Stage BTS SIO 2 de Quentin RODRIGUES

Retour accueil stage Quentin RODRIGUES

Compte-rendu de stage hebdomadaire

Okantis - Le digital autrement

A propos de mon stage :

Adresses :

```
Siège social GIP SILPC, 2 rue Jean Monnet, 87170 Isle *
Second site sur Limoges, 8 rue Soyouz, 87280 Limoges
Pôle République III, 14 rue des Landes, 86000 Poitiers
* Lieu où mon stage se déroule
```

Horaires :

```
Lundi : 9h - 17h
Mardi : 9h - 17h
Mercredi : 9h - 17h
Jeudi : 9h - 17h
Vendredi : 9h - 17h
```

Maitre de stage :

```
Identité : Laurent BREDA
Adresse mail : laurent.breda@silpc.fr
Téléphone : 05 55 43 99 00
```

Toutes les semaines du 03/01/2022 au 18/02/2022 :

Semaine 1	Semaine 1 (03-01-2022 au 07-01-2022)
Semaine 2	Semaine 2 (10-01-2022 au 14-01-2022)
Semaine 3	Semaine 3 (17-01-2022 au 21-01-2022)
Semaine 4	Semaine 4 (24-01-2022 au 28-01-2022)
Semaine 5	Semaine 5 (31-01-2022 au 04-02-2022)
Semaine 6	Semaine 6 (07-02-2022 au 11-02-2022)
Semaine 7	Semaine 7 (14-02-2022 au 18-02-2022)

Semaine 1 (03/01/2022 - 07/01/2022) :

Lundi :

Mon stage de deuxième année de BTS SIO commence le 03 Janvier 2022. J'ai rejoint l'équipe de développeurs que je connaissais déjà, car mon premier stage était dans la même entreprise.

On me fournit un ordinateur et on me rappelle mes identifiants de connexion pour le réseau interne de l'entreprise.

Je visite ensuite le bâtiment car ce n'est pas le même que mon premier stage. L'équipe de développeur a changé de site, mais reste dans la ville de Limoges.

Je participe à une **première réunion** de l'équipe des développeurs, qu'ils appellent **“Sprint”**, le but est de voir les tâches restantes, d'estimer le temps nécessaires, et de répartir les tâches, le tout pour un même projet.

Après la pause du midi, je participe à une réunion pour découvrir le projet qui va m'être confié.

Je vais travailler sur **un système de réservation de véhicule**. La réservation de véhicule se fera sur la même application WEB qu'a déjà créé Valentin MAZABRAUD (Etudiant en BTS SIO, et alternant). L'application WEB permet la création d'ordre de mission, et j'ajouterai la réservation de véhicule dans cette même application.

Un ordre de mission est un document permettant un déplacement hors de l'entreprise, les véhicules sont ceux de l'entreprise.

Pour cette application de réservation de véhicule, je vais utiliser un système d'API, pour avoir d'un côté le **BACK**, et d'un autre le **FRONT**. Je vais me servir également de component, que je vais devoir créer.

Pour l'**API**, je vais utiliser **Express JS**, et pour le **front**, je vais utiliser **vue.js**.

Afin de comprendre le projet, Valentin LECOMPTE m'a demandé de regarder un des projets de l'entreprise déjà faits. Pour cela j'ai regardé certains composants de leur bibliothèque qu'ils ont créé.

J'ai utilisé **GitLab** pour découvrir leurs projets.

Ci dessous un morceau d'un contrôleur back que j'ai pu découvrir :

```
import Etablissement, { IEtablissement } from '../models/Etablissement'
import Paginate, { ResponsePaginated } from '../../../utils/paginate'
import InvalidArgumentError from
'../../../utils/errors/InvalidArgumentError'
import { UpdateWriteOpResult } from 'mongoose'

export default class EtablissementController {
    /**
     * GET all Etablissements
     * @param  {Paginate} pagination
```

```
     * @returns Promise
     */
    public async getEtablissements (pagination: Paginate)
:Promise<ResponsePaginated<IEtablissement>> {
        const rows = await Etablissement.find({ isActive: true })
            .limit(pagination.itemsPerPage)
            .skip(pagination.itemsPerPage * pagination.page)
            .sort(pagination.sortParser())
        const count = await Etablissement.countDocuments({ isActive: true })
        return new ResponsePaginated({ rows, count }, pagination)
    }

    /**
     * Retrieve one Etablissement
     * @param  {string} id
     * @returns Promise
     */
    public async getEtablissement (id: string):Promise<IEtablissement> {
        return Etablissement.retrieveById(id)
            .catch(() => {
                throw new InvalidArgumentError('Etablissement not found')
            })
    }
```

Et voici un exemple de front en vue.js :

```
<template>
          <v-btn
            color="primary"
            class="white--text"
            @click="onClickAjout"
            small
               >
            ajouter un etablissement
          </v-btn>
</template>

<script>
export default {
  name: 'EtaCreationButton',
  props: {
  },
  data () {
    return {
    }
  },
  methods: {
    onClickAjout () {
      this.$emit('click')
    }
```

```
  }

}
</script>
<style scoped>

</style>
```

Mardi :

Le mardi 04 Janvier 2022 est mon deuxième jour de stage. Je vais donc **approfondir** mes recherches sur les API avec **ExpressJS** et le front avec **vue.js**.

Je vais suivre un tutoriel **FullStack** disponible sur la plateforme YouTube qui montre comment créer une **API Express**, ainsi que la partie **Vue.js**, le tout lié à une base **MongoDB**.

Je réalise dans un premier temps la partie **BACK**.

Je commence par créer un projet et y ajouter les **librairies** nécessaires, pour se faire j'utilise les commandes suivante :

```
npm init
npm i express cors body-parser mongodb
npm i -D nodemon
```

Express est le framework pour créer mon API, cors permet un bon fonctionnement de l'application, body-parser me permet de mettre des données en brut pour mes requêtes POSTS et MongoDB est mon système de base de données.

Ainsi j'obtiens un fichier **package.json** qui ressemble à ceci :

```
{
  "name": "tuto-express",
  "version": "1.0.0",
  "description": "VueJS / ExpressJS",
  "main": "index.js",
  "scripts": {
    "start": "node server/index.js",
    "dev": "nodemon server/index.js"
  },
  "author": "Quentin RODRIGUES",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.19.1",
    "cors": "^2.8.5",
    "express": "^4.17.2",
    "mongodb": "^4.2.2"
  },
  "devDependencies": {
    "nodemon": "^2.0.15"
```

```
  }
}
```

Une fois mon projet initialisé, je vais créer mon fichier index de base :

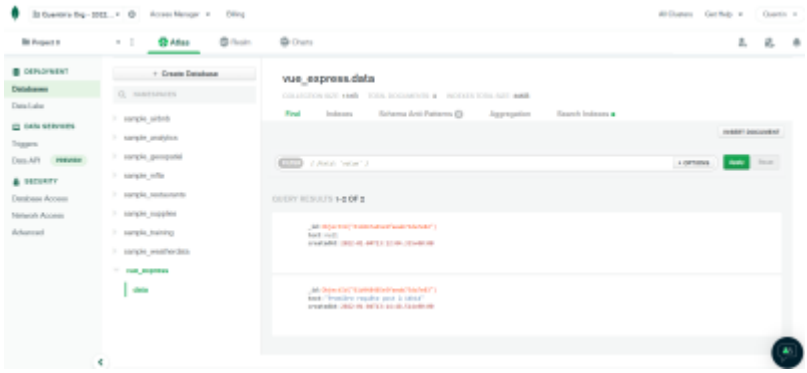
```
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');

const app = express();

// Middleware
app.use(bodyParser.json());
app.use(cors());

const posts = require('./routes/api/posts');

app.use('/api/posts', posts);

const port = process.env.PORT || 3000;

app.listen(port, ()=>{console.log(`Server started on port ${port}`)});
```

Celui-ci va permettre le lancement de l'application sur le **port** choisi, et également lier les routes disponibles dans le dossier comprenant le fichier des routes. Il va aussi importer les **librairies** nécessaires.

