Dossier de projet professionnel

Deepkit

Alexandre SCALISI

Sommaire

Introduction **3**
Présentation 3
Résumé du projet 4
Compétences couvertes par le projet 5
Cahier des charges **6**
Analyse de l'existant 6
Les utilisateurs du projet 6
Les fonctionnalités attendues 6
Contexte technique 13
Conception du projet **13**
Choix des frameworks 13
Organisation du projet 15
Architecture logicielle 16
Conception du frontend de l'application **17**
Plan de l'application mobile 17
Maquettage 20
Conception du backend de l'application **22**
Développement du backend **23**
Création base de données 23
Création API 25
Tests 29
Authentification 30
Gestion des droits 31
Upload de fichier 32
Sécurité 33
Exemple de problématique rencontrée 34
Exemple de recherche anglophone 34
Développement du frontend **35**
Modules Principaux 35
Authentification 38
Validations 40
Routes 42
Conclusion **42**

Introduction :

Présentation

Je m'apelle Alexandre SCALISI, et j'ai 27 ans.

Mon introduction à la programmation s'est faite avec openclassroom où j'ai pu apprendre les bases de la programmation avec Java avant de commencer à apprendre à developper des jeux avec Unity et C#.

Fin 2020 je décidais de rejoindre une formation de developpeur web et web mobile à l'AFPA de Istres afin d'obtenir un titre professionnel et découvrir le développement web que je ne connaissais pas encore.

Enfin en septembre 2021, je décidais de rejoindre la coding school 2 de la plateforme afin de continuer à monter en compétence tout en acquérant une expérience professionnelle en alternance.

Résumé du projet

Deepkit est une application d'emprunt de matériel visant à aider des entreprises comme des bibliothèques par exemple à permettre à leurs client d'emprunter facilement du materiel grâce à un système de qr code.

L'application est en deux parties :

- Une partie backend pour l'API qui communique avec la base de données
- Une partie frontend qui est l'application mobile que l'utilisateur final utilise et qui envoie des requêtes à l'API.

Les client doivent créer un compte sur l'application et préciser l'url de l'entreprise où ils veulent emprunter pour pouvoir commencer à emprunter du materiel.

Les clients ont la possibilité de voir une liste des objets disponibles, ils peuvent réserver, mettre en favoris ou bien emprunter à l'aide d'un qr code.

Les clients peuvent aussi accéder à leur profil, modifier leur profil, donner leur avis et voir l'historique de leurs actions.

L'application est pensée pour permettre à des entreprises qui stockent leur matériel dans plusieurs salles différentes d'avoir un admin pour chaque salles.

Il existe donc deux types d'administrateurs :

- L'administrateur d'une salle qui peut gérer le matériel de sa salle à l'aide d'un panel admin et à qui on doit donner le materiel à rendre pour qu'il le scan pour valider le rendu
- Le SuperAdmin qui peut ajouter du matériel, et ajouter des nouveaux administrateurs de salles.

Compétences couvertes par le projet

Ce projet couvre les compétences du titre suivantes :

- Maquetter une application
- Développer des composants d’accès au données
- Concevoir une base de données
- Mettre en place une base de données
- Développer des composants dans le langage d’une base de données
- Concevoir une application
- Développer des composants métier
- Construire une application organisée en couche
- Développer une application mobile
- Préparer et exécuter des plans de tests d’une application
- Préparer et exécuter le deploiement d’une application

Cahier des charges :

Analyse de l'existant

Il existe déjà quelques application d'emprunt de matériel comme « Take A Book » mais qui est spécialisée dans les bibliothèques.

Mon application se veut plus généraliste et veut permettre à plein d'entreprise de se digitaliser dans le domaine de l'emprunt de materiel et leur faire gagner du temps.

Les utilisateurs du projet

L'application se décompose en trois parties :

- La partie utilisateur qui permet à toutes les personnes inscritent de consulter l'inventaire, d'emprunter, et de réserver du matériel.
- La partie administrateur de salles qui permet à un administrater de valider le rendu de matériel et de gérer sa salle.
- La partie SuperAdmin qui a le plus de droit et qui peut même créer un admin de salle ou ajouter du matériel et des nouvelles catégories.

Les fonctionnalités attendues

Global :

Page selection serveur :

Sur cette page on doit avoir une liste de serveurs enregistrés.

Il doit y avoir un bouton pour pouvoir ajouter de nouveau serveurs.

Chaque serveur dans la liste est représenté par une petite card contenant le nom du serveur ainsi que des boutons d'actions.

Il doit y avoir un bouton pour éditer le serveur, un pour le supprimer, et un pour se login (choix utilisateur, admin salle, superadmin)

Page ajouter serveur :

Cette page sert à enregister un nouveau serveur à l'aide d'un formulaire et on peut aussi récupérer l'url du serveur en scannant un code barre si disponible.

Page inscription :

Cette page nous permet de créer un nouvel utilisateur à l'aide d'un formulaire.

Page connexion :

Cette page nous permet de nous connecter à l'aide d'un formulaire et contient un lien vers la page mot de passe oublié

Page modifier serveur :

Dans cette page nous pourrons modifier un serveur.

Il y aura deux formulaire :

- Un pour modifier les infos du serveur
- Un pour modifier l'utilisateur lié au serveur

Page mot de passe oublié :

Sur cette page, il faut entrer notre email et la personne correspondant à l'email recevra le code si l'email existe.

Page nouveau mot de passe :

Ici nous pouvons réinitialiser modifier notre mot de passe à condition que notre code corresponde avec le code envoyé par email précédemment.

Utilisateur :

Page Inventaire :

C'est la page principal de l'utilisateur, il peut y retrouver tous les objets disponibles sur le serveur où il est connecté.

Il peut faire une recherche par nom, filtrer par catégories ou garder que les objets disponibles en stock.

Il peut aussi ranger par ordonner par nom, par note ou par date d'ajout.

Les objets sont représentés par des cards contenants une photo, le nom, les catégories, la description, s'il est en stock ou non, une possibilité de donner notre avis, de reserver, d'ajouter en favoris, ou de cliquer sur un bouton pour voir plus de détails sur le produit.

Page Inventaire Details :

Cette page liste les disponibilités / reservations / emprunts par salles.

Nous pouvons faire une recherche par salle ou bien montrer que les salles qui ont des objets en stock ou choisir les salles que l'on veut afficher

Nous pouvons aussi ordonner par nom (croissant / décroissant)

Page avis :

Ici nous pouvons voir les avis des utilisateurs concernants un objet.

Notre avis (s'il existe) apparaîtra en premier.

Nous aurons la possibilité de trier par date de création des avis

Page profil :

Cette page permet à l'utilisateur de se créer un profil ou de modifier ses informations comme son mot de passe, sa photo ou son pseudo.

La page est divisée en deux parties :

- Un formulaire pour voir / modifier ses informations
- Un formulaire pour modifier son mot de passe

Pour finir il y aura un bouton nous permettant de supprimer notre compte. Cependant il y aura un message d'alerte nous interdisant de supprimer notre compte si nous avons encore des emprunts en cours.

Page historique :

Ici nous pourrons voir l'historique de tous nos emprunts, nos réservations et nos rendus.

Il y aura un trois code couleurs pour différencier chaque actions :

- Du vert pour les emprunts
- Du bleu pour les reservations
- Du rouge pour les rendus

Nous pourrons aussi filtrer pour voir ce qui nous interesse mais aussi nous pourrons trier par date.

Page mes emprunts / mes reservations / mes rendus / suivis :

Quatre pages differentes comportant respectivement la liste de tous nos emprunts / reservations / rendus / suivis avec des cards suivant les codes couleur de la page historique.

Nous pouvons également trier par date.

Page emprunter :

La page emprunter contient un scanner permettant de scanner un qr d'un objet pour l'emprunter. Un message d'erreur apparaitra s'il est déjà emprunté.

Administrateurs de salles :

Page valider rendu :

L'utilisateur rend son objet à l'admin de la salle qui ensuite doit scanner le qr code de l'objet pour valider le rendu.

Cette page contient donc un scanner pour valider le rendu.

Page gérer inventaire :

Cette page permet à un administrateur de salle de gérer l'inventaire de sa salle.

Il peut voir les objets disponibles, en augmenter leur quantité, ou en supprimer.

Page Inventaire details:

Cette page permet à un admin de salle de voir les détails d'un objet.

Page gérer mes elements d'inventaires :

Cette page permet à un admin de salle de voir chaque élement unique de l'inventaire pour savoir son status, ou le supprimer

Page Historique :

Historique de toutes les actions effectuées dans la salle. Utilise le même code couleur que l'historique de l'utilisateur basique.

