



atsuhikoMochizuki

PIZZAHUTTE

Création d'une application Javascript FullStack
complète



PIZZAHUTTE

Système de gestion administrative d'une pizzeria
Journal de bord de la conception



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.....	6
Posons-nous.....	10
2. Analyse.....	11
2.1. Diagramme des cas d'utilisation.....	13
2.2. Organisation sous forme de sprints.....	14
2.3. Découpage fonctionnel des sprints.....	15
3. LANCEMENT DES SPRINTS.....	16
3.1. USA001 – Authentification page login.....	17
3.1.1. Analyse de la tâche.....	18
3.1.2. Planification du sprint.....	19
3.1.3. USA001 – 1 Création d'un serveur HTTP.....	20
3.1.3.1. Modélisation de la fonctionnalité.....	20
3.1.3.1.1. Premier Jet.....	20
3.1.3.1.1.1. Diagramme de séquence.....	20
3.1.3.1.1.2. Diagramme des classes participantes.....	21
3.1.3.1.2. Second jet.....	22
3.1.3.1.2.1. Diagramme de séquence corrigé et approfondi.....	22
3.1.3.1.2.2. Dédution d'un diagramme des classes participantes beaucoup plus réaliste.....	24
3.1.3.2. Choix des technologies.....	25
3.1.3.2.1. Génération des pages côté serveur.....	25
3.1.3.2.2. Choix du framework NodeJS pour l'implémentation du serveur en Javascript.....	25
3.1.3.2.2.1. Surcouche applicative Express.....	26
3.1.3.2.2.2. Génération des pages : moteur de vue PUG.....	26
3.1.3.2.3. Mise page du front.....	26
3.1.3.2.3.1. Choix de la librairie Bootstrap pour optimiser le temps de développement et le responsive du front.....	26
3.1.3.2.4. Système de gestion de base de données relationnelles :.....	27
3.1.3.2.5. Outils de scripting : BASH.....	27
3.1.3.3. Mise en place de environnement de travail.....	27
3.1.3.3.1. Structuration de l'arborescence du projet.....	27
3.1.3.3.2. Création du dépôt github et clonage du projet vierge.....	27
3.1.3.3.3. Création du projet node via npm.....	27
3.1.3.3.3.1. installation des dépendances.....	27
3.1.3.3.3.2. Paramétrage du fichier de configuration package.json.....	27
3.1.3.3.3.2.1.1. Affectation de nodemon à la commande start de l'objet script.....	27
3.1.3.3.3.2.2. Prise en charge des modules ECMAScript (import, export).....	27
3.1.3.3.4. Consignation de la procédure dans un script BASH. Permettra de générer immédiatement le projet en cas de « pépin ».....	28
3.1.3.3.5. Création pour l'occasion d'une petite application : le MochizukiServerProjectGenerator.....	29

3.1.3.3.5.1. Code source.....	30
3.1.3.4. Codage de la fonctionnalité.....	33
3.1.3.4.1. Serveur.....	33
3.1.3.4.2. Interface de gestion de l'application Express (app.js).....	35
3.1.3.4.3. Routage administrateur (index.js).....	36
3.1.3.4.4. Routage des clients (users.js).....	36
3.1.3.4.4.1. Remarques.....	37
3.1.3.4.4.1.1. Création de launchClientBrowser et implémentation d'une fonction d'appel système en Javascript.....	37
3.1.4. USA001 – 3 Création d'une base de données.....	38
3.1.4.1. Incorporer une base de données à notre projet.....	38
3.1.4.1.1. Choix du système de gestion de la base de données (SGBDR) : MySQL.....	38
3.1.4.1.1.1. Installation des outils MySQL.....	39
3.1.4.1.1.1.1. Installation du serveur MySQL.....	39
3.1.4.1.1.1.1.1. Démarrage.....	39
3.1.4.1.1.1.1.2. Redémarrage.....	39
3.1.4.1.1.1.1.2.1. Rechargement de la configuration.....	39
3.1.4.1.1.1.1.2.2. Forcer la prise en compte de la nouvelle configuration....	39
3.1.4.1.1.1.1.2.3. Connaître la version.....	39
3.1.4.1.1.1.2. Création de la base de données avec un utilisateur associé.....	40
3.1.4.1.1.1.3. Sélection de la base de données.....	41
3.1.4.1.1.1.4. Installation du client MySQL : SQLTools.....	41
3.1.4.1.1.1.4.1. Installation du driver MySQL.....	41
3.1.4.1.1.1.5. Connexion à la base de données.....	42
3.1.4.1.1.1.6. Modélisation de la base de données.....	45
3.1.4.1.1.1.7. Accès dynamique à la base de données.....	45
3.1.4.1.1.2. Gestion d'une base données.....	45
3.1.4.1.1.2.1. Création de la connexion à la base de données.....	46
3.1.4.1.1.3. Appel de la fonction de connexion depuis notre application app.js.....	46
3.1.4.1.1.4. Appel des fonctions de gestion des requêtes SQL de l'utilisateur : newPwd_query.js.....	48
USA001 – 4 Codage de la page d'authentification.....	49
3.1.4.2. La puissance de l'infrastructure d'application WEB NodeJs EXPRESS.....	49
3.1.4.3. Organisation des vues.....	50
3.1.4.4. Le moteur de vue PUG.....	51
3.1.4.4.1. Avantages.....	51
3.1.4.4.2. Inconvénients.....	51
3.1.4.5. Reprise nécessaire des fondamentaux de l'outil Bootstrap.....	52
3.1.4.5.1. En résumé : ce qu'il est fondamental de saisir du système Bootstrap, comment penser cet outil puissant et responsive.(apporter précisions).....	53
3.1.4.6. Codage du head commun aux vues (layout .pug).....	56
3.1.4.6.1. Particularités rencontrées.....	56
3.1.4.6.2. Liaison des dépendances.....	56
3.1.4.6.3. Code source de layout.pug.....	56

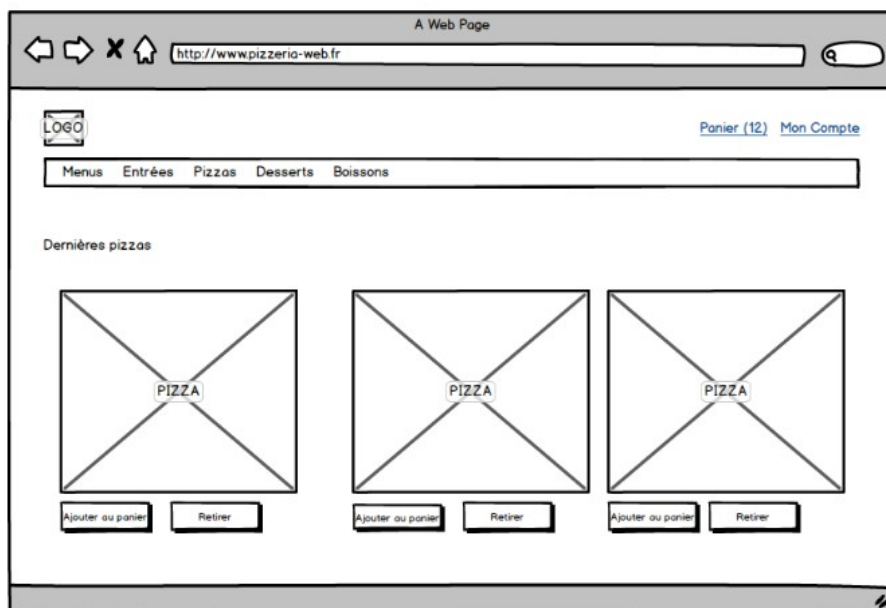
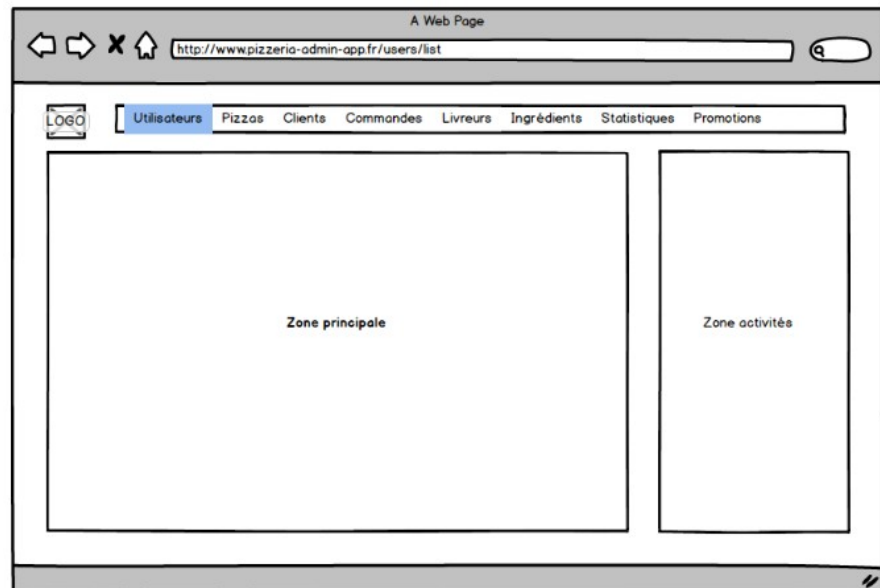
3.1.4.6.4. Codage de la page d'authentification de l'administrateur(index.pug).....	57
3.1.4.7. Aperçu côté Front.....	58
3.1.5. USA001 – 5 Fonctions de récupération de la saisie utilisateur.....	59
3.1.6. USA001 – 6 administration de la base de données.....	59
3.1.7. USA001 – 7 Codage de la page d'accueil.....	59
3.1.8. USA001 – 8 Codage de l'automatisation d'envoi de mail.....	59
3.2. USA002 : Naviguer sur dans tout le site à l'aide d'une barre de menu.....	60
3.2.1. Faire bifurquer le code PUG à l'aide de la commande block.....	61
3.2.2. Création de la vue principale pour pouvoir afficher le menu.....	61
3.2.2.1. Configuration du router users.js.....	62
3.2.2.1.1. Affectation de la vue de la page d'accueil à la route principale »/ » du routeur des utilisateurs (users.js).....	62
3.2.3. Code source du menu de admin.....	63
3.2.4. Aperçu côté navigateur depuis la page d'accueil clients users.pug.....	64
4. Annexes.....	65
4.1. Les bases de données.....	66
4.1.1. Définition.....	66
4.1.2. Conception des bases de données relationnelles.....	67
4.1.2.1. Modélisation conceptuelle.....	68
4.1.2.1.1. Les entités.....	68
4.1.2.1.2. Les attributs : Caractéristiques et propriétés des entités.....	69
4.1.2.1.3. Les relations.....	69
4.1.2.1.3.1. Cardinalité.....	69
4.1.2.1.3.1.1. Cardinalité un à un.....	70
4.1.2.1.3.1.2. Cardinalité un à plusieurs.....	70
4.1.2.1.3.1.3. Cardinalité plusieurs à plusieurs.....	71
4.1.2.1.4. L'identifiant.....	72
4.1.2.1.5. Comment construire un schéma conceptuel.....	74
4.1.2.2. Modélisation logique relationnelle.....	75
4.1.2.2.1. Le domaine.....	75
4.1.2.2.2. L'attribut.....	76
4.1.2.2.3. La relation ou table.....	76
4.1.2.2.4. Les contraintes d'intégrité.....	77
4.1.2.2.5. La théorie de la normalisation.....	78
4.1.2.2.5.1. Règles à suivre pour concevoir un schéma relationnel :.....	79
4.1.2.2.5.1.1. Règle I : Toute entité est traduite en une table relationnelle dont les caractéristiques sont les suivantes :.....	79
4.1.2.2.5.1.2. Règle 2 : Toute relation binaire plusieurs à plusieurs est traduite en une table relationnelle dont les caractéristiques sont les suivantes :.....	79
4.1.2.2.5.1.3. Règle III : Toute relation binaire un à plusieurs est traduite :.....	81
4.1.2.2.5.1.4. Règle 4 : Toute relation binaire un à un est traduite, au choix, par l'une des trois solutions suivantes :.....	82
4.1.2.2.6. Exemple de traduction d'un schéma conceptuel en schéma relationnel.....	83
4.1.2.3. Le langage SQL.....	85

4.1.2.3.1. Les opérateurs de l'algèbre relationnelle.....	85
4.1.2.3.1.1. Les opérateurs de base.....	86
4.1.2.3.1.1.1. Projection.....	86
4.1.2.3.1.1.2. Sélection.....	87
4.1.2.3.1.1.3. Jointure (R1,R2, condition d'égalité entre attributs).....	88
4.1.2.3.1.2. Les opérateurs ensemblistes.....	90
4.1.2.3.1.2.1. UNION(R1,R2).....	90
4.1.2.3.1.2.2. INTERSECTION(R1,R2).....	90
4.1.2.3.1.2.3. DIFFERENCE (R1, R2).....	92
4.1.2.3.1.2.4. PRODUIT (R1, R2).....	93
4.1.2.3.2. Commandes SQL.....	94
4.1.2.3.2.1. Création, modification de table.....	94
4.1.2.3.2.2. Interrogation des données.....	94
4.1.2.3.3. Exemple de traduction d'un schéma relationnel en langage SQL :.....	96
4.1.3. Conception de la base de données avec MySQLwork-bench.....	97

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, des bases de données 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 méthodes qui nous sembleront les plus judicieux pour mettre en œuvre le développement de l'application.

Nous pourrons alors aborder le codage de notre modèle en ayant mis de notre côté le maximum d'atouts pour concevoir un système cohérent et 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.

