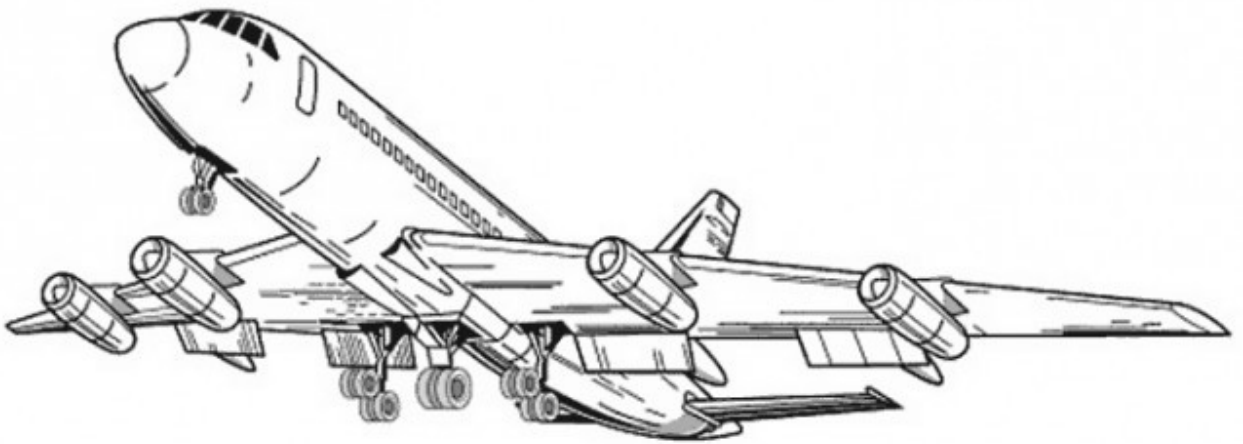


Projet 2020

Gestion de voyageur en Avion



Sommaire

I. Présentation du projet	3
II. Présentation des différentes tâches.....	4
III. Analyse UML	5
1. Diagramme de cas d'utilisation.....	5
2. Diagramme de déploiement.....	6
3. Diagramme de classes	7
3. Diagramme de séquence.....	8
IV. Mes tâches	9
1. Langages de programmation et outils utilisés.....	9
2. Création du projet.....	10
3. La balance.....	13
4. Connexion à la B.D.D.....	14
5. Template N°1: Scanner.....	15
6. Template N°2: Identification.....	22
7. Template N°3: Payement.....	30
V. Temps prévisionnel	37
VI. Avis personnel	37
VII. Conclusion.....	38
VIII. Annexes.....	38

I. Présentation du projet

Contexte : Une personne souhaite voyager en un avion. Il achète son billet d'avion sur un site web et loue un véhicule en location, qu'il récupérera au moment de l'arrivée à destination.

Objectif : Notre projet consiste à prendre en charge un voyageur, de l'achat du billet d'avion, jusqu'à l'embarquement. La location d'un véhicule en liaison avec l'arrivée et le départ de l'avion sera également proposée.

On doit offrir la possibilité de rechercher et réserver un billet d'avion, en offrant plusieurs compagnies aériennes, plusieurs horaires, plusieurs prix et des options telle que vols avec ou sans escale.

Si le voyageur choisit un vol, nous devons récupérer ses informations personnelles et mettre en place un moyen de paiement sécurisé en ligne, pour l'achat de billet. Une fois le paiement effectué, il faudra confirmer la réservation par l'envoi d'un mail avec un billet électronique comportant un numéro de billet unique (E-ticket).

Le jour du départ, une fois à l'aéroport, le voyageur pourra s'enregistrer auprès d'un comptoir d'enregistrement automatique, à l'aide de son code-barres se trouvant sur son passeport et son numéro de billet. Une fois enregistrer, la carte d'embarquement s'imprimera.

Cependant, si le voyageur dispose de bagages en soute, il devra se présenter devant un comptoir d'enregistrement, où un opérateur procédera à la pesée des bagages. Un ticket comportant un code-barres sera imprimé pour chaque bagage et fixé sur celui-ci et un double de ce ticket sera remis au voyageur. L'opérateur générera aussi la carte d'embarquement. En cas de dépassement en poids des bagages, un paiement électronique lui sera proposé.

Lors de l'accès à l'avion, le code-barres se trouvant sur sa carte d'embarquement sera lu, ce qui confirmera la présence du voyageur sur le vol.

En parallèle à la gestion de billet, le voyageur aura la possibilité de louer des véhicules à l'aéroport via un site web. Le voyageur pourra indiquer manuellement les jours et les heures désirées, afin qu'un choix de véhicules lui soit proposé.

Soit il pourra renseigner son numéro E-ticket, afin qu'une série de véhicules lui soit proposé pour des dates correspondant aux horaires d'arrivée et de départ de l'avion.

Un billet de réservation devra être imprimé lorsque la location sera effectuée.

II. Présentation des tâches

Le projet est divisé en 4 parties :

étudiant 1 (Salavudeen Hadji) : Conception d'une IHM permettant de rechercher un vol avec des différents choix + conception d'une base de données. Il devra récupérer les informations des clients avec leur numéro de passeport et proposer un moyen de paiement sécurisé en ligne. Une fois le paiement effectué, il devra enregistrer les informations du voyageur dans la base de données et mettre en place la génération d'un billet électronique en PDF, avec les informations du voyageur et du vol ainsi que l'envoi de mail automatiquement avec le PDF généré.

étudiant 2 (Samba ba) : Conception d'une IHM d'une borne d'enregistrement automatique, pour que le voyageur puisse s'enregistrer et imprimer sa carte d'embarquement à l'aide du numéro de billet électronique et de son passeport.

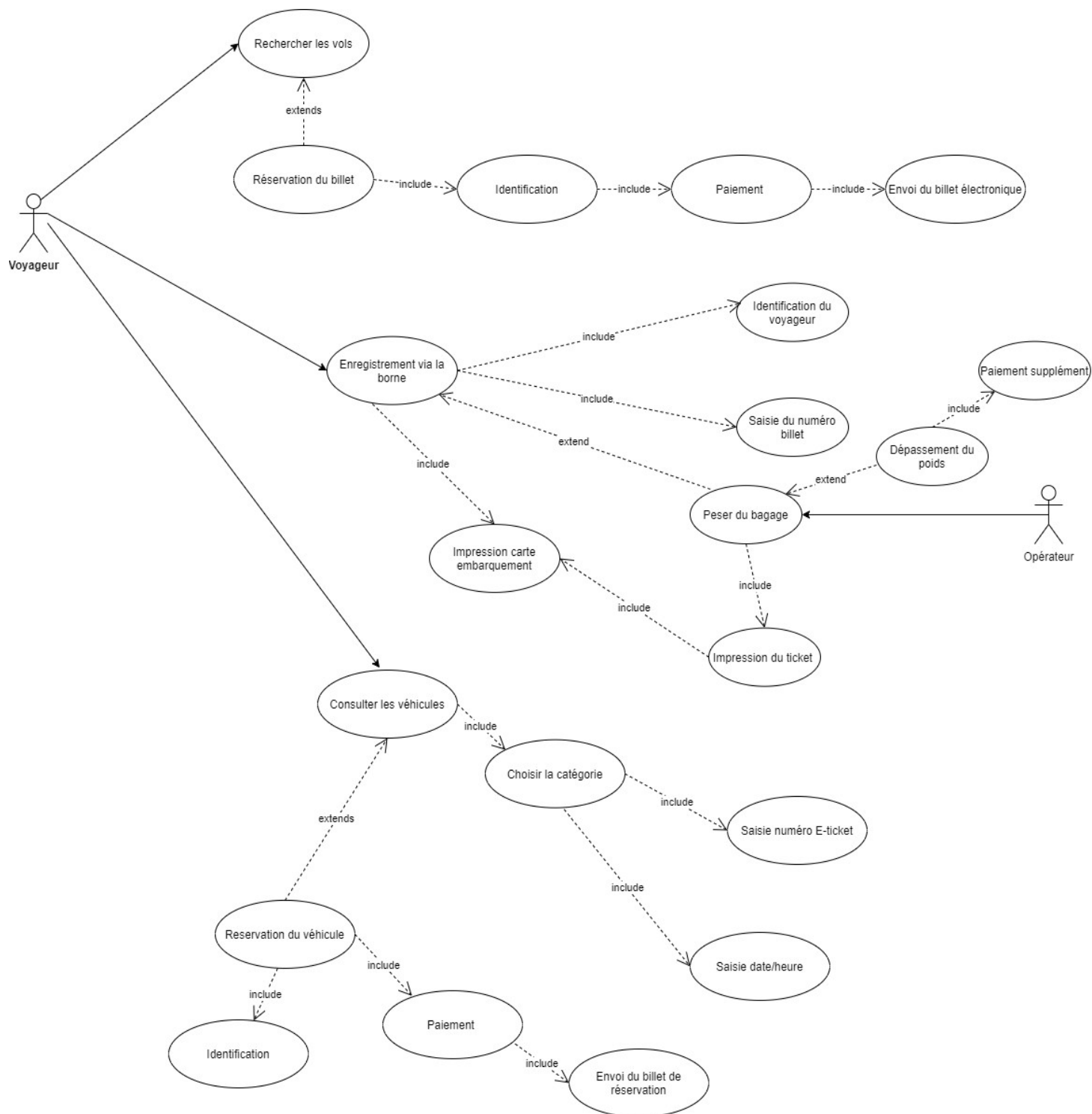
Il est en charge de vérifier la carte d'embarquement lors d'accès à l'avion, pour confirmer la présence du voyageur sur ce vol.

étudiant 3 (Moi) : Conception d'une IHM pour l'enregistrement des bagages et la génération de la carte d'embarquement. Un opérateur s'occupera d'enregistrer le voyageur en récupérant son billet d'avion et son passeport. Il doit aussi peser les bagages si le voyageur en possède et gérer le paiement en cas de dépassement du poids maximum (10 kg). En parallèle, il doit imprimer deux tickets bagage et la carte d'embarquement.

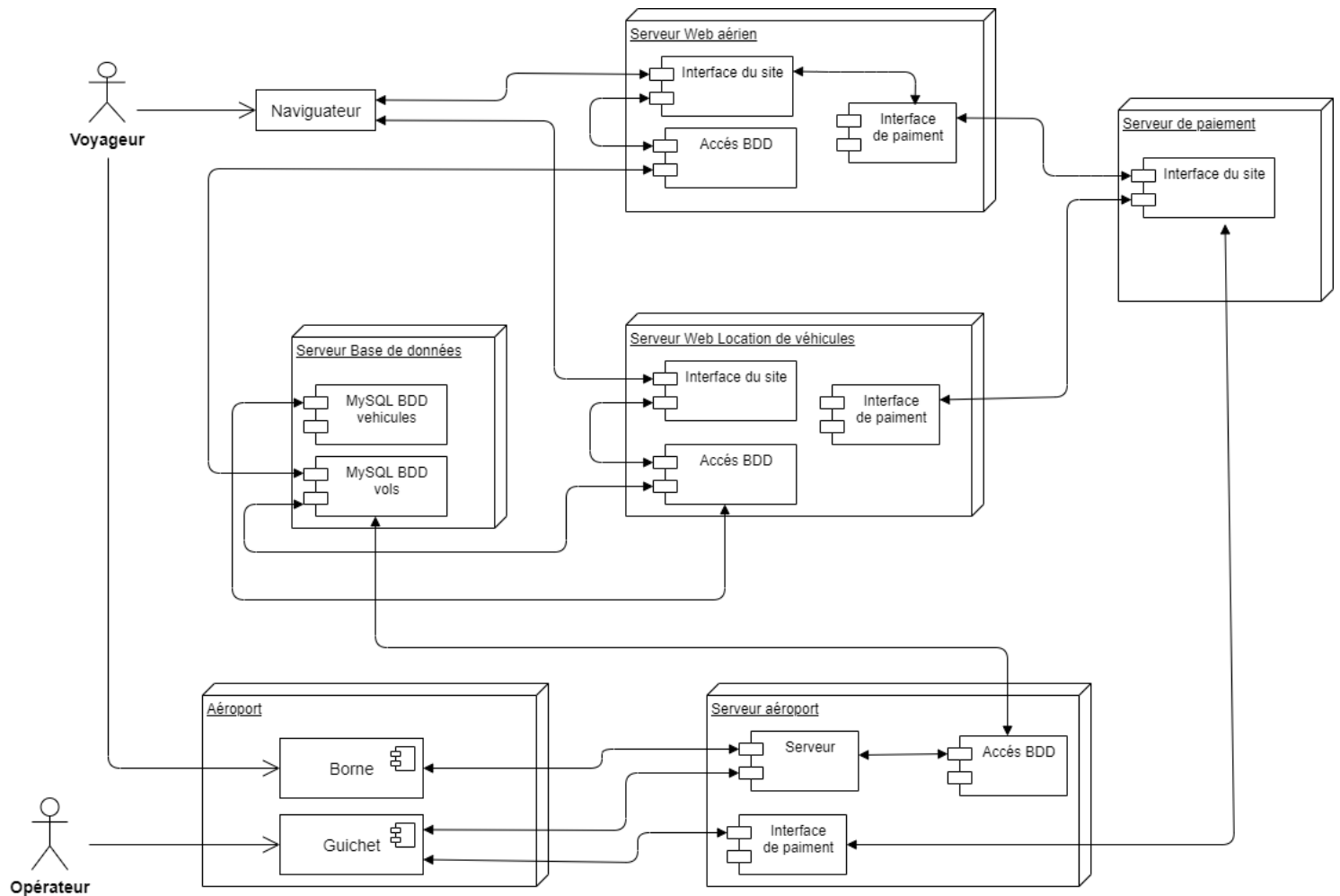
Matériel : Balance (USB / RS232), douchette (USB), imprimante thermique (USB / RS232),

étudiant 4 (Mohamed Djemaa) : Conception d'une IHM permettant de rechercher des véhicules disponibles, en fonction des différentes heures et jours. Il doit mettre en place un paiement sécurisé pour la location et mettre en place une impression de ticket de location lors de prises de véhicules.