Page Dashboard :

Le dashboard de l'admin de salle donne des statistiques par rapport à la salle.

Super Admin :

Page gérer utilisateurs :

La page gérer utilisateurs permet de voir les utilisateurs du serveur et les modérer mais aussi elle permet d'ajouter un autre superadmin grace à son mail (il recevra un mot de passe autogénéré par mail)

Page gerer les salles :

Cette page permet de gérer les salles.

On peut cliquer sur un bouton pour voir les détails d'une salle et sur un autre pour voir l'inventaire de la salle.

Page Ajouter salle :

Permet de créer une nouvelle salle avec un admin de salle et un inventaire.

Page salle details / editer salle :

Permet de voir les détails d'une salle ou de la modifier.

On peut ajouter un admin de salle avec son email (il recevra un mot de passe autogénéré par mail)

Page gérer les inventaires

Pareil que pour admin de salles mais peut voir les inventaires de toutes les salles et peu en créer et en modifier contrairement à l'admin de salle

Page Ajouter / Modifier inventaire

Permet d'ajouter ou de le modifier

Page inventaire détails gerer les elements inventaires :

Pareil que pour admin salle

Page gérer les catégories

Permet au superadmin de gérer les catégories

Page Ajouter / Modifier catégories

Le superadmin peut créer et modifier des catégories

Page Historique :

Cette page permet de voir l'historique de toutes les transactions de toutes les rooms.

Page Dashboard :

Permet de voir quelques stats des rooms à l'aide de graphiques.

Contexte technique

L'application devra être accessible sur android dans un premier temps puis sera aussi disponible sur iOS

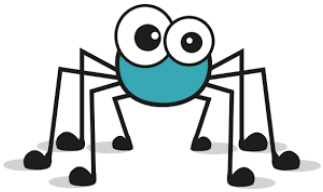
Conception du projet :

Choix des frameworks :

Pour mon projet j'avais besoin d'une application mobile et d'une API pour me permettre de persister des données.

Mon application se divise a donc une partie backend et une partie frontend.

Backend :



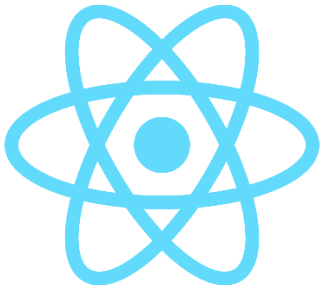
Pour mon API j'ai fait le choix d'utiliser API Platform, un framework PHP permettant de créer des API, couplé à symfony.

J'ai fait ce choix car je connaissais déjà un peu symfony et qu'API platform est très bien pour créer des API tout en étant assez facile d'utilisation et permet de faire beaucoup de choses.

Il nous permet de rapidement créer des endpoints CRUD pour nos entités tout en nous permettant de valider les données et choisir les données à envoyer ou reçevoir.

API platform nous permet aussi de créer automatiquement une documentation aux normes OpenAPI ce qui permet à des utilisateurs exterieurs de bien comprendre notre application.

Frontend :



Pour mon application mobile j'ai choisi le framework React Native (avec Expo).

Ce choix a été très simple car mon langage de programmation préféré est javascript, qu'on a un peu travaillé avec ReactJs au début de l'année et que j'utilise React pour mes projets d'alternance.

React Native est assez simple à utiliser et permet de créer des applications mobiles réactives.

React Native utilisant node, nous avons accès à de nombreux packages de npm.

J'utilise le package React navigation pour gérer la navigation de l'application et qui permet de créer des onglets (Tabs), empiler des pages (Stacks) ou créer un drawer.

J'utilise aussi le package react native paper pour avoir accès à des composants respectant le material design de google.

Organisation du projet :

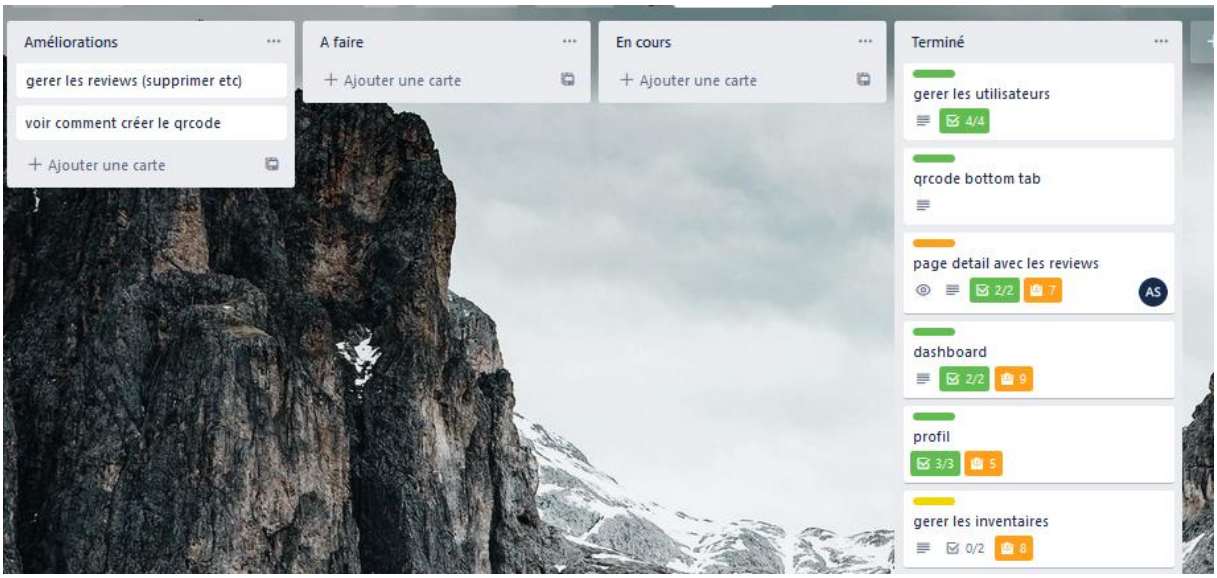
Ce projet étant assez gros, je savais qu'il allait me prendre plusieurs mois.

Pour mieux optimiser mon temps je devais avoir une bonne organisation.

Pour m'aider j'ai utilisé le fameux outil Trello.

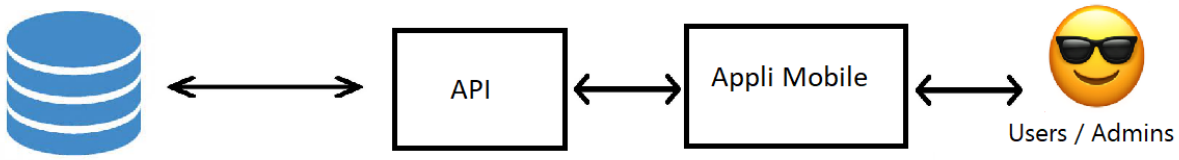
Pour ce faire j'ai créé plusieurs tableaux pour plusieurs sous tâches même si la plupart du temps je ne suivais pas les tâches à la lettre.

J'avais plusieurs cartes (améliorations, à faire, en cours, terminé) et aussi un power-up « Card Size » pour donner des poids à mes tâches afin d'estimer le temps que je pense que la tâche me prendra.



Architecture logicielle :

L'API communique avec la base de donnée et l'application mobile et l'utilisateur communique avec l'application mobile.

