Documentation du projet groupe hôtelier : Valery MBELE

A. Spécifications techniques

Server :

- Ubuntu 20.14
- LAMP
- PHP7.4
- Docker
- Postgres
- MySql

Frontend :

- React
- Material ui
- Axios
- React-hook-form
- React-query
- Js Cookie

Backend :

- Symfony
- Api-Platform
- Express
- Sequelize
- Multer
- Jwt
- Nodemon

Utilitaires :

- Figma
- Gitmind
- Trello

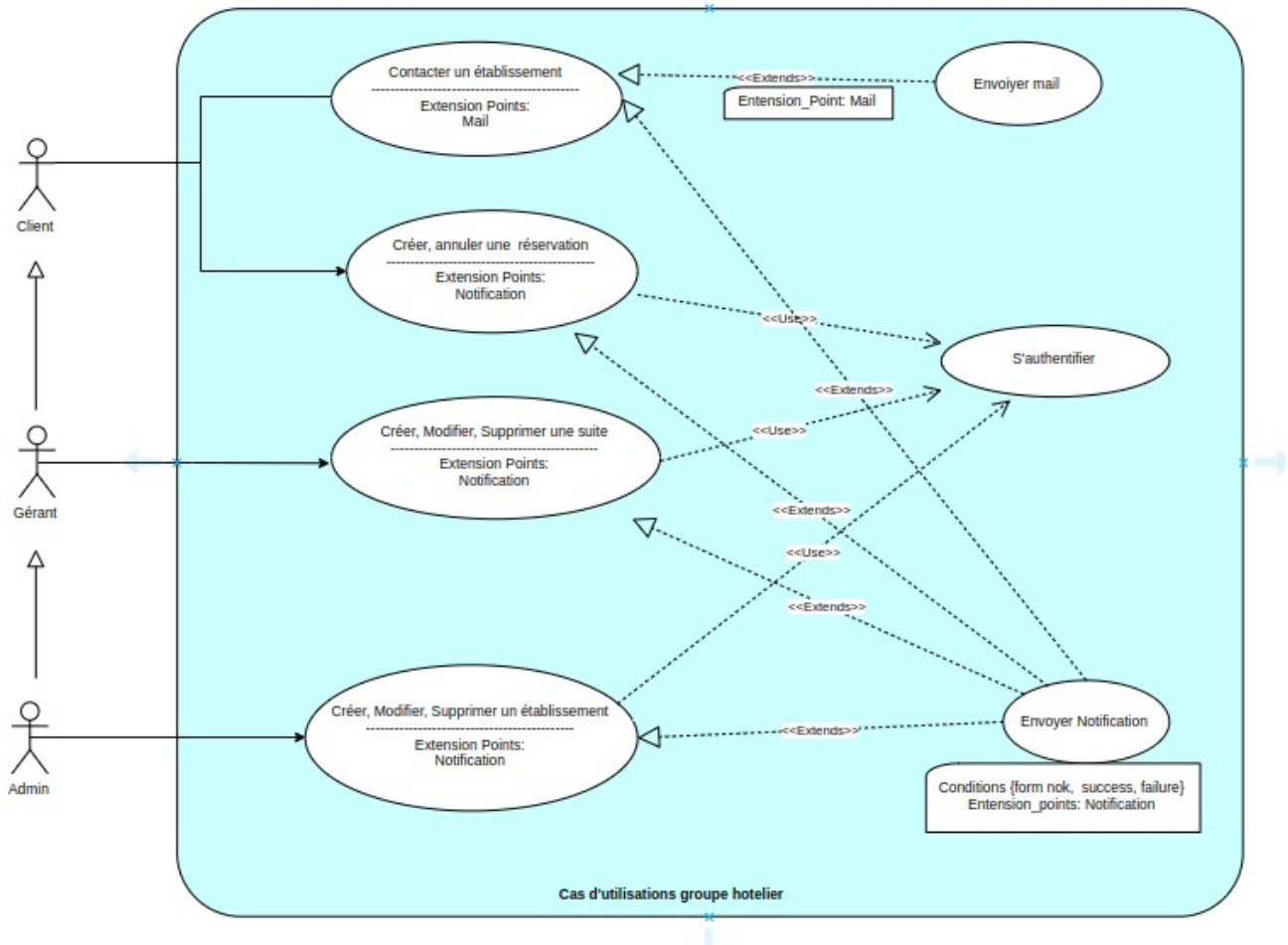
Code :

- VS Code
- Eslint
- Prettier

CI :

- github actions
- github

B. Diagramme de cas d’utilisation

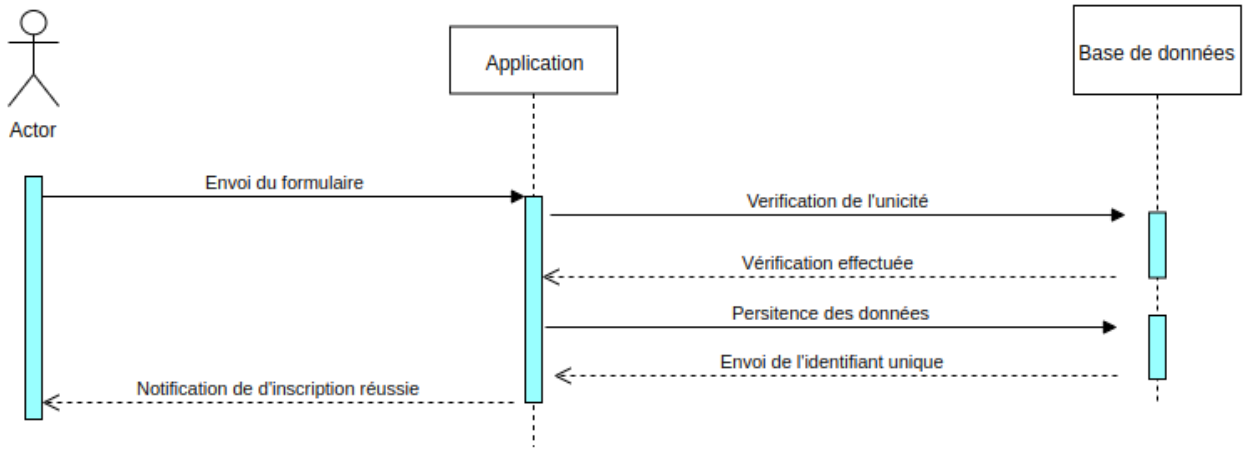


Dans ce diagramme , j’ai compris que l’authentification était primordiale. Mais aussi qu’il fallait avoir un système de notification fiable pour souvent informer l’utilisateur de ce qui se passe .

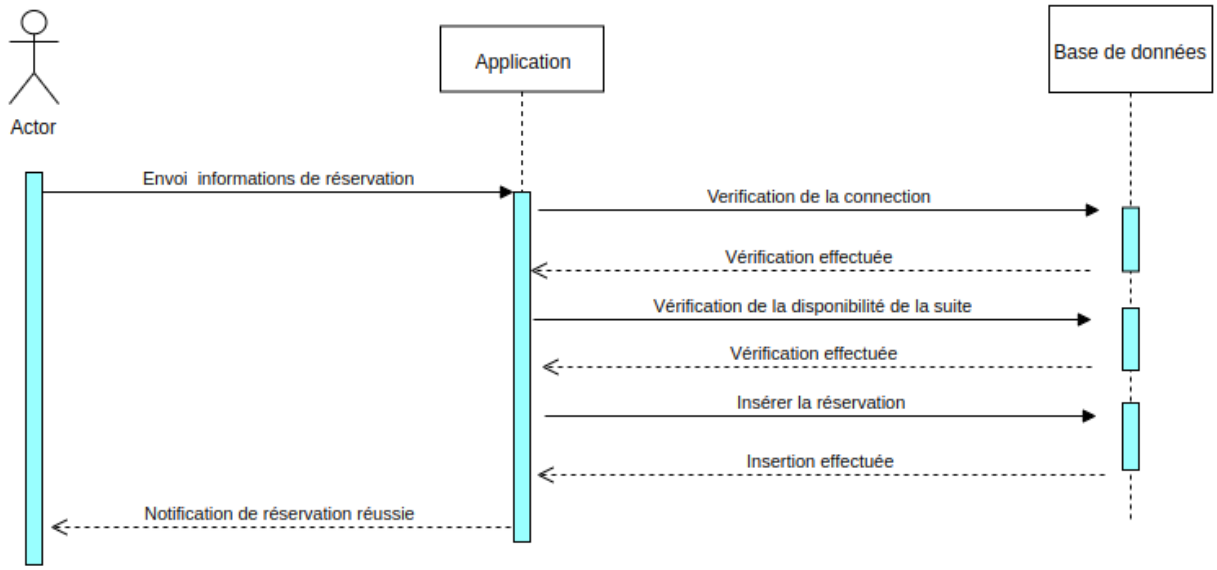
L’administrateur occupe un rôle prépondérant un rôle clé.

