Mescourses : un gestionnaire de listes de courses

Travail de Bachelor

Octave Plancherel

Juillet 2022

Supervisé par :

Prof. Dr. Jacques Pasquier-Rocha
Software Engineering Group

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Groupe Génie Logiciel
Département d'Informatique
Université de Fribourg (Suisse)

Software Engineering Group

Remerciements

Je remercie le Prof. Dr. Jacques Pasquier-Rocha de m'avoir donné l'opportunité de réaliser ce présent travail. Merci également pour son accompagnement et ses conseils. Je tiens également à remercier mes parents pour leur grand soutien moral.

Table des matières

Liste des figures

Liste des tableaux

Liste des codes source

1
Introduction

1.1. Motivation et objectifs

Tout le monde doit faire des courses pour vivre. Se rendre au supermarché sans se préparer, c'est prendre le risque d'oublier d'acheter certains produits ou au contraire de se retrouver avec des articles à double chez soi : donc une perte de temps et d'argent. C'est pour éviter ces deux désagréments principaux qu'il est utile de faire une liste de courses. De plus, cette dernière a l'avantage de réduire les achats impulsifs dûs à des promotions ou à un ventre vide. Enfin, lorsqu'on prépare sa liste de courses, on ouvre le frigo et on fait l'inventaire de ce qui nous reste, ce qui réduit le gaspillage. Avant les smartphones, les gens notaient ce qu'il leur fallait sur un bout de papier et traçaient le produit à chaque fois qu'ils l'ajoutaient au caddie. Cependant, il y avait des inconvénients : on pouvait la perdre, la déchirer voire l'oublier. Le but de ce travail est donc de créer une application web de gestion de listes de courses qui reprend le concept de la liste de courses en papier en supprimant ses désagréments et en ajoutant des fonctionnalités utiles.

1.2. Structure du rapport

Chapitre 1 : Introduction

L'introduction présente les motivations et les objectifs de ce travail ainsi que la structure du rapport qui est un bref résumé de chaque chapitre présenté.

Chapitre 2 : fonctionnalités et modélisation

Dans ce chapitre, on va découvrir quelles fonctionnalités utiles on va mettre en oeuvre dans notre application. Pour ce faire, deux applications concurrentes de listes de courses seront analysées et une approche plus formelle avec des personas, scénarios et user stories sera utilisée. Ensuite, ces fonctionnalités seront modélisées dans un diagramme de cas d'utilisation et le stockage des données sera montré dans un diagramme entité-association.

Chapitre 3 : Développement

Dans ce chapitre, les technologies qui seront utilisées pour le développement de l'application sont expliquées. De plus, une explication concrète de comment Mescourses a été implémentée est donnée.

Chapitre 4 : Utilisation de l'application

Ce chapitre reprend un scénario élaboré dans le chapitre 2 et l'applique concrètement dans l'application Mescourses.

Conclusion

La conclusion présente le bilan de ce travail ainsi que plusieurs améliorations possibles de l'application.

2

Fonctionnalités et modélisation

2.1. Fonctionnalités et modélisation

Comme énoncé dans la motivation, le but de ce travail est de créer une application web de gestion de listes de courses. Une application web ne demande pas d'installation sur la machine du client. Elle est normalement installée sur un serveur et est directement utilisable depuis un navigateur internet d'un client[36]. Une première étape lors de la conception d'une application est de découvrir quelles fonctionnalités celle-ci va offrir aux utilisateurs, c'est-à-dire quelles possibilités aura un utilisateur de l'application. Le mieux reste de trouver le bon équilibre entre trop peu de fonctionnalités (application inutile) et trop de fonctionnalités (application trop compliquée, ralentissements). Le but de cette partie est de trouver quelles fonctionnalités utiles seront proposées dans l'application. Une fois sélectionnées, elles seront modélisées à l'aide d'un diagramme de cas d'utilisation. La modélisation est une étape importante car elle facilite la communication avec le client (dans le cadre de ce travail le professeur) et l'équipe de développeurs.

Pour commencer à découvrir des fonctionnalités utiles, on va analyser les applications développées par la concurrence et voir quelles avantages et inconvénients elles offrent pour ensuite intégrer leurs points positifs et/ou améliorer leurs points négatifs pour notre application. Deux applications de listes de courses populaires vont être prospectées :

Bring ! et Listonic. Il s'agit d'applications très téléchargées sur le Google Play Store au 4 juillet 2022. Elles sont également bien notées par un grand nombre d'utilisateurs.

Ensuite, on utilisera une approche plus formelle avec des personas, des scénarios et des user stories afin d'éventuellement découvrir de nouvelles fonctionnalités qui ne seraient pas présentes dans les deux applications analysées.

2.2. Applications existantes

2.2.1. Bring !

Bring !, de son nom complet « Bring ! Shopping List » est une application développée par l'entreprise Suisse Labs AG qui a pour but de créer et de partager des listes de courses avec de la famille ou des amis. L'application compte plus de 10 millions d'utilisateurs dans le monde et est notée 4.5/5 sur par plus de 118'000 personnes sur le Google Play Store[7][8][16].