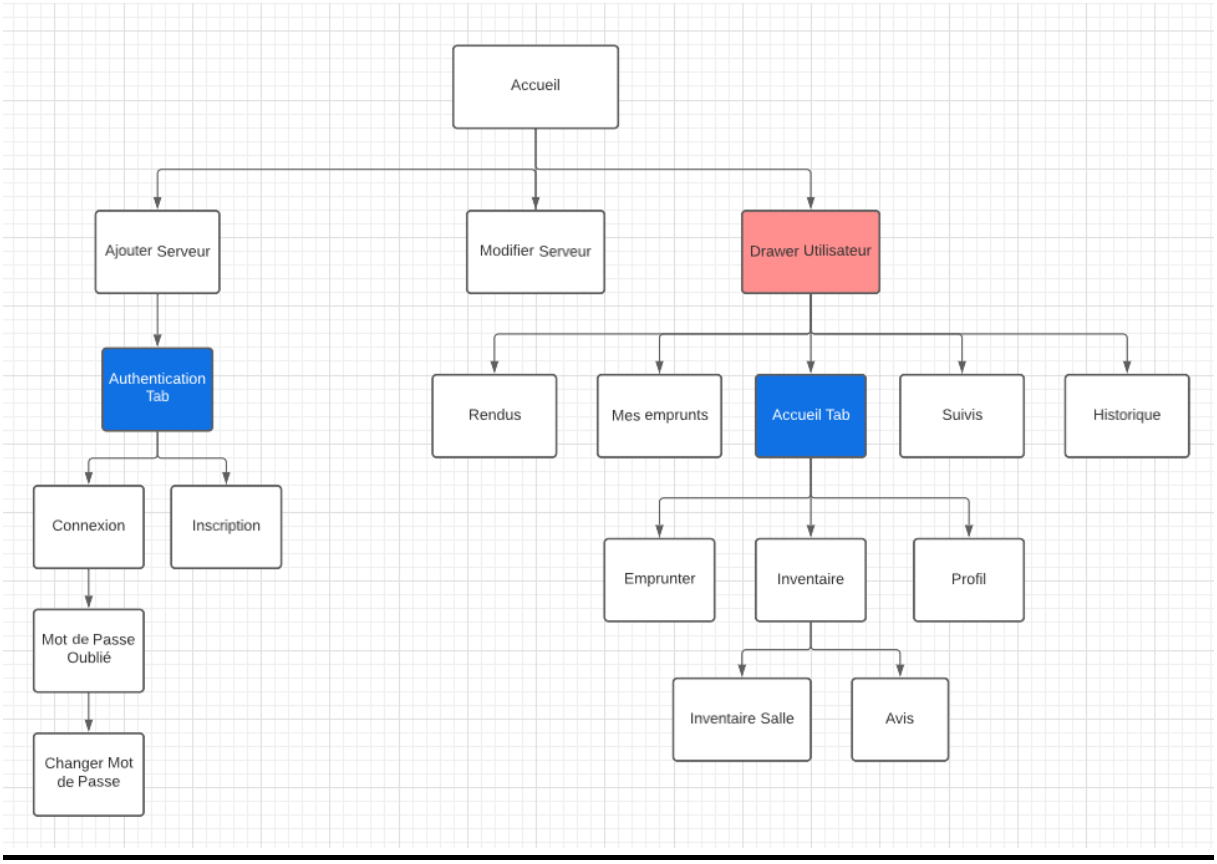


Conception du Frontend de l'application :

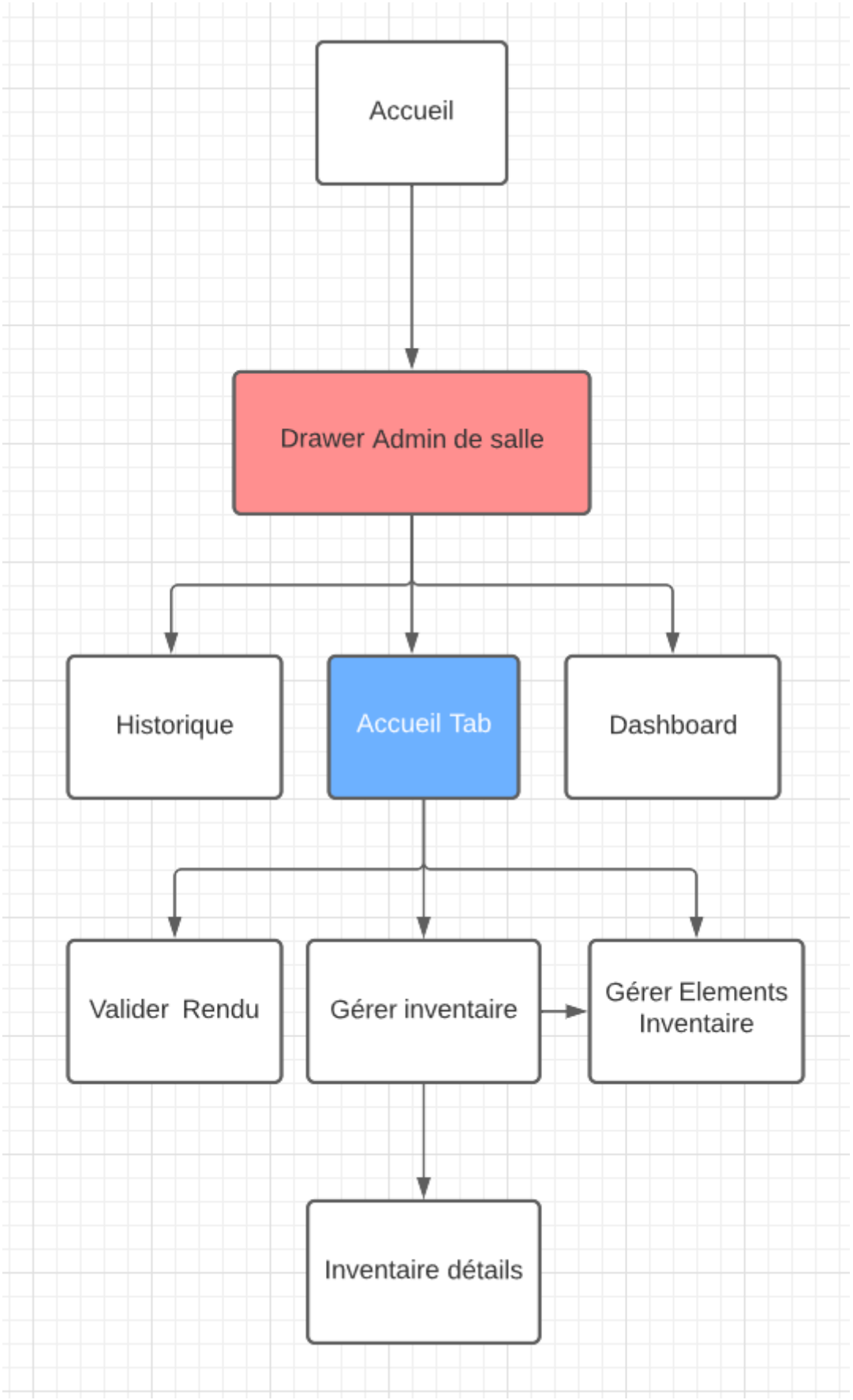
Plans de l'application mobile :

Pour me faire une idée de la structure de l'application j'ai fait un plan de l'application.

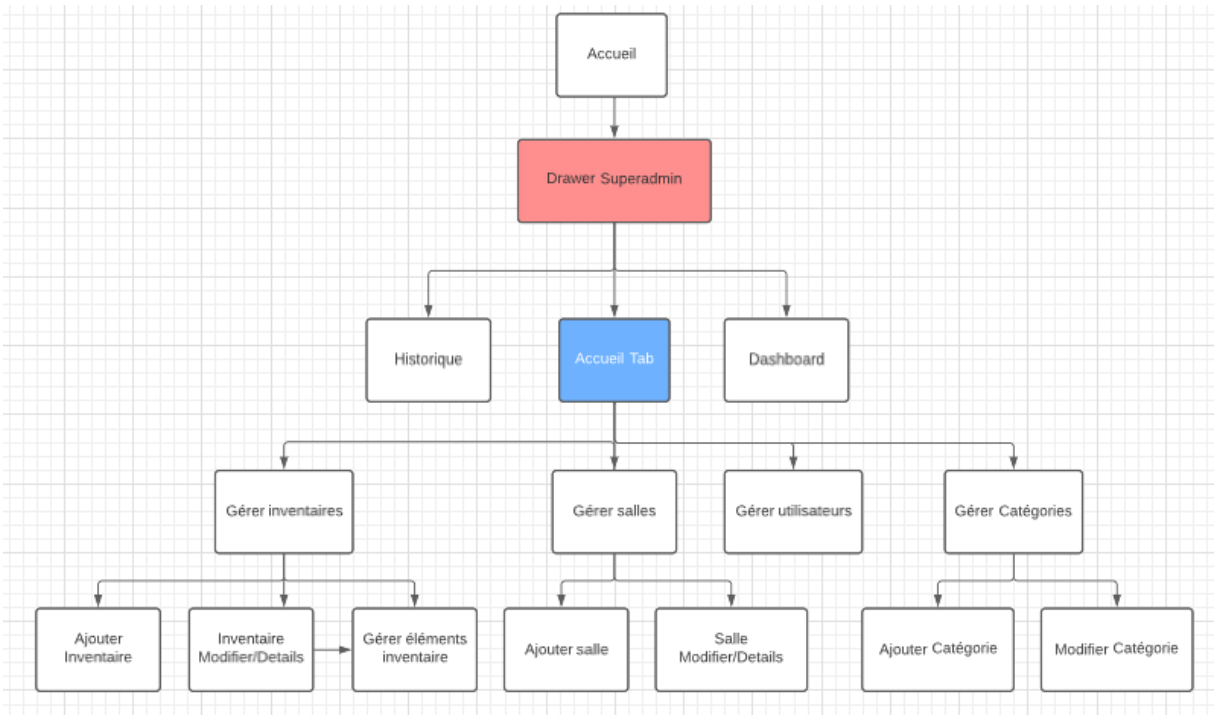
Utilisateur :