Avant de continuer le développement de l'API, je vais avoir besoin d'une base de donnée MongoDB.

MongoDB est un système de gestion de base de données orienté documents, répartissable sur un nombre quelconque d'ordinateurs et ne nécessitant pas de schéma prédéfini des données. Il est écrit en C++. Wikipédia.

Pour se faire je vais utiliser Atlas, un service de MongoDB pour créer gratuitement des petites bases de données et pouvoir obtenir un lien pour se connecter à la base enregistrer dans le cloud. J'ai dû créer un compte, et après la simple création d'une base de données, j'obtiens ceci :



Ma base de données est celle nommée **“vue_express”**, et elle contient déjà quelques **enregistrement** car j'ai avancé le travail avant la prise de cette capture d'écran. Les autres bases de données sont disponible par Atlas pour faire des tests.

La base de données à une **collection** que j'ai nommé “data”.

Afin d'utiliser la base de données dans mon projet, je vais utiliser ce morceau de code :

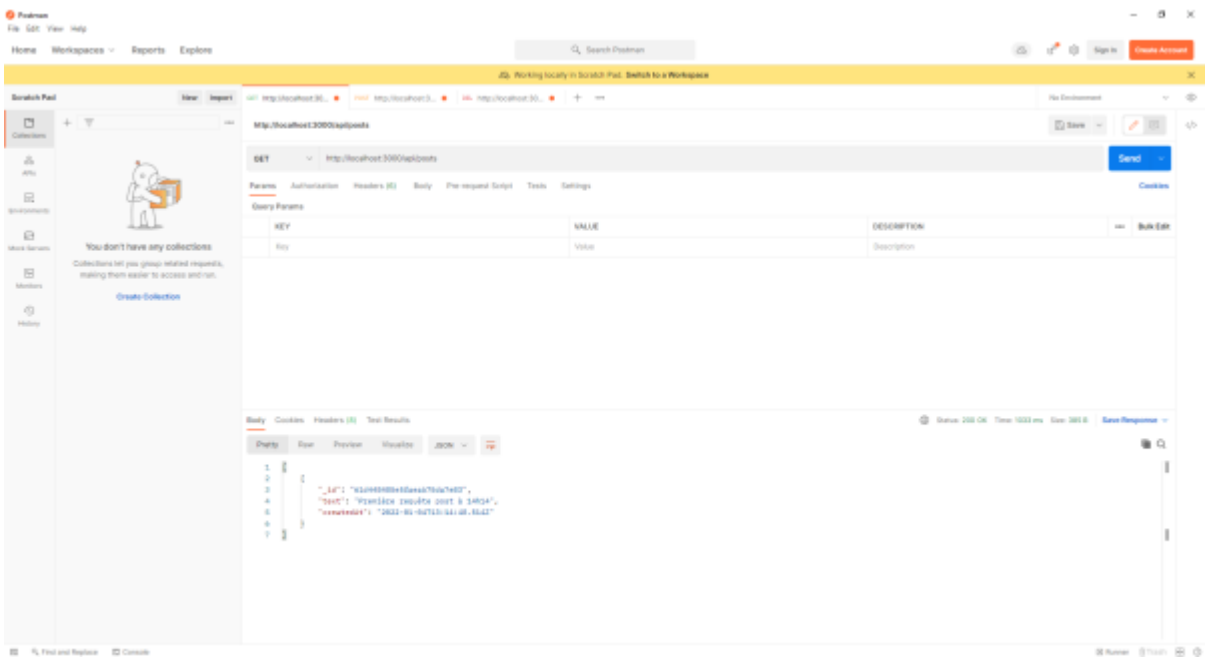
```
async function loadPostsCollection(){
    const client = await
mongodb.MongoClient.connect('mongodb+srv://user1:user1@cluster0.dylhx.mongodb.net/vue_express?retryWrites=true&w=majority', {
        useNewUrlParser: true
    });

    return client.db('vue_express').collection('data');
}
```

Maintenant que notre base de données est prête, je vais pouvoir continuer le développement. Au niveau des routes, je vais créer trois routes différentes. Je vais faire **GET**, **POST** et **DELETE**.

```
// Get
router.get('/', async (req, res)=>{
    const posts = await loadPostsCollection();
    res.send(await posts.find({}).toArray());
})

// Post
router.post('/', async (req, res) =>{
    const posts = await loadPostsCollection();
    await posts.insertOne({
        text: req.body.text,
        createdAt: new Date()
    })
    res.status(201).send();
})

// Delete
router.delete('/:id', async (req, res)=>{
    const posts = await loadPostsCollection();
    await posts.deleteOne({_id: new mongodb.ObjectId(req.params.id)});
    res.status(200).send();
})
```

Les deux routes qui ne sont pas en GET renvoie un statue pour être sûr que la requête ait bien fonctionné.

Enfin, je vais pouvoir tester mes différentes routes avec l'outil Postman.



Comme nous pouvons le voir, la requête affiche un enregistrement, avec son texte et sa date. Et il retourne un statut 200. Tout est fonctionnel.

Mercredi :

Le mercredi de cette première semaine est mon premier jour en **télétravail**. (Je suis en télétravail 3 jours par semaine dû aux nouvelles règles sanitaires contre le/la Covid-19.)

Durant la matinée, je vais terminer le **tutoriel** sur ExpressJS et Vue.js.

Voici **l'application** que j'ai réalisé :

L'application ressemble à un système de commentaire ou de poste.

Un commentaire correspond à un texte et à une date. La date est celle au moment de la création du commentaire.

Lorsqu'on arrive sur la page, vue.js envoie une requête **GET** à l'API Express pour obtenir les différents commentaires présente sur la base de données.

En cas de double click sur un commentaire, vue.js va faire une requête **DELETE** pour supprimer le commentaire de la base mongoDB, et de nouveau faire un GET pour actualiser les commentaires.

Il y a également une section pour ajouter un commentaire en **POST**.

Pour **lier** la partie **FRONT et BACK**, j'ai utilisé **axios**. Axios est une librairie qui va faire les requêtes à mon API, de cette manière :

```
import axios from 'axios';

const url = 'http://localhost:3000/api/posts';

class PostService{
    //Get
    static getPosts() {
        return new Promise ((resolve,reject) => {
            axios.get(url).then((res) => {
                const data = res.data;
                resolve(
                    data.map(post => ({
                        ...post,
                        createdAt: new Date(post.createdAt)
                    }))
                );
            })
            .catch((err)=> {
                reject(err);
            })

        });
    }

    //Post
    static insertPost(text){
        return axios.post(url, {text});
    }

    //Delete
    static deletePost(id){
        return axios.delete(`${url}/${id}`);
    }
}

export default PostService;
```

Jeudi et Vendredi :

J'ai choisi de **réunir** le jeudi et vendredi de ma semaine car j'ai fait le même travail. Je suis en **télétravail** durant ces deux jours.

