



atsuhikoMochizuki

PIZZAHUTTE

Création d'une application Javascript FullStack
complète



PIZZAHUTTE

Système de gestion administrative d'une pizzeria



Projet individuel

Sous la direction de Rossi Oddet

Avril 2023

git clone <https://github.com/atsuhikoMochizuki/pizzahute.git>



Table des matières

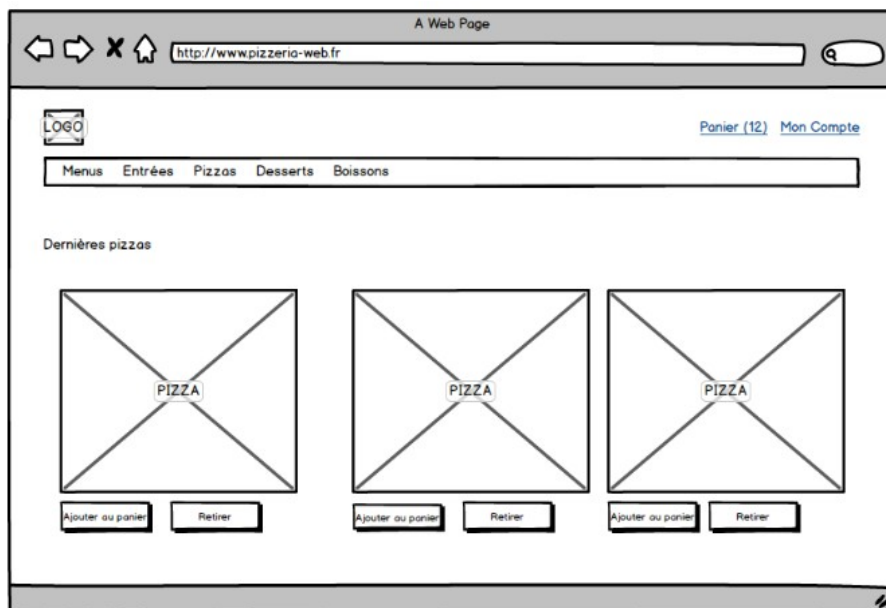
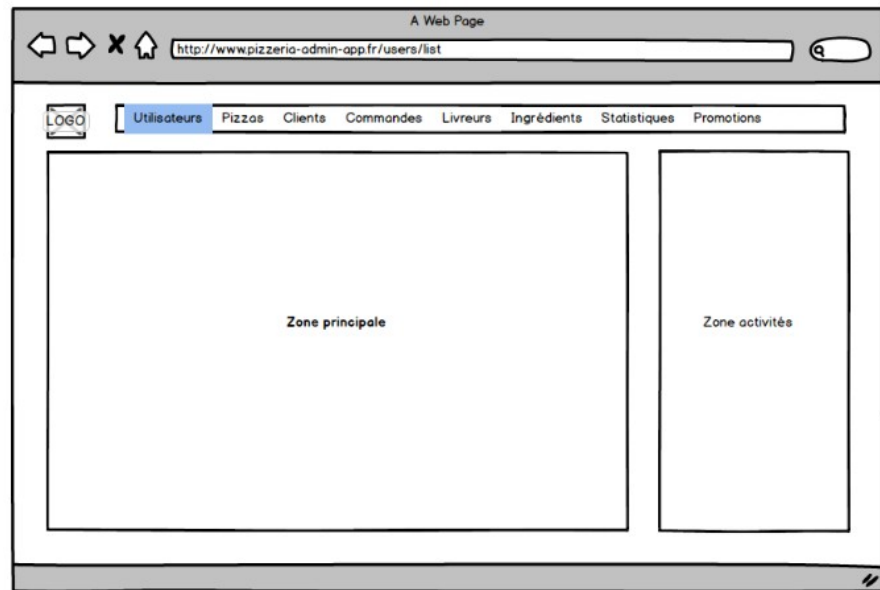
1. Cahier des charges.....	4
Posons-nous.....	8
2. Analyse.....	9
2.1. Diagramme des cas d'utilisation.....	11
2.2. Organisation sous forme de sprints.....	13
2.3. Découpage fonctionnel des sprints.....	14
3. LANCEMENT DES SPRINTS.....	15
3.1. USA001 – Authentification page login.....	16
3.1.1. Analyse de la tâche.....	17
3.1.2. Planification du sprint.....	18
3.1.3. USA001 – 1 Création d'un serveur HTTP.....	18
3.1.3.1. Modélisation de la fonctionnalité.....	18
3.1.3.1.1. Premier Jet.....	18
3.1.3.1.1.1. Diagramme de séquence.....	18
3.1.3.1.1.2. Diagramme des classes participantes.....	20
3.1.3.1.2. Second jet.....	21
3.1.3.1.2.1. Diagramme de séquence corrigé et approfondi.....	21
3.1.3.1.2.2. Dédution d'un diagramme des classes participantes beaucoup plus réaliste.....	23
3.1.3.2. Choix des technologies.....	24
3.1.3.2.1. Génération des pages côté serveur.....	24
3.1.3.2.2. Choix du framework NodeJS pour l'implémentation du serveur en Javascript.....	24
3.1.3.2.2.1. Surcouche applicative Express.....	25
3.1.3.2.2.2. Génération des pages : moteur de vue PUG.....	25
3.1.3.2.3. Mise page du front.....	25
3.1.3.2.3.1. Choix de la librairie Bootstrap pour optimiser le temps de développement et le responsive du front.....	25
3.1.3.2.4. Système de gestion de base de données relationnelles :.....	25
3.1.3.2.5. Outils de scripting : BASH.....	26
3.1.3.3. Mise en place de environnement de travail.....	26
3.1.3.3.1. Structuration de l'arborescence du projet.....	26
3.1.3.3.2. Création du dépôt github et clonage du projet vierge.....	26
3.1.3.3.3. Création du projet node via npm.....	26
3.1.3.3.3.1. installation des dépendances.....	26
3.1.3.3.3.2. Paramétrage du fichier de configuration package.json.....	26
3.1.3.3.3.2.1.1. Affectation de nodemon à la commande start de l'objet script.....	26
3.1.3.3.3.2.2. Prise en charge des modules ECMAScript (import, export).....	26
3.1.3.3.4. Consignation de la procédure dans un script BASH. Permettra de générer immédiatement le projet en cas de « pépin ».....	26
3.1.3.3.5. Création pour l'occasion d'une petite application : le MochizukiServerProjectGenerator.....	28
3.1.3.3.5.1. Code source.....	29
3.1.3.4. Codage de la fonctionnalité.....	32
3.1.3.4.1. Serveur.....	32
3.1.3.4.2. Interface de gestion de l'application Express (app.js).....	34

3.1.3.4.3. Routage administrateur (index.js).....	35
3.1.3.4.4. Routage des clients (users.js).....	35
3.1.3.4.4.1. Remarques.....	36
3.1.3.4.4.1.1. Création de launchClientBrowser et implémentation d'une fonction d'appel système en Javascript.....	36
3.1.4. USA001 – 3 Création d'une base de données.....	37
3.1.5. USA001 – 4 Codage de la page d'authentification.....	37
3.1.5.1. La puissance de l'infrastructure d'application WEB NodeJs EXPRESS.....	37
3.1.5.2. Organisation des vues.....	38
3.1.5.3. Le moteur de vue PUG.....	39
3.1.5.3.1. Avantages.....	39
3.1.5.3.2. Inconvénients.....	39
3.1.5.4. Reprise nécessaire des fondamentaux de l'outil Bootstrap.....	40
3.1.5.4.1. En résumé : ce qu'il est fondamental de saisir du système Bootstrap, comment penser cet outil puissant et responsive.(apporter précisions).....	41
3.1.5.5. Codage du head commun aux vues (layout .pug).....	43
3.1.5.5.1. Particularités rencontrées.....	43
3.1.5.5.2. Liaison des dépendances.....	43
3.1.5.5.3. Code source de layout.pug.....	44
3.1.5.5.4. Codage de la page d'authentification de l'administrateur(index.pug).....	44
3.1.5.6. Aperçu côté Front.....	45
3.1.6. USA001 – 5 Fonctions de récupération de la saisie utilisateur.....	47
3.1.7. USA001 – 6 administration de la base de données.....	47
3.1.8. USA001 – 7 Codage de la page d'accueil.....	47
3.1.9. USA001 – 8 Codage de l'automatisation d'envoi de mail.....	47
3.2. USA002 : Naviguer sur dans tout le site à l'aide d'une barre de menu.....	48
3.2.1. Faire bifurquer le code PUG à l'aide de la commande block.....	49
3.2.2. Création de la vue principale pour pouvoir afficher le menu.....	49
3.2.2.1. Configuration du router users.js.....	50
3.2.2.1.1. Affectation de la vue de la page d'accueil à la route principale »/ » du routeur des utilisateurs (users.js).....	50
3.2.3. Code source du menu de admin.....	51
3.2.4. Aperçu côté navigateur depuis la page d'accueil clients users.pug.....	52
4. Annexes.....	53

1. Cahier des charges

L'objectif du projet est de réaliser un système de gestion d'une pizzeria avec deux applications :

Une application d'administration des données utilisée par les gestionnaires de la pizzeria



Une application utilisée par des clients pour effectuer des commandes

PIZZAHUTTE

Création d 'un application Javascript FullStack complète

Contraintes techniques

- Les données sont sauvegardées dans une base de données (relationnelle ou non)
- L'application d'administration est réalisée en multi-pages (génération de pages côté serveur)

L'application Web destinée aux clients, vous avez le choix :

- soit une application multi-pages
- soit une application basée sur une seule page (Single Page Application)

Fonctionnalités à développer

La symbolique :dart désigne le périmètre minimum à réaliser.

A chaque fois qu'une fonctionnalité est réalisée, cocher la case correspondante dans ce fichier README.md. Pour cocher une case, ajouter une croix `[x]`.

Exemple :

- [x] [USA001 - Authentification - Page Login](issues/admin/usa001.md)

Les écrans ne sont pas contractuelles, n'hésitez pas à être force de proposition.