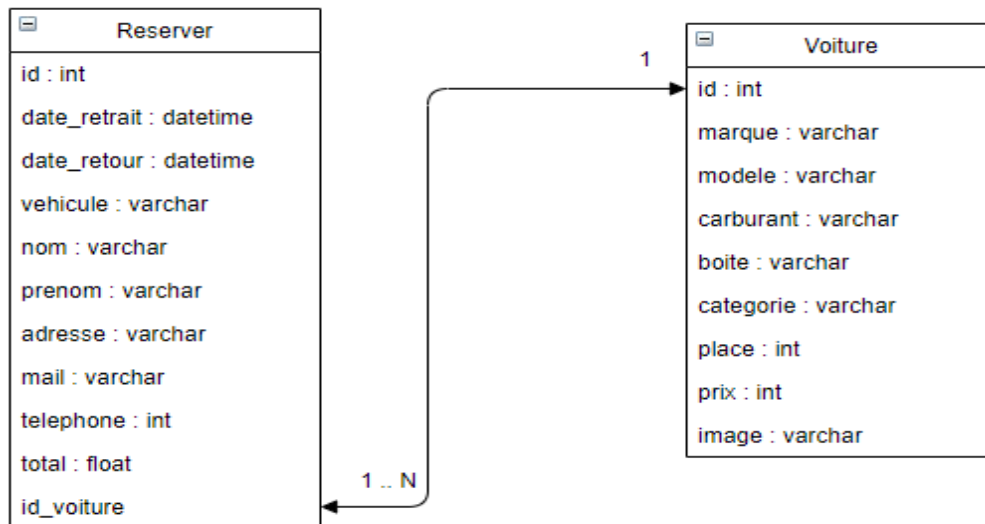
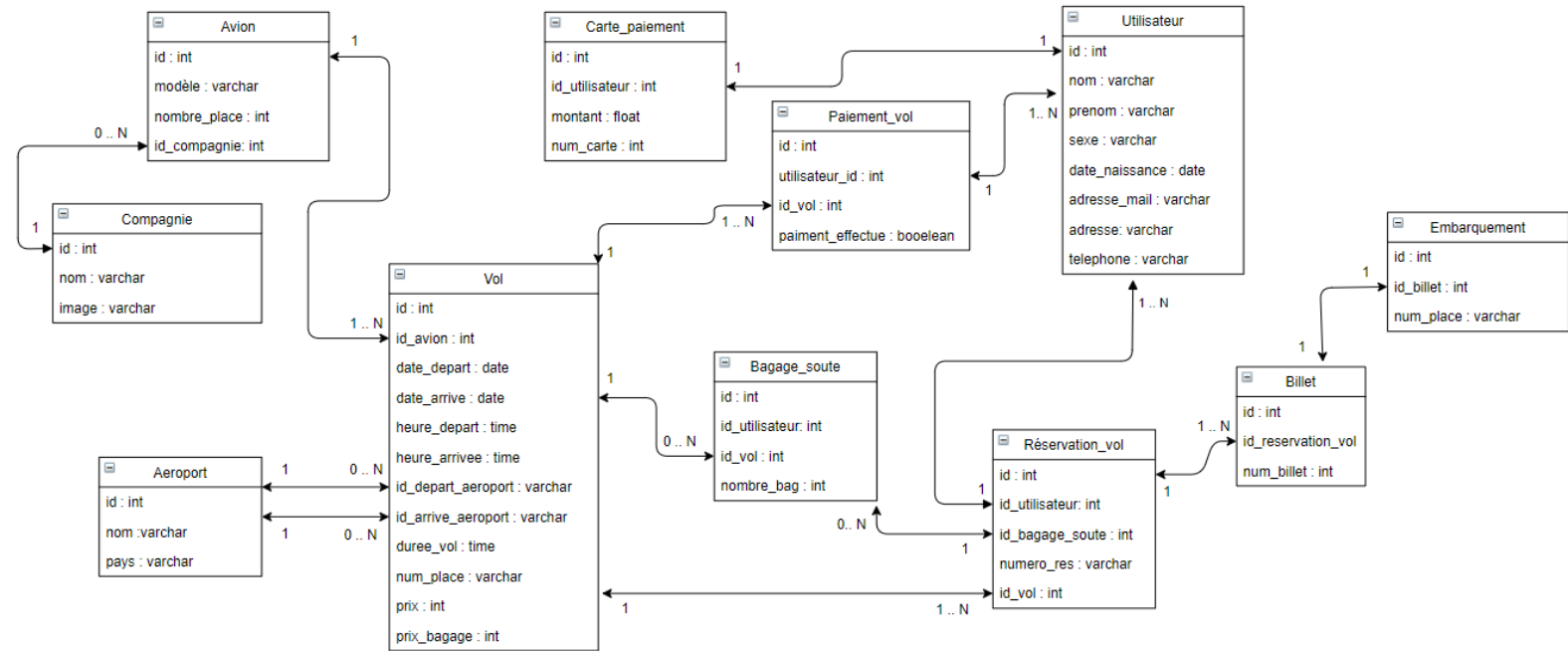
III.1. Diagramme de cas d'utilisation



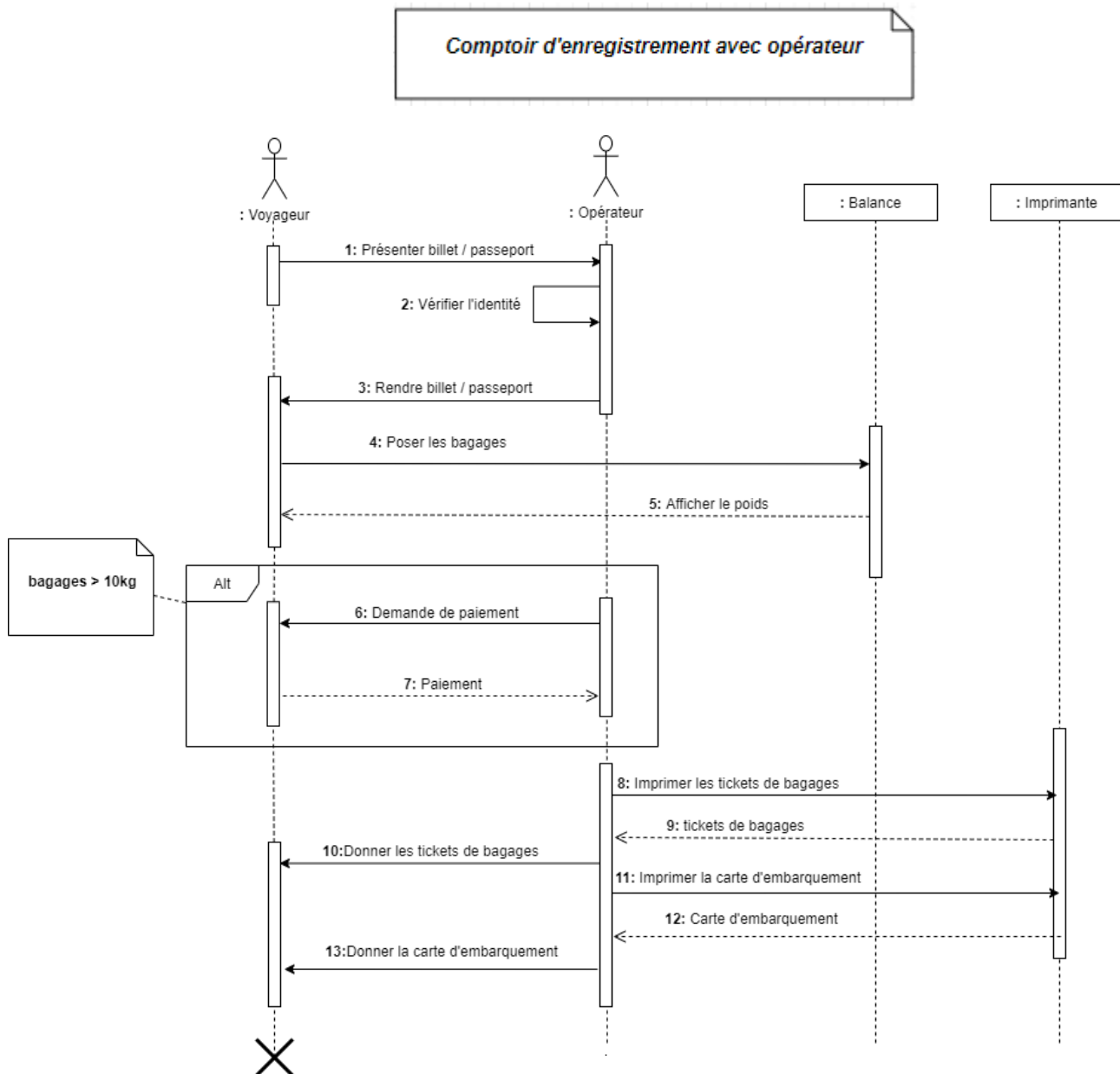
III.2. Diagramme de déploiement



III.3. Diagramme de classes



III.4. Diagramme de séquence

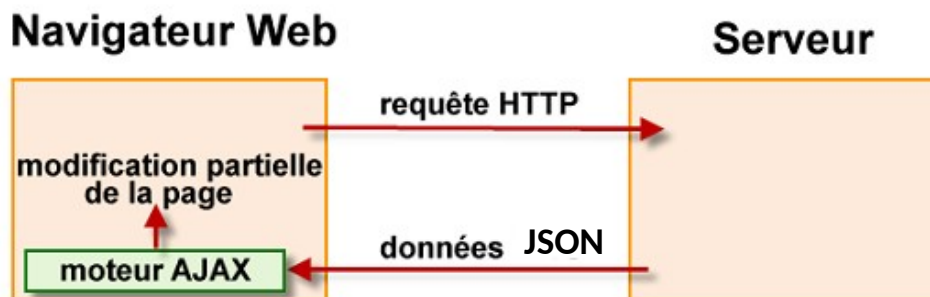


IV.1.Langages et outils

Ayant longuement réfléchi sur la conception de mon projet, j'ai décidé de réaliser mon projet avec les langages suivants:

Pour le côté **front-end**, je suis resté sur la base :

- HTML
- CSS + le **framework** Bootstrap
- JavaScript + la **bibliothèque** jQuery
- AJAX (*Asynchronous JavaScript and XML*) : J'ai utilisé AJAX pour:
 - Récupérer des données côté serveur pour ensuite les traiter du côté client.
 - Envoyer des données côté client vers le serveur.



Pour le côté **back-end** j'hésitais entre **PHP** et **Node.js**. J'ai décidé finalement de tout faire en JavaScript car cela me plaît et je désirais développer mes connaissances dessus, j'ai donc choisi Node.js.

NodeJS est une plateforme de développement qui permet de faire du JavaScript coté serveur et qui met à disposition plusieurs bibliothèques JavaScript.

Node.js permet de faire du **routing**, de réaliser des requêtes **SQL**...

Il repose entièrement sur le moteur d'exécution V8 de Google Chrome qui permet d'avoir des performances de très haut niveau.

N'ayant aucune connaissance en Node.js j'ai dû faire beaucoup de recherches là dessus.

J'ai découvert plusieurs fonctionnalités et plusieurs paquets intéressant à l'aide de **npm** (Gestionnaire de paquets de Node.js), comme par exemple « SerialPort » ou « body-parser » que j'expliquerai par la suite.

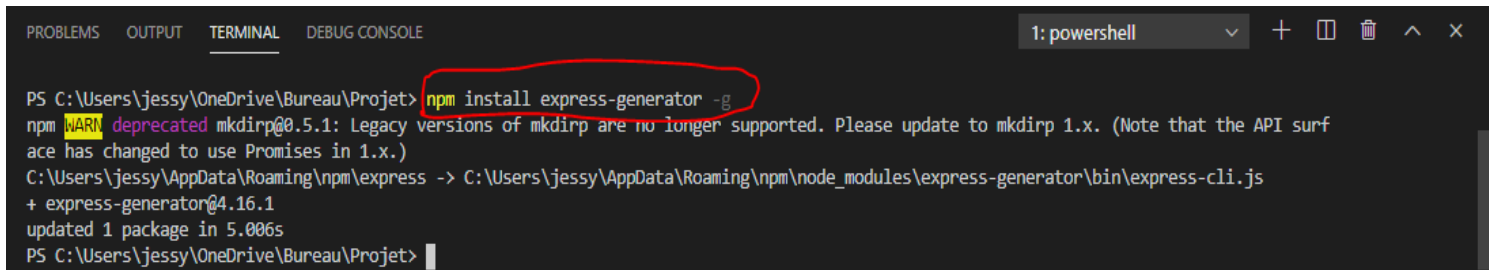
Comme **outils de développement**, j'ai utilisé **Visual Studio Code** et **phpMyAdmin** (application web de gestion pour les systèmes de gestion de base de données) à l'aide de **Laragon** (environnement de développement Web) pour gérer la base de données (B.D.D).

IV.2.Création du projet

Une fois Node.js installé, j'ai commencé par créer un projet à l'aide de **Express.js**.

Express.js est un framework pour node.js qui permet de créer des applications de façon simple. Il est basé sur le concept de **routing** et de **middleware** (fonction qui s'exécute les unes après les autres et qui peuvent accéder à l'objet request (req), response (res) et next).