Bring ! encourage ses utilisateurs à se connecter. Les utilisateurs ont le choix de le faire en utilisant Google, Facebook ou un courriel. Il y a également la possibilité d'utiliser l'application sans se connecter, mais dans ce cas, on perd certaines fonctionnalités telles que partager sa liste de courses avec des amis ou encore accéder à partir d'autres appareils (cela est normal dès lors que la liste de courses n'est pas enregistrée dans la base de données). On perd également d'autres possibilités comme recevoir des notifications push ou créer plusieurs listes de courses, ce qui pourrait être un choix délibéré des concepteurs pour inciter l'utilisateur à se connecter.

L'application possède quatre pages principales. La page d'accueil (voir figure 2.1) -qui est celle sur laquelle on est redirigé après un login- est une liste de courses par défaut appelée « Maison ». Quatre manières différentes sont proposées pour peupler la liste. La première façon de faire est d'ajouter des produits via les promotions disponibles dans les magasins « Trouver des offres adaptées ». La deuxième est de sélectionner le produit via la rubrique « aliments de saison » qui regroupe plusieurs denrées actuelles. Le troisième moyen est de sélectionner le produit qui nous intéresse via diverses catégories. Il est intéressant de voir que les catégories ne peuvent, ni être crées, ni modifiées, ni supprimées. La dernière manière d'ajouter un produit est de taper son nom manuellement dans le champ prévu à cet effet. Une bonne fonctionnalité, lors de la sélection d'un produit avant l'ajout dans la liste, est la possibilité de préciser la quantité et le magasin ainsi que d'ajouter une photo ou une description. Il est également possible de trier les produits par catégories. La liste peut aussi être partagée avec des amis qui peuvent ajouter ou retirer des produits de la liste à tout moment.

FIGURE 2.1. – Page d'accueil de Bring !

La deuxième page (voir figure 2.2) contient des suggestions de recettes. Sur ce volet de l'application, l'utilisateur peut s'inspirer culinairement et si une recette l'intéresse, il peut en un clic ajouter les ingrédients qui lui manquent. Ceux-ci viendront directement s'ajouter dans sa liste de courses.

FIGURE 2.2. – Page inspiration de Bring !

La troisième page (voir figure 2.3) montre une version numérique des magazines promotionnels de différents commerces suisses. Cette page a pour but d'informer sur les différentes offres hebdomadaires des enseignes suisses. A noter que Bring ! est en partenariat avec la Migros[9]. Une bonne fonctionnalité est proposée avec le géant orange : l'ajout d'un article en promotion en un clic dans la liste. C'est très probablement en raison de cette collaboration que les promotions d'Aldi, Coop ou Lidl n'y sont jamais référencées.

FIGURE 2.3. – Page offre de Bring !

La quatrième page (voir figure 2.4) contient les paramètres de l'application. Elle propose aussi de configurer l'assistant Google afin de pouvoir ajouter des produits dans la liste avec la voix de manière aussi facile que : « OK Google, ajoute un steak dans ma liste maison ».

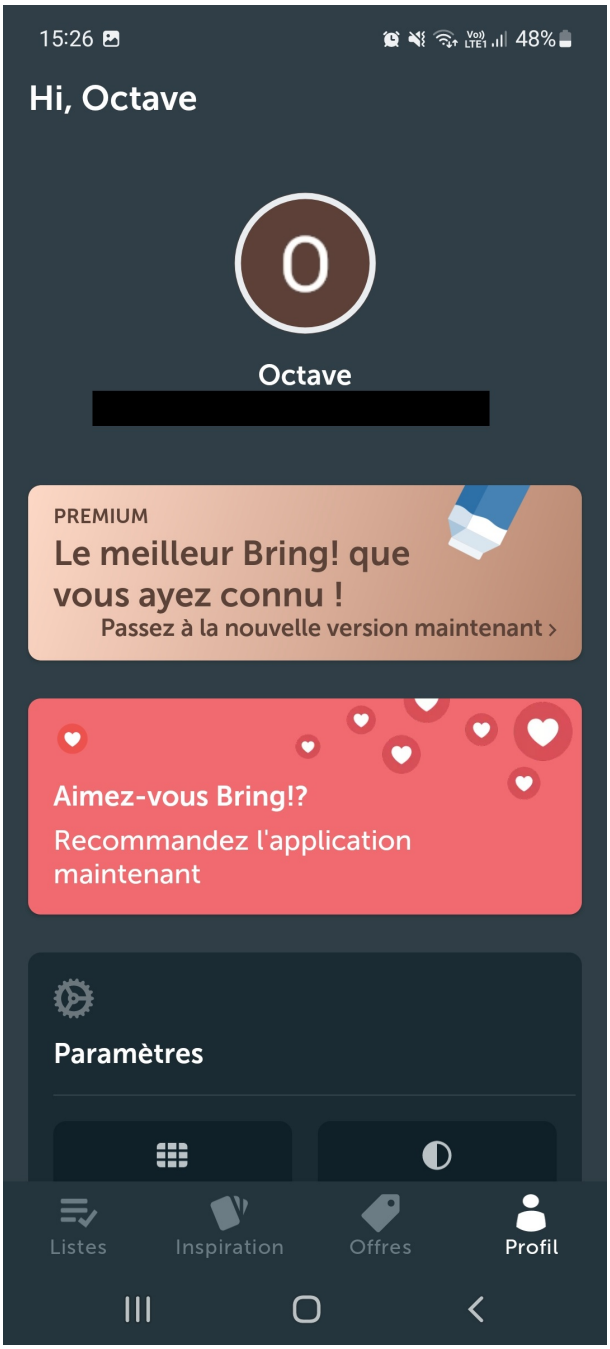

FIGURE 2.4. – Page profil de Bring !

