

# CQP Développeur Nouvelles Technologies

## Évaluation bloc 2 : développement

### Rapport d'activité

DUMON Gaël



Introduction.....	3
A - Présentation générale du système.....	4
1. Architecture globale.....	4
a) Système général.....	4
b) Hemengo API REST (Express).....	5
c) Hemengo App (Ionic/Angular).....	6
d) Hemengo Scanner (Android natif).....	7
2. Persistance des données.....	8
a) Justification de la technologie utilisée.....	8
b) Rappel de la modélisation des données (MLD).....	9
c) Exemples de requêtes et de définition de modèle.....	10
3. Back-end.....	12
a) Documentation.....	12
b) Exemples de réponses.....	13
c) Postman.....	15
4. Front-end.....	16
a) Quelques vues.....	16
5. Mobile.....	18
a) Partie Ionic.....	18
b) Partie Android native.....	19
B – Validation des critères.....	20
1. Algorithmique.....	20
2. HTML – CSS.....	21
3. Interactions client – serveur.....	22
4. Programmation Javascript et Express.....	23
5. Programmation Typescript.....	24
6. Programmation Java et Android natif.....	25
7. Mobile Ionic.....	26
8. Angular.....	27
9. Les tests – API & App Ionic/Angular.....	28
10. Gestion de versions (Git).....	29
11. Docker .....	30

## INTRODUCTION

Pour ce bloc n°2 concernant le développement d'un projet, j'ai décidé de rester sur le sujet n°3 choisi lors du bloc n°1 (modélisation) : **Hemengo Distrib.**



L'idée de l'application est de proposer un moyen de passer des commandes de produits du pays basque, à venir retirer sur un ensemble de distributeurs automatiques (de type casiers à déverrouiller) sur l'ensemble du territoire local. Ces distributeurs sont remplis de produits de producteurs locaux uniquement, par intermédiaire de l'entreprise Hemengo. Ci-contre: le logo du projet.

Pour mieux se représenter à quoi un distributeur Hemengo pourrait ressembler, j'ai créé cette maquette :



Par nécessité de concision, ce rapport va se concentrer sur une fonctionnalité principale : la **récupération d'une commande** par un client à un distributeur donné. Lors de cette action, l'utilisateur va se présenter physiquement devant le distributeur auquel il a passé commande, puis sur notre application mobile il clic sur « Récupérer commande », à ce moment, l'application lui demande de **scanner le QR Code** présent sur le distributeur en guise d'identification de la machine. Dans ce QR Code est stocké le **uuid** (universal unique identifier) du distributeur en question.

Si le uuid du QR Code scanné = le uuid du distributeur associé à la commande, alors la commande peut être débloquée.

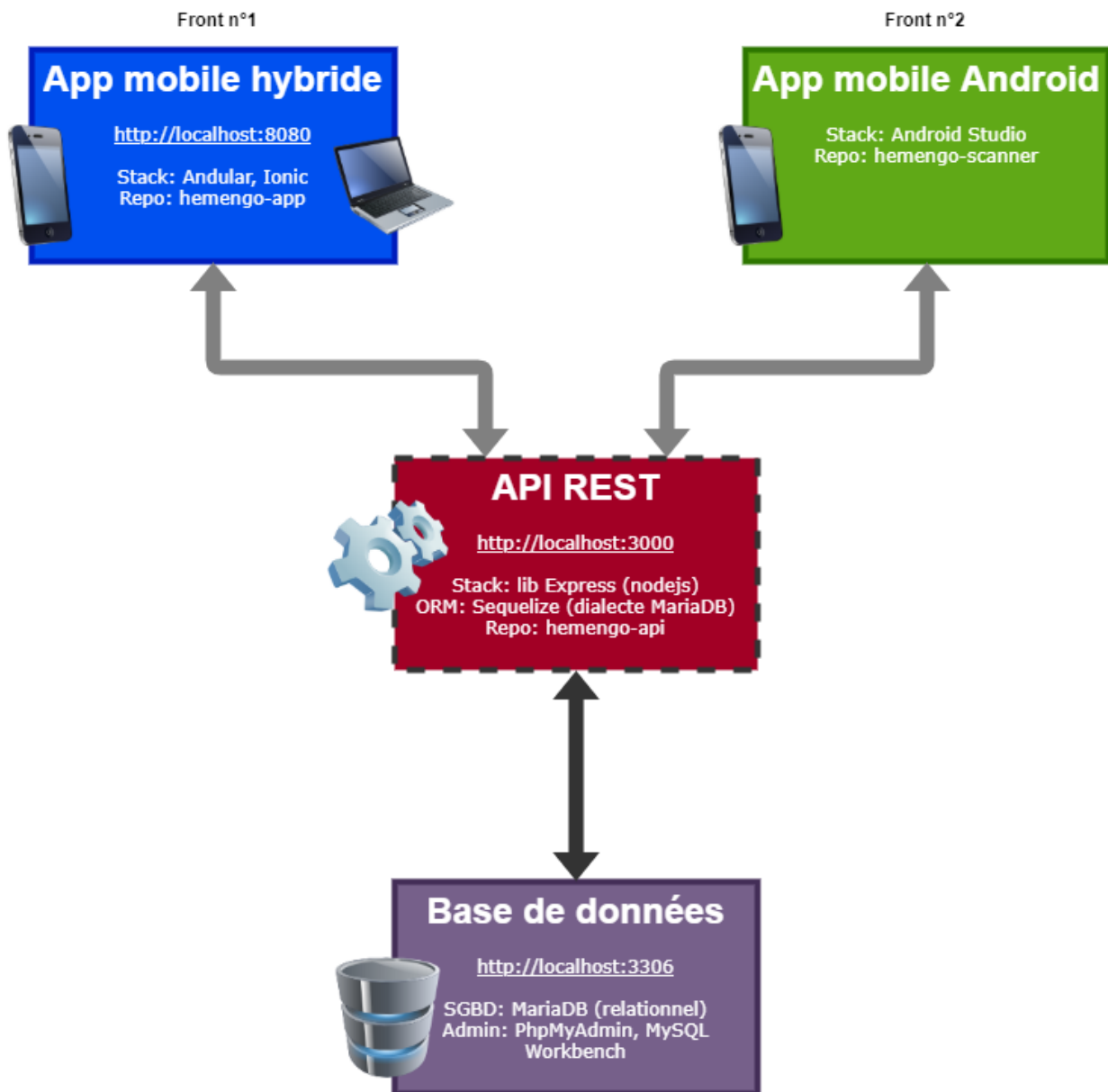
Le déblocage d'une commande se manifeste par le déverrouillage de tous les casiers dans lesquels il y a les produits de la commande.

Le distributeur est analogue à une grille ou un tableau, où chaque casier = une cellule. Chaque casier est associé à une référence de type COLONNE LIGNE, où COLONNE = une lettre et LIGNE = un chiffre. Ainsi la réf A1 = 1ere ligne – 1ere colonne, A2 = 2eme ligne – 1ere colonne etc. Par défaut, le maximum de lignes est de 6 et le maximum de colonnes est de 5 (de A à E).

Il est important de noter que ce projet (divisés en 3 sous projets) en est au stade de « Proof Of Concept ».

### 1. Architecture globale

#### a) Système général



L'ensemble des projets ont été versionnés avec **Git** tout au long du développement (même ce rapport), ainsi, **Github** a été choisi pour héberger les répertoires distants de chaque projet :

- le repository Github du projet **Hemengo App** est disponible ici : <https://github.com/hyperdestru/hemengo-app>
- le repository Github du projet **Hemengo API** est disponible ici : <https://github.com/hyperdestru/hemengo-api>
- le repository Github du projet **Hemengo Scanner** est disponible ici : <https://github.com/hyperdestru/hemengo-scanner>

## b) Hemengo API (Express)

<div> <div>HEMENG-API</div> <div> <div>app</div> <div> <div>helpers</div> <div> <div>util.js</div> <div>jwt</div> <div> <div>check.js</div> <div>models</div> <div>routes</div> <div>db.config.js</div> <div>server.js</div> <div>node_modules</div> <div>public</div> <div>spec</div> <div>support</div> <div> <div>jasmine.json</div> <div>order.spec.js</div> <div>.dockerignore</div> <div>.env</div> <div>.gitignore</div> <div>docker-compose.yml</div> <div>Dockerfile</div> <div>Jenkinsfile</div> <div>package-lock.json</div> <div>package.json</div> </div> </div> </div> </div> </div> </div>	<div> <div> <div>jwt/check.js</div> <div>Middleware qui vérifie le token <b>JWT</b> créé dans les routes d'authentification. Passé à la plupart des routes de l'API afin d'assurer une certaine sécurité.</div> </div> <div> <div>db.config.js</div> <div>Construit et exporte une instance de <b>Sequelize</b> en lui passant les identifiants de la BDD contenu dans le fichier .env (non versionné). Lance la synchronisation entre modèles et BDD.</div> </div> <div> <div>server.js</div> <div>Instancie l'application, appelle toutes les routes et middlewares nécessaires, se connecte à la BDD via l'instance Sequelize et lance le serveur.</div> </div> <div> <div>spec/</div> <div>Fichiers de tests (lib <b>Jasmine</b>).</div> </div> <div> <div>models/</div> <div>Modèles Sequelize. Un modèle = une table en BDD. <b>Il n'y a pas besoin de créer des modèles pour les tables d'associations</b>, elle sont automatiquement créées et gérées par Sequelize lorsque <i>belongsToMany()</i> est appelée entre deux modèles (cf. models/index.js).</div> </div> <div> <div>models/index.js</div> <div>Importe tous les modèles, y accrochent les associations (relations 1-N, N-N) grâce aux méthodes d'associations de Sequelize (<i>belongsTo</i>, <i>belongsToMany</i>...) et exporte les modèles avec leurs associations.</div> </div> <div> <div>routes/</div> <div>Routes et contrôleurs pour chaque ressource.</div> </div> </div>	<div> <div>HEMENG-API</div> <div> <div>models</div> <div> <div>City.js</div> <div>index.js</div> <div>Locker.js</div> <div>MatrixElement.js</div> <div>Order.js</div> <div>Producer.js</div> <div>Product.js</div> <div>ProductCategory.js</div> <div>Status.js</div> <div>User.js</div> <div>VendingMachine.js</div> </div> <div>routes</div> <div> <div>auth.js</div> <div>city.js</div> <div>index.js</div> <div>locker.js</div> <div>matrixElement.js</div> <div>order.js</div> <div>producer.js</div> <div>product.js</div> <div>productCategory.js</div> <div>status.js</div> </div> </div> </div>
---	---	--

