



DOSSIER PROFESSIONNEL (DP)

<i>Nom de naissance</i>	► Scognamiglio
<i>Nom d'usage</i>	► Scognamiglio
<i>Prénom</i>	► Laura
<i>Adresse</i>	► 57 allée Léon Gambetta 13001 Marseille

Titre professionnel visé

Concepteur Développeur d'Applications

MODALITÉ D'ACCÈS :

- 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

Intitulé de l'activité-type n° 1	p. .	5
► Intitulé de l'exemple n° 1	p.	p.
► Intitulé de l'exemple n° 2	p.	p.
► Intitulé de l'exemple n° 3	p.	p.
Intitulé de l'activité-type n° 2	p. .	
► Intitulé de l'exemple n° 1	p.	p.
► Intitulé de l'exemple n° 2	p.	p.
► Intitulé de l'exemple n° 3	p.	p.
Intitulé de l'activité-type n° 3	p. .	
► Intitulé de l'exemple n° 1	p.	p.
► Intitulé de l'exemple n° 2	p.	p.
► Intitulé de l'exemple n° 3	p.	p.
Titres, diplômes, CQP, attestations de formation (<i>facultatif</i>)	p. .	
Déclaration sur l'honneur	p. .	
Documents illustrant la pratique professionnelle (<i>facultatif</i>)	p. .	
Annexes (<i>Si le RC le prévoit</i>)	p. .	

**EXEMPLES DE
PRATIQUE
PROFESSIONNELLE**

DOSSIER PROFESSIONNEL (DP)

Activité-typ

e 1 Développer des composants d'interface

Exemple n°1 ▶ Chuu Chat

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

Chuu Chat est une application dédiée à la communauté de fans de K-pop. Suite à la demande de nombreux passionnés qui cherchaient un espace sécurisé pour discuter de leurs groupes préférés, cette plateforme a vu le jour.

Elle a pour mission de faciliter les échanges entre fans tout en offrant des fonctionnalités spécifiques comme la création de rooms pour des groupes spécifiques, la gestion de profil, ou encore la mise en place d'emojis personnalisés pour enrichir les conversations.

Scénarios utilisateurs :

Après une analyse approfondie des besoins de la communauté K-pop, plusieurs scénarios utilisateurs ont été esquissés, offrant une perspective claire sur les attentes des fans vis-à-vis de l'application.

- Page d'accueil : Affichage des rooms actuellement populaires
- Menu de navigation : Les options comprennent "Accueil", "Messages", "Profil", d'autres fonctions sont disponibles au sein des pages.
- Profil utilisateur : L'option de personnaliser son profil, y compris son groupe de K-pop préféré, sa photo de profil, et d'autres informations pertinentes.
- En cliquant sur une "Messages", l'utilisateur est dirigé vers une interface de chat en direct avec d'autres fans.

Technologies choisies :

Suite à l'élaboration du cahier des charges, j'ai recherché la technologie la plus adaptée pour réaliser ce projet. Ayant déjà une expérience avec React Native et Node.js, ces technologies sont devenues le choix évident pour moi. Node.js est excellent pour les applications en temps réel telles que les chats, et React Native facilite la création d'interfaces intuitives.

Node.js, en combinaison avec WebSockets, permet de fournir une expérience de chat en temps réel. De plus, la sécurité des utilisateurs est renforcée par des packages comme bcrypt pour le hashage des mots de passe.

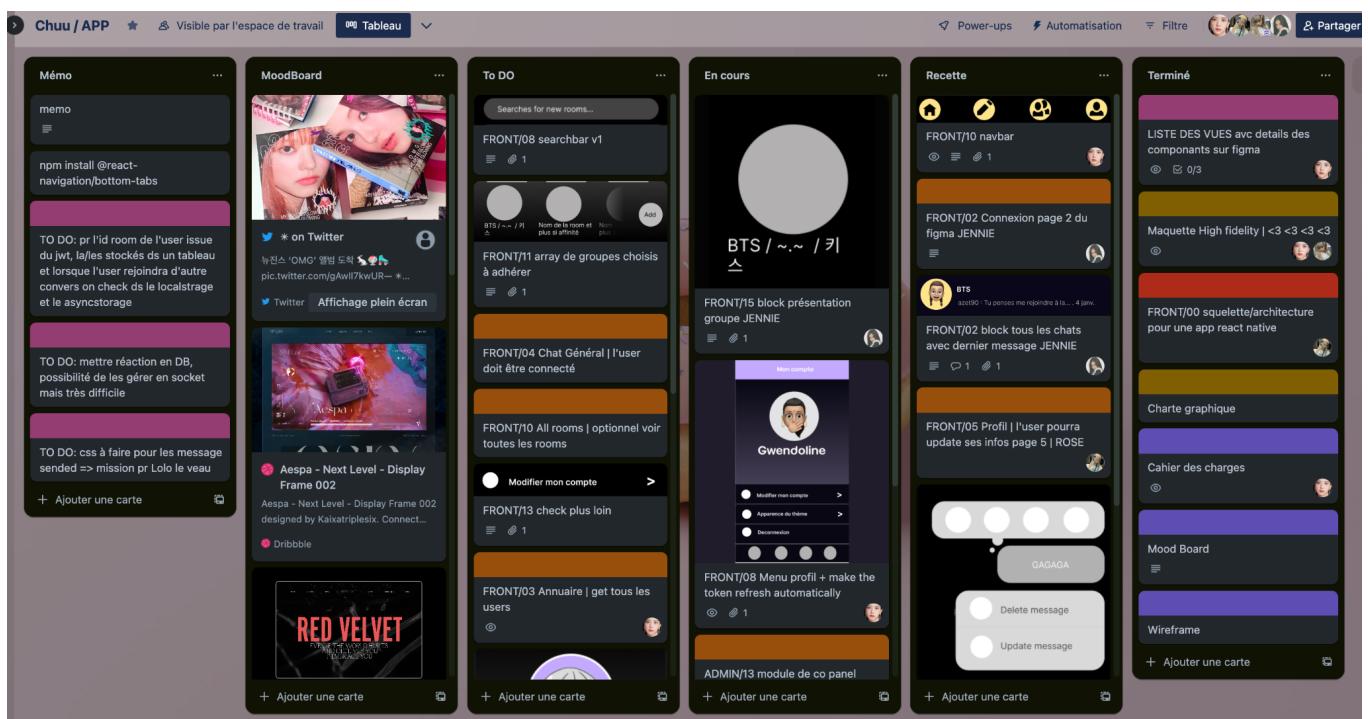
Pour la gestion des données, j'ai utilisé une base de données relationnelle, et pour l'interface entre cette base de données et l'application, une solution ORM (Object-Relational Mapping) a été intégrée.

DOSSIER PROFESSIONNEL (DP)

Conception visuelle :

Un mood board a été créé en amont des choix visuels et suite à une recherche approfondie des tendances actuelles du design liées à la K-pop, un wireframe puis une maquette high fidelity ont pu être élaborées.

Sur Trello l'un de nos tableaux était dédié aux interfaces



wireframe de la web app :

DOSSIER PROFESSIONNEL (DP)

The wireframe illustrates the administrative interface of the Dossier Professionnel (DP) application, organized into several sections:

- Admin**: A page with a large red 'X' indicating it is not yet developed.
- Admin / message**: A list of messages categorized under "Général", "Room 1", and "Room 2". It includes navigation arrows and buttons for "Delete" and "Update".
- Admin / room**: A list of rooms (Room 1 to Room 4) with "Delete" and "Update" buttons. It also features a "Creation Room" button and navigation arrows.
- Admin / User**: A list of users with their login, email, and role information. Each entry has a "Delete" and "Update" button. Navigation arrows are present at the bottom.
- Admin / Delete , update message**: A page showing a list of messages with small edit icons next to them.
- Admin / Formulaire update room**: A form for updating a room, containing fields for "Nom de la room" and an "Update" button.
- Admin / Formulaire modification user**: A form for modifying a user, containing fields for "Login : new", "Role user : new", and an "Update" button.
- Admin / Formulaire création room**: A form for creating a new room, containing fields for "Nom de la room" and a "Création" button.