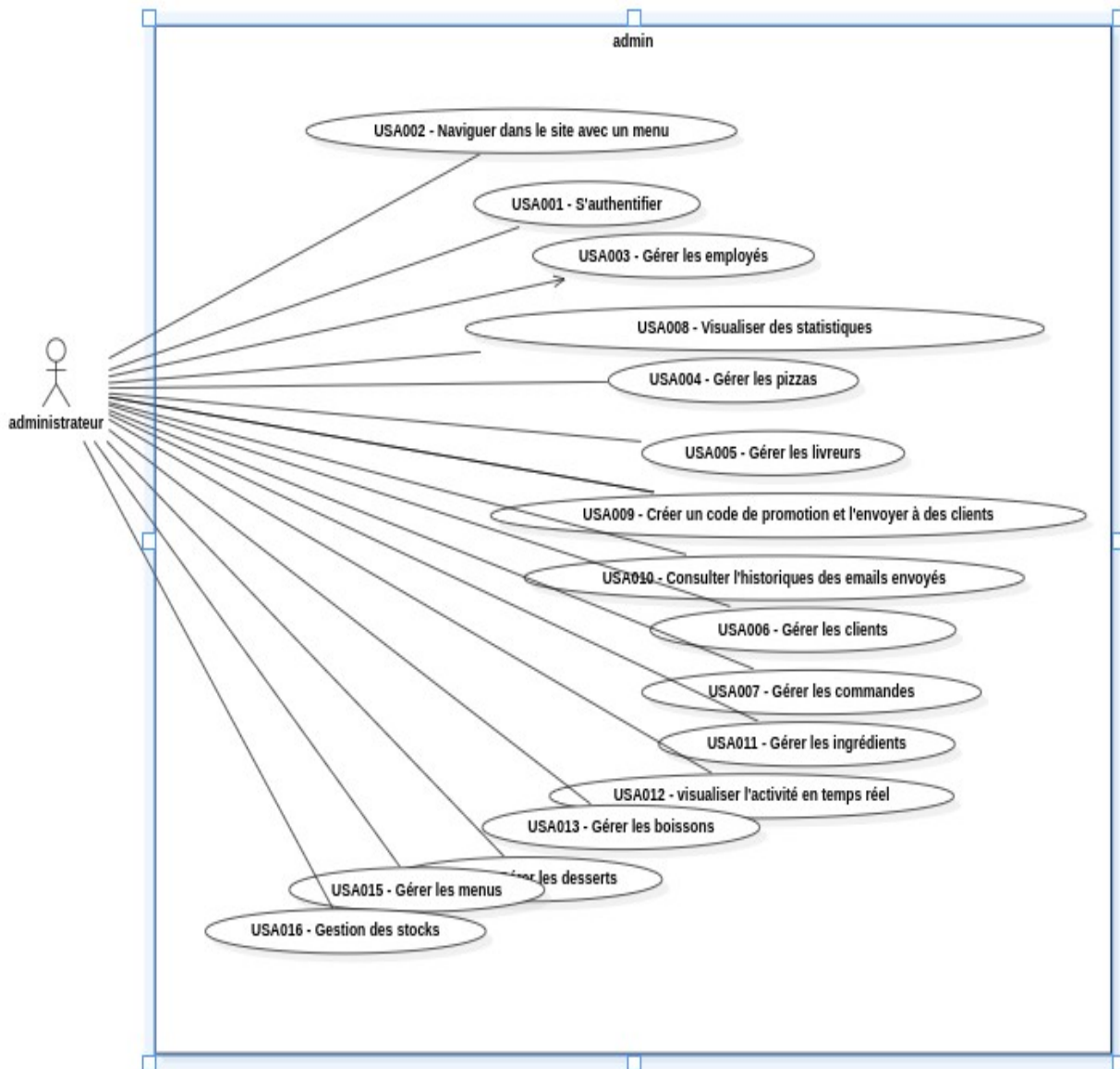
Un exercice très facile qui a l'avantage de pouvoir nous *rattacher à un de point départ*, ce qui écarte d'office le syndrome de « feuille blanche ». Il impose en outre :

- **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

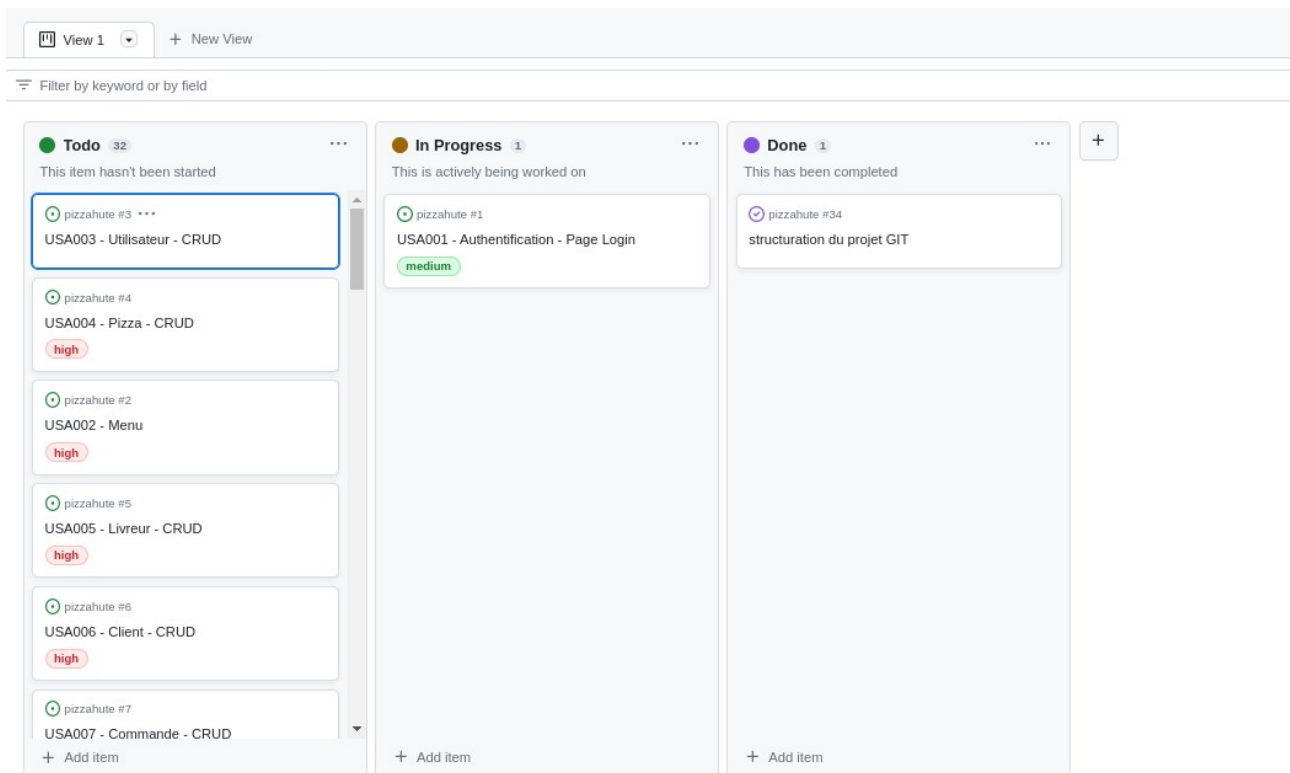
Au vu du travail à réaliser, le temps imparti ne nous permettra dans l'état actuel de nos connaissances des langages de modéliser l'application entière.

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 – Authentification 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 un lien Régénérer un nouveau mot de passe et le stocker	Gérer une boucle évènementielle Implémenter de la logique Administrer un SGBD	Créer un serveur HTTP ou un socket Gérer une application temps réel Mettre en œuvre un serveur de donnée Créer une base de données
Afficher une page d'authentification et d'accueil	Envoyer un mail Générer des pages WEB	Automatiser l'envoi de mail Implémenter de la logique Router les requêtes	Créer un serveur HTTP Implémenter une application Créer un serveur HTTP
Vérifier les informations de connexion	Récupérer de la saisie utilisateur Les comparer à des données	Gérer une boucle évènementielle Implémenter de la logique	Implémenter du html, css et du javascript Créer un serveur HTTP ou un socket Gérer une application

administrer
données

des
Administer un SGBD
Créer une base de
données

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

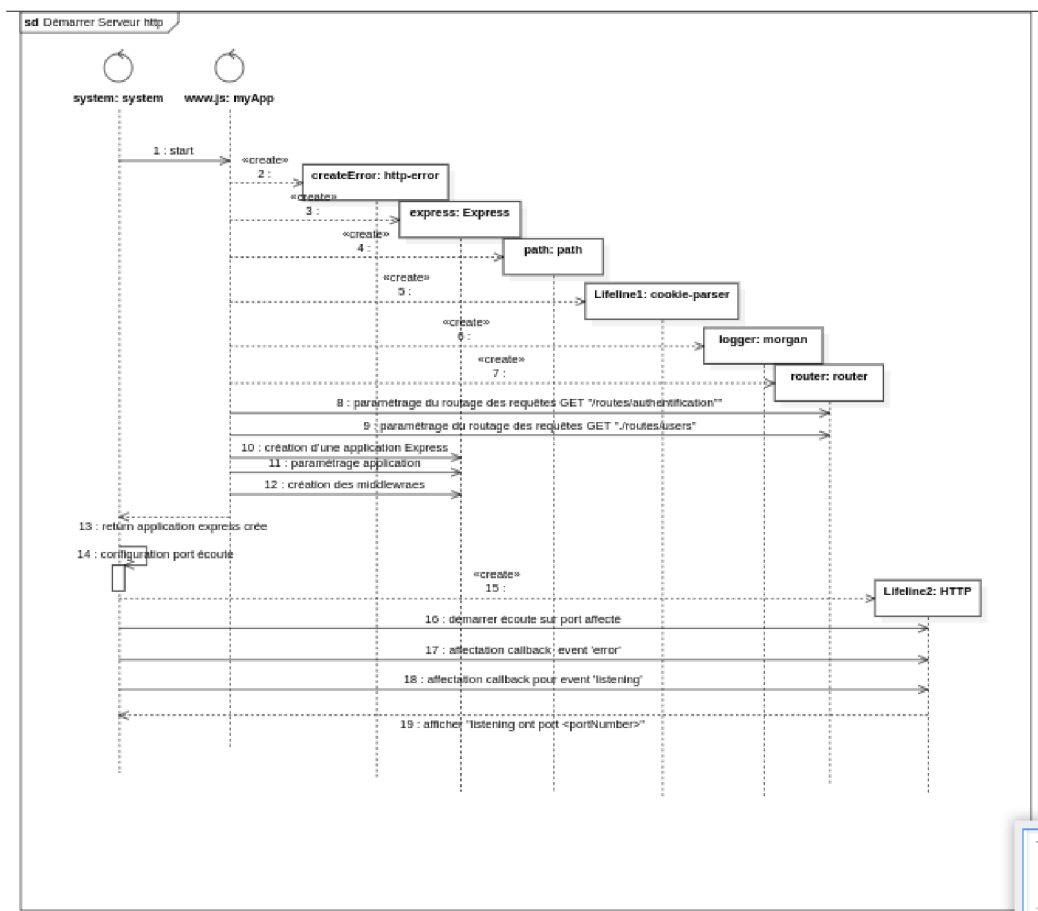
USA001 – 8 Codage de l'automatisation d'envoi de mail.

3.1.3. USA001 – 1 Création d'un serveur HTTP

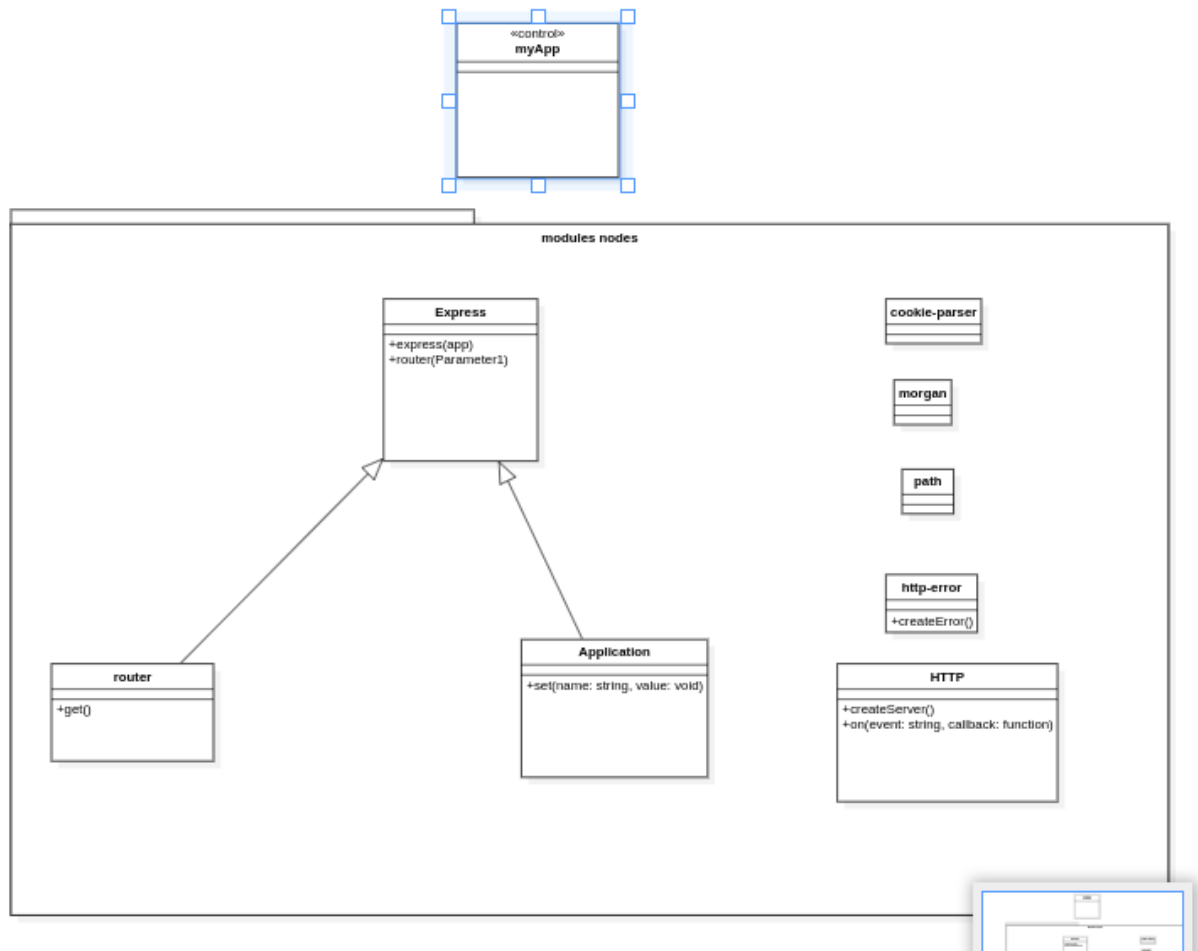
3.1.3.1. Modélisation de la fonctionnalité