Accueil
Ajouter Serveur
Modifier Serveur
Drawer Utilisateur
Authentication Tab
Rendus
Mes emprunts
Accueil Tab
Suivis
Historique
Connexion
Inscription
Emprunter
Inventaire
Profil
Mot de Passe Oublié
Inventaire Salle
Avis
Changer Mot de Passe

Admin de salle :



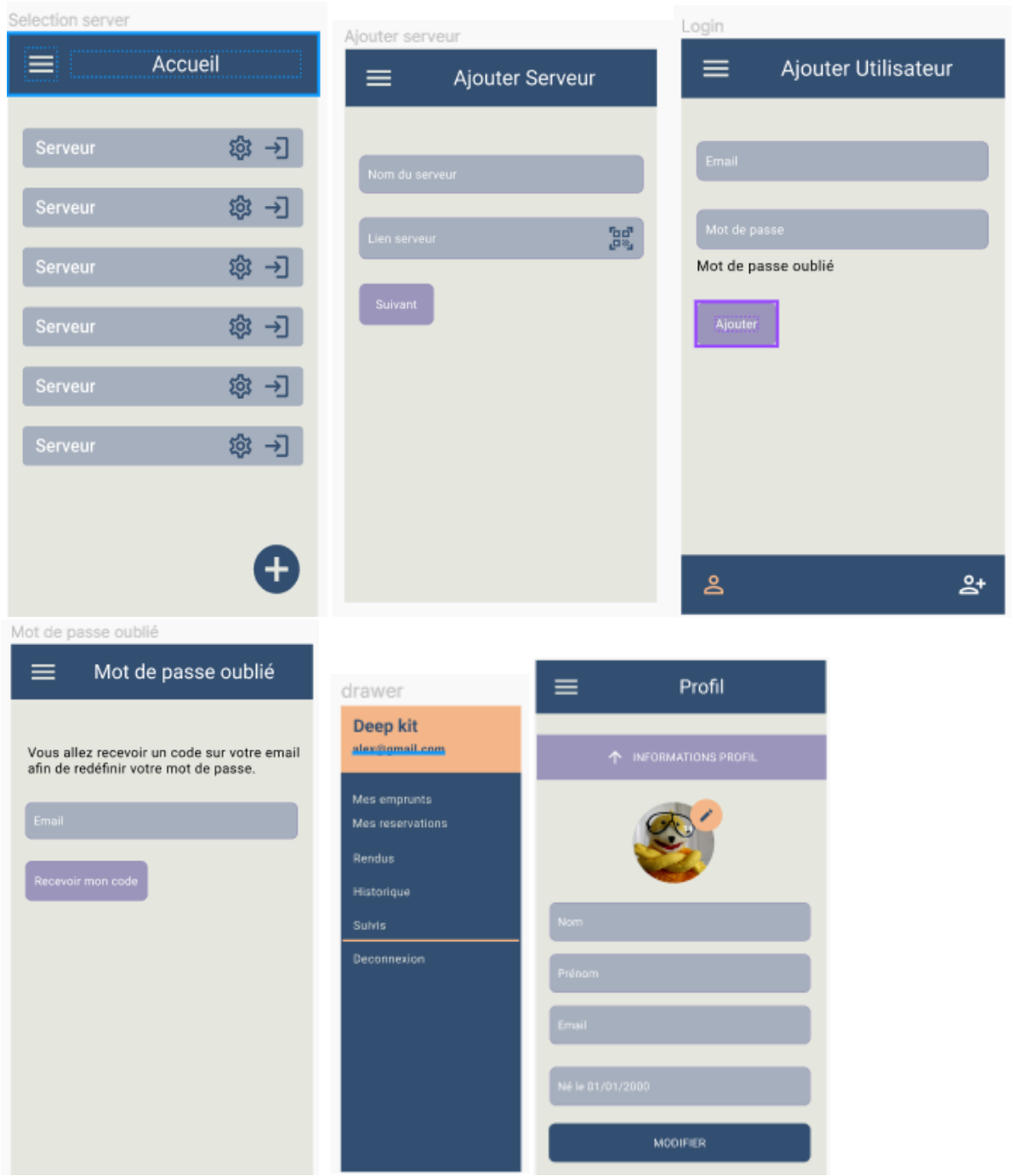
Superadmin :

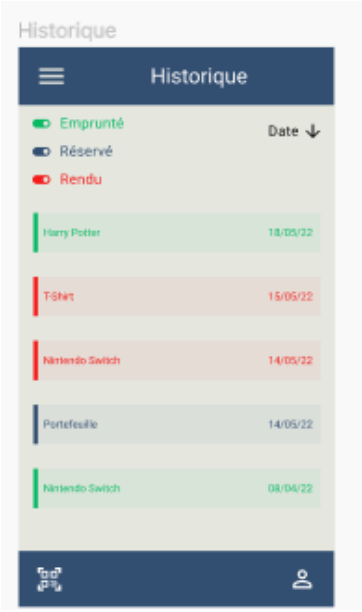

Accueil
Drawer Superadmin
Historique
Accueil Tab
Dashboard
Gérer inventaires
Gérer salles
Gérer utilisateurs
Gérer Catégories
Ajouter Inventaire
Inventaire Modifier/Details
Gérer éléments inventaire
Ajouter salle
Salle Modifier/Details
Ajouter Catégorie
Modifier Catégorie

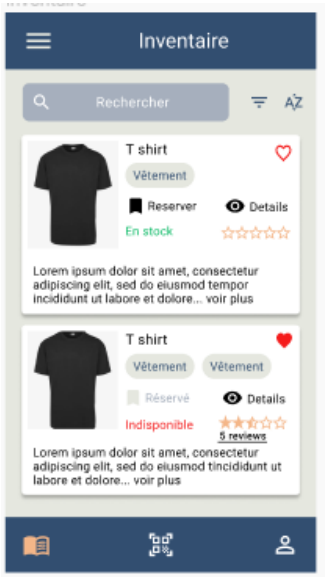
Maquettage :

J’ai aussi fait des maquettes sur Figma pour finaliser mon design.

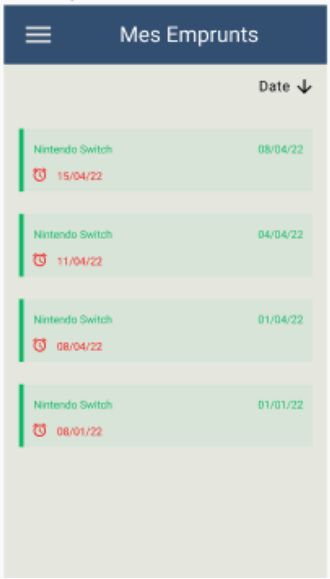
En voici quelques pages :

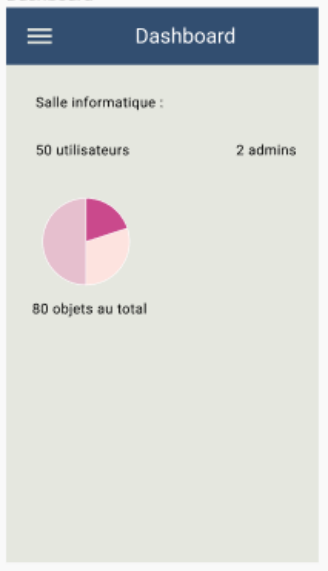



Historique
Historique
Emprunté
Réservé
Rendu
Date
Harry Potter
18/05/22
T-Shirt
15/05/22
Nintendo Switch
14/05/22
Portefeuille
14/05/22
Nintendo Switch
08/04/22


Inventaire
Rechercher
T shirt
Vêtement
Reserver
Details
En stock
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore... voir plus
T shirt
Vêtement
Vêtement
Réservé
Details
Indisponible
5 reviews
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tincididunt ut labore et dolore... voir plus


Filtre inventaire
Filtrer par:
En stock
Catégories
Vêtement
Livre
Vêtement
Sort inventaire
Nom
Note
Date d'ajout