Administration

- [] [USA001 - Authentification - Page Login](issues/admin/usa001.md)
- [] :dart: [USA002 - Menu](issues/admin/usa002.md)
- [] [USA003 - Utilisateur - CRUD](issues/admin/usa003.md)
- [] :dart: [USA004 - Pizza - CRUD](issues/admin/usa004.md)
- [] :dart: [USA005 - Livreur - CRUD](issues/admin/usa005.md)
- [] :dart: [USA006 - Client - CRUD](issues/admin/usa006.md)
- [] :dart: [USA007 - Commande - CRUD](issues/admin/usa007.md)
- [] [USA008 - Statistiques (temps réel)](issues/admin/usa008.md)
- [] [USA009 - Promotions - CRUD](issues/admin/usa009.md)
- [] [USA010 - Historique des emails envoyés](issues/admin/usa010.md)
- [] [USA011 - Ingrédients - CRUD](issues/admin/usa011.md)
- [] :dart: [USA012 - Visualiser activités (temps réel)](issues/admin/usa012.md)
- [] [USA013 - Boissons - CRUD](issues/admin/usa013.md)
- [] [USA014 - Desserts - CRUD](issues/admin/usa014.md)
- [] [USA015 - Menu - CRUD](issues/admin/usa015.md)
- [] [USA016 - Gestion des stocks (temps réel)](issues/admin/usa016.md)

Client

- [] [USW001 - Accueil](issues/web/usw001.md)
- [] [USW002 - Lister pizzas](issues/web/usw002.md)
- [] [USW003 - Ajouter au panier](issues/web/usw003.md)
- [] [USW004 - Gérer panier](issues/web/usw004.md)
- [] [USW005 - S'inscrire](issues/web/usw005.md)
- [] [USW006 - Se connecter](issues/web/usw006.md)
- [] [USW007 - Créer une commande](issues/web/usw007.md)
- [] [USW008 - Mon compte](issues/web/usw008.md)
- [] [USW009 - Multilangues](issues/web/usw009.md)
- [] [USW010 - Détail de la commande](issues/web/usw010.md)
- [] [USW011 - Gestion de la promotion](issues/web/usw011.md)
- [] [USW012 - Lister les menus](issues/web/usw012.md)
- [] [USW013 - Lister les boissons](issues/web/usw013.md)
- [] [USW014 - Lister les entrées](issues/web/usw014.md)
- [] [USW015 - Lister les desserts](issues/web/usw015.md)
- [] [USW016 - Notation des Pizzas](issues/web/usw016.md)
- [] [USW017 - Composer sa pizza](issues/web/usw017.md)

Posons-nous

La lecture de ce cahier des charges apparaît à nos yeux, novices depuis seulement trois semaines dans les technologies WEB, relativement dense.

Une connaissance incomplète des concepts, des langages et plus généralement une acquisition encore trop partielle des fondamentaux de la programmation client-serveur nous prive du recul nécessaire pour aborder cet exercice au pied levé.

Nous allons donc consacrer un temps pour bien saisir la problématique, et cerner les contours de ce qui nous est demandé.

Nous aurons ainsi une base pour pouvoir concevoir notre modèle

- au niveau structurel (qui va réaliser quoi en son sein?)
- Au niveau logique (de quelle façon?)
- Au niveau séquentiel (a quel moment? dans quel ordre) ?

L'objectif est ici de prendre un maximum de recul et d'amener notre pensée dans une direction sobre, claire, et efficace.

Nous pourrons alors effectuer en connaissance de cause les choix technologiques et les concepts qui nous sembleront les plus judicieux pour mettre en œuvre pour le développement.

Nous pourrons alors aborder le codage de notre modèle en ayant mis de notre côté le maximum d'atouts pour concevoir une application robuste.

Reste à savoir jusqu'où nous pourrons aller dans le temps qui nous est imparti.

Pour l'instant l'horizon n'est pas encore visible...

2. Analyse

Le cahier des charges qui nous est fourni est très structuré et clair.

Nous n'aurons donc pas à clarifier et modéliser l'expression d'un éventuel besoin mal formulé.

Néanmoins comme il faut bien commencer par quelque chose, nous proposons de re-lister rapidement dans un diagramme de cas d'utilisation les fonctionnalités que doit accomplir notre système.

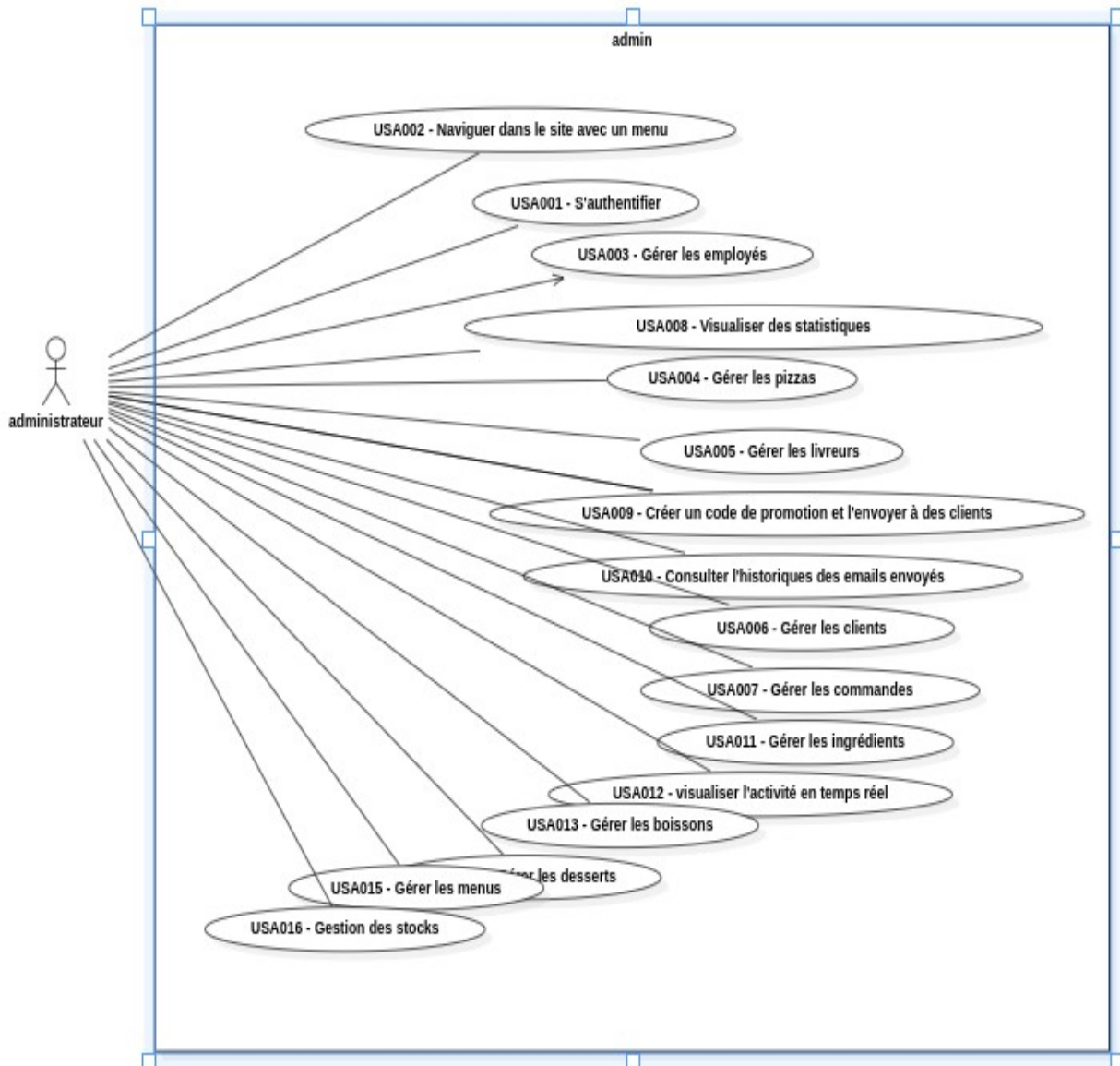
Un exercice très facile qui a l'avantage de pouvoir nous *rattacher à un de point départ*, ce qui écarte d'office le syndrome de « feuille blanche ». Il impose en outre :

- **une relecture du cahier** des charges qui permet de mieux cerner ce qui nous est demandé, *élaborer le flux d'informations, clarifier notre pensée*, nous *imprégner* du point de vue *métier*.
- de *manipuler graphiquement des concepts* que nous allons implémenter *mentalement*.
- d'avoir une *représentation visuelle du système* à concevoir, et donc un certain **recul, nécessaire**.

Nous mettons ainsi en place, s'en-même s'en rendre compte, **une représentation mentale de ce qui n'existe pas encore**, tout en positionnant notre esprit dans une attitude mentale d'analyse.

Ceci nous apparaît comme un excellent point de départ pour le projet.