C. Diagrammes de séquence .

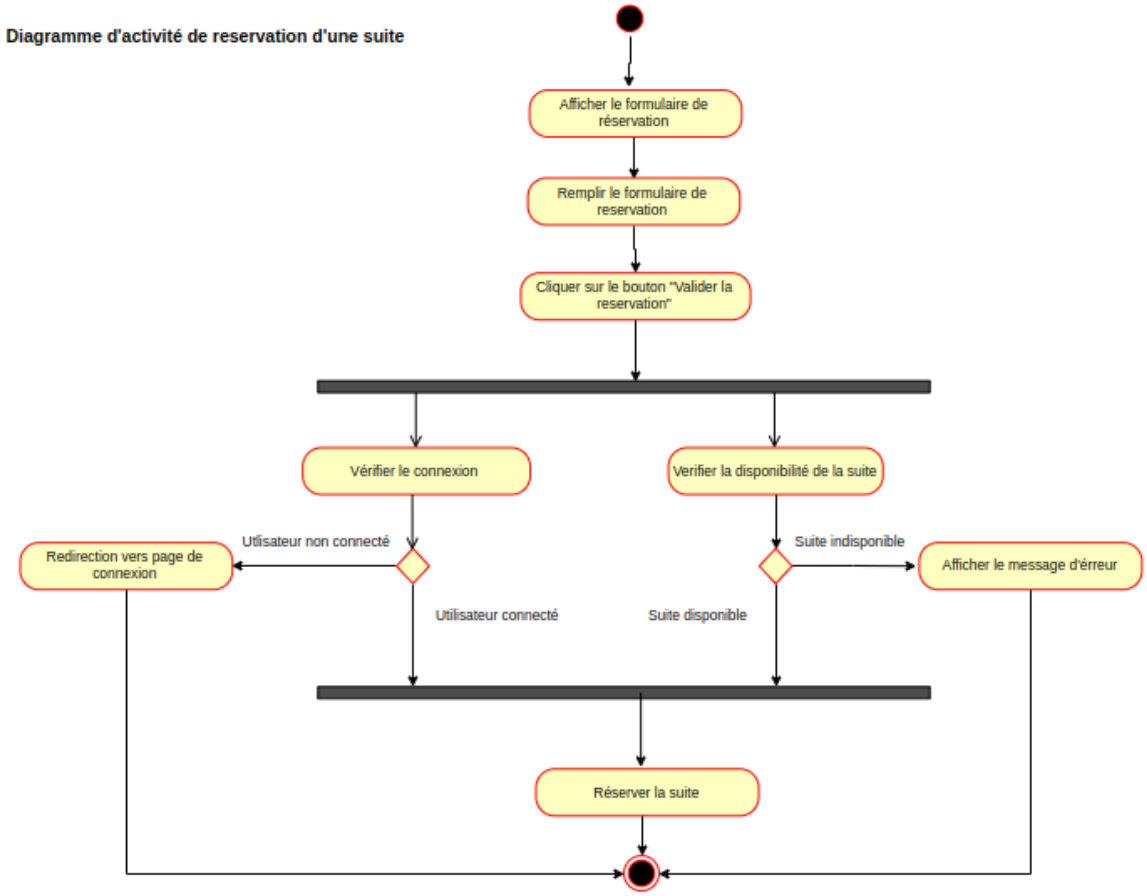
C1. Séquence authentification : cas réussi



C2. Séquence réservation réussie

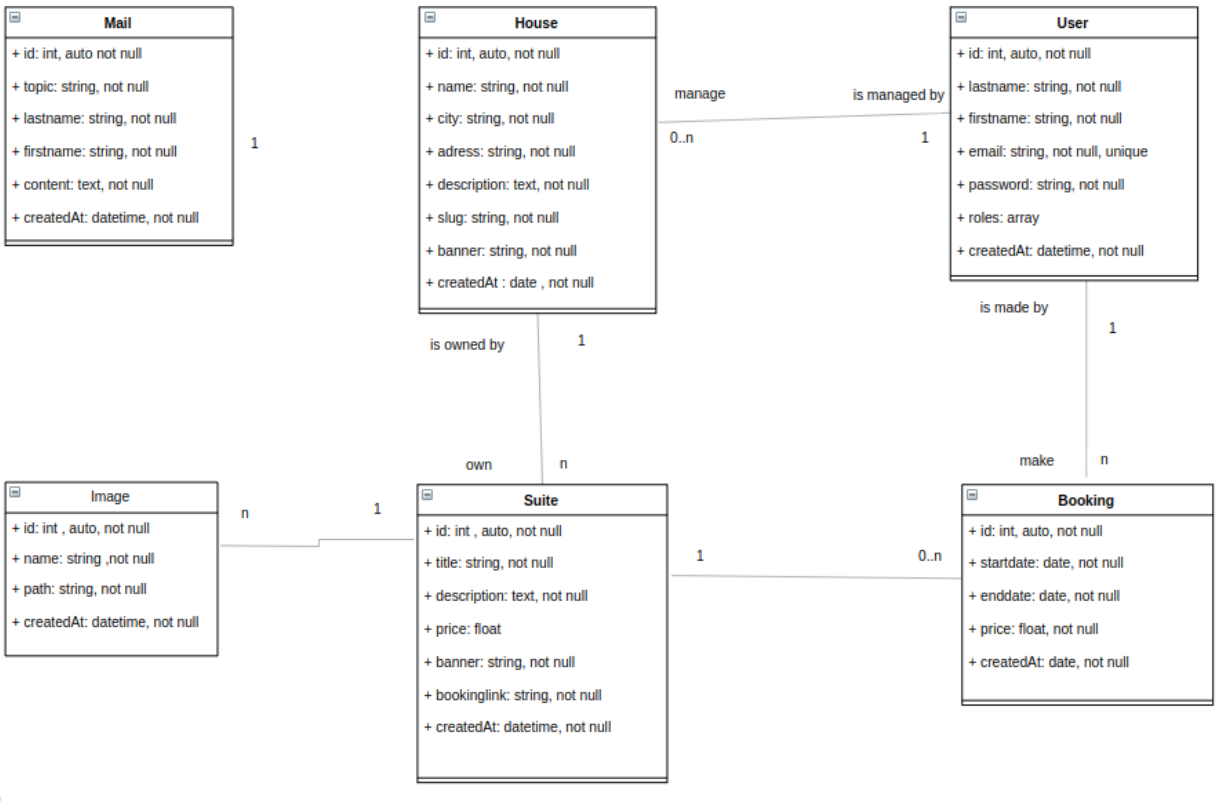


D. Diagramme d’activité de réservation :



Ce schéma est assez clair , et ne laisse pas la place à l’improvisation pendant la phase de développement.

E. Diagramme de classe



Les trois problématiques qui se sont posées sur le modèle de données étaient :

- Les gérants doivent t-ils être une entité à part ? J'ai opté pour le non , car ils ont exactement les mêmes attributs qu'un user, la différence se joue sur le fait qu'ils soient liés ou non à l'établissement

- Faut-il faire une relation n to n entre la Suite et le User ? J'ai opté pour le non . Car dans la pratique , le n to n passe souvent par une table intermédiaire

- Faut-il gérer les images directement comme un attribut des entités qui les utilisent ? J'ai encore dit non , car je ne suis pas sur que ce soient une chose simple que de faire des recherches sql dans un champs qui est un tableau.

E. Reformulation des users stories

US1. Gérer les établissements

US1.1 Gérer les établissements
Titre : En tant qu'administrateur , je veux créer, modifier ou supprimer un établissement pour mettre à jour les informations

La page ne doit pas se recharger

Critères d'acceptation :
scénario 1 : Tentative de modification avec un compte valide
Étant donné que je possède un compte utilisateur (Robert)
Lorsque j'accède page de gestion des établissements
Et que je clique sur créer un établissement ,
Et que je remplis le formulaire
Et que je clique sur le bouton créer un formulaire
Alors j'ai une notification que l'établissement a été crée,
Et je suis redirigé vers la liste des établissement

US1.2 Gérer les gérants
Titre : En tant qu'administrateur , je veux accéder au profil d'un utilisateur pour pouvoir le rendre ou non gérant d'un établissement

US1.3 En tant qu'administrateur, je veux accéder au profil d'un gérant d'établissement afin de modifier ses informations

US2. Gérer les suites

US2.1
Titre : En tant que gérant , je dois accéder à la gestion de l'établissement pour créer , modifier , supprimer une suite
Règles métier :
Le même formulaire sera utilisé pour la création et la modification
La validation de la suppression se fait par une modale et ne recharge pas la page
Le gérant ne peut pas accéder à la page de gestion des autres établissements

US3. Découvrir le catalogue des établissements

US31 Découvrir la liste des établissements
Titre : En tant que Visiteur , je souhaite consulter la liste des établissement ou rechercher un établissement afin de faire mon choix

US32. Découvrir les suites d'un établissements
Titre : En tant que Visiteur , je souhaite consulter la page d'un établissement afin de découvrir ce qu'elle offre : suites , photos , description.
Règle : La page est présentée sous forme de liste

US4. Réserver une suite
Titre : En tant que Visiteur ou Client , je souhaite réserver une suite pour mon séjour
Règle : La réservation se fait sur une page dédiée
On doit pouvoir savoir si la suite est disponible sans rechargement de page

US5. Voir les réservations

US51. Voir la liste des réservations
Titre : En tant que Client , je dois pouvoir accéder à ma liste de réservation afin de m'organiser

US52. Annuler une réservation
Titre : En tant que Client , je souhaite annuler une réservation
Règle : annulation uniquement 3 jours avant au plus tard

US6. Accélérer la réservation d'une suite

US61. En tant que Visiteur ou Client , je dois accéder à la page de réservation depuis la page d'accueil d'un établissement .
Règle : Le remplissage des informations de la suite dans la page de réservation est automatique

US7. Contacter le groupe hôtelier
Titre : En tant que Visiteur ou Client , je veux contacter l'administrateur par mail via un formulaire de contact afin de commander un service non inclus pour le jour J
Règles :
Le nom , prénom , email du client se saisiront automatiquement
Le mail sera envoyé à l'administrateur

F. Afin d'être au plus près du besoin de nos utilisateurs , j'ai défini 4 personas qui pourraient etre assez représentatifs de notre cas .