3.1.3.1.1. Premier Jet

3.1.3.1.1.1. Diagramme de séquence



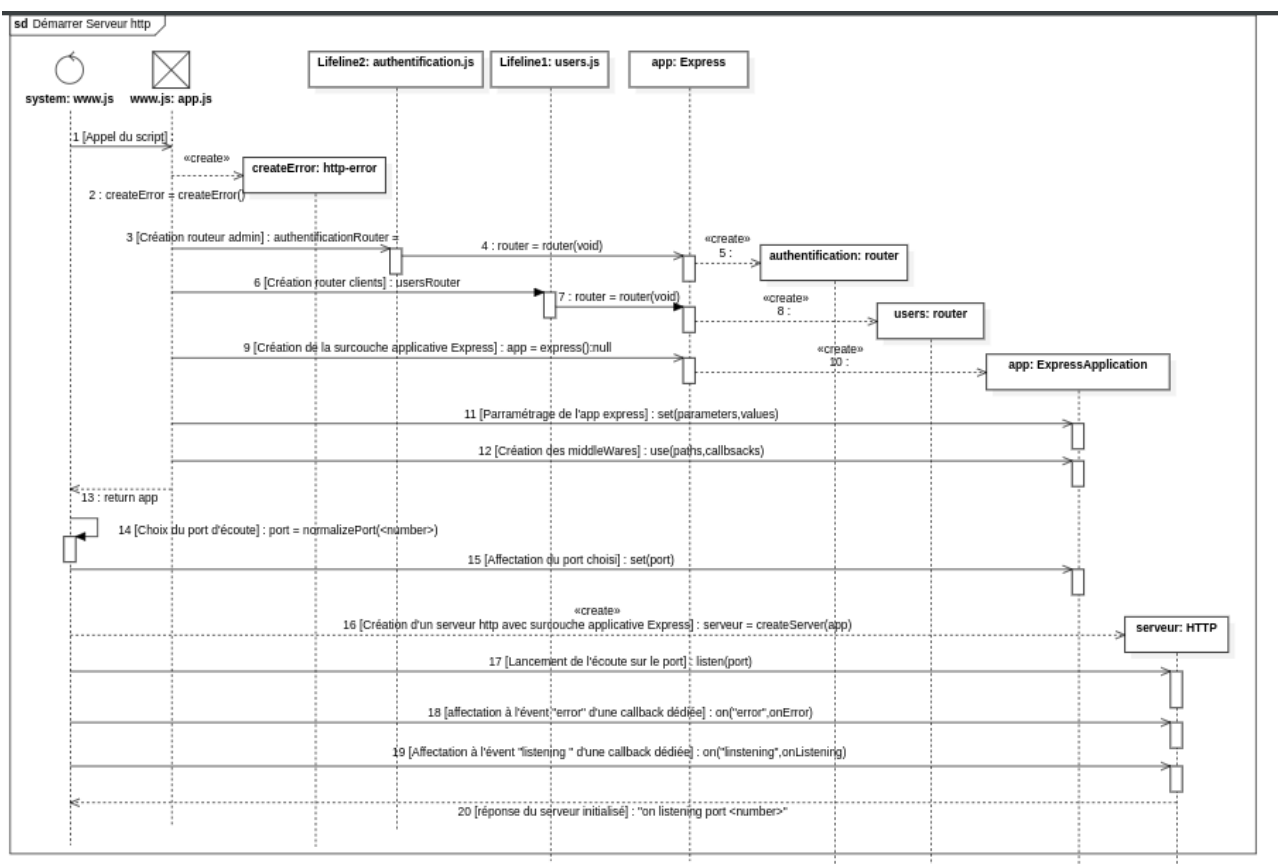
3.1.3.1.2. Diagramme des classes participantes

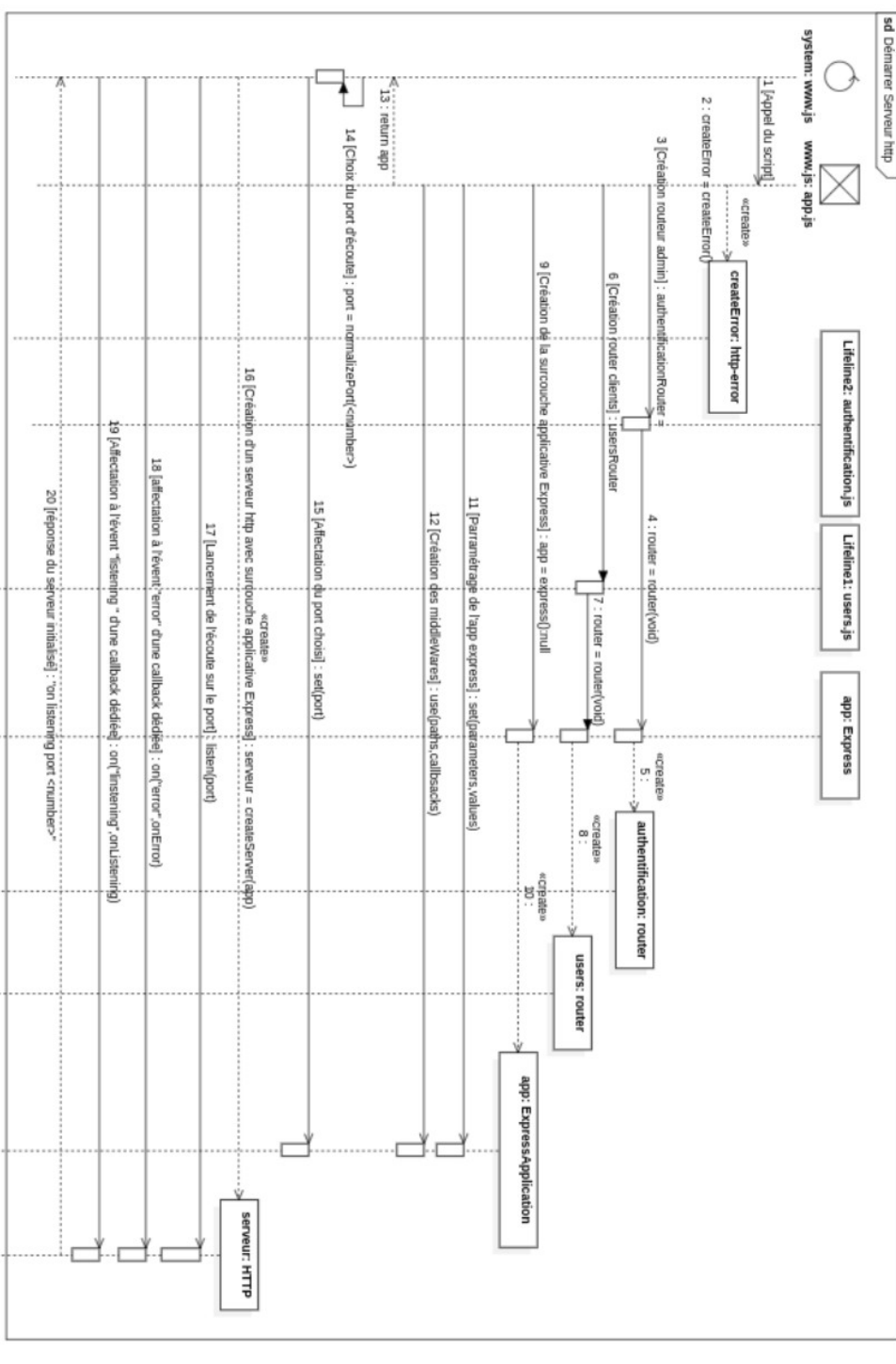


3.1.3.1.2. Second jet

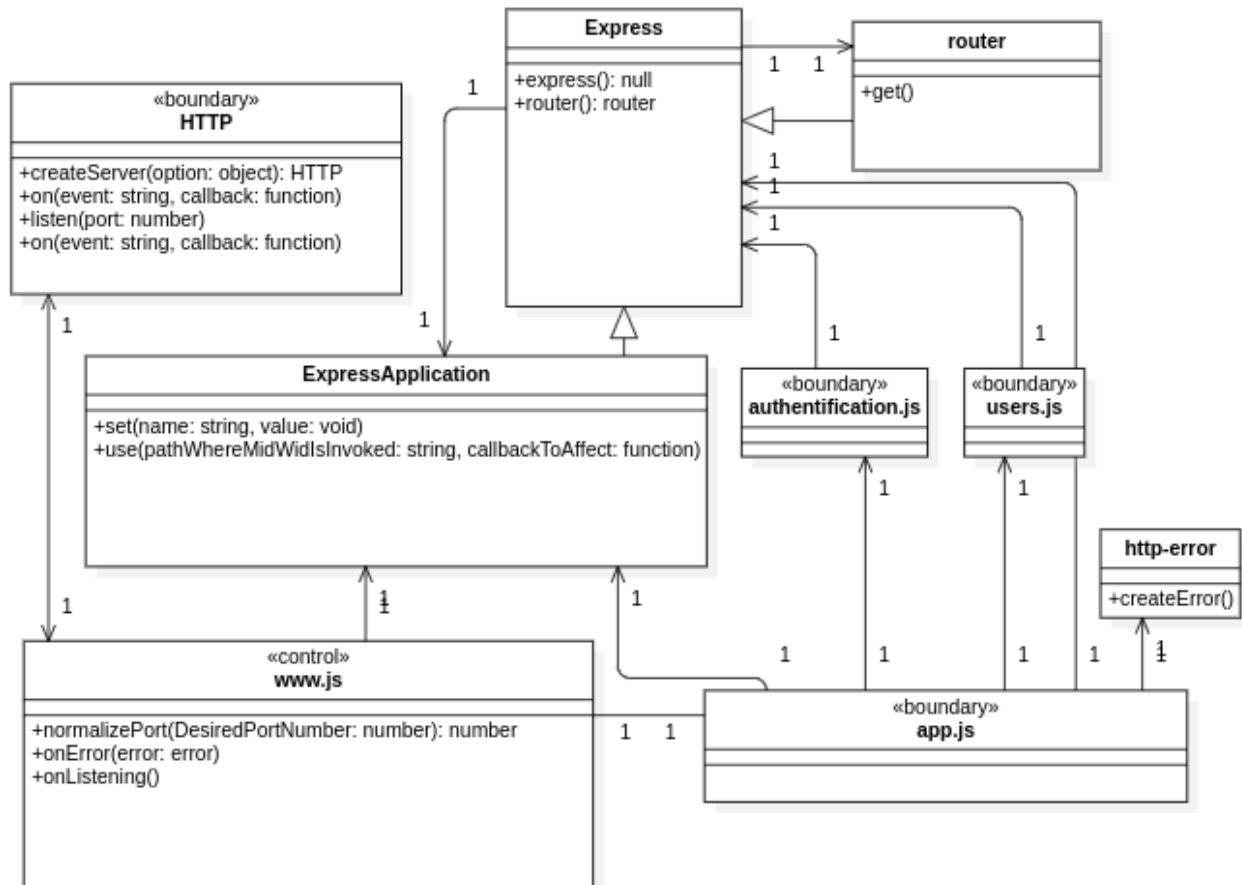
Nous allons affecter les méthodes des classes aux différentes interactions entre les objets de l'application

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
```



```
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
```

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]

```

echo "[ --socket.io"
npm install socket.io --save
echo "[ --body-parser"
npm install body-parser --save

echo "[Copie de la bibliothèque Bootstrap dans le dossier public"
cp ./node_modules/bootstrap/dist/css/bootstrap.min.css ./public/stylesheets/bootstrap.min.css
cp ./node_modules/bootstrap/dist/css/bootstrap.css.map ./public/stylesheets/bootstrap.css.map
cp ./node_modules/bootstrap/dist/css/bootstrap.min.css.map
./public/stylesheets/bootstrap.min.css.map
cp ./node_modules/bootstrap/dist/js/bootstrap.min.js ./public/javascripts/bootstrap.min.js

echo "[Insertion du lien vers la bibliothèque Bootstrap dans le layout général des vues"
rm ./views/layout.pug
touch ./views/layout.pug
cat >> ./views/layout.pug<< EOF
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

```

PIZZAHUTTE

Création d 'un application Javascript FullStack complète

```

        .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

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"

```



```
//=====
// !** => "listening"
function onListening() {
  var addr = server.address();
  var bind = typeof addr === "string" ? "pipe " + addr : "port " + addr.port;
  debug("Listening on " + bind);
}

// !** => "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

Nous allons pour cela utiliser mySQL.

Ce qui implique de devoir nous former à l'outil.

La recherche sur ce sujet est consignée en annexes.

3.1.4.1. *Incorporer une base de données à notre projet*

Nous commencerons par choisir notre système de base de données. Nous choisirons un système de base de données relationnelles.

3.1.4.1.1. Choix du système de gestion de la base de données (SGBDR) : MySQL

MySQL est un système de gestion de bases de données relationnelles (SGBDR).

Il est distribué sous une double licence GPL et propriétaire. Il fait partie des logiciels de gestion de base de données les plus utilisés au monde², autant par le grand public (applications web principalement) que par des professionnels, en concurrence avec Oracle, PostgreSQL et Microsoft SQL Server.

MySQL est un système de gestion de bases de données relationnelles (SGBDR) propriétaire, gratuit, performant, très populaire, multi-threadé, multi-utilisateurs...