2.2.2. Listonic

Listonic est une application développée en 2010 qui permet de créer des listes de courses simplement et de les partager avec des amis[19]. Celle-ci compte plus de 6 millions d'utilisateurs dans le monde et est notée 4.5/5 par plus de 218'000 personnes sur le Play Store[21][20]. Son modèle économique est basé sur du freemium. En effet, l'application est gratuite mais il est possible de payer pour enlever les bannières de publicité et pour accéder à toutes les fonctionnalités. L'application contient une page d'accueil (voir figure 2.5) et un menu hamburger également appelé menu latéral (voir figure 2.7). L'utilisateur

peut créer facilement une liste de courses en appuyant sur le « plus » (voir 2.5) et ensuite il peut y ajouter des articles en appuyant sur le « plus » (voir 2.6). A chaque produit, il peut attribuer soit une catégorie personnalisée ou une catégorie préexistante. Il peut également changer la vue et y ajouter un prix, ce qui calculera dynamiquement le prix des article cochés / non-cochés.

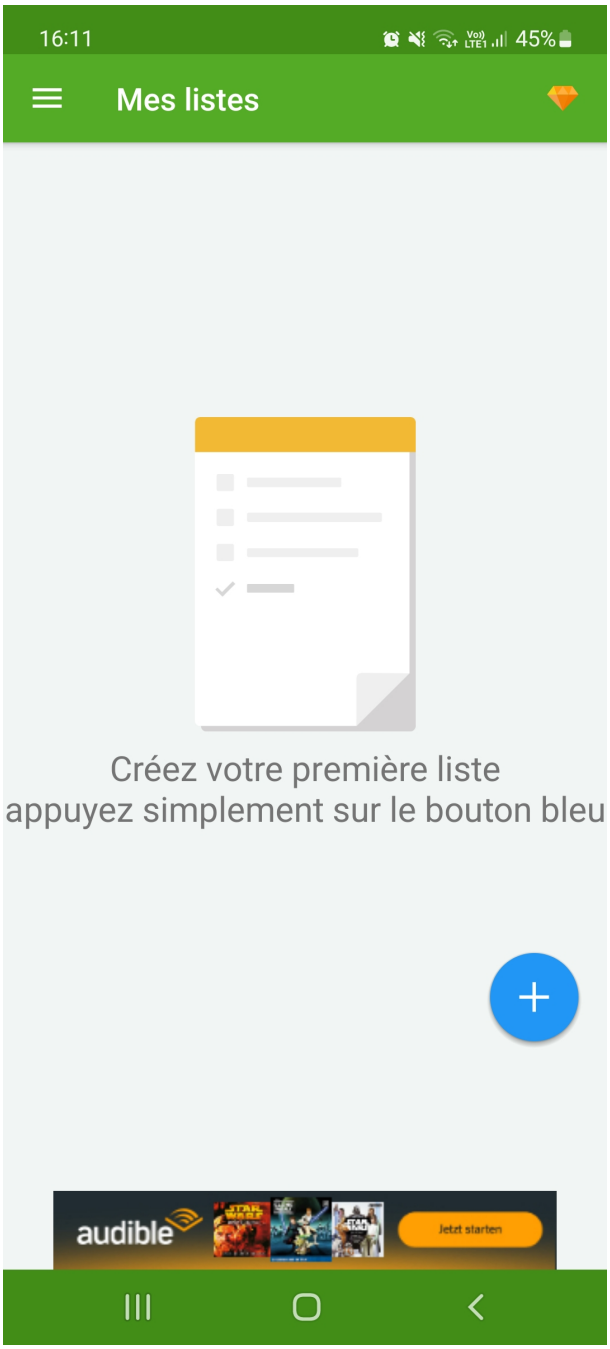

FIGURE 2.5. – Page d'accueil de Listonic

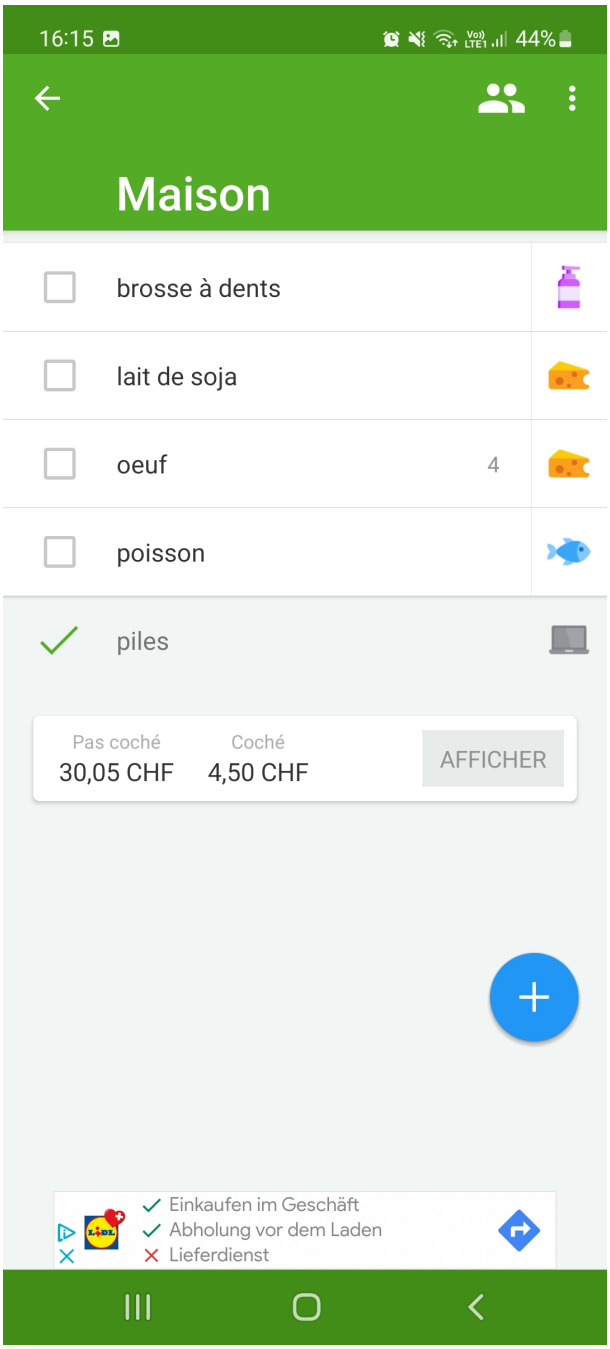

FIGURE 2.6. – Ajout de produits dans la liste de courses Listonic

Dans le menu latéral (voir figure 2.7), diverses actions sont possibles : un utilisateur peut se connecter et synchroniser la liste avec le serveur pour pouvoir y accéder depuis un autre appareil et partager la liste avec d'autres utilisateurs. Il y a également une fonctionnalité pour modifier la langue.

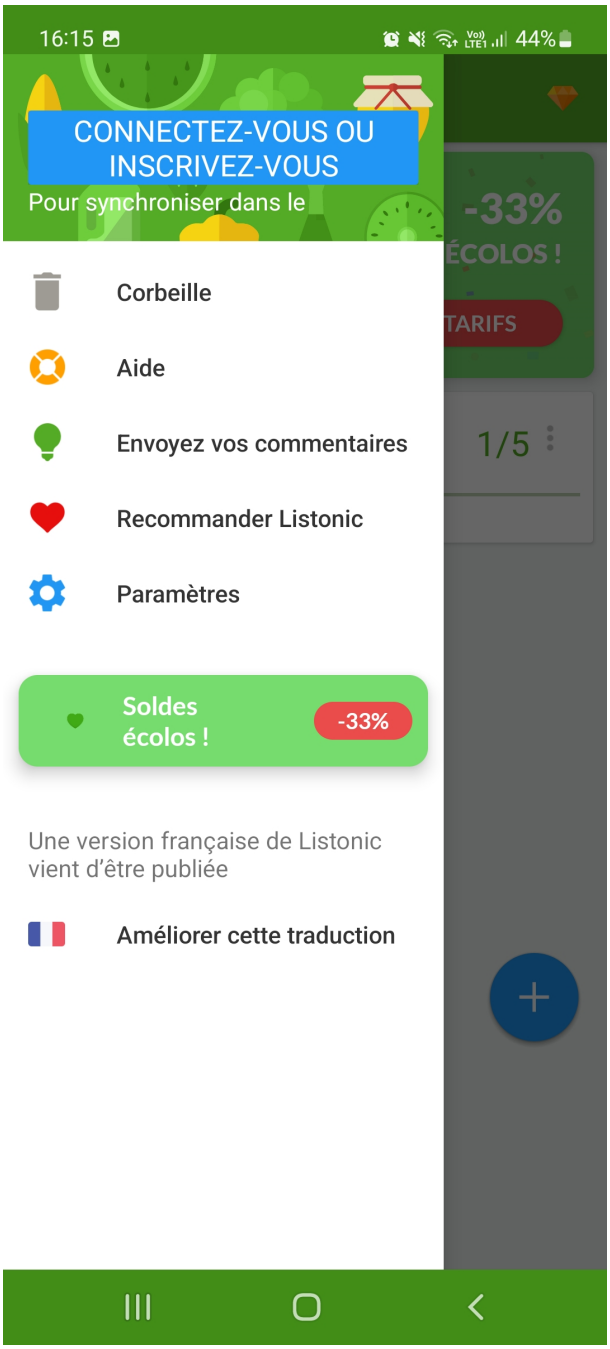

FIGURE 2.7. – Menu hamburger de Listonic

2.2.3. Résultats de la découverte

Ces deux applications populaires remplissent un même but, à savoir fournir un gestionnaire de listes de courses. Cependant, elles sont différentes dans leur conception. Autant Listonic reprend très simplement et efficacement le principe de la liste de courses papier, autant Bring ! va plus loin en proposant des recettes, des magazines et un design moderne de la liste de course avec des tuiles.