J'ai commencé à travailler sur mon projet de stage. Je rappelle le projet ci-dessous :

Je vais travailler sur **un système de réservation de véhicule**. La réservation de véhicule se fera sur la même application WEB qu'a déjà créé Valentin MAZABRAUD (Etudiant en BTS SIO, et alternant). L'application WEB permet la création d'ordre de mission, et j'ajouterai la réservation de véhicule dans cette même application.

Un ordre de mission est un document permettant un déplacement hors de l'entreprise, les véhicules sont ceux de l'entreprise.

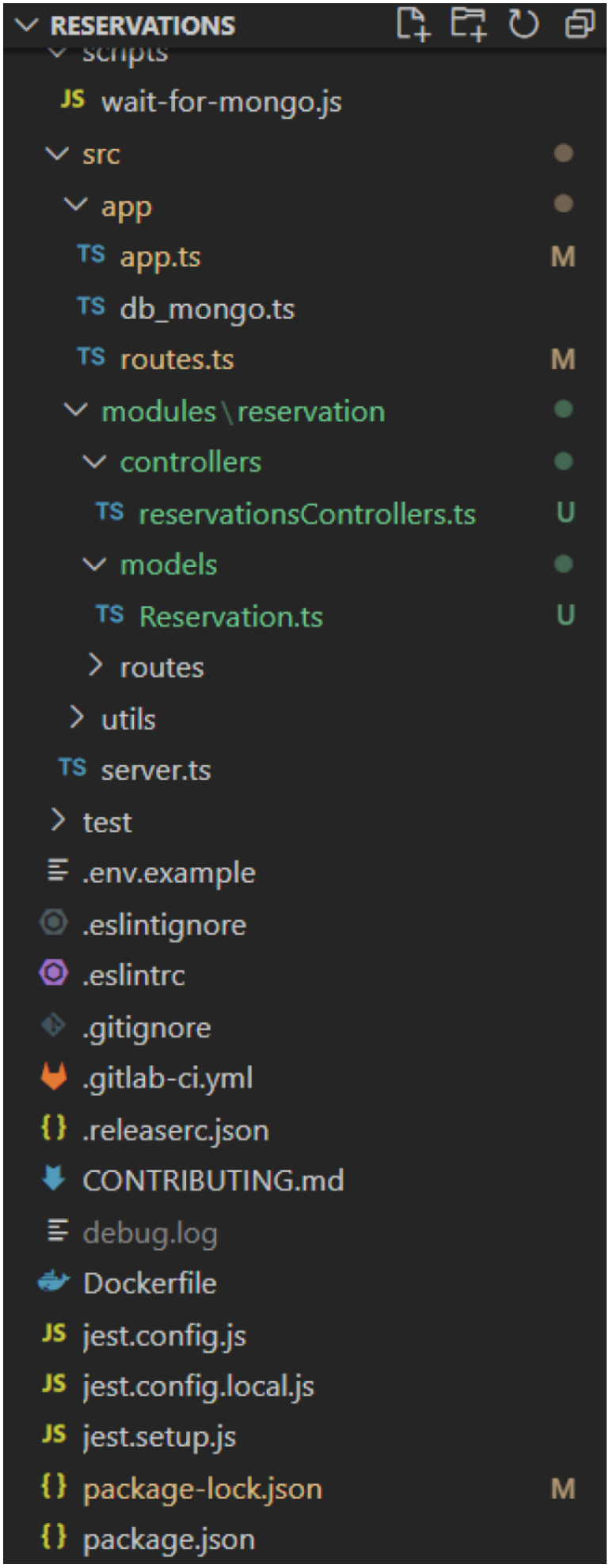
Pour cette application de réservation de véhicule, je vais utiliser un système d'API, pour avoir d'un côté le **BACK**, et d'un autre le **FRONT**. Je vais me servir également de component, que je vais devoir créer.

Pour l'**API**, je vais utiliser **Express JS**, et pour le **front**, je vais utiliser **vue.js**.

Pour se faire j'ai commencé par la parti **BACK** en faisant commençant une API Express lié à une base mongodb.

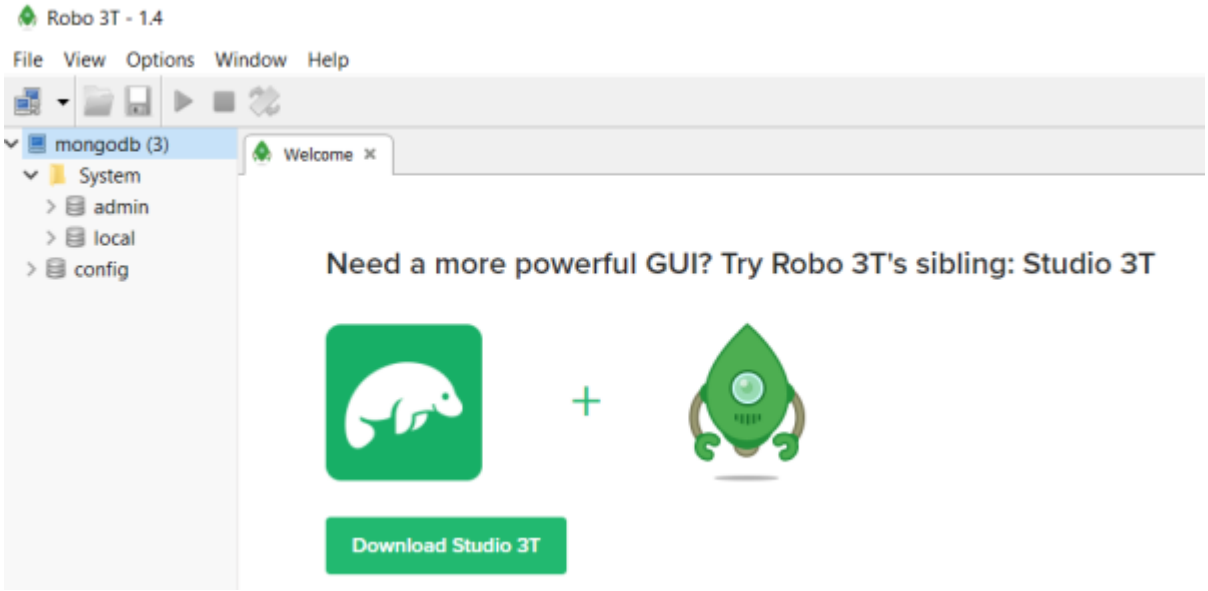
Pour l'instant, j'ai uniquement les grands axes du projet, mais aucun détail, je n'ai aucun idée du modèle de données que je vais devoir créer, donc je vais faire l'API de manière à ce quelle soit fonctionnelle.

Pour commencer, j'ai clone le template de base d'une API **Express** que Valentin LECOMPTE à générer sur **GitLab**. Et j'ai bien regardé la structure du projet qui est presque identique à celle du tutoriel que j'ai suivi. La structure ressemble à ceci :


RESERVATIONS
scripts
wait-for-mongo.js
src
app
app.ts M
db_mongo.ts
routes.ts M
modules \ reservation
controllers
reservationsControllers.ts U
models
Reservation.ts U
routes
utils
server.ts
test
.env.example
.eslintignore
.eslintrc
.gitignore
.gitlab-ci.yml
.releaserc.json
CONTRIBUTING.md
debug.log
Dockerfile
jest.config.js
jest.config.local.js
jest.setup.js
package-lock.json M
package.json

Afin de créer une **API fonctionnelle**, il m'a fallut une base de données. J'ai utilisé mongodb. Pour cela j'ai fait un **container docker** avec l'image **mongo**.