2.1. Diagramme des cas d'utilisation



2.2. Organisation sous forme de sprints

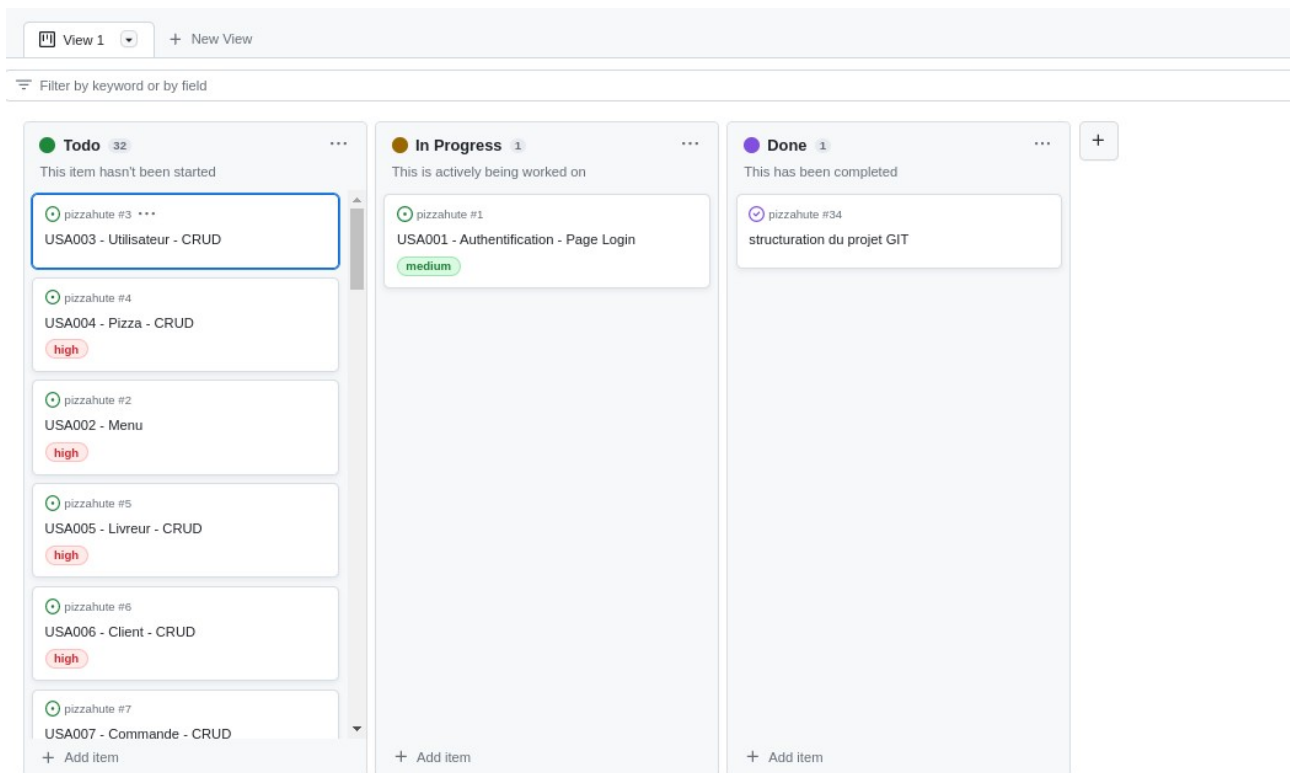
Au vu du travail à réaliser, le temps imparti ne nous permettra dans l'état actuel de nos connaissances des langages de modéliser l'application entière.

Etant donné que de toute façon le découpage est déjà fait, nous allons donc organiser notre développement sur le modèle de sprint, afin d'optimiser notre projet.

A savoir que nous nous consacrerons au développement, fonctionnalités par fonctionnalité, on pourrait par *users stories*, et ainsi améliorer de façon incrémentale notre application.

Dans une démarche d'apprentissage, nous avons réorganisé les tâches au sein d'un tableau Kaban.

L'idée ici était de profiter de du projet pour se familiariser aux méthodes de type Agile, et de faire connaissance avec les outils de gestion projet que propose GitHub , de manipuler ce dernier, de l'explorer et tirer le meilleur de cet outil puissant qui se découvre à nos yeux de plus en plus un précieux allié sur de nombreux plans.



2.3. Découpage fonctionnel des sprints

Pour chacune des fonctionnalités demandées, nous allons descendre de façon granulaire et lister les points à développer pour atteindre les objectifs métiers.

Une fois cette liste établie, le développement s'effectuera **dans la direction inverse**.

Ceci permettra de construire les briques qui permettront les blocs et in fine réaliser la fonctionnalité.

3. LANCEMENT DES SPRINTS

3.1. USA001 – Authentication page login

En tant qu'administrateur, je dois pouvoir m'authentifier afin d'accéder aux fonctionnalités du système.

- Les informations d'authentification sont vérifiées dans une table (user par exemple).
- Un utilisateur de l'application Admin possède les informations suivantes :
 - Email
 - Mot de passe
 - Nom
 - Prénom


Une fois connectée, l'utilisateur est redirigé vers la page d'administration des commandes.

Lorsque l'utilisateur clique sur le lien mot de passe oublié, un nouveau mot de passe est généré et est envoyé par email à l'utilisateur. Le système attend la confirmation de l'utilisateur (clic sur un lien envoyé) en l'invitant à saisir un nouveau mot de passe.

Un utilisateur non connecté, n'a accès à aucune fonctionnalité.

3.1.1. Analyse de la tâche

Un petit tableau nous permet de rapidement saisir la portée sous-jacente d'un exercice à première vue anodin :



Points clés exigés pour résoudre la fonctionnalité métier	Implique de	Implique de	Implique de
Gérer le mot de passe oublié	Détecter un évènement (clic) sur lien	Gérer une boucle événementielle	Créer un serveur HTTP ou un socket
	Régénérer un nouveau mot de passe et le stocker	Implémenter de la logique	Gérer une application temps réel
		Administrer un SGBD	Mettre en œuvre un serveur de donnée Créer une base de données
Afficher une page d'authentification et d'accueil	Envoyer un mail	Automatiser l'envoi de mail Implémenter de la logique	Créer un serveur HTTP Implémenter une application
	Générer des pages WEB	Router les requêtes	Créer un serveur HTTP
		De coder les pages	Implémenter du html, css et du javascript
Vérifier les informations de connexion	Récupérer de la saisie utilisateur	Gérer une boucle événementielle	Créer un serveur HTTP ou un socket
	Les comparer à des données	Implémenter de la logique	gérer une application
		administrer des données	Administer un SGBD

La colonne rouge du tableau laisse clairement apparaître le gros du travail pour accomplir le sprint, à savoir :

- **créer un serveur HTTP**
- **implémenter une application en logique événementielle**
- **Créer une base de données**
- **L'administrer via un système de Gestion de Base de Données Relationnelle.**

3.1.2. Planification du sprint

Comme nous l'avons dit plus haut, nous remontant dans le sens inverses de ce que nous avons listé, ce qui nous permet d'organiser le sprint de cette forme :

USA001 – 1 création d'un serveur HTTP

USA001 – 2 Installation d'un serveur de données

USA001 – 3 Création d'une base de données

USA001 – 4 Codage de la page d'authentification

USA001 – 5 Fonctions de récupération de la saisie utilisateur

USA001 – 6 administration de la base de données

USA001 – 7 Codage de la page d'accueil

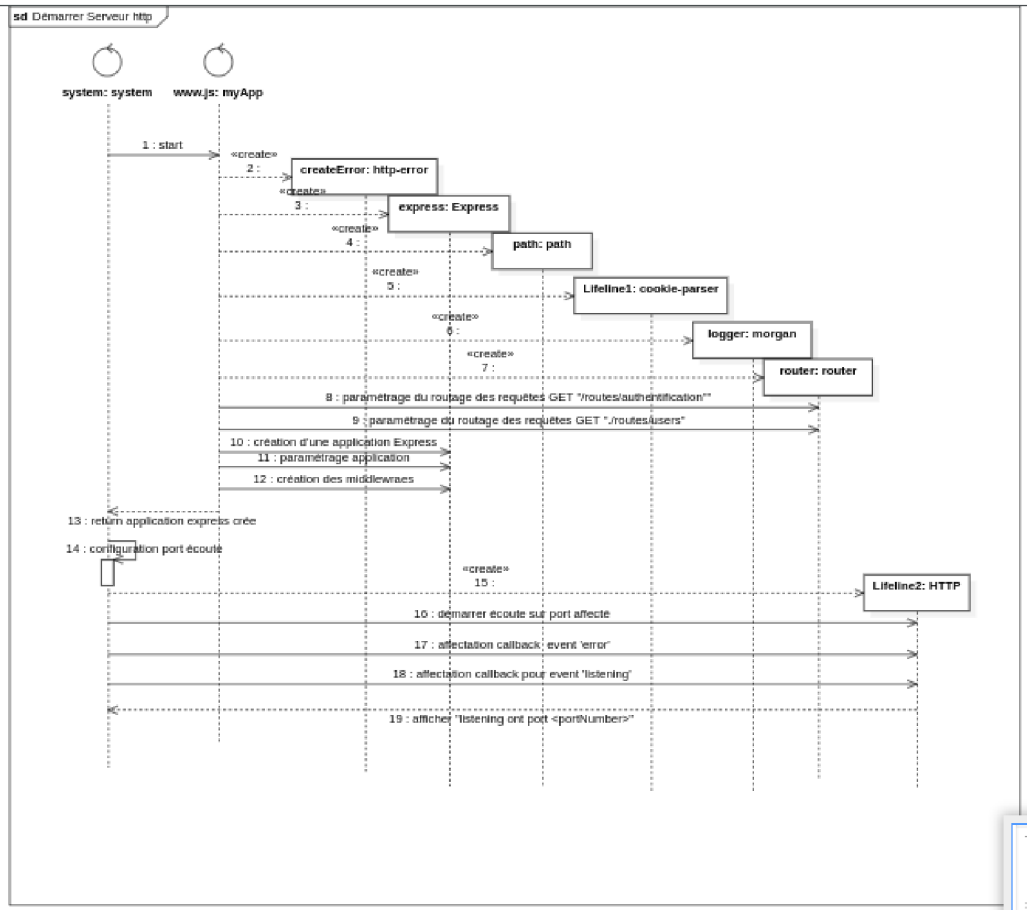
USA001 – 8 Codage de l'automatisation d'envoi de mail.