MySQL appartient à Oracle. Il existe un fork open-source, plus communautaire, 100% compatible MySQL et a priori plus performant créé et maintenu par Michael Widenius fondateur de MySQL. Il s'agit de MariaDB, qui est également disponible sur Ubuntu, et choisi par défaut sur Debian.

MySQL est principalement un serveur de bases de données. Pour s'y connecter localement ou à distance, on utilise un client. Il peut s'agir de la commande mysql, ou couramment d'un script PHP. Il faudra dans ce cas installer le module php-mysql qui permet à PHP de communiquer avec un serveur MySQL.

3.1.4.1.1.1. Installation des outils MySQL

3.1.4.1.1.1.1. Installation du serveur MySQL

```
$ sudo apt-get update
$ sudo apt-get remove --purge mysql-server mysql-client mysql-common
$ sudo apt-get autoremove
$ sudo apt-get autoclean
$ sudo apt install mysql-server
$ sudo apt-get update
$ sudo apt-get install mysql-server
```

3.1.4.1.1.1.1.1. Démarrage

```
$ sudo systemctl start mysql
```

3.1.4.1.1.1.1.2. Redémarrage

```
$ sudo systemctl restart mysql
```

3.1.4.1.1.1.1.2.1. Rechargement de la configuration

Pour que MySQL prenne en compte les modifications de sa configuration, commande suivante dans un terminal:

```
$ sudo systemctl reload mysql
```

3.1.4.1.1.1.1.2.2. Forcer la prise en compte de la nouvelle configuration

```
$ sudo systemctl force-reload mysql
```

3.1.4.1.1.1.1.2.3. Connaître la version

```
$ mysql --version
```

Faisons un essai : nous nous connectons en root avec le mot de passe que nous avons renseigné durant l'installation :

```
$ mysql -uroot -p
```

```
chowyounfat@chow-IdeaPad-3-15IGL05:~$ mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 12
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> █
```

3.1.4.1.1.1.2. Création de la base de données avec un utilisateur associé

Cette procédure est la plus courante, c'est celle qu'on réalise lorsqu'on installe une application web en production.

Pour des raisons de sécurité **chaque application doit se connecter avec un utilisateur MySQL qui lui est dédié, et qui n'a accès qu'à la base correspondante.**

```
mysql> create database pizzahutte_admin_db;
mysql> CREATE USER 'pizzahutte-admin'@'localhost' IDENTIFIED WITH
mysql_native_password BY 'pizzahutte';
mysql> GRANT ALL ON pizzahutte_admin_db.* TO 'pizzahutte-admin'@'localhost'
';
mysql> FLUSH PRIVILEGES;
mysql> QUIT;
```

Nous venons ainsi de créer la base de données pizzahutte_admin_db à laquelle l'utilisateur pizzahutte-admin aura accès.

PIZZAHUTTE

Création d'un application Javascript FullStack complète

3.1.4.1.1.3. Sélection de la base de données

mysql -upizzahutte-admin - D pizzahutte_admin_db -p

```
chowyounfat@chow-IdeaPad-3-15IGL05:~$ mysql -u pizzahutte-admin pizzahutte_admin_db -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 20
Server version: 8.0.33 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

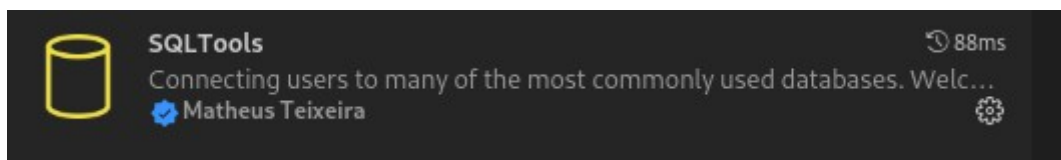
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> █
```

Et voilà, nous avons accès à notre base de données avec l'utilisateur dédié.

3.1.4.1.1.4. Installation du client MySQL : SQLTools

SQLTools est une extension sous VSCode.

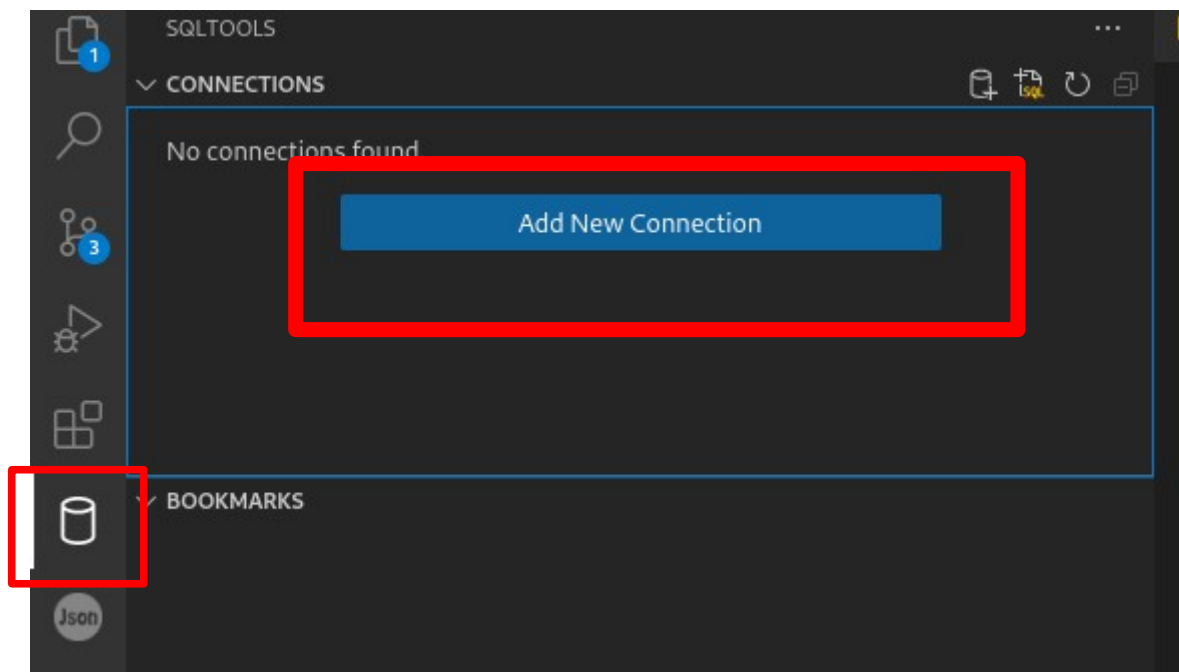


3.1.4.1.1.4.1. Installation du driver MySQL



3.1.4.1.1.5. Connexion à la base de données

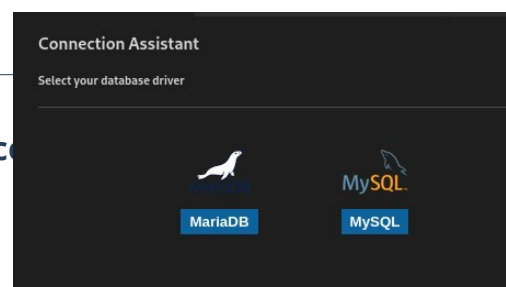
On commence par créer une nouvelle connexion :

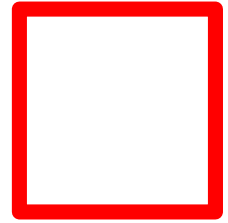



On sélectionne le driver SQL :

PIZZAHUTTE

Création d'une application Javascript FullStack c





 **Connection Assistant**

Connection Settings

Connection name*

pizzahutte_connexion

Connection group

Connect using*

Server and Port

Server Address*

localhost

Port*

3306

Database*

pizzahutte_admin_db

Username*

pizzahutte-admin

Password mode

Ask on connect

MySQL driver specific options

Authentication Protocol

default

Try to switch protocols in case you have problems.

Connection Timeout

Show records default limit

50

SAVE CONNECTION

et on sauvegarde.

Il suffit ensuite de se connecter, en cliquant sur la connexion créée, un point vert indique la connexion OK.

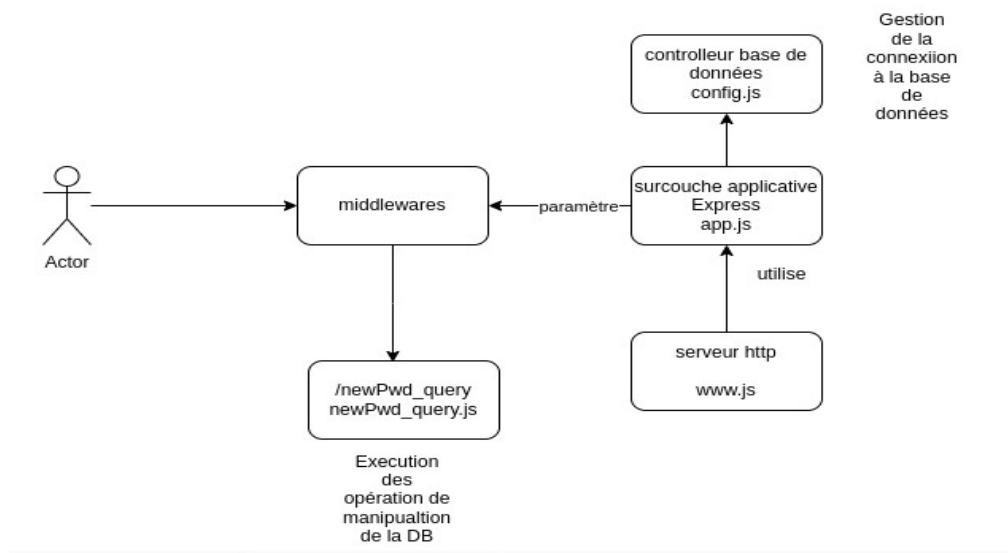
3.1.4.1.1.1.6. Modélisation de la base de données

Nous allons utiliser MySQL Workbench pour modéliser notre base de données. Le logiciel nous permettra ainsi de générer le script pour pouvoir implémenter notre modèle au sein de la base de données que nous avons créées.

3.1.4.1.1.1.7. Accès dynamique à la base de données

3.1.4.1.1.2. Gestion d'une base données

Dans un premier temps, le modèle que nous allons mettre en place au sein de notre application pour pouvoir gérer la base de données est le suivant :



Nous créons un dossier « config » dans lequel nous allons implémenter le fichier config.js, qui va nous permettre de gérer la base de données.

La première des choses à faire est d'insérer un lien vers le module mysql :

```
config.js  
  
const mysql = require("mysql");
```

3.1.4.1.1.2.1. Création de la connexion à la base de données

L'objectif est de pouvoir modéliser la base de données sous MySQL-workbench, d'en générer le script dans un fichier SQL qui serait analysé et exécuté par le module mysql.

Pour pouvoir réaliser ceci, il est capital de mettre le paramètre multipleStatements à true dans la fonction createConnection

```
config.js  
  
const db = mysql.createConnection({  
  host: "localhost",  
  user: "pizzahutte-admin",  
  password: "pizzahutte",  
  database: "pizzahutte_admin_db",  
  multipleStatements: true,  
});  
  
module.exports = db;
```

C'est ce mécanisme qui va permettre de **pouvoir séquencer plusieurs requêtes SQL**.

Dans un premier temps, et dans un but pédagogique, nous allons appeler les requêtes une par une pour construire notre base de données.

3.1.4.1.1.3. Appel de la fonction de connexion depuis notre application app.js

La connexion sera récupérée dans le fichier app.js de notre application à l'aide des instructions suivantes :

```
const db = require('./config/config');  
  
//connect to database once app is started  
db.connect((err) => {  
  if (err) {
```

```

        throw err;
    }
    console.log('connected')
  });

  //make the connection global
  global.db = db;

```

Le code source du module de connexion est le suivant :

config.js

```

const mysql = require("mysql");
const path = require("path");
const fs = require("fs");

// Constants
const MODULE_NAME = "Db-admin";

const FAILURE = "\033[0;31m";
const WARNING = "\033[1;33m";
const SUCCESS = "\033[0;32m";
const INFO = "\033[0;34m";
const ENTITY = "\033[0;35m";
const NO_COLOR = "\033[0m";

const P = NO_COLOR + "[" + WARNING + MODULE_NAME + "-manager" + NO_COLOR + "]";

console.log(`${P}Connexion à la base de données pizzahutte_admin_db`);
var con = mysql.createConnection({
  database: "pizzahutte_admin_db",
  host: "localhost",
  user: "pizzahutte-admin",
  password: "pizzahutte",
});
con.connect((err) => {
  if (err) {
    console.log("Error connecting to Db");
    return;
  }
  console.log("Connection established");
});

const query0 = `DROP TABLE users ;`;
con.query(query0, (err, res) => {
  if (err) throw err;
  console.log("insert OK");
});

const query1 = `CREATE TABLE if not exists users (
  id INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,

```

PIZZAHUTTE

Création d'un application Javascript FullStack complète

```

        email VARCHAR(70) NULL,
        pass VARCHAR(70) NULL,
        nom VARCHAR(70) NULL,
        prenom VARCHAR(70) NULL
    );`;
    con.query(query1, (err, res) => {
        if (err) throw err;
        console.log("insert OK");
    });

    const query2 = `INSERT INTO users(email,pass,nom,prenom)
VALUES
("gino@pizzahutte.com","gino1972","Giacomucci","Gino"),
("alberto@pizzahutte.com","alberto1994","Zacchi","Alberto"),
("bernado@pizzahutte.com","bernado1992","Lapi","Bernado"),
("alba@pizzahutte.com","alba1984","Dello-Iavoco","Alba");`;
    con.query(query2, (err, res) => {
        if (err) throw err;
        console.log("insert OK");
    });

    con.end((err) => {
        // The connection is terminated gracefully
        // Ensures all remaining queries are executed
        // Then sends a quit packet to the MySQL server.
    });

    module.exports = con;