On m'a également conseillé l'outil **Robo 3T** pour visualiser la base de données.



J'ai eu certains **blocages** à ce moment pour continuer lié à ma non-connaissance de **l'asynchrone** du javascript et notamment les **interfaces**, et surtout les **Promises**. Donc Valentin LECOMPTE m'a fait un cours d'environ une demi-heure pour me présenter son fonctionnement.

Je vais pouvoir continuer dès la semaine prochaine le travail sur l'API.

Conclusion de fin de semaine :

Cette première semaine de stage chez Okantis était riche en apprentissage. J'ai appris de nouvelle chose, notamment Express JS, la création d'une API, un peu de Vue.js. J'ai également compris le langage asynchrone, et la manière dont Javascript utilise les Promises.

J'ai hâte d'avoir un premier rendu sur le projet qui m'a été confié.

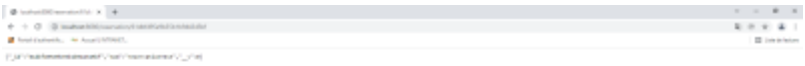
Puis, j'ai fait une partie de télétravail ce qui me permet d'être plus autonome et de ne pas hésiter à contacter quelqu'un en cas de soucis.

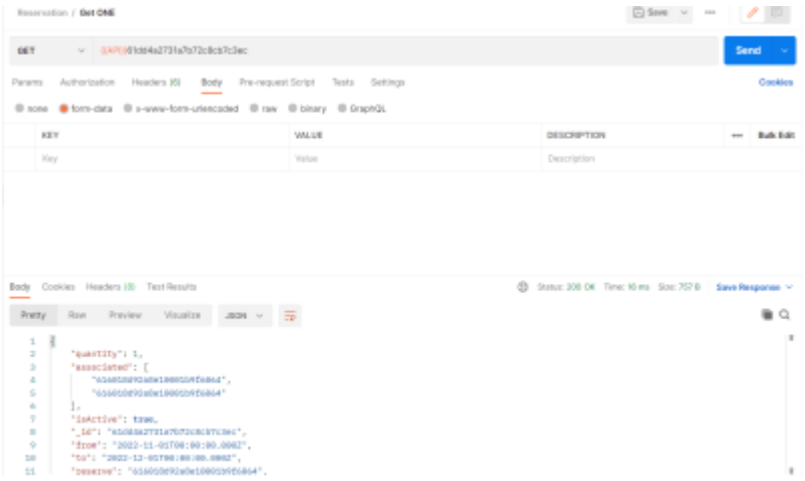
Semaine 2 (10/01/2022 - 14/01/2022) :

Lundi :

Le lundi de cette deuxième semaine de stage était en présentiel.

J'ai finalisé toutes mes routes de mon API, puis j'ai vérifié dans le navigateur ainsi que avec Postman quelle fonctionnaient bien toutes :





Ensuite, j'ai regardé comment mes données sont importés dans mongo. Avec le logiciel Robo 3T présenté la semaine dernière.

Cette journée était simple, mais m'a permis de bien me préparer pour le lendemain avec le vrai modèle de données.

Mardi :

Le mardi 11 janvier 2022, j'ai reçu le modèle de données pour compléter mon API.

Le modèle est séparé en deux modèles de données. Le premier est la réservation en général, et le second est le matériel réservable. Voici un de mes deux modèles :

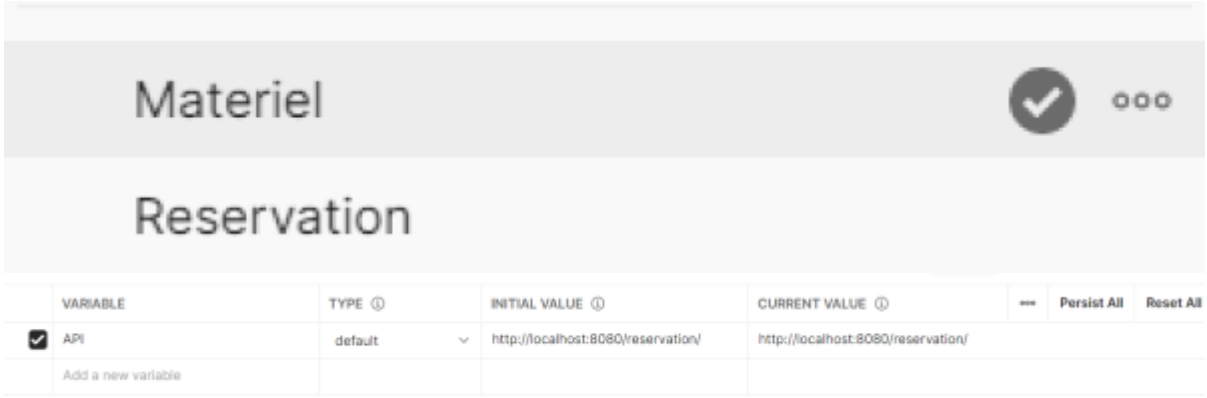
```
import { Document, Model, model, Schema } from 'mongoose'
import InvalidArgumentError from
'../../../utils/errors/InvalidArgumentError'

const ReservationSchema: Schema = new Schema({
    // Dates de la réservation
    from: {
        type: Date,
        required: true
    },
    to: {
        type: Date,
        required: true
    },

    // Élément réservé; plusieurs ref possible (cf:
https://mongoosejs.com/docs/populate.html#dynamic-ref)
    reserve: {
        type: Schema.Types.ObjectId,
        required: true,
        refPath: 'reserveType'
    },
    quantity: {
        type: Number,
        default: 1
    },
    reserveType: {
        type: String,
        required: true,
        enum: ['Materiel']
    },

    // Personnes qui louent
    for: {
        type: Schema.Types.ObjectId,
        ref: 'User',
        required: true // Personne qui réserve
    },
    associated: [{
        type: Schema.Types.ObjectId,
        ref: 'User' // Plusieurs personnes dans une même voiture par exemple
    }],
```

```
    // Logs
    isActive: {
        type: Boolean,
        default: true
    },
    createAt: {
        type: Date,
        required: true
    },
    createdBy: {
        type: Schema.Types.ObjectId,
        ref: 'User',
        required: true
    },
    updates: [{
        at: {
            type: Date
        },
        by: {
            type: Schema.Types.ObjectId,
            ref: 'User'
        }
    }]
})

/**
 * Static method
 */
ReservationSchema.statics.retrieveById = function (id: string):
Promise<IReservation> {
    return this.findById(id).exec()
}

/**
 * Interface
 */

// -- Reflection about current schema
export interface IReservation extends Document {
    from: Date
    to: Date
    reserve: String
    quantity: Number
    reserveType: String
    for: String
    associated: String[]
    isActive: Boolean
    createAt: Date
    createdBy: String
    updates:{
        at: Date
```