Au niveau des fonctionnalités, dans les deux applications, il est possible de créer une liste de courses et d'y d'ajouter un produit, de modifier le produit ou de le supprimer de la

liste. Les deux invitent l'utilisateur à se connecter et à partager la liste avec des amis. Il est également possible dans les deux applications de marquer un produit comme étant ajouté dans le caddie : sur Listonic il suffit de cocher le produit tandis que sur Bring ! presser dessus le retire de la liste. Également, chaque produit est associé à une catégorie. Dans Bring ! les catégories sont déjà définies et il n'est pas possible d'en créer de nouvelles contrairement à Listonic. Pour chaque produit, il est possible d'en définir une quantité, mais il est uniquement possible d'y attribuer un prix dans l'application Listonic. Bring ! fournit également des fonctionnalités pratiques telles qu'utiliser Google Assistant pour ajouter simplement des produits dans la liste ou la possibilité d'enregistrer toutes les cartes fidélité sur l'application.

Pour l'application Mescourses : un gestionnaire de listes de courses, on retient les fonctionnalités suivantes : CRUD liste de courses, CRUD produit, CRUD catégorie et le prix par article où CRUD veut dire Create, Read, Update, Delete soit créer, lire, modifier, supprimer.

2.3. Découverte structurée

Il est possible de décrire d'une multitude de façons différentes des personas, scénarios ou user stories. Ce travail utilise la structure proposée dans l'ouvrage de M. Sommerville[3].

2.3.1. Personas

Une persona est un potentiel utilisateur, fictif, qui utilisera l'application. Il est utile d'en créer car cela permet d'avoir un point de vue différent du nôtre en se mettant dans le rôle de cette persona. Avec chaque persona, il va être possible de dériver de nouvelles fonctionnalités utiles pour notre application grâce aux motivations de chacune. En effet, chacune va utiliser différemment le logiciel et cela va également donner au développeur un design d'interface. Par exemple, un ingénieur en informatique et un psychologue utiliseront différemment une application. Il est recommandé de représenter quatre principaux aspects pour une persona.

- La personnalisation qui consiste à donner un nom, un âge et un lieu d'habitation.
- Un travail si c'est pertinent pour l'application.
- Une éducation qui décrit le parcours scolaire de la personne et leur niveau de compétences. Ceci est un point important car suivant le niveau de compétence informatique de la personne, le design de l'application changera.
- La pertinence de cette persona, à savoir les raisons de l'utilisation du logiciel.

Première persona Antoine : Antoine a 18 ans et vit seul dans un studio à Fribourg. Il est apprenti électricien et doit, durant la semaine, jongler entre les cours et le travail. Ayant un diplôme du cycle d'orientation, il a été confronté modérément à la technologie. Antoine, ayant des journées bien remplies, oublie souvent des détails quotidiens et en particulier les produits qu'il est censé se procurer. De plus, il n'a pas beaucoup de temps libre et peu de moyens financiers avec son salaire d'apprenti.

Deuxième persona Jeanne : Jeanne, âgée de 45 ans vit à Berne. Elle était secrétaire et suite à l'arrivée de son deuxième enfant elle est maintenant femme au foyer. Elle a donc du temps pour cuisiner. Ayant un CFC d'employée de commerce, elle a été confrontée

plusieurs fois avec divers logiciels et donc sait utiliser une application. Elle adore faire de la pâtisserie et doit s'assurer d'avoir toujours tous les ingrédients : c'est pourquoi elle demande une application qui lui permettra de combler ce besoin.

2.3.2. Scénarios

Un scénario est une petite histoire qui va mettre en scène un utilisateur (persona) qui utilise l'application dans une certaine situation. Le protagoniste est confronté à un problème et va le résoudre au moyen de l'application. Un scénario doit se focaliser sur ce que pense la persona plutôt que de couvrir toutes les utilisations possibles de l'application ; il est également concevable de créer autant de scénarios que nécessaire pour chaque persona. Un scénario se doit d'avoir une structure dans le but de ne pas oublier des détails. Il doit donc comporter :

— Un nom de scénario.

— Une ou des persona(s) qui nous renseigne sur ses compétences et sa motivation.

— Un objectif général qui décrit l'objectif du scénario.

— Une implication, en d'autres termes tout ce qui est impliqué dans le but d'atteindre l'objectif.

— Un problème.

— Les moyens par lesquels le problème pourrait être résolu, soit une solution.

Scenario : Liste de courses. Antoine demande à avoir une liste de course pratique et une fonction budget. En utilisant la liste de courses, Antoine voudrait pouvoir noter facilement toutes les courses à faire sans perdre la liste tout en gardant un œil sur les aliments déjà en sa possession. Il souhaiterait aussi faire des économies, n'ayant pas beaucoup de moyens. C'est pourquoi il demande une application qui reprend le principe de la liste de course en papier qui dresse également un budget par ingrédient ainsi que le total actuel de ses dépenses.