```

3.1.4.1.1.4. Appel des fonctions de gestion des requêtes SQL de l'utilisateur : *newPwd_query.js*

```

var express = require("express");
var router = express.Router();
let bodyParser = require("body-parser");

router.use(express.json()); // for parsing application/json
router.use(express.urlencoded({ extended: true })); // for parsing
application/x-www-form-urlencoded
router.use(bodyParser.urlencoded({ extended: true }));
/* GET users listing. */
router.post("/", function (req, res, next) {
    console.log(req.body.p1);
    res.json(req.body);

    console.log("Génération d'un nouveau mot de passe");
    let newPwd = generatePwd();
});

module.exports = router;

```


USA001 – 4 Codage de la page d'authentification

3.1.4.2. 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) :

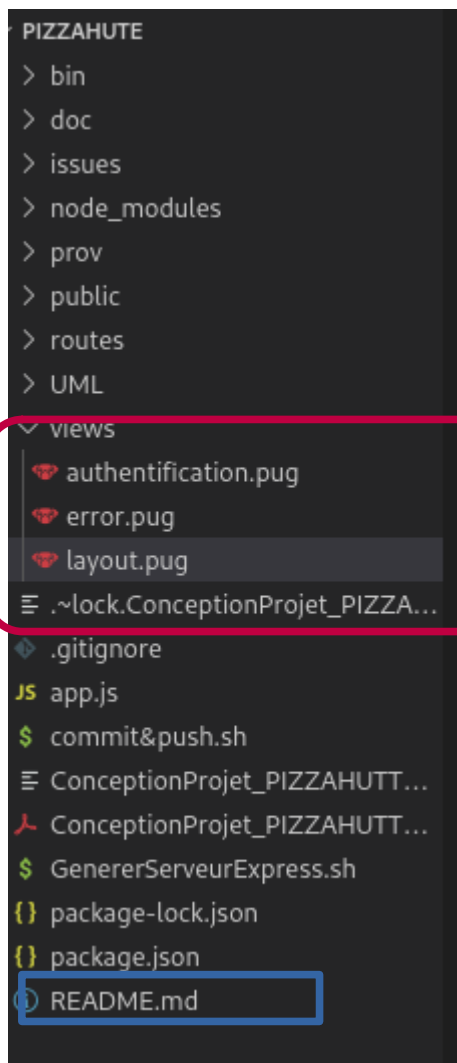
```
// Dependencies  
var express = require("express");  
var router = express.Router();  
  
//GET request  
router.get("/", function (req, res, next) {  
  res.render("index", {  
    title:  
      "PIZZAHUTTE-admin",  
  });  
});  
  
module.exports = router;
```

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.

3.1.4.3. 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.

3.1.4.4. 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.

3.1.4.4.1. 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.

3.1.4.4.2. 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

3.1.4.5. 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'outil 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.

3.1.4.5.1. 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 différent 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-wrap

- 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**
 - **justify-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 :

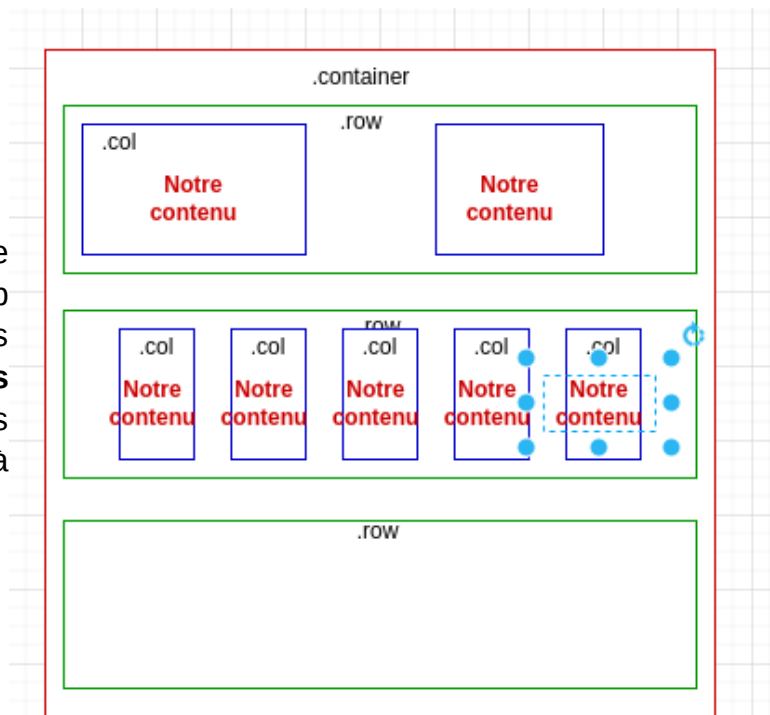
- **col align-self-start**
- **col align-self-center**
- **col align-self-end**

Nota : attention de bien faire attention que *align-self-** est une propriété dédiéees 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 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.



3.1.4.6. *Codage du head commun aux vues (layout .pug)*

3.1.4.6.1. Particularités rencontrées

3.1.4.6.2. 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 posséderons 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.

3.1.4.6.3. 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

```

3.1.4.6.4. 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
          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

```

3.1.4.7. Aperçu côté Front



- 3.1.5. USA001 – 5 Fonctions de récupération de la saisie utilisateur**
- 3.1.6. USA001 – 6 administration de la base de données**
- 3.1.7. USA001 – 7 Codage de la page d'accueil**
- 3.1.8. 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
```

PIZZAHUTTE

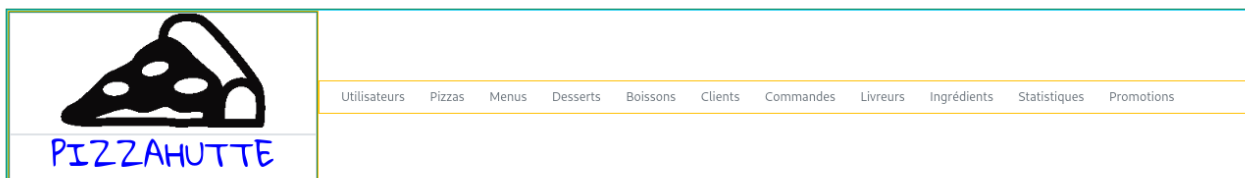
Création d 'un application Javascript FullStack complète