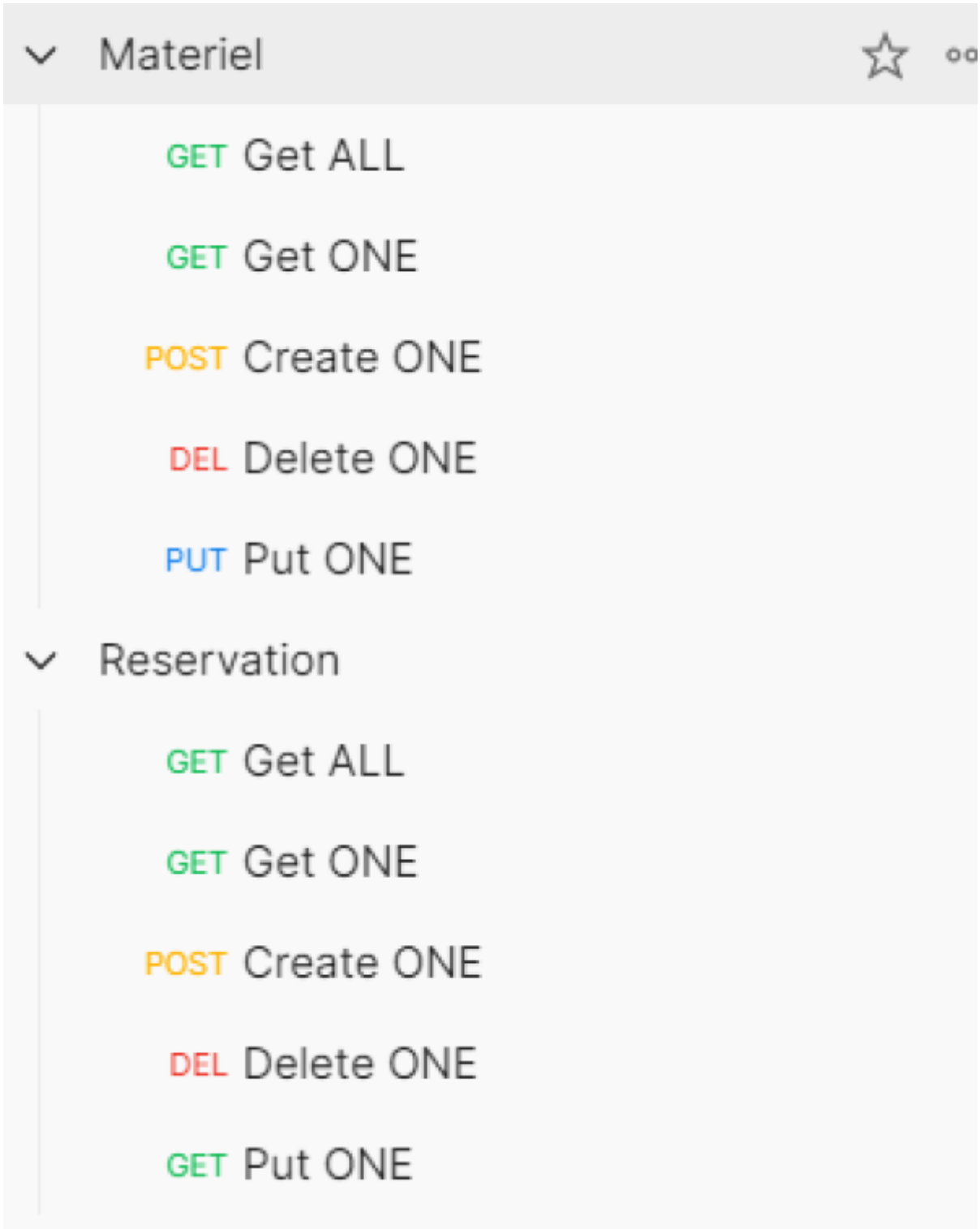
```
        by: String
    }
}

// -- Don't allow some field to be updated
ReservationSchema.statics.filterUpdate = function (data: IReservation): any
{
    delete data._id
    delete data.id
    return data
}

// -- Reflection to the current schema to add some method
export interface IReservationModel extends Model<IReservation> {
    retrieveById(id: string): Promise<IReservation>;
    filterUpdate(data: any): any;
}

export default model<IReservation, IReservationModel>('Reservation',
ReservationSchema)
```

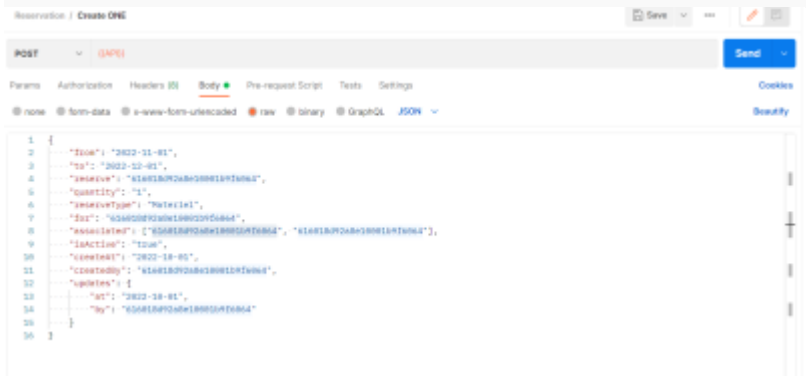
Puis j'ai un code similaire pour le matériel. Ainsi, j'ai configuré Postman pour pouvoir travaillé plus facilement.

Dans un premier temps j'ai créé deux environnements, correspondant aux deux modèles de données :



Et j'ai fait deux collections, avec tous les requêtes nécessaires, qui correspondent toujours aux différents modèles :





Après avoir fait mes tests complet, je vois que mon API est entièrement fonctionnelle.

Je peux maintenant commencer à créer quelques composants FRONT. En clonant un template de l'entreprise.

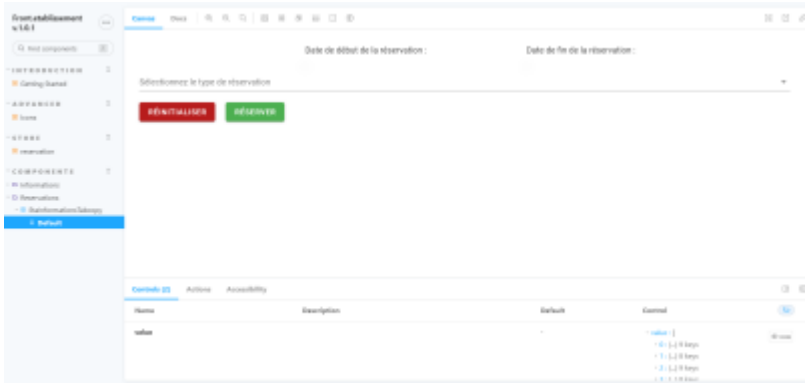
Pour la partie front je vais utiliser StoryBook.

Mercredi, Jeudi et Vendredi :

J'ai choisi de lier ces trois jours de télétravail pour un même compte-rendu quotidien.