Scenario : Lister les ingrédients. Jeanne demande à avoir une liste des ingrédients dans laquelle elle peut noter les quantités ainsi que cocher chaque ingrédient dans son panier afin d'être sûre d'en oublier aucun. Ayant déjà fait un grand nombre de recettes, Jeanne sait que certains ingrédients nécessitent une certaine quantité et ne sont pas disponibles dans tous les magasins et demande une application qui précisera la quantité de chaque ingrédient ainsi que le magasin dans lequel le trouver.

2.3.3. User stories

Les user stories découlent des scénarios. Elles sont beaucoup plus précises et structurées que les scénarios et permettent de communiquer les résultats souhaités sans entrer dans les détails de l'implémentation. Elles ont souvent la forme persona + besoin + finalité[34]. D'un point de vue humain, les user stories forment une liste de petits défis à réaliser pour les développeurs. Formellement, le modèle suivant est proposé pour les formuler :

— Etant un(e) <rôle>, j'(e) <veux/ai besoin> de <faire quelque chose>

Du scénario liste de courses, on obtient les user stories suivantes :

— Etant apprenti, je veux pouvoir noter tous mes achats.

— Etant apprenti, j'ai besoin d'établir un budget précis sur combien je peux dépenser par article.

— Etant apprenti, je veux voir le prix total.

— Etant apprenti, j'ai besoin de créer des catégories pour chaque produit.

Du scénario lister les ingrédients découlent :

— Etant femme au foyer, j'ai besoin de noter quelle quantité de produit je dois acheter.

— Etant femme au foyer, j'ai besoin de savoir dans quel magasin me rendre pour acheter mon produit.

— Etant femme au foyer, je veux pouvoir cocher un article dans ma liste de courses à chaque fois que je l'ajoute dans mon caddie.

— Etant femme au foyer, je veux pouvoir réutiliser une liste de courses.

2.4. Modélisation

2.4.1. Diagramme de cas d'utilisation

Les cas d'utilisations (use case en anglais) décrivent les différentes interactions entre des acteurs et un système, plus précisément comment se comporte un acteur dans un système. Les cas d'utilisations sont modélisés au moyen d'un diagramme de cas d'utilisation UML (Langage de Modélisation Unifié) qui est le standard actuel. Les diagrammes permettent de montrer les exigences du système entier ou d'une partie de celui-ci. Il est également possible de modéliser l'application à l'aide d'un seul ou plusieurs diagrammes. Un autre avantage du modèle de cas d'utilisations est l'identification des classes requises pour le projet. D'un point de vue humain, ces croquis participent à la bonne compréhension des fonctions que l'application va offrir aussi bien pour les clients que pour les développeurs. Les diagrammes de cas d'utilisation contiennent les éléments suivants[18] :

— Des **cas d'utilisation** qui sont des fonctions utiles (ou action) qui doivent faire partie du système.

— Les **acteurs** qui représentent le rôle d'un utilisateur employant le système. L'acteur peut tout autant être un humain qu'une API.

— Le **système** qui est, dans le cas présent, l'application. Il peut également être une partie du système si plusieurs diagrammes de cas d'utilisation sont créés.

— Les **communications** qui relient les acteurs aux cas d'utilisation. Ils lient donc un acteur à un comportement. Chaque cas d'utilisation doit être connecté à au moins un acteur.

Pour l'application Mescourses : un gestionnaire de listes de courses, les cas d'utilisations suivants ont été dérivés à la fois des user stories et des applications populaires. Les fonctionnalités représentées dans le diagramme de cas d'utilisation seront toutes mises en oeuvre dans l'application.