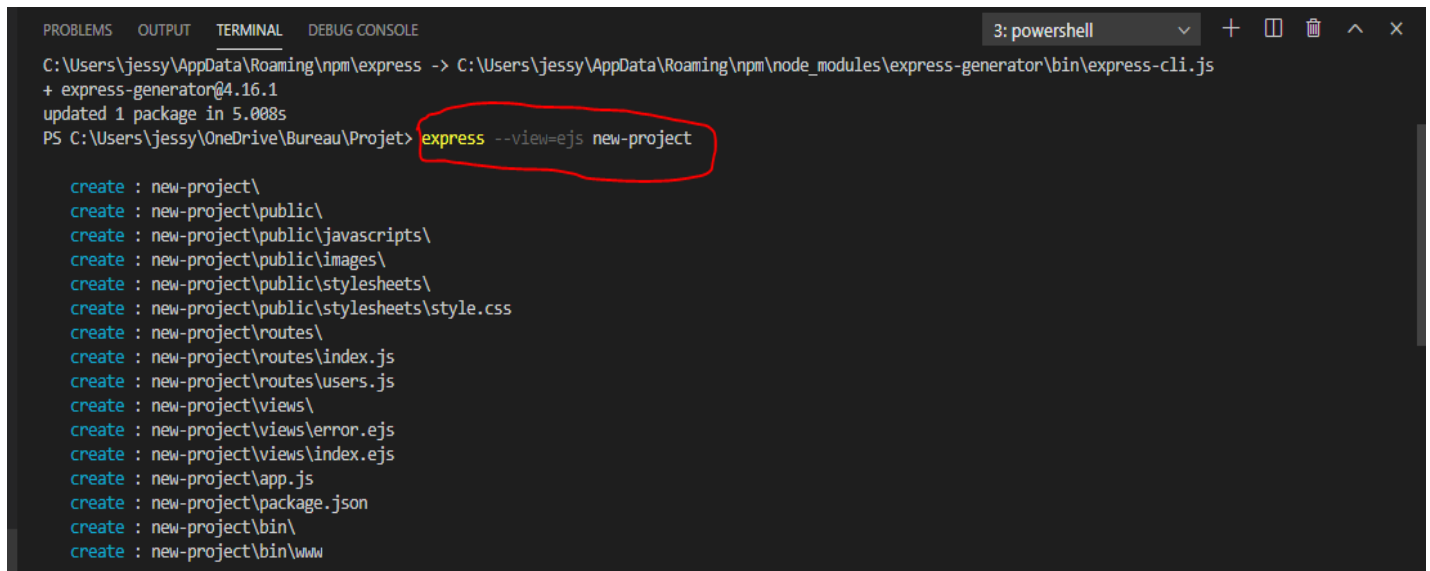
Express nous offre la possibilité de générer un nouveau projet déjà structuré, à l'aide de la commande : `npm install express-generator -g`



```
PS C:\Users\jessy\OneDrive\Bureau\Projet> npm install express-generator -g
npm WARN deprecated mkdirp@0.5.1: Legacy versions of mkdirp are no longer supported. Please update to mkdirp 1.x. (Note that the API surface has changed to use Promises in 1.x.)
C:\Users\jessy\AppData\Roaming\npm\express -> C:\Users\jessy\AppData\Roaming\npm\node_modules\express-generator\bin\express-cli.js
+ express-generator@4.16.1
updated 1 package in 5.006s
PS C:\Users\jessy\OneDrive\Bureau\Projet>
```

Une fois la commande exécutée, le générateur Express s'installe sous la forme d'un package global. Cela nous permet d'exécuter la seconde commande :

« `express --view=ejs + le nom du projet` » `view = ej`s signifie que nous utiliserons **EJS** pour gérer les templates (il en existe plein d'autres : Twig, Smarty, Hamlet, JSP, Jade...)



```
C:\Users\jessy\AppData\Roaming\npm\express -> C:\Users\jessy\AppData\Roaming\npm\node_modules\express-generator\bin\express-cli.js
+ express-generator@4.16.1
updated 1 package in 5.008s
PS C:\Users\jessy\OneDrive\Bureau\Projet> express --view=ejs new-project

create : new-project\
create : new-project\public\
create : new-project\public\javascripts\
create : new-project\public\images\
create : new-project\public\stylesheets\
create : new-project\public\stylesheets\style.css
create : new-project\routes\
create : new-project\routes\index.js
create : new-project\routes\users.js
create : new-project\views\
create : new-project\views\error.ejs
create : new-project\views\index.ejs
create : new-project\app.js
create : new-project\package.json
create : new-project\bin\
create : new-project\bin\www
```

Cela crée un nouveau projet Express dans le dossier new-project, lui-même placé dans le répertoire de travail courant.

Voici à quoi ressemble le projet une fois créé :

Le dossier **bin** contient le fichier exécutable qui démarre l'application. Il démarre le serveur (sur le port 3000) et configure une gestion des erreurs de base.

Le dossier **public** est le dossier accessible aux personnes qui se connectent à l'application. C'est ici que seront stockés le CSS, le JavaScript, et les images.

Le dossier **routes** est l'emplacement des fichiers de routeur.

Le dossier **views** est l'endroit où sont stockés les fichiers utilisés par le moteur de template.

Le dossier **node_modules** contient les bibliothèques installées à partir de npm.

Le **package-lock.json** se crée après l'exécution de la commande:

```
PS C:\Users\jessy\OneDrive\Bureau\Projet\new-project> npm install
```

Cette commande est très utile pour transporter notre projet d'un pc à un autre, car elle télécharge toutes les dépendances (bibliothèques) du projet dans le pc (attention, il faut être dans le répertoire de l'application).

app.js configure notre application express et assemble toutes les pièces ensemble, il est le cœur de l'application. Par exemple, Il contient le chargement des bibliothèques obligatoires (**require** = exiger) :

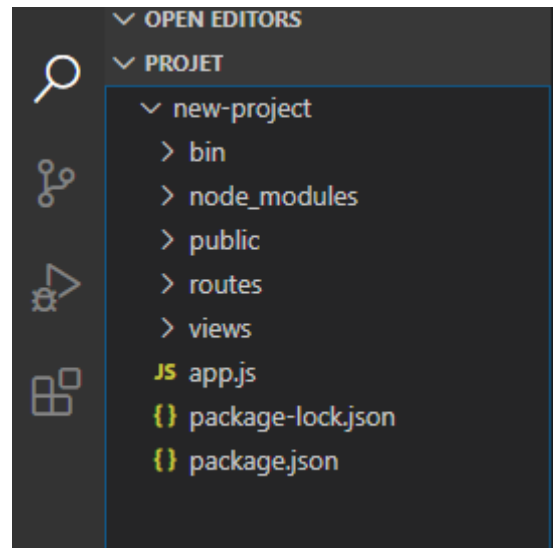
```
JS app.js  X
new-project > JS app.js > ...
1  var createError = require('http-errors');
2  var express = require('express');
3  var path = require('path');
4  var cookieParser = require('cookie-parser');
5  var logger = require('morgan');
6
```

Ou la création d'un objet app appelant la fonction express() :

```
10  var app = express();
```

Ou encore la configuration du moteur de vue, à l'aide de la bibliothèque **path**(chemin) :

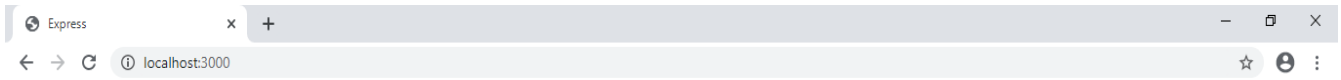
```
12  // view engine setup
13  app.set('views', path.join(__dirname, 'views'));
14  app.set('view engine', 'ejs');
```



Pour lancer l'application, il suffit d'entrer la commande suivante (attention, il faut être dans le répertoire de l'application):

```
PS C:\Users\jessy\OneDrive\Bureau\Projet\new-project> npm start
> new-project@0.0.0 start C:\Users\jessy\OneDrive\Bureau\Projet\new-project
> node ./bin/www
```

Cette commande lancera le fichier exécutable du dossier **bin**. L'application est prête à démarrer (elle écoute le port 3030):



Express

Welcome to Express

Voici comment la page s'est exécuté:

Dans le dossier **routes**, **index.js** et **users.js** sont créés lors de la création du projet.

Voici ce que contient le fichier **index.js** :

```
JS index.js x
new-project > routes > JS index.js > ...
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('index', { title: 'Express' });
7 });
8
9 module.exports = router;
10
```

res.render() : Compile le template et envoie la chaîne HTML au client.

router : Instance de la classe `express.Router()`, qui permet de créer des gestionnaires de route.

Dans le dossier **views**, **index.ejs** est aussi créé lors de la création du projet :

```
<> index.ejs x
new-project > views > <> index.ejs > ...
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title><%= title %></title>
5     <link rel='stylesheet' href='/stylesheets/style.css' />
6   </head>
7   <body>
8     <h1><%= title %></h1>
9     <p>Welcome to <%= title %></p>
10  </body>
11 </html>
12
```

La route (router) du fichier **index.js** est exporté et importé vers le cœur de l'application, **app.js** :

```
7 var indexRouter = require('./routes/index');

22 app.use('/', indexRouter);
```

app.use() permet d'utiliser la route (les routes si il y en a plusieurs) de **index.js** dans l'application.

IV.3. La balance

Avant de me focaliser sur le design, j'ai tout d'abord cherché une solution pour la balance, à savoir comment récupérer et afficher le poids. Ayant fait de longues recherches sur internet, je n'ai pas trouvé beaucoup d'informations. J'ai notamment essayé un logiciel (bill redirect) peu convaincant, qui par la suite je l'ai laissé tomber.

Node.js était la réponse : Parmi les nombreuses bibliothèques **NPM**, la bibliothèque **SerialPort** (permet d'accéder aux port séries) m'a beaucoup intéressée.

SerialPort :

```
const SerialPort = require('serialport')
const Readline = require('@serialport/parser-readline')
const port = new SerialPort('/dev/tty-usbserial1')

const parser = port.pipe(new Readline({ delimiter: '\r\n' }))
parser.on('data', console.log)
```

Premièrement j'ai installé **SerialPort** sur mon projet à l'aide de la commande :

```
PS C:\Users\jessy\OneDrive\Bureau\Projet\new-project> npm install serialport
```

Ensuite sur le fichier **index.js** j'ai importé la bibliothèque :

```
var express = require('express');
var router = express.Router();
//SerialPort
const SerialPort = require('serialport');
const Readline = require('@serialport/parser-readline');
//Attention à mettre le bon port.
const port = new SerialPort('COM24', { baudRate: 9600 });
const parser = port.pipe(new Readline({ delimiter: '\n' }));
```

Puis j'ai créé une autre route (méthode **GET**) qui a pour chemin **/balance** qui affiche le poids :

```
router.get('/balance', function(req, res, next) {
  parser.on('data', (data=>{
    res.send(data)
  })))
})
```

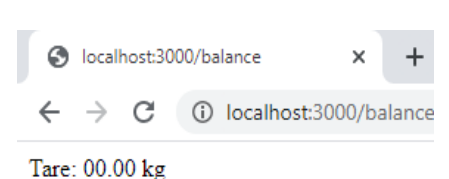
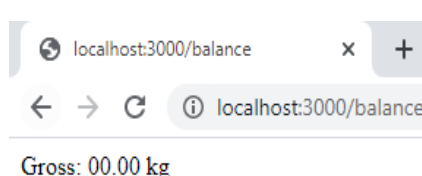
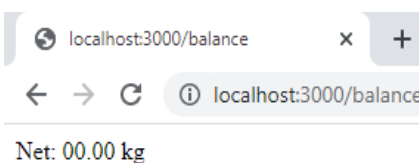
res.send() : Envoie une réponse sous forme de chaîne dans un format autre que JSON (XML, texte brut...)

Une fois la balance branchée, j'étais satisfait du résultat, car tout fonctionnait correctement.

Le poids s'affichait bien sur la route qui a pour chemin **/balance**.

Cependant la balance était composée de trois data :

- Le Net
- Le Gross
- Le Tarre

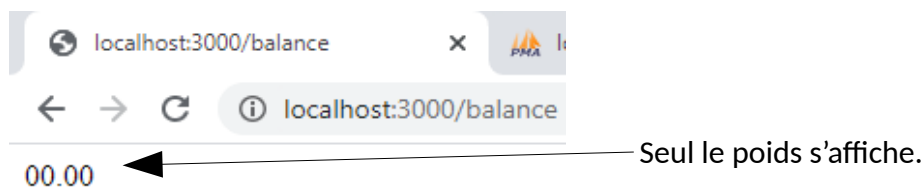


C'est trois data se répétaient sans cesse, je devais seulement conserver le poids du «Net» tout en supprimant le «Net :»

Après plusieurs jours de recherches, j'ai fini par trouver une solution intéressante :

```
router.get('/balance', function(req, res, next) {
  parser.on('data', (data) => {
    //Fonction split(): divise une chaîne de caractères à partir d'un séparateur,
    pour fournir un tableau de sous chaîne.
    let decoupe = data.split(":");
    //decoupe[0]: Retire tout ce qu'il y a après le séparateur ":"
    et conserve seulement "Tare","Net" et "Gross".
    //decoupe[1]: Retire tout ce qu'il y a avant le séparateur ":"
    et conserve seulement le poids du Tare, Net et Gross +"kg".
    let decoupe2 = decoupe[1].split(" ")
    //decoupe2[0]: Retire tout ce qu'il y'a après le séparateur " " /conserve
    seulement le poids en retirant le "kg".
    //decoupe2[0]: Contient le "kg".
    //Condition: Si decoupe[0] est égale à Net,
    alors on affiche seulement le decoupe2[0] (poids) du Net, et non les 2 autres.
    if (decoupe[0] === "Net" ){
      res.send(decoupe2[0])
    }
  });
});
```

Voici le résultat :



IV.4. Connexion à la B.D.D

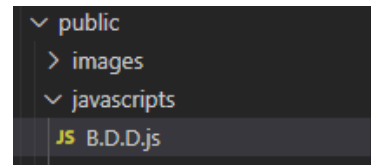
Une fois le code de la balance terminé, je me suis concentré sur la connexion entre mon application et la base de données (B.D.D). Mon collègue qui s'occupait de la B.D.D, me la transmise par clé USB, afin que je puisse travailler dessus.

J'ai utilisé l'interface web **phpMyAdmin** (ouvert à partir de **Laragon**) pour gérer la base.