3.1.3. USA001 – 1 Création d'un serveur HTTP

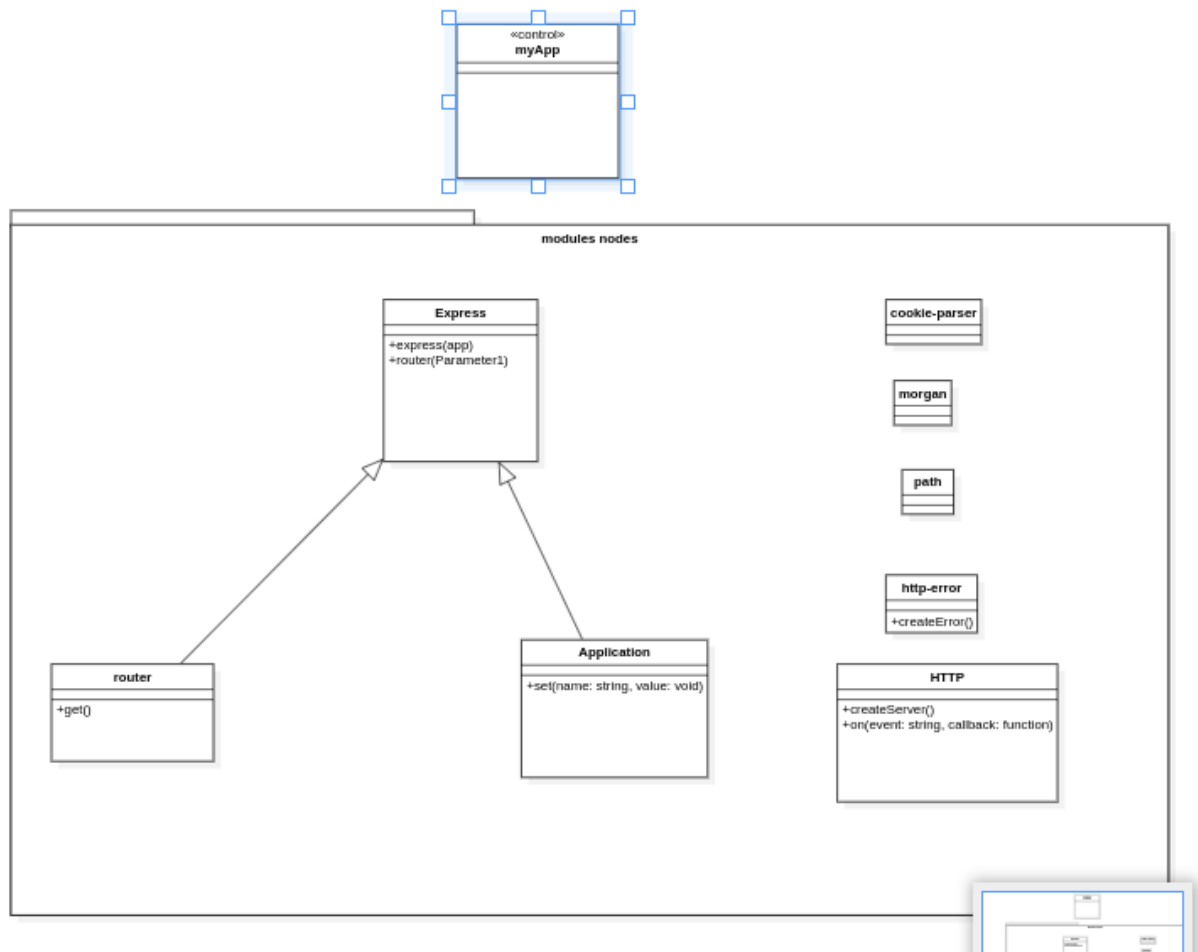
3.1.3.1. *Modélisation de la fonctionnalité*

3.1.3.1.1. Premier Jet

3.1.3.1.1.1. *Diagramme de séquence*



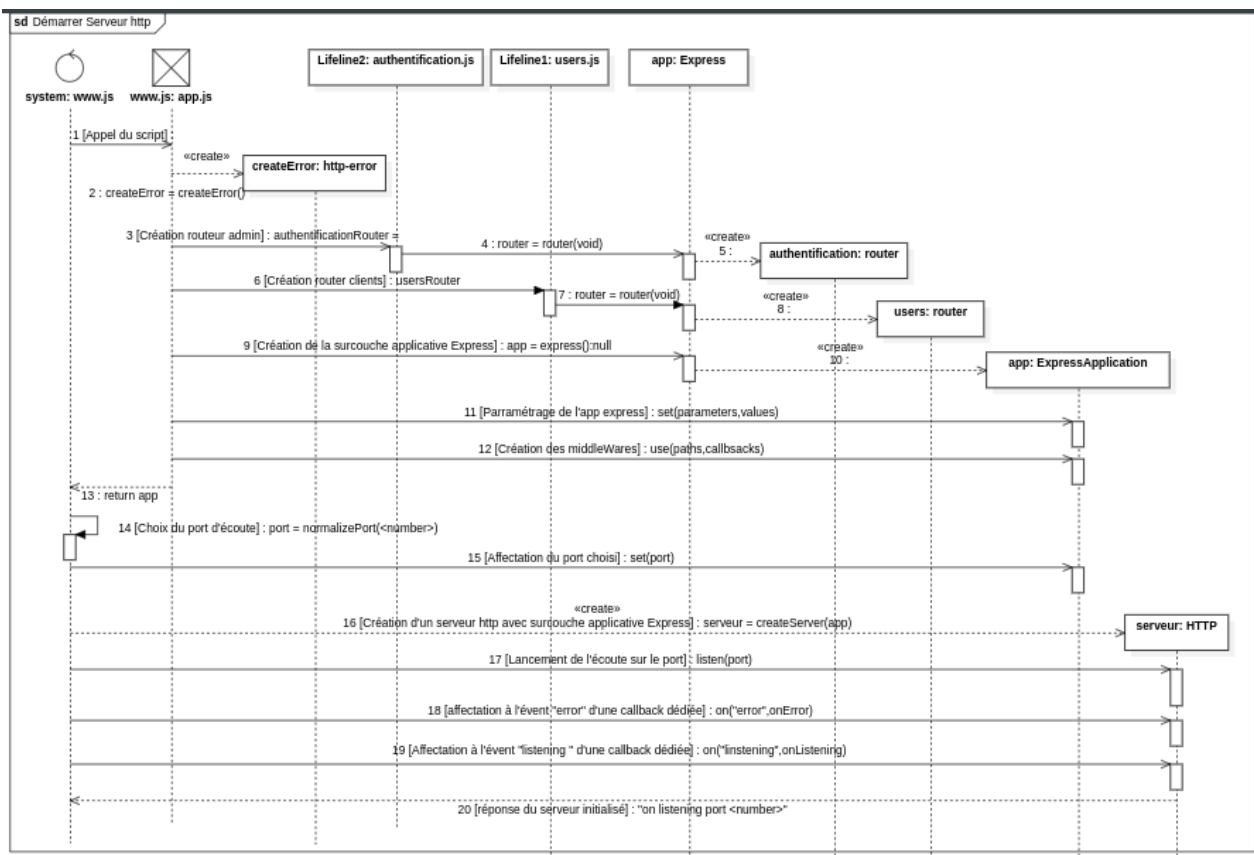
3.1.3.1.2. Diagramme des classes participantes