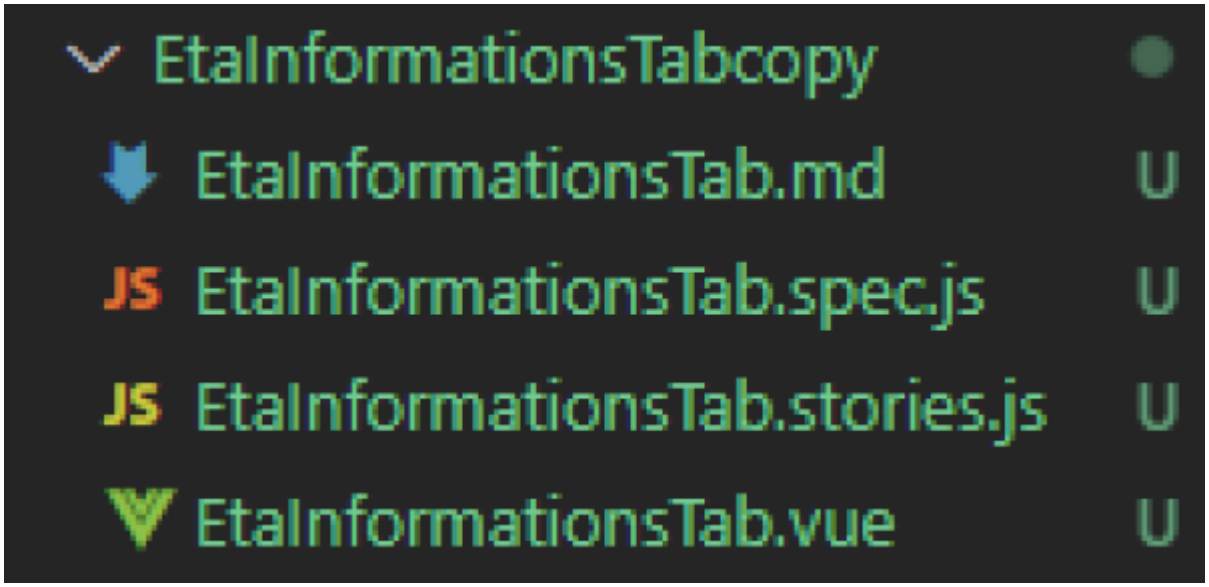
Cette fin de semaine est dirigé sur la partie Front de mon projet comme précédemment planifié.

Pour commencer j'ai dû découvrir Storybook.js, c'est une librairie qui va me permettre d'afficher séparément mes composants Front. Voici un exemple d'affichage sur storybook :



J'ai cloné un projet précédemment utilisé par l'entreprise pour débuter.

J'ai créé mon premier composants qui va permettre de créer une réservation, avec l'architecture ci-dessous :



Le fichier terminant par .stories.js est le fichier de configuration pour storybook, les autres fichiers représente le composant.

Conclusion de fin de semaine :

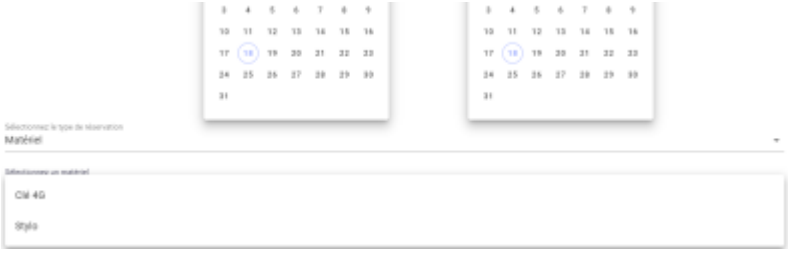
Semaine 3 (17/01/2022 - 21/01/2022) :

Lundi :

Le lundi de cette troisième semaine, j'ai continué la création de mon composant pour faire une réservation.

Ainsi, j'ai ajouté un sélecteur, pour choisir le matériel voulu pour la réservation. Le sélecteur de matériel s'affiche uniquement si le type de réservation choisi est un matériel (Et non un véhicule).

Mon sélecteur est comme ceci, à l'aide de vuetify :



Et la condition est celle-ci :

```
<v-select
      v-model="selectMateriel"
      :items="materiels"
      :rules="[(v) => !!v || 'Veuillez choisir un matériel.']"
      label="Sélectionnez un matériel"
      required
      @change="setSelectedMateriel"
      v-if="this.select == 'Matériel'"
    ></v-select>
```

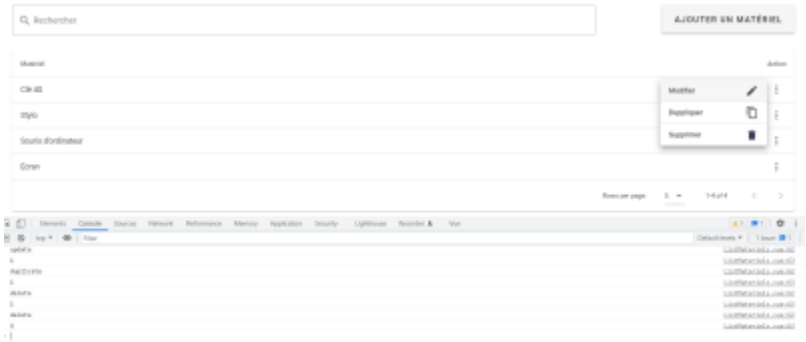
J'ai également travaillé sur un référentiel de véhicule et de matériel. Ainsi j'ai deux nouvelles API pour gérer les véhicules et matériels.

Mardi :

Le mardi 18 janvier, j'ai créé un nouveau component. Afin d'afficher tous mes matériels dans un tableau, ainsi que d'avoir un bouton d'action pour modifier, supprimer et ajouter un matériel.

Voici une capture en cours de création du composant :

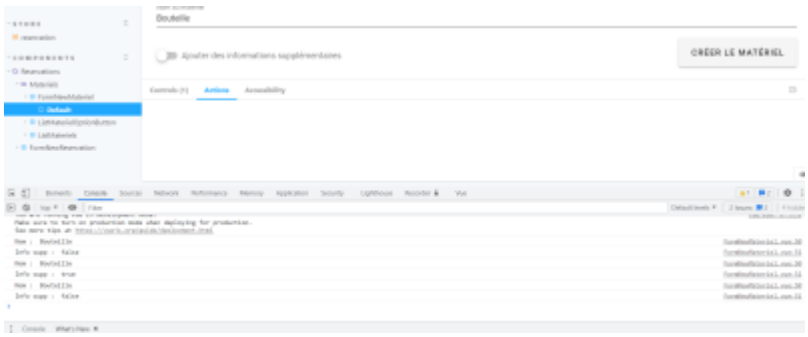
En début d'après-midi, j'ai terminé mon composant en théorie. Il est capable de récupérer l'ID d'un matériel, et l'action choisi par l'utilisateur. L'affichage ressemble à cela :



La barre de recherche est fonctionnelle à l'aide de la fonction search des data list de vuetify :

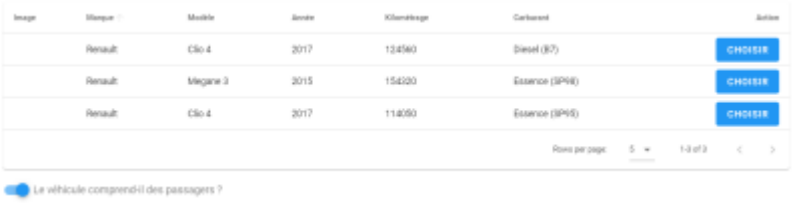
```
<v-data-table
  :headers="headers"
  :items="materiels"
  :items-per-page="5"
  :search="search"
  class="elevation-1"
>
```

Pour terminer cette journée, j'ai créé un formulaire de création de matériel. Et j'ai utilisé la console pour vérifier que les valeurs sont bien passés (en attendant de faire les vraies requêtes API) :



Mercredi, Jeudi et Vendredi :

Durant cette fin de troisième semaine, j'ai avancé mes components. Par exemple j'ai continué le formulaire de création d'une réservation, et pour les véhicules, j'ajoute un tableau pour choisir un véhicule d'une liste (Qui sera récupéré par l'API plus tard.