DOSSIER PROFESSIONNEL (DP)

The image displays a grid of 10 mobile application screens, likely from a wireframe or design prototype, illustrating various features of a social networking platform:

- connexion**: A "SIGN IN" screen with fields for "your login:" and "your password:", and links for "NEW CHUU CHAT" and "SIGN UP HERE".
- inscription**: A "SIGN UP" screen with fields for "your login:", "your email:", "your password:", and "your confirm password:", along with a "Next" button.
- Mon compte**: A user profile screen for "Gwendoline" showing options to "Modifier le compte", "Sécurité et confidentialité", "Apparence du thème", and "Deconnexion".
- Display all rooms**: A screen showing a list of rooms to "Add new chuu chat rooms!", with placeholder names like "BTS / ~ / 카드" and "Nom de la room et plus si affinité".
- Display all my roo...**: A screen showing a list of contacts with messages, such as "Frais Vallongoria" and "Marie_golo".
- confirm deconne...**: A confirmation dialog asking "Voulez-vous vraiment vous déconnecter?" with "Oui" and "Non" buttons.
- Mon avatar**: A screen for modifying the user's profile picture, showing the current picture and input fields for "Login" (Gwendoline) and "Email" (gwen@protonmail.com).
- chat(room)**: A chat screen for the group "BLACKPINK" between users Lisa and Rosé, with messages from 02 OCT. 2022 at 18:34 and 05 OCT. 2022 at 19:50.
- chat messages pr...**: A message preview screen for Romain on 07 OCT. 2022 at 19:50.
- chat(room)**: A detailed view of the "BLACK-PINK" chat between Lisa and Romain, showing a message from Romain with options to "Delete message" or "Update message".
- confirmation delete**: A confirmation dialog asking "Êtes-vous sûr de vouloir supprimer ce message?" with "Oui" and "Non" buttons.
- Update message**: A screen for updating a message from Rosé on 02 OCT. 2022 at 18:34, with a text input field and a "Modification du message" button.

DOSSIER PROFESSIONNEL (DP)

Wireframe

Comme montré si dessus, nous devions former le “squelette” de notre application, c'est-à-dire le wireframe aussi dit la maquette “low fidelity” (“low-fi”). Il permet la visualisation complète de la structure de l'application. Il ne doit pas être détaillé mais doit comprendre l'ensemble des éléments, leur agencement et l'articulation des uns avec les autres. C'est une étape importante de la conception UX (user experience) qui permet de réfléchir sur le côté instinctif de l'application et de son accessibilité. Ainsi que d'avoir une première vue d'ensemble sur les différents composants à réaliser, ceux qui seront réutilisables par d'autres plus grands etc...c'est utile si l'on code en React.

Nous l'avons réalisé avec Figma mais il est tout à fait possible de le faire à l'aide d'un crayon sur une feuille de papier.

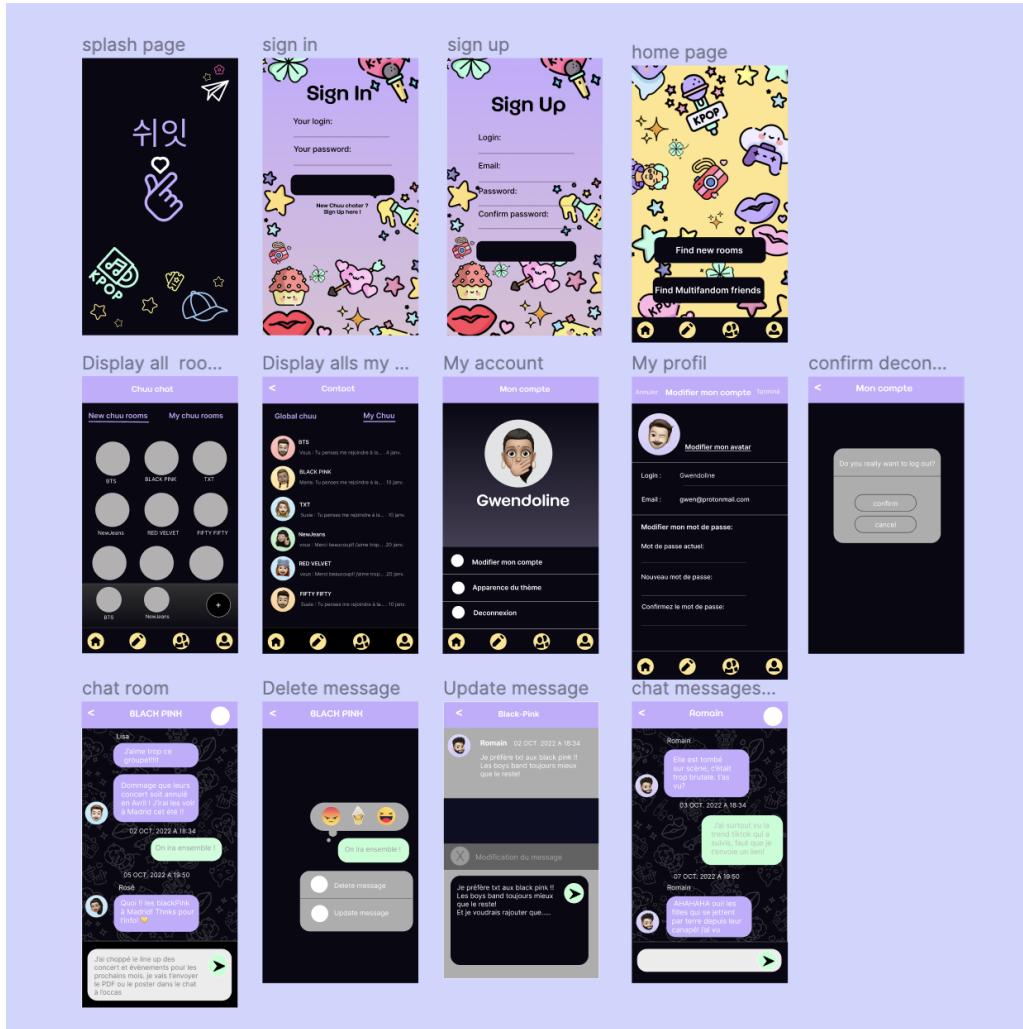
Étant une application mobile, la conception a été, de manière logique, mobile first (ça aurait été le cas également si l'application pouvait être web).

Maquette high-fidelity

Lorsque la maquette low-fidelity est validée, la réalisation de la maquette high-fidelity peut commencer. Cette étape consiste à merger l'UI (user interaction) et l'UX ensemble - afin que cette maquette puisse ressembler le plus possible au produit final. Elle permet un appui aux développeurs et un guide notamment pour la partie “front”. La maquette finale devrait également comprendre un design graphique. Ce dernier a pour objectif de réunir tous les éléments visuels, graphiques etc (couleurs, typographie, espacements entre les éléments...) dans un même système standardisé. Ce dernier peut, bien évidemment, évoluer mais il permet aux développeurs de construire des normes stylistiques à réutiliser dans tout le site/l'application afin d'avoir une homogénéité. Par exemple à l'aide de la mise en place de mixins.

Nous les avons réalisés à l'aide de Figma.

DOSSIER PROFESSIONNEL (DP)



L'application mobile contient plusieurs pages qui sont accessibles via une tab-bar - barre de navigation.

Page de connexion : Elle sert de page "d'accueil" - lorsque l'utilisateur lance l'application, il est invité à se connecter à l'aide d'un nom d'utilisateur et d'un mot de passe. S'il ne possède pas de compte sur le chat, il peut alors s'inscrire.

L'idée étant de hermétiser l'application dès le départ.

Page d'inscription : Sur celle-ci, l'utilisateur qui n'a pas de compte peut en créer un afin de participer à la vie de l'application. Il aura besoin d'un pseudo et d'un mot de passe (qu'il devra confirmer également).

DOSSIER PROFESSIONNEL (DP)

Page “feed” : Une fois connecté, l’utilisateur est dirigé vers une page recensant l’ensemble des rooms auxquelles il appartient (ainsi que le dernier message envoyé dessus). Il y a également un onglet pour filtrer seulement la chatroom globale et la différencier des autres rooms plus “privées”. L’utilisateur peut cliquer sur une des rooms pour y voir les messages.

Page messages : Une fois cliqué sur une discussion, l’utilisateur va pouvoir voir tous les messages de la room. Il peut également poster un message lui-même.

Page “find new rooms” : Il existe également une page permettant à l’utilisateur de voir toutes les rooms dans lesquelles il n’appartient pas encore. Il peut ainsi en cliquant dessus s’y inscrire s’il le souhaite. S’il est banni d’une room, il ne peut pas s’y inscrire.

Page profil : Sur celle-ci, l’utilisateur peut voir ses informations personnelles. Il peut changer son avatar, son pseudo et son mot de passe.

L’application web est exclusivement accessible par l’administrateur afin de pouvoir administrer l’application et ses utilisateurs.

Page index : Une simple page introductrice.

Page users : L’admin peut update les rôles des utilisateurs (donc les bannir) et leur pseudo.

Page rooms : L’admin peut supprimer une room et en créer une et update un nom.

2. Précisez les moyens utilisés :

DOSSIER PROFESSIONNEL (DP)

Éditeur de texte

J'ai utilisé Visual Studio Code (VS Code), un éditeur de code développé par Microsoft. Avec diverses extensions et langages de programmation, VS Code offre une interface utilisateur efficace et fluide, facilitant la programmation avec Node.js et React Native.

Création des maquettes

J'ai opté pour Figma pour la conception des maquettes. Figma est un outil de design d'interface basé sur le cloud qui permet la collaboration en temps réel. Il est particulièrement apprécié pour sa facilité d'utilisation et la fluidité de ses transitions entre le design et le prototypage.

Cahier des charges

Nous avons rédigés et appliqués au mieux un cahier des charges, il regroupe les points clés suivant:

Description de l'existant , Les utilisateurs du projet, Les fonctionnalités du projet, Budget, Temps.