```

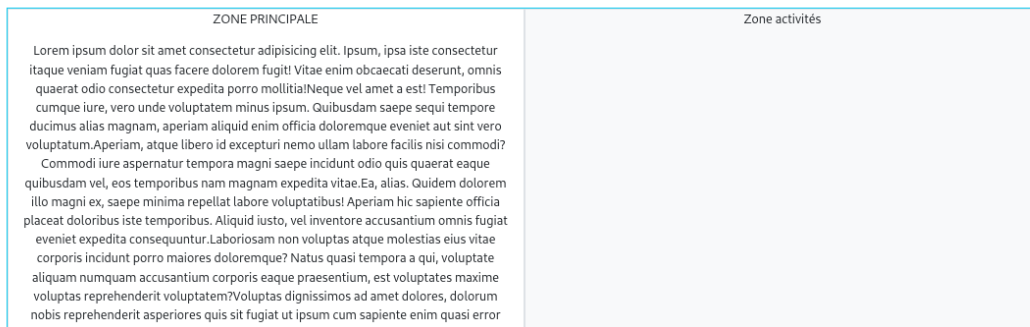
        a.nav-link.disabled Statistiques
      li.nav-item
        a.nav-link.disabled Promotions
    .row.border.border-info
      .col.border
        p ZONE PRINCIPALE
      .col.border.bg-light
        p.bg-light Zone activités
  block mainContent

```

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



ENTREE SUR LA PAGE USERS



4. Annexes

4.1. Les bases de données

Dana Torres, Luis Gonzales, Sara Tassini

4.1.1. Définition

Une base de données est un **ensemble de données mémorisées sur des supports accessibles par un ordinateur** pour satisfaire simultanément plusieurs utilisateurs de façon sélective et en temps très court.

Elles constituent **le cœur du système d'information**.

Il existe **4 types de base de données** :

- **BD Hiérarchiques** : les plus anciennes fondées sur une modélisation arborescente des données.
- **BD Relationnelles** : organisation des données sous forme de tables et exploitation à l'aide d'un langage déclaratif (ex: Oracle, MySQL, Access).
- **BD Déductives** : organisation de données sous forme de table et exploitation à l'aide d'un langage logique.
- **BD Objets** : organisation des données sous forme d'instances de classes hiérarchisées qui possèdent leur propres méthodes d'exploitation.

Dans les lignes qui suivent nous aborderons la conception et la mise en place de bases de données relationnelles.

4.1.2. Conception des bases de données relationnelles

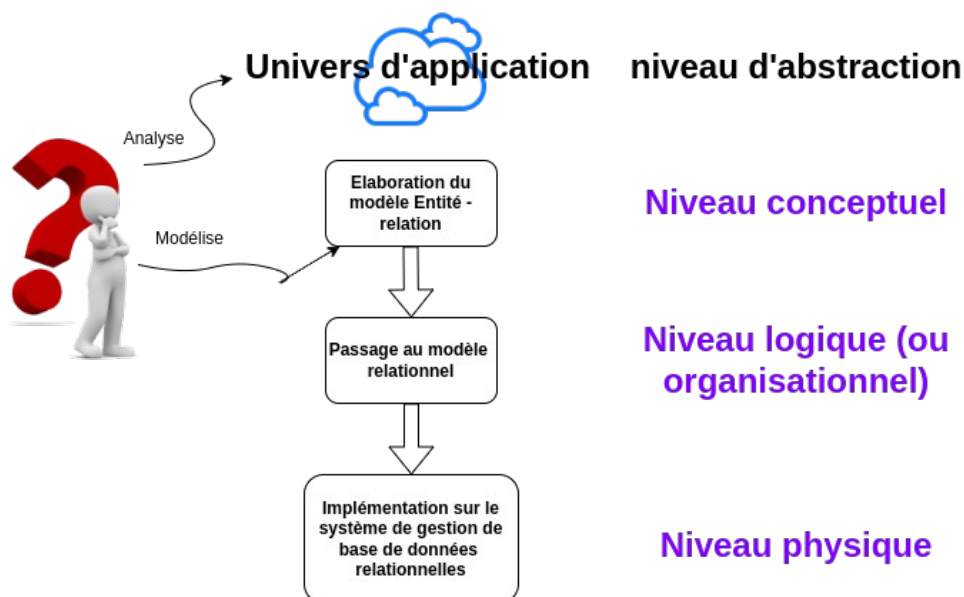
Les bases de données constituent le coeur du système d'information. La conception de ces bases est la tâche *la plus ardue* du processus de développement du système d'information.

Les méthodes de conception préconisent une démarche en *étapes* et font appel à des *modèles* pour représenter les **objets qui composent les systèmes d'information**, les **relations existantes entre ces objets** ainsi que les règles sous-jacentes.

La *modélisation* se réalise en trois étapes qui correspondent à **trois niveaux d'abstraction différents** :

1. **Niveau conceptuel** : représente le *contenu* de la base en termes conceptuels, *indépendamment de toute considération informatique*.
2. **Niveau logique relationnel** : résulte de la traduction du schéma conceptuel en un *schéma propre à une base de données*
3. **Niveau physique** : est utilisé pour *décrire les méthodes d'organisation et d'accès aux bases de données*.

La démarche de conception



4.1.2.1. Modélisation conceptuelle

La **modélisation** est une étape *fondamentale* de la conception de la BD dans la mesure où, d'une part, **on y détermine le contenu de la BD** et, d'autre part, **on y définit la nature des relations entre les concepts principaux**.

Les éléments de base du modèle ER (Entité-Relation) aussi appelé E-A :

- **Les entités**
- **Les attributs**
- **Type de relation : cardinalités**
- **L'identifiant**

4.1.2.1.1. Les entités

Une **entité** se définit comme **un objet pouvant être identifié distinctement**.

Il existe **deux catégories d'entités** :

- **Entités régulières** : son existence *ne dépend pas* de l'existence d'une autre entité.
- **Entités faibles** : son existence *dépend* de l'existence d'une autre entité.

Ex : l'entité CONTRAT est faible car elle n'existe que si l'entité CLIENT correspondante est présente.



4.1.2.1.2. Les attributs : Caractéristiques et propriétés des entités

Un **attribut** peut être **obligatoire** ou **facultatif** et avoir un **domaine de valeurs**.

CLIENT	VEHICULE	CONTRAT	ACCIDENT
Num Client Nom Adresse	Num Véhicule Marque Modèle Année	Num Contrat Type Date	Num Accident Nom Adresse

4.1.2.1.3. Les relations

Contrairement aux entités, les relations n'ont pas de relations propres.

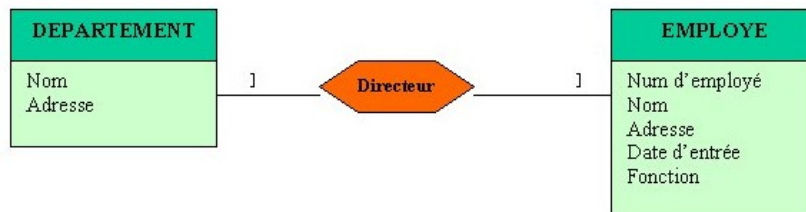
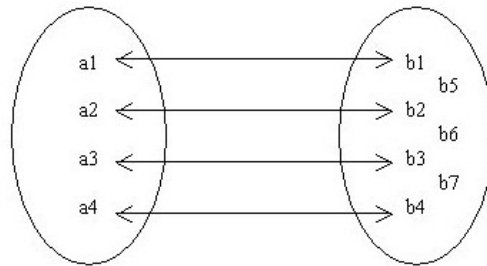
Les relations sont caractérisées, comme les entités , par un **nom** et *éventuellement* des **attributs**.

4.1.2.1.3.1. Cardinalité

La description complète d'une relation nécessite la définition précise de la participation des entités. La cardinalité est le nombre de participations d'une entité à une relation.

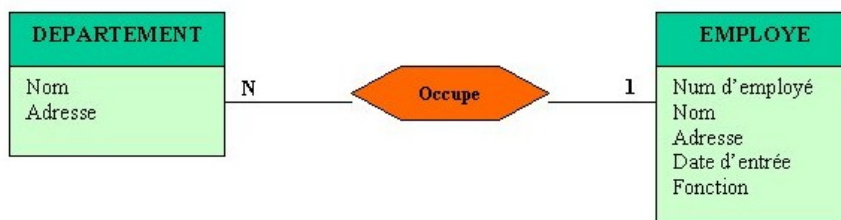
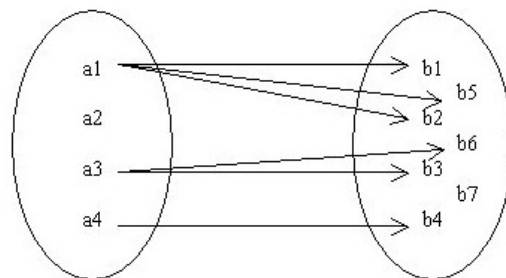
4.1.2.1.3.1.1. Cardinalité un à un

Si et seulement si un employé ne peut être directeur que dans un seul département et un département n'a qu'un seul employé comme directeur.



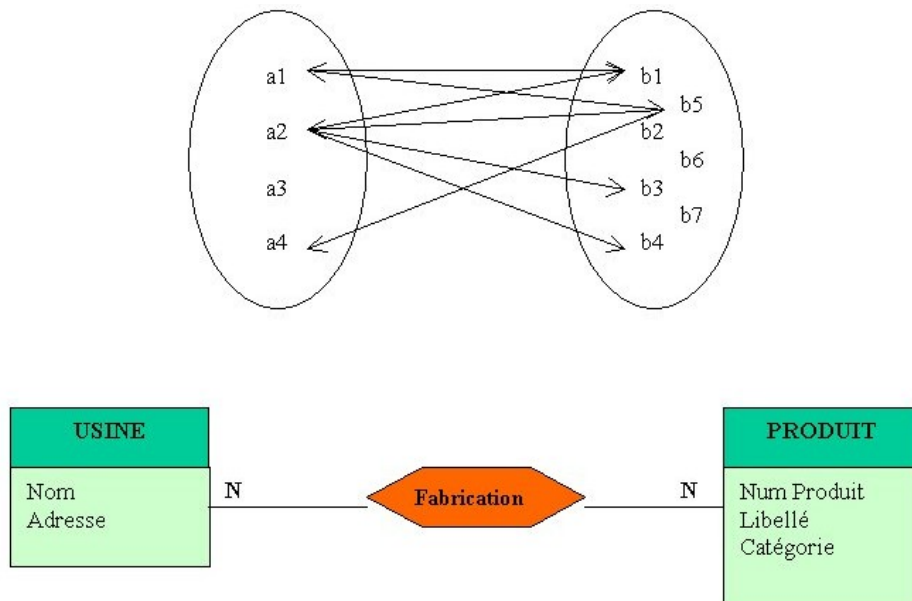
4.1.2.1.3.1.2. Cardinalité un à plusieurs

Un département peut occuper plusieurs employés qui réalisent différentes fonctions mais chaque employé ne fait partie que d'un seul département.



4.1.2.1.3.1.3. Cardinalité plusieurs à plusieurs

Un type de produit peut être fabriqué en plusieurs usines et une usine donnée peut fabriquer plusieurs types de produits.



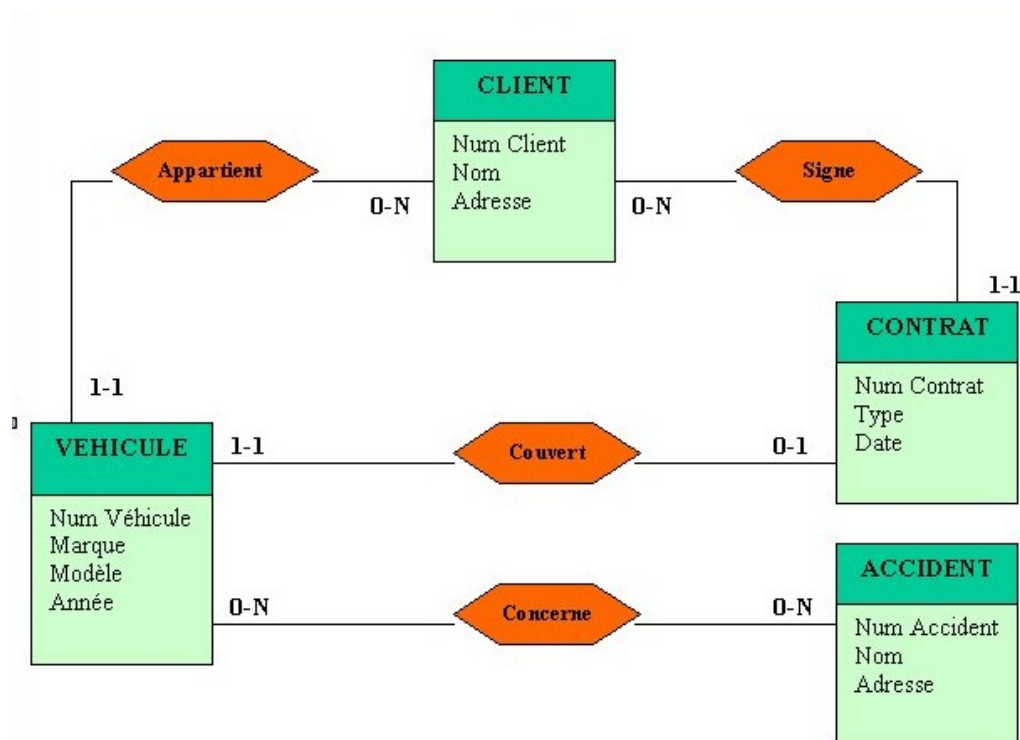
Les cardinalités présentées ci-dessus sont appelées **cardinalités maximales** dans la mesure où **elles représentent le nombre maximum de participations d'une entité à une relation.**

En revanche, la **cardinalité minimale** est le nombre minimal de participations d'une entité à une relation.

La cardinalité **minimale** peut être **0 ou 1.**

Les cardinalités maximales et minimales traduisent **les contraintes propres aux entités et relations**. Dans un schéma conceptuel, elles sont représentées comme suit :

- **0-1** aucune ou une seule
- **1-1** une et une seule
- **0-N** aucune ou plusieurs
- **1-N** une ou plusieurs

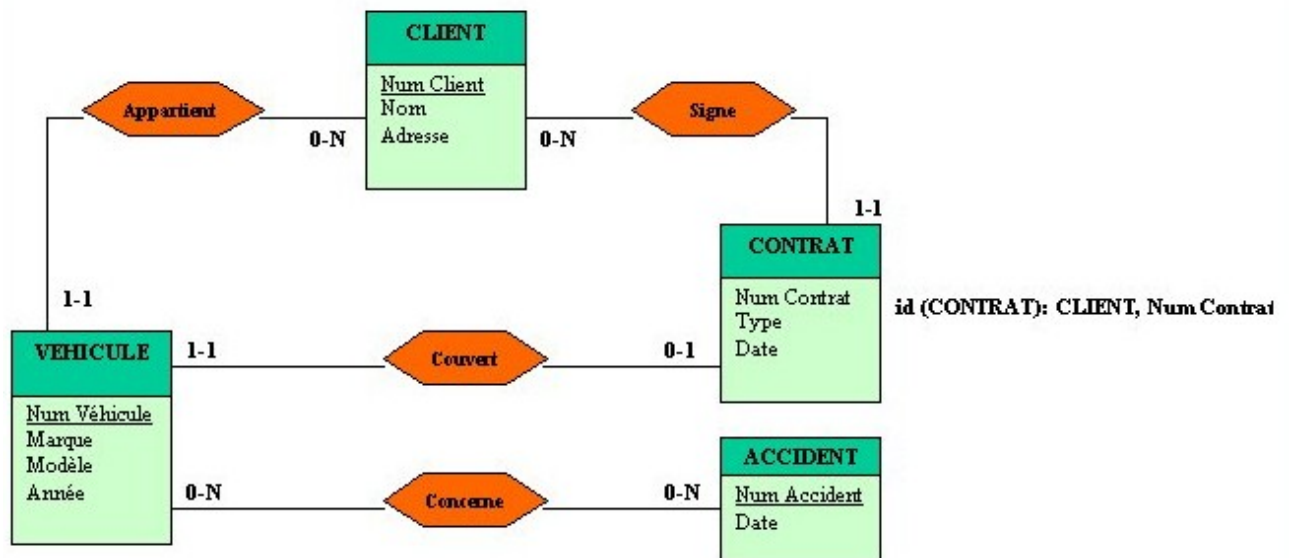


4.1.2.1.4. L'identifiant

Parmi tous les *attributs de l'entité*, **l'identifiant** est un attribut ou un ensemble d'attributs permettant de déterminer **une et une seule entité à l'intérieur de l'ensemble**.

Graphiquement les identifiants sont les **attributs soulignés**.

L'entité faible aura un identifiant composé de **l'identifiant de l'entité dont elle dépend** et d'un autre attribut.



Une situation à modéliser peut avoir plusieurs schémas différents, chaque modèle présentant des avantages et des inconvénients.

Pour mesurer la *qualité* d'une modélisation ER il existe plusieurs critères à utiliser de manière **combinée** :

- **L'expressivité**

Elle traduit la richesse sémantique du schéma et peut être caractérisée par exemple par le nombre de concepts et/ou contraintes exprimés dans le tableau

- **La minimalité**

Elle tend à privilégier les schémas avec un nombre de redondances minimales

- **La lisibilité**

consiste à évaluer la représentation graphique proprement dite.

- **La simplicité**

privilégie les schémas contenant un nombre de concepts minimum. On peut la mesurer par exemple en calculant le nombre d'entités et d'associations présentes sur un schéma.

4.1.2.1.5. Comment construire un schéma conceptuel

La construction d'un schéma conceptuel peut se réaliser de la manière suivante :

1. Déterminer la **liste des entités**.
2. Pour chaque entité :
 1. établir la *liste de ses attributs* ;
 2. parmi ceux-ci, *déterminer un identifiant*.
3. Déterminer les *relations* entre les *entités*.
4. Pour chaque *relation* :
 1. la liste des attributs propres à la relation ;
 2. la *dimension* (binaire, ternaire, etc.) ;
 3. définir les *cardinalités*.
5. Vérifier le **schéma obtenu**, notamment :
 1. *supprimer les transitivités* ;
 2. *s'assurer que le schéma est connexe* ;
 3. *s'assurer qu'il répond aux demandes*.
6. Valider avec les utilisateurs.

4.1.2.2. Modélisation logique relationnelle

Dans le *modèle relationnel*, les entités du schéma conceptuel sont transformées en tableaux à deux dimensions.

Le modèle relationnel s'appuie sur *trois concepts fondamentaux* :

- le domaine
- l'attribut
- la relation : on l'appelle ici la table

4.1.2.2.1. Le domaine

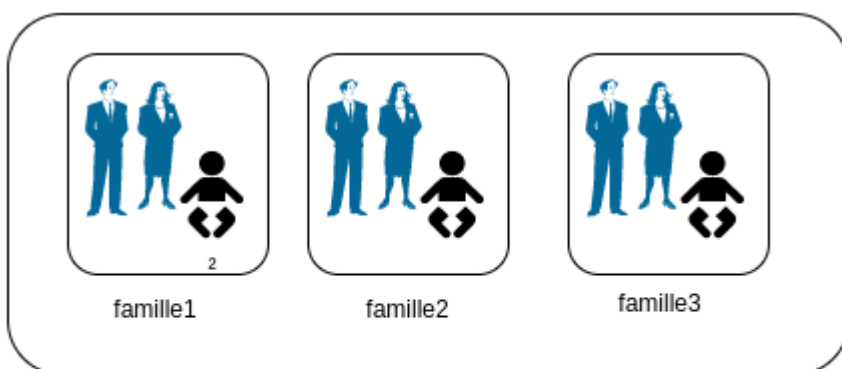
Un domaine est un *ensemble de valeurs*

En logique relationnelle, un domaine peut-être *simple* ou *composé*

- Dans un domaine *simple*, tous les éléments sont **atomiques, indivisibles**