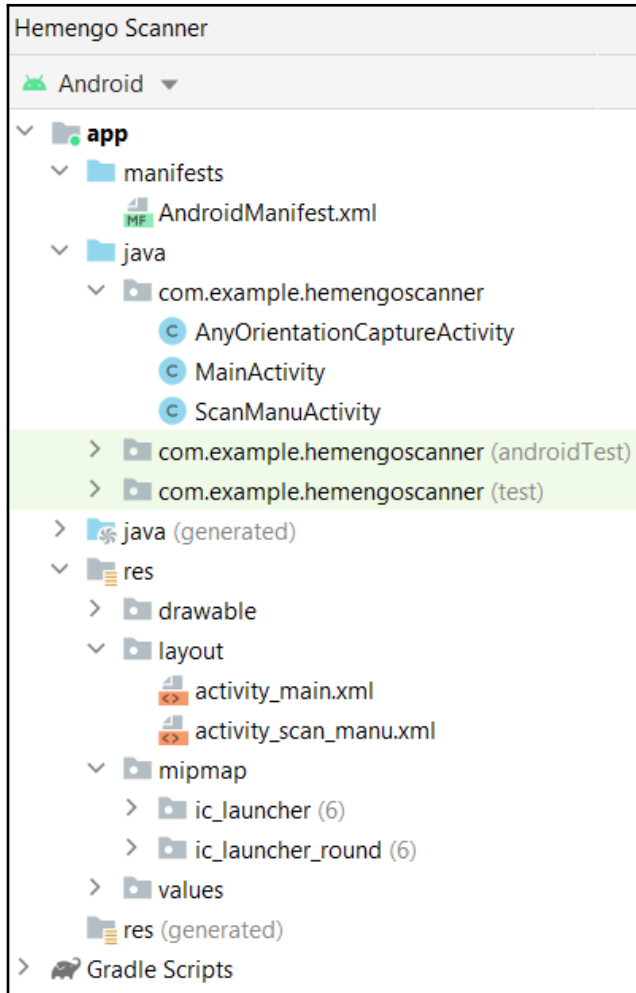
Chaque fichier instancie le routeur Express, appelle au besoin ses méthodes get, post, patch.. et réalise le fetching des ressources dans leurs callback.

### c) Hemengo App (mobile hybride Ionic/Angular)

▼ HEMENGO-APP	
> android	<u>android/</u>
> e2e	Fichiers Android créés lors d'un build vers la plateforme Android (commandes <i>ionic cap add android</i> puis <i>ionic cap copy</i> ).
> node_modules	
> resources	<u>components/</u>
▼ src	Composant du menu écran-séparé (split pane), composant du layout menu burger pour le responsive. Les composants sont ré-utilisables au sein de pages et d'autres composants parents, ils permettent un degré d'abstraction intéressant pour éviter le code dupliqué par exemple.
▼ app	<u>helpers/</u>
> components	Intercepteur de token, guards de navigation, utilitaires.
> helpers	
> interfaces	<u>interfaces/</u>
▼ pages	Interfaces aidant notamment (mais pas que) pour le typage des retours des services.
> demo	
> login	<u>pages/</u>
> logout	Pages entières Angular, implémentent les services, les interfaces, les souscriptions aux observables retournés par les services, les styles, les structures HTML à l'aide des composants Ionic.
> not-found	
> profile	
> register	
> splash	
> services	<u>services/</u>
TS app-routing.module.ts	Environ un service par ressource. Instancient la classe HttpClient d'Angular.
<> app.component.html	Contiennent les méthodes de requêtes à l'API Hemengo, où chacune de ses méthodes retourne un <b>Observable RxJs</b> typé grâce aux interfaces.
🔗 app.component.scss	
TS app.component.spec.ts	
TS app.component.ts	

#### d) Hemengo Scanner (mobile Android natif)

Hemengo Scanner est une application de scan de QR Code. Elle s'insère dans le parcours utilisateur au moment de récupérer la commande au distributeur.



##### MainActivity.java

Activité (écran) principale, sur laquelle nous arrivons lorsque nous lançons l'application. Responsable de toute la logique derrière cette activité. Hérite de *AppCompatActivity* et implémente l'interface *View.OnClickListener*. Attache les listeners aux boutons. Instancie la classe *IntentIntegrator* de la lib **Zxing** utilisée pour implémenter le scanner. Configure, lance puis traite le résultat renvoyé par le celui ci.

##### AnyOrientationCaptureActivity.java

Par défaut l'orientation de la caméra lancée par Zxing est en paysage. Pour obtenir une orientation verticale comme voulu, cette activité vide est créée, puis dans le *AndroidManifest.xml* elle est déclarée avec une propriété *screenOrientation* valant *fullSensor* (vertical). Ensuite on peut dire à notre instance de l'*IntentIntegrator* d'utiliser cette activité lors de la capture (lorsque la caméra est ouverte) grâce à la méthode *setCaptureActivity*.

##### ScanManuActivity.java

Il fallait une alternative si le QR Code était absent ou plus visible sur la machine. Cette activité permet la saisie manuelle d'un id présent ailleurs sur le distributeur, pour pouvoir avancer dans le parcours utilisateur.

##### layout/

Les vues des différentes activités. *activity\_main.xml* consiste en un logo Hemengo Distrib suivi d'un bouton de lancement de capture.

## 2. Persistance des données

### a) Justification de la technologie utilisée

Mon choix s'est porté sur un système relationnel SQL avec MariaDB en SGBD.

Les bases de données relationnelles assurent une bonne intégrité des données (elles suivent le modèle ACID\*), dans un contexte d'applicatif incluant des commandes et des paiements cela m'a semblé judicieux.

Cela m'a également paru intéressant d'un point de vue académique/professionnel : apprendre un **ORM** relationnel (ici **Sequelize**).

\*Modèle A.C.I.D :

**Atomique** → Une transaction réussie complètement ou échoue complètement. L'état de la base de données reste inchangé dans le 2eme cas.

**Correct** → Les données écrites dans la base suivent strictement les règles construites à la création de celle ci : contraintes de clés, suppressions en cascades, déclencheurs... Chaque transaction est donc valide.

**Isolé** → Plusieurs transactions lisant et écrivant la base en même temps (= concurrentielles) laissent la base dans le même état que si les transactions s'étaient effectuées en séquence (l'une après l'autre).

**Durable** → La base est stockée en mémoire non-volatile (SSD, HDD...).

*En anglais : **Atomicity – Correctness – Isolation – Durability***

La structure de la base de données **hemengo\_distrib** dans l'outil d'administration PhpMyAdmin (j'ai également utilisé MySQL Workbench alternativement) :