Elle adore voyager et découvrir des mondes qu'elle n'a pas eu le temps d'explorer quand elle était active. Car sa situation financière le permet.
Elle est très attachée aux valeurs écologiques .
Elle adore aller sur les réseaux sociaux via son téléphone quand elle se repose dans un cadre naturel et paradisiaque.

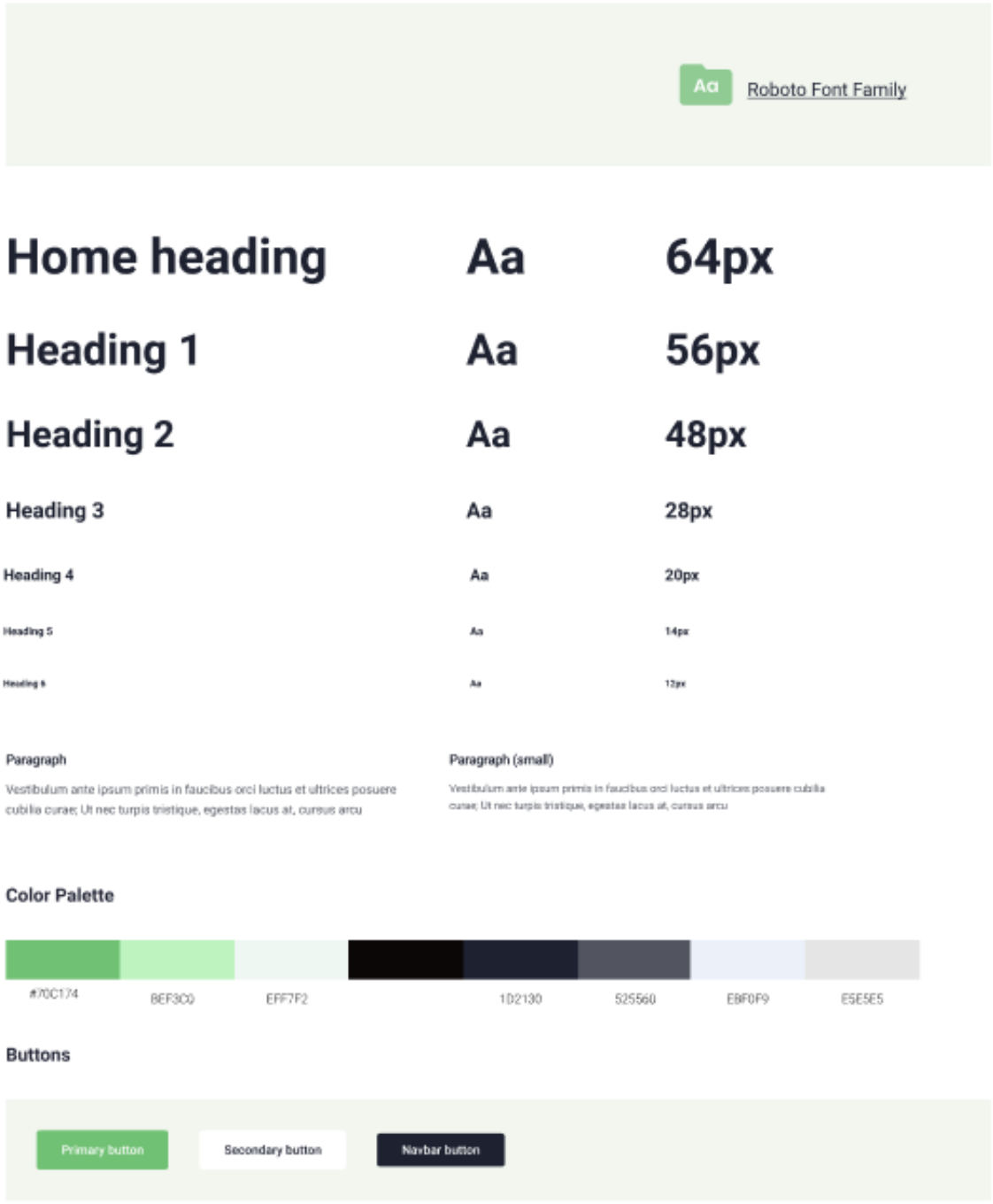
Valery , 50 ans , Visiteur.
C'est un haut cadre de la finance qui adore le luxe . Il a 3 enfants avec lesquel il adore partager des destinations de rève. Il est néanmoins très exigeant , mais se laisse facilement tenter quand ses copains lui parlent de suite luxueuse et écologiques.

Daniel, 43 ans , gérant d'hôtel.
Daniel n'aime pas du tout passer son temps au téléphone. Ce qu'il préfère avant tout , c'est le contact humain et la vie réelle . Il est tout le temps pressé car les clients n'attendent pas.

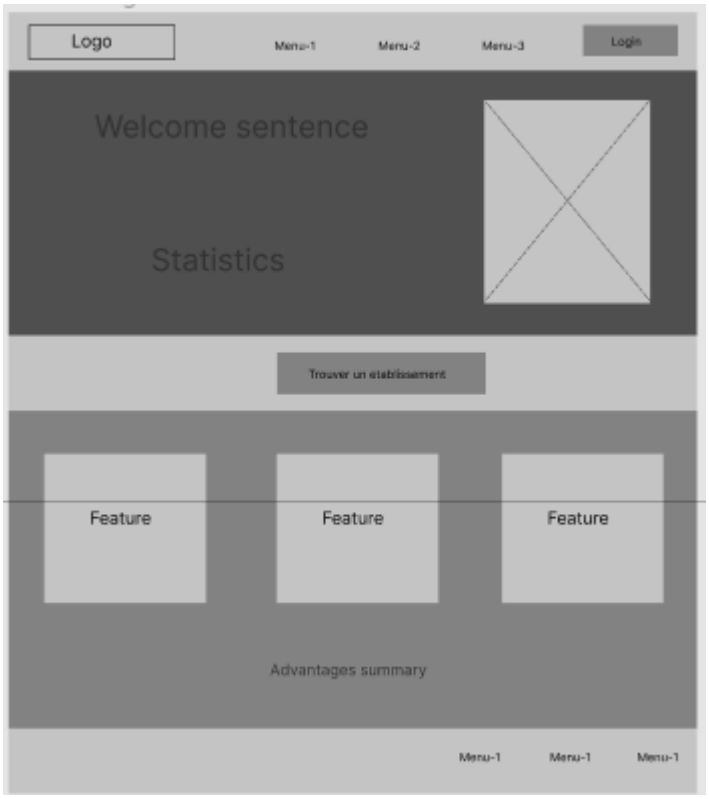
Anne , 36 ans est une mère de famille posée. Elle est déjà administratrice dans son groupe hôtelier , et a attendu avec hâte que la numérisation arrive . Elle a le sens du détail.

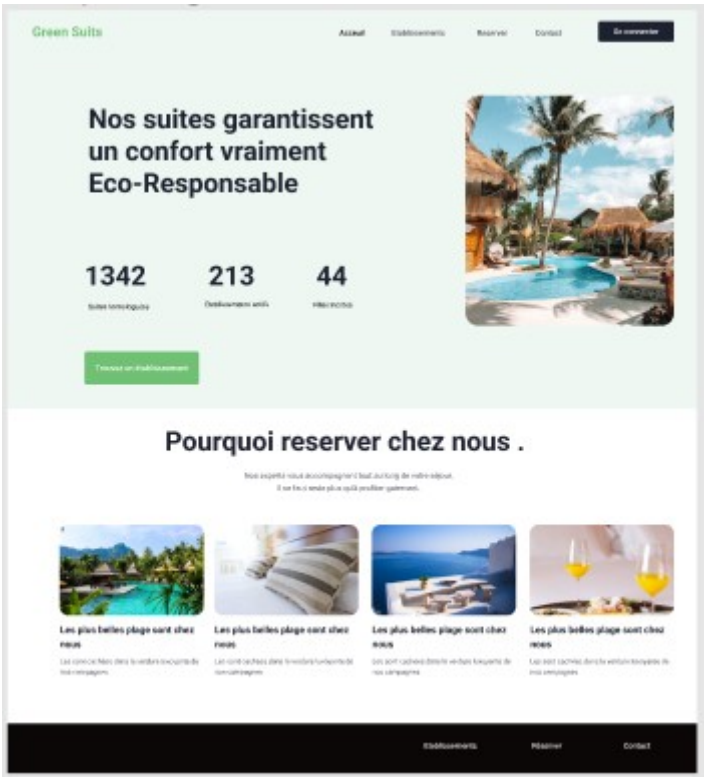
L’application va donc s’approcher d’un thème un peu écologique

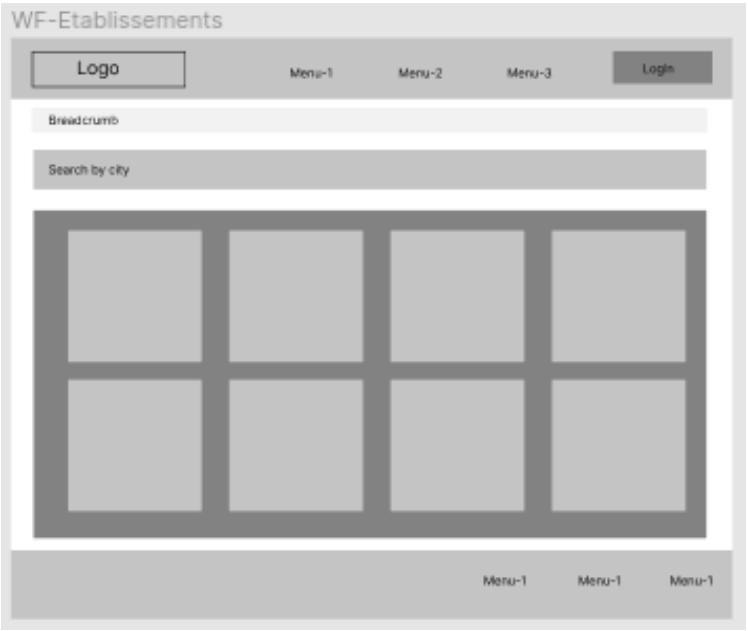
G .Charte graphique du site :

