée en couches offre plusieurs avantages à notre application :

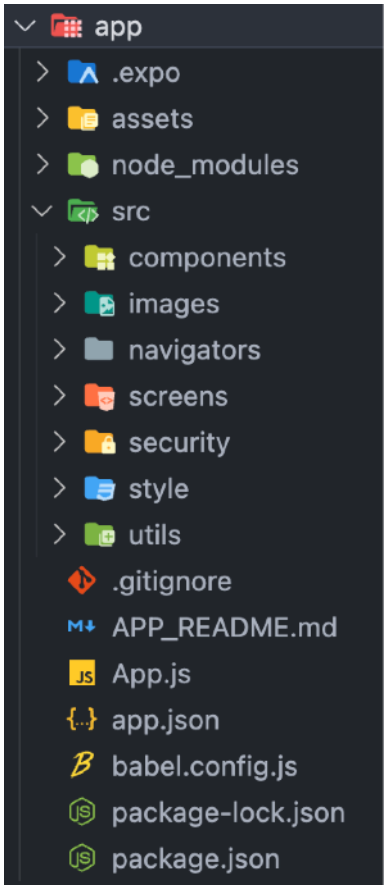
- Les responsabilités sont séparés en couches logiques, cela facilite grandement la compréhension et la maintenabilité de l'application. On a donc répartie la persistance des données, la sécurité, le temps réel, etc afin de préserver la logique de notre application et notre API.

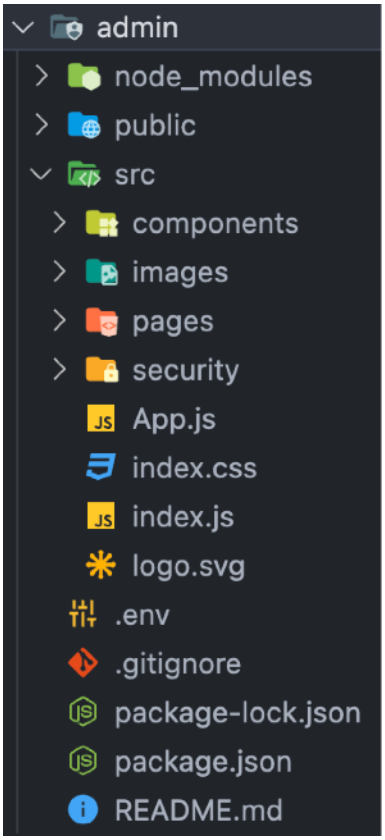
- Il est très facile de réutiliser du code entre différentes parties de l'application, comme par exemple les middlewares de l'API qui sont appelés dans chaque routeur, ou les méthodes appelées pour vérifier les droits des utilisateurs.

- Il est très facile de faire évoluer notre application et de rajouter, modifier ou supprimer une fonctionnalité sans pour autant générer d'effet de bord. Les tests unitaires assurent cette évolution en pointant le moindre problème lorsqu'une modification est apportée.

Comme je l'ai abordé dans la partie 'concevoir une application', j'ai fait le choix d'utiliser une architecture multicouches pour préserver l'intégrité de notre application.







Si l'on prend le cas de l'API et de son organisation, les entités sont présentes dans le dossier 'models', les méthodes d'accès aux données sont présentes dans le dossier 'services', les endpoints sont déclarés dans le dossier 'router', les middlewares sont répertoriés dans le dossier du même nom, etc.

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

1. Je crée un fichier au nom de l'entité dans le dossier 'models' où je déclare ses champs
2. J'importe l'entité dans le fichier commun aux entités et qui contient la configuration pour la base de données
3. Je crée un fichier au nom de l'entité dans le dossier 'services' où je déclare les méthodes possibles pour l'entité, par exemple de type CRUD
4. Je crée un fichier au nom de l'entité dans le dossier 'router' où je déclare le ou les endpoint(s) correspondant au(x) méthode(s) précédemment créée(s).
5. Je crée un fichier au nom de l'entité dans le dossier 'tests' où j'écris les tests unitaires associés au(x) endpoint(s) crée(s).

