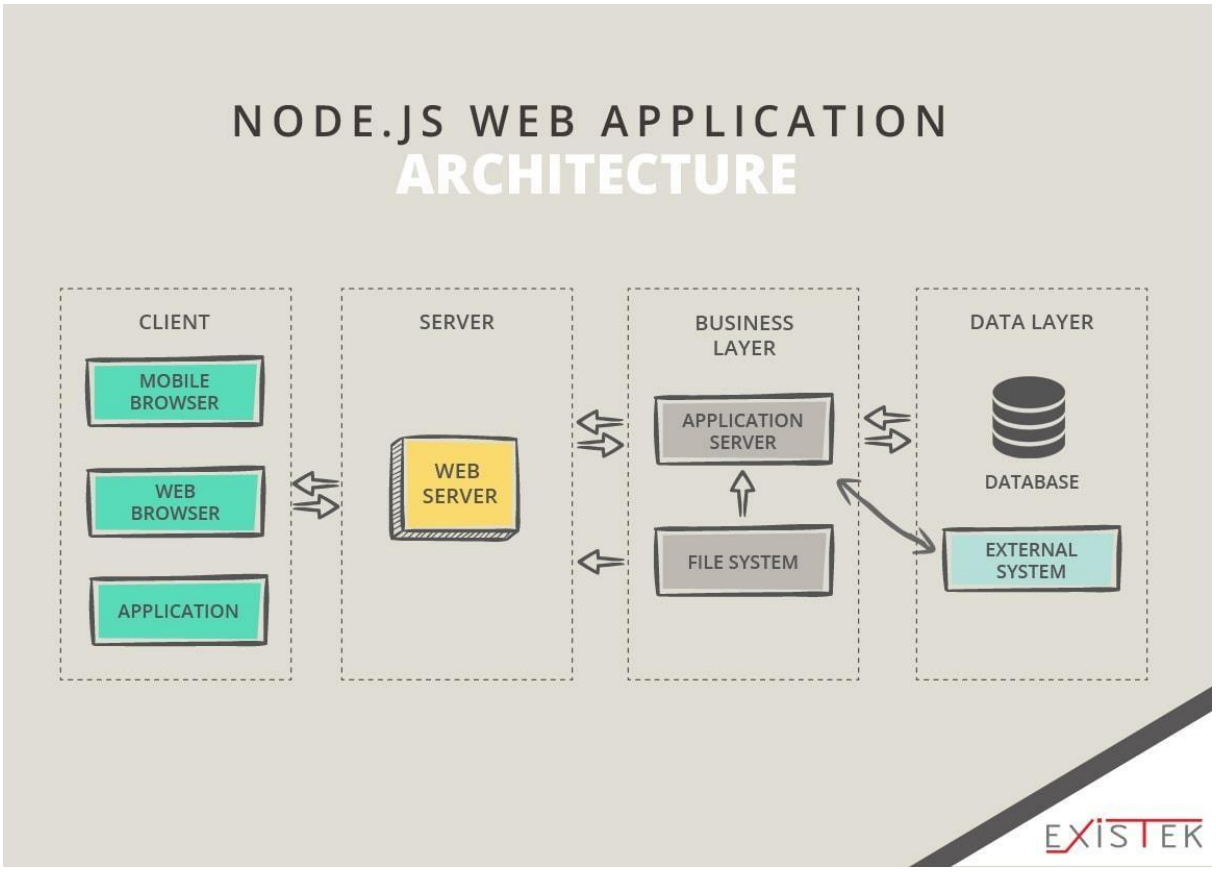
Maitrise technique

Présentation : Cette maitrise technique a pour but d'expliquer en détail la conception ainsi que les choix de conceptions du projet MYAPP.

1) Schéma logique de l'application.

MyApp se présentera sous la forme d'une web app constitué d'un client, d'un serveur web, ainsi que d'une base de donnée. La base technique sera donc une application node.js développé à l'aide du Framework react spécialisé dans ce type d'usage. Mongoose qui permet de créer le business logique permettant l'échange d'information avec la base de données qui sera une base de type nosql. MongoDB sera utilisé afin de stocker les données.



La partie client sera le navigateur de l'utilisateur, qui récupèrera les informations du serveur. Un business logique permet de communiquer les information utilisateur en temps réel avec la base de données.