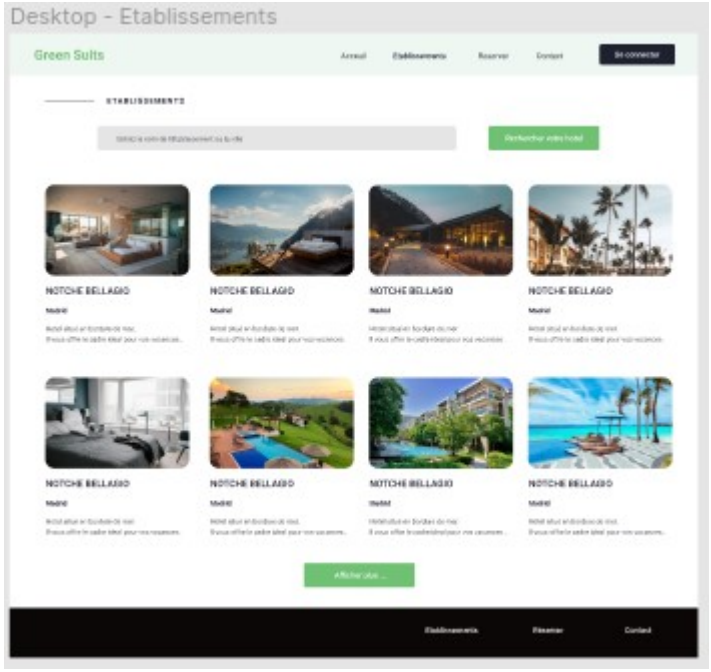


Les couleurs restent assez épurées et légère.