Famille
Domaine
simple



consultations
journalières

Domaine
composé

- Dans un domaine *composé* **chaque entité** peut-être **décomposer**:

4.1.2.2. L'attribut

chaque *colonne* est appelée **attribut** et contient un **ensemble des valeurs d'un domaine**. Chaque ligne représente un **tuple**.

4.1.2.2.3. La relation ou table

Une *relation* est un **tableau à deux dimensions**.

Le *degré* de la relation est le **nombre de colonnes** ou **des domaines considérés**.

SALARIE				⇒	Nom table
Matricule	Nom	Grade	Salaire	⇒	Attributs
100	Müller	cadre	12'000	⇒	Tuples
101	Roche	employé	4'500		
102	Chapuis	assistant	4'000		
				⇒	Domaines

Table SALARIE (Code : entier,
Nom : chaîne de caractères,
Grade : {cadre, employé, assistant},
Salaire : [12'000-4'000])

Degré de la table : 4

4.1.2.2.4. Les contraintes d'intégrité

Elles permettent d'**assurer la cohérence des données**.

Les contraintes d'intégrité peuvent être :

- **Des contraintes de domaine**

restriction de l'ensemble des valeurs possibles d'un attribut.

- **Contrainte de clé**

définit un sous-ensemble minimal des colonnes tel que la table ne puisse contenir deux lignes ayant mêmes valeurs pour ces colonnes.

Il existe trois types de clé :

- **Clé primaire** : Ensemble minimum d'attributs qui permet de distinguer chaque n-uplet de la table par rapport à tous les autres.

Chaque table doit avoir une clé primaire.

- **Clé candidate** : Ensemble minimum d'attributs *susceptibles de jouer le rôle de la clé primaire*
- **Clé étrangère** : fait référence à *la clé primaire d'une autre table*.

- **Contrainte obligatoire**

précise qu'un attribut ou plusieurs attributs **doivent toujours avoir une valeur**.

- **Contrainte d'intégrité référentielle ou d'inclusion:**

lie deux colonnes ou deux ensembles de colonnes de deux tables différentes.

4.1.2.2.5. La théorie de la normalisation

La théorie de la normalisation permet de **définir formellement** la **qualité** des **tables** au regard du problème posé *par la redondance des données*.

La théorie de la normalisation s'appuie sur la **dépendance fonctionnelle**.

Codd a défini un *ensemble de formes normales* caractérisant les tables relationnelles :

- **Première forme normale**

La table ne contient que des **attributs atomiques**.

- **Deuxième forme normale**

- La table ne contient que des **attributs atomiques**
- il n'existe pas de dépendance fonctionnelle entre une partie d'une clé et une colonne non clé de la table.

- **Troisième forme normale**

- La table ne contient que des attributs atomiques
- Il n'existe pas de dépendance fonctionnelle entre une partie d'une clé et une colonne non clé de la table
- Aucune dépendance fonctionnelle entre les colonnes non clé.

Ainsi, **plus une table est normalisée** *moins elle comporte de redondances* et donc de **risques d'incohérence sémantiques dans les schémas relationnels**.

4.1.2.2.5.1. Règles à suivre pour concevoir un schéma relationnel :

Les règles principales de transformation d'un schéma conceptuel Entité-Relation en un schéma relationnel sont :

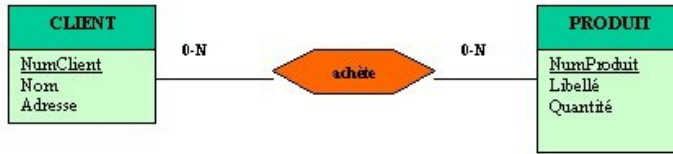
4.1.2.2.5.1.1. Règle 1 : Toute entité est traduite en une table relationnelle dont les caractéristiques sont les suivantes :

- **le nom de la table** est le nom de **l'entité** ;
- la **clé** de la **table** est **l'identifiant de l'entité** ;
- *les autres attributs de la table forment les autres colonnes de la table.*

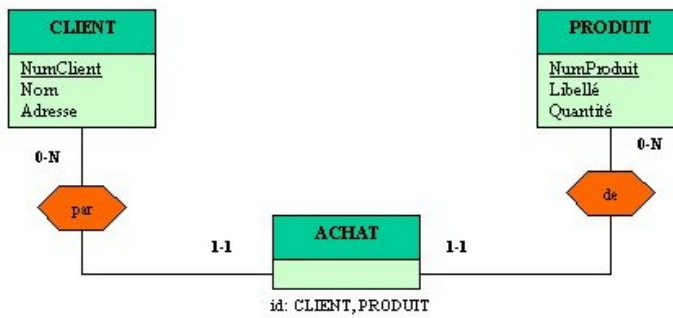
4.1.2.2.5.1.2. Règle 2 : Toute relation binaire plusieurs à plusieurs est traduite en une table relationnelle dont les caractéristiques sont les suivantes :

- Le **nom de la table** est le **nom de la relation**
- la **clé de la table** est formée par la **concaténation des identifiants des entités participant à la relation**
- les **attributs spécifiques de la relation** forment **les autres colonnes de la table**.
- Une **contrainte d'intégrité référentielle** est générée entre **chaque colonne clé de la nouvelle table** et la **table d'origine de cette clé**.

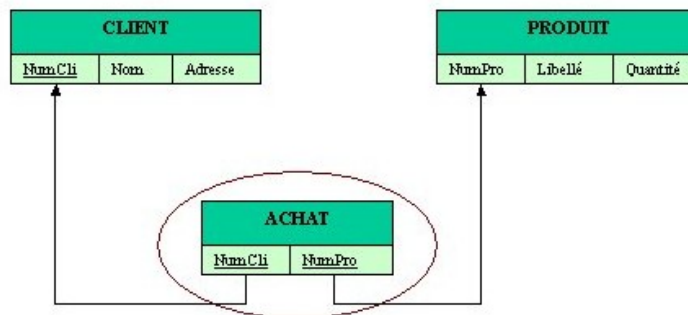
Schéma conceptuel :



Transformation du schéma conceptuel :



Traduction:



4.1.2.2.5.1.3. Règle III : Toute relation binaire un à plusieurs est traduite :

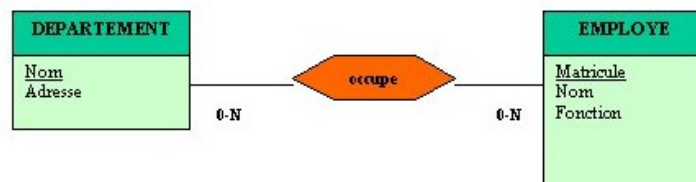
- soit par un **report de clé** : l'identifiant de l'entité participant à la relation côté *N* est **ajoutée comme colonne supplémentaire à la table représentant l'autre entité**.

Cette colonne est parfois appelée **clé étrangère**.

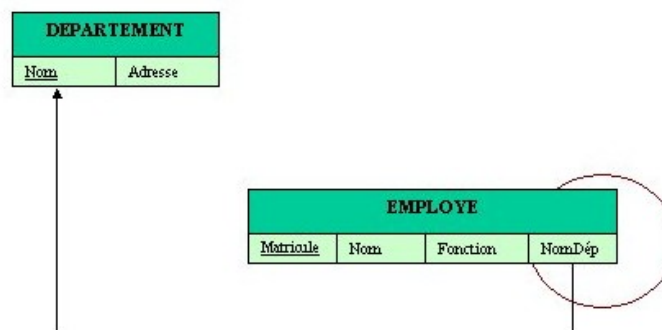
Le cas échéant, les attributs spécifiques à la relation sont eux aussi ajoutés à la même table ;

- soit par **une table spécifique** dont les *caractéristiques* sont les suivantes :
 - le **nom de la table** est le **nom de la relation**
 - la **clé de la table** est l'identifiant de l'entité participant à la relation côté **1**
 - les **attributs spécifiques de la relation** forment les autres colonnes de la table.

Schéma conceptuel :



Traduction :

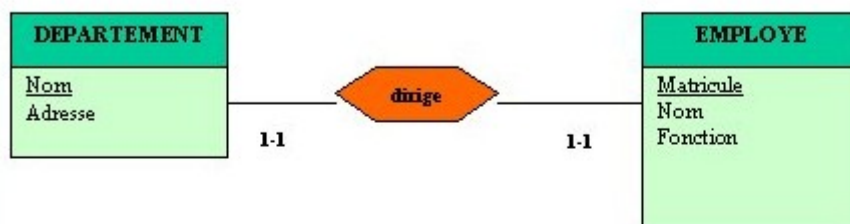


4.1.2.2.5.1.4. Règle 4 : Toute relation binaire un à un est traduite, au choix, par l'une des trois solutions suivantes :

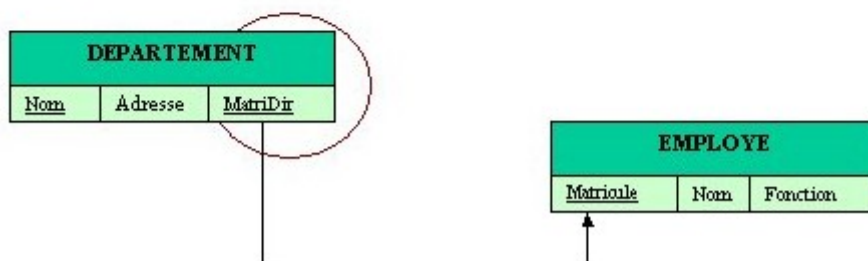
- **Choix1** : fusion des tables des entités qu'elle relie
- **Choix2** : report de clé d'une table dans l'autre
- **Choix3** : création d'une table spécifique reliant les clés des deux entités

Les *attributs spécifiques de cette relation* sont ajoutés à la table résultant de la fusion (choix1), reportés avec la clé (choix2), ou insérés dans la table spécifique (choix3).

Schéma conceptuel :

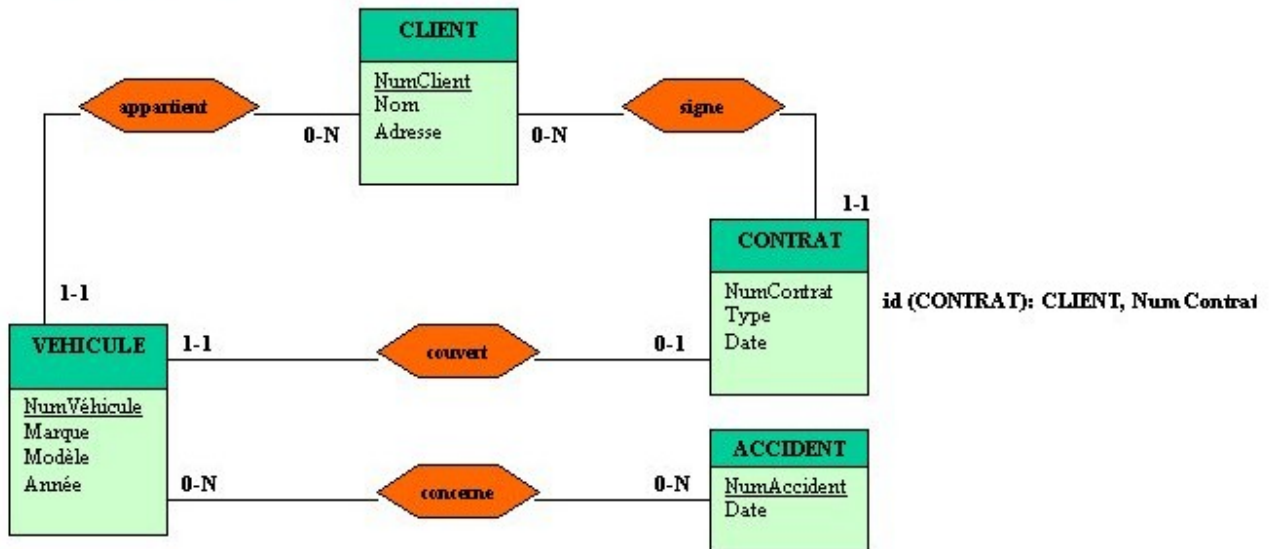


Traduction (choix2) :

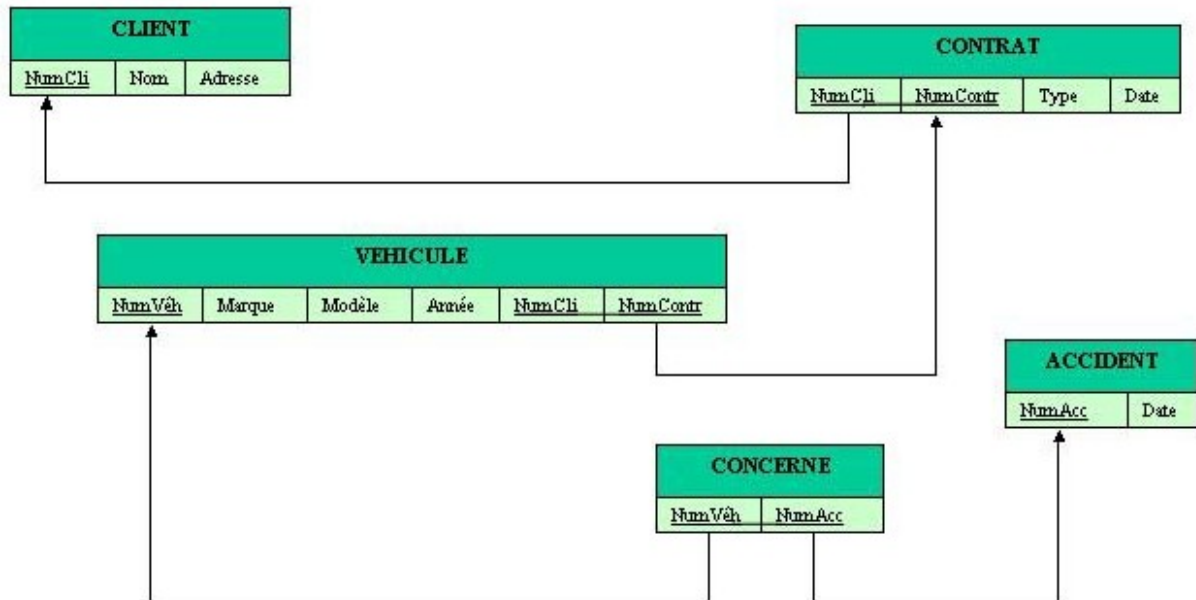


4.1.2.2.6. Exemple de traduction d'un schéma conceptuel en schéma relationnel

Schéma conceptuel :



Traduction :



4.1.2.3. *Le langage SQL*

Le langage **SQL (Structured Query Langage)** s'appuie sur les **opérateurs de l'algèbre relationnelle** définis en 1970 par **Codd**, mathématicien, chercheur chez IBM.

Le langage SQL est basé sur le concept de relation de la théorie des ensembles.

4.1.2.3.1. Les opérateurs de l'algèbre relationnelle

Les opérations de bases sont :

- La projection
- La sélection
- La jointure

les opérations ensemblistes sont :

- L'union
- L'intersection
- la différence
- le produit cartésien

4.1.2.3.1.1. Les opérateurs de base

4.1.2.3.1.1.1. Projection

Cet opérateur ne porte que **sur une relation**.

Il permet de ne retenir que *certaines attributs spécifiés d'une relation*.

On obtient tous les n-uplets de la relation à l'exception des doublons.

--	--	--	--	--	--

Forme de l'opération de projection :

SELECT DISTINCT liste d'attributs FROM table ;

SELECT liste d'attributs FROM table ;

Exemples :

SELECT DISTINCT Espèce FROM Champignons ;

SELECT DISTINCT Espèce, Catégorie FROM Champignons ;

Nota : La clause DISTINCT permet d'éliminer les doublons.

4.1.2.3.1.1.2. Sélection

Cet opérateur porte sur **1 relation**.

Il permet de ne retenir **que les n-uplets répondant à une condition exprimée à l'aide des opérateurs arithmétiques** (=, >, <, >=, <=, <>) ou **logiques de base (ET, OU, NON)**.

Tous les attributs de la relation sont conservés.

Un attribut peut ne pas avoir été renseigné pour certains n-uplets. Si une condition de sélection doit en tenir compte, on indiquera simplement : nom_attribut "non renseigné".

Forme de l'opération de sélection :

SELECT * FROM table WHERE condition ;

Exemple :

SELECT * FROM Champignons WHERE Catégorie="Sec" ;

La *condition de sélection* exprimée derrière la *clause WHERE* peut être spécifiée à l'aide :

- **des opérateurs de comparaison** : =, >, <, <=, >=, <>
- **des opérateurs logiques** : AND, OR, NOT
- **des opérateurs** : IN, BETWEEN, LIKE, IS

Autres exemples :

Soit la table ETUDIANT (N°Etudiant, Nom, Age, CodePostal, Ville) :