Ainsi, on peut aisément faire le chemin inverse lorsque l'on souhaite supprimer ou modifier une entité. Les logiques sont bien réparties au travers de l'application et on ne devrait jamais voir une seule requête SQL dans une page Web.

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

2. Précisez les moyens utilisés :

- J'ai utilisé l'éditeur de code **Visual Studio Code**.
- J'ai utilisé la documentation officielle **Express** et **Sequelize**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de développement de composants dans le langage d'une base de données seul.

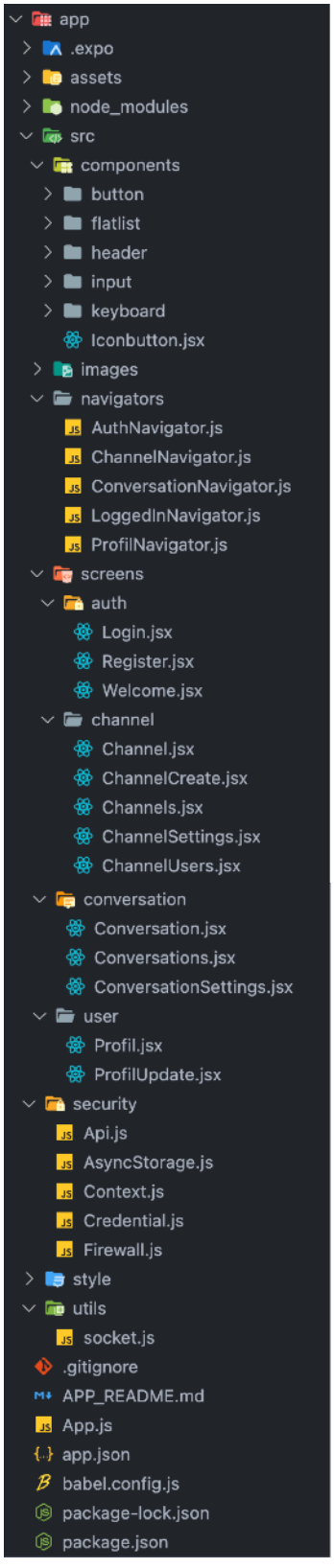
4. Contexte

Période d'exercice ▸ Du 04/01/2023 au 06/01/2023

5. Informations complémentaires (facultatif)

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

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



Suite au développement de l'API de notre application de messagerie en temps réel, nous sommes passé au développement de l'application mobile.

Pour **rappel des fonctionnalités**, un utilisateur souhaitant utiliser l'application doit s'inscrire et se connecter pour accéder à ses conversions de groupe ou privées.

Il peut créer un groupe, privé ou publique et gérer par la suite la confidentialité ainsi que les membres présents dans son groupe. Il peut bien évidemment rédiger, éditer et supprimer des messages au sein du groupe auquel il appartient.

Il peut également créer une conversation privée pour dialoguer avec un utilisateur. Il peut gérer cette conversation en bloquant ou non l'individu, et envoyer, éditer et supprimer des messages au sein de la conversation.

Via l'API, ces fonctionnalités sont connectés à des socket pour mettre à jour en temps réel la création, l'édition et la suppression de groupes, conversations privées et messages, évitant ainsi la nécessité de rafraichir la page pour mettre à jour l'affichage.

L'architecture de l'application mobile est présentée dans l'image ci-contre. Le coeur de l'application se situe dans le dossier **src** qui contient plusieurs couches de dossiers pour répartir la logique et faciliter la maintenance.

Le dossier **components** contient principalement des composants réutilisables et utilisés pour l'accessibilité, l'affichage de données ou la navigation.

Le dossier **navigators** contient divers routeurs React Navigation, pour la navigation inter-entités, par exemple, AuthNavigator.js contient le screen Welcome.jsx, le screen Login.jsx et le screen Register.jsx.