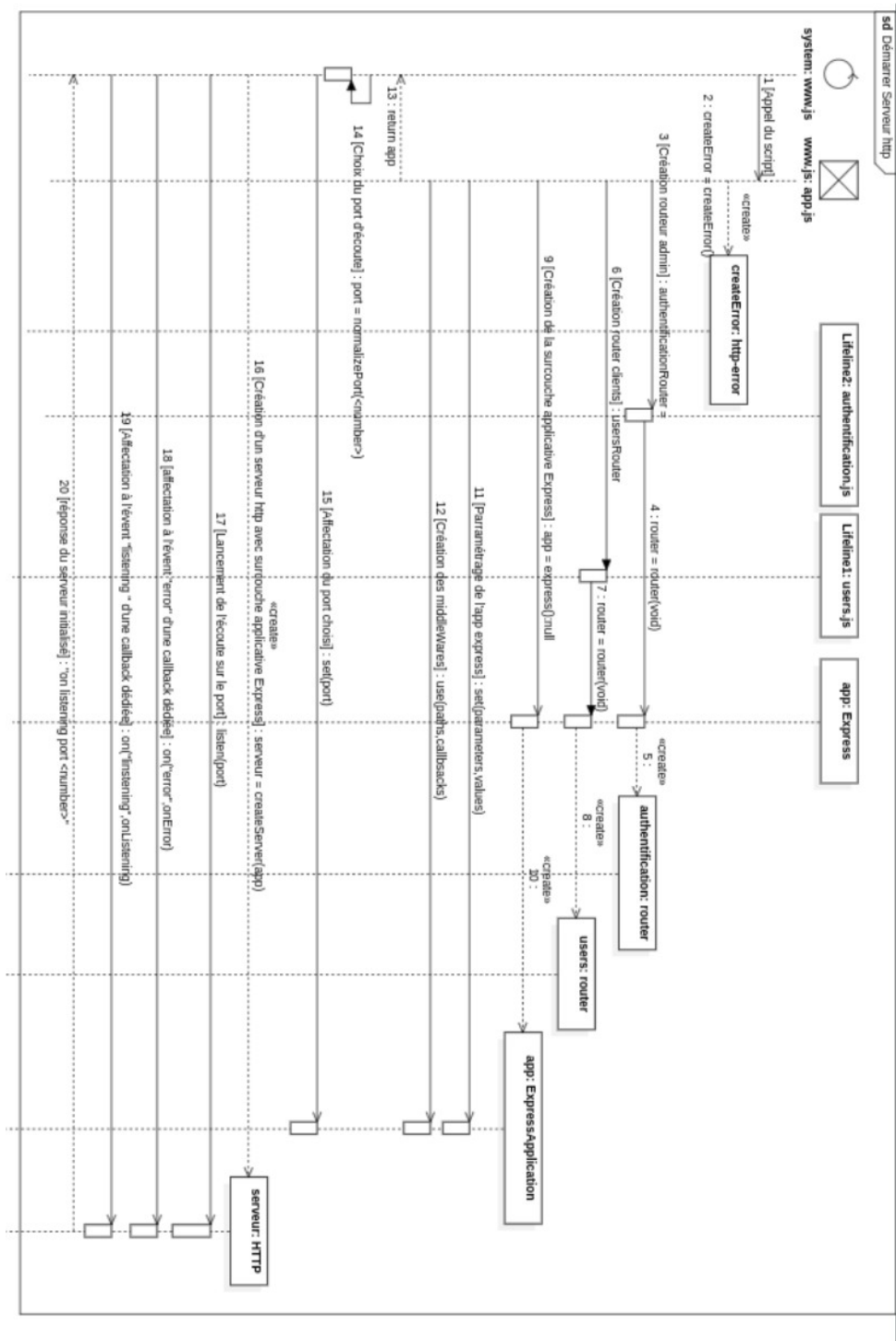


3.1.3.1.2. Second jet

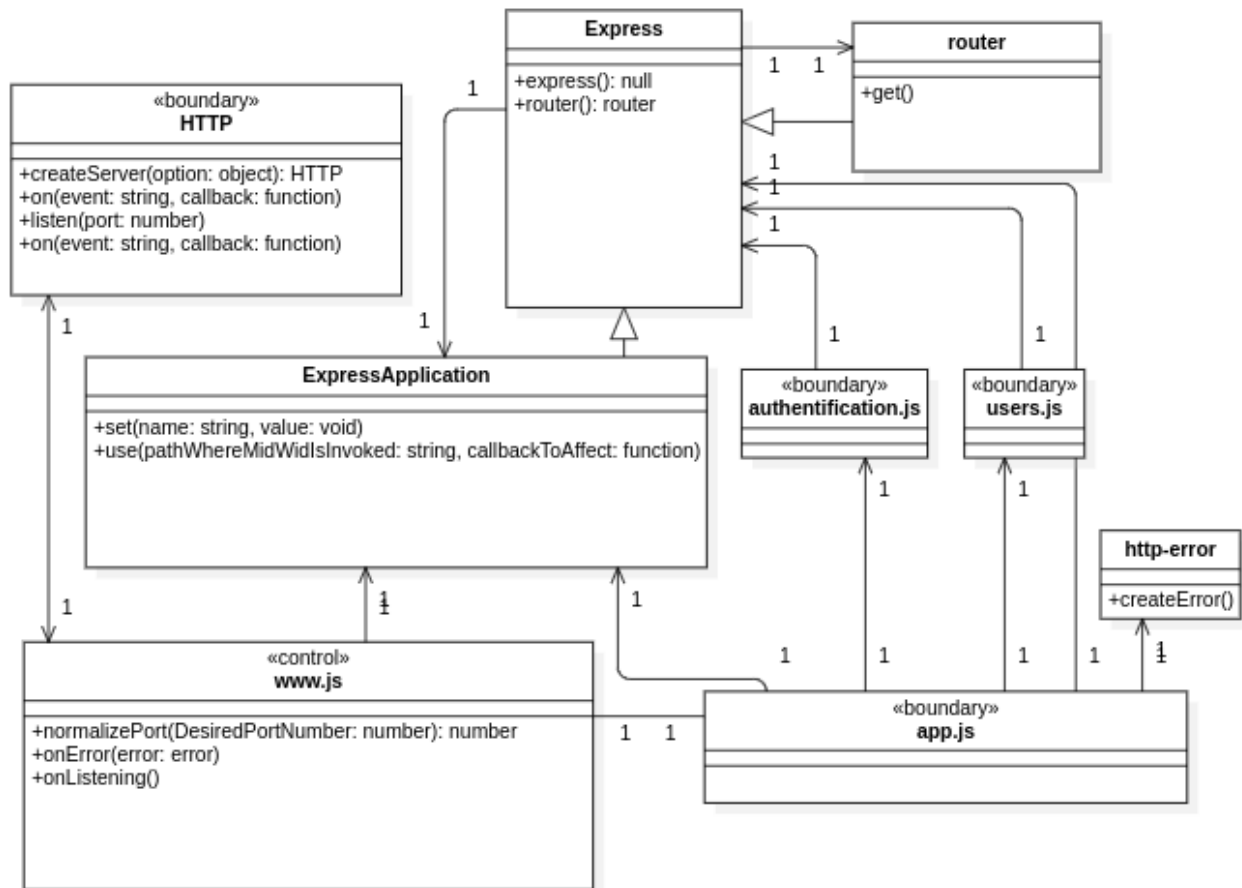
Nous allons affecter les méthodes des classes aux différentes interactions entre les objets de l'application

3.1.3.1.2.1. Diagramme de séquence corrigé et approfondi





3.1.3.1.2.2. Dédution d'un diagramme des classes participantes beaucoup plus réaliste



3.1.3.2. *Choix des technologies*

3.1.3.2.1. **Génération des pages côté serveur**

Les pages seront générées dynamiquement côté serveur. Ceci va nous permettre :

- D'approfondir les bases de la programmation côté serveur
- Générer les pages de façon dynamique
- Sécuriser le site
- Administrer la base de données

Ce travail est pour le novice un parfait support pour approfondir et saisir dans leurs substances les fondamentaux de la programmation-serveur, dont nous souffrons actuellement pour pouvoir progresser.

Nous consacrerons donc **un temps certain à la compréhension de la mécanique « serveur »**.

C'est une priorité qui nous apparaît fondamentale pour se construire une solide formation et accélérer par la suite et de façon exponentielle la productivité.

3.1.3.2.2. **Choix du framework NodeJS pour l'implémentation du serveur en Javascript**

Les pages seront générées **dynamiquement** par un serveur.

Nous utiliserons **NodeJS** pour implémenter ce serveur.

Node.js est une plateforme logicielle libre en JavaScript, orientée vers les applications réseau évènementielles hautement concurrentes qui doivent pouvoir monter en charge.

Parmi les modules natifs de Node.js, on retrouve http qui permet le développement de serveur HTTP. Ce qui autorise, lors du déploiement de sites internet et d'applications web développés avec Node.js, de ne pas installer et utiliser des serveurs webs tels que Nginx ou Apache.

Concrètement, Node.js est un environnement bas niveau permettant l'exécution de JavaScript côté serveur.

Nous utiliserons également et en surcouche le module **Express** de Node qui va nous permettre une gestion du serveur beaucoup plus aisée robuste.

3.1.3.2.2.1. Surcouche applicative Express

Express est une infrastructure d'applications Web Node.js minimaliste et flexible qui fournit un ensemble de fonctionnalités robuste pour les applications Web et mobiles.

Grâce à une foule de méthodes utilitaires HTTP et de middleware mise à votre disposition, la création d'une API robuste est simple et rapide.

Express apporte une couche fine de fonctionnalités d'application Web fondamentales, sans masquer les fonctionnalités de Node.js

3.1.3.2.2.2. Génération des pages : moteur de vue PUG

3.1.3.2.3. Mise page du front

3.1.3.2.3.1. Choix de la librairie Bootstrap pour optimiser le temps de développement et le responsive du front

Bootstrap est une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option.

C'est un point dont nous allons également profiter avec ce projet.

Nous avons beaucoup de point *d'incompréhension* avec l'utilisation de Bootstrap, et plus généralement avec l'organisation des éléments de la page avec les propriétés CSS Flex, qui nous font perdre un temps *considérable* dans notre développement ainsi qu'en *qualité de code*.

Maîtriser le CSS ainsi qu'outil de Bootstrap, qui nous apparaît comme un précieux allié, est un des objectifs fondamentaux visés dans ce projet, avec l'implémentation des serveurs sous NodeJS.

3.1.3.2.4. Système de gestion de base de données relationnelles :

Nous utiliserons MySQL pour administrer la base de données

3.1.3.2.5. Outils de scripting : BASH

Nous consignerons certaines procédures dans des scripts BASH dans un but de ré-implémentation et d'automatisation.

3.1.3.3. *Mise en place de environnement de travail*

3.1.3.3.1. Structuration de l'arborescence du projet

3.1.3.3.2. Création du dépôt github et clonage du projet vierge

3.1.3.3.3. Création du projet node via npm

```
$ npm init -y
```

3.1.3.3.3.1. *installation des dépendances*

```
npm install nodemon --save-dev  
npm install bootstrap --save  
npm install socket.io --save  
npm install body-parser --save
```

3.1.3.3.3.2. *Paramétrage du fichier de configuration package.json*

3.1.3.3.3.2.1.1. Affectation de nodemon à la commande start de l'objet script

```
"start": "nodemon ./bin/www"
```

3.1.3.3.3.2.2. Prise en charge des modules ECMAScript (import, export)

```
"type": "module"
```

3.1.3.3.4. Consignation de la procédure dans un script BASH. Permettra de générer immédiatement le projet en cas de « pépin »

Les étapes que nous avons listées précédemment sont à présent listées dans un script bash pour nous ferons gagner du temps pour les projets qui suivront

```
#!/usr/bin/bash  
clear  
cat << EOF
```

PIZZAHUTTE

Création d'un application Javascript FullStack complète

PIZZAHUTTE

Générateur de projet
version 1.0
by Atsuhiko Mochizuki

EOF

```
read -p "Entrez le nom de l'application svp:" uservar
echo "[Génération de l'application $uservar..."
echo "[Moteur de vue:PUG"
echo "[Génération de CSS :sass"
echo "[Création d'un .gitignore"

express --pug -css sass --view pug --git $uservar
cd $uservar

echo "[Installation des dépendances..."
echo "[      --nodemon"
npm install nodemon --save-dev
echo "[      --bootstrap"
npm install bootstrap --save
echo "[      --socket.io"
npm install socket.io --save
echo "[      --body-parser"
npm install body-parser --save

echo "[Test du serveur"
echo -e "\033[31m [Serveur en attente : veuillez entrer dans un navigateur 'localhost:3000'\033[1;32m"
DEBUG=$uservar:* npm start
```

PIZZAHUTTE

Création d'un application Javascript FullStack complète

3.1.3.3.5. Création pour l'occasion d'une petite application : le MochizukiServerProjectGenerator

A l'idée du script Bash déroulé précédemment, il nous apparaît intéressant d'en profiter pour généraliser le script et créer un petit capable de nous générer instantanément un projet fonctionnel clé-en-main.