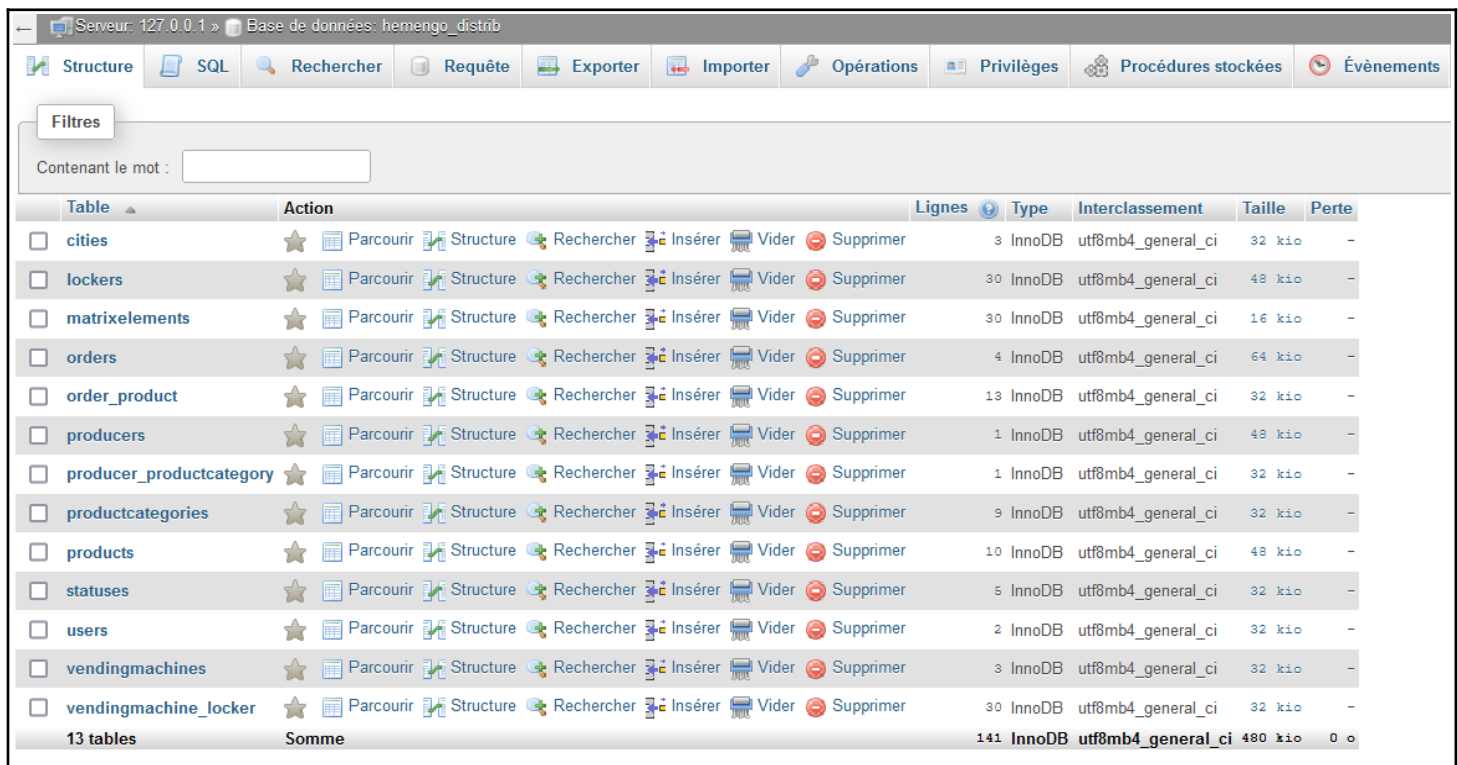
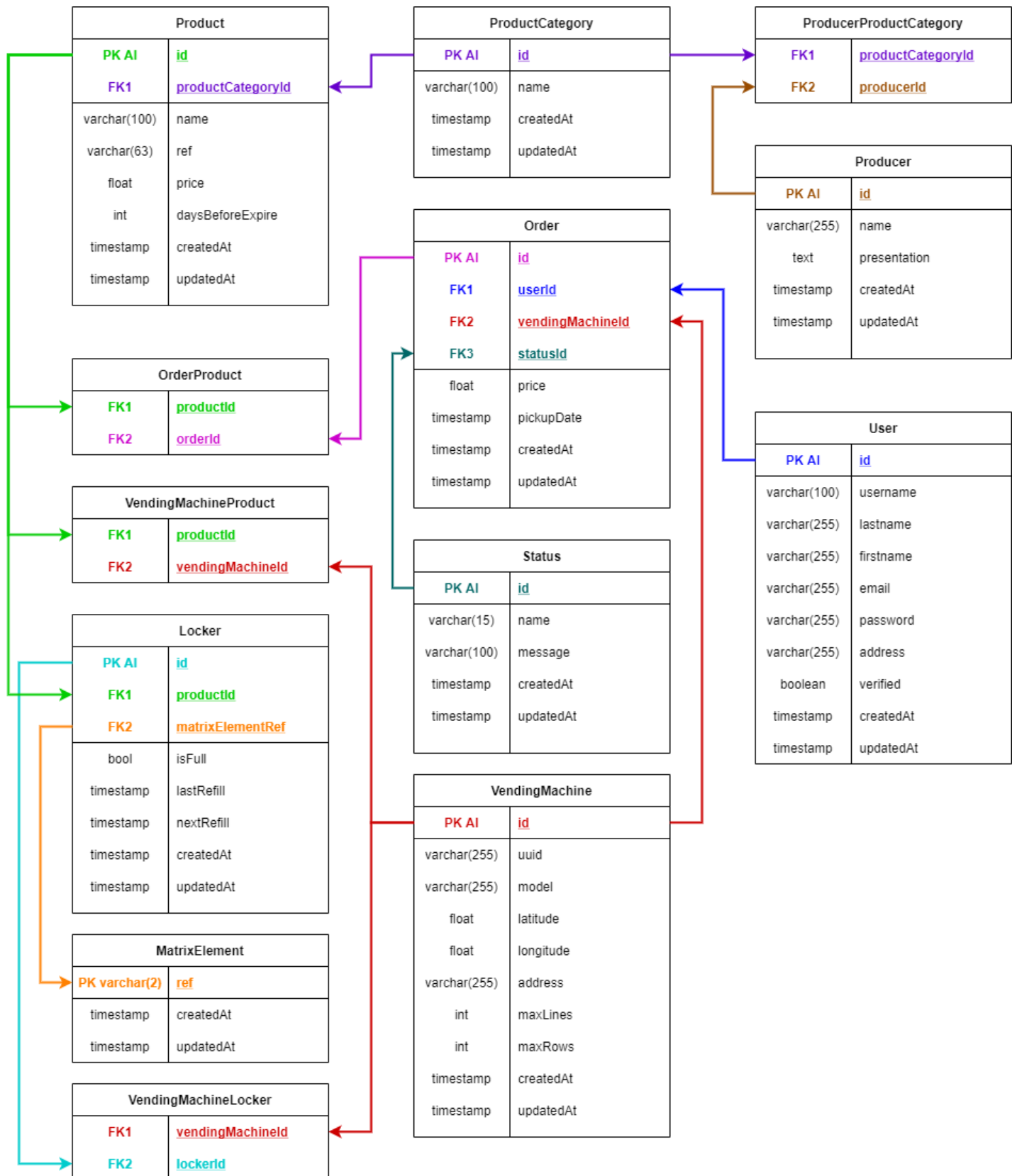


Table	Action	Lignes	Type	Interclassement	Taille	Perte
<input type="checkbox"/> cities	★ Parcourir Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8mb4_general_ci	32 kio	-
<input type="checkbox"/> lockers	★ Parcourir Structure Rechercher Insérer Vider Supprimer	30	InnoDB	utf8mb4_general_ci	48 kio	-
<input type="checkbox"/> matricelements	★ Parcourir Structure Rechercher Insérer Vider Supprimer	30	InnoDB	utf8mb4_general_ci	16 kio	-
<input type="checkbox"/> orders	★ Parcourir Structure Rechercher Insérer Vider Supprimer	4	InnoDB	utf8mb4_general_ci	64 kio	-
<input type="checkbox"/> order_product	★ Parcourir Structure Rechercher Insérer Vider Supprimer	13	InnoDB	utf8mb4_general_ci	32 kio	-
<input type="checkbox"/> producers	★ Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	48 kio	-
<input type="checkbox"/> producer_productcategory	★ Parcourir Structure Rechercher Insérer Vider Supprimer	1	InnoDB	utf8mb4_general_ci	32 kio	-
<input type="checkbox"/> productcategories	★ Parcourir Structure Rechercher Insérer Vider Supprimer	9	InnoDB	utf8mb4_general_ci	32 kio	-
<input type="checkbox"/> products	★ Parcourir Structure Rechercher Insérer Vider Supprimer	10	InnoDB	utf8mb4_general_ci	48 kio	-
<input type="checkbox"/> statuses	★ Parcourir Structure Rechercher Insérer Vider Supprimer	5	InnoDB	utf8mb4_general_ci	32 kio	-
<input type="checkbox"/> users	★ Parcourir Structure Rechercher Insérer Vider Supprimer	2	InnoDB	utf8mb4_general_ci	32 kio	-
<input type="checkbox"/> vendingmachines	★ Parcourir Structure Rechercher Insérer Vider Supprimer	3	InnoDB	utf8mb4_general_ci	32 kio	-
<input type="checkbox"/> vendingmachine_locker	★ Parcourir Structure Rechercher Insérer Vider Supprimer	30	InnoDB	utf8mb4_general_ci	32 kio	-
13 tables	Somme	141	InnoDB	utf8mb4_general_ci	480 kio	0 o



b) Rappel du MLD (Modèle Logique de Données)



### c) Exemples de requêtes et de définition de modèle

Fichier : routes/order.js | route : GET order/user/:id/archive

```
1  models.Order.findAll({
2    where: {
3      UserId: userId,
4      StatusId: {
5        [Op.or]: [2, 4, 5]
6      }
7    },
8    raw: true
9  })
```

Retourne toutes les commandes d'un utilisateur qui sont en statut 2, 4 ou 5 (commandes archivées/annulées/récupérées).

Équivaut par exemple à :

```
SELECT o.* FROM orders o WHERE o.userId = userId
AND (o.statusId = 2 OR o.statusId = 4 OR o.statusId = 5);
```

Ligne 8 - La propriété *raw* lorsqu'elle est set à *true*, renverra une réponse en données brutes c'est à dire sans toutes les méthodes et propriétés Sequelize

qu'il pourrai y avoir sur le modèle. Donc dans le cas où on voudrait réaliser une requête d'association via les méthodes spéciales d'associations Sequelize une fois la réponse obtenue, il faudrait supprimer la ligne 8 pour avoir accès aux dites méthodes.

Fichier : routes/order.js | route : POST /order

```
1  models.Product.sum('price', {
2    where: { id: products }
3  }).then((calculatedPrice) => {
4    models.Order.create({
5      UserId,
6      StatusId,
7      VendingMachineId,
8      pickupDate,
9      price: calculatedPrice
10   }).then(order => {
11     order.addProducts(products).then(() => {
12       res.status(200).json({
13         message: "order created"
14       })
15     })
16   }).catch(err => {
17     res.status(500).json({
18       message: "unable to create order",
19       error: err
20     })
21   })
22 })
```

Création d'une commande. *products* est un array d'ids de produits que l'on passe à la route en payload.

Ligne 1 - l'utilitaire *sum()* sur le champ *price* va calculer la somme des prix de chaque produits dont les ids sont dans *products*, on passe ce prix total à la 1ere méthode *then* de la promesse. Remarquez qu'on ne set pas la propriété *raw* de l'objet de config à *true*.

Ligne 2 - Sequelize nous facilite la vie ici, on peut passer directement un tableau à la propriété *id* sans utiliser l'opérateur *in*, ainsi la version « longue » serait → `id: { [Op.in]: [1,2,3] }`

Ligne 4 – création d'une ressource *order*.

Ligne 11 – la méthode *create* nous

retourne la ressource nouvellement créée en base, cette réponse n'est pas brute elle a donc les méthodes/mixins/propriétés du modèle attachées. Parmi elles la méthode *addProducts*. Cette méthode existe parce qu'il y a une association N-N entre les modèles Order et Product : dans models/index.js ceci est déclaré :

`Order.belongsToMany(Product, { through: "order_product" })` = Création d'une **table de jointure** *order\_product* en BDD.

Ainsi, *addProducts* nous aide à peupler facilement cette table *order\_product*, avec l'id de la ressource *order* insérée et l'id de chaque produit (contenus dans l'array *products*). Sur le modèle *Order* il existe aussi les méthodes spéciales *getProducts*, *removeProducts*, *hasProducts*... Ces méthodes varient en fonction du type d'association créée (*hasOne*, *belongsTo*, *belongsToMany*, *hasMany*...).

Ma source de documentation pour ce sujet : <https://sequelize.org/v6/manual/assocs.html#special-methods-mixins-added-to-instances>

Fichier : *models/City.js*

```
1  const City = db.define('City', {
2    id: {
3      type: DataTypes.INTEGER(11),
4      primaryKey: true,
5      autoIncrement: true
6    },
7    name: {
8      type: DataTypes.STRING(255),
9      allowNull: false
10   },
11   postalCode: {
12     type: DataTypes.STRING(6),
13     allowNull: false
14   },
15   inseeCode: {
16     type: DataTypes.STRING(6),
17     allowNull: false
18   }
19 }, {
20   {
21     paranoid: true,
22     indexes: [{ unique: true, fields: ["name"] }]
23   }
24 })
```

Création du modèle *City* grâce à la méthode *define* de l'instance *Sequelize* importée depuis *app/db.config.js*.