Le dossier **screens** contient les écrans nécessaires à l'application, pour répondre aux fonctionnalités exprimées dans le cahier des charges. On retrouve un dossier par entité, ainsi que des fichiers contenant diverses opérations de type CRUD.

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

Le dossier **utils** contient la configuration et la connexion des Socket côté client.

On retrouve une documentation de l'application mobile dans le fichier **APP_README.md**, dans lequel est mentionné les langages utilisés, comment lancer l'application depuis le terminal ou encore l'architecture du projet.

Nous avons développé notre application mobile en plusieurs étapes :

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

Le coeur de l'application présente dans le fichier App.js :

```
const Stack = createStackNavigator();

export default function App() {
  const [user, setUser] = React.useState(0);

  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>
  );
}
```

Dans l'extrait de code ci-dessus, on retrouve plusieurs contrôle au niveau de la sécurité de l'application. En effet, on utilise la méthode getUserId( ) qui retourne l'ID de l'utilisateur lorsque celui-ci est connecté. Le résultat est ensuite stocké dans la variable user via le 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);
  }
}
```

Cela permet de contrôler rapidement si l'utilisateur est bien connecté, à saisit ses identifiants et qu'ils correspondent à ceux stocké en base de données. Si tel est le cas, il peut alors profiter pleinement de l'application et utiliser les navigators présent dans le groupe nommé App.

Ensuite, l'application hérite d'un provider dans lequel on passe la valeur de l'ID de l'utilisateur. Ce dernier sera ensuite accessible et modifiable depuis n'importe quel composant enfants, sans forcément devoir le passer en props.

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

Enfin, les navigators sont englobés par un firewall, qui check si l'ID de l'utilisateur est différent de 0 (valeur initiale) et active la connexion au socket. Il effectue également une régénération des tokens de l'utilisateurs pour qu'il reste connecté une fois ses identifiants saisis et contrôlés.

```
function Firewall({ children }) {
  const { user } = React.useContext(AuthState);

  React.useEffect(() => {
    if(user != 0) {
      socket.auth = { user };
      socket.connect();
    }

    let interval = setInterval(() => {
      check()
    }, 1000 * 60 * 5); // 1000ms * 60s * 5 = 5min

    return function cleanUp() {
      clearInterval(interval);
    }
  }, [user])

  const check = async () => {
    if(user != 0) regenerateToken();
  }

  return <>{children}</>
}
```

Au niveau des requêtes vers l'API, nous avons fait le choix de développer des méthodes réutilisables et génériques, qui prennent selon les besoin de l'API différents paramètres. Ainsi, on retrouve des requêtes qui envoie du contenu ou non, dont on définit lors de l'utilisation, la méthode (GET, POST, PUT, DELETE) et l'endpoint demandé.

Pour les méthodes nécessitant le token de l'utilisateur, on utilise l'asyncStorage pour récupérer celui-ci et l'envoyer dans le headers. Il sera ensuite accessible et exploitable au sein de l'API pour s'assurer que l'utilisateur possède les droits demandés pour effectuer des opérations en base de données depuis un endpoint.

Chaque méthode est auto-documentée pour aisément retrouver les paramètres nécessaires 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**.
- J'ai utilisé la documentation officielle **React Native**.
- J'ai utilisé la documentation officielle **Expo**.
- J'ai utilisé la documentation officielle **AsyncStorage**.
- J'ai utilisé la documentation officielle **Socket**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de développement d'une application mobile en groupe.

4. Contexte

Période d'exercice ▸ Du 09/01/2023 au 27/01/2023

5. Informations complémentaires (facultatif)

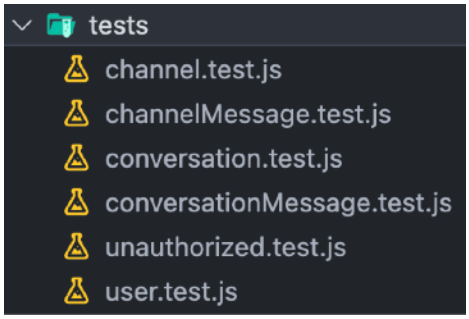
- **Ce projet m'a permis de valider la compétence « Développer une application mobile »**

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

Les tests unitaires sont très utilisés dans les entreprises puisqu'ils permettent de s'assurer que les résultats attendus sont cohérents avec les résultats obtenus. Par exemple, si l'on souhaite qu'une fonction nous retourne une valeur de type string, on peut effectuer des tests unitaires pour s'assurer qu'elle retourne uniquement une valeur de type string et non un integer, un boolean ou autre…

Dans le cadre de notre projet de messagerie en temps réel, il me semblait primordial d'utiliser les tests unitaires sur l'API, et plus précisément sur chacun des endpoints. En effet, la quasi-totalité des endpoints nécessitant un token, il est primordial de tester la sécurité de ces derniers dans le cas où un individu tente d'accéder à des requêtes sans envoyer de token valide.

Pour répondre à cette problématique, j'ai commencé par créer un dossier nommé 'tests', dans lequel j'ai créé un fichier de test pour chacune des entités.



Ensuite, j'ai installé la dépendance Jest et Supertest qui permettent la création de tests unitaires pour tester des API développées en Node.js.

Exemple d'écriture de tests unitaires pour l'entité USER et les services qui lui sont associés :