Pour importer cette datalist, j'ai fait un component à part, et je l'ajoute dans mon formulaire de cette façon :

Dans la partie script :

```
import ReservationVehicule from
'../ReservationVehicule/ReservationVehicule.vue'
```

Dans la partie template :

```
<ReservationVehicule v-if="this.select == 'Véhicule'"></ReservationVehicule>
```

Conclusion de fin de semaine :

Cette troisième semaine était très bien car j'ai enfin commencé à voir un premier visuel de ce que je vais devoir produire, et selon moi, obtenir un premier résultat visuel d'un travail est la partie la plus importante car nous pouvons voir l'avancé d'un projet.

Semaine 4 (24/01/2022 - 28/01/2022) :

Lundi :

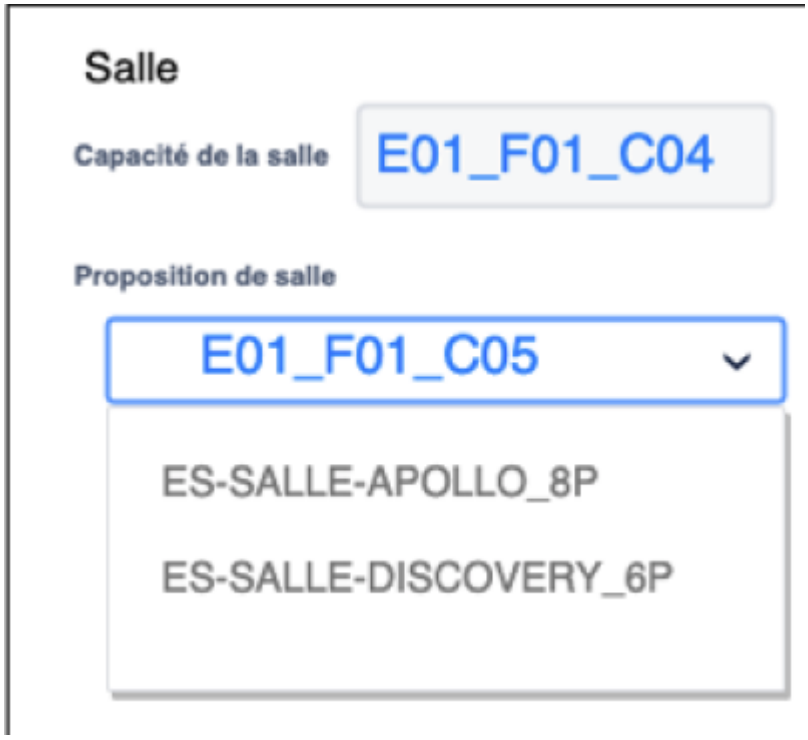
Le lundi de cette quatrième semaine de stage, j'ai reçu une maquette de l'application :

Ainsi, j'ai eu un premier aperçu du rendu souhaité.

Valentin LECOMPTE m'a donc préparé des issues sur GitLab, afin que je sois guidé, voici la liste de chose à faire pour le moment :

J'ai complété le modèle de données Salle, et j'ai également fait un composant pour le formulaire pour réserver une salle.

Objectif :

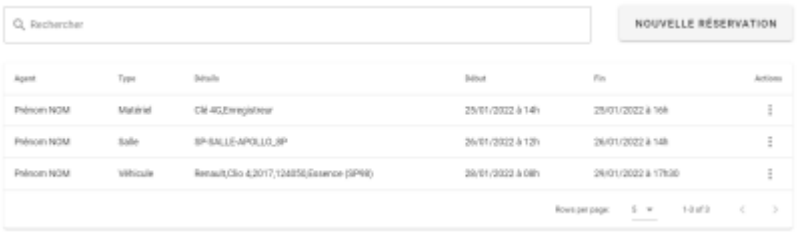


Rendu :

Mardi :

Durant la matinée de ce mardi, j'ai créé le formulaire de réservation, en me basant sur les composants que j'ai déjà créé. Voici une capture du résultat obtenu :

J'ai ensuite travaillé sur une liste pour afficher toutes les réservations actuelles :



Le code pour cette liste sous vue.js ressemble à ceci :

```
<v-col>
    <v-row>
      <v-col md="9">
        <v-text-field
          v-model="search"
          outlined
          label="Rechercher"
          prepend-inner-icon="mdi-magnify"
          clearable
        ></v-text-field>
      </v-col>

      <v-col align="right">
        <v-btn elevation="2" x-large class="right"
@click="clickNewReservation"> Nouvelle réservation </v-btn>
      </v-col>
    </v-row>

    <v-data-table
      :headers="headers"
      :items="reservations"
      :items-per-page="5"
      :search="search"
      class="elevation-1"
    >
      <template v-slot:[`item.action`]="{ item }">
        <list-reservations-option-button
          @action="OnAction($event, item._id)"
        ></list-reservations-option-button>
      </template>
    </v-data-table>
  </v-col>
```

Il y a également les méthodes pour émit des actions :

```
methods: {
    OnAction (event, id) {
      this.$emit(event, id)
    },
    clickNewReservation(){
      this.$emit('newReservation')
    }
  }
```

Et on les écoutes comme ceci dans le fichier stories de storybook :

```
const actions = {
  onAction: action('action'),
  onUpdate: action('update'),
```

```
  onDelete: action('delete'),
  onNewReservation: action('newReservation')
}

const Template = genereTemplate({
  components: { ListReservations },
  methods: actions,
  template: '<v-row class="justify-center"><list-reservations v-
bind="$props" @action="onAction" @update="onUpdate" @delete="onDelete"
@newReservation="onNewReservation" /></v-row>'
})
```

Mercredi, Jeudi et Vendredi :

Cette fin de semaine en télétravail m'a permis de finaliser les détails de mon travail depuis le début du stage. Voici une liste de chose que j'ai amélioré :

- API Back pour réservation et salle
- Component Front pour la création d'une réservation (Gérer les emits)
- Gestion du repos GitLab avec des merges requests.

J'ai participé à une réunion, afin de voir le travail de Valentin Mazabraud, il a créé une application pour la gestion des ordres de mission (Déplacement), auquel j'intégré les différents modules de réservation que j'ai créé.

J'ai également présenté mon travail dans cette réunion.

Conclusion de fin de semaine :

Cette quatrième semaine était vraiment intéressante car j'ai pu retravaillé sur mes travaux déjà réalisé dans le but de les rendre plus fonctionnels. Ainsi, je suis prêt à intégrer mon travail dans l'application prévu à cet effet la semaine prochaine.

Semaine 5 (31/01/2022 - 04/02/2022) :

Lundi :

Je vais débuter l'intégration de mon travail dans l'application. Cette journée est donc longue car je dois faire l'installation des dépendances avec la commande

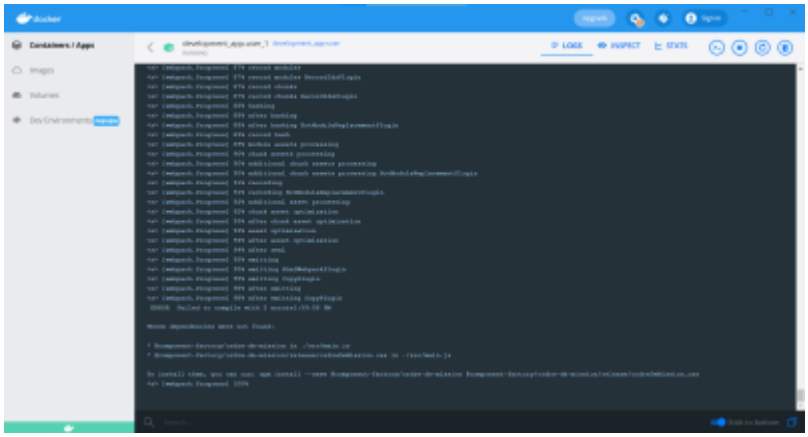
```
npm i
```

cette commande peut être parfois assez longue.