2) Technique

Notre web-app en développé sous node.js sera donc une application javascript. Afin de simplifier le déploiement, nous utiliserons le Framework javascript REACT qui permet le déploiement rapide et simplifié de web-app.

J'utilise donc un stack MERN :

MangoDB = Base de données nosql

Express = gestionnaire infrastructure Node

JS React = Framework web-app

React Native = Framework app android

Node JS = environnement d'exécution javascript

LeafletReact = gestionnaire de live map React

Je détaille par la suite comment s'articule ces différentes technologies et leurs utilisations à travers ce projet.

2.1) Frontend WebApp

Utilisable par n'importe quel service web, l'application comprendra 1 écrans qui sera une carte ou des marqueurs indiqueront les emplacement des utilisateurs.

2.1.1) React-Router-dom

Se sera REACT-router-DOM (un composant de react) qui gèrera et permettra la navigation entre les pages via des route qui sont associé à des Link ce qui permettra de switcher d'une page à l'autre.

```
const Root = () =>
{
  return (
  <Router>
    <Switch>
      <Route exact path='/' component={App} />
      <Route exact path='/Inscription' component={Inscription} />
```

2.1.2) JSX

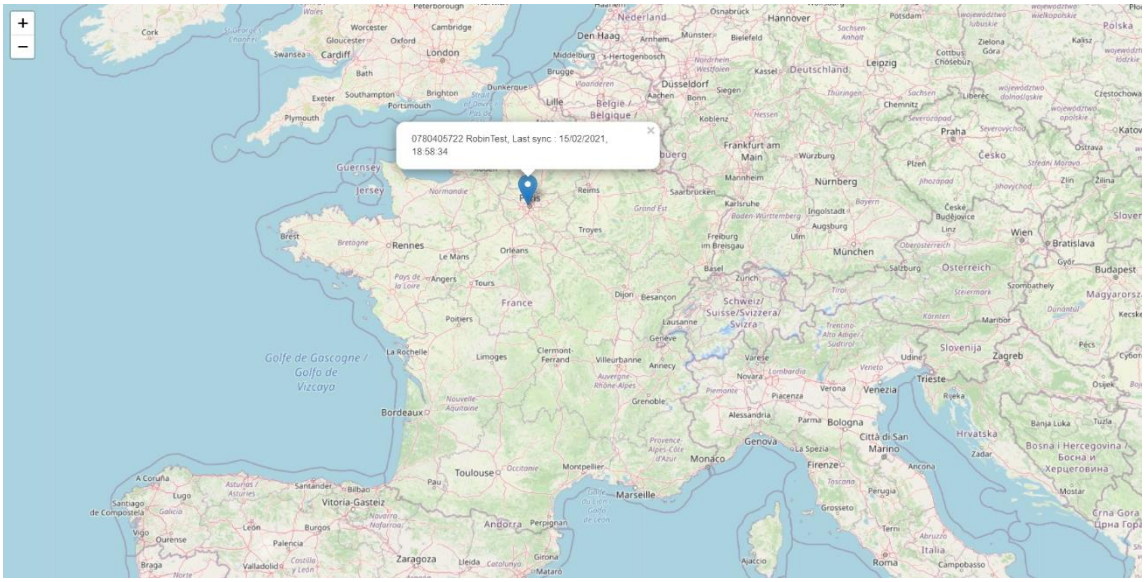
Le JSX est une syntaxe proposée par react permettant de développé en JS (javascript) tout en utilisant une syntaxe très proche du html, ce qui est très utilise dans mon cas on peut parfaitement y implémenter Leaflet React.

2.1.3) Bootstrap

Pour rendre le site parfaitement responsif, nous utiliseront le Framework Bootstrap qui permet de facilité le développement, l'évolutivité et le côté adaptatif du site. De plus une fois intégré au niveau de la business logique, on peut alléger le site en réutilisant une fonction pour changer des éléments graphiques. La version de Bootstrap utilisé est la 4.5.2. Cela nous permettra une fois intégrer au JSX de ne pas avoir à gérer l'intégration.

Le minimum d'élément est utilisé pour garder un site parfaitement fluide et intuitif.