NPM propose la bibliothèque **mysql**, qui permet de nous connecter très facilement à la B.D.D. Tout d'abord j'ai installé la bibliothèque **mysql** sur mon projet :

```
PS C:\Users\jessy\OneDrive\Bureau\Projet\new-project> npm install mysql
+ mysql@2.18.1
added 3 packages from 11 contributors and audited 280 packages in 11.213s
```

Dans le dossier Javascripts, j'ai créé un fichier **B.D.D.js** :
Ce fichier, permet de gérer la connexion.



Le fichier **B.D.D.js** contient la connexion :

```
const mysql = require('mysql')
var connection = mysql.createConnection({
  host      : 'localhost',
  user      : 'root',
  password  : '',
  database  : 'aeroport_db'
});
connection.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
module.exports = connection;
```

j'ai importé la bibliothèque **mysql** dans le fichier, à l'aide de **require**.

Attention à mettre correctement le **mdp** et le nom de la **B.D.D** dans la connexion

Le **module.exports** permet d'exporter cette connexion vers d'autres fichiers.

La condition dans la fonction **.connect** permet de vérifier si la connexion à fonctionné ou non.

Pour vérifier la connexion, je l'ai importé vers l'application **app.js**:

```
var mysql = require ("./public/javascripts/B.D.D")
```

Une fois l'application lancée, la connexion fonctionne correctement :

```
PS C:\Users\jessy\OneDrive\Bureau\Projet\new-project> npm start
> new-project@0.0.0 start C:\Users\jessy\OneDrive\Bureau\Projet\new-project
> node ./bin/www
Connected!
```

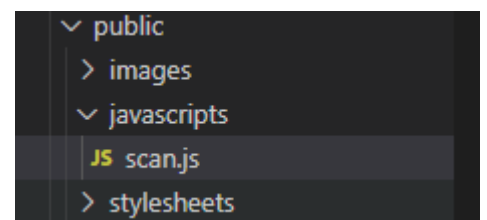
IV.5.Template N°1: Scanner

Une fois le code de la balance terminé et la connexion réalisé, je me sentais prêt à avancer sur le design de l'application.

Tout d'abord, j'ai supprimé tout ce que contenait le template de la racine (/) de l'application (index.ejs), pour pouvoir construire la première page (page d'accueil) de l'application.

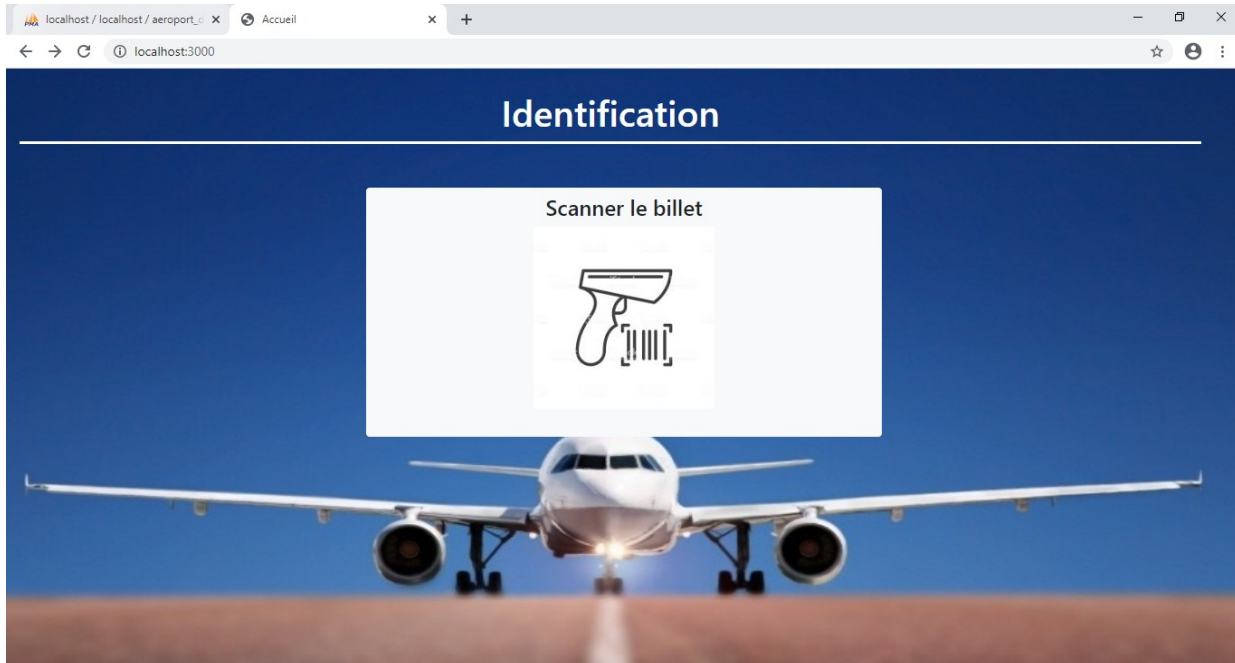
J'ai créé un fichier **scan.js** dans le dossier **javascripts** :

Dans ce fichier, j'ai mis tout le code javascript (jQuery) utilisé pour la page **index.ejs** (page d'accueil).



Sur la view **index.ejs**, j'ai lié le script **scan.js**, ajouté la bibliothèque **jQuery** et le framework **bootstrap**.

J'ai réalisé une page d'accueil assez simple mais bien structurée :

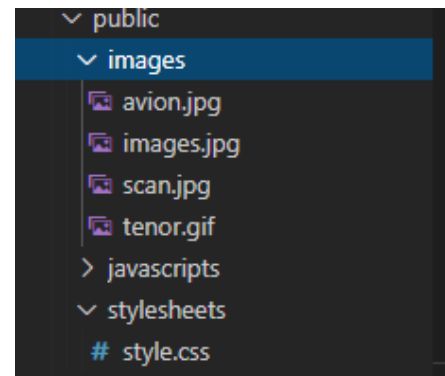


Toutes les images sont stockés dans le dossier images :

J'ai aussi lié un fichier css (style.css) à tous les templates.
Ce fichier permet d'insérer une image de fond et gérer le style de l'entête :

```
body{
  background-image:url(/images/avion.jpg);
  background-repeat: no-repeat;
  background-attachment: fixed;
  background-size: cover;
  background-size: 100% 100%;
}

#contener{
  color: white;
  text-align: center;
}
```



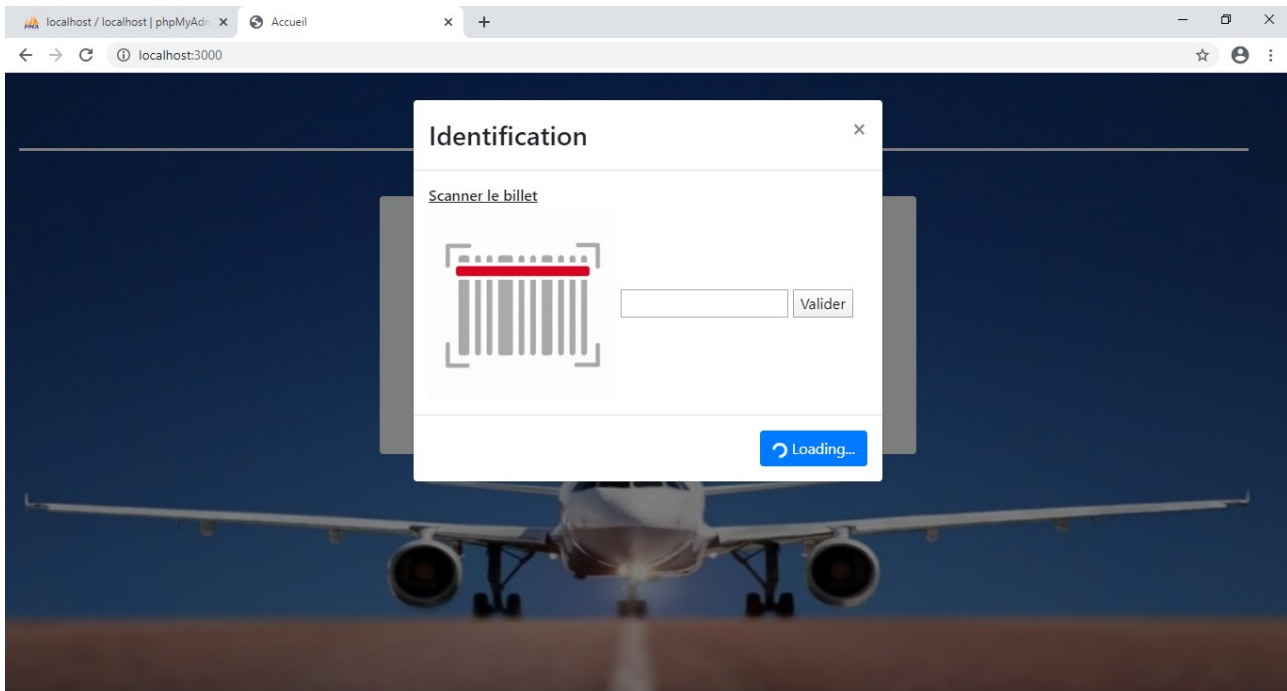
Voici l'entête de la page:

```
<!--En tête-->
<div class="container-fluid row" id="contener" >
  <div class="col-md-12 col-sm-12 col-lg-12 col-ld-12">
    <div style="border-bottom: solid;"><h1>Identification</h1> </div>
  </div>
</div>
```

Dans **Bootstrap** :

- 1 row = 12 col
- container-fluid** = Toute la largeur de la fenêtre.
- md, sm...** = taille d'écran.

La div du milieu affiche une **page modal** (fenêtre qui apparaît au-dessus de la page), une fois cliqué dessus :



J'ai réalisé cette page modal (Il en existe plusieurs modèles) à l'aide de **Bootstrap**. Dans la page modale, j'ai inséré un **input** qui permet d'afficher le numéro du billet scanné par la douchette et un **button** «valider» qui permet de vérifier si le numéro existe ou pas dans la base de données.

J'ai donc copié le code de la page modal à partir du site de **Bootstrap**, puis collé le code sur mon fichier **index.ejs** :

```
<!--Div Scanner le billet-->
  <div class="row">
    <div class="col-md-3"></div>
  <div class="col-md-6 col-sm-6 btn btn-light"
data-toggle="modal" data-target="#ModalPeser" id="num">
    <h4 class="text-center">Scanner le billet</h4>
    
    <br>
  </div>
<div class="col-md-3"></div>
```

data-target: lie la page modal à la div, à l'aide de l'Id de la page modal.

data-toggle: déclencheur

```

<!--Page modal scanner-->
<div class="modal fade" id="ModalPeser" tabindex="-1" role="dialog"
  aria-labelledby="exampleModalLongTitle" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h3 class="modal-title" id="exampleModalLongTitle">Identification</h3>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span>
        </button>
      </div>
      <div class="modal-body">
        <h6 style="text-decoration: underline;"> Scanner le billet</h6>
        <img src ="images/tenor.gif" style="width: 200px; text-align: center;" >
        <!--Input-->
        <input type="text" id="txt">
        <!--Button-->
        <button id="btn">Valider</button>
      </div>
    </div>
  </div>

```

Au pied de page de la page modal, j'ai créé un bouton «loading» (spinner) :



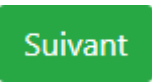
J'ai ajouté deux autres boutons en dessous de celui-ci :

```

<!--Loading-->
  <div class="modal-footer" id = "FOOTER_IDENTIFICATION_LOADING">
    <button class="btn btn-primary" id="BTN_IDENTIFICATION1" >
      <span class="spinner-border spinner-border-sm"></span>
      Loading...
    </button>
  </div>
  <!--Réessayer-->
  <div class="modal-footer" id = "FOOTER_IDENTIFICATION_REESAYER">
    <button class="btn btn-danger " id="BTN_IDENTIFICATION2" >
      <span class="glyphicon glyphicon-asterisk"></span>
      Réessayer
    </button>
  </div>
  <!--Suivant-->
  <div class="modal-footer" id = "FOOTER_IDENTIFICATION_SUIVANT">
    <button class="btn btn-success" id="BTN_IDENTIFICATION3" >
      Suivant
    </button>
  </div>

```

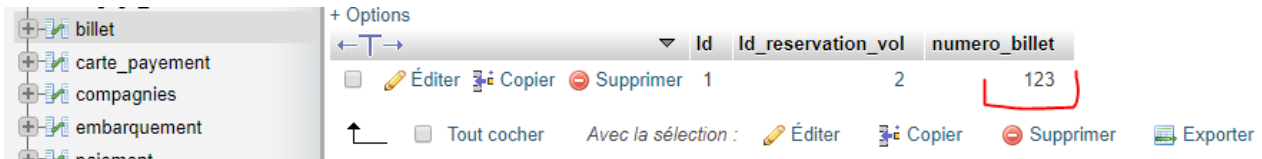
Un bouton Réessayer :  Il s'affiche si le numéro du billet est introuvable dans la base de données.

Un bouton Suivant :  Il s'affiche si le numéro du billet est identifié dans la base de données.

Ces deux boutons sont d'abord cachés à l'aide de la fonction **hide()**, dans le fichier **scan.js**:

```
//cache les boutons
$('#FOOTER_IDENTIFICATION_REESSAYER').hide();
$('#FOOTER_IDENTIFICATION_SUIVANT').hide();
```

Pour pouvoir identifier le voyageur, l'opérateur doit d'abord vérifier si le numéro du billet (E-ticket) existe dans la **table** «billet» de la B.D.D :



Id	Id_reservation_vol	numero_billet
1	2	123

Je me suis aidé d'**AJAX** pour envoyer la valeur du input (numéro scanné) vers le serveur, qui par la suite exécute une requête **SQL** et envoie le résultat vers AJAX, qui vérifie si le numéro existe ou pas.