```
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();
});
```

```
/**
 * 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;

  beforeAll(async () => {
    await request(app)
      .post('/api/register')
      .send({
        username: 'username__test',
        email: 'username@test.com',
        password: 'test',
        firstname: 'first__test',
        lastname: 'last__test',
      });
  });

  beforeAll(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');
  });
});
```

Pour expliquer brièvement l'écriture des tests unitaires, qui sont répliquées pour chaque entités selon leurs endpoints, j'effectue une connexion et déconnexion à la base de données entre chaque test.

Ensuite, s'il existe des endpoints public j'effectue une requête asynchrone et je m'assure que le résultat renvoyé correspond à celui retourné lorsque l'opération s'est bien déroulé (code statut égal à 200)

En revanche, pour les endpoint privés nécessitant un token, je suis obligé 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, j'écris les tests unitaires correspondants aux endpoints à tester. Généralement, il s'agit d'une modification, d'une lecture de données ou de la suppression.

En lançant la commande '*npm run test*' dans le terminal, on observe très rapidement 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**.
- J'ai utilisé le **terminal** de commande de mon PC.
- J'ai utilisé la documentation officielle **Jest**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de préparation et exécution de tests unitaires seul.

4. Contexte

Période d'exercice ▸ Du 06/03/2023 au 07/03/2023

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 ▸ *Portfolio personnel*

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