Lors de la synchronisation (méthode *sync*), *Sequelize* créera une table *city* en BDD, avec en clé primaire auto-incrémentée le champ *id*, puis des champs *name*, *postalCode* et *inseeCode* (code commune unique d'une ville), tous doivent être non null (*allowNull : false*).

**!/ Attention bug Sequelize :** au début je définissais mes index unique en passant *unique : true*, directement dans l'objet du champ en question, par exemple :

```
name : { unique : true }
```

Or en faisant comme ça, à chaque tour de synchronisation (c'est à dire plusieurs dizaines de fois lorsque je travaillais en mode *watch* en lançant *npm run dev*), *Sequelize* créait un nouvel index unique sans écraser celui déjà existant.

Ainsi, je me retrouvais en BDD avec un

champ *name*, et plusieurs index identiques qui pointaient vers *name* : *name\_index\_1*, *name\_index\_2*, *name\_index\_3* etc.

J'ai donc cherché une solution, je suis tombé sur une issue ouverte sur le repo Github de *Sequelize* qui correspondait à mon problème, il y est conseillé d'appliquer le fix visible ligne 22 : au lieu de spécifier dans chaque config de champ si un index unique doit être créé, cela se fait simplement dans l'objet de config général du modèle en second paramètre, on peut spécifier en une fois plusieurs champs que l'on veut unique, ce qui créera les index associés.

Après ce fix, plus d'index dupliqués en BDD. J'ai supprimé les anciens index en doublons et tout est ok.

L'issue Github en question avec ma réponse apportée (pseudo « hyperdestru ») après avoir testé que la solution marchait sur mon projet :

<https://github.com/sequelize/sequelize/issues/12889#issuecomment-994486240>

### 3. Back-end

#### a) Documentation de l'API

Extrait de documentation pour la ressource Order et les routes d'authentification

MÉTHODES	ROUTES	PARAMÈTRES / CORPS / INFOS	RÉPONSES JSON
POST	auth/login	{ email: string, password: string }	{ accessToken: string }
POST	auth/register	{ email: string, password: string }	{ accessToken: string }
GET	order/	Aucun	{ orders: [] }
GET	order/:id	id: int → id commande	{ order : {} }
GET	order/:id/products <sup>(1)</sup>	id: int → id commande	{ products : [] }
GET	order/user/:id <sup>(2)</sup>	id: int → id utilisateur	{ orders : [] }
GET	order/user/:id/active <sup>(3)</sup>	id: int → id utilisateur	{ orders : [] }
GET	order/user/:id/archive <sup>(4)</sup>	id: int → id utilisateur	{ orders : [] }
POST	order/	{ UserId: int, StatusId: int, VendingMachineId: int, pickupDate: string, products: int[] } Création d'une commande.	Statut : 200 { message: string }
PATCH	order/:id	id: int → id commande { } → Tous champs faisant parti du modèle. Par exemple passer ceci dans le body: { StatusId : 4, pickupDate : '2022-02-10 00:00:00' } mettra à jour le statut et la date de récupération de la commande.	{ message: string }
DELETE	order/trash/:id	id: int → id commande « Soft delete ». Garde la ressource en BDD mais init son champ <i>deletedAt</i> au timestamp actuel.	Statut : 204 { message: string }
DELETE	order/:id	id: int → id commande « Hard delete ». N'existe plus en BDD après ça.	Statut : 204 { message: string }
POST	order/untrash/:id	id: int → id commande Restaure une commande précédemment supprimée en mode soft.	Statut : 204 { message: string }

(1) Retourne les produits d'une commande – (2) Retour les commandes d'un utilisateur – (3) Retourne les commandes actives d'un utilisateur (en statut « confirmée » et dont la date de recup n'est pas encore passée) – (4) Retourne les commandes archivées d'un utilisateur (en statut « archivée », « annulée », « récupérée »), en somme l'historique de commandes d'un utilisateur.

## b) Exemples de réponses

Fichier : routes/upload.js

```
1 router.get('/qrcode/:name', (req, res) => {
2   const file = req.params.name
3   const filepath = `${process.env.APP_ROOT}/public/upload/qrcodes/${file}.png`
4
5   fs.access(filepath, constants.R_OK, err => {
6     if (err) {
7       console.log(err)
8       res.status(500).send("qrcode does not exists or cannot be read")
9     } else {
10      res.status(200).sendFile(filepath)
11    }
12  })
13 })
```

Cette route retourne un fichier QR Code dont le nom est passé en paramètre et qui potentiellement est contenu dans le répertoire `app/public/upload/qrcodes`.

Ligne 5 – Je m’assure que le fichier recherché existe, avec le module Node.js **fs (file system)** et sa méthode `access`. Avec `constants.R_OK` je m’assure que je peux au moins lire le fichier (comprendre READ OK). La callback de cette méthode me dit s’il y a une erreur, si c’est le cas, le fichier n’est pas libre en lecture et donc je renvoie un statut 500 et un message bref expliquant la nature du problème : le fichier n’existe pas ou ne peut pas être lu.

Ligne 10 – S’il n’y a pas d’erreur je peux retourner le fichier. Pour retourner un fichier il faut utiliser la méthode `sendFile` de la réponse du routeur. Côté front on pourra simplement utiliser le chemin ainsi retourné, qui pointera vers le bon fichier sur le serveur.

Ces fichiers QR Code sont créés à partir du contrôleur de la route POST `/vendingmachine` (création d’un distributeur), grâce au package **NPM qrcode**.

Fichier : routes/vendingMachine.js

```
1 QRCode.toFile(`public/upload/qrcodes/${qrCodeFileName}`, machineUuid, {
2   color: { light: '#0000' }
3 }, function (err) {
4   if (err) console.log(err)
5   console.log("QRCode successfully created")
6 })
```

Le nom du fichier QR Code = valeur du champ `uuid` de la ressource `VendingMachine` créée + `.png` en extension de fichier.

Le uuid d’un distributeur est créé avec le module **node.js crypto** et sa méthode `randomUUID()` :

```
const machineUuid = crypto.randomUUID();
```

```
1 router.post('/login', (req, res) => {
2   const { email, password } = req.body
3
4   if (!email || !password) {
5     return res.status(400).json({
6       message: "incorrect email or password"
7     })
8   }
9
10  models.User.findOne({
11    where: { email: email }
12  }).then(user => {
13    if (user === null) {
14      return res.status(401).json({
15        message: "account does not exists"
16      })
17    }
18
19    bcrypt.compare(password, user.password).then(test => {
20      if (!test) {
21        return res.status(401).json({
22          message: "incorrect password"
23        })
24      }
25
26      const accessToken = jwt.sign({
27        id: user.dataValues.id,
28        email: user.dataValues.email
29      }, process.env.JWT_SECRET, {
30        expiresIn: process.env.JWT_DURATION
31      })
32
33      return res.json({
34        accessToken
35      })
36    }).catch(err => {
37      res.status(500).json({
38        message: "login process failed",
39        error: err
40      })
41    })
42  }).catch(err => {
43    res.status(500).json({
44      message: "database error",
45      error: err
46    })
47  })
48 })
```

Ceci est le contrôleur de la route qui permet la connexion d'un utilisateur.

Ligne 4 – Si la propriété *email* ou *password* n'existe pas dans le body de la requête alors je retourne un statut 400.

Ligne 10 – J'utilise la méthode *findOne* de Sequelize pour trouver un seul utilisateur (ici via son email).

Ligne 13 – Si aucun utilisateur n'est trouvé avec cet email (le email est en unique) alors je retourne un statut 401.

Ligne 19 – Autrement je continue. Il faut s'assurer que le password passé dans le body (entré à la connexion via un formulaire par exemple) correspond au **hash** stocké en BDD (j'utilise **bcrypt** pour hasher les mots de passe, cf. route *auth/register*). Pour réaliser cette comparaison j'utilise la méthode *compare* de *bcrypt*.

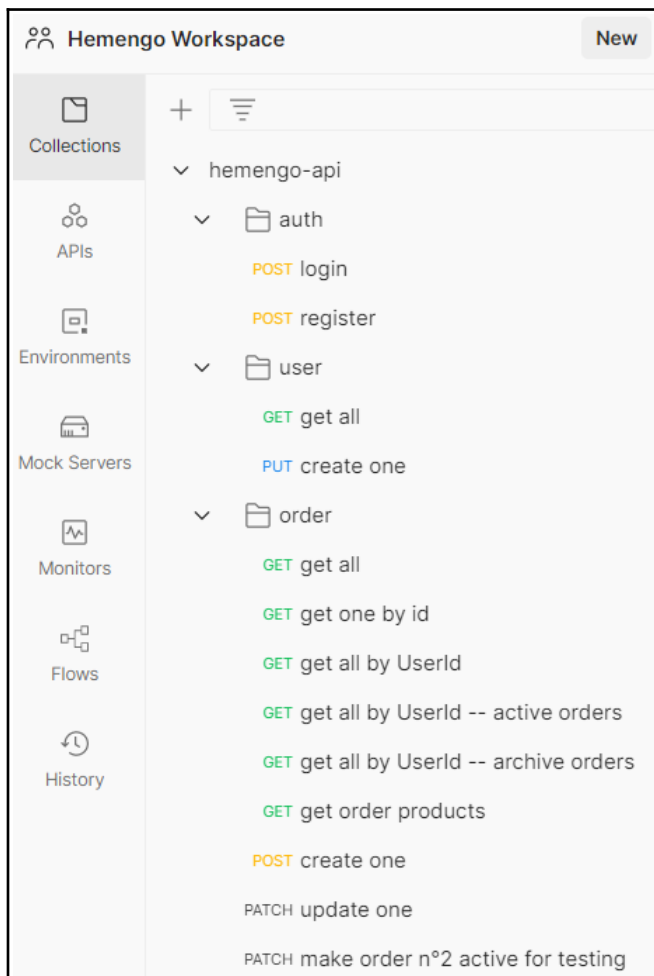
Ligne 20 – Si le test de comparaison n'est pas bon (mot de passe erroné) je retourne un statut 401.