Gestion de projet

Pour organiser les tâches, gérer le workflow et collaborer efficacement avec mon équipe, j'ai utilisé Trello. C'est un outil de gestion de projet qui permet de visualiser clairement les tâches à faire, en cours et terminées.

Environnement de développement

J'ai travaillé avec Node.js, une plateforme d'exécution JavaScript côté serveur, idéale pour développer des applications réseau évolutives. Elle m'a permis d'implémenter efficacement les fonctionnalités back-end nécessaires à l'application.

En complément, React Native a été mon choix pour le développement de l'application mobile. C'est un framework qui permet de créer des applications natives pour iOS et Android à partir d'un seul codebase JavaScript.

Documentations

En ce qui concerne Node.js, le site officiel est une excellente ressource pour comprendre ses nombreuses fonctionnalités et packages. Mais surtout StackOverflow

Pour React Native, la documentation officielle est une source inestimable d'informations sur les composants, les API et les meilleures pratiques.

Quant à Figma, j'ai pu approfondir mes compétences grâce aux tutoriels et ressources disponibles sur leur site.

3. Avec qui avez-vous travaillé ?

j'ai travaillé avec Naomie Monderer Laura Savickaite et Dorian Palace

4. Contexte

DOSSIER PROFESSIONNEL (DP)

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

Chantier, atelier, service **Chuu Chat**

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

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

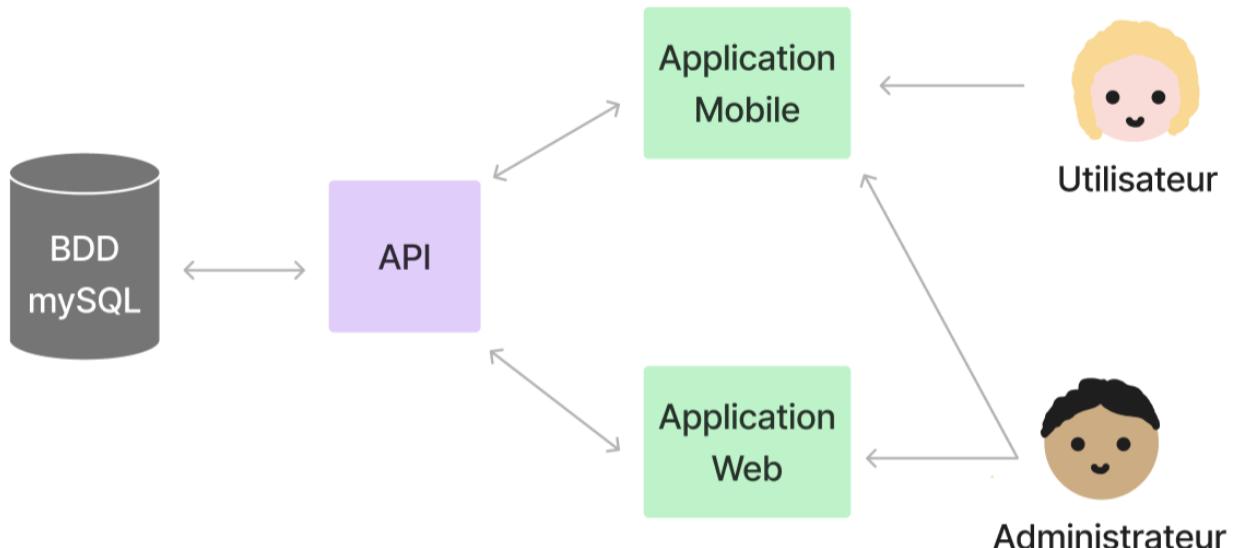
Activité-type 2 Développer la persistance des données

Exemple n° 1 ▶ Chuu Chat

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

1. Phase d'analyse:

Pour mieux comprendre et le cheminement des données nous avons schématisé les relations entre L'API, les applications et la BDD:



Dans notre architecture logicielle, nous avons prévu différents niveaux d'accès pour nos utilisateurs. Les utilisateurs standards pourront utiliser l'application mobile, tandis que les administrateurs auront accès à la fois à l'application web et mobile.

Pour assurer une bonne synchronisation des données, nous utilisons une API commune pour l'application mobile et le site web. Cette API sert d'intermédiaire entre nos interfaces utilisateur et notre base de données MySQL.

Ainsi, lorsque nous, en tant qu'utilisateurs, interagissons avec l'application mobile ou le site web, nos demandes sont envoyées à l'API. L'API se charge alors de communiquer avec notre base de données MySQL pour récupérer ou enregistrer les informations nécessaires.

DOSSIER PROFESSIONNEL (DP)

Ce choix d'architecture nous permet de gérer les données de manière centralisée et d'offrir une expérience utilisateur cohérente entre l'application mobile et le site web. De plus, cela facilite la maintenance et les mises à jour, car les modifications apportées à l'API sont automatiquement prises en compte dans les deux interfaces utilisateur.

Ce travail m'a conduit à définir :

Les données essentielles telles que les messages, les utilisateurs et les groupes de discussion.
Les données composées comme le statut de lecture d'un message.

De la sécurité à travers l'API et l'utilisation de Middleware, hachage de mot de passe, requête préparées etc..

middleware:

```
// [BACK/07] get the details from 1 user
router.get('/details/:userId', signIn, getUserDetails);
```

requête préparée:

```
const addUserToRoom = (req, res) => {

  const verifyRoles = `SELECT role FROM users INNER JOIN roles
  ON roles.id = users.id_role WHERE users.id = ?`
  db.query(verifyRoles, [req.user.id], function (error, data) {
```

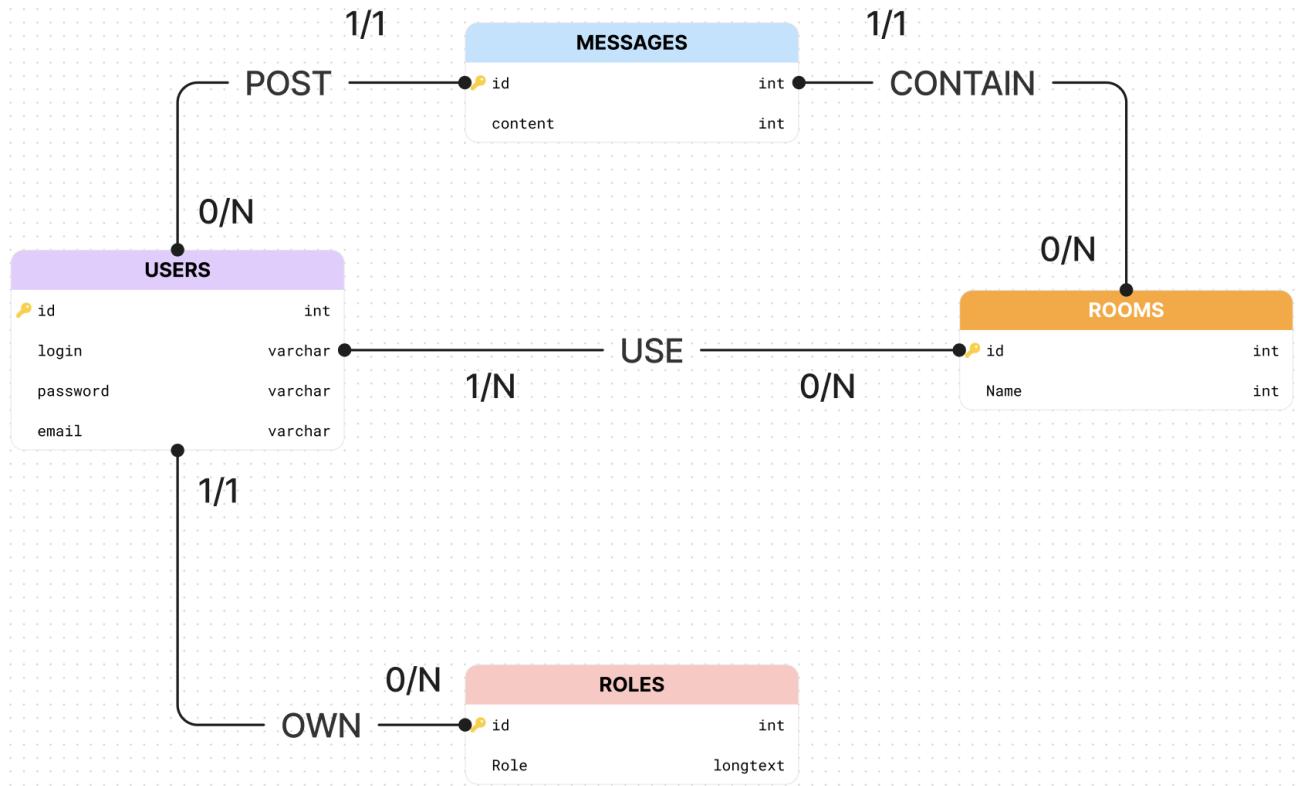
2. Phase de conception technique:

I. J'ai élaboré le Modèle Conceptuel de Données (MCD) pour mettre en évidence les relations entre les entités telles que :