Mes Emprunts
Date
Nintendo Switch
08/04/22
15/04/22
Nintendo Switch
04/04/22
11/04/22
Nintendo Switch
01/04/22
08/04/22
Nintendo Switch
01/01/22
08/01/22

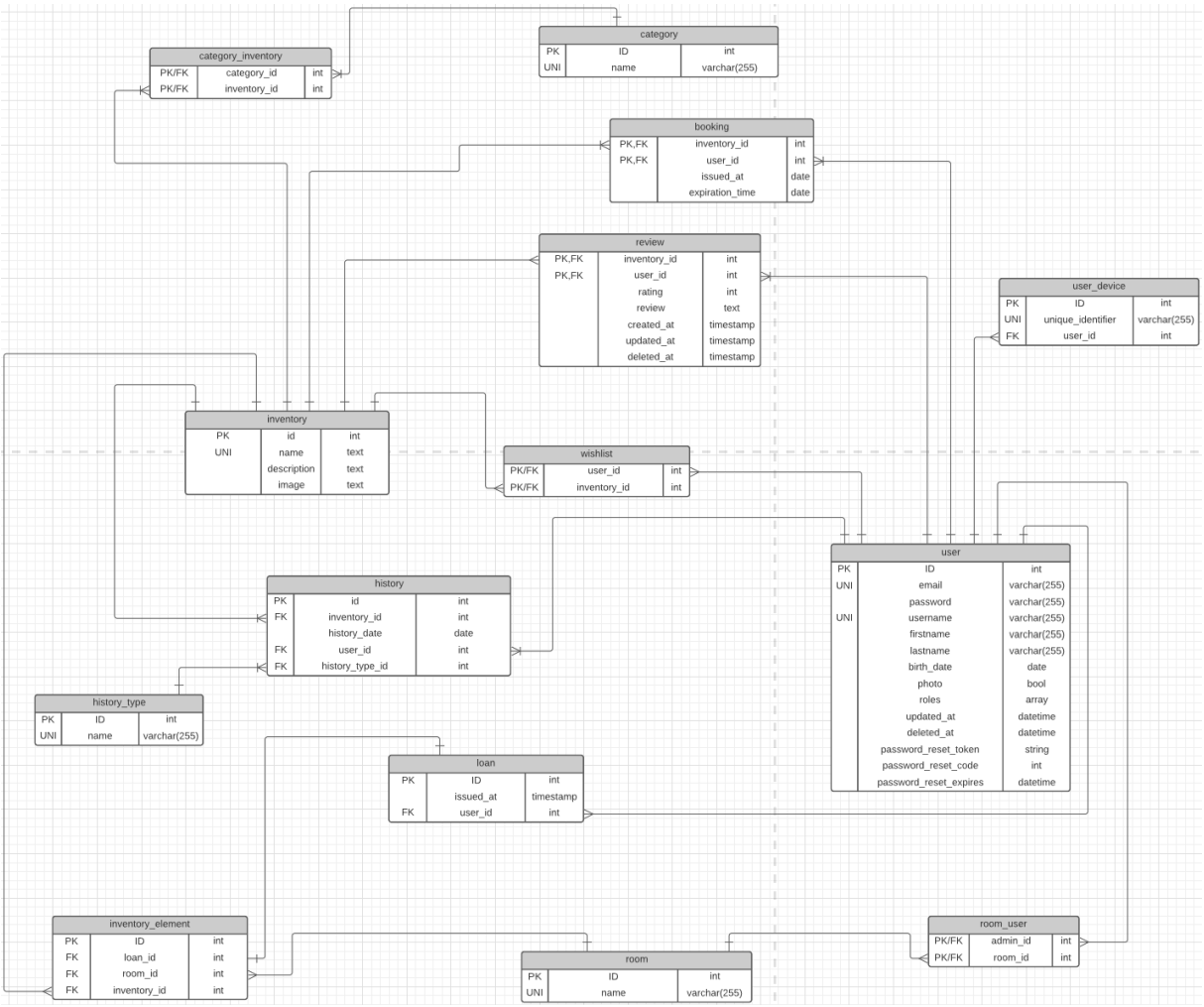

Dashboard
Salle informatique :
50 utilisateurs
2 admins
80 objets au total


Gérer inventaires
Rechercher
nom
quantité
image
actions
tshirt
12
Page 1 / 10

Conception du Backend de l’application :

Pour ma base de données, j’ai fait le choix de MySQL car c’est ce que je connais le mieux.

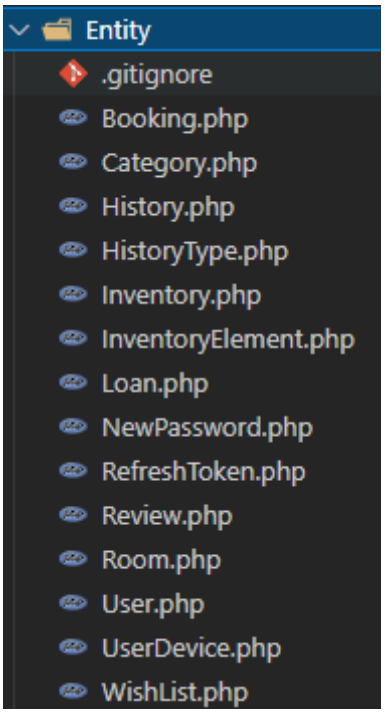
Pour concevoir ma base de données j’ai directement construit un Model Physique de données avec LucidChart.



Développement du backend:

Creation Base De Données

Pour créer mon API avec API Platform, j'ai d'abord commencer par créer mes entités qui sont une representation orientée objet des tables de MySQL.




```php
#[ORM\Entity(repositoryClass: CategoryRepository::class)]
#[ApiResource(
    normalizationContext: ["groups" => ['category:read:collection']]
)]
#[UniqueEntity('name')]
class Category
{
    #[Groups(['inventory:read:collection', 'category:read:collection'])]
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column(type: 'integer')]
    private $id;

    #[Groups(['inventory:read:collection', 'category:read:collection'])]
    #[ORM\Column(type: 'string', length: 255, unique: true)]
    private $name;

    #[ORM\ManyToMany(targetEntity: Inventory::class, inversedBy: 'categories')]
    private $inventories;

    public function __construct()
    {
        $this->inventories = new ArrayCollection();
    }

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getName(): ?string
    {
        return $this->name;
    }

    public function setName(string $name): self
    {
        $this->name = $name;

        return $this;
    }
```

Apres avoir créé notre bdd à l'aide de la commande `symfony console doctrine:database:create`

Grace à la commande `symfony console make:migration` nous pouvons autogénérer du code SQL que nous pouvons ensuite éxecuter grâce à la commande `symfony console doctrine:database:create` .

A ce moment là la structure de notre base de données et créée mais elle est vide.

Ce que permet symfony c'est aussi de créer des fausses données à l'aide de fixtures pour pouvoir avoir une base de données remplie rapidement lorsque l'on est en développement sans le faire manuellement.

```php
class AppFixtures extends Fixture


{

    public function load(ObjectManager $manager): void
    {

        HistoryTypeFactory::createOne(['name' => 'LOAN']);
        HistoryTypeFactory::createOne(['name' => 'RETURN']);
        HistoryTypeFactory::createOne(['name' => 'BOOKING']);
        HistoryTypeFactory::createOne(['name' => 'CANCEL']);

        UserFactory::createOne(
            ['email' => 'alexlemia13@gmail.com', 'firstname' => 'Alexandre',
        );

        UserFactory::createMany(5);
```

Création API

Api platform est un framework permettant de créer des API bien documentée et respectants les normes OpenAPI.

Par défault les réponses utilisent le format JSON-ld qui est un format qui nous permet d'avoir beaucoup d'information sur une réponse.

Exemple de réponse :

```
"@context": "/api/contexts/Inventory",
"@id": "/api/inventories",
"@type": "hydra:Collection",
"hydra:member": [
  {
    "@id": "/api/inventories/1",
    "@type": "Inventory",
    "id": 1,
    "name": "deleniti",
    "description": "Rodolphe et Emma l'aperçut dans la compagnie pour aller fermer la pharmacie. Il y avait le délire; e
our Rouen. Or, comme le paysage qui nous démontre, en passant, différents.",
    "categories": [],
    "isFavorite": true,
    "isAvailable": false,
    "isBooked": true,
    "waitingPos": 4,
    "rating": {
      "ratingsAvg": "2.2000",
      "ratingsCount": 5,
      "myRating": 3
    },
    "isLoaned": true,
    "image": "/uploads/images/inventory/one-piece-livre-copie-62b5ba66e3c16302464189.jpg"
  }
],
"hydra:totalItems": 50,
"hydra:view": {
  "@id": "/api/inventories?itemsPerPage=1&page=1",
```


Les 5 méthodes utilisées dans API Platform sont :

- Get pour récupérer une ou plusieurs resources. (Status succes: 200)
- Post pour créer une resource. (Status succes : 201)
- Put pour modifier une resource (Status succes : 200)
- Patch pour modifier partiellement une resource. (Status succes : 200)
- Delete pour supprimer une resource (Status succes : 204)

Par défault lorsque nous ajoutons l'annotation (ou attribut) ApiResource nous avons 6 resources créées, une pour chaque resource sauf pour get qui peut renvoyer une ou plusieurs resources.