Ligne 26 – Autrement je peux créer le token JWT avec l'email et l'id utilisateur dans sa payload.

Ligne 33 – Je peux enfin retourner ce token.

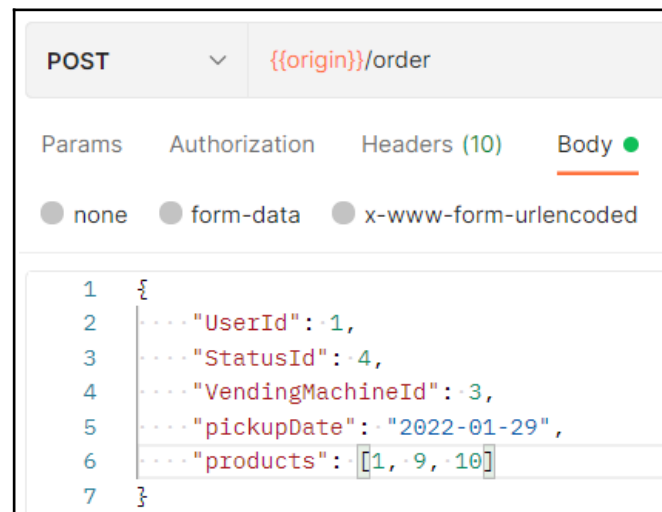
Lignes 36 – 42 – Les méthodes *catch* des promesses de *compare* et *findOne* me permettent de retourner un statut 500 si quelque chose se passe mal.

### c) Tester son API avec Postman



J'ai créé un workspace Hemengo dans lequel j'ai créé une collection hemengo-api afin de sauvegarder mes requêtes de test.

Exemple ci-dessous pour faire une création de commande, il n'y a plus qu'à cliquer *Send* et visualiser la réponse dans le panneau *Response*.



Pour avoir utilisé l'outil CURL en CLI auparavant j'ai fortement gagné en productivité en utilisant Postman.

Les routes de l'API passent par le *checkTokenMiddleware* (fichier `app/jwt/check.js`) qui vérifie la présence et la validité du token JWT dans le Authorization Header (champ Bearer Token) de chaque requête. A chaque fois que l'on veut tester une route il faut donc mettre le token reçu lors du login/register dans le header de la requête, pour éviter cette tâche répétitive on peut assigner une variable globale à la collection au moment du login et ensuite passer cette variable au Bearer Token de chaque requête pour que tout se fasse automatiquement :



## 4. Front-end

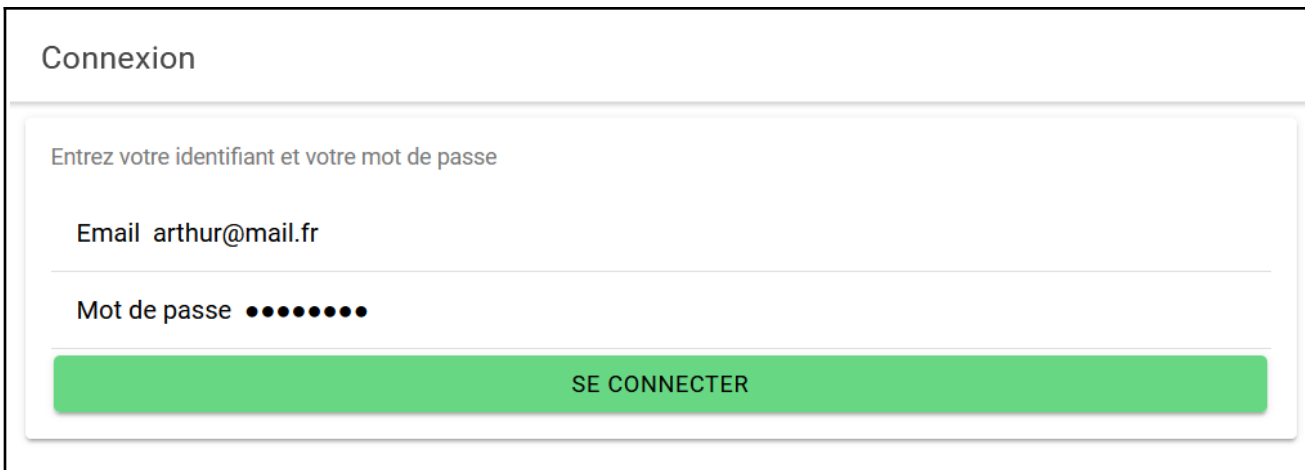
### a) Quelques vues

#### Page Accueil



```
$ ls src/app/pages/splash/  
splash.module.ts splash.page.html splash.page.scss splash.page.spec.ts splash.page.ts splash-routing.module.ts
```

#### Page Login



```
$ ls src/app/pages/login/  
login.module.ts login.page.html login.page.scss login.page.spec.ts login.page.ts login-routing.module.ts
```



**Bienvenue sur Hemengo**  
arthur@mail.fr

- Mon profil
- Demo
- Deconnexion

### Mes commandes à venir

Commande 1

Commande confirmée

29.5€ TTC

pomme sagartzea
 navet
 piment d'espelette
 fuet
 fleurs des montagnes

A venir chercher le 20/02/2022

Au distributeur de BAYONNE (64100)

DETAILS

RÉCUPÉRER COMMANDE

```
$ ls src/app/pages/profile/
profile.module.ts  profile.page.html  profile.page.scss  profile.page.spec.ts  profile.page.ts  profile-routing.module.ts
```

```
$ ls src/app/components/dashboard/
dashboard.component.html  dashboard.component.scss  dashboard.component.spec.ts  dashboard.component.ts
```

**Bienvenue sur Hemengo**  
arthur@mail.fr

- Mon profil
- Demo
- Deconnexion

Commande récupérée

### Distributeur de demonstration

UUID : 85d9376b-9a13-4665-ad59-19d90c107eca

A1	A2	A3	A4	A5	A6
B1	B2	B3	B4	B5	B6
C1	C2	C3	C4	C5	C6
D1	D2	D3	D4	D5	D6
E1	E2	E3	E4	E5	E6

SCAN DISTRIB

```
$ ls src/app/pages/demo/vending-machine/
vending-machine.module.ts  vending-machine.page.spec.ts
vending-machine.page.html  vending-machine.page.ts
vending-machine.page.scss  vending-machine-routing.module.ts
```

## 5. Mobile

### a) Partie Ionic

Ionic est un framework permettant de créer à partir **d'une seule codebase**, un applicatif web, Android et iOS. On peut partir au choix d'une architecture Vue.js, Angular, React ou Javascript/Typescript pur.

Lancer l'app Ionic en mode watch (lors d'un dev -- se rafraîchit automatiquement lorsqu'un changement dans le code est détecté)

```
ionic serve
```

---

Faire un simple build web ([dossier hemengo-app/www](#))

```
ionic build
```

Ajouter la plateforme native Android au projet ([dossier hemengo-app/android](#))

```
ionic capacitor add android
```

Build le projet, compile le code web en code android, update la plateforme android du projet (si par exemple on a fait des modifications dans le code android directement)

```
ionic capacitor sync android
```

Ouvre Android Studio afin de procéder au build de l'APK (similaire à un build classique que l'on fait pour une appli native, comme par exemple hemengo-scanner)

```
ionic capacitor open android
```

---

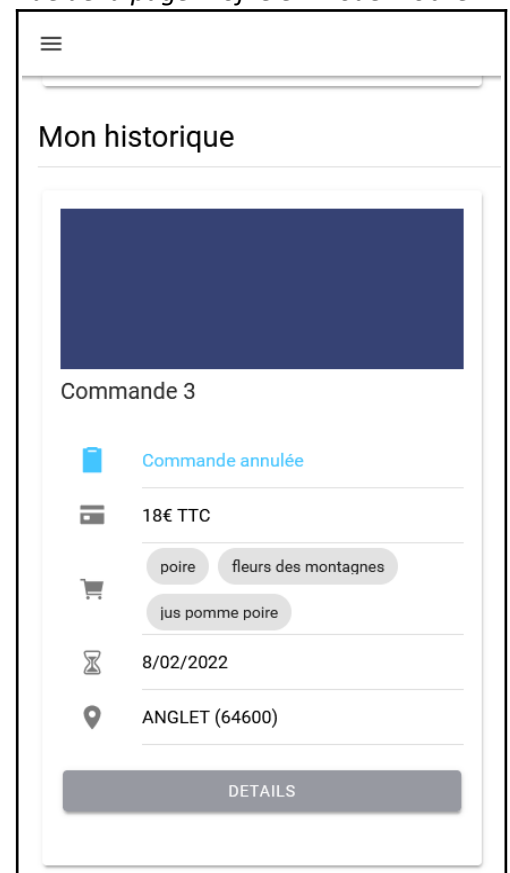
Générer une ressource de type Page dans l'application Ionic/Angular

```
ionic generate page pages/login
```

Générer une ressource de type Service dans l'application Ionic/Angular

```
ionic generate service services/user
```

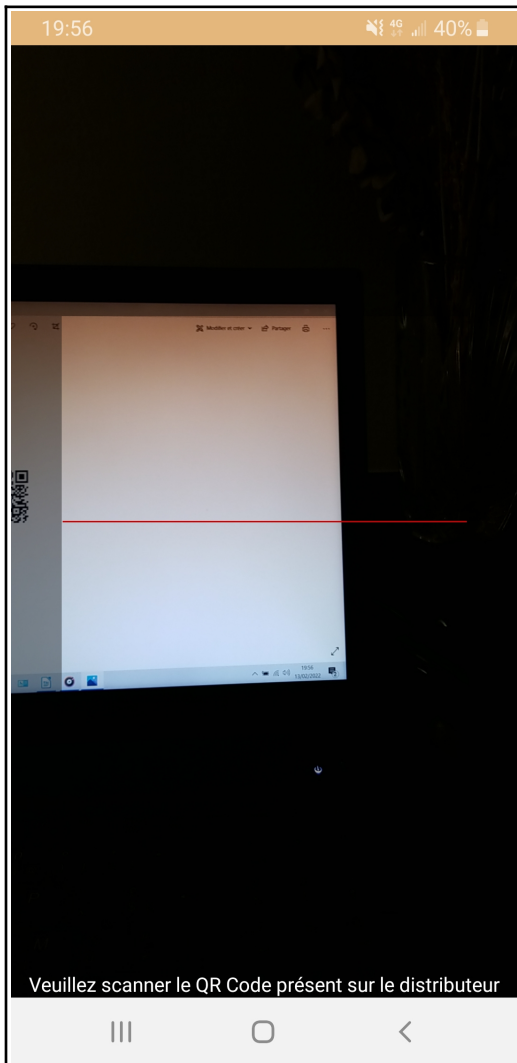
Vue de la page Profile en mode mobile



## b) Partie Android native

Quelques vues :

*MainActivity.java – intentIntegrator.initiateScan()*



*MainActivity.java – OnActivityResult()*



Fichier : *hemengo-scanner/app/src/main/res/layout/activity\_main.xml*

```
1 <Button
2     android:id="@+id/btnScan"
3     android:layout_width="220dp"
4     android:layout_height="64dp"
5     android:layout_marginBottom="240dp"
6     android:backgroundTint="@color/grass"
7     android:text="@string/cta_scan"
8     app:layout_constraintBottom_toBottomOf="parent"
9     app:layout_constraintEnd_toEndOf="parent"
10    app:layout_constraintHorizontal_bias="0.502"
11    app:layout_constraintStart_toStartOf="parent"
12    app:layout_constraintTop_toBottomOf="@+id/imgHemengoLogo"
13    app:layout_constraintVertical_bias="0.416" />
```

Ceci est le layout du bouton « Scanner le distributeur » de la vue ci-dessus.