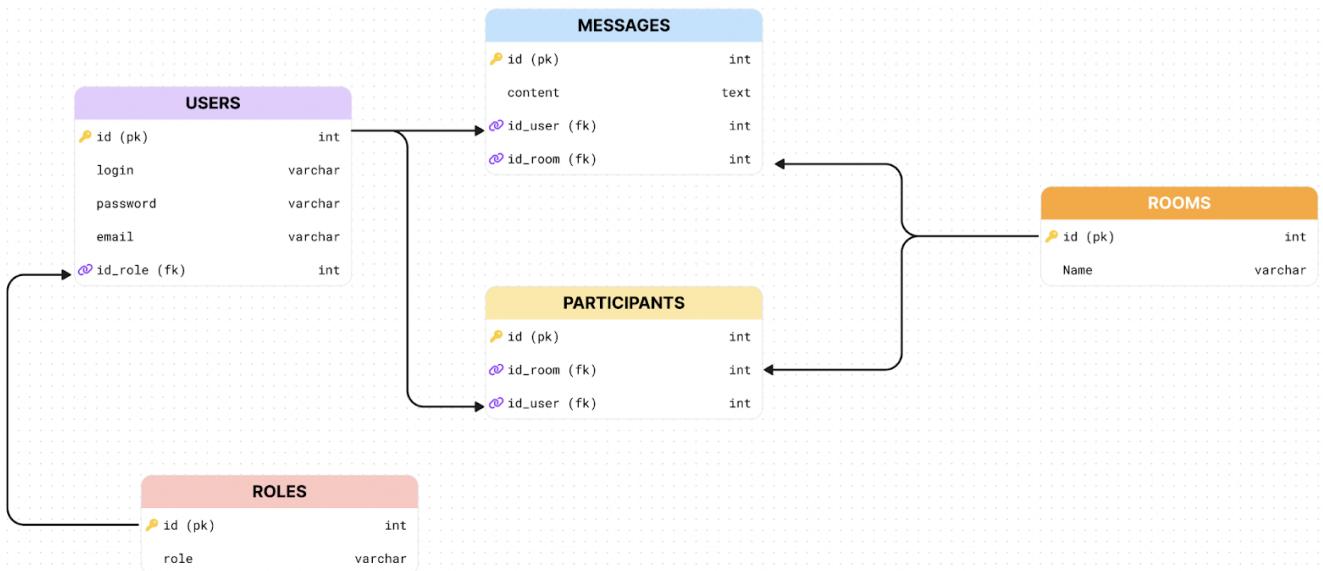
Un utilisateur peut envoyer aucun ou une multitude de message

Un message est associé à un seul émetteur et appartient obligatoirement à une seule room

DOSSIER PROFESSIONNEL (DP)



II. Le Modèle Logique de Données (MLD) a été conçu ensuite pour détailler la structure de la base de données.



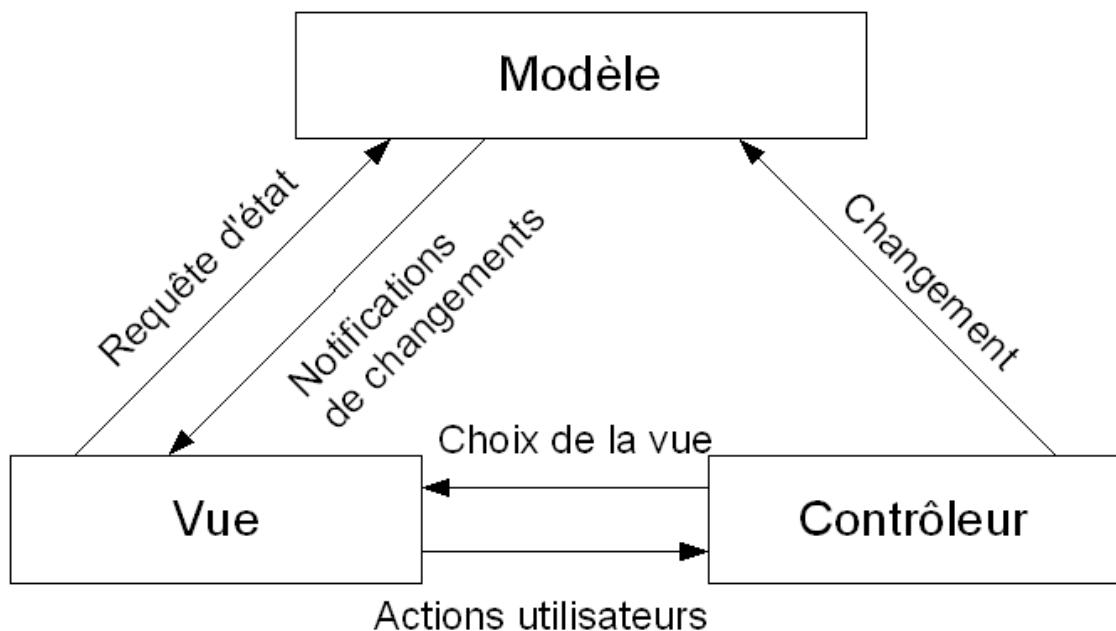
III. Le Modèle Physique de Données (MPD) n'a pas été réalisé car le projet est encore en cours

DOSSIER PROFESSIONNEL (DP)

de réalisation mais il est utilisé pour représenter les entités, leurs attributs (par exemple, varchar, int) et les clés étrangères.

IV. Suite à l'identification des exigences essentielles de "Chuu Chat", en parallèle j'ai structuré et choisi une architecture MVC:

Le MVC (model, view, controller) est un design pattern permettant de structurer la manière de coder. Le modèle (côté serveur) détient les données, il permet de les récupérer de la base de données et de les gérer afin d'être envoyés vers le controller. Le controller, lui, traite ces données et les manipulent, servant ainsi d'intermédiaire entre le modèle et la view. Il fait référence à la logique du code. Une fois ces données maniées, le controller peut laisser place à la view (côté client) qui va présenter ces données à l'utilisateur.



Source : <https://rosedienglab.defarsci.org/>

3. Phase de réalisation:

Le routeur utilisé est celui d'express. Il se forme simplement d'un URI et d'une requête (get, post, etc) : "app.requête(route, fonction)".

"App" est une instance d'express, la "requête" est la requête http, la route se présente sous forme d'uri et la fonction est le controller qui traite la donnée et renvoie une réponse au client.

DOSSIER PROFESSIONNEL (DP)

Nous avons dans un premier temps, un index.js qui recense l'ensemble des routes principales. La première ligne indique donc à l'application que pour toutes les routes ayant un URI commençant par “/users”, il faudra aller chercher la suite dans le dossier users.

```
var users = require('./src/routes/users');
```

Ce dernier comporte l'entièreté des routes, requêtes possibles, offertes pour la partie “users”. Cela nous évitait également de devoir préciser /user à chaque route users.

```
//Users route
app.use('/users', users);

//Participants route
app.use('/participants', participants);

// Verify route
app.use('/connected', signIn, users )

//Admin route
app.use('/admin', [signIn, isAdmin], admin)

//Message route
app.use('/chat', chat);

//Rooms route
app.use('/rooms', rooms);
```

Cette requête commence par une instance du routeur d'express que nous appelons plus haut dans le fichier.

```
var router = express.Router();
```

La route permet donc d'obtenir des données dès qu'on demandait /users (ici tous les utilisateurs). Lorsque cette requête est lancée, le routeur va trouver le controller correspondant avec la fonction indiquée, ici getUsers.

DOSSIER PROFESSIONNEL (DP)

```
var {
    registerUsers,
    authUsers,
    connectedUser,
    addUserToRoom,
    getUsers,
    getUserDetails,
    getUserRole,
    updateUser,
    refreshToken,
    getAllFromUsers
} = require('../controllers/usersController')
```

```
const getUsers = (req, res) => {
    const sql = 'SELECT `login` FROM users'
    db.query(sql, function (error, data) {
        if (error) throw error;
        else res.send(data);
    })
}
```

Il est important de noter que la manière dont une requête est réalisée de la sorte, nous avons d'abord le port et l'URI qui suit.

```
// [BACK/03-get-all-users]: Une route qui
router.get('/', getUsers);
```

<http://localhost:3000/admin/users/1/update/role>

Middleware:

DOSSIER PROFESSIONNEL (DP)

Le middleware est un logiciel qui permet une connectivité entre au moins deux applications ou des composants d'application. Dans notre projet, il prend la forme d'une fonction qui se dote également d'un rôle précis, fournit un service précis. Se situant entre deux couches logicielles, dans ExpressJS on le retrouve généralement entre la requête et la réponse. De base, le framework possède quelques middleware mais il est possible d'en créer au besoin de l'application. D'ordre général, il peut configurer et contrôler les connexions et les intégrations, gérer le trafic de manière dynamique via des systèmes distribués et sécuriser les connexions et le transfert de données. C'est principalement cette dernière fonctionnalité qu'on a utilisé pour notre projet.