La fonction AJAX se lance lorsque le bouton «valider» est enclenché, voici le code sur le fichier **scan.js**:

```
//btn valider
$('#btn').click(function(){
    //let a: récupère la valeur du input.
    let a = $('#txt').val()
    //Fonction ajax
    $.ajax({
        url: "/recup", //route visé.
        method: "POST", //choix de la méthode.
        data: 'recup=' + a //variable recup: prend la valeur de la variable a (valeur du input)
                        pour être exploité côté serveur.
    }).done(function (data) { //récupère la valeur envoyée par le serveur pour ensuite l'exploiter.
        //Condition: Si le data est vide, alors le bouton Réessayer s'affiche (show())
        et les deux autres se cachent (hide()).
        if (data == ""){
            $('#FOOTER_IDENTIFICATION_LOADING').hide();
            $('#FOOTER_IDENTIFICATION_REESSAYER').show();
            $('#FOOTER_IDENTIFICATION_SUIVANT').hide()
            //Sinon si le data contient des données,
            alors le bouton Suivant s'affiche et les deux autres se cachent
        } else{
            $('#FOOTER_IDENTIFICATION_SUIVANT').show()
            $('#FOOTER_IDENTIFICATION_LOADING').hide();
            $('#FOOTER_IDENTIFICATION_REESSAYER').hide();
        }
    })
})
```

Dans le fichier **index.js** j'ai importé la connexion avec la B.D.D (depuis le fichier B.D.D.js) afin de pouvoir l'interroger:

```
var mysql = require ("./public/javascripts/B.D.D")
```

La bibliothèque **mysql** permet de réaliser des requêtes **SQL** à l'aide de la fonction **.query()**, voici le code de la route qui a pour chemin **/recup** (méthode **POST**) dans le fichier **index.js** :

```
//Verification numéro de billet
router.post('/recup',function(req,res,next){
  //var a: récupère la valeur de la variable recup (numéro du billet), envoyé à partir de AJAX.
  var a = req.body.recup;
  //requête SQL: sélectionne toutes les données (table billet)
  du voyageur qui a pour numéro de billet, la valeur du input.
  mysql.query('SELECT * FROM billet WHERE numero_billet = ' + a,(req, row)=>{
    //envoie le résultat de la requête vers le client,
    récupéré par AJAX à l'aide de la fonction done().
    res.send(row)
    console.log(row)
  })
})
```

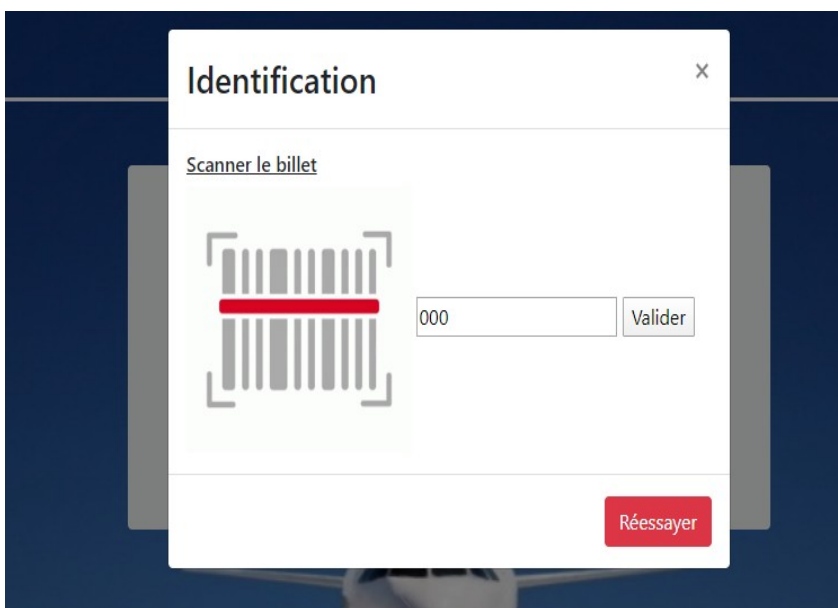
Voici le résultat du **console.log(row)**, pour le voyageur qui a pour numéro de billet «123» :

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
[ RowDataPacket { Id: 1, Id_reservation_vol: 2, numero_billet: 123 } ]
```

Les données sont en format **JSON**.

Donc une fois le bouton «valider» enclenché, AJAX vérifiera si le **data** reçu est vide ou contient des données (au format JSON) .

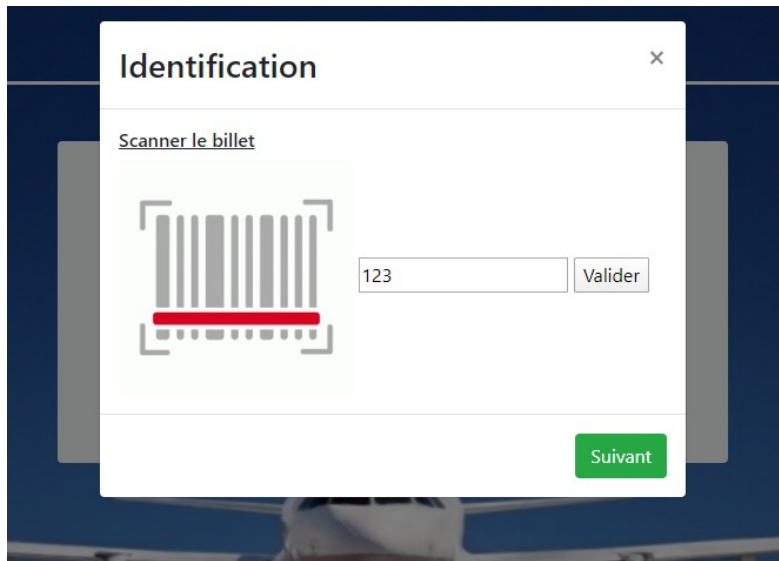
Exemple en cas d'erreur (le data est vide):



Le bouton «Réessayer» permet seulement d'être remplacé par le bouton «Loading» et de vider le input pour permettre une nouvelle entrée:

```
//Réessayer à loading
$('#BTN_IDENTIFICATION2').click(function(){
  $('#FOOTER_IDENTIFICATION_REESAYER').hide()
  $('#FOOTER_IDENTIFICATION_LOADING').show();
  //vide le input
  $('#txt').val("")
});
```

Exemple en cas de réussite (le data contient des données):



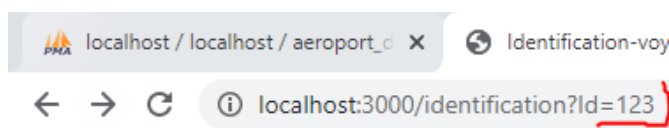
Le bouton «Suivant» est quant à lui plus complexe, car il gère la redirection. Pour garder la trace du voyageur lors de la redirection, j'ai décidé de mettre le numéro de son billet en paramètre d'URL.

Pour cela j'ai repris la même fonction AJAX que pour le bouton «valider», afin d'obtenir les informations du voyageur de la table «billet». Ensuite j'ai seulement exploité la donnée qui m'était utile: `data[0].numero_billet`.

Voici le code:

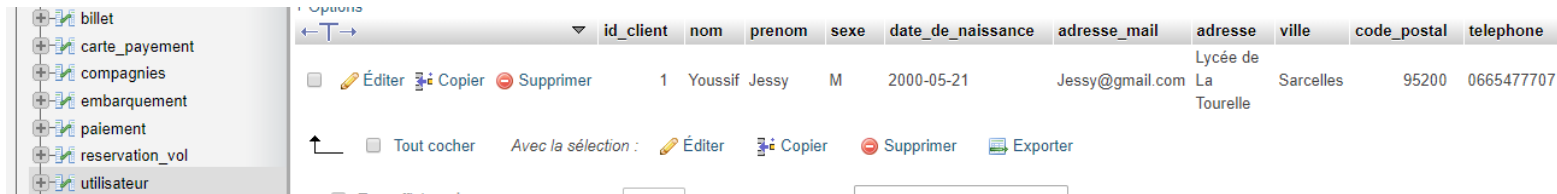
```
//btn suivant
$('#BTN_IDENTIFICATION3').click(function(){
  //let a: récupère la valeur du input.
  let a = $('#txt').val()
  //fonction AJAX
  $.ajax({
    url: "/recup",
    method: "POST",
    data: 'recup=' + a
  }).done(function (data) {
    //redirection vers la page suivante (identification) en attribuant le paramètre Id à l'URL
    //qui a pour valeur le numéro de billet.
    location.href = "/identification?Id="+ data[0].numero_billet
  })
});
```

Une fois le bouton «suivant» enclenché, la redirection vers la route qui a pour chemin **/identification** s'effectue, tout en ajoutant à l'URL le numéro de billet en paramètre (Id):



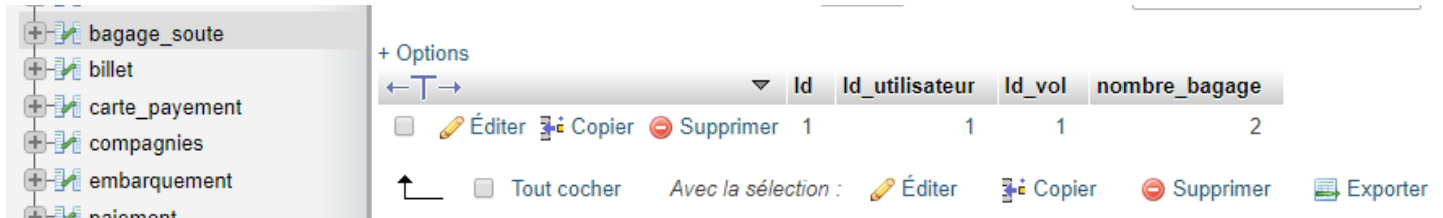
IV.6.Template N°2: Identification

J'ai créé une nouvelle view (dans le dossier **views**) nommée **identification.ejs**. Dans cette page, j'ai affiché les données du voyageur (nom, prénom, adresse...) récupéré à partir de la **table** «utilisateur» de la B.D.D :



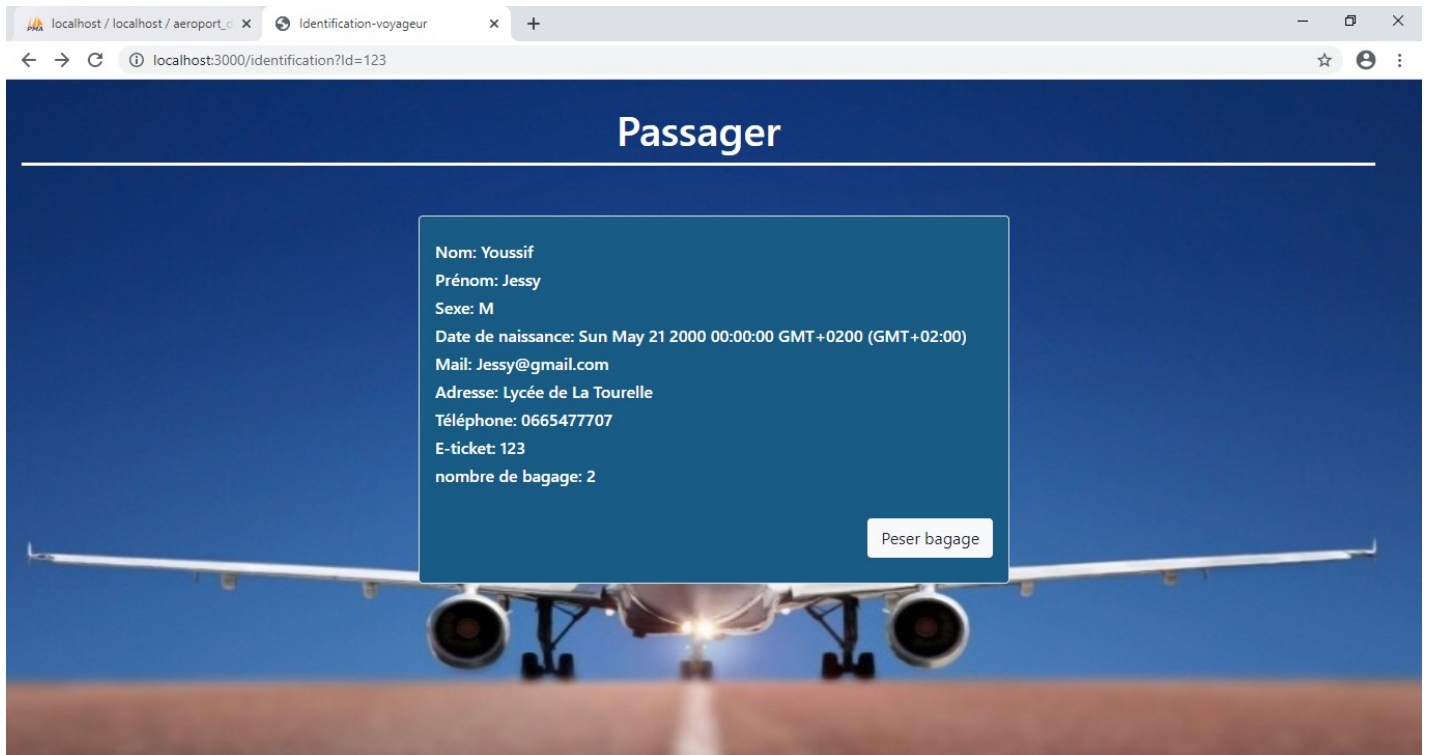
	id_client	nom	prenom	sexe	date_de_naissance	adresse_mail	adresse	ville	code_postal	telephone
	1	Youssif	Jessy	M	2000-05-21	Jessy@gmail.com	Lycée de La Tourelle	Sarcelles	95200	0665477707

Mais aussi le nombre de bagages en soute du voyageur à partir de la **table** «bagage_soute» :

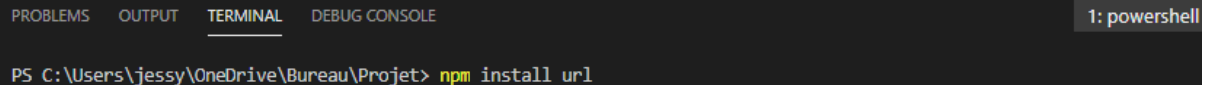


	Id	Id_utilisateur	Id_vol	nombre_bagage
	1	1	1	2

J'ai réalisé un design assez simple, à l'aide de Bootstrap:



Pour afficher les données du voyageur, je me suis servi de la bibliothèque **URL**:



```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 1: powershell
PS C:\Users\jessy\OneDrive\Bureau\Projet> npm install url
```

Après l'avoir installé, je l'ai ensuite importé au fichier **index.js**: `var url = require('url');`

Voici la route qui a pour chemin **/identification** dans le fichier **index.js**:

```
router.get('/identification', function(req, res, next) {
  //var pathname: contient le query string de l'URL (pour récupérer le numéro
  de billet du voyageur)
  var pathname = url.parse(req.url, true).query;
  //requête SQL: Sélectionne toutes les informations (table utilisateur) et
  le nombre de bagages en soute (table bagage_soute) du voyageur à l'aide
  du numéro de billet (pathname.Id).
  mysql.query('SELECT utilisateur.nom,utilisateur.prenom, utilisateur.sexe,
  utilisateur.date_de_naissance, utilisateur.adresse_mail,utilisateur.adresse,
  utilisateur.telephone, billet.numero_billet, bagage_soute.nombre_bagage
  from utilisateur,billet, bagage_soute,reservation_vol
  WHERE reservation_vol.Id_utlisateur = utilisateur.id_client
  AND billet.numero_billet = '+ pathname.Id ,(req, row)=>{
    //renvoie au client la view identification.ejs + la variable row qui
    contient le résultat de la requête SQL (row) au format JSON.
    res.render('identification', { row :row});
    console.log(row)
  });
});
```

Voici le `console.log(row)`, soit le résultat de la requête SQL:

```
[
  RowDataPacket {
    nom: 'Youssif',
    prenom: 'Jessy',
    sexe: 'M',
    date_de_naissance: 2000-05-20T22:00:00.000Z,
    adresse_mail: 'Jessy@gmail.com',
    adresse: 'Lycée de La Tourelle',
    telephone: '0665477707',
    numero_billet: 123,
    nombre_bagage: 2
  }
]
GET /identification?Id=123 200 320.342 ms - 3887
```


La route qui a pour chemin **/identification** envoie (**render**) au client la view **identification.ejs**, mais aussi la variable «row» qui contient les données du voyageur.

Grâce à **EJS**, j'ai exploité cette variable dans le code **HTML**. Pour intégrer des variables EJS dans le code **HTML**, il faut respecter la syntaxe suivante: `<%= variable %>` .

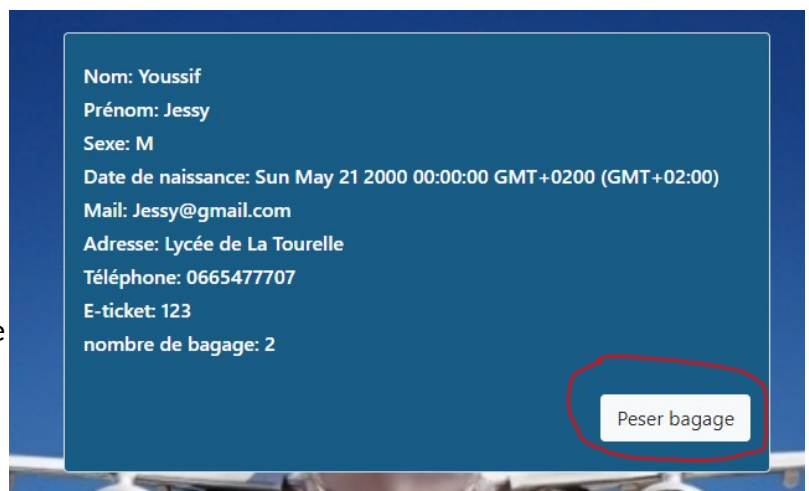
Voici le code de la **div** où sont stockées les informations du voyageur, dans le fichier **identification.ejs**:

```
<div class="row">
  <div class="col-md-3 col-sm-3 col-lg-3 col-ld-3"></div>
  <div class="col-md-6 col-sm-6 border col-lg-6 col-ld-6 rounded"
style="background-color:rgb(24, 91, 133); color: white;">
  <section>
    <br>
    <h6>Nom: <%= row[0].nom %> </h6></span>
    <h6>Prénom: <%= row[0].prenom %> </h6>
    <h6>Sexe: <%= row[0].sexe %> </h6>
    <h6>Date de naissance: <%= row[0].date_de_naissance %> </h6>
    <h6> Mail: <%= row[0].adresse_mail %> </h6>
    <h6>Adresse: <%= row[0].adresse %> </h6>
    <h6>Téléphone: <%= row[0].telephone %> </h6>
    <h6>E-ticket: <%= row[0].numero_billet %> </h6>
    <h6 id="nbre_bag">nombre de bagage: <%= row[0].nombre_bagage %></h6><br>
    <div class="text-right">
      //button:enclenche une page modal.
      <button type="button" class="btn btn-light" data-toggle="modal"
data-target='#m'>Peser bagage</button>
    </div>
  </section>
</div>
```

J'ai créé le bouton «Peser bagage» :

Ce bouton est lié à une page modal par le **data-target** (Id de la page modal).

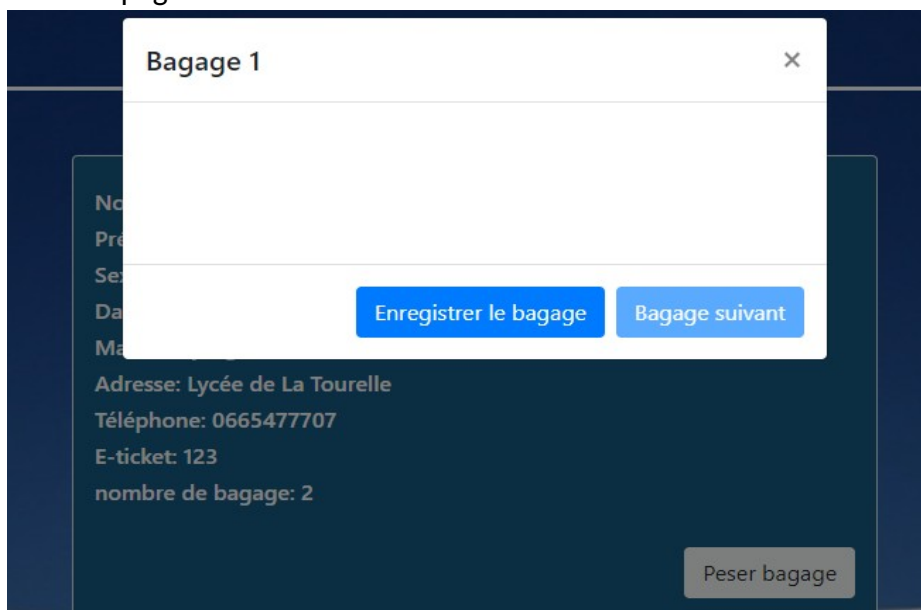
Cette page modal permet de gérer le pesage des bagages et la redirection vers le paiement une fois le pesage terminé.



Voici le code de la page modal déclenché par le bouton «Peser bagage»:

```
<!--Page modal peser bagage-->
<div class="modal fade" id="m" tabindex="-1" role="dialog"
aria-labelledby="exampleModalLongTitle" aria-hidden="true">
  <div class="modal-dialog" role="document">
    <div class="modal-content">
      <div class="modal-header">
        <h5 class="modal-title" id="exampleModalLongTitle_Voyageur">Bagage 1 </h5>
        <button type="button" class="close" data-dismiss="modal" aria-label="Close">
          <span aria-hidden="true">&times;</span></button>
      </div>
      <div class="modal-body">
        <p id="val"></p>
        <br><br><br>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-primary"
id="BTN_ENREGISTREMENT_VOYAGEUR">Enregistrer le bagage</button>
        <button type="button" class="btn btn-primary disabled"
id="BTN_SUIVANT_VOYAGEUR">Bagage suivant</button>
        <button type="button" class="btn btn-primary" id="BTN_PAYER_VOYAGEUR">
Passer au paiement
</button>
      </div>
    </div>
  </div>
</div>
</div>
```

La page modal ressemble à ceci:



Le bouton «Enregistrer le bagage» permet de vérifier si le bagage est supérieur au poids maximal (10 kg).

Le bouton «Bagage suivant» permet de passer au bagage suivant. Il est d'abord bloqué (disabled) car le bagage doit être enregistré avant de passer au suivant .

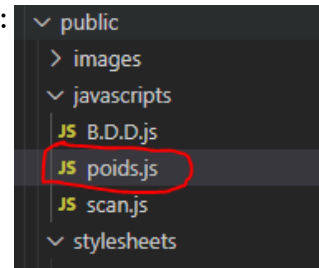
J'ai aussi créé un bouton «Passer au paiement», qui est caché pour le moment.

Une fois tous les bagages pesés, il viendra remplacer le bouton «Bagage suivant».

Pour ne pas tout mélanger, j'ai créé et lié un fichier **javascript** pour chaque page (view).
Pour cette page, j'ai créé le fichier **poids.js** qui gère le javascript (jQuery):

Dans ce fichier, j'ai d'abord caché le bouton «Passer au paiement»:

```
$('#BTN_PAYER_VOYAGEUR').hide();
```



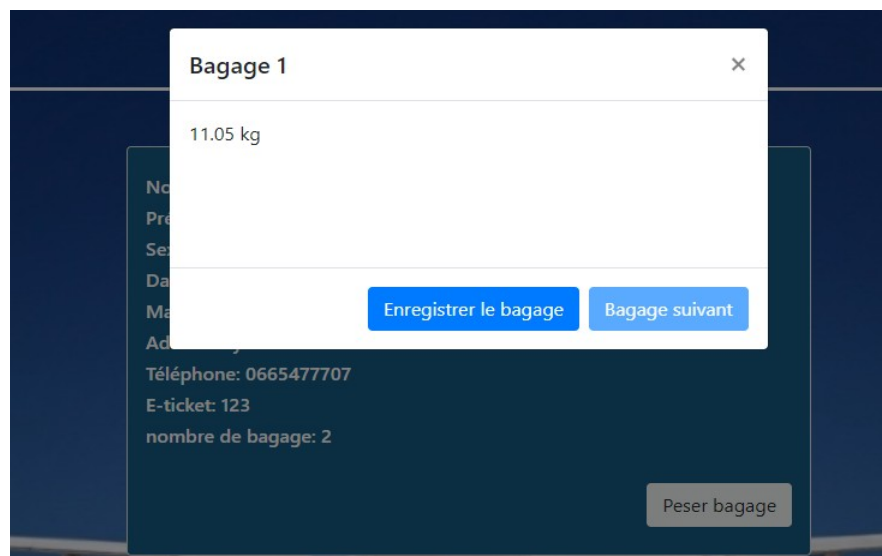
Avant de programmer ces trois boutons (enregistrer, suivant, paiement), j'ai d'abord affiché le poids de la balance (récupéré précédemment) à l'intérieur du **modal-body**, dans la balise **<p>** qui a pour **id** «val»:

```
</button>
</div>
<div class="modal-body">
  <p id="val"></p>
  <br><br><br>
</div>
```

Pour cela, j'ai ajouté un **script** en dessous du code HTML, dans le fichier **identification.ejs**:

```
<script>
//Mettre à jour le poids toutes les 150 Millisecondes, à l'aide de la fonction setInterval().
setInterval(getmessage, 150);
//fonction AJAX : récupère la valeur rendue (res.send) de la route /balance (soit le poids).
function getmessage() {
$.ajax({
  url: "/balance",
  method: "GET"
}).done(function (data) {
  //affiche la valeur (le poids) + " kg" dans la page modal.
  $("#val").html(data + " kg")
})
}
</script>
```

Le poids s'affiche bien dans modal-body de la page modal et se met à jour rapidement:



Une fois le poids affiché, j'ai programmé les boutons de cette page modal.
Dans le fichier **poids.js** j'ai commencé par créer trois variables:

```
//numéro du bagage
var bag = 1;
//somme à payer
var somme = 0;
//retire le "nombre de bagage:" et garde que le chiffre à l'aide la fonction split()
let nbr = $('#nbre_bag').html().split(": ");
```

La variable «bag», permet d'afficher dans la page modal le numéro de bagage qu'il s'agit:



La variable «somme», permet de faire la somme de tous les bagages supérieur à 10 kg.
Et enfin la variable «nbr», permet de garder seulement le nombre de bagages :



Voici le fonctionnement du bouton «Enregistrer le bagage»:

```
//Bouton enregistrement
$('#BTN_ENREGISTREMENT_VOYAGEUR').click(function(){
//Condition qui permet d'enregistrer une seule fois le bagage
if ($('#BTN_ENREGISTREMENT_VOYAGEUR').attr('class') === "btn btn-primary"){
//vérifie si le poids est supérieur à 10kg. Si c'est vrai alors somme
s'incrémente de 10 (10€).
if (($("#val").val()) > 10){somme = somme + 10}
//vérifie si la variable bag est égale au nombre de bagages du voyageur,
afin de cacher le bouton «suivant» et afficher le bouton «paiement».
if(bag == nbr[1]){
$('#BTN_SUIVANT_VOYAGEUR').hide()
$('#BTN_PAYER_VOYAGEUR').show()
}
//Si le bouton «enregistrement» est activé (primary), alors il se désactive (disabled)
et débloquent (primary) le bouton «suivant». Sinon une alerte s'affichera.
if ( $('#BTN_ENREGISTREMENT_VOYAGEUR').attr('class') === "btn btn-primary"){
$('#BTN_ENREGISTREMENT_VOYAGEUR').attr('class','btn btn-primary disabled')
$('#BTN_SUIVANT_VOYAGEUR').attr('class','btn btn-primary')
}else { alert('bagage déjà enregistrer')}
})
```

Une fois cliqué sur ce bouton, une condition vérifiera si le bagage est déjà enregistré ou pas, puis s'il est supérieur ou non à 10 kg. Une autre condition vérifiera s'il faut passer au bagage suivant ou passer au paiement. Et enfin une autre condition permettra de bloquer ce bouton (en disabled) et libérer le bouton suivant (en primary). Si ce bouton est déjà bloqué, une alerte s'affichera.

Exemple pour un bagage supérieur à 10 kg:

Bagage 1

11.25 kg

Enregistrer le bagage

Bagage suivant



Bagage 1

11.25 kg

Enregistrer le bagage

Bagage suivant

Le changement des boutons s'effectue bien, et la variable «somme» s'incrémente correctement :