Ligne 7 – j'appelle la string *cta\_scan* du fichier de ressources *res/values/strings.xml*, de cette manière il y a conformité avec le standard **i18n** (internationalisation).

❗ Dans cette partie comme dans l'ensemble du rapport, les commentaires dans le code du projet ont été temporairement enlevés lors des captures afin d'économiser de la place. Toutes les captures sont évidemment de courts extraits de chaque fichier.

### 1. Algorithmique

Fichier : *hemengo-app/src/app/demo/vending-machine.page.ts*

```
1 const orderProductsIds = this.order.products.map(p => p.id);
2 const filtAssoc = this.productRefAssoc.filter(a => orderProductsIds.includes(a.product.ProductId));
3 const refsToUnlock = filtAssoc.map(a => a.locker.MatrixElementRef);
4 const cols = Array.from(document.getElementsByClassName("vending-machine-cols"));
5 const foundLockers = refsToUnlock.map(ref => cols.find(el => el.textContent === ref));
```

Ligne 1 – Je crée un array d'id de produits avec la méthode *map*, que j'appelle sur les produits de la commande en train d'être récupérée (propriété *order* de la classe *VendingMachinePage*).

Ligne 2 – La propriété *productRefAssoc* est un array 2 dimensions où chaque entrée est un array composé d'un objet *ILocker* (interface locker) et d'un objet *IProduct* (interface product), elle permet de faire concorder casier du distributeur et produit à l'intérieur du casier. Avec la méthode *filter*, je filtre ce tableau pour n'avoir que les casiers qui ont bien les produits de la commande (méthode *includes* sur les id de *orderProductIds*).

Ligne 3 – Je crée un array des références de casiers (**A1, A2, E6, D4...**) à débloquent à partir du filtrage de *productRefAssoc*.

Ligne 4 – Je crée un array des éléments du DOM représentant le casier (grille à 2 dimensions).

Ligne 5 – Je récupère les refs à débloquent en trouvant les refs des casiers de la commande dans les refs du casier du DOM.

Fichier : *hemengo-app/src/app/demo/vending-machine.page.ts*

```
1 private makeGrid(refs: string[]): string[][] {
2     const grid = [];
3     while (refs.length) grid.push(refs.splice(0, 6));
4     return grid;
5 }
```

Ma méthode *makeGrid* retourne un array 2 dimensions à partir d'un array simple, ce nouvel array se composera de lignes de 6 éléments. La méthode *splice* « découpe » à chaque passe les 6 premières entrées du tableau *refs*. Cela fonctionne car *splice*

**modifie** le tableau sur lequel elle est appelée (ici *refs*), il y a **passage par référence** et non par valeur.

Fichier : *hemengo-api/app/helpers/util.js*

```
1 function isInTheFuture(date) {
2     try {
3         const possibleFutureTime = new Date(date).getTime();
4         return possibleFutureTime > Date.now();
5     } catch (err) {
6         if (err) return false;
7     }
8 }
```

Ma fonction *isInTheFuture* retourne *true* si la date passée en paramètre est ultérieure à la date où la fonction s'exécute. Retourne *false* si ce n'est pas le cas ou si une erreur est rencontrée. La méthode *getTime* et la méthode statique

*Date.now()* retournent un timestamp UNIX en millisecondes correspondant au temps écoulé depuis le 1<sup>er</sup> Janvier 1970.

## 2. HTML – CSS

Fichier : `hemengo-app/src/app/demo/vending-machine.page.html`

```
1 <ion-content>
2   <ion-card>
3     <ion-card-header>
4       <ion-card-title>
5         Distributeur de demonstration
6       </ion-card-title>
7       <ion-card-subtitle>
8         UUID : {{ machine.uuid }}
9       </ion-card-subtitle>
10    </ion-card-header>
11    <ion-card-content>
12      <div class="vending-machine-container">
13        <div class="vending-machine-grid">
14          <div class="vending-machine-lines" *ngFor="let line of grid">
15            <div class="vending-machine-cols" *ngFor="let col of line">
16              <p class="label">{{ col.toUpperCase() }}</p>
17            </div>
18          </div>
19        </div>
20        <div class="qr-code-container">
21          <div><img [src]="machine.qrCode" alt="QR Code"/></div>
22          <ion-button (click)="launchUnlocking()" size="small">
23            Scan distrib
24          </ion-button>
25        </div>
26      </div>
27    </ion-card-content>
28  </ion-card>
29 </ion-content>
```

Fichier : `hemengo-app/src/app/demo/vending-machine.page.scss`

```
1 .vending-machine-lines {
2   display: flex;
3   flex-direction: row;
4   justify-content: space-between;
5 }
6
7 .vending-machine-cols {
8   width: 100px;
9   height: 50px;
10  background-color: #5f9ea0;
11 }
12
13 .label {
14   color: #fff;
15   font-weight: 600;
16   padding: 0.5em;
17 }
18
19 .vending-machine-cols[unlock="true"] {
20   background-color: #98fb98;
21 }
```

Ligne 1 – Les casiers de chaque ligne du distributeur sont mis à la suite horizontalement et espacés de manière égale sur ce même axe horizontal (*flex-direction* spécifie l'axe principal du container et *justify-content* espace les éléments sur cet axe principal). Pour espacer les éléments sur l'axe secondaire (ici la verticale) il faut utiliser *align-items* en supplément.

Ligne 19 – Les éléments (casiers) de la classe possédant l'attribut *unlock* valant « true » alors ces éléments se voient attribués une couleur vert pâle. On peut ajouter un attribut à un élément du DOM avec la méthode *setAttribute* de cet élément.

### 3. Interactions client-serveur

Fichier : *hemengo-app/src/environments/environment.ts (front)*

```
1 export const environment = {
2   production: false,
3   accessToken: "accessToken",
4   endpoint: {
5     upload: "http://localhost:3000/upload/",
6     auth: "http://localhost:3000/auth/",
7     user: "http://localhost:3000/user/",
8     order: "http://localhost:3000/order/",
9     city: "http://localhost:3000/city/",
10    status: "http://localhost:3000/status/",
11    product: "http://localhost:3000/product/",
12    vendingMachine: "http://localhost:3000/vendingmachine/"
13  }
14 }
```

Je réunis en un seul endroit tous les endpoints de base principaux de mon API.

Ligne 3 – Je me sers de cette propriété pour aller chercher ce nom dans le Local Storage au besoin (méthodes *saveToken* et *getToken* de [services/token.service.ts](#)).

Fichier : *hemengo-app/src/app/services/order.service.ts (front)*

```
1 public getProducts(orderId: number): Observable<{ products: IProduct[] }> {
2   return this.http.get<{ products: IProduct[] }>(
3     `${environment.endpoint.order}${orderId}/products`
4   );
5 }
```

Ma méthode *getProducts* va joindre mon endpoint order. Elle retourne un observable typé avec l'interface *IProduct*.

Fichier : *hemengo-app/src/app/pages/profile.page.ts (front)*

```
1 this.orderService.getProducts(order.id).subscribe(res => {
2   this.orders[i].products = res.products;
3 });
```

L'utilisation de *getProducts* du service order pus haut. Je souscris à l'observable et je set les produits trouvés à la propriété order.

Fichier : *hemengo-api/app/routes/order.js (back) | route GET order/:id/products*

```
1 order.getProducts().then(products => {
2   if (products === null) {
3     return res.status(404).json({
4       message: "no products exist for the order"
5     })
6   }
7   return res.json({ products })
8 }).catch(err => {
9   res.status(500).json({
10    message: "unable to get order product(s)",
11    error: err
12  })
13 })
```

Côté API, extrait du contrôleur qui retourne les produits d'une commande (via le getter d'association *getProducts* de Sequelize).

Ligne 8 – La réponse utilisée plus haut dans *profile.page.ts* côté front, *products* y est de type *IProduct[]* (un tableau d'objets *IProduct*, cf. capture de *order.service.ts*).

Par exemple *this.orders[0].products* proposera donc les champs associé au modèle *Product* de l'API (j'ai fait au maximum correspondre l'interface avec le modèle).