Un middleware peut être appliqué à une unique route ou à plusieurs.

```
// [BACK/07] get the details from 1 user
router.get('/details/:userId', signIn, getUserDetails);
```

Ici, le middleware en question est signIn - avant de se diriger vers le controller attendu, la requête va être mise "en suspens" pour que des données nécessaires soient vérifiées. On ne veut pas que n'importe qui ait accès aux détails d'un utilisateur, on veut seulement que les membres inscrits et connectés puissent le faire. SignIn nous permet de le faire en vérifiant le token de l'utilisateur (s'il y a bien un token).

```
const mySecret = "mysecret";
const decoded1 = jwt.verify(tokenToUse, mySecret);
req.user = decoded1;
const decoded2 = jwt.decode(tokenRefresh)
var now = new Date().getTime() / 1000;
```

Dans le cas où ce n'est pas le cas, la requête se verra automatiquement échouée.

```
} catch (err) {
    // console.log('auth.js : ', err);
    return res.status(401).send(err);
}
```

Cela permet également de rajouter une couche de sécurité et de pouvoir rediriger les utilisateurs de l'application vers les services disponibles selon leur statut.

Controller:

DOSSIER PROFESSIONNEL (DP)

Le rôle classique du controller est de gérer les réponses renvoyées par le model, les traiter et les envoyer sur la view. Il fait office d'intermédiaire entre les deux pôles, ainsi qu'entre le serveur et le client.

Dans notre application, nous avons décidé de nous passer de model et de tout rassembler dans le controller pour une raison de simplicité principalement. Alors que le model s'occupe de réaliser les demandes à la base de données d'ordinaire, nous avons intégré ceci dans le controller.

```
const getAllFromUsers = (req, res) => {
    const sql = 'SELECT * FROM users'
    db.query(sql, function (error, data) {
        if (error) throw error;
        else res.send(data);
    })
}
```

Aussi ce dernier détient et envoie non seulement la requête sql mais également s'occupe de la traiter s'il est nécessaire.

La plupart de nos fonctions sont similaires à celle présentée au-dessus, à savoir qu'il n'y a qu'une seule requête et un traitement minime (justifiant encore une fois notre décision d'utiliser que le controller). Il arrive que le traitement doit être plus spécifique comme ci-dessous.

```
const addUserToRoom = (req, res) => {

    const verifyRoles = `SELECT role FROM users INNER JOIN roles
    ON roles.id = users.id_role WHERE users.id = ?`
    db.query(verifyRoles, [req.user.id], function (error, data) {

        if (data.length !== 0) {
            if (data[0].role !== "ban") {

                const verifyParticipation = `SELECT id_room FROM participants WHERE id_user = ? AND id_room = ?`
                db.query(verifyParticipation, [req.user.id, req.params.idRoom], function (error, dataIdRoom) {
                    if (dataIdRoom[0] == undefined) {
                        var insertUser = insertToRoom(req.body.id_room, req.user.id);
                        res.status(200).send({ message: 'Request succeed.' })

                    } else res.status(400).send({ message: 'The id user ' + [req.user.id] + ' is already related to t
                    }
                } else res.status(400).send({ message: 'You were ban of this room.' })
            }
        }
    })
}
```

DOSSIER PROFESSIONNEL (DP)

Dans cette partie, un utilisateur peut s'ajouter à une room seulement s'il n'est pas banni de celle-ci. Dans un MVC classique, tout le traitement de la data aurait été dans le controller seulement.

Sécurité:

Nous avons vu précédemment comment les middlewares permettaient une première sécurisation. En effet, ils permettent, principalement, dans notre application de vérifier les accès.

Il y a également une mise en place de rôles. En effet, il y en a trois disponibles : l'utilisateur lambda, l'administrateur et l'utilisateur banni - selon l'attribution de ce dernier, l'utilisateur aura différentes possibilités et droits au sein de l'application. Cela se présente sous forme d'un id_role qui est de 1 pour un utilisateur lambda, de 2 pour l'admin et de 0 pour un banni. Il faut savoir que dans le cadre de l'administrateur, les deux points évoqués sont intimement liés :

```
//Admin route
app.use('/admin', [signIn, isAdmin], admin)
```

Effectivement, pour ce dernier, nous avons mis en place un middleware spécifique.

```
exports.isAdmin = (req, res, next) => {
    console.log(req.body)
    console.log('req id role admin', req.body.id_role_admin)

    if (parseInt(req.body.id_role_admin) === 2) {
        return next();
    } else {
        return res.status(400).json({ message: "You are not an administrator" });
    }
}
```

Ici on vérifie l'id du rôle.

Évidemment, de manière classique nous avons les vérifications au niveau de la connexion : login et mot de passe correspondants.

DOSSIER PROFESSIONNEL (DP)

```
const authUsers = (req, res) => {
    const login = req.body.login;
    const password = req.body.password;

    db.query(`SELECT users.id, users.login, users.email, users.id_role, users.password, GROUP_CONCAT(participants.
        console.log("results1: ", results)
        console.log(results.length);
        if (results.length > 0) {
            console.log('compare bcrypt:', bcrypt.compareSync(password, results[0].password))
            if(bcrypt.compareSync(password, results[0].password)) {
                const rooms = results[0].rooms?.split(',')
            }
        }
    
```

Mais il y a également, la mise en place de bcrypt pour le password notamment. Bcrypt permet le hachage du mot de passe en base de données : au lieu d'avoir le mot de passe directement rentré dedans, il est transformé pour éviter la fraude par force brute.

Aussi, il est important de le déchiffrer lors de la vérification du password lors du login.

jennie \$2b\$10\$7tMGWzG2VhbwZW8HAfDFr.phidwV^k44pgFA3RqDxS^Zo...

Enfin, la nouvelle instance de sécurité emmenée est le JWT aussi dit Json Web Token. Le Json Web Token (JWT) est un standard ouvert (protocole de communication) qui échange des informations entre deux parties sous forme d'objet JSON. L'information est signée de manière digitale via un algorithme HMAC ou RSA.

Le jeton fourni est composé de trois parties :

Un en-tête (header), utilisé pour décrire le jeton. Il s'agit d'un objet JSON.

Une charge utile (payload) qui représente les informations embarquées dans le jeton. Il s'agit également d'un objet JSON.

Une signature numérique.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09o1PSyXnrXCjTwXyr4BsezdI1AVTmud2fU4

On peut le décoder grâce au site jwt.io.

Ce jeton est généré lors de la connexion de l'utilisateur : ce dernier entre ses informations et une demande est envoyée au serveur afin de fournir le token. Ce token sera alors vérifié lors des actions réalisées par l'utilisateur : il contient ses renseignements (son id, login, son rôle, ses rooms) et permet de s'assurer que l'utilisateur a des droits d'accès sur certaines rooms etc.

DOSSIER PROFESSIONNEL (DP)

Il a également une durée de vie - nous étions donc obligés de le régénérer.

```
const tokenToUse = req.headers.token1;
const tokenRefresh = req.headers.refreshtoken;
```

Généralement le token et son refresh se trouvent dans le header de la requête.

```
const mySecret = "mysecret";
const decoded1 = jwt.verify(tokenToUse, mySecret);
req.user = decoded1;
var now = new Date().getTime() / 1000;
```

Une fois récupérés, on vérifie le token (const decoded1 - s'il y a une erreur on atterrit dans le catch directement) puis on s'occupe principalement du refreshToken : voir s'il a expiré ou pas.

```
//verification avec la date actuelle, si l'expiration
const decoded2 = jwt.verify(tokenRefresh, mySecret)
var now = new Date().getTime() / 1000;

//mettre decoded2.iat et pas .exp
if (now > decoded2.exp) {
    /* expired */
    //le token est disponible ds le scope grace au callback ds refreshToken du usersController
    return refreshToken(decoded1.id, token => {

        // console.log('first')
        res.status(417).send(token)
    });
}

next();
```

Dans le cas où le refresh est périmé, on relance un nouveau refreshToken.

Le token a été une nouvelle problématique avec laquelle on devait jouer - il y a eu malheureusement beaucoup de problèmes pour le mettre en place et l'appliquer correctement. Cependant, voici quelques pistes d'amélioration qu'avec le recul, nous allons pouvoir mettre en place afin d'utiliser le JWT de manière un peu plus correcte :

Si un utilisateur effectuait une action qui nécessitait une mise à jour de l'état du compte, comme la suppression, le blocage ou la suspension du compte, nous pourrions invalider le JWT associé à cet utilisateur en maintenant une liste blanche ou une liste noire des JWT valides/invalides. Ainsi, à chaque requête, nous vérifierons si le JWT fourni est toujours valide.

Pour limiter la fenêtre d'exposition en cas de changement d'état du compte ou de modification des autorisations, nous pourrions définir des durées de validité plus courtes pour les JWT. Cela nécessiterait une régénération plus fréquente des JWT, mais offrirait une meilleure réactivité en cas de changements importants.

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

API : Interface de programmation qui permet à deux logiciels de communiquer entre eux.

Application mobile et web : Deux interfaces utilisateur qui interagissent avec la base de données via l'API.

BDD (Base De Données) : Où les données sont stockées. Dans ce contexte, c'est une base de données MySQL.