Api platform nous permet aussi de choisir les champs que nous voulons accepter lors de l'envoies ou lors de la réception grâce aux denormalization_context et normalization_context respectivement.

Pour ce faire nous devons créer des groupes :

```
#[ApiResource(
    normalizationContext: ['groups' => ['user:read:collection']]
```

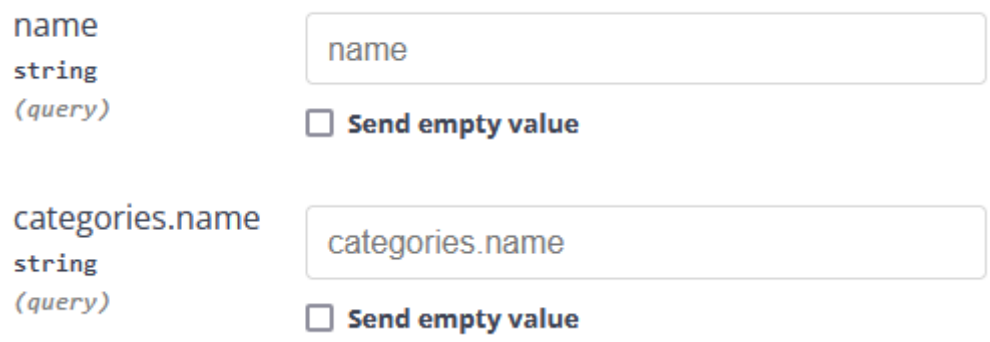
Puis les assigner aux propriétés voulues

```
#[Groups(['user:read:collection'])]
#[ORM\Id]
#[ORM\GeneratedValue]
#[ORM\Column(type: 'integer')]
#[ApiProperty(identifier: true)]
private $id;

#[Groups(['user:read:collection', 'auth:write', 'user:update'])]
#[ORM\Column(type: 'string', length: 180, unique: true)]
#[Assert\Email(groups: ['auth:validate'])]
#[Assert\Length(groups: ['auth:validate'], max: 180)]
#[Assert\NotBlank(groups: ['auth:validate'])]
private $email;
```

Nous pouvons aussi filtrer les requêtes grâce à des filtres

```
#[ApiFilter(SearchFilter::class, properties: ['name' => 'partial', 'categories.name' => 'exact'])]
```



Il existe plusieurs type de filtres comme les search filters ou boolean filters.

Nous pouvons aussi créer des dataproviders qui modifient le comportement par default d'API platform.

Pour cela nous devons implementer des interfaces

```
class InventoryDataProvider implements ContextAwareCollectionDataProviderInterface, ItemDataProviderInterface,
```

Puis nous pouvons modifier l'opération

```
public function getItem(string $resourceClass, $id, string $operationName = null, array $context = [])
{
  /** @var User|null */
  $user = $this->security->getUser();

  $context['has_identifier_converter'] = false;

  if (!$user) {
    throw new UnauthorizedHttpException("Vous n\'êtes pas connecté");
  }

  $inventory = $this->itemDataProvider->getItem($resourceClass,  $id, $operationName, $context);

  $review = $this->inventoryRepository->getReviews($inventory, $user);
  unset($review[0]['id']);

  $inventory->setIsAvailable($this->inventoryRepository->getAvailableCount($inventory) > 0);
  $inventory->setRating($review);
  $inventory->setWaitingPos($this->inventoryRepository->findWaitingPos($inventory, $user)[0]['waitingPos']);
  $inventory->setIsFavorite($user->getFavoritesInventories());
  $inventory->setIsBooked($user->getBookedInventories());
  $inventory->setIsLoaned(!empty($this->inventoryRepository->checkIfUserAlreadyLoanedInventories($inventory, $user)));

  return $inventory;
}
```

Il existe aussi des DataProviders qui nous permettent de modifier la resource avant qu'elle ne soit persistée.

Pour cela il faut encore implémenter une interface

```
class UserDataPersister implements DataPersisterInterface
```

```
public function supports($data): bool
{
  return $data instanceof User;
}

public function persist($data): void
{
  /** @var User $data */
  if ($data->getPlainPassword()) {
    $data->setPassword($this->passwordHasher->hashPassword($data, $data->getPlainPassword()));
  }
  if ($data->getNewPassword()) {
    $data->setPassword($this->passwordHasher->hashPassword($data, $data->getNewPassword()));
  }
  $this->entityManager->persist($data);
  $this->entityManager->flush();
}
```

La methode supports permet de préciser sur quelle entité on veut utiliser le persister et la methode persist sert à écrire nos instructions.

J'ai aussi utilisé les EventListeners de doctrine pour par exemple rajouter une nouvelle entrée dans ma table history lorsqu'il y a une réservation

```php
public function postPersist(Booking $booking)
{
  $history = new History();
  $history->setInventory($booking->getInventory());
  $history->setUser($booking->getUser());
  $history->setDate(DateTimeImmutable::createFromMutable(new DateTime()));
  $history->setHistoryType($this->historyTypeRepository->find(3));
  $this->em->persist($history);
  $this->em->flush();
}
```

Pour cela il faut enregistrer le listener dans l'entité et rajouter deux lignes dans services.yaml

```php
#[ORM\EntityListeners(['App\Doctrine\BookingEventListener'])]
```

```yaml
App\Doctrine\BookingEventListener:
  tags: ["doctrine.orm.entity_listener"]
```

Tests

J’ai aussi fait beaucoup de tests grâce à phpunit.

Je m’en suis surtout servi pour faire des assertions sur mes requêtes pour m’assurer que tout se passe comme prévue.

```php
Run | Show in Test Explorer
public function testGetLoggedInUserReturnsEmail()
{
    $token = $this->getUserToken();
    self::createClient()->request('GET', "/api/users/{$this->getMyUser()->getId()}", [
        'headers' => $this->getAuthorizationHeader($token)
    ]);


    $this->assertJsonContains([
        "@type" => "User",
        "email" => $this->getMyUser()->getEmail()
    ]);
}
```

```php
Run | Show in Test Explorer
public function testGetAllUserReturns401IfNotLoggedIn()
{
    $user = $this->getMyUser();


    self::createClient()->request('GET', "/api/inventories/{$user->getId()}", []);

    $this->assertResponseStatusCodeSame(401);
}
```

```
➜ deepkit-api git:(f/test-refactorisation-tests) ✗ bin/phpunit
PHPUnit 9.5.20 #StandWithUkraine

Testing
.....................................................           54 / 54 (100%)

Time: 00:37.969, Memory: 114.00 MB

OK (54 tests, 90 assertions)
```

Authentification

J'ai créé mon entité User grace au bundle Security qui me permet de créer un utilisateur avec quelques propriétés prédéfinies mais qui me permet aussi d'avoir un hasheur de mot de passe d'utilisateur ainsi que de pouvoir injecter la classe Security lors de certaines requêtes pour récupérer l'utilisateur connecté.

Pour s'enregister j'ai créé une route /api/register que j'ai créé en mettant ceci dans ApiResource

```
'register' => [
    'method' => 'POST',
    'path' => '/register',
    'validation_groups' => ['auth:validate'],
    'openapi_context' => ['tags' => ['Auth']],
    'denormalization_context' => ['groups' => ['auth:write']],
],
```

Pour la connexion j'utilise le bundle lexik/lexikJWTAuthenticationBundle ainsi que gesdinet/jwt-refresh-token-bundle.

Après avoir faire configuré le fichier security.yaml , nous avons un endpoint /api/token et un autre /api/token/refresh qui nous permettent d'obtenir un token jwt qui servira de preuve d'authentification.

```
firewalls:
  dev:
    pattern: ^/_(profiler|wdt)
    security: false
  main:
    stateless: true
    provider: app_user_provider
    entry_point: jwt
    json_login:
      check_path: /api/token
      username_path: email
      password_path: password
      success_handler: lexik_jwt_authentication.handler.authentication_success
      failure_handler: lexik_jwt_authentication.handler.authentication_failure
    jwt: ~
    refresh_jwt:
      check_path: /api/token/refresh
```

Gestion des droits

Le bundle security permet aussi de gérer les droits.