```
SELECT *  
FROM ETUDIANT  
WHERE Age IN (19, 20, 21, 22, 23) ;
```

```
SELECT *  
FROM ETUDIANT  
WHERE Age BETWEEN 19 AND 23 ;
```

```
SELECT *  
FROM ETUDIANT  
WHERE CodePostal LIKE '42%' ;
```

```
SELECT *  
FROM ETUDIANT  
WHERE CodePostal LIKE '42____' ;
```

```
SELECT *  
FROM ETUDIANT  
WHERE Ville IS NULL ; // Etudiants pour lesquels la ville n'est pas renseignée
```

```
SELECT *  
FROM ETUDIANT  
WHERE Ville IS NOT NULL ; // Etudiants pour lesquels la ville est renseignée
```

4.1.2.3.1.1.3. Jointure (R1,R2, condition d'égalité entre attributs)

Cet opérateur porte sur **2 relations qui doivent avoir au moins un attribut défini dans le même domaine** (ensemble des valeurs permises pour un attribut).

La condition de jointure peut porter **sur l'égalité d'un ou de plusieurs attributs définis dans le même domaine** (mais n'ayant pas forcément le même nom).

Les n-uplets de la relation résultat sont formés par la concaténation des n-uplets des relations d'origine qui vérifient la condition de jointure.

Remarque : Des jointures plus complexes que l'équijointure peuvent être réalisées en généralisant l'usage de la condition de jointure à d'autres critères de comparaison que l'égalité (<,>, <=,>=, <>).

Forme de l'opération JOINTURE (équijointure) :

En SQL, **il est possible d'enchaîner plusieurs jointures dans la même instruction SELECT.**

En SQL de base :

```
SELECT * FROM table1, table2, table3, ...  
WHERE table1.attribut1=table2.attribut1 AND table2.attribut2=table3.attribut2  
AND ...;
```

Exemple :

```
SELECT * FROM Produit, Détail_Commande  
WHERE Produit.CodePrd=Détail_Commande.CodePrd ;
```

ou en utilisant des alias pour les noms des tables :

```
SELECT * FROM Produit A, Détail_Commande B  
WHERE A.CodePrd=B.CodePrd ;
```

4.1.2.3.1.2. Les opérateurs ensemblistes

4.1.2.3.1.2.1. UNION(R1,R2)

Cet opérateur porte sur **2 relations qui doivent avoir le même nombre d'attributs définis dans le même domaine** (ensemble des valeurs permises pour un attribut).

On parle de relations *ayant le même schéma*.

La **relation résultat** possède les **attributs des relations d'origine et les n-uplets de chacune, avec élimination des doublons éventuels**.

Forme de l'opération UNION :

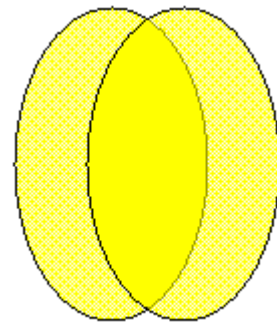
SELECT liste d'attributs FROM table1

UNION

SELECT liste d'attributs FROM table 2 ;

Exemple :

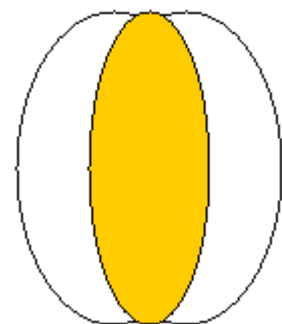
```
SELECT n°enseignant, NomEnseignant FROM E1
UNION
SELECT n°enseignant, NomEnseignant FROM E2 ;
```



4.1.2.3.1.2.2. INTERSECTION(R1,R2)

Cet opérateur porte sur **deux relations de même schéma**.

La *relation résultat* possède **les attributs des relations d'origine et les n-uplets communs à chacune**.



Par exemple :

E1 : Enseignants élus au CA

E2 : Enseignants représentants syndicaux

Si on désire connaître les enseignants du CA qui sont des représentants syndicaux :

$R2 = \text{INTERSECTION}(E1, E2)$

Forme de l'opération INTERSECTION :

En SQL de base :

```
SELECT attribut1, attribut2, ... FROM table1
WHERE attribut1 IN (SELECT attribut1 FROM table2) ;
```

ou avec SQL2 :

```
SELECT attribut1, attribut2, ... FROM table1
INTERSECT
SELECT attribut1, attribut2, ... FROM table2 ;
```

Exemple :

```
SELECT n°enseignant, NomEnseignant FROM E1
WHERE n°enseignant IN (SELECT n°enseignant FROM E2) ;
```

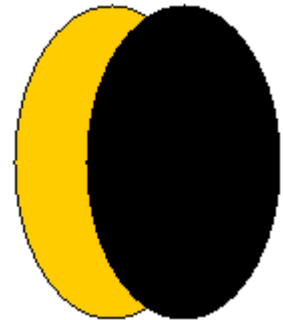
OU

```
SELECT n°enseignant, NomEnseignant FROM E1
INTERSECT
SELECT n°enseignant, NomEnseignant FROM E2 ;
```

4.1.2.3.1.2.3. DIFFERENCE (R1, R2)

Cet opérateur porte sur **deux relations de même schéma**.

La relation résultat possède **les attributs des relations d'origine et les n-uplets de la première relation qui n'appartiennent pas à la deuxième**.



Attention ! DIFFERENCE (R1, R2) ne donne pas le même résultat que DIFFERENCE (R2, R1)

Si par exemple on désire obtenir la liste des enseignants du CA qui ne sont pas des représentants syndicaux : $R3 = \text{DIFFERENCE}(E1, E2)$

Forme de l'opération DIFFERENCE :

En SQL de base :

```
SELECT attribut1, attribut2, ... FROM table1
WHERE attribut1 NOT IN (SELECT attribut1 FROM table2) ;
```

```
SELECT table1.attribut1, table1.attribut2, ...
FROM table1 LEFT OUTER JOIN table2 ON table1.attribut1 = table2.attribut1
WHERE table2.attribut1 IS NULL ;
```

Exemple :

```
SELECT n°enseignant, NomEnseignant FROM E1
WHERE n°enseignant NOT IN (SELECT n°enseignant FROM E2) ;
```

OU

```
SELECT n°enseignant, NomEnseignant FROM E1
EXCEPT
SELECT n°enseignant, NomEnseignant FROM E2 ;
```

ou encore

PIZZAHUTTE

Création d 'un application Javascript FullStack complète

```
SELECT E1.n°enseignant, E1.NomEnseignant
FROM E1 LEFT OUTER JOIN E2 ON E1.n°enseignant = E2.n°enseignant
WHERE E2.n°enseignant IS NULL ;
```

4.1.2.3.1.2.4. PRODUIT (R1, R2)

Cet opérateur porte **sur deux relations**.

La relation *résultat* possède **les attributs de chacune des relations d'origine et ses n-uplets sont formés par la concaténation de chaque n-uplet de la première relation avec l'ensemble des n-uplets de la deuxième**.

Exemple :

Etudiants

Epreuves

Examen = PRODUIT (Etudiants, Epreuves)

Forme de l'opération produit cartésien :

SELECT * FROM table1, table2 ;

Exemple :

```
SELECT * FROM Etudiants, Epreuves ;
```

4.1.2.3.2. Commandes SQL

4.1.2.3.2.1. Création, modification de table

La création d'une table peut être réalisée au moyen de la commande suivante :

CREATE TABLE nom_table

Le langage SQL dispose également des commandes suivantes :

- **ALTER TABLE**
permet :
 - **modification de la structure d'une table**
 - **ajout d'une colonne**
 - **ajout d'une contrainte**
 - **modification de la définition d'une colonne**
 - **suppression d'une contrainte, etc.**
 -
- **DROP TABLE** : permet de supprimer une table.
- **RENAME** : permet de modifier le nom d'une table.

4.1.2.3.2.2. Interrogation des données

Le langage de manipulation des données comprend quatre instructions principales :

- **SELECT** : pour l'interrogation d'une ou plusieurs tables
- **INSERT** : pour l'ajout des lignes dans une table
- **UPDATE** : pour la modification des lignes
- **DELETE** : pour la suppression des lignes

Les **contraintes de colonnes** et les **contraintes de table** matérialisent les différentes contraintes d'intégrité dont les SBD prend en charge la vérification systématique :

- **La contrainte d'unicité : UNIQUE**

Elle permet *d'assurer qu'il n'existe pas de valeur dupliquée dans la colonne*

- **La contrainte d'obligation : NOT NULL**

Elle *autorise et gère* une valeur particulière appelée **la valeur nulle**.

Cette valeur nulle traduit à la fois la valeur *manquante* et la valeur *inexistante* (ex. le nom de jeune fille pour un homme). NOT NULL **est toujours une contrainte colonne, et ne peut pas être une contrainte de table.**

- **La contrainte de clé primaire : PRIMARY KEY**

Elle permet de **choisir une colonne (ou un groupe de colonnes) unique privilégiée dans une table ;**

- **La contrainte d'intégrité référentielle**

Elle admet deux syntaxes selon qu'elle porte soit

- sur une colonne (contrainte de colonne)

On utilise alors la clause **REFERENCES**

- sur plusieurs colonnes (contrainte de table).

- On utilise l'expression **FOREIGN KEY**.

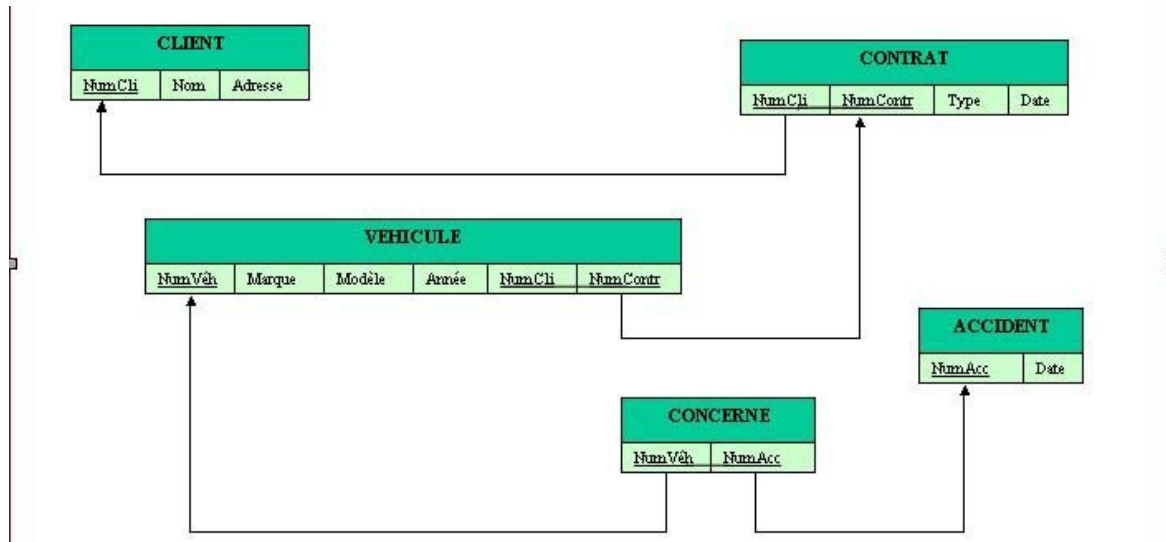
La clé étrangère fait référence à la clé primaire d'une autre table.

Elle traduit *un lien sémantique* avec une autre table.

- **La contrainte sémantique : CHECK**

Elle permet de **spécifier les conditions logiques portant sur une ou plusieurs colonnes d'une même table.**

4.1.2.3.3. Exemple de traduction d'un schéma relationnel en langage SQL :



```

create table CLIENT (
    NumCli char (12) not null,
    Nom char (38) not null,
    Adresse char (60) not null,
    primary key (NumCli)
);
create table CONTRAT (
    NumCli char (12) not null,
    NumContr char (38) not null,
    Type decimal (4) not null,
    Date date not null,
    primary key (NumCli, NumCtr),
    foreign key (NumCli) references CLIENT
);
create table VEHICULE (
    NumVeh char (16) not null,
    Marque char (30) not null,
    Modele decimal (30) not null,
    NumCli char (12) not null,
    NumContr decimal (8) not null,
    primary key (NumVeh),
    unique (Numcli, NumContr),
    foreign key (NumCli) references CONTRAT
);
create table ACCIDENT (
    NumAcc char (10) not null,
    Date date not null,
    primary key (NumAcc)
);
create table CONCERN (
    NumVeh char (16) not null,
    NumAcc char (10) not null,
    primary key (NumVeh, NumAcc),
    foreign key (NumVeh) references CLIENT foreign key (NumAcc) references CLIENT
);

```


4.1.3. Conception de la base de données avec MySQLwork-bench