Synchronisation des données : Assurer que les données sont cohérentes entre différentes plateformes ou interfaces.

Middleware : Logiciel intermédiaire qui traite les demandes entre la requête initiale et la réponse finale. Dans ce contexte, il vérifie les accès et sécurise les données.

Hachage de mot de passe : Technique pour sécuriser les mots de passe en les transformant en une chaîne de caractères cryptée.

Requête préparée : Technique pour exécuter des requêtes SQL de manière plus sécurisée.

Modèle Conceptuel de Données (MCD) : Représente les relations entre différentes entités dans une base de données.

Modèle Logique de Données (MLD) : Détaille la structure de la base de données.

Modèle Physique de Données (MPD) : Représente la manière dont les entités sont stockées physiquement, y compris les types de données et les clés étrangères.

Architecture MVC (Model, View, Controller) : Design pattern qui sépare l'application en trois composants interconnectés pour simplifier le développement.

Routeur d'express : Outil qui gère les routes et requêtes dans une application Express.js.

Controller : Gère les réponses renvoyées par le modèle, traite les données et les envoie à la vue.

Sécurité : Mesures prises pour protéger les données et les interactions des utilisateurs dans l'application.

Rôles : Définissent les permissions et les accès pour différents utilisateurs.

Bcrypt : Bibliothèque de cryptage pour hasher les mots de passe.

Json Web Token (JWT) : Protocole ouvert qui échange des informations entre deux parties sous

DOSSIER PROFESSIONNEL (DP)

forme d'objet JSON, généralement utilisé pour l'authentification.

Refresh Token : Jeton utilisé pour obtenir un nouveau JWT lorsque le précédent expire.

3. Avec qui avez-vous travaillé ?

j'ai travaillé avec Naomie Monderer Laura Savickaite

4. Contexte

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

Chantier, atelier, service ► *Chuu Chat*

Période d'exercice ► Du : *01/01/2023* au : *01/07/2023*

5. Informations complémentaires (*facultatif*)

DOSSIER PROFESSIONNEL (DP)

Activité-tye 3 Développer une application n-tiers / multicouches

Exemple n° 1 ▶ Chuu Chat

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

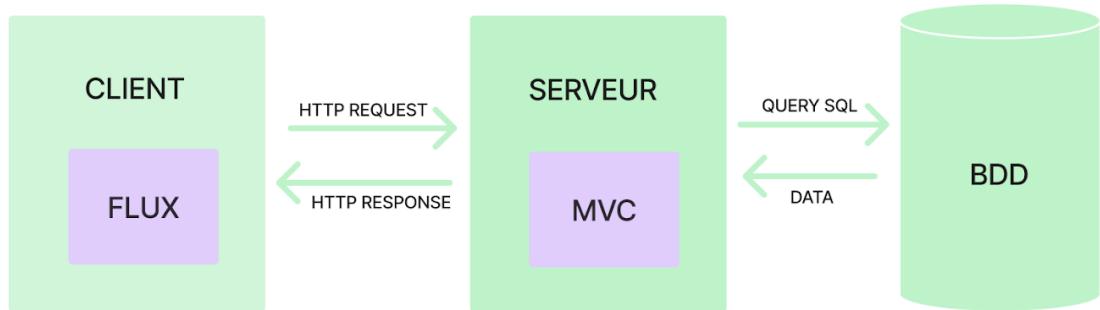
1. Multicouches

Notre projet se base sur une **architecture 3-tiers** (ou plus généralement appelée **multi-couches**). Les différentes couches de l'application sont séparées:

- **La couche client** matérialisée par l'application mobile et web
- **La couche applicative** (couche de logique métier) matérialisée par l'API
- **La couche de persistance des données** matérialisée par la base de données.

Cette architecture 3-tiers est également une architecture "**client-serveur**". Plus précisément, il s'agit de l'environnement dans lequel nos applications de machines clientes (les applications web et mobile) communiquent avec notre application de machines de type serveurs (l'api).

Architecture 3-Tiers



Pour mettre en avant cette architecture 3 tiers je vais vous présenter un jeu d'essai qui naviguera dans toutes les couches de l'application:

Navigation

Pour la navigation, nous avons utilisé React Navigation, notamment Stack qui ajoute un effet de "stacking" entre les pages (installation via npm).

DOSSIER PROFESSIONNEL (DP)

```
return () =>
  <Stack.Navigator
    screenOptions={{
      headerStyle: {
        backgroundColor: '#C5AAFF',
      },
    }}
    initialRouteName={loggedIn ? ROUTES.HOME : ROUTES.LOGIN}>
    <Stack.Screen
      name={ROUTES.HOME}
      component={TabBar}
      options={{ headerShown: false }}>
    />
    <Stack.Screen name={ROUTES.LOGIN} component={Login} options={{ headerShown: false }}/>
    <Stack.Screen name={ROUTES.PROFILE} component={Profil} />
    <Stack.Screen name={ROUTES.REGISTER} component={Register} options={{>
      headerShadowVisible: false, // applied here
      headerBackTitleVisible: false,
      headerTintColor: 'black'
    }}>
  </Stack.Navigator>
};
```

La manière dont le navigateur marche est principalement grâce à des noms et des composants (screens) associés. Le stack.navigator est le composant de base où les screens sont répertoriés. Le notre prend en props initialRouteName qui permet de choisir un nom à la route renderée en premier. On a également un screenOptions qui permet de personnaliser le navigateur.

Ensuite, s'empilent les stack.screens : chacun a un nom et un composant associé - on peut également y ajouter des options comme ici en disant qu'on ne veut pas que le header soit visible. En effet, pour certaines pages, il ne nous était pas utile.

C'est le navigateur à son état le plus basique qui permet une passation entre les pages selon les "press" de l'utilisateur : c'est celui qui permet par exemple dans la page "messages" de cliquer sur une room utilisée par l'utilisateur et d'atterrir sur la page des messages de cette room.

Cependant, notre navigateur a également une tab-bar située sur le bas pour une accessibilité directe avec le pouce de l'utilisateur. Aussi, nous l'avons codé de la sorte :

DOSSIER PROFESSIONNEL (DP)

```
<Tab.Navigator {...{screenOptions}} >
  <Tab.Screen name={ROUTES.CHATROOMS} component={ChatRoomStack} options={{<
    tabBarIcon: ({focused}) => (
      <View style={{alignItems: 'center', justifyContent: 'center',}}>
        <Image
          source={require('..../assets/icons/chuu-purpl.png')}
          resizeMode='contain'
          style={{width: 35,
                  height: 35,
                  tintColor: focused ? '#B2FFDF' : '#ADADAD'}}
        />
        <Text style={{color: focused ? '#B2FFDF' : '#ADADAD', fontSize: 10}}>
          rooms
        </Text>
      </View>
    ),>
    headerShown:false
  >
</Tab.Navigator>
```

De la tab-bar, l'utilisateur pouvait avoir accès à ses rooms, ses messages ainsi qu'à son profil. Le concept est le même : un navigator contient des screens avec un nom qui renvoient à des composants (screens) particuliers.

Socket.io

Afin de mimer la messagerie instantanée, nous avons mis en place Socket.io. Il dispose d'une connexion bidirectionnelle signifiant que le serveur peut pousser des informations (ici messages) aux clients connectés dans la chatroom.

Il faut commencer par installer socket.io via npm dans le projet.

Par la suite, nous avons dû faire une connexion du côté serveur.

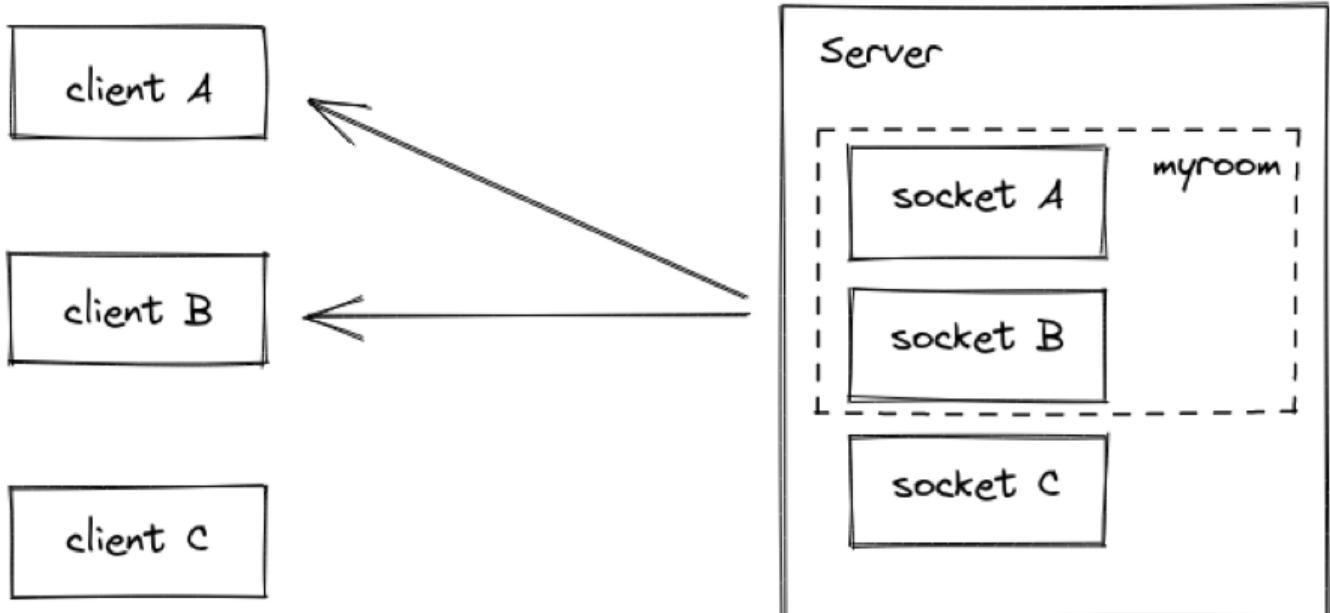
```
const io = require('socket.io')(server)
```

```
io.on('connection', (socket) => {
  console.log('a user connected');
  socket.on('disconnect', () => {
    console.log('user disconnected');
  });
  socket.on('joinIn', (id_room) => {
    socket.join(id_room);
  })
});

server.listen(port, () => {
  console.log(`Socket.IO server running on port :${port}`);
});
```