#### 4. Programmation Javascript et Express

Fichier : *hemengo-api/app/server.js*

```
1 const express = require('express')
2 const cors = require('cors')
3 const checkTokenMiddleware = require('./jwt/check')
4
5 let db = require('./db.config')
6
7 const router = require('./routes/index')
8
9 const app = express()
10
11 app.use(cors())
12 app.use(express.json())
13 app.use(express.urlencoded({ extended: true }))
14
15 app.get('/', (req, res) => { res.send("Welcome to the Hemengo Distrib API") })
16
17 app.use('/auth', router.auth)
18 app.use('/upload', router.upload)
19 app.use('/city', checkTokenMiddleware, router.city)
20 app.use('/user', checkTokenMiddleware, router.user)
21 app.use('/order', checkTokenMiddleware, router.order)
22 app.use('/locker', checkTokenMiddleware, router.locker)
23 app.use('/status', checkTokenMiddleware, router.status)
24 app.use('/product', checkTokenMiddleware, router.product)
25 app.use('/producer', checkTokenMiddleware, router.producer)
26 app.use('/matricelement', checkTokenMiddleware, router.matrixElement)
27 app.use('/vendingmachine', checkTokenMiddleware, router.vendingMachine)
28 app.use('/productcategory', checkTokenMiddleware, router.productCategory)
29
30 app.get('*', (req, res) => { res.status(501).send("Route not implemented") })
31
32 db.authenticate().then(() => {
33   console.log("Database connection ok")
34 }).then(() => {
35   app.listen(process.env.SERVER_PORT, () => {
36     console.log(`Server is running on port ${process.env.SERVER_PORT}`)
37   })
38 }).catch(err => {
39   console.log('Database error', err)
40 })
```

Ligne 1 – 9 - Import des modules de bases nécessaires, du middleware qui va vérifier le token avant de passer à la suite (cf lignes 19 à 28), de l'instance Sequelize, de toutes les routes, instantiation d'express.

Ligne 30 – Route joker, équivalent à une page « not found » en statut 404 côté front, pour toutes les routes qui ne seraient pas dans les routes gérées (ligne 15 à 28).

Ligne 32 – Je m'authentifie/connecte à la base de donnée, en Javascript les méthodes *then* des promesses sont **chainables**. Dans le 2eme appel à *then* je lance le serveur avec *listen*, pour cette API il écoutera sur le port 3000. Le serveur ne démarre que si l'API se connecte à la BDD avec succès.

 Vous trouverez d'autres exemples de programmation Javascript (et Typescript) dans presque chacune des pages de ce rapport.

## 5. Programmation Typescript

Fichier : *hemengo-app/src/app/interfaces/order.ts*

```
1 import { IVendingMachine } from "../vendingMachine";
2 import { ICity } from "../city";
3 import { IProduct } from "../product";
4 import { IStatus } from "../status";
5
6 export interface IOrder {
7     id: number,
8     price: number,
9     pickupDate: string,
10    pickupToday: boolean,
11    StatusId: number,
12    status: IStatus,
13    VendingMachineId: number,
14    vendingMachine: IVendingMachine,
15    city: ICity,
16    products: IProduct[]
17 }
```

Fichier : *hemengo-app/src/app/helpers/toaster.ts*

```
1 import { toastController } from "@ionic/core";
2
3 /**
4  * Crée un toast qui s'affiche en haut pendant 3s.
5  * Utilise le ToastController de ionic/core.
6  * @param message Message que le toast affichera
7  * @param icon Intitulé d'icone ionic
8  * @param color "primary", "secondary", "warning"...
9  * @see https://ionic.io/ionicons
10 */
11 export async function classicToast(message: string, icon: string, color: string) {
12     const toast = await toastController.create({
13         position: "top",
14         color: color,
15         duration: 2000,
16         message: message,
17         icon: icon
18     });
19
20     await toast.present();
21     return await toast.onDidDismiss();
22 }
```

J'ai laissé la doc que j'ai écrite pour cette fonction dans cette capture afin d'illustrer l'utilisation de **JSDoc** dans le monde Typescript. Ligne 20 – J'attends que le message toast s'affiche. Ligne 21 – Je retourne la promesse de *onDidDismiss* (lorsque le message disparaît), de cette manière dans *src/app/pages/demo/vending-machine.page.ts* je peux lancer une action seulement lorsque le toast est bien disparu (ici la navigation vers la page profile) :

```
classicToast("Commande récupérée", "checkmark-circle-sharp", "success").then(() => { this.router.navigate(['/profile']) });
```



## 6. Programmation Java et Android natif

Fichier : `hemengo-scanner/app/src/main/java/com/example/hemengoscanner/MainActivity.java`

```
1 package com.example.hemengoscanner;
2
3 import android.content.Intent;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.TextView;
8 import android.widget.Toast;
9
10 import androidx.annotation.Nullable;
11 import androidx.appcompat.app.AppCompatActivity;
12
13 import com.google.zxing.integration.android.IntentIntegrator;
14 import com.google.zxing.integration.android.IntentResult;
15
16 public class MainActivity extends AppCompatActivity implements View.OnClickListener {
17     Button scanBtn;
18     Button manuBtn;
19
20     @Override
21     protected void onCreate(Bundle savedInstanceState) {
22         super.onCreate(savedInstanceState);
23         setContentView(R.layout.activity_main);
24
25         scanBtn = findViewById(R.id.btnScan);
26         manuBtn = findViewById(R.id.btnManu);
27
28         scanBtn.setOnClickListener(this);
29         manuBtn.setOnClickListener(new View.OnClickListener() {
30             @Override
31             public void onClick(View v) {
32                 Intent intentScanManu = new Intent(MainActivity.this, ScanManuActivity.class);
33                 startActivity(intentScanManu);
34             }
35         });
36
37         TextView helpTextMain = (TextView) this.findViewById(R.id.helpTextMain);
38         helpTextMain.setSelected(true);
39     }
40
41     @Override
42     public void onClick(View v) {
43         IntentIntegrator intentIntegrator = new IntentIntegrator(this);
44
45         intentIntegrator.setDesiredBarcodeFormats(IntentIntegrator.QR_CODE);
46         intentIntegrator.setPrompt("Veuillez scanner le QR Code présent sur le distributeur");
47         intentIntegrator.setCaptureActivity(AnyOrientationCaptureActivity.class);
48         intentIntegrator.setOrientationLocked(true);
49         intentIntegrator.initiateScan();
50     }
51
52     @Override
53     protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
54         super.onActivityResult(requestCode, resultCode, data);
55
56         IntentResult intentResult = IntentIntegrator.parseActivityResult(requestCode, resultCode, data);
57
58         if (intentResult != null) {
59             if (intentResult.getContents() == null) {
60                 Toast.makeText(getBaseContext(), "Scan annulé", Toast.LENGTH_LONG).show();
61             } else {
62                 String msgContent = "Distributeur reconnu ! (data: " + intentResult.getContents() + ")";
63                 Toast.makeText(getBaseContext(), msgContent, Toast.LENGTH_LONG).show();
64             }
65         } else {
66             super.onActivityResult(requestCode, resultCode, data);
67         }
68     }
69 }
```

Ligne 13 – 14 – J'utilise la librairie **Zxing** pour mettre en place le scanner de QR Code. Il faut importer la classe *IntentIntegrator* pour lancer la camera et configurer le scan et la classe *IntentResult* pour récupérer les données résultantes du scan.

Ligne 16 – On implémente l'interface *OnClickListener* de *View* car il y a 2 boutons dont on doit gérer le clic dans cette activité (un qui lance la camera pour le scan et l'autre qui lance l'activité *ScanManuActivity* pour le mode manuel).

Lignes 45 – 48 – Config du scanner Zxing.

Ligne 49 – Lancement de la camera au clic.

Ligne 56 – Lorsque l'activité rend un résultat, on parse se résultat (de type *IntentResult*).

Lignes 58 – 64 – Si *intentResult* n'est pas null et s'il a un contenu on affiche ce contenu (*getContents*) dans un Toast, dans notre cas le contenu = le uuid du

distributeur.

## 7. Programmation mobile avec Ionic

Fichier : *hemengo-app/src/app/pages/profile.page.ts*

```
1 import { Platform } from '@ionic/angular';
2
3 private launchOrderPickupAction(order: IOrder) {
4   if (this.platform.is('android')) {
5     window.open('android-app://com.example.hemengoscanner', "_system");
6     classicToast("Plateforme Android", "logo-android", "success");
7   } else {
8     this.router.navigate(['demo', 'order', order.id]);
9   }
10 }
```

Je peux avoir accès à la plateforme d'exécution actuelle de l'application avec la classe *Platform* d'Ionic et ainsi lancer des actions différentes en fonction du matériel. Parmi les plateformes que peut renvoyer la méthode *is()* on compte entre autres *android*, *desktop*, *mobile*, *mobileweb*, *ios*... Lorsque la vue adaptative est activée dans le navigateur (cf. exemple en dessous), changer la plateforme d'émulation changera aussi la valeur retournée par *Platform.is*.

Dans notre cas il faudrait idéalement lancer la MainActivity de Hemengo Scanner pour pouvoir scanner le QR Code. Ce n'est pas quelque chose que j'ai réussi à réaliser dans le temps imparti mais toutefois la structure est là et je lance la démonstration (ligne 8) si la plateforme n'est pas Android.

Fichier : *hemengo-app/src/app/pages/splash/splash.page.html* (splash = splash screen = accueil)

```
1 <ion-col size-xl="4" size-md="6">
2   <h1>
3     
4   </h1>
5   <h5 class="catchphrase">
6     Commandez des produits du Pays-Basque
7   </h5>
8   <ion-button expand="block" routerLink="/login">
9     Commencer
10  </ion-button>
11  <p class="link-to-register">
12    <a routerLink="/register">S'inscrire</a>
13  </p>
14 </ion-col>
```

Ligne 1 – Les attributs Ionic *size* me permettent de spécifier le nombre de **colonnes** que doit prendre le composant **responsive** *ion-col*, les tailles *xl*, *lg*, *md*, *sm* que l'on colle à l'attribut correspondent aux **breakpoints** des différentes **tailles** d'écrans. Ma source d'information sur ce sujet : <https://ionicframework.com/docs/layout/grid#default-breakpoints>  
Ici je spécifie que la grille doit prendre 4 colonnes pour les devices de type PC de bureau, laptops etc, et 6 colonnes pour les devices de type tablettes (ipad...). Par défaut la grille occupe l'espace maximal c'est à dire 12 colonnes.

Afin de tester ce comportement je lance l'application en vue adaptative dans mon navigateur (ctrl + shift + m) :



 D'autres informations sur la partie Ionic et Android en partie A – 5 (page 18).

## 8. Programmation Angular

Fichier : `hemengo-app/src/app/pages/profile/profile.page.html`

```
1 <ion-chip *ngFor="let product of order.products">
2   <ion-label>
3     {{ product.name }}
4   </ion-label>
5 </ion-chip>
```

Ligne 1 - Directive structurale **ngFor** utilisée ici pour boucler sur les produits d'une commande. Donc pour chaque produit, un composant *ion-chip* est créé.