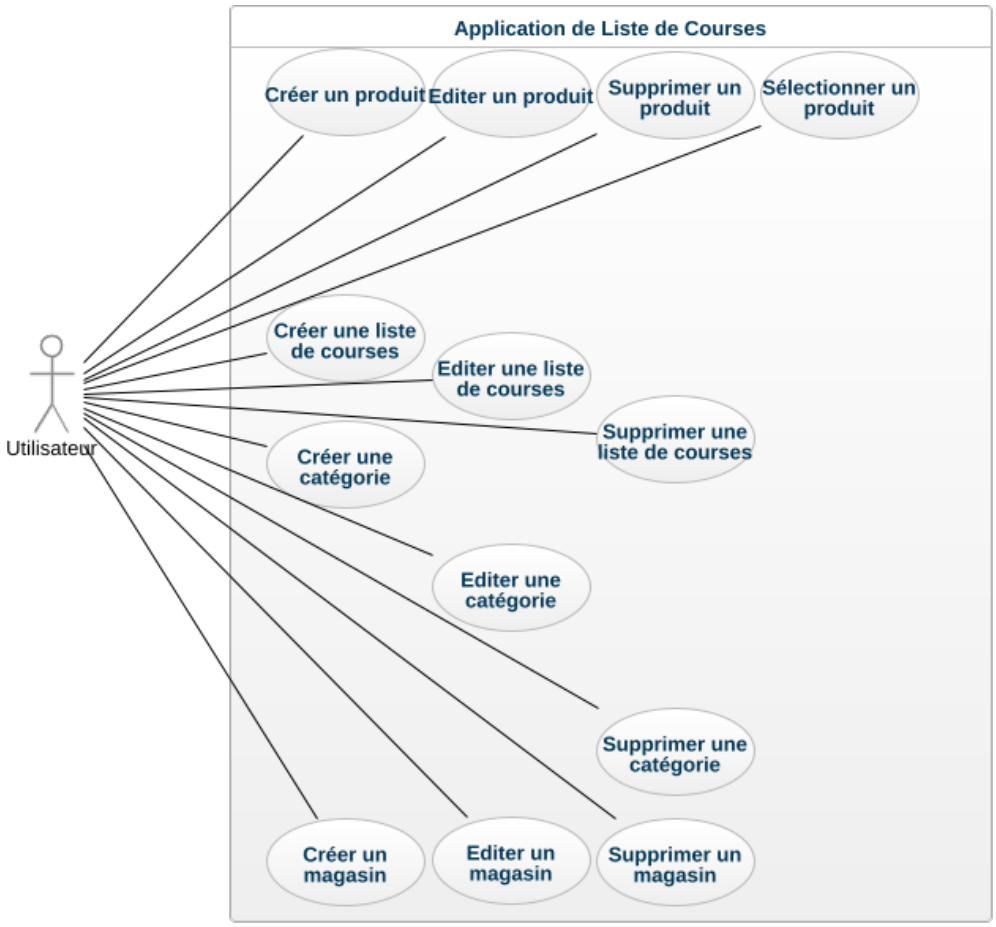

FIGURE 2.8. – Diagramme de cas d'utilisation UML

Un cas d'utilisation s'accompagne non seulement d'une représentation graphique comme ci-dessus en 2.8 mais aussi d'une description textuelle structurée, comme précisée dans le cours de M.Pasquier-Rocha[2].

Uniquement les cas d'utilisation consistant à "créer une liste de courses", "éditer une liste de courses" et "supprimer une liste de courses" sont précisés, étant donné la forte ressemblance entre eux (par exemple les cas d'utilisation "créer une liste de courses" et "créer une catégorie" seront quasiment identiques)

Nom	Créer une liste de courses.
Précondition	L'utilisateur se trouve sur la page d'accueil.
Déclencheur	L'utilisateur presse sur ajouter une liste de courses.
Action	L'utilisateur donne le nom de la liste de courses.
Postcondition	La liste de courses est créée.
Variation 1	L'utilisateur ne donne pas le nom de la liste de courses.
Postcondition	La liste de courses n'est pas créée.

TABLE 2.1. – Cas d'utilisation : créer une liste de courses

Nom	Editer une liste de courses.
Précondition	1. L'utilisateur a créé une liste de courses. 2. L'utilisateur se trouve sur la page d'accueil.
Déclencheur	L'utilisateur presse sur l'icône modifier de la liste de courses en question.
Action	L'utilisateur modifie les champs voulus.
Postcondition	La liste de courses est correctement modifiée.
Variation 1	L'utilisateur annule l'opération.
Postcondition	La liste de courses n'est pas modifiée.

TABLE 2.2. – Cas d'utilisation : éditer une liste de courses

Nom	Supprimer une liste de courses.
Précondition	1. L'utilisateur a créé une liste de courses. 2. L'utilisateur se trouve sur la page d'accueil.
Déclencheur	L'utilisateur presse sur l'icône supprimer de la liste de courses en question.
Action	L'utilisateur confirme la suppression.
Postcondition	La liste de courses est correctement supprimée.
Variation 1	L'utilisateur annule l'opération.
Postcondition	La liste de courses n'est pas supprimée.

TABLE 2.3. – Cas d'utilisation : supprimer une liste de courses

2.4.2. Modélisation des données

La modélisation des données se fait traditionnellement au moyen d'un modèle entité-association qui permet de conceptualiser les bases de données relationnelles. Le modèle

est représenté au moyen de la notation Chen [10][1]. Il démontre comment des entités tels qu'un produit ou une catégorie sont reliées entre elles au sein d'un système[12]. Les élipses sont des attributs qui peuvent être reliés soit aux entités, soit aux associations. Les attributs soulignés sont les clés primaires qui permettent d'identifier de manière unique chaque entrée. Les entités sont des noms et sont représentées par des rectangles. Les associations sont des verbes et sont désignées par des losanges. Les lignes montrent les connections. Les entités et les relations imitent une structure grammaticale[12]. On peut donc lire le modèle 2.9 ci-dessous ainsi :

- Une liste de courses spécifique contient zéro, un ou plusieurs produit(s).
- Un produit spécifique est contenu par, exactement, une liste de courses.

- Un produit est associé à une catégorie.
- Une catégorie est associée à zéro, un ou plusieurs produit(s).

- Un produit réfère un magasin.
- Un magasin est référé à zéro, un ou plusieurs produit(s).