A savoir le script va générer en plus

- un dossier Office structuré clé-en-main , avec une mise en page standardisée, prêt à la rédaction
- Une page de d'accueil du serveur clé-en-main
- Des logos

En résumé :

- Le temps de mise en place de l'environnement du projet est **réduit à néant**.

En un clic :

- **Le serveur est généré et démarré avec une page d'accueil et d'authentification.**
- **Les vues PUG sont routées et prêtes à la rédaction**
- **Le dossier de conception est prêt :**
 - **mise en page OK**
 - **Plan d'analyse, de conception et de développement structuré**
 - **les logos, pieds de page, titres, polices, tables des matières sont prêtes à l'emploi.**
- Le deuxième est qu'il va permettre de systématiser **une méthodologie de conception** qui pourra s'améliorer au fil du temps, à partir d'une base solide.
- Le dernier avantage est qu'il va permettre de **standardiser la forme de nos projets dans le dépôt GitHub**, qui pourrait devenir une *vitrine professionnelle* pour la future activité professionnelle en sortie de formation.

3.1.3.3.5.1. Code source

Mochizuki_ServerProjectGenerator.sh

[illegible]

PIZZAHUTTE

Création d 'un application Javascript FullStack complète

```

doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/bootstrap.css.map')
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css.map')

    link(rel='javascripts', href='/javascripts/bootstrap.min.js')

  body
    block content
EOF

echo "[Copie des fichiers..."
mkdir public/images/logos/
cd ..
cp ressources/mochizuki_logo.png ./$uservar/public/images/logos/mochizuki_logo.png
cp ressources/github_logo_large.png ./$uservar/public/images/logos/github_logo_large.png
cp ressources/github_logo_xsmall.png ./$uservar/public/images/logos/github_logo_xsmall.png
cp ressources/ConceptionProjet_Template.pdf ./$uservar/README.md
cp ressources/ConceptionProjet_Template.odt ./$uservar/README.md
cp ressources/commitAndPush.sh ./$uservar/commitAndPush.sh
cp ressources/README.md ./$uservar/README.md
cp ressources/ConceptionProjet_Template.pdf ./$uservar/ConceptionProjet_Template.pdf
cp ressources/ConceptionProjet_Template.odt ./$uservar/ConceptionProjet_Template.odt

echo "[Génération d'une page d'accueil"
rm ./$uservar/views/index.pug
touch ./$uservar/views/index.pug
cat >> ./$uservar/views/index.pug<< EOF
extends layout
block content
  .container.text-center
    img(src="./images/logos/mochizuki_logo.png" class="text-center")
    .row.align-items-center
      h1.col PIZZAHUTTE
    .row.mt-3
      form
        .mb-3
          .row
            .col-3.bg-white
              input#exampleInputEmail1.form-control(type='email', aria-describedby='emailHelp'
class="col")
            .col-3.bg-white
              label.form-label(for='exampleInputEmail1' class="col") Email
        .mb-3
          .row
            .col-3.bg-white
              input#exampleInputPassword1.form-control(type='password' class='col')
            .col-3.bg-white
              label.form-label(for='exampleInputPassword1') Mot de passe
        .mb-3
          a(href='#')
            button.btn.btn-primary.mb-2(type='submit') Se connecter
          a.text-decoration-none(href='#')
            p mot de passe oublié ?
        .mt-5
          a.text-decoration-none.mt-2(href='https://github.com/atsuhikoMochizuki')
            img(src="./images/logos/github_logo_xsmall.png" alt="")
            p https://github.com/atsuhikoMochizuki
            p @Atsuhiko_Mochizuki - 2023 - Tous droits réservés
EOF

echo "[Génération du script de lancement"
touch ./$uservar/DemarrageServeur.sh
cat >> ./$uservar/DemarrageServeur.sh << EOF
#Commande de lancement du serveur.
#Ce dernier est par défaut en écoute sur le port 3000.
#Soit localhost:8888 dans un navigateur

```

PIZZAHUTTE

Création d 'un application Javascript FullStack complète

```

printf "Mochizuki"
printf "ServerProjetGenerator v1.1"
printf "by Atsuhiko_Mochizuki - 2023\n"
echo "Lancement du serveur sur le port 3000..."
printf "${WARNING}"
npm start
EOF

touch ./$uservar/ATTENTION_aucun_dossier_.git_PRESENT
cat >> ./$uservar/ATTENTION_aucun_dossier_.git_PRESENT << EOF
Pour terminer la procédure de création du projet, copier le contenu du dossier présent dans un
projet Git cloné vierge, puis lancer le script commitAndPush.sh"
@Atsuhiko Mochizuki
EOF
printf "\n${SUCCESS}[]Génération du projet terminé.\n"

printf "\n${SUCCESS}[${WARNING}ATTENTION${SUCCESS}] Pour terminer la procédure, veuillez copier son
contenu dans un projet Git cloné vierge, puis lancer le script commitAndPush.sh${NC}\n"

```



```

// !** => "error"
function onError(error) {
  if (error.syscall !== "listen") {
    throw error;
  }
  var bind = typeof port === "string" ? "Pipe " + port : "Port " + port;
  /*Error handles*/
  switch (error.code) {
    case "EACCES":
      console.error(bind + " requires elevated privileges");
      process.exit(1);
      break;
    case "EADDRINUSE":
      console.error(bind + " is already in use");
      process.exit(1);
      break;
    default:
      throw error;
  }
}

//Miscellaneous functions
//=====
// Normalize a port into a number, string, or false
function normalizePort(val) {
  var port = parseInt(val, 10);
  if (isNaN(port)) {
    // named pipe
    return val;
  }
  if (port >= 0) {
    // port number
    return port;
  }
  return false;
}

// Launch default browser client on selected port
function launchClientBrowser(host) {
  console.log(`${P}${INFO}Lancement du browser coté client sur ${host}...`);
  const shellCommand = "xdg-open " + host;
  exec(shellCommand, (error, stdout, stderr) => {
    if (error) {
      console.error(`${FAILURE}[]...Lancement browser ECHEC: ${error}`);
      console.log(`stdout: ${stdout}`);
      console.error(`stderr: ${stderr}`);
      console.error(
        `${WARNING}[]Pour accéder à la page d'accueil entrez manuellement dans la page d'accueil de
votre navigateur l'adresse suivante:
${host}...${NO_COLOR}`
      );
    } else {
      console.log(`${P}${SUCCESS}...Lancement browser OK${NO_COLOR}`);
    }
  });
}

```


3.1.3.4.2. Interface de gestion de l'application Express (app.js)

```
// Dependencies
// =====
var createError = require("http-errors");
var express = require("express");
var path = require("path");
var cookieParser = require("cookie-parser");
var logger = require("morgan");
var indexRouter = require("./routes/index");
var usersRouter = require("./routes/users");
// =====

var app = express();

// view engine setup
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "pug");

// Middlewares
app.use(logger("dev"));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, "public")));
app.use("/", indexRouter);

app.use("/users", usersRouter);

// catch 404 and forward to error handler
app.use(function (req, res, next) {
  next(createError(404));
});

// error handler
app.use(function (err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get("env") === "development" ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render("error");
});

module.exports = app;
```

3.1.3.4.3. Routage administrateur (index.js)

```
var express = require("express");

var router = express.Router();


/* GET home page. */
router.get("/", function (req, res, next) {
  res.render("authentification", { title: "Pizzahutte" });
});

module.exports = router;
```

3.1.3.4.4. Routage des clients (users.js)

```
var express = require('express');
var router = express.Router();


/* GET users listing. */
router.get('/', function(req, res, next) {
  res.send('respond with a resource');
});

module.exports = router;
```


3.1.4. USA001 – 3 Création d'une base de données

3.1.5. USA001 – 4 Codage de la page d'authentification

3.1.5.1. *La puissance de l'infrastructure d'application WEB NodeJs EXPRESS*

Nous nous reposerons ici **sur toute la puissance de Express**.

Le codage de app.js que les routeurs sont des middlewares qui font être lancés lorsque la demande chargement de la page dédié sera demandée côté client :

```
app.use("/", indexRouter);  
app.use("/users", usersRouter);
```

Nous avons pour l'instant *deux routeurs* qui sont configurés, **admin** et **client** et prêts à générer les pages lorsque le client en fait la demande via des requêtes de son navigateur.

Observons la forme du router de l'administrateur de la pizzeria (index.js) :

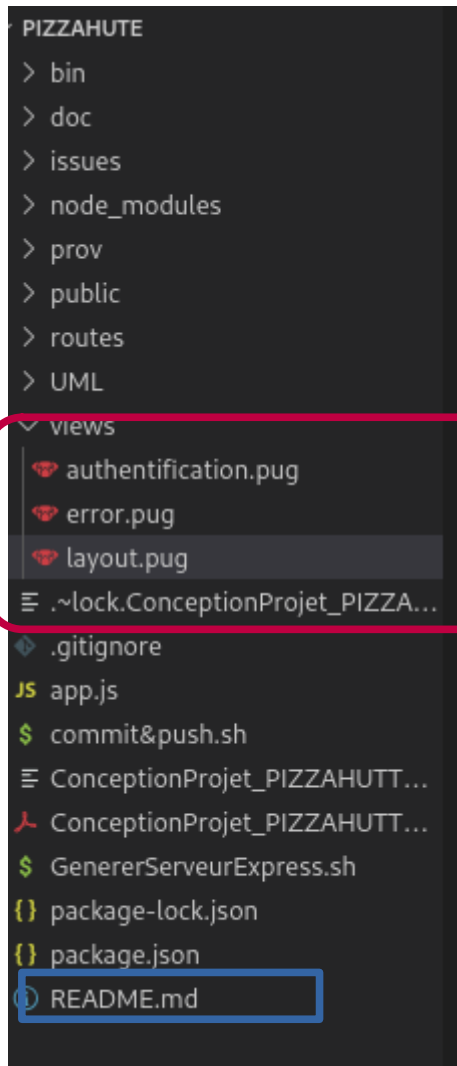
```
// Dependencies  
var express = require("express");  
var router = express.Router();  
  
//GET request  
router.get("/", function (req, res, next) {  
  res.render("index", {  
    title:  
      "PIZZAHUTTE-admin",  
  });  
});  
  
module.exports = router;
```

Nous renvoyons ici le rendu HTML de la vue index via la fonction de rappel.