Voici un exemple d'intégration de mon Front et Back dans la partie **app.user**, je complète un fichier json :

```
{
            "isFront": false,
            "name": "reservations",
            "packageName": "@component-factory/reservations",
            "version": "0.0.1",
            "gitUrl":
"https://gitlab.silpc.net/architecture-et-design/component-factory/back/rese
rvations.git"
        },
        {
            "isFront": true,
            "name": "reservations",
            "packageName": "@component-factory/reservations",
            "version": "0.0.1",
            "gitUrl":
"https://gitlab.silpc.net/architecture-et-design/component-factory/front/res
ervations.git"
        }
```

Ensuite, je vais utiliser grunt pour build le projet, et créé des containers Docker.



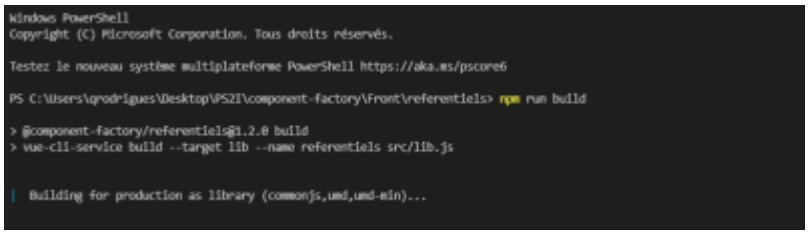
Je n'ai pas eu le temps de terminer l'intégration de l'app sur mon pc, je vais pouvoir continuer le lendemain avant d'y intégrer mon travail sur les réservations.

Mardi :

Aujourd'hui j'ai continué la mise en place de l'application sur mon pc. J'ai donc build les différents projets à l'aide de la commande

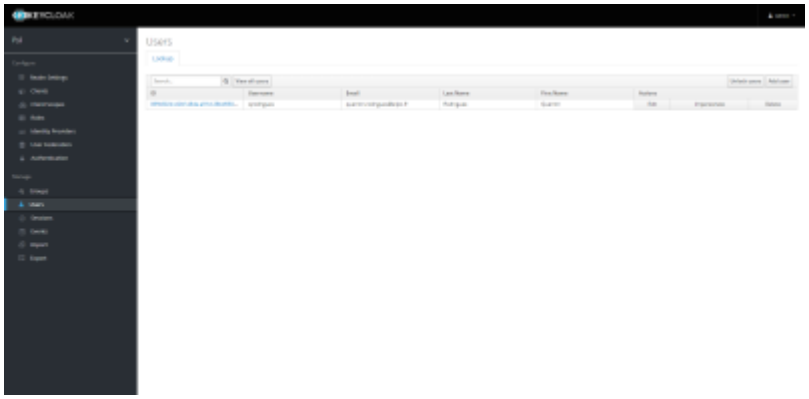
```
npm run build
```

Voici une capture de l'utilisation de cette commande :

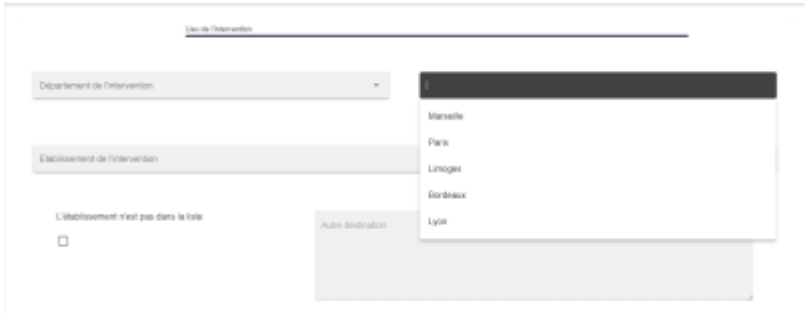


Ensuite j'ai réussi à obtenir le premier aperçu de l'app, avec l'affichage de Keycloak.

Keycloak permet de faire un système d'authentification, c'est un outil tier.



Enfin, j'ai complété mes différents référentiels pour obtenir des listes déroulantes remplies sur l'application. Ci-dessous nous pouvons voir le référentiel de villes, et un aperçu de l'application gérant les ordres de missions (Créée par Valentin Mazabraud).

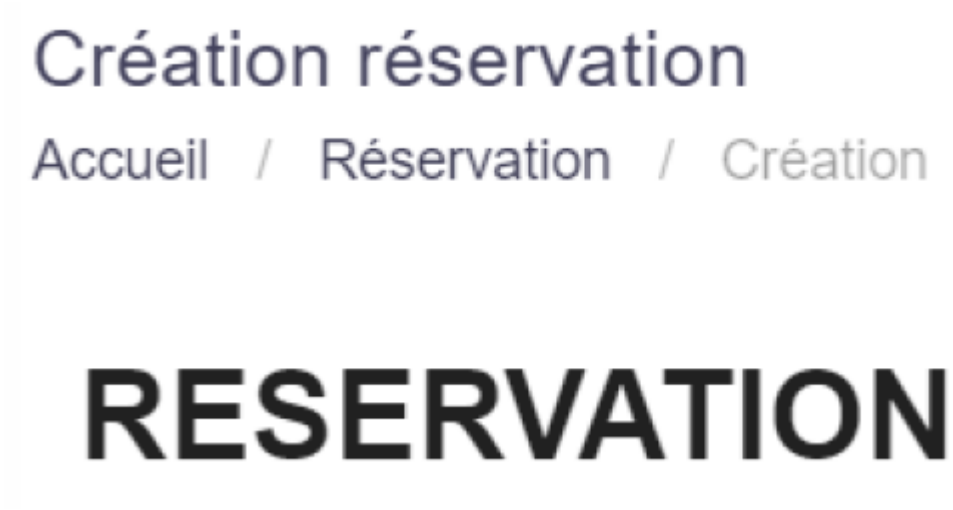


Mercredi :

Le mercredi de cette cinquième semaine, j'ai réussi à cloner entièrement l'application dans laquelle je vais intégrer l'outil de réservation.

Le reste de cette journée, je vais essayer d'y intégrer mes formulaires de réservation. Et j'ai commencé par ajouter l'onglet “Demande de réservation” sur le menu disponible sur la capture d'écran ci-dessus.

J'ai un peu personnalisé l'affichage pour que l'on puisse retrouver le chemin (dans l'application) comme ci-dessous.



Il me reste maintenant à intégrer les différents composants que j'ai créé les semaines précédentes, mais c'est une partie assez difficile sur laquelle je bloque, je vais donc essayé d'avancer dès le lendemain pour essayé d'avoir un résultat plus rapidement.

Jeudi et Vendredi :

Pour cette fin de cinquième semaine, je n'ai pas de capture d'écran intéressante car je suis resté sur l'intégration de ma partie réservation en ayant des difficultés. La semaine suivant me permettra de me concentrer davantage pour réussir cette intégration.

Conclusion de fin de semaine :

Cette cinquième semaine a été très constructive pour moi, en effet j'ai commencé à voir un morceau de la fin du projet car j'ai cloné l'application finale, dans le but d'y ajouter ma partie. Ainsi, je peux obtenir un premier réel rendu et voir que le travail fourni précédemment n'est pas inutile.