DOSSIER PROFESSIONNEL (DP)

Cette connexion de socket io se réalise à chacune des rooms qui sont identifiées par leur id grâce à “join”.



Pour que le serveur connaisse les id des rooms, on appose à socket un “événement” nommé “joinIn” - dans l'exemple au-dessus, il est écouté via le “.on”.

Cet événement est émis du côté client dans notre composant ChatRoom qui détient l'ensemble des id des rooms. C'est également dans ce composant qu'on appelle celui des Messages à qui on fait passer la connexion socket en props.

```
import { io } from 'socket.io-client';
```

```
///  
const [socket, setSocket] = useState(io("http://localhost:3000"));  
  
useEffect(() => {  
    socket.emit('joinIn', route.params.id_room)  
    <Messages style={{ height: '40%' }} idRoom={route.params.id_room} socket={socket} />  
}, [route.params.id_room]);
```

Puis, dans le composant Message, on écoutait l'événement “newMessage”.

```
props.socket.on('newMessage', message => setMessages(messages => [...messages, message]));
```

DOSSIER PROFESSIONNEL (DP)

Ce dernier était émis depuis notre contrôleur de messages dans la fonction qui s'occupait d'envoyer un message dans la base de données.

```
io.to(parseInt(req.params.roomId)).emit('newMessage', message)
```

Etant donné que nos messages étaient un useState, une fois qu'on récupérait l'entièreté des messages, on avait un setter qui en faisait un array qu'on pouvait parcourir et donc afficher sur l'écran par la suite.

```
getMessages(data => {
  setMessages(data);
  console.log('getMessages:', setMessages);
})
```

Aussi, on disait à socket que sous l'événement "newMessage", il devait re-set un nouveau tableau avec en plus l'information "fraîchement" envoyée. Ce qui faisait que lorsqu'on "map"-pait notre array de messages, on avait toujours également le dernier message envoyé dans la chatroom.

Socket s'occupait donc d'éviter un refresh qu'on aurait dû avoir en temps normal - de cette manière il avait un accès au côté serveur et envoyait en instantané à tous les clients participants à la room.

Contact page : get answers from the API

Semblable à un jeu d'essai, je développe dans cette sous-partie une fonctionnalité de notre application. Lorsqu'un utilisateur se connecte à l'application, s'il participe à des rooms, il est redirigé vers une page qui répertorie toutes ses chatrooms et le dernier message envoyé dans chacune de ces rooms.

DOSSIER PROFESSIONNEL (DP)

Global chuu

My Chuu



BTS

Vous : Tu penses me rejoindre à la... . 4 janv.



BLACK PINK

Marie: Tu penses me rejoindre à la... . 10 janv.



TXT

Susie : Tu penses me rejoindre à la... . 10 janv.



NewJeans

vous : Merci beaucoup!! j'aime trop... .20 janv.



RED VELVET

vous : Merci beaucoup!! j'aime trop... .20 janv.



FIFTY FIFTY

Susie : Tu penses me rejoindre à la... . 10 janv.

Pour ce faire, dans le screen contact.js, nous appelons l'API et faisons une requête http (ci-dessous "url") au serveur depuis le client.

DOSSIER PROFESSIONNEL (DP)

```
axios({
    method: 'get',
    url:`${API + uri}`,
    headers: {
        'Content-Type' : 'application/json',
        token1: token,
        refreshtoken: refresh
    }
}).then((response) => {
    setContacts(response.data)
})
```

L'uri était le chemin que devait prendre le routeur pour retrouver le controller correspondant.

uri = "/rooms/contact"

Tandis que l'API était une constante qu'on importait.

Le routeur allait donc à l'index.js qui répertorier l'ensemble des "grandes routes".

```
//Rooms route
app.use('/rooms', rooms);
```

Ensuite, il retrouvait la route correspondant aux rooms.

```
// route get dernier message, d'un chat name etc...dans lequel le participant participe
router.get('/contact', signIn, displayRoomsAndChat);
```

Et dans cette dernière, il retrouvait enfin le controller demandé.

Dans roomsController.js, on trouvait donc :

```
// get dernier message, d'un chat name etc...dans lequel le participant est
const displayRoomsAndChat = (req, res) => {
    const sql = `SELECT rooms.id, rooms.name, messages.content FROM messages CROSS JOIN rooms ON messages.id_room = rooms.id WHERE rooms.id = ${req.params.id} ORDER BY messages.id DESC LIMIT 1`;
    db.query(sql, function(error, data){
        if (error) throw error;
        else res.status(200).send(data);
    })
}
```

DOSSIER PROFESSIONNEL (DP)

S'il y avait une erreur, elle était traitée. Dans le cas contraire, le serveur renvoyait une réponse positive et les informations demandées par le côté client.

Du côté client, nos contacts étaient un useState alors on “set”-tait (setContacts) la réponse fournie par l'API dans notre state.

```
contacts.length > 0 ?  
  contacts.map((contact) =>  
    <ContactMenu  
      key={contact.id}  
      contact={contact}  
      onPress={() => navigation.navigate(ROUTES.MESSAGES, {id_room: contact.id, room_na  
    />  
)  
:  
<View style={styles.container2}>  
  <Text style={styles.text}>  
    You are not subscribed to any rooms.  
  </Text>  
  <TouchableOpacity style={styles.cta}>  
    <Text style={styles.textCta}> Get a room </Text>  
  </TouchableOpacity>  
</View>
```

Si la réponse était “vide”, donc que l’array contacts se trouvait vide (l’utilisateur ne participait à aucune room), on envoyait une view simple avec un texte lui indiquant qu’il n’avait aucune chatroom.

DOSSIER PROFESSIONNEL (DP)

```
<ScrollView style={styles.bg}>
  <View style={styles.tabs}>

    <TouchableOpacity onPress={() => underlined(2)}>
      <Text style={underline === 2 ? styles.selected : styles.notSelected}>More bands</Text>
    </TouchableOpacity>
    <TouchableOpacity onPress={() => underlined(1)}>
      <Text style={underline === 1 ? styles.selected : styles.notSelected}>My chuu bands</Text>
    </TouchableOpacity>
  </View>
  <View style={styles.container}>
    {
      rooms.length >= 1 ?
      rooms.map((room) =>
        <BlocRoom
          key={room.id}
          name={room.name}
          room={room}
          tab={underline}
          pressRoom={setNewRooms}
          // specialClass={moreRooms.find(moreRoom => moreRoom.id === room.id) ? true : false}
          // touchable={touchable}
          tb = {false}
          disabled = {disabled}
          moreRooms = {moreRooms}
        />
      )
      :
      <Text>No rooms.</Text>
    }
  </View>
```

Ici par exemple, nous avons un “composant” qui est réellement une page entière. On peut d’ailleurs remarquer que BlocRoom y est utilisé, il est instancié plus haut par import.

```
import BlocRoom from '../components/BlocRoom';
```

Il y a, néanmoins, une différence notable entre React JS et React Native. En effet, le React que je mentionne tout au long du dossier est React Native. React Native est utilisé pour la création d’applications mobiles qui s’adaptent aussi bien en IOS qu’en android tandis que React JS est plutôt utilisé pour les applications web qui nécessitent donc un browser. La raison étant que React JS se sert de Javascript, CSS et HTML pour construire les interfaces lorsque React Native use en plus des composants natifs comme on peut le voir sur les captures d’écran du dessus : l’un des plus basiques est le composant `<View>` qui est un conteneur capable de supporter les flexbox, style, contrôle d’accessibilité etc.

Aussi, un détail important à noter est que le style de nos composants était rédigé sur le même fichier et non sur un fichier css dédié. Pour ce faire, on avait créé une constante `styles` qui était au final une fonction avec un retour d’objet que l’on parcourait en donnant la clef souhaitée (`classname`) à notre composant natif.