Au cours de ma précédente formation, j'ai décidé de développer mon portfolio en React.js pour présenter mes compétences ainsi que mes travaux professionnels et personnels.

Une fois, la maquette et le développement réalisés, j'ai acheté un nom de domaine sur OVH, un hébergeur français, ainsi qu'un hébergement pour mettre mon site en production.

N'ayant pas besoin de base de données pour ce projet, je n'ai pas eu à configurer une base de données sur l'hébergeur, cependant, j'ai utilisé la réécriture d'URL pour pouvoir utiliser le routeur de React en production et rediriger automatiquement vers un protocole sécurisé https.

En utilisant la commande '*npm run build*' qui permet de générer un build optimisé pour le déploiement, j'ai pu via un logiciel de FTP nommé Filezila envoyer mon projet sur mon hébergeur en pointant vers mon nom de domaine.

Après quelques minutes seulement, mon portfolio était en ligne et accessible.

2. Précisez les moyens utilisés :

- J'ai utilisé le logiciel **FileZila**.
- J'ai utilisé la documentation officielle **OVH**.

3. Avec qui avez-vous travaillé ?

J'ai effectué ce travail de développement de composants dans le langage d'une base de données seul.

4. Contexte

Période d'exercice ▸ Du 04/01/2023 au 06/01/2023

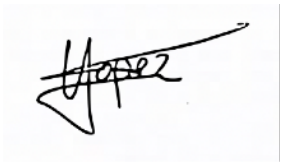
5. Informations complémentaires (facultatif)

- **Ce projet m'a permis de valider la compétence « Préparer et exécuter le déploiement d'une application »**

Déclaration sur l’honneur

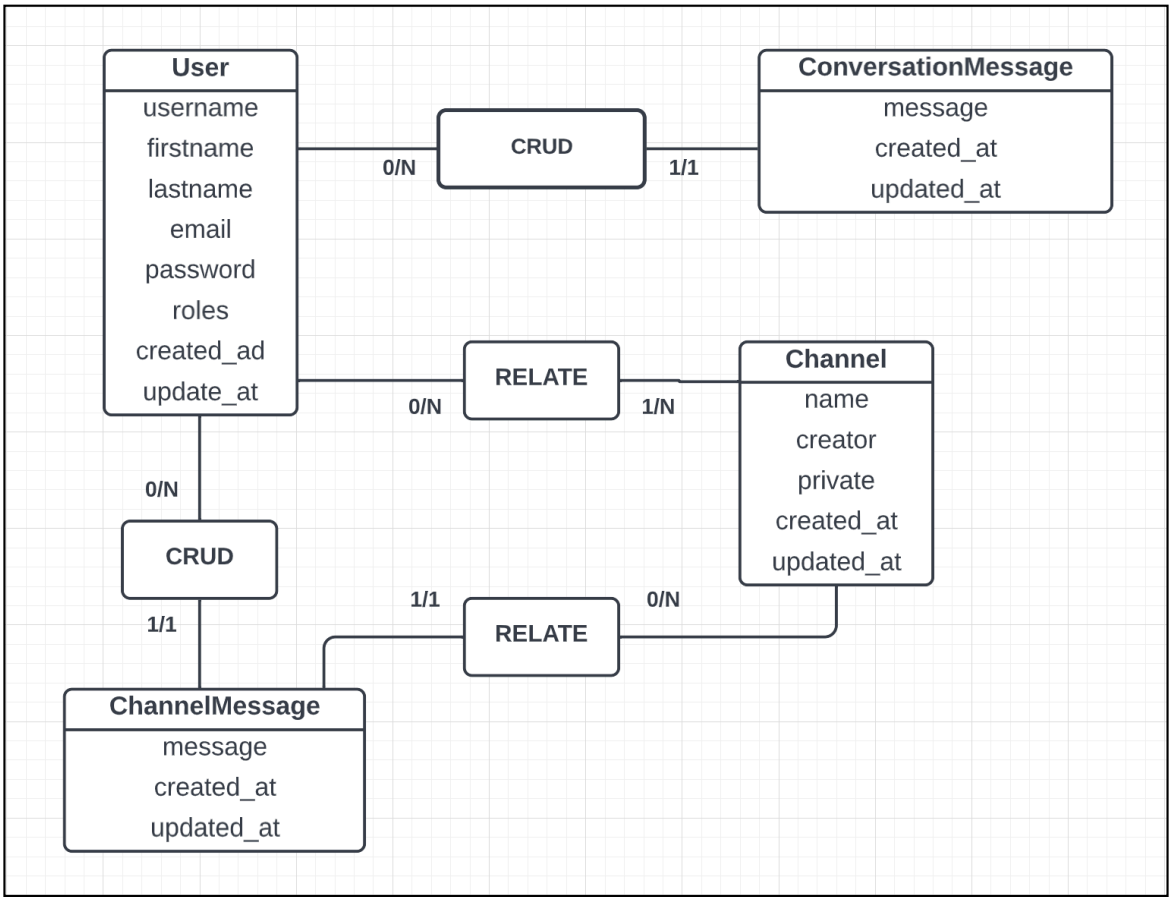
Je soussigné(e) RÉMI LOPEZ, déclare sur l’honneur que les renseignements fournis dans ce dossier sont exacts et que je suis l’auteur(e) des réalisations jointes.

Fait à MARSEILLE le 08/03/2022 pour faire valoir ce que de droit.

Signature : 

ANNEXES

MCD



MLD