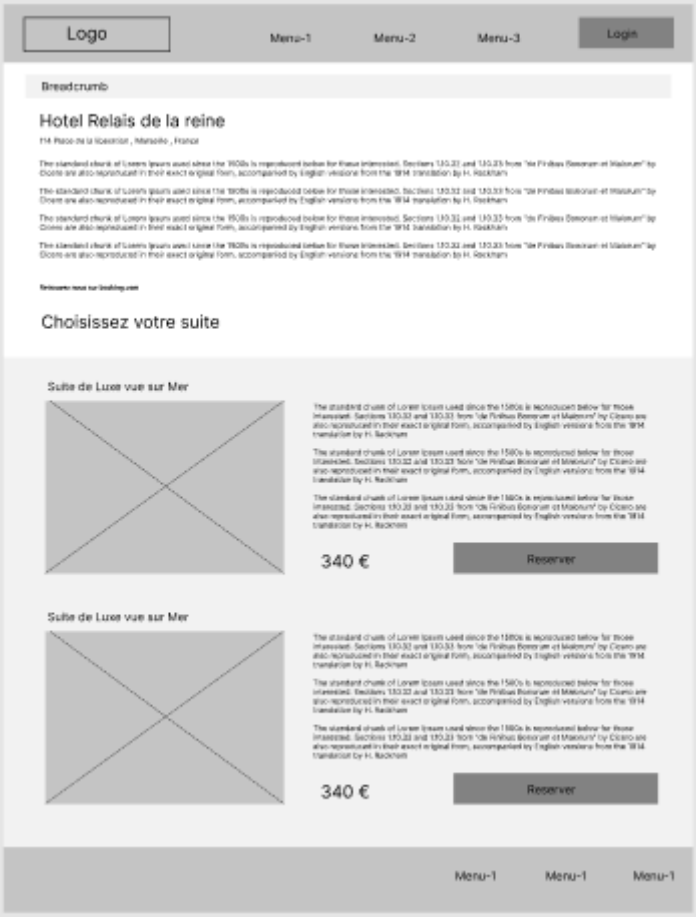




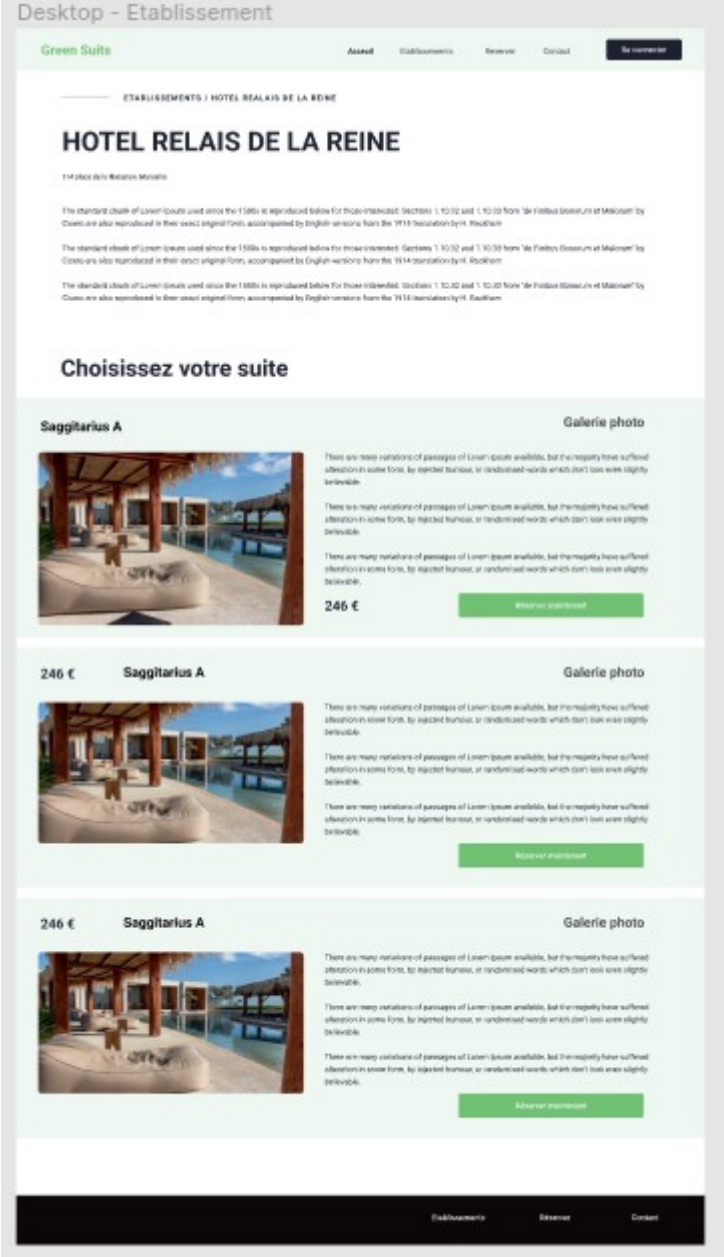




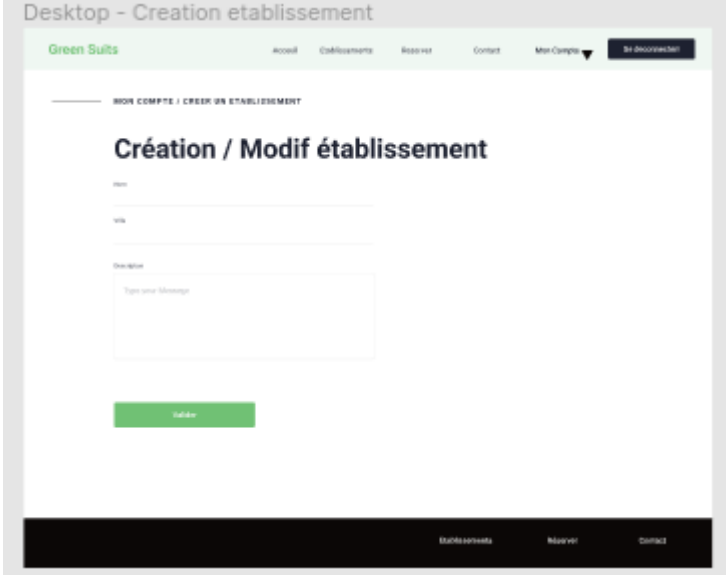
WF-Etablissement



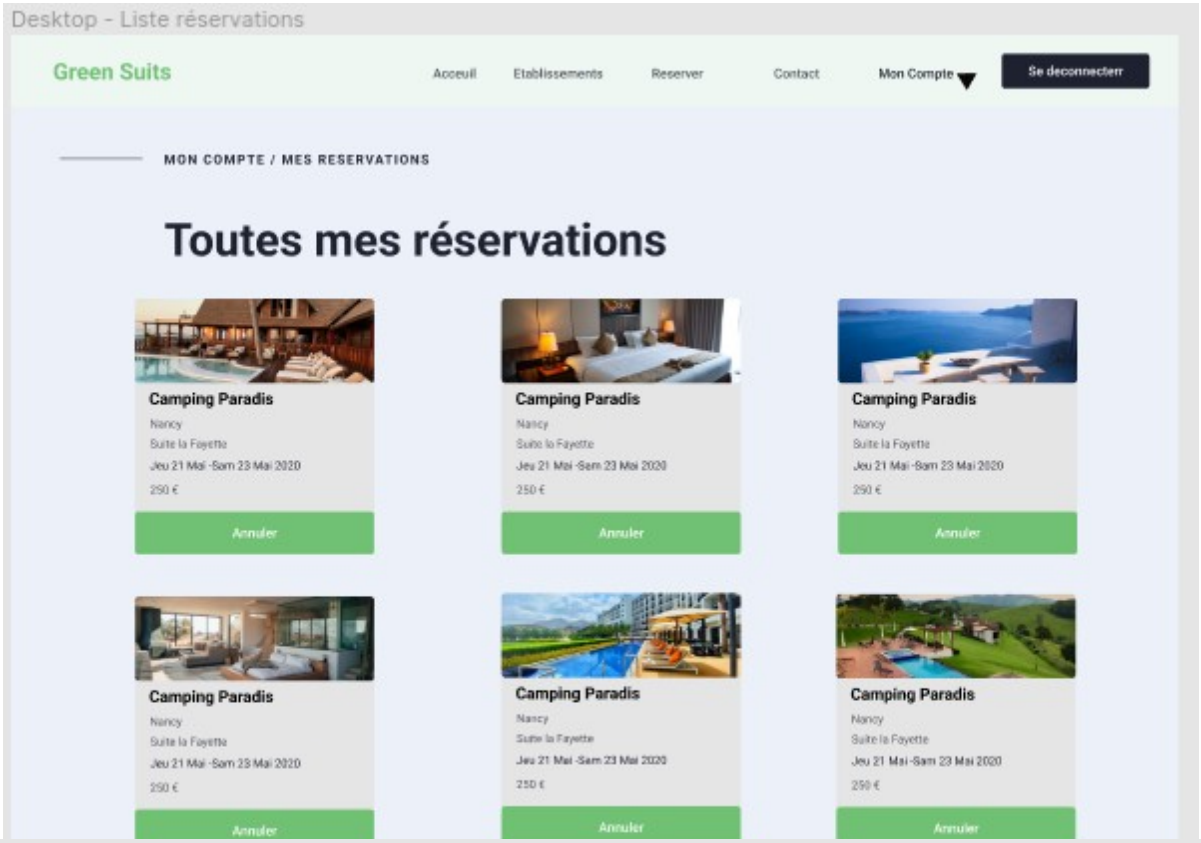
Desktop - Etablissement



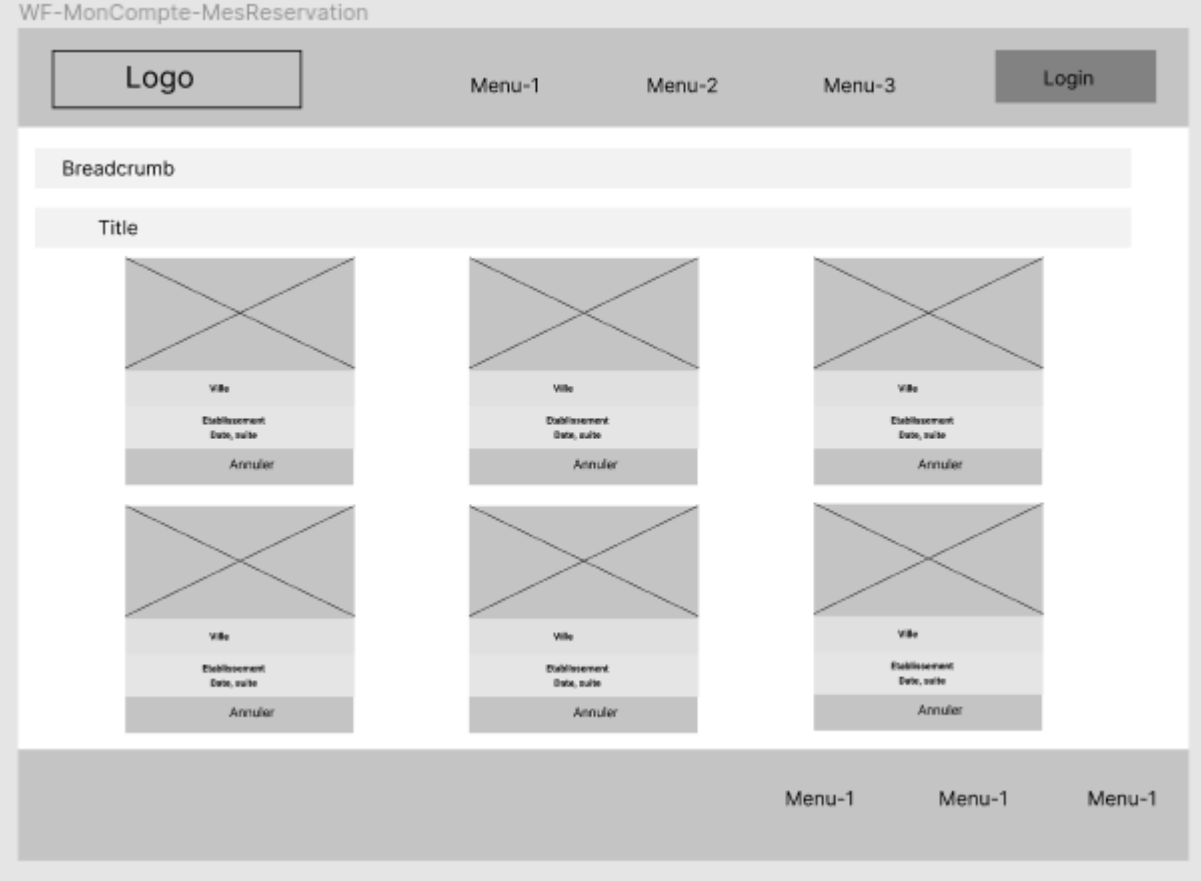
Desktop - Creation etablissement

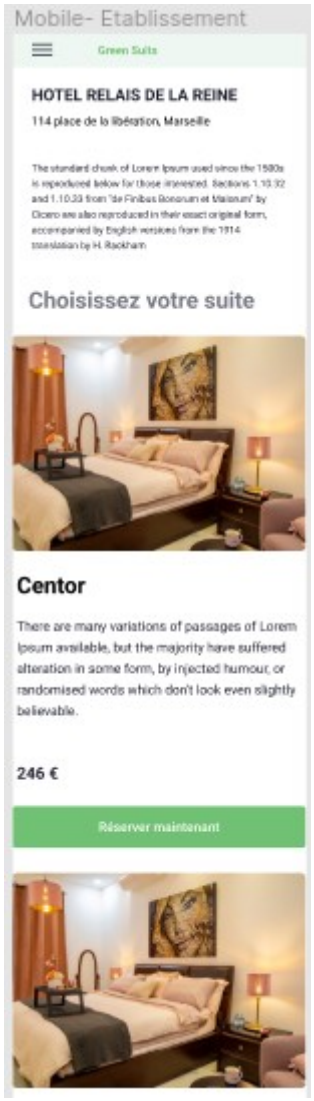


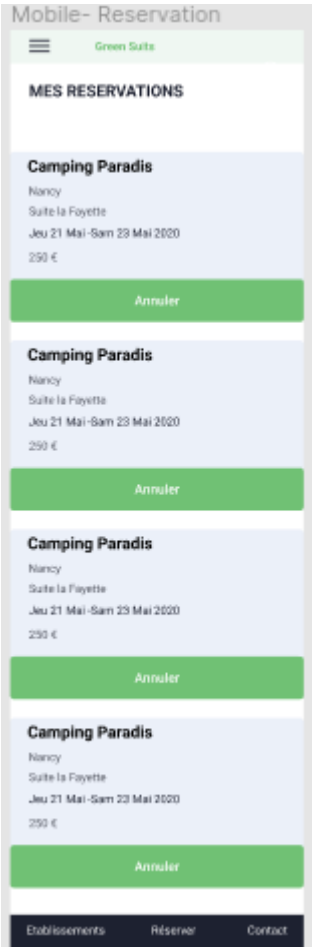
Desktop - Liste réservations



WF-MonCompte-MesReservation







Synthèse :

Pour un projet de ce genre , j'ai trouvé que le couplage Symfony Réact était adapté avec de pour donner du dynamisme à ces couleurs coté front , et de la puissance en backend . C'est la raison pour laquelle j'ai opté pour ApiPlatform/Reactjs.
De plus utiliser docker dans une application symfony facilite grandement la tache . Le gain de temps est phénoménal .

En ce qui concerne postgres et mysql , la différence était faible dans ce cadre. Notamment aussi parce que , si besoin , les recettes sql peuvent se faire directement sur symfony comme sur Express.

Conclusion :

Le fait d’avoir terminé avec Express js en backend m’a permis de réellement avoir sur un même projet un comparaison de deux frameworks , de deux langages . Le dénominateur commun reste le langage de base de données.

Je trouve aussi que c’est vraiment difficile de faire un projet complet sans mettre en place des tests unitaires et fonctionnel , car les régressions de code sont vraiment source de beaucoup de perte de temps. Sur la partie symfony, je les ai mis en place pour la partie Entités.