Cette fonction utilise la méthode **render()** de la classe Application de l'API express (depuis notre objet). Cette dernière va ainsi permettre de pouvoir générer la page côté navigateur.

Le codage de la page d'authentification de fera donc ici dans le fichier index.

L'implémentation de cette partie nous fait découvrir la puissance de la surcouche applicative Express.



Notre serveur est paramétré et les routeurs sont en attente.

Il nous faut simplement afficher la page.

Nous utilisons le **moteur de vue PUG**.

Nous n'aurons donc dans notre application qu'à décrire, à l'aide de la syntaxe très intuitive et rapide de PUG, les éléments de la page dans la vue prévue à cet effet.

```
layout.pug
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
    link(rel='stylesheet', href='/bootstrap.min.css')
    meta(name="viewport" content="width=device-width, initial-scale=1.0")
  body
    block content
```

Express s'occupe alors de tous.

3.1.5.2. Organisation des vues

Nous avons choisi d'utiliser PUG dans notre application pour la génération de nos vues :

```
app.set("view engine", "pug");
```

Penchons nous un instant pour comprendre ce que peut nous apporter cet outil.

3.1.5.3. Le moteur de vue PUG

Pug est un **outil de templatage** qui permet de **générer du code HTML en compilant via une fonction Javascript**.

L'objectif de cet outil est de **se débarrasser des inconvénients que peut rencontrer un développeur front-end**. Il est utilisé en complément à **Node.js**.

Pug permet l'utilisation de **variables**. En compilant du code source Pug, ces dernières sont remplacées par les valeurs réelles, puis le résultat est envoyé au client dans une chaîne HTML.

3.1.5.3.1. Avantages

- La **simplification du code HTML**.
- L'utilisation de balises n'est plus nécessaire, grâce à un système d'indentations.
- Les classes et les id sont définis par des raccourcis, respectivement "." et "#".
- Le code obtenu est plus clair et plus agréable à lire.
- Le langage Pug est peu éloigné du Javascript, ce qui permet l'injection de code Javascript dans les fichiers Pug.
- A l'inverse du HTML *ou aucune indication n'est donnée en cas d'erreur* (cela sera visible uniquement à l'affichage de la page), le compilateur de Pug **repère directement les éventuelles erreurs**.
- Possibilité de définir des variables, d'ajouter de petits éléments de code comme des éléments de logique de base (if, each, unless...). On peut en outre écrire du JS directement depuis le fichier .pug, qui sera converti en page HTML.

3.1.5.3.2. Inconvénients

- l'utilisation d'indentations à la place du balisage peut s'avérer à double tranchant : une erreur d'indentation peut être fatale.
- peu d'IDEs prennent en charge le langage Pug : une demi-douzaine d'IDE seulement acceptent Pug, les plus connus étant Web Storm, PHP Storm et IntelliJ. Cela implique qu'il n'y a ni colorisation ni autosuggestions.
- Pug est incompatible avec le code HTML, c'est-à-dire qu'il est impossible de copier du code déjà existant. Cela pose évidemment un problème dans le cadre du refactoring, ou pour l'utilisations de parties de code HTML existants.

- Comme tous les langages, il faut passer par une étape d'installation et de familiarisation avec Pug. Cela s'avérer utile sur le long terme, mais il faut bien se poser la question du rapport temps passé à apprendre le langage par rapport au temps gagné sur le développement. La rentabilité augmente notamment avec la taille du projet et la diversité des pages HTML à écrire.

Les vues de notre projets seront organisées de la façon suivante :

- **index.pug**

Contiendra le contenu de la page d'authentification de l'administrateur de la pizzeria

- **users.js**

Contiendra le contenu de la page d'accueil du site

- **layout.pug**

Nous définirons dans cette vue un template de head qui sera appliqué à toutes les pages du site via la fonction Pug extends

3.1.5.4. Reprise nécessaire des fondamentaux de l'outil Bootstrap.

De nombreux points de frictions apparaissent rapidement avec Bootstrap, mettant en évidence une *incompréhension considérable sur la manière de penser et utiliser cet outil*, qui doit pourtant de coder très vite et responsive.

Nous allons donc sacrifier un temps considérable du projet à reprendre l'outils à la base, pour en tirer tous les bénéfices qu'il peut apporter en tant que développeur Front.

Les bénéfices en seront très profitables pour la suite de l'évolution du panels d'outils efficaces à maîtriser impérativement.

3.1.5.4.1. En résumé : ce qu'il est fondamental de saisir du système Bootstrap, comment penser cet outil puissant et responsive.(apporter précisions)

Le site de Bootstrap contient une documentation très bien faite, il n'est pas utile de retenir les noms des propriétés, qui, par ailleurs sont très intuitives et se mémorisent rapidement avec la pratique.

Ce qui est important de saisir, c'est plutôt la manière de penser l'outil. Voici ce que nous en avons pu en déduire après une étude approfondie de la documentation :

On utilise des **grilles flexbox pour agencer la page**.

Une grille Bootstrap est **hiérarchisée** de la manière suivante :

- Le **container** est l'**élément parent le plus haut dans la hiérarchie**. Il va **centrer et remplir horizontalement le contenu**, en **s'adaptant aux différents types de support**, selon six niveaux de taille d'écran.

Il enveloppe des rangées ou lignes ou rows.

Pour que le container utilise *toute la largeur de l'écran*, on le suffixe par **-fluid**.

Lorsque l'espace pour contenir les éléments **n'est plus suffisant**, il est possible **d'envoyer les éléments à la ligne**, afin qu'ils ne soient pas coupés à l'affichage, à l'aide de la propriété **wrap** :

- Nota : Il faudra dans ce cas préciser la propriété d-flex :
 - d-flex flex-nowrap
 - d-flex flex-wrap
 - flex-wrap-reverse.
- **Une row**, enveloppée dans un container, va permettre, elle-même, **d'envelopper des colonnes** (12 col par row).

Ce dans le but de pouvoir :

- **Aligner les colonnes qu'elle contient**, c'est à dire à *quelles position dans sa hauteur ces colonnes vont s'aligner*, à l'aide de la propriété **align-items ***:

- **align-items-start**
- **align-items-center**
- **align-items-end**
- **Répartir l'espace horizontal entre les col qu'elle contient**, à l'aide de la propriété **justify-content-*** :
 - **justify-content-start**
 - **ustify-content-center**
 - **justify-content-end**
 - **justify-content-around**
 - **justify-content-between**
 - **justify-content-evenly**
- Les **colonnes** (col), enveloppées par la row, vont permettre in fine de **placer notre contenu**.

A la manière de la propriété align-items-* pour les row, **align-self-*** permet le **même mécanisme à échelle individuelle** pour chaque colonne dans la row :

- **col align-self-start**
- **col align-self-center**
- **col align-self-end**

Nota : attention de bien faire attention que *align-self-** est une propriété dédiées aux col.

Ce petit exemple montre comment disposer les éléments facilement avec bootstrap :

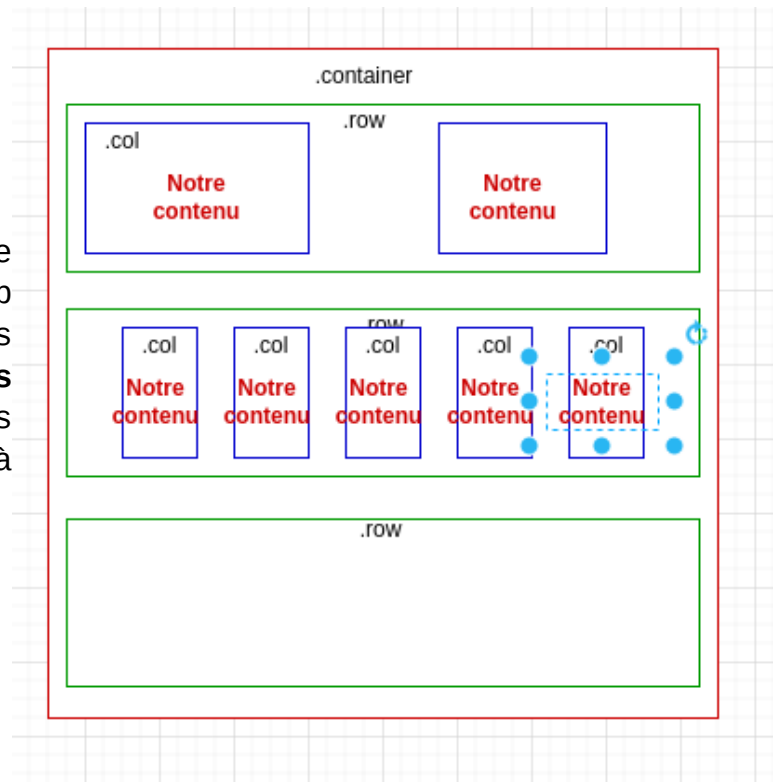
```
<div class="container text-center">
  <div class="row align-items-center justify-content-between">
    <div class="col align-self-center">
      1 of 2
    </div>
    <div class="col align-self-start">
      2 of 2
    </div>
  </div>
</div>
```

```

</div>
<div class="row">
  <div class="col">
    1 of 3
  </div>
  <div class="col">
    2 of 3
  </div>
  <div class="col">
    3 of 3
  </div>
</div>
</div>

```

Nota : Je remarque que sur le site de Bootstrap tous les exemples montrent **des classes associées à des div**. Nous procéderons ainsi à l'avenir.



3.1.5.5. Codage du head commun aux vues (layout .pug)

3.1.5.5.1. Particularités rencontrées

3.1.5.5.2. Liaison des dépendances

Elle sont réalisées à l'aide de la balise **link**, où l'attribue **rel** permet de spécifier la nature de la relation entre les ressources.