2.1.4) Leaflet React




```
<MapContainer center={defaultposition} zoom={this.state.defaultposition.zoom} style={{height : '800px'}}>
    <TileLayer
    attribution='&amp;copy <a href="http://osm.org/copyright">OpenStreetMap</a> contributors'
    url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
    />
    {}
    <Marker position={defaultposition}>
        <Popup>
            {phoneworker}, {nameworker}
            Last sync : {lastsyncworker}
        </Popup>
        </Marker>
</MapContainer>
```

Afin de générer les marqueurs, nous recouperont le nombre d'utilisateurs enregistrer pour nous répétons une fonction afin de parcourir la base et nous ajoutons le bout de code correspondant à la map qui s'actualise automatiquement. Ce procédé permet des performances très correct, et une réactivité optimale.

2.2) Backend

2.2.1) Buisness logic

Cela correspond à la couche applicative permettant de faire le lien entre la partie serveur et la partie base de données en recevant des donnée utilisateur en entrée saisie via le frontend par exemple. Il faut voir cela comme un « pont » permettant aux données de circuler. De plus notre bisness logic permet pouvoir traiter les données en comparant les réponses de l'élevé au réponse attendu.

Pour ce projet plusieurs plugins vont nous aider à faire fonctionner celle-ci.

➔ BodyParser permettra encore une fois de gérer des requêtes mais cette fois à destination de la base de données via des Endpoint. Cela permet d'actualiser les informations nécessaires sans avoir besoins d'actualisé la page et de pouvoir faire simultanément plusieurs choses en arrière-plan. Exemple ci-dessous de la réception du résultats de l'élève

```
router.get('/',async function(req,res)
{
    const results =await orderModel.find({})
    res.send(results)
})
```

➔ Mongoose permet quant à lui de communiquer avec notre base de données MongoDB. Exemple ci-dessous du contenu de la fonction de connexion à la base.

```
//mongoose
var mongoose = require('mongoose')

mongoose.Promise = Promise
mongoose.connect('mongodb+srv://dbUser:6B6SorPz7dL7fYHQ@cluster0-rkhme.mongodb.net/test?retryWrites=true&w=majority', { useNewUrlParser: true

const db = mongoose.connection;
db.on("error", console.error.bind(console, "connection error:"));
db.once("open", function() {
console.log("Connexion");
});
```

Ces 2 applications ensembles permettent d'échanger des requêtes à donner variables à partir de fonctions développer pour notre application auxquels nous faisons appels quand nécessaires.

2.2.2) Base de données

La base de données mongoDB est hébergé sur Atlas afin de ne pas avoir un service local (autant se simplifier la vie en utilisant un service externe pour une si petite application)

L’architecture

Voici l’architecture de la base de données.

Ces modèles appelés schéma seront appelé dès qu’il faudra créer un utilisateur.

User:

nom = string ;
pass = string ;
phone = string;
lastsync= string;
gps = string;

MongoDB

Ce service nosql permet d’héberger les données sous format json. Des schémas sont prédéfinis ce qui permet ensuite de récupérer les données via la partie javascrit.

Schema > JS UserSchema.js > ...

```
1   const mongoose = require ('mongoose');
2   const Schema = mongoose.Schema;
3
4   //Schema
5   let UserSchema = new Schema({
6     phone: {type: String, required: true},
7     password: {type: String, required: true},
8     nom: {type: String, required: true},
9     gps:  {type: String, required: true},
10    lastsync:  {type: String, required: true},
11  })
12
13
14
15  let User = mongoose.model('User', UserSchema)
16  module.exports = User
```

Exemple de stockage en base de données


```
[
  {
    "_id": "6032985b0f563a6c34456787",
    "phone": "0780405722",
    "password": "$2b$10$sKsYgsuB5k/nOGFHm0YMeupn5CWSyylRok6oz/lr2f8P9DtG2ukIq",
    "nom": "fsdfg",
    "gps": "[51.505, -0.09]",
    "lastsync": "21/02/2021, 18:28:59",
    "__v": 0
  },
  {
    "_id": "603299720f563a6c34456788",
    "phone": "0780405723",
    "password": "$2b$10$9dmMVSAyLa8awmi/CROfbuxIMz7xboOsKdmf7.w7W1EvrUu0SOvl2",
    "nom": "fsdfg",
    "gps": "[51.505, -0.09]",
    "lastsync": "21/02/2021, 18:33:38",
    "__v": 0
  },
  {
    "_id": "603299910f563a6c34456789",
    "phone": "0780405724",
    "password": "$2b$10$zBNcK1XMfLkY7Ibjo48hG.KV1UKYFYyBVLY/TQ7XzSNhSxBM8M8rC",
```

API

L’api est gérée par express qui gère les différentes routes permettant de traiter les différentes requêtes.