La plupart des routes de mon API nécéssitent d'être connecté, d'autres nécéssitent d'être un administrateur ou ne renvoient pas les même données en fonction de leur rôle.

Nous pouvons configurer les droits dans le fichier security.yaml

```yaml
access_control:
  - { path: ^/docs, roles: PUBLIC_ACCESS } # Allows accessing the Swagger UI
  - { path: ^/api/(login|token/refresh), roles: PUBLIC_ACCESS }
  - { path: ^/api/register$, roles: PUBLIC_ACCESS }
  - { path: ^/api$, roles: PUBLIC_ACCESS }
  - { path: ^/api/new-passwords, roles: PUBLIC_ACCESS }
  - { path: ^/api/connect$, roles: PUBLIC_ACCESS }
  - { path: ^/api, roles: IS_AUTHENTICATED_FULLY }
```

Ou dans API Resource

```
'get' => ["security" => "is_granted('ROLE_ADMIN') or is_granted('IS_OWNER', object)"],
```

Nous pouvons aussi créer des voters pour créer des authorization customisées

```php
class UserVoter extends Voter
{
  protected function supports($attribute, $subject): bool
  {
    $supportsAttribute = in_array($attribute, ['IS_OWNER']);
    $supportsSubject = $subject instanceof User;

    return $supportsAttribute && $supportsSubject;
  }
```

```php
protected function voteOnAttribute($attribute, $subject, TokenInterface $token): bool
{
  $user = $token->getUser();

  if (!$user instanceof User) {
    return false;
  }

  switch ($attribute) {
    case 'IS_OWNER':
      if ($user === $subject) {
        return true;
      }
  }

  return false;
}
```

Upload de fichier

Pour envoyer des fichier j'ai choisi le bundler vich/uploader-bundle.

Il s'occupe de récupérer et de valider les fichiers envoyer en formData.

Pour cela il faut configurer le fichier vich_uploader.yaml et rajouter l'annotation @Vich/Uploadable dans l'entité et ajouter l'annotation @Vich/UploadableField au dessus du champs qui reçoit le fichier.

```php
/**
 * @Vich\UploadableField(mapping="user_photo", fileNameProperty="photoFile")
 * @var File
 */
#[Assert\File(mimeTypes: ["image/png", "image/jpeg"], maxSize: '50M', groups: ['user:update:validate'])]
private $photoFile;
```

Le problème c'est que nous ne pouvons pas envoyer de fichier dans la base de données donc j'utilise un normalizer pour remplacer le fichier par son chemin lors de l'envoie dans la base de donnée

```php
final class InventoryImageSerialize implements ContextAwareNormalizerInterface, NormalizerAwareInterface
{
    use NormalizerAwareTrait;

    private const ALREADY_CALLED = 'USER_PHOTO_NORMALIZER_ALREADY_CALLED';

    public function __construct(private StorageInterface $storage, private EntityManagerInterface $em)
    {
    }

    public function normalize($object, ?string $format = null, array $context = []): array|string|int|float|bool|\ArrayObjec
    {
        $context[self::ALREADY_CALLED] = true;

        if ($this->storage->resolveUri($object, 'imageFile')) {
            $objectUrl = str_replace("\\", '/', $this->storage->resolveUri($object, 'imageFile'));
            $object->setPhoto(str_replace('/public', '', $objectUrl));
        }

        $this->em->flush();


        return $this->normalizer->normalize($object, $format, $context);
    }


    public function supportsNormalization($data, ?string $format = null, array $context = []): bool
    {
        if (isset($context[self::ALREADY_CALLED])) {
            return false;
        }

        return $data instanceof Inventory;
    }
}
```

Sécurité

La sécurité a une très grande importance dans une API car elle communique avec notre base de données et nous ne voulons pas par exemple qu'un utilisateur se fasse voler son mot de passe ou bien qu'un utilisateur se fasse passer pour un admin…

Par chance API Platform, Symfony et doctrine s'occupent de beaucoup de choses pour nous.

- Pour hasher les mots de passes des utilisateur il y a le UserPasswordHasher de Security
- Pour les injections SQL doctrine nous permet d'ajouter des paramêtres

```
= $this->createQueryBuilder('i')
->leftJoin('i.inventoryElements', 'ie')
->leftJoin('ie.loan', 'l')
->andWhere('i.id = :inventory')
->andWhere('l.user = :user')
->setParameter('inventory', $inventory)
->setParameter('user', $user)
->getQuery()
->setMaxResults(1)
->getOneOrNullResult();
```

- Le bundle Security s'occupe des problemes d'access control
- VichUploader nous protège contre les attaques de File Upload grace aux validations
- Le token JWT a une durée assez courte pour empécher un éventuel voleur de JWT de rester connecté trop longtemps. Le refresh token permet à l'utilisateur de regénérer des token dans devoir se reconnecter.

Exemple de Problématique rencontrée

Pour une commande SQL dans le querybuilder de doctrine, j'avais besoin d'accéder à l'id de la relation d'une entité

```
= $this->createQueryBuilder('i')
>leftJoin('i.inventoryElements', 'ie')
>leftJoin('i.bookings', 'b')
>select("(count(DISTINCT ie.id) - count(DISTINCT ie.loan) - count(DISTINCT(CONCAT(IDENTITY(b.inventory) , ' ', IDENTITY(b.user))))) availableCount")
>addSelect('i.id')
>andWhere('i.id in (:inventories)')
>addGroupBy('i.id')
>setParameter('inventories', $inventories)
>getQuery()
>getResult();
```

La solution que j'ai trouvé était d'utiliser IDENTITY.

Exemple de recherche anglophone

Pour résoudre mon problème evoqué précédemment j'ai du chercher sur google.

You may also want to look at the IDENTITY() function (Doctrine version >2.2).

Example:

```
SELECT IDENTITY(t.User) AS user_id from Table
```

Should return:

```
[ ['user_id' => 1], ['user_id' => 2], ... ]
```

Traduction :

Tu devrais peut être regarder la fonction « Identity() » (version de Doctrine > 2.2)

Exemple :

_____

Devrait renvoyer :

_____

Développement du frontend:

Modules Principaux

Mon application mobile utilise react native ainsi que plusieurs modules.

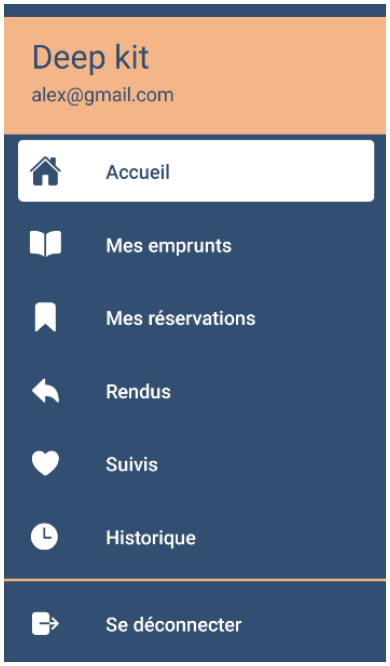
Parmi eux le principal est react navigation qui me permet de faire la navigation dans mon application.

Il permet de créer des Stack qui sont un peu comme une navigation sur un navigateur web, des bottoms tabs qui sont des onglets en bat de l'écran et un drawer.

L'un des autres points forts de react navigation est qu'il nous permet d'envoyer des données aux autres pages à la manière des props dans react native.

Exemple de bottom tab :

Exemple de drawer :



Pour créer la navigation il faut d'abord créer un provider global qui entoure notre app

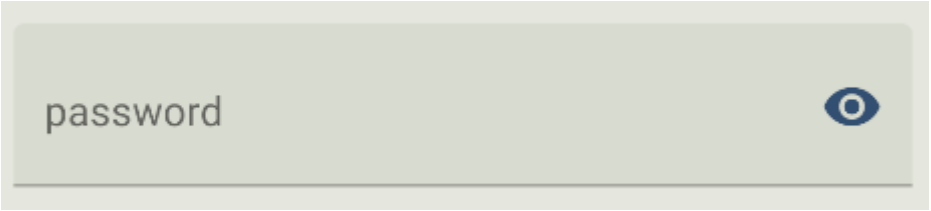
```
export default function App() {
  return (
    <PaperProvider theme={CombinedTheme}>
      <NavigationProvider theme={CombinedTheme}>
        <StatusBar style="light" />
        <MainNavigation />
      </NavigationProvider>
    </PaperProvider>
  );
}
```

Le second module le plus important dans mon application est react native paper.