Liens utiles :

repository projet sur symfony : https://github.com/zoutigo/hotel-group.git
repository projet sur express : https://github.com/zoutigo/hotels.git
Trello : https://trello.com/b/F6N7qDcN/ecf
GitMing : https://gitmind.com/app/diagrams/dir/dc689ccaba4774a621b229a0fa700aa0

Extraits de code importants

1. Backend
Le backend sera fait avec Express, Sequelize , Postgres sous Nodejs

1.1 Sequelize

1.1.1 Setup

Pour installer la dépendance sequelize avec le driver postgres, je lance la commandes suivantes :

npm install –save sequelize pg pg-hsstore

Ensuite , puisque je vais utiliser le syteme de migration de sequelize en CLI (command line interface) en développement, j'installe la dépendance avec la commande

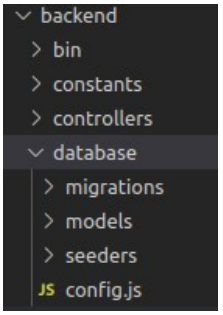
npm install sequelize-cli - - save-dev

Les dépendances étant prètes , je peux desormais initialiser mon projet.
A la racine de mon projet , je joue la commande

`npx sequelize-cli init

Cette commande va créer les dossiers migrations, models , seeders , ainsi qu'un fichier config pour la base de donnée.
Afin de mieux structurer mon projet , je déplace tous ces éléments dans un dossier nommé `backend` à la racine de mon projet et je le structure. Puis je crée un fichier .sequelizerc à la racine de mon projet .

```
const path = require('path')      zoutigo, last week • Feature/Entities

module.exports = {
  'models-path': path.resolve('backend', 'database', 'models'),
  'seeders-path': path.resolve('backend', 'database', 'seeders'),
  'migrations-path': path.resolve('backend', 'database', 'migrations'),
  config: path.resolve('backend', 'database', 'config.js'),
}
```



Le fichier de configuration config.js se trouve en annexe.
Je m'assure que le service posgresql est démarré sur ubuntu avec la commande

sudo systemctl start postgresql

Je crée ma base de données de développement avec la commande

npx sequelize-cli db:create

Je peux vérifier que ma base est bien crée , sur un terminal , je me connecte avec mon user `postgres` , je rentre les commandes suivantes :

> sudo i -u postgres
> psql
> \l

```
hotels_devs-# \l
                                  List of databases
       Name        |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-------------------+----------+----------+-------------+-------------+-----------------------
 augustin          | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 |
 augustin-dev-db   | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 |
 dvdrental         | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 |
 hotels_dev        | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 |
 hotels_devs       | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 |
 postgres          | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 |
 sequelize-tutorial | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 |
 template0         | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 | =c/postgres          +
                   |          |          |             |             | postgres=CTc/postgres
 template1         | postgres | UTF8     | fr_FR.UTF-8 | fr_FR.UTF-8 | =c/postgres          +
                   |          |          |             |             | postgres=CTc/postgres
(9 rows)
```

Je vois donc que ma base de données `hotels_dev` a bien été crée

1.1.2 Création des modèles de données

Pour créer mon modèle

```
npx sequelize-cli model:generate --name Example --attributes firstName:string,lastName:string,email:string
```

Cette commande va générer deux fichiers que je vais personnaliser : le modèle , et la migration.

1.1.2.1 Migration

1.1.2.1.a Migration simple

```
module.exports = {
  async up(queryInterface, Sequelize) {
    const transaction = await queryInterface.sequelize.transaction()
    try {
      await queryInterface.sequelize.query(
        'CREATE EXTENSION IF NOT EXISTS "uuid-ossp";'
      )
      await queryInterface.createTable(
        'users',
        {
          id: {
            allowNull: false,
            autoIncrement: true,
            primaryKey: true,
            type: Sequelize.INTEGER,
          },
          uuid: {
            type: Sequelize.UUID,
            defaultValue: Sequelize.literal('uuid_generate_v4()'),
          },
          lastname: {
            type: Sequelize.STRING(30),
            allowNull: false,
            validate: {
              notNull: {
                msg: 'le prénom est obligatoire',
              },
              len: {
                args: [2, 30],
                msg: 'le prénom doit avoir entre 2 et 30 caractères',
              },
            },
          },
          firstname: {
            type: Sequelize.STRING(30),
            allowNull: false,
            validate: {
              notNull: {
                msg: 'le nom est obligatoire',
              },
              len: {
                args: [2, 30],
                msg: 'le nom doit avoir entre 5 et 30 caractères',
              },
            },
          },
```

Dans ce fichier qui crée un transaction sql j'ai,
- rajouté une requête qui met en place l'uuid, car c'est avec uuid que je vais souvent traiter .
- défini ma clé primaire sur l'id et lui avoir mis l'auto incrément , ainsi que le type INTEGER de postgres,
- défini le champ uuid
- en enfin pour tous les autres champs , j'ai mis les validations de données que je souhaitais.

Ce fichier est le reflet de ce qui va se passer dans ma base de donnée dorénavant.

Je peux donc jouer ma migration avec la commande :

npx sequelize-cli db:migrate

Ceci va exécuter la migration et créer un table dans ma base de donnée.
Je procède donc de la même manière pour toutes les autres tables .

1.1.2.1.b Migration relation

Pour une relation entre User et House , je créée la migration suivante :

```
module.exports = {
  async up(queryInterface, Sequelize) {
    const transaction = await queryInterface.sequelize.transaction()
    try {
      await queryInterface.addColumn(
        'houses',
        'userId',
        {
          type: Sequelize.DataTypes.INTEGER,
          allowNull: true,
          references: {
            model: 'users',
            key: 'id',
            onUpdate: 'CASCADE',
            onDelete: 'SET NULL',
          },
        },      zoutigo, last week • Feature/Entities (#2) …
        { transaction }
      )
      await transaction.commit()
    } catch (error) {
      await transaction.rollback()
      throw error
    }
  },

  async down(queryInterface, Sequelize) {
    const transaction = await queryInterface.sequelize.transaction()
    try {
      await queryInterface.removeColumn('houses', 'userId', { transaction })
      await transaction.commit()
    } catch (error) {
      await transaction.rollback()
      throw error
    }
  },
}
```

Dans ce fichier ,je fais une transaction qui crée une table supplémentaire dans la table `houses` , en lui ajoutant le champ userId qui sera non null.
Je procède ainsi pour toutes les autres relations de mon modèle de données.

Pour vérifier toutes ces manipulations , je vais sur mon terminal

```
hotels_devs-# \dt
              List of relations
 Schema |     Name      | Type  |  Owner
--------+---------------+-------+----------
 public | SequelizeMeta | table | postgres
 public | bookings      | table | postgres
 public | houses        | table | postgres
 public | images        | table | postgres
 public | mails         | table | postgres
 public | suites        | table | postgres
 public | users         | table | postgres
(7 rows)
```

> \c hotels_dev
>\dt

Je constate donc que toutes mes tables ont bien été crées .

1.1.2.2 Model

Le modèle n'a pas d'incidence directe sur la base de donnée , mais c'est lui que je vais manipuler dans mon application Express. Je dois donc lui définit les mêmes caractéristiques que mes migrations , pour assurer l'intégrité des données , à savoir les validations, les contraintes et les relations.

Voici le modèle User

```
module.exports = (sequelize, DataTypes) => {
  You, 5 minutes ago | 2 authors (zoutigo and others)
  class User extends Model {
        You, 1 second ago • Uncommitted changes
    static associate({ house, booking }) {
      house.belongsTo(this, {
        onDelete: 'SET NULL',
        onUpdate: 'CASCADE',
      })
      this.hasOne(house, {
        foreignKey: 'userId',
      })

      this.hasMany(booking, {
        foreignKey: 'userId',
        onDelete: 'SET NULL',
        onUpdate: 'CASCADE',
      })
      booking.belongsTo(this)
    }

    toJSON() {
      return { ...this.get(), id: undefined }
    }
  }
```

```
User.init(
  {
    uuid: {
      type: DataTypes.UUID,
      defaultValue: Sequelize.UUIDV4,
    },
    lastname: {
      type: DataTypes.STRING(30),
      allowNull: false,
      validate: {
        notNull: {
          msg: 'le prénom est obligatoire',
        },
        len: {
          args: [2, 30],
          msg: 'le prénom doit avoir entre 2 et 30 caractères',
        },
      },
    },
    firstname: {
      type: DataTypes.STRING(30),
      allowNull: false,
      validate: {
        notNull: {
          msg: 'le nom est obligatoire',
        },
        len: {
          args: [2, 30],
          msg: 'le nom doit avoir entre 5 et 30 caractères',
        },
      },
    },

    email: {
      type: DataTypes.STRING(50),
      unique: true,
      validate: {
        isEmail: {
          msg: "ce format mail n'est pas valide",
        },
      },
    },
```

Dans la première partie de la classe , je définis mes associations
Dans la seconde , je mets les critères qui étaient déjà dans la migration.
Je reproduis donc la même opération pour toutes mes entités.

1.1.2.2 Seed ou fixtures

Contrairement au modèle et a la migration , le seed se crée avec une autre commande .

npx sequelize-cli seed:generate –name demo-user

Ceci a pour effet de créer un fichier dans le répertoire seeder que je dois améliorer.