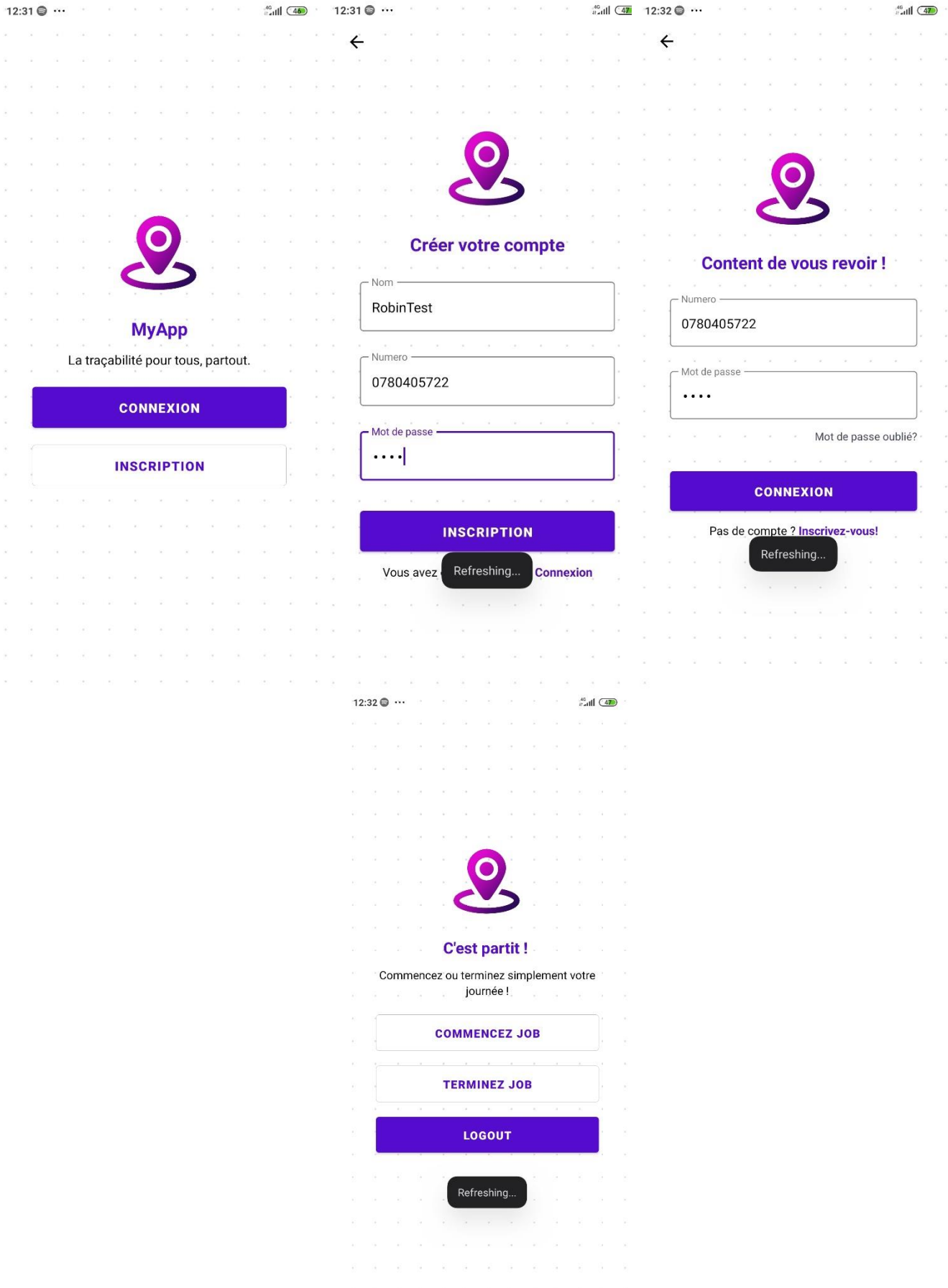
```
//route
exports.router = (function() {
  var apiRouteur = express.Router();

  // Users routes
  apiRouteur.route('/users/register').post(usersCtrl.register) ; //s'enregistrer
  apiRouteur.route('/users/login').post(usersCtrl.login) ; //se connecter
  apiRouteur.route('/users/verify/:phone').get(usersCtrl.verify) ; //verifier si l'email est deja utiliser
  apiRouteur.route('/users/getall').get(usersCtrl.getall) ; //recupere la liste des users
```

```
//imports
const bcrypt = require('bcrypt');
const User = require('../Schema/UserSchema');


//routes
module.exports = {
    //inscription
    verify:async function(req,res){
        console.log("verify")
        const filter = { phone: req.params.phone}
        let user = await User.findOne(filter ,{phone: req.params.phone}, {new :true})
        res.send(user ? true : false)
    },

    register:async function(req,res){
        let user = req.body
        let userpass = (req.body.password)
        try{
            let hash = await bcrypt.hash(userpass, await bcrypt.genSalt())
            user.password = hash;
            var userhashed = new User(user)
            } catch (error) {
            console.log(error.message)
            }
        userhashed.save()
        .then(() => {
        res.status(200).json()
        })
        .catch(err => {
            res.status(400).json(err)
            console.log(err)
        })
    },
```