Bien que ce travail n'utilise pas de base de données relationnelles, il est intéressant de voir comment les données sont stockées et quelles relations existent entre elles.

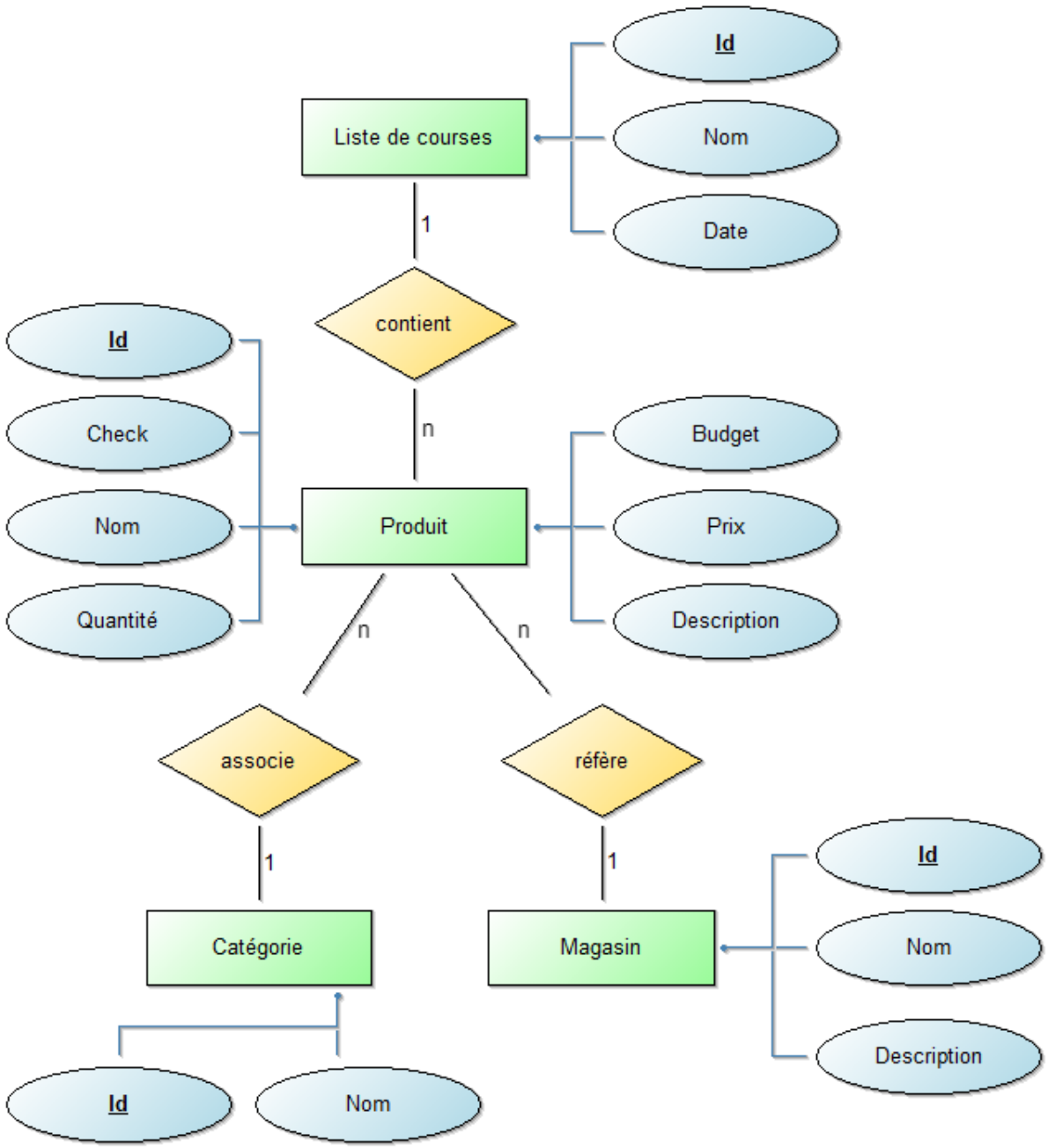

FIGURE 2.9. – Modèle entité-association

3
Développement

3.1. Stack MEAN

Cette application de gestion de liste de courses est développée en utilisant le stack MEAN. Un stack est un ensemble de logiciels utilisés pour développer une application et qui ne requiert pas d'autres logiciels[33]. Le stack MEAN par conséquent contient les logiciels MongoDB, Express.js, Angular et Node.js qui utilisent JavaScript pour construire des applications web dynamiques. Tous ces logiciels sont gratuits. Le stack MEAN est une architecture trois tiers, soit une architecture à trois niveaux. Le but de cette formation est de séparer une application en trois couches logicielles empilées qui ont chacune un rôle clairement défini et qui communiquent entre elles.[22]

— La présentation des données affiche les données et dialogue avec l'utilisateur.

— Le traitement des données décrit les opérations que l'application opère sur les données en fonction des requêtes des utilisateurs. Elle s'appuie sur la couche inférieure et renverra à la couche supérieure les résultats calculés. Par exemple, ce qui se passe une fois qu'un utilisateur clique sur un bouton.

— L'accès aux données gère l'accès aux données de l'application.