React native étant un framework basé sur les composants, quoi de mieux qu'un module qui nous propose des composants très utiles et qui respectent le material design de google.

Ainsi je n'ai pas à réinventer la roue, le design de mon application est cohérent et je gagne un peu de temps de développement.

Exemple d'input de react native paper nous permettant de rajouter une icone:



Une autre fonctionnalité très importante dans l'application est la connexion entre l'API et l'application à l'aide de fetch qui nous permet d'envoyer et de recevoir des requêtes.

C'est grâce à fetch que nous pouvons nous connecter et récupérer notre jwt mais aussi grâce à lui que nous pouvons récupérer l'inventaire de tous les objets du serveur.

```
useEffect(() => {
  (async () => {
    try {
      const response = await fetch(routes(currentServer.url, "category"), {
        headers: {
          authorization: "Bearer " + currentServer.token,
          "Content-Type": "application/json",
        },
      });

      const data = await response.json();

      const fetchedCategories = data["hydra:member"];
```

Authentification

L'authentification consiste à récupérer un jwt lors de la connexion et le stocker dans le téléphone à l'aide d'expo-secure-store

```
const body = {
  email,
  password,
};

try {
  const response = await fetch(routes(url, "token"), {
    method: "POST",
    body: JSON.stringify(body),
    headers: {
      "Content-Type": "application/json",
    },
  });

  const data = await response.json();

  console.log(data);

  if (data.code === 401) {
    setGlobalErrors(data.message);
    return;
  }

  if (data["@type"] === "ConstraintViolationList") {
    data.violations.forEach((violation) => {
      errorsObj[violation.propertyPath] = violation.message;
    });
  }

  if (response.ok) {
    const token = data.token;
    const newServers = servers.map((server) =>
      server.id === serverId ? { ...server, token } : server
    );

    await SecureStore.setItemAsync("servers", JSON.stringify(newServers));
    setServers(newServers);
    if (navigation) {
      navigation.navigate("ServerSelection");
    }
  }
```

Ensuite sur la page d'accueil, lorsque l'on clique sur le bouton login on envoie une requête pour savoir si le serveur existe et on envoie le jwt.

Si le jwt correspond à un utilisateur alors nous sommes connecté sur la page inventaire.


```
const connectMe = async (server, navigation) => {
  const pushToken = await (await Notifications.getExpoPushTokenAsync()).data;

  try {
    const response = await fetch(routes(server.url, "connectMe"), {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        authorization: `Bearer ${server.token}`,
      },
      body: JSON.stringify({ deviceIdentifier: pushToken }),
    });

    const data = await response.json();

    if (!navigation) return data.data.user;
    if (data?.message === "Vous êtes bien sur Deepkit API") {
      navigation.navigate("Drawer", {
        currentServer: server,
        user: data.data.user,
      });
      return true;
```

Validations

Lors de mes requêtes il peut arriver qu'il y ai des problêmes de validations.

Si c'est le cas la réponse renvoyée par ma requêtes contiendras le code 422

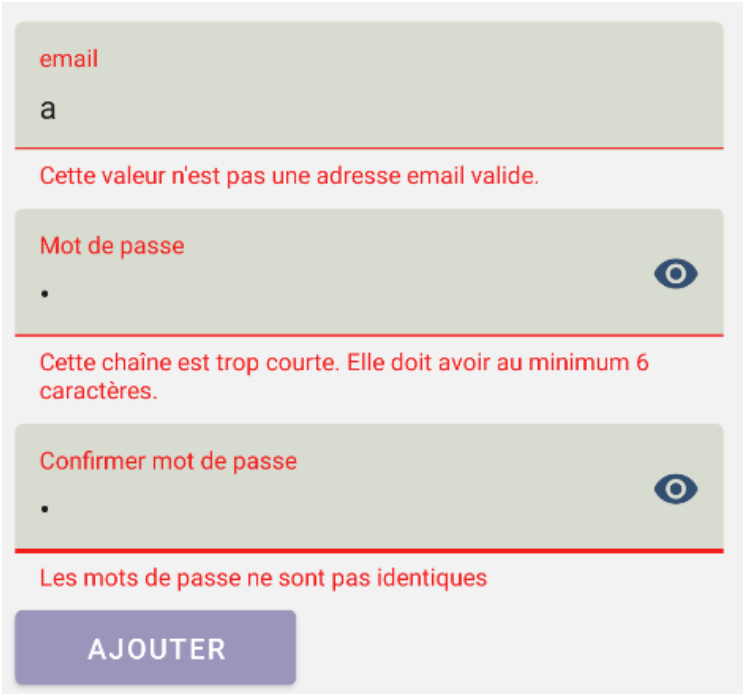
Dans ce cas là, je récupère les messages d'erreurs et les mets dans un objet.

```
if (response.code === 422) {
  data.violations.forEach((violation) => {
    errorsObj[violation.propertyPath] = violation.message;
  });
}
```

Puis j'ai créé un composant input custom qui affiche les erreurs s'il y en a

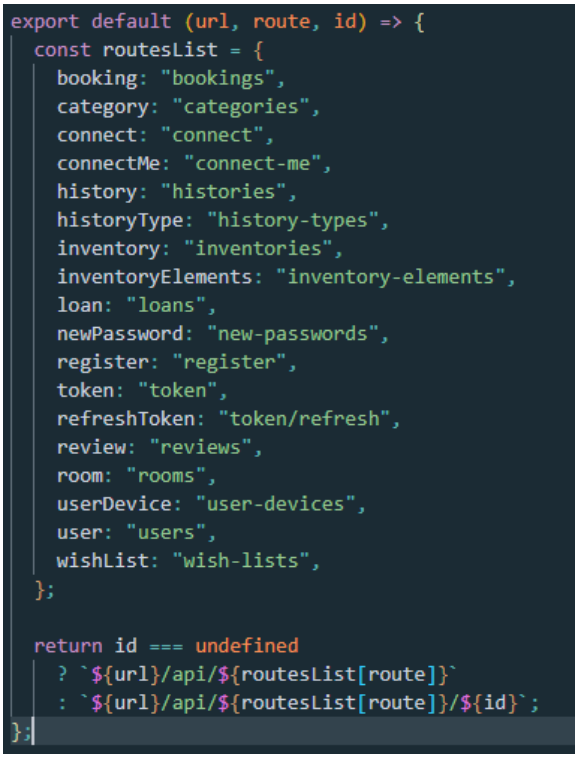
```
const InputWithError = (props) => {
  return (
    <View style={styles.textInput}>
      <TextInput
        label={
          props.label[0].toUpperCase() + props.label.toLowerCase().slice(1)
        }
        onChangeText={(input) => props.cb(input)}
        error={props.errors[props.name]}
        {...props}
      />
      <HelperText type="error" visible={props.errors[props.name]}>
        {props.errors[props.name]}
      </HelperText>
    </View>
  );
};
```

Ce qui donne


email
a
Cette valeur n'est pas une adresse email valide.
Mot de passe
.
Cette chaîne est trop courte. Elle doit avoir au minimum 6 caractères.
Confirmer mot de passe
.
Les mots de passe ne sont pas identiques
AJOUTER

Routes

J’ai aussi créé un fichier pour lister mes routes et m’assurer de ne pas faire d’erreur.



```
export default (url, route, id) => {
  const routesList = {
    booking: "bookings",
    category: "categories",
    connect: "connect",
    connectMe: "connect-me",
    history: "histories",
    historyType: "history-types",
    inventory: "inventories",
    inventoryElements: "inventory-elements",
    loan: "loans",
    newPassword: "new-passwords",
    register: "register",
    token: "token",
    refreshToken: "token/refresh",
    review: "reviews",
    room: "rooms",
    userDevice: "user-devices",
    user: "users",
    wishList: "wish-lists",
  };

  return id === undefined
    ? `${url}/api/${routesList[route]}`
    : `${url}/api/${routesList[route]}/${id}`;
};
```

Conclusion :

J’ai énormément appris et acquis d’expérience durant ce projet.

C’est le plus gros projet que j’ai réalisé jusqu’à présent donc il m’a fallu bien m’organisé et c’est ce que j’ai fait

Pour conclure, je suis très fier de ce projet qui aura été très difficile et qui m’aura demandé beaucoup de temps et j’en ressors grandi.