DOSSIER PROFESSIONNEL (DP)

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 52,
    alignItems: 'center'
  },
  boxTitle: {
    marginTop: 90,
  },
  boxForm: {
    marginTop: 60,
  },
})
```

```
<Text style={styles.title}>
```

2. Tests

Les tests en développement permettent de vérifier que les fonctionnalités développées marchent correctement et que s'il y a des erreurs, ces dernières sont traitées via le code. Il existe plusieurs tests qui peuvent être aussi bien manuels qu'automatisés :

- les tests unitaires : consistent à tester de manière individuelle et isolée chaque fonction du projet en comparant le résultat effectif et le résultat attendu en fonction d'un jeu de données test prédéfinies.
- les tests fonctionnels : consistent à tester une fonctionnalité complète afin de vérifier un scénario d'utilisation de bout en bout (ex = connexion d'un utilisateur à l'application).
- les tests de non régression : consistent à spécifiquement vérifier que la version en cours n'a pas généré de régression par rapport à la version précédente.

Pour un projet aussi minime, nos tests n'étaient pas forcément les plus développés. Néanmoins, nous avons utilisé Postman pour tester notre API et ses requêtes manuellement. Dans un premier temps, nous avons créé la collection Postman recensant l'ensemble des appels possibles vers notre API.

DOSSIER PROFESSIONNEL (DP)

The screenshot shows the Postman application interface. At the top, there are navigation tabs: Home, Workspaces, API Network, and Explore. A search bar says "Search Postman". On the right, there are buttons for "Invite", "Upgrade", and a profile icon. Below the header, the URL "chuu localhost" is shown.

In the main workspace, there is a collection named "chuu localhost". Under this collection, a POST request for "Chuu Chat / USER / SIGN IN" is displayed. The request method is "POST" and the URL is "{{baseURL}}/users/auth".

The "Body" tab is selected, showing the following JSON payload:

```
1 var token = pm.response.json().token;
2 pm.environment.set("authToken", token);
3
4 var rToken = pm.response.json().refresh;
5 pm.environment.set("refreshToken", rToken);
```

On the right side of the interface, there is a table for environment variables:

Variable	Initial value	Current value
login	Naomi	Naomix
password	Naomi999	Naomi999
authToken		eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e yJsb2dpbi6Ik5hb21peCismlhdCI6MTY4N jMXNDQzNCwidHlwZSI6ImF1dGh0b2tbsl mVtYWisjoizW1haWxAdGVzdC5j..
refreshToken		eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.e

Below the table, under "Globals", it says "No global variables".

At the bottom of the interface, there are tabs for Body, Cookies, Headers (9), and Test Results. The "Pretty" tab is selected. The "Test Results" tab shows the response status: 200 OK, Time: 138 ms, Size: 896 B, and a "Save as Example" button. The response body is displayed as:

```
1 {
2   "status": true,
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
yJsb2dpbi6Ik5hb21peCismlhdCI6MTY4NjMxNDQzNCwidHlwZSI6ImF1dGh0b2tbsi  
sImVtYWisIjoiZW1haWxAdGVzdC5jb20iLCJpZCI6IjciLCJpZF9yb2xLI  
joxLCJpZF9yb29tcyI6WyIwIiwiMiJdLCJleHAiOjE20TE00Tg0MzR9.i5Hi0GZhk2Xm7SnBXacmWSuih1V3MKUMIUOKYhCaDS8",
4   "refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
eyJzZXNzYWdlIjoiemVmcmVzaCBUb2tbsiBpbmZvIiwiWF0IjoxNjg2MzE0NDM0LCJ0eXB1IjoidG9rZW4iLCJ1bWFpbCI6ImVtYWisQHR1c3QuY29tIiwibG9naW4i  
iJOYW9taXgilCJpZF9yb29tcyI6WyIwIiwiMiJdLCJpZCI6IjciLCJpZF9yb2xlijoxLCJleHAiOjE20DcxNzg0MzR9.  
cLM1PFxWZ3hB71KFN9cSBz2gxAD3KiyjEUX4VWds9g"
```

At the bottom of the interface, there are various status indicators and links: Online, Find and replace, Console, Import Complete, Runner, Capture requests, Cookies, Trash, and Help.

DOSSIER PROFESSIONNEL (DP)

The screenshot shows a dark-themed API documentation interface. At the top left is a '+' button, followed by a search bar with a magnifying glass icon, and at the top right are three dots (...). The main content area is titled 'Chuu Chat' and contains the following sections:

- USERS**
 - > POST SIGN IN
 - > GET VERIF SIGNIN
 - > POST INSCRIPTION
 - > GET GET ALL USERNAMES
 - > POST UPDATE USER INFO
- PARTICIPANTS**
- ROOMS**
 - > GET GET UNSUBSCRIBED ROOM...
 - > GET GET ALL ROOMS
 - > GET GET LAST MESSAGE FROM ...
 - > GET GET LAST MESSAGE FROM ...
- MESSAGES**
 - > POST POST MESSAGE IN GLOBAL ...
 - > POST POST MESSAGE IN CHATRO...
 - > DEL DELETE MESSAGE
 - > POST EDIT MESSAGE
 - > GET GET MESSAGES BY ROOM
 - > GET GET MESSAGES FROM GLO...
- ADMIN**
 - > DEL (ADMIN) DELETE MESSAGE ...
 - > DEL (ADMIN) DELETE ROOM
 - > POST (ADMIN) CREATE ROOM
 - > PATCH (ADMIN) UPDATE ROOM
 - > PATCH (ADMIN) UPDATE USER NAME
 - > PATCH (ADMIN) UPDATE USER ROLE

DOSSIER PROFESSIONNEL (DP)

2. Précisez les moyens utilisés :

Architecture 3-tiers (Multicouches)

Architecture 3-tiers: Il s'agit d'une architecture qui divise une application en trois composantes principales :

Couche client : interface utilisateur.

Couche applicative : où réside la logique métier de l'application.

Couche de persistance des données : où sont stockées les données de l'application.

Client-serveur: Modèle de communication dans lequel les requêtes des clients sont traitées par un ou plusieurs serveurs. Cette méthode est couramment utilisée dans les applications web modernes.

Navigation

React Navigation: Il s'agit d'une solution de navigation spécifiquement conçue pour les applications React Native. Elle permet aux utilisateurs de naviguer entre différentes vues ou "screens" d'une application mobile.

Communication en temps réel

Socket.io: Bibliothèque permettant une communication en temps réel entre le serveur et le client. Très utilisée pour les fonctionnalités de messagerie instantanée, elle permet aux serveurs de pousser des informations aux clients sans que ces derniers aient à en faire la demande.

API et accès aux données

API (Interface de Programmation d'Application): C'est un ensemble de règles et de spécifications que les applications peuvent suivre pour communiquer entre elles.

HTTP (HyperText Transfer Protocol): Protocole standard utilisé pour la communication entre le client et le serveur sur le web.

URI (Uniform Resource Identifier): Il s'agit d'une chaîne de caractères utilisée pour identifier une ressource, généralement sur le web.

Contrôleur: Dans le développement web, un contrôleur est une partie du code qui reçoit les requêtes HTTP, effectue la logique nécessaire et renvoie une réponse.

Tests

Postman: Outil utilisé pour tester des APIs. Il permet de simuler des requêtes HTTP vers le serveur et de visualiser les réponses.

Tests unitaires: Ces tests vérifient le fonctionnement de portions spécifiques du code de manière isolée.

DOSSIER PROFESSIONNEL (DP)

Tests fonctionnels: Ces tests vérifient le bon fonctionnement d'une fonctionnalité complète, de l'entrée initiale de l'utilisateur jusqu'à la sortie ou le résultat final.

Tests de non-régression: Ces tests ont pour but de s'assurer que les modifications apportées au code (nouvelles fonctionnalités, corrections de bugs, etc.) n'ont pas introduit de nouveaux problèmes.

Développement Frontend

React Native: Framework utilisé pour créer des applications mobiles pour iOS et Android en utilisant JavaScript et React.

React JS: Librairie JavaScript utilisée pour construire des interfaces utilisateur pour des applications web.

Composants natifs: Ce sont des composants spécifiques à React Native utilisés pour construire des interfaces utilisateur pour des applications mobiles.

3. Avec qui avez-vous travaillé ?

Avec Naomie Monderer et Laura Savickaite

4. Contexte

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

Chantier, atelier, service ► *Chuu Chat*

Période d'exercice ► Du : *01/01/2023* au : *01/07/2023*

5. Informations complémentaires (facultatif)

DOSSIER PROFESSIONNEL (DP)

Titres, diplômes, CQP, attestations de formation

(facultatif)

DOSSIER PROFESSIONNEL (DP)

Déclaration sur l'honneur

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

Fait à Marseille le **01/07/2023**

pour faire valoir ce que de droit.

Signature :

Laura Scognamiglio

DOSSIER PROFESSIONNEL^(DP)

Documents illustrant la pratique professionnelle

(facultatif)

Intitulé

Cliquez ici pour taper du texte.

DOSSIER PROFESSIONNEL ^(DP)

ANNEXES

(Si le RC le prévoit)