Je définis l'admin et son mot de pass , puis les managers et quelques clients

```
module.exports = {
  async up(queryInterface, Sequelize) {
    // password hash

    const transaction = await queryInterface.sequelize.transaction()

    try {
      const password = await hashPassword('password')

      devAdmin.password = password
      const clients = devClients.map((client) => ({
        ...client,
        password: password,
      }))

      const managers = devManagers.map((manager) => ({
        ...manager,
        password: password,
      }))

      await queryInterface.bulkInsert('users', [devAdmin, ...managers], {
        transaction,
      })
      await queryInterface.bulkInsert('users', clients, {
        transaction,
      })

      await transaction.commit()
    } catch (err) {
      await transaction.rollback()
      throw err
    }
  },

  async down(queryInterface, Sequelize) {
    const transaction = await queryInterface.sequelize.transaction()

    try {
      await queryInterface.bulkDelete('users', null, { transaction })

      await transaction.commit()
    } catch (error) {
      await transaction.rollback()
      throw error
    }
  },
}
```

```
const devManagers = []
for (let a = 0; a < 10; a++) {
  const devManager = {
    lastname: faker.name.lastName(),
    firstname: faker.name.firstName(),
    email: `manager${a}@test.com`,
    password: 'password',
    roles: ['manager'],
    createdAt: new Date(),
    updatedAt: new Date(),
  }
  devManagers.push(devManager)
}

const devAdmin = {
  lastname: faker.name.lastName(),
  firstname: faker.name.firstName(),
  email: 'admin@test.com',
  password: 'password',
  roles: ['admin'],
  createdAt: new Date(),
  updatedAt: new Date(),
}

const devClients = []
for (let i = 0; i < 10; i++) {
  const client = {
    lastname: faker.name.lastName(),
    firstname: faker.name.firstName(),
    email: faker.internet.email(),
    password: 'password',
    roles: ['client'],
    createdAt: new Date(),
    updatedAt: new Date(),
  }
  devClients.push(client)
}
```

Je peux désormais jouer la fixture grace à la commande

npx sequelize-cli db:seed:all

Toutes ces données vont donc être crées dans ma base de donnée. Je le vérifie sur psql

```
hotels_devs=# SELECT email FROM users ;
             email
-------------------------------
 admin@test.com
 manager0@test.com
 manager1@test.com
 manager2@test.com
 manager3@test.com
 manager4@test.com
 manager5@test.com
 manager6@test.com
 manager7@test.com
 manager8@test.com
 manager9@test.com
 Maximo.Dietrich@gmail.com
 Van90@yahoo.com
 Marlen.Runolfsson24@yahoo.com
 Ward_Gleason@gmail.com
 Ken.Jaskolski@yahoo.com
 Shawn.Waelchi72@hotmail.com
 Christelle62@yahoo.com
 Danielle_Cronin39@hotmail.com
 Madyson_Zulauf17@yahoo.com
 Bethel11@hotmail.com
(21 rows)
```

Je peux donc me consacrer à la création du backend avec express.

1.2 Express

1.2.1 Initialisation et setup

1.2.1.1 Initialisation

Je préfère utiliser le générateur d'applications express , il est donc installé sur ma machine en global par la commande

```
npm install express-generator -g
```

Je peux donc l'utiliser pour initialiser mon projet en jouant

```
express backend
```

Cette commande va créer un dossier backend avec cette structure :

Je supprime dossier views
je déplace le dossier public à la racine du projet
je mets supprime le package.json , car c'est celui qui est à la racine du projet que j'utilise

Ensuite dans mon package.jon , je mets le bon script pour la commande 'start' :

```
"start": "cd backend && node ./bin/www",
```

1.2.1.2 Setup

J'installe nodemon pour avoir le redémarrage du server à chaque enregistrement

```
npm i nodemon
```

et je lui applique une petite configuration basique dans le package.json

```
"nodemonConfig": {
  "ignore": [
    "test/*",
    "docs/*",
    "dist/*"
  ],
  "delay": "2500",
  "events": {
    "crash": "sh -c 'lsof -i :${PORT:-3500} -t | xargs kill'"
  }
}
```

Je mets en place un système de gestion des erreurs personnalisés , avec les bons codes réponse :

Je mets en place un système de gestion des erreurs personnalisés , avec les bons codes réponse.

Ainsi , dans mon fichier app.js , je peux utiliser le middleware handleError

```
app.use(handleErrors)
```

Errors.js

```
class GeneralError extends Error {
  constructor(message) {
    super()
    this.message = message
  }

  getCode() {
    if (this instanceof BadRequest) {
      return 400
    }
    if (this instanceof NotFound) {
      return 404
    }
    if (this instanceof Unauthorized) {
      return 401
    }
    if (this instanceof Forbidden) {
      return 403
    }
    if (this instanceof TokenInvalid) {
      return 498
    }
    if (this instanceof PreConditionFailed) {
      return 412
    }
    if (this instanceof Conflit) {
      return 409
    }

    return 500
  }
}
```

handleErrors

```
const { GeneralError } = require('../utils/errors')

const handleErrors = (err, req, res, next) => {
  if (err instanceof GeneralError) {
    return res.status(err.getCode()).json({
      status: 'error',
      message: err.message,
    })
  }

  return res.status(500).json({
    status: 'error',
    message: err.message,
  })
}
```

Dans mon fichier app.js définis la Cors dans laquelle j'autorise 3 adresses uniquement.

```
const allowedOrigins = [
  'http://localhost:3000',
  'http://localhost:3001',      You, yesterday • 🐛 FIX:
  process.env.SERVER_ADRESS,
]

app.use(
  cors({
    origin: function (origin, callback) {
      if (!origin) return callback(null, true)
      if (allowedOrigins.indexOf(origin) === -1) {
        const msg =
          'The CORS policy for this site does not ' +
          'allow access from the specified Origin.'
        return callback(new Error(msg), false)
      }
      return callback(null, true)
    },
    credentials: true,
    exposedHeaders: ['authorization'],
  })
)
```

Je définis le répertoire d'images et j'autorise express à aller dans ces répertoires

```
app.use(express.static(path.join(__dirname, 'public')))
app.use('/images', express.static(path.join(__dirname, '..', 'public/images')))
```

Je décris ce que j'accepte comme entête de requettes

```
app.all('', (req, res, next) => {
  res.header('Access-Control-Allow-Origin', '*')
  res.header('Access-Control-Allow-Methods', 'PUT, GET, POST, DELETE, OPTIONS')
  res.header(
    'Access-Control-Allow-Headers',
    'Origin, X-Requested-With, Content-Type, Accept, Authorization'
  )
  next()
})
```

1.2.2 Création des routes

Les routes seront les points d'entrée de mon application après les controles que je viens de mettre en place précédemment.

Je les implémente dans le fichier app.js de la manière suivante :

```
app.use('/api/login', loginRouter)
app.use('/api/users', usersRouter)
app.use('/api/suites', suitesRouter)
app.use('/api/houses', housesRouter)
app.use('/api/bookings', bookingsRouter)
app.use('/api/mails', mailsRouter)
```

```
const usersRouter = require('./routes/users')
const suitesRouter = require('./routes/suites')
const housesRouter = require('./routes/houses')
const bookingsRouter = require('./routes/bookings')
const mailsRouter = require('./routes/mails')
const loginRouter = require('./routes/login')
```

Toutes les routes sont stockées dans un dossier routes
Le login se fait dans une route à part.
Prenons l'exemple de la route suites

```
/* GET suites listing. */
router.get('/', getSuiteList)

/* POST suites creating. */
router.post('/', verifyTokenService, uploadImages.array('files', 20), postSuite)

/* GET suite. */
router.get('/:suiteUuid', getSuite)

/* PUT suites updating. */
router.put(
  '/:suiteUuid',
  uploadImages.array('files', 20),
  verifyTokenService,
  putSuite
)

/* DELETE suites updating. */
router.delete('/:suiteUuid', verifyTokenService, deleteSuite)

/* DELETE suite image. */
router.delete('/:suiteUuid/:imageUuid', verifyTokenService, deleteImage)

module.exports = router
```

Pour obtenir la liste des suites , ce sera la route '/suites avec la méthode get . C'est la méthode getSuiteList du controller traitera cette requette.
Pour obtenir une suite en particulier , il faut mettre son uuid en paramètre de la requette.
Pour créer une suite , on fait appel à deux middleware , l'un pour récupérer les informations qui vont permettre autorisation, l'autre pour le stockage des images.

Dans le cadre de ce mémoire , intéressons nous à l'authentification, l'autorisation et à la création d'une suite pour le reste de la démonstration.

Les controllers ayant besoins de services , on va créer quelques services avant de passer aux controllers

1.2.3 Création des services

Nous allons regarder 3 services en particulier
- Authentification
- Création d'une suite
- Vérification de la disponibilité d'une suite pour réservation