Les routes sont liées a des fonctions qui permettent de traiter l’information.
L’api fait le lien entre les plateformes utilisateurs et la base de données. Elle utilise le framwork express.

Application Android

L’application android est développé sous react native. Elle permet la création d’un compte, la connexion et de commencer/terminer le travaille donc de faire remonter on nous les données utilisateurs.



```
const onSignUpPressed = async() => {

  const nameError = nameValidator(name.value)
  const numberError = numberValidator(number.value)
  const passwordError = passwordValidator(password.value)
  if (numberError || passwordError || nameError) {
    setName({ ...name, error: nameError })
    setNumber({ ...number, error: numberError })
    setPassword({ ...password, error: passwordError })
    return
  }
  if (await FindUsedNumber() === true){
    setNumber({ ...number, error: 'Ce numero est deja utilisé' });
  } else {
    console.log("OK")

    let user = {phone: number.value, password: password.value, nom: name.value, gps:"[51.505, -0.09]", lastsync: new Date().toLocaleString()  };
    fetch('http://localhost:4000/api/users/register', {
          method: 'post',
          headers: { 'Content-type': 'application/json' },
          body:JSON.stringify(user),
        })
        .then(res => {
          navigation.reset({
            index: 0,
            routes: [{ name: 'LoginScreen' }],
          })
        })
  }
}
```

J'utilise le framework react-native-paper pour la partie graphique qui integre des élément d'intégration graphique. Exemple des boutons et des fonctions ratachées :

```
<Button
  mode="outlined"
  onPress={() => {
    setUser({ isonjob: true })
    jobChecker();
    findCoordinates();
    alert('Vous entrez en fase de travail');

  }}
  title="Press Me"
>
 Commencez job
</Button>

<Button
  mode="outlined"
  onPress={() => {
    setUser({ isonjob: false })
    jobChecker();
    alert('Vous terminez votre fase de travail');

  }}
>
 Terminez job
</Button>
```

FindCoordonate est la fonction qui permet de récupérer les coordonnées GPS du smartphone. Elle est utilisée après jobChecker qui vérifie si l'utilisateur est bien en période de travail. Une fois fait, la valeur retournée est mise a jour en base via une fonction fetch (put), qui met également a jour la valeur lastsync.

```
async function findCoordinates () {
  navigator.geolocation.getCurrentPosition(
    position => {
      const location = JSON.stringify(position);
      console.log(location)
    },
    error => console.log(error),
    { enableHighAccuracy: true, timeout: 20000, maximumAge: 1000 }
  );
};
```

Test :

Les tests se sont révélé etre concluant jusqu'ici. L'impacte sur l'autonomie avec une remonté toute les 90s est négligeable, de meme pour la consomation de data qui est réduite au stricte minimum en n'envoyant que les valeur est non les objects entiers recuperer par les fonctions.

Seul le rafraichissement de la map se bloque parfois de manière « aléatoire » je pense à une probleme liée directement au naviagteur qui stop le rafraichissement apres une certaine inactivité, je dois trouver une solution.

Le reste correspond parfaitement au attente du cahier des charges et permet de passer a une phase d'amélioration des fonctionnalité afin de pouroir prétendre a un produit commercialement intérréssant.