- Nos feuilles de styles posséderont l'attribut 'stylesheet'
- Nos scripts js l'attribut 'javascripts'
- Notre favicon 'shortcut icon'

Si elles sont locales, nos dépendances seront placées dans le dossier 'public' du projet.

Nous créons donc un raccourci vers ce dossier, dans notre application Express (app.js) :

```
app.use(express.static(path.join(__dirname, "public")));
```

Il pourra ainsi être utilisé dans nos middleware.

Nota : Nous créons aussi, mais avec app.set, le chemin vers le dossier des vues

```
app.set("views", path.join(__dirname, "views"));
```

Nous pouvons à présent lier nos dépendance de la façons suivante :

```
link(rel='stylesheet', href='/stylesheets/style.css')
```

Afin de garder le même schéma, nous avons placés nos bibliothèques bootstrap dans le dossier public.

3.1.5.5.3. Code source de layout.pug

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
    link(rel="stylesheet", href="https://fonts.googleapis.com/css2?family=Gloria+Hallelujah&display=swap")
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/bootstrap.css.map')
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css.map')
    link(rel='javascripts', href='/javascripts/bootstrap.min.js')
    link(rel="shortcut icon", href="/images/favicon.ico")
  body
    block content
```

3.1.5.5.4. Codage de la page d'authentification de l'administrateur(index.pug)

```
extends layout
block content
  .container.text-center
    img(src="./images/logos/logo.png" class="text-center mb-2")
    h1 PIZZAHUTTE
    .row.align-items-center
    .row.mt-3
    form
      .mb-3
      .row
        .col-3.bg-white
          input#exampleInputEmail1.form-control(type='email', aria-
describedby='emailHelp' class="col")
        .col-3.bg-white
```

PIZZAHUTTE

Création d'un application Javascript FullStack complète

```

    label.form-label(for='exampleInputEmail1' class="col") Email
.mb-3
    .row
        .col-3.bg-white
            input#exampleInputPassword1.form-control(type='password' class='col')
        .col-3.bg-white
            label.form-label(for='exampleInputPassword1') Mot de passe
.mb-3
    a(href='#')
        button.btn.btn-primary.mb-2(type='submit') Se connecter
    a.text-decoration-none(href='#')
        p mot de passe oublié ?
.mb-5
    a.text-decoration-none.mb-2(href='https://github.com/atsuhikoMochizuki')
        img(src="./images/logos/github_logo_xsmall.png" alt="")
    p https://github.com/atsuhikoMochizuki
    p @Atsuhiko_Mochizuki - 2023 - Tous droits réservés

```

3.1.5.6. *Aperçu côté Front*



Email

Mot de passe

Se connecter

[mot de passe oublié ?](#)



<https://github.com/atsuhikoMochizuki>

@Atsuhiko_Mochizuki - 2023 - Tous droits réservés

- 3.1.6. USA001 – 5 Fonctions de récupération de la saisie utilisateur**
- 3.1.7. USA001 – 6 administration de la base de données**
- 3.1.8. USA001 – 7 Codage de la page d'accueil**
- 3.1.9. USA001 – 8 Codage de l'automatisation d'envoi de mail.**

3.2. USA002 : Naviguer sur dans tout le site à l'aide d'une barre de menu

En tant qu'administrateur, je souhaite avoir accès à une barre de menu sur toutes les pages permettant d'aller sur les listes des différentes rubriques (utilisateur, pizza, client,...).

L'objectif est de mettre en place la structure générale du site.

Les liens disponibles :

- *Utilisateurs*
- *Pizzas*
- *Menus*
- *Desserts*
- *Boissons*
- *Clients*
- *Commandes*
- *Livreurs*
- *Ingrédients*
- *Statistiques*
- *Promotions*

Les liens sont désactivés tant que la fonctionnalité associée n'est pas développée.

La zone activités reste grisée pour le moment.

3.2.1. Faire bifurquer le code PUG à l'aide de la commande block

Nous devons coder une barre de navigation du site, ce qui va nous permettre par la suite **de mettre en place la structure générale du site.**

Etant donné que ce menu sera **commun** à plusieurs pages, nous allons le placer dans le **layout.PUG**.

En revanche, *il ne doit pas apparaître dans la page d'authentification.*

Nous allons donc permettre au code de layout.pug une **bifurcation dédiée** à **index.js** de la façon suivante à l'aide de l'instruction block :

```
layout.pug
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
    link(rel="stylesheet", href="https://fonts.googleapis.com/css2?family=Gloria+Hallelujah&display=swap")
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/bootstrap.css.map')
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css.map')
    link(rel='javascripts', href='/javascripts/bootstrap.min.js')
    link(rel="shortcut icon", href="/images/favicon.ico")
  body
    block autent-admin
    //Le menu sera codée dans cette zone
    block mainContent
```

3.2.2. Création de la vue principale pour pouvoir afficher le menu

Etant donné que nous n'avons qu'une seule page générée, la page d'authentification admin, et que cette dernière ne doit pas contenir de menu, nous devons **créer la vue de la page d'accueil pour tester notre menu.**

Elle sera accessible à **http://localhost/users/**

Nous pouvons dès le départ insérer la référence « mainContent » qui va nous permettre **de faire la liaison**, à l'aide de la fonction **block**, avec le **layout.pug**

3.2.2.1. Configuration du router *users.js*

3.2.2.1.1. Affectation de la vue de la page d'accueil à la route principale »/ » du routeur des utilisateurs (*users.js*).

La route principale aura pour rôle de générer la page d'accueil.

Nous créons donc dans le dossier *views* une nouvelle vue que nous nommons **users.pug**.

Puis nous affectons cette vue à la route principale du routeur :

users.js

```
// Dependencies
var express = require("express");
var router = express.Router();

/* GET users listing. */
router.get("/", function (req, res, next) {
  res.render("users", {
    title: "PIZZAHUTTE - Pizzeria à Perpette-les-pranades(Doubs)",
  });
});
module.exports = router;
```

Ceci fait, nous pouvons à présent coder notre petit menu

3.2.3. Code source du menu de admin

En accord avec les critères du cahier des charges pour la fonctionnalité USA0002, le code source du menu du site sera la suivant :

layout.pug

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
    link(rel="stylesheet", href="https://fonts.googleapis.com/css2?family=Gloria+Hallelujah&display=swap")
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css')
    link(rel='stylesheet', href='/stylesheets/bootstrap.css.map')
    link(rel='stylesheet', href='/stylesheets/bootstrap.min.css.map')
    link(rel='javascripts', href='/javascripts/bootstrap.min.js')
    link(rel="shortcut icon", href="/images/favicon.ico")
  body
    block admin-authentification
      header
        .container-fluid.border.border-success.text-center
          .row.border.border-info
            .col-3.border.border-warning
              .row.border.border-success.d-flex.flex-column.align-items-center
                .col.border
                  img(src="../images/logos/logo_white_no_lines.png" class="text-center mb-2")
                .col.border
                  h1 PIZZAHUTTE
            .col.border.border-warning.align-self-center
              ul.nav.nav-underline
                li.nav-item
                  a.nav-link.disabled Utilisateurs
                li.nav-item
                  a.nav-link.disabled Pizzas
                li.nav-item
                  a.nav-link.disabled Menus
                li.nav-item
                  a.nav-link.disabled Desserts
                li.nav-item
                  a.nav-link.disabled Boissons
                li.nav-item
                  a.nav-link.disabled Clients
                li.nav-item
                  a.nav-link.disabled Commandes
                li.nav-item
                  a.nav-link.disabled Livreurs
                li.nav-item
                  a.nav-link.disabled Ingrédients
                li.nav-item
                  a.nav-link.disabled Statistiques
                li.nav-item
                  a.nav-link.disabled Promotions
          .row.border.border-info
            .col.border
```

PIZZAHUTTE

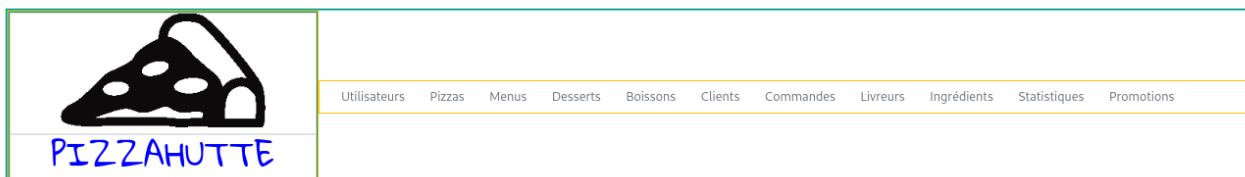
Création d 'un application Javascript FullStack complète

```

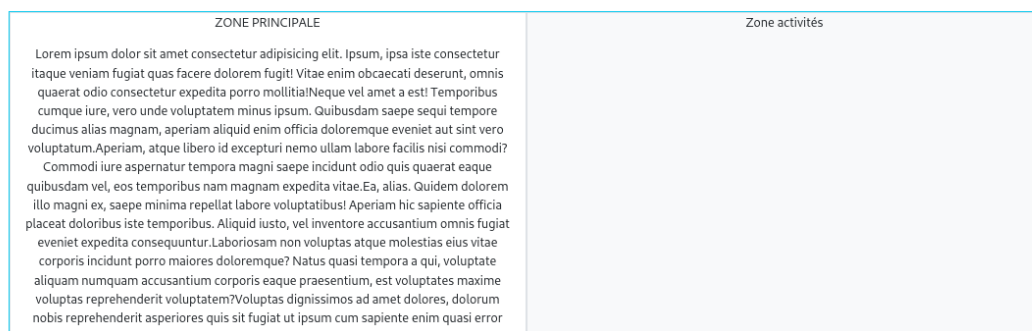
    p ZONE PRINCIPALE
.col.border.bg-light
    p.bg-light Zone activités
block mainContent

```

3.2.4. Aperçu côté navigateur depuis la page d'accueil clients users.pug



ENTREE SUR LA PAGE USERS



4. Annexes