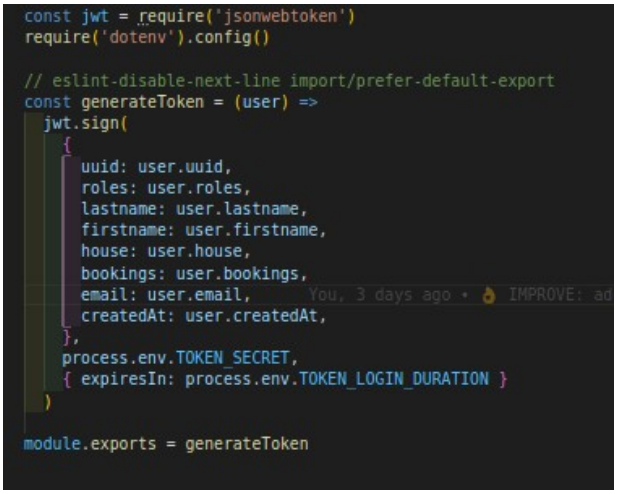
1.2.3.1 Service authentification

L'utilisateur va donner un mot de pass et un email . Le but de ce service est de lui retourner un token valable pour une durée définie dans les variables d'environnement

authenticate.js

```
const bcrypt = require('bcrypt')
const { BadRequest } = require('../../utils/errors')
const { user, house, suite } = require('../../database/models')
const generateToken = require('../../utils/generateToken')
const { userInclude, userTokenInclude } = require('../../constants/includes')

const authenticate = async (req, res, next) => {
  const { username: email, password } = req.body
  if (!email || !password) {
    return next(new BadRequest('password or email missing'))
  }

  try {
    const userVerified = await user.findOne({
      where: { email },
      include: userTokenInclude,
    })

    if (!userVerified) {
      return next(new BadRequest('utilisateur inconnu'))
    }

    // check password
    const passwordVerified = await bcrypt.compare(
      password,
      userVerified.password
    )
    if (!passwordVerified)
      return next(new BadRequest('email ou mot de pass invalide'))

    return res.status(200).send({
      message: 'connection effectuée avec succès',
      token: generateToken(userVerified),
    })
  } catch (error) {
    return next(error)      You, 1 second ago • Uncommitted changes
  }
}

module.exports = authenticate
```

generateToken.js



```
const jwt = require('jsonwebtoken')
require('dotenv').config()

// eslint-disable-next-line import/prefer-default-export
const generateToken = (user) =>
  jwt.sign(
    {
      uuid: user.uuid,
      roles: user.roles,
      lastname: user.lastname,
      firstname: user.firstname,
      house: user.house,
      bookings: user.bookings,
      email: user.email,      You, 3 days ago • IMPROVE: ad
      createdAt: user.createdAt,
    },
    process.env.TOKEN_SECRET,
    { expiresIn: process.env.TOKEN_LOGIN_DURATION }
  )

module.exports = generateToken
```

Pour réaliser ce service on a besoin de deux dépendances
bcrypt et jsonwebtoken

on lance donc la commande
npm i bcrypt jsonwebtoken

Dans le script du fichier authenticate ,
- on vérifie si la requette possède un mail et un mot de pass , sinon on renvoie un erreur BadRequest par le système de gestion des erreurs que nous avons implémenté

- on vérifie s'il existe dans la base de donnée , grace à une requette faite sur le modèle de donnée user que nous avons crée avec sequelize . S'il n'existe pas , on renvoie une erreur.
Cette requette du userVerified aurait pu se faire en sql directement dans le code , mais elle aurait été trop longue à écrire dans la mesure où nous demandons que la requette nous retourne des tables qui sont liées à l'utilisateur , comme la table booking ou bien la table house. La syntaxe sequelize est donc plus appropriée ici.

- on décrypte le mot de pass avec bcrypt , et on compare dans la foulée au mot de pass fourni dans la requette

Si tout est bon , on renvoie une réponse de réquette réussie , en créant le token .

Si la création du token échoue , la réponse du server sera une erreur 500 puisqu'on est toujours dans notre block try-catch

Sinon , l'utilisateur recevra son token qui sera décodé en frontend

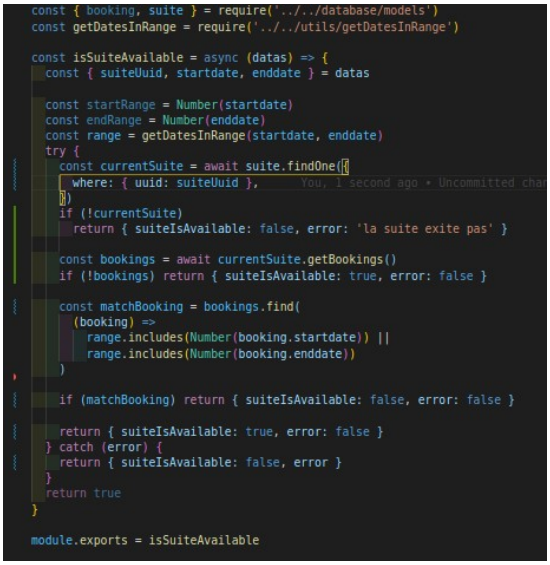
1.2.3.2 Création d'une suite

Ce service est appelé depuis le controller.
Le controller lui a fourni les données qu'il faut pour créer la 'suite'. Parmis ces données figurent un tableaux qui contient les chemins des images , et l'uuid de la 'house'. Car en base de donnée une suite est liée à une 'house'. Les autres informations sont celles qui sont propres à la suite , comme son nom , sa description, etc..

```
const { suite, house } = require('../../database/models')
const getValidationErrorsArray = require('../sequelize/getValidationErrorsArray')

const CreateSuiteService = async (data) => {
  const { images, houseUuid, ...rest } = data
  const createdAt = new Date()
  try {
    const requestedHouse = await house.findOne({ where: { uuid: houseUuid } })
    if (!requestedHouse) {
      return { error: "l'établissement n'existe pas" }
    }

    await requestedHouse.createSuite({ ...rest, createdAt })

    const createdSuite = await suite.findOne({ where: { createdAt } })

    const filenames = await Promise.all(
      images.map(async (image) => {
        await createdSuite.createImage(image)
        return image.filename
      })
    )

    if (filenames.leght === images.lenght) {
      return { createdSuite }
    }
    return { error: "une erreur s'est produite" }
  } catch (error) {      zoutigo, 5 days ago • Feature/users-endpoints-logic (#4) ...
    return { errors: getValidationErrorsArray(error) }
  }
}

module.exports = CreateSuiteService
```

- on récupère l'uuid de la 'house' et le tableau des images en destructurant les data recues,
- on crée une qui va nous permettre de faire une requette un un objet nouvellement crée.
- on recherche la house concernée,
- grâce aux spécificités des models sequelize , on dit que 'la house crée une suite '.
C'est un raccourci qui nous affranchi de rechercher l'id , ça tombe bien , dans nos modèles on empêche le renvoie de l'id.
- on récupère donc l'instance de la suite fraîchement crée grâce à une requette par la date de création
- on utilise le systeme de promesse de ES6 pour créer toutes images qui vont à cette instance de 'suite'.
- on compare si la longueur du tableaux des images envoyées pas le controller est égale à celle des actions exécutées dans la promesse .
- Puis on envoie en réponse au controller soit la nouvelle suite , soit une erreur.
Si quelque chose se passe mal , le bloc catch renvoie aussi une erreur au controlleur.

1.2.3.2 Vérification de la disponibilité d’une suite
Le principe est le même que pour la création d’une suite .
Le point de vigilance ci est de vérifier si le créneau est disponible



```
const { booking, suite } = require('../../database/models')
const getDatesInRange = require('../../utils/getDatesInRange')

const isSuiteAvailable = async (datas) => {
  const { suiteUuid, startdate, enddate } = datas

  const startRange = Number(startdate)
  const endRange = Number(enddate)
  const range = getDatesInRange(startdate, enddate)
  try {
    const currentSuite = await suite.findOne({
      where: { uuid: suiteUuid },
    })
    if (!currentSuite)
      return { suiteIsAvailable: false, error: 'la suite exite pas' }

    const bookings = await currentSuite.getBookings()
    if (!bookings) return { suiteIsAvailable: true, error: false }

    const matchBooking = bookings.find(
      (booking) =>
        range.includes(Number(booking.startdate)) ||
        range.includes(Number(booking.enddate))
    )

    if (matchBooking) return { suiteIsAvailable: false, error: false }

    return { suiteIsAvailable: true, error: false }
  } catch (error) {
    return { suiteIsAvailable: false, error }
  }
  return true
}

module.exports = isSuiteAvailable
```

```
const getDatesInRange = (startDate, endDate) => {
  // startdate and enddate are timestamp : 1656028800000
  const date = new Date(startDate)

  const dates = []

  while (date <= endDate) {
    dates.push(new Date(date).getTime())
    date.setDate(date.getDate() + 1)
  }

  return dates
}

module.exports = getDatesInRange
```

J’ai choisi ici de manipuler les dates en javascript timestamp.
- Je commence par écrire un petit utilitaire ‘getDatesInRange’ qui va créer un tableau de dates en timestamp
- Je transforme donc la date de début et de fin en un tableau de dates.
- Je fais une requette pour récupérer la suite , puis toutes les réservations associées par le getter.
- Je cherche dans le tableau des réservations si la date de début ou bien la date de fin sont compris dans la liste des dates que j’ai généré. Si oui , alors le créneau n’est pas disponible. Dans tous les cas , je renvoie une réponse que controller qui indique une erreur et une disponibilité.

1.2.3 Création des controllers

Les controllers sont stockés dans le répertoire ./backend/controllers
Chaque controller a des méthodes qui sont invoquées dans les routes

Dans u login

1.2.4 Essais

2. React

Annexes.

1. config js de sequelize

```
require('dotenv').config()      zoutigo, last wee
module.exports = {
  development: {
    username: process.env.POSTGRES_USER,
    password: process.env.POSTGRES_PASSWORD,
    database: process.env.POSTGRES_DB,
    host: '127.0.0.1',
    port: 5432,
    dialect: 'postgres',
    dialectOptions: {
      bigNumberStrings: true,
    },
  },
  test: {
    username: process.env.POSTGRES_USER,
    password: process.env.POSTGRES_PASSWORD,
    database: process.env.POSTGRES_DB,
    host: '127.0.0.1',
    port: 3306,
    dialect: 'postgres',
    dialectOptions: {
      bigNumberStrings: true,
    },
  },
  production: {
    username: process.env.PROD_DB_USERNAME,
    password: process.env.PROD_DB_PASSWORD,
    database: process.env.PROD_DB_NAME,
    host: process.env.PROD_DB_HOSTNAME,
    port: process.env.PROD_DB_PORT,
    dialect: 'postgres',
    dialectOptions: {
      bigNumberStrings: true,
    },
  },
}
```