```
somme = 10
```

poids.js:47

Une alerte s'affiche si l'on réappuie sur le bouton:

Bagage 1

11.25 kg

Enregistrer le bagage

Bagage suivant

localhost:3000 indique
bagage deja enregistrer

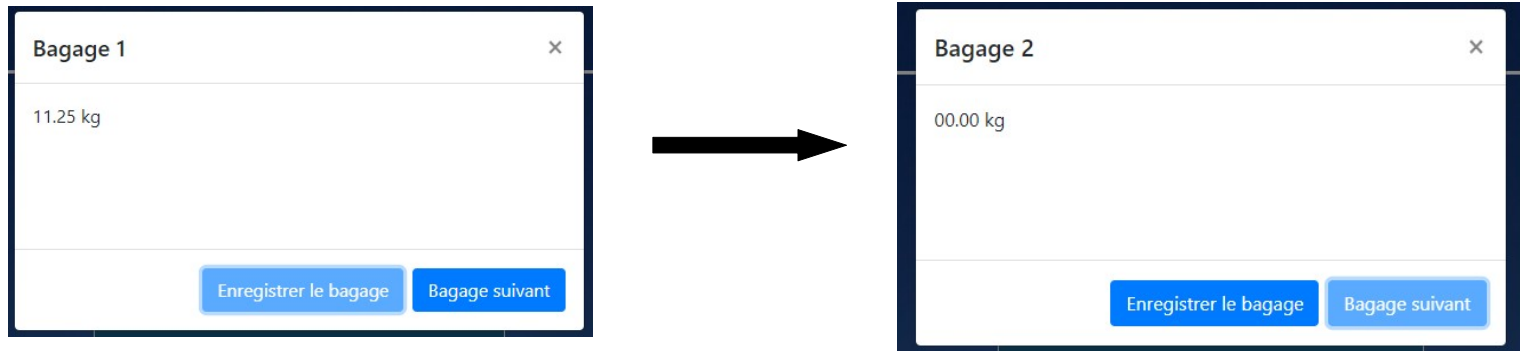
OK

Le fonctionnement du bouton «Bagage suivant» est plus simple:

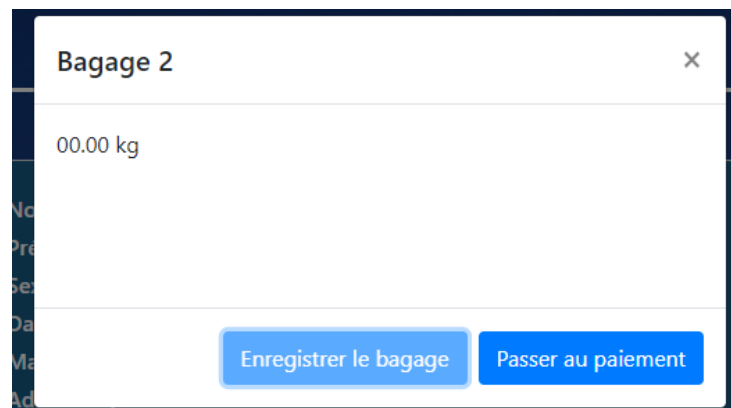
```
//Bouton suivant
$('#BTN_SUIVANT_VOYAGEUR').click(function(){
//Condition qui vérifie si le bouton est désactivé (disabled)
if ( $('#BTN_SUIVANT_VOYAGEUR').attr('class') === "btn btn-primary"){
//incrémentation du numéro de bagages
bag = bag+1
// modifie le numéro du bagage sur la page modal
$('#exampleModalLongTitle_Voyageur').html("Bagage " + bag)
//le bouton se désactive (disabled) et active le bouton enregistrement (primary)
$('#BTN_ENREGISTREMENT_VOYAGEUR').attr('class','btn btn-primary')
$('#BTN_SUIVANT_VOYAGEUR').attr('class','btn btn-primary disabled')
}
})
```

Une fois ce bouton enclenché, une condition vérifie si le bouton est activé ou pas.
Si le bouton est activé, la variable «bag» s'incrémente (bagage suivant) et s'affiche sur la page modal. Le bouton se désactive (disabled) et active le bouton enregistrement (primary).

Exemple:



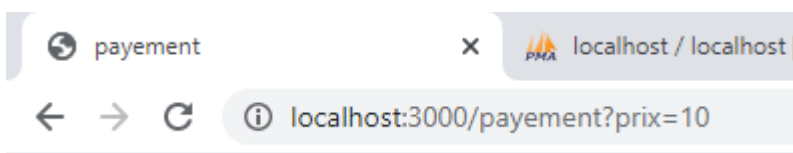
Une fois le dernier bagage enregistré, le bouton «Passer au paiement» remplace le bouton «Bagage suivant»:



Ce bouton gère une redirection vers une nouvelle route qui a pour chemin **/payement**, ajoutant en paramètre d'URL, la valeur de la somme:

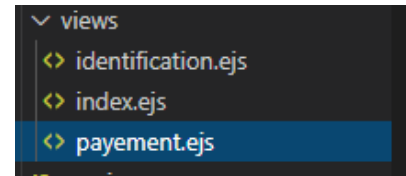
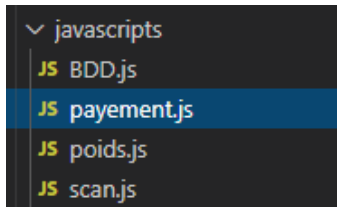
```
//Bouton payer
$('#BTN_PAYER_VOYAGEUR').click(function(){
  location.href = "/payement?prix="+somme
})
```

Exemple pour un seul bagage supérieur à 10 kg (soit 10€):



IV.7.Template N°3: Paiement

Dans le dossier **views**, j'ai créé une dernière view, **payement.ejs**.
J'ai aussi créé un fichier **payement.js** pour gérer le JavaScript de cette page:



Cette dernière page gère le paiement.

Avant de commencer la réalisation de cette page, j'ai d'abord créé une table **carte_paiement** dans la B.D.D:

<

Cette carte est rattaché à un voyageur par la **clé étrangère** «id_utilisateur».

Elle possède un numéro et un montant.

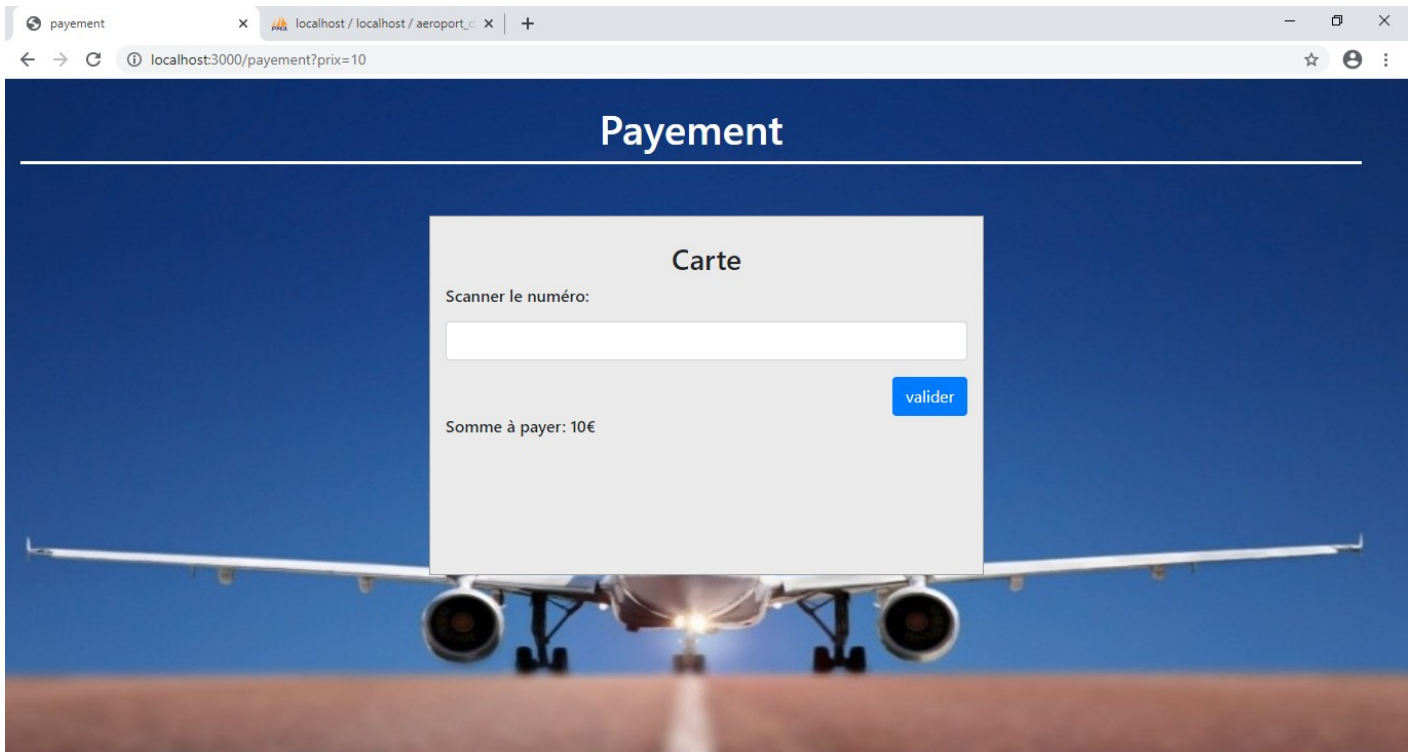
Pour qu'elle soit reconnue, l'opérateur scannera le numéro de la carte.

Voici le code de la route qui a pour chemin **/paiement** dans le fichier **index.js**:

```
//paiement
router.get('/paiement', function(req, res, next) {
  //var pathname: contient le query string de l'URL (montant de la somme)
  var pathname = url.parse(req.url, true).query;
  //renvoie au client la view paiement.ejs + la variable prix (soit le montant de la somme)
  res.render('paiement',{prix : pathname.prix})
})
```

La variable «prix» permet d'exploiter la valeur du montant de la somme, vers le côté client à l'aide d'EJS.

Voici la view, **payement.ejs** qui est renvoyé (render) par cette route :



Dans cette page j'ai créé un cadre à l'aide du **css** (dans le fichier style.css), dans lequel j'ai ajouté un **input** pour afficher le numéro de la carte scanné. J'ai aussi affiché la somme à payer à l'aide d'**EJS** et un **bouton** «valider» qui permet d'afficher le solde de la carte, et vérifier si le solde est suffisant ou non:

```
<div>
  <br>
  <h3 class="text-center">Carte</h3>
  <div class="form-group">
    <label for="username"><h6>Scanner le numéro: </h6></label><br>
    <input type="text" name="username" id="username" class="form-control">
  </div>
  <div class="text-right">
    //bouton valider
    <button class="btn btn-primary" id="btnValider">valider</button>
    //montant de la somme
    <h6 id="somme" class="text-left">Somme à payer: <%= prix %>€</h6>
    //montant du solde
    <h6 id="solde" class="text-left"></h6>
  </div>
  <br>
</div>
```

Le solde du compte s'affichera dans la balise **<h6>** qui a pour **id** «solde», une fois le numéro de la carte validé.

Dans ce cadre j'ai créé deux solutions possibles :

- En cas de solde suffisant : un bouton s'affichera pour procéder au paiement :

Payer

- En cas de solde insuffisant : une alerte s'affichera :

Solde insuffisant

Ces deux cas, sont placés au bas du cadre :

```
<button class="btn btn-success" id="payer" data-toggle="modal" data-target="#myModal">Payer</button>
<div class="alert alert-danger" role="alert" id="insuffisant"> Solde insuffisant</div>
```

Ils sont d'abord cachés (hide()), dans le fichier **payement.js**:

```
$('#payer').hide()
$('#insuffisant').hide()
```

Pour savoir lequel des cas afficher, une fois le numéro de la carte scanné, il faut vérifier si le solde du compte est suffisant ou pas. Voici le code du bouton «valider» dans le fichier **payement.js**:

```
//valider
$('#btnValider').click(function(){
  var a = $('#username').val() //récupère la valeur du input (numéro de la carte)
  $.ajax({ //Fonction AJAX
    url: "/payer",
    method: "POST",
    data: 'recup=' + a
  }).done(function (data) { //récupère la valeur envoyée (send) par la route /payer.
    //affiche le solde du compte dans la page.
    $('#solde').html("Solde du compte: " + data[0].montant + "€")
    var som = $('#somme').html().split(" ") //permet de contenir seulement le nombre de la somme.
    var sol = $('#solde').html().split(" ") //permet de contenir seulement le nombre du solde.
    var a = parseInt((som[3])) //transforme le nombre de la somme en entier.
    var b = parseInt((sol[3])) //transforme le nombre du solde en entier.
    //condition: vérifie si le solde du compte est suffisant.
    if(a > b){ //si le solde est insuffisant, alors l'alerte s'affiche.
      $('#insuffisant').show()
      $('#payer').hide()
    }else { //sinon, le bouton payer s'affiche.
      $('#payer').show()
      $('#insuffisant').hide()
    }
  })
})
```


Voici la route qui à pour chemin **/payer**, avec laquelle AJAX communique :