Ligne 3 - Chaque *ion-label* aura comme texte le nom d'un produit grâce à la syntaxe d'interpolation **{{ }}**.

```
1 <ion-button
2   expand="block"
3   [disabled]="!order.pickupToday"
4   (click)="launchOrderPickupAction(order)">
5   Récupérer commande
6 </ion-button>
```

Ligne 3 – Liaison de propriété (property binding) grâce à la syntaxe **[ ]**. Ici la valeur de l'attribut *disabled* du bouton est liée à la valeur de la propriété *pickupToday* (un booléen) d'une commande. Si *order.pickupToday* vaut *true* alors *disabled* vaudra *false* et le bouton sera actif (clicquable).

Ligne 4 – Liaison d'événement (event binding), au clic

sur le bouton, la méthode *launchOrderPickupAction* de `profile.page.ts` sera exécutée.

Fichier : `hemengo-app/src/app/pages/login/login.page.html`

```
1 <ion-note *ngIf="errorStatus" color="danger">
2   {{ getErrorMsg() }}
3 </ion-note>
```

Ligne 1 – Directive structurale **ngIf** utilisée ici pour afficher un message d'erreur au niveau du formulaire de login si une erreur est retournée par l'API (mauvais email par exemple), alors la propriété *errorStatus* n'est plus undefined et on affiche le retour de la méthode

*getErrorMsg* de `login.page.ts` (ci dessous).

```
1 getErrorMsg(): string {
2   if (this.errorStatus >= 400 && this.errorStatus < 500) {
3     return "Connexion non autorisé avec ces identifiants";
4   } else if (this.errorStatus < 400 || this.errorStatus >= 500) {
5     return "Echec connexion";
6   }
7 }
```

Fichier : `hemengo-app/src/app/app-routing.module.ts`

```
1 {
2   path: 'profile',
3   loadChildren: () => import('./pages/profile/profile.module')
4     .then(m => m.ProfilePageModule), canActivate: [AuthGuard]
5 },
6 {
7   path: 'demo/order/:orderId',
8   loadChildren: () => import('./pages/demo/vending-machine/vending-machine.module')
9     .then(m => m.VendingMachinePageModule), canActivate: [AuthGuard]
10 }
```

Extrait du routage de l'application.

Ligne 2 – 4 – L'URL `http://localhost:3000/profile` rendra la vue de la page Profile. Mais je laisse l'utilisateur joindre cette route seulement si la méthode *canActivate* du **AuthGuard** (`src/app/helpers/auth.guard.ts`) retourne *true* : c'est à dire si la méthode *isLoggedIn()* de `src/app/services/token.service.ts` retourne *true* également, ce qui signifierait qu'un token existe bien dans le LocalStorage. En d'autres termes les routes avec *canActivate* : `[AuthGuard]` ne sont accessibles qu'aux personnes **authentifiées**.

## 9. Les tests – API & App Ionic/Angular

Pour les tests d'API j'ai utilisé la lib Jasmine installée avec NPM avec la commande : `npm install jasmine`

Fichier : `hemengo-api/app/spec/support/jasmine.json`

```
1 {
2   "spec_dir": "spec",
3   "spec_files": [
4     "**/*[sS]pec.?(m)js"
5   ],
6   "helpers": [
7     "helpers/**/*.?(m)js"
8   ],
9   "env": {
10    "stopSpecOnExpectationFailure": false,
11    "random": true
12  }
13 }
```

Fichier de config de Jasmine.

Ligne 2 – Les fichiers de tests devront se trouver dans le dossier spec.

Ligne 3 – Jasmine prendra en compte pour test tous les fichiers JavaScript (modules .mjs inclus) du dossier spec (ou Spec).

Ligne 11 – Les tests se feront en l'ordre aléatoire.

Fichier : `hemengo-api/app/spec/order.spec.js`

```
1 const axios = require("axios")
2
3 describe("Order", function () {
4   const endpoint = "http://localhost:3000/order"
5
6   describe("GET order without previous login", function () {
7     it("returns Unauthorized response", function () {
8       axios.get(endpoint).catch(err => {
9         expect(err.response.status).toBe(401)
10      })
11    })
12  })
13 })
```

Avant de réaliser ce test de requête il faut lancer l'API avec `npm run dev` ou `npm run start` (ce sont des raccourcis, en réalité `npm run dev` lance `nodemon -r dotenv/config app/server.js`).

Pour de la concision j'utilise le package **Axios** pour requêter mon API.

Ligne 3 – Je déclare une **Suite** « Order » qui va tester tout ce qui concerne la router order.

Ligne 6 – Je déclare une sous-Suite qui va devoir répondre à la question « Puis-je

recupérer les commandes de la BDD sans m'être authentifié auparavant ? ».

Ligne 7 – La callback de la fonction `it()` contient mon **Expectation**.

Ligne 9 – Je construis une expectation avec la fonction `expect`. Ici, je « m'attends » à ce que l'API, lorsque je joint la route order sans authentification ou Header d'autorisation avec un Bearer Token spécifié, me retourne un statut 401. Si cette expectation retourne true alors le test passe.

Fichier : `hemengo-app/src/app/app.component.spec.ts`

```
1 it('should create the app', waitForAsync(() => {
2   const fixture = TestBed.createComponent(AppComponent);
3   const app = fixture.debugElement.componentInstance;
4   expect(app).toBeTruthy();
5 }));
```

L'instance du composant parent de l'application (sélecteur `app-root`, rien au dessus) doit être **truthy** : ne pas valoir `null`, `undefined`, `false`, `0`...

## 10. Gestion de versions (Git)

 Voir page 4 pour les liens vers les repos distants des 3 projets sur Github.

Hemengo API, Hemengo App et Hemengo Scanner ont été versionnés avec Git et Github.

Voici par exemple les commandes effectuées à la création d'un projet :

*Initialise un repo Git en local*

```
git init
```

Crée un fichier caché .gitignore (les noms de fichiers inscrits ne seront pas versionnés, crée ensuite un fichier README

```
touch .gitignore & touch README.md
```

Index pour versioning tous les fichiers courants (sauf ceux dans le .gitignore)

```
git add *
```

Enregistre les modifications apportées aux fichiers

```
git commit -m "Init project"
```

Ajoute un repo distant, ici nommé origin, pointant vers son adresse Github (repo Github créé avant via mon compte)

```
git remote add origin https://github.com/hyperdestru/hemengo-app.git
```

Pousse les modifications apportées sur le repo

```
git push -u origin master
```

Et voici par exemple le workflow habituel suivi lors d'un travail sur une nouvelle fonctionnalité :

Création d'une nouvelle branche de travail, je **préfixe** le nom par la nature de la tâche : *feature* pour une fonctionnalité, *bugfix* pour un bug

```
git branch feature/pickup-order
```

Je me place sur la branche précédemment créée

```
git checkout feature/pickup-order
```

Ajout du fichier sur lequel je viens de travailler

```
git add src/app/services/vending-machine.service.ts
```

J'enregistre mon travail

```
git commit src/app/services/vending-machine.service.ts -m "Pickup order - Ajoute methode qui fetch les lockers d'une machine"
```

Je pousse les modifications sur la branche de travail

```
git push origin feature/pickup-order
```

Je me replace sur la branche principale

```
git checkout master
```

Je fusionne la branche principale avec ma branche de travail, l'option `--no-ff` (no fast-forward) permet de toujours generer un commit de merge, cela produit un historique (git log) plus clair.

```
git merge --no-ff feature/pickup-order
```

Je pousse ce merge sur ma branche principale

```
git push origin master
```

Je peux maintenant supprimer ma branche de travail distante et locale

```
git push origin --delete feature/pickup-order
```

```
git branch -d feature/pickup-order
```

*Extrait de la commande **git log** sur le projet hemengo-app*

```
commit 4587d5e9720ff145caef11b4f776ec2a16d9c773
```

```
Author: hyperdestru <slotuser@gmail.com>
```

```
Date: Sat Feb 5 17:58:16 2022 +0100
```

```
Pickup order - Ajoute methode qui fetch les lockers d'une machine.
```

## 11. Docker

Fichier : *hemengo-api/docker-compose.yml*

```
1 version: '3'
2 services:
3   hemengo-api:
4     image: hemengo-api:latest
5     container_name: "hemengo-api"
6     build: .
7     ports:
8       - 3000:3000
9     environment:
10      SERVER_PORT: $SERVER_PORT
11      DB_HOST: $DB_SGBD
12      DB_PORT: $DB_PORT
13      DB_NAME: $DB_NAME
14      DB_USER: $DB_USER
15      DB_PASS: $DB_PASS
16      JWT_SECRET: $JWT_SECRET
17      JWT_DURING: $JWT_DURING
18      BCrypt_SALT_ROUND: $BCRYPT_SALT_ROUND
19     depends_on:
20       - mariadb
21
22   mariadb:
23     image: mariadb
24     container_name: "mariadb"
25     ports:
26       - 3306:3306
27     environment:
28      MYSQL_DATABASE: $DB_NAME
29      MYSQL_USER: $DB_USER
30      MYSQL_PASSWORD: $DB_PASS
31      MYSQL_ROOT_PASSWORD: $DB_ROOT_PASS
32     volumes:
33       - ../data:/var/lib/mysql
```

```
PS C:\Users\forwi\prog\hemengo-api> docker-compose up
Creating network "hemengo-api_default" with the default driver
Pulling mariadb (mariadb:latest)...
latest: Pulling from library/mariadb
08c01a0ec47e: Pull complete
a2bcb14c13a1: Pull complete
29c56760f879: Pull complete
a9500a218fc: Pull complete
a765d76e68d9: Pull complete
c6945738f085: Pull complete
62787b7c58c5: Pull complete
d2987a30cfe4: Pull complete
cbc65983d8b5: Pull complete
db216f91595e: Pull complete
Digest: sha256:ca31f38b6e325ece985d857db7eba1fe59928b4fd83ff8a55cb912c9684b9e43
Status: Downloaded newer image for mariadb:latest
Building hemengo-api
failed to get console mode for stdout: Descripteur non valide
[+] Building 72.8s (10/10) FINISHED
```

Pour lancer le container de l'API je lance **Docker Desktop** puis je lance la commande ***docker-compose up*** à la racine du fichier *docker-compose.yml* dans une CLI **Powershell**.