```
router.post('/payer',function(req,res,next){
  //var a: récupère la valeur de la variable recup (numéro de la carte), envoyé à partir de AJAX.
  var a = req.body.recup;
  //requête SQL: sélectionne toutes les données de la carte (table carte_payement)
  qui a pour numéro de carte, la valeur du input.
  mysql.query('SELECT * FROM carte_payement WHERE numero_carte = ' + a,(req, row)=>{
    //envoie le résultat de la requête (données) vers le côté client.
    res.send(row)
  })
})
```

Exemple en cas de solde insuffisant:

numéro de carte scanné

Carte

Scanner le numéro:

99

valider

Somme à payer: 10€

bouton «valider» enclenché

Carte

Scanner le numéro:

99

valider

Somme à payer: 10€

Solde du compte: 1€

Solde insuffisant

Exemple en cas de solde suffisant:

numéro de carte scanné

Carte

Scanner le numéro:

21

valider

Somme à payer: 10€

bouton «valider» enclenché

Carte

Scanner le numéro:

21

valider

Somme à payer: 10€

Solde du compte: 7300€

Payer

J'ai ensuite terminé par programmer le bouton «payer», qui est lié à une **page modal** (myModal):

```
<button class="btn btn-success" id="payer" data-toggle="modal" data-target='#myModal'>
  Payer
</button>
```

```
<!--Page modal-->
<div id="myModal" class="modal fade">
  <div class="modal-dialog modal-confirm">
    <div class="modal-content">
      <div class="modal-header">
        <div class="icone" style="display: flex;">
          <div class="icon-box" >
            <i class="material-icons">&#xE876;</i>
          <br>
        </div>
        <h4 style="margin-left: 10px;color: white;">paiement accepté</h4>
      </div>
    </div>
    <!--Bas de page modal-->
    <div style="display: flex;">
      <div class="modal-body text-left">
        <!--nouveau solde-->
        <h6 id="solde_new" style="margin-top: 18px;"></h6>
      </div>
      <div class="modal-body text-right">
        <button class="btn">Imprimer les tickets</button>
      </div>
    </div>
  </div>
</div>
```

Donc une fois le bouton «payer» enclenché, une page modal s'affiche pour annoncer que le paiement c'est bien déroulé.

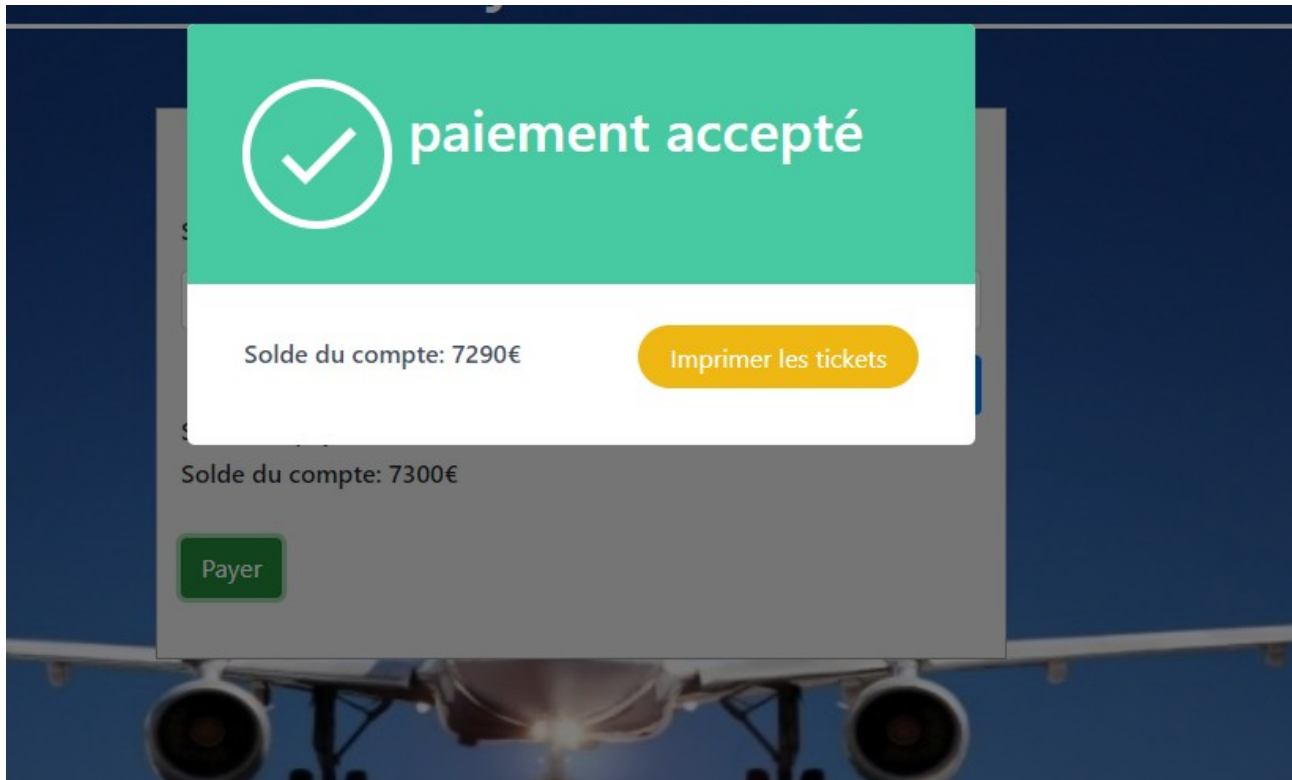
Cette page modal affiche le nouveau solde du compte dans la balise **<h6>** qui a pour **id** «solde_new»,

et un bouton

Imprimer les tickets

Le bouton «payer» permet aussi d'insérer le nouveau solde du compte dans la B.D.D, à l'aide d'AJAX.

Le design de cette page modal ressemble à ceci:



Voici le code du bouton «payer» dans le fichier paiement.js:

```
//payer
$('#payer').click(function(){
    var som = $('#somme').html().split(" ") //permet de contenir seulement le nombre de la somme.
    var sol = $('#solde').html().split(" ") //permet de contenir seulement le nombre du solde.
    var a = parseInt((som[3])) //transforme le nombre de la somme en entier.
    var b = parseInt((sol[3])) //transforme le nombre du solde en entier.
    var c = b - a //retire le montant de la somme au solde de la carte.
    $('#solde_new').html("Solde du compte: "+c +"€") //affiche le nouveau solde dans la page modal.
    var num = $('#username').val() //récupère la valeur du input (numéro de la carte)
    $.ajax({ //fonction AJAX
        url: "/solde",
        method: "POST",
        //Envoie deux variable au côté serveur: le nouveau solde et le numéro de la carte.
        data: 'recup1=' + c + '&recup2=' + num
    })
})
```

Voici la route qui a pour chemin **/solde**, avec laquelle AJAX communique :

```
//insere le nouveau solde
router.post('/solde',function(req,res,next){
  //récupère la variable recup1 (le nouveau solde), envoyé à partir d'AJAX.
  var solde = req.body.recup1;
  //récupère la variable recup2 (le numéro de la carte), envoyé à partir d'AJAX.
  var num_compte = req.body.recup2;
  //requête SQL: modifie le solde de la carte qui a pour numéro de carte la valeur du input,
  par le nouveau solde.
  mysql.query('UPDATE carte_paiement SET montant =' + solde +
  'WHERE numero_carte = ' + num_compte,(req, row)=>{
    })
}
```

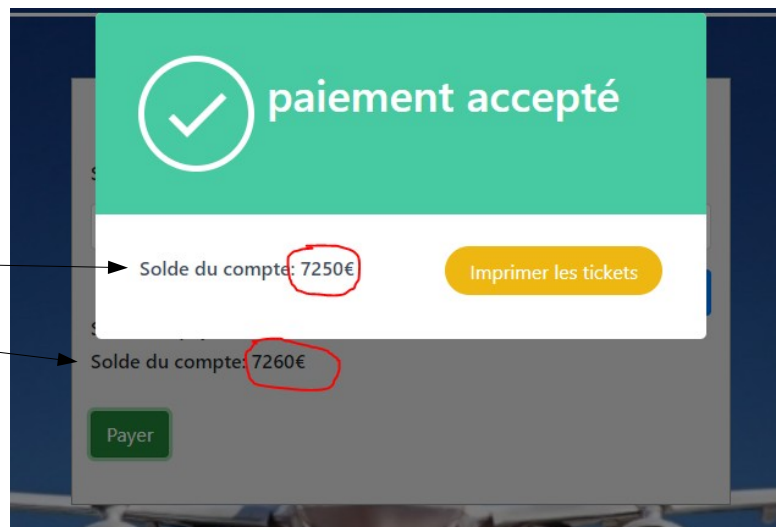
Exemple (somme à payer: 10€):
Le solde du compte est d'abord
à 7260 €.

+ Options									
					Id	Id_utilisateur	montant	numero_carte	
<input type="checkbox"/>	Éditer	Copier	Supprimer		1	1	7260	21	
<input type="checkbox"/>	Éditer	Copier	Supprimer		2	1	1	99	

Une fois le numéro de la carte vérifié et le bouton
«payer» déclenché, la page modal s'affiche :

Nouveau solde

Ancien solde



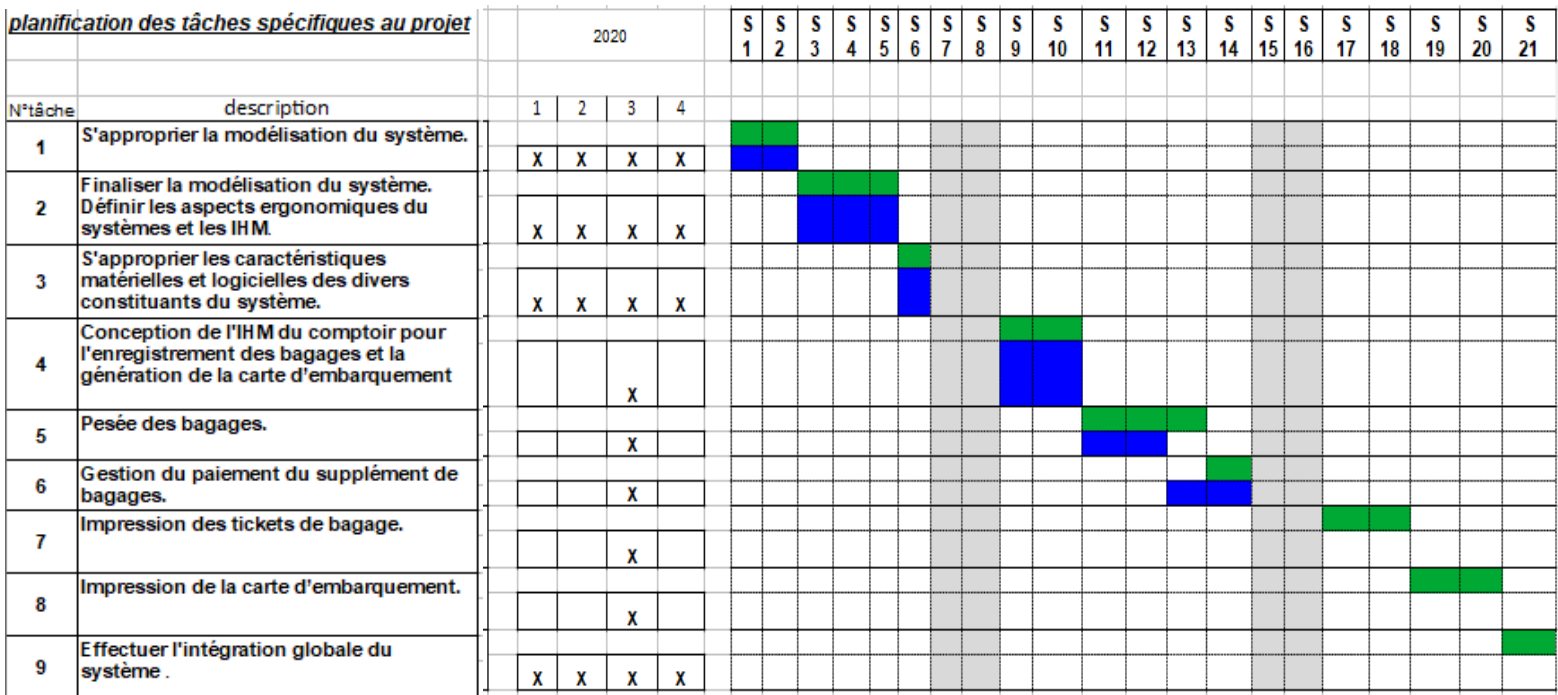
Côté serveur, le montant a bien changé, il est passé à 7250 €:

					Id	Id_utilisateur	montant	numero_carte	
<input type="checkbox"/>	Éditer	Copier	Supprimer		1	1	7250	21	
<input type="checkbox"/>	Éditer	Copier	Supprimer		2	1	1	99	

Une fois le paiement terminé, je me suis arrêté ici car je n'avais pas l'imprimante à la maison (dû au confinement). Je n'ai donc pas pu programmer le bouton «imprimer les tickets» et générer la carte d'embarquement.

V. Temps prévisionnelle

Diagramme de Gantt :



Prévision: 

Réalisation: 

VI. Avis personnel

J'ai apprécié:

- Travailler en groupe.
- Travailler en autonomie.
- Apprendre de nouvelles choses.



J'aurais aimé:

- Poursuivre le projet.



VII.Conclusion

Pour conclure, je peux dire que ce projet a été une bonne expérience. J'ai acquis plusieurs compétences et découvert de nouvelles technologies. En plus de cela, j'ai appris à travailler en équipe et à mieux organiser mon travail.

Ce projet a renforcé mon désir de poursuivre mes études dans la programmation web.

Prochain objectif: Durant le confinement, j'ai commencé à apprendre **Angular** (framework JavaScript). Mon prochain objectif est de remplacer le côté client de mon projet (jQuery + AJAX) par du **Angular**.

IV.5.Annexes

Voici les documents qui m'ont été utiles:

- Vidéos sur Node.js (Grafikart): <https://www.youtube.com/watch?v=0PA69L88HeI>
- Gestionnaire de module NPM: <https://www.npmjs.com/>
- Documentations Node.js: <https://nodejs.org/en/docs/>
<https://devdocs.io/node/>
- Documentation JavaScript: <https://developer.mozilla.org/fr/docs/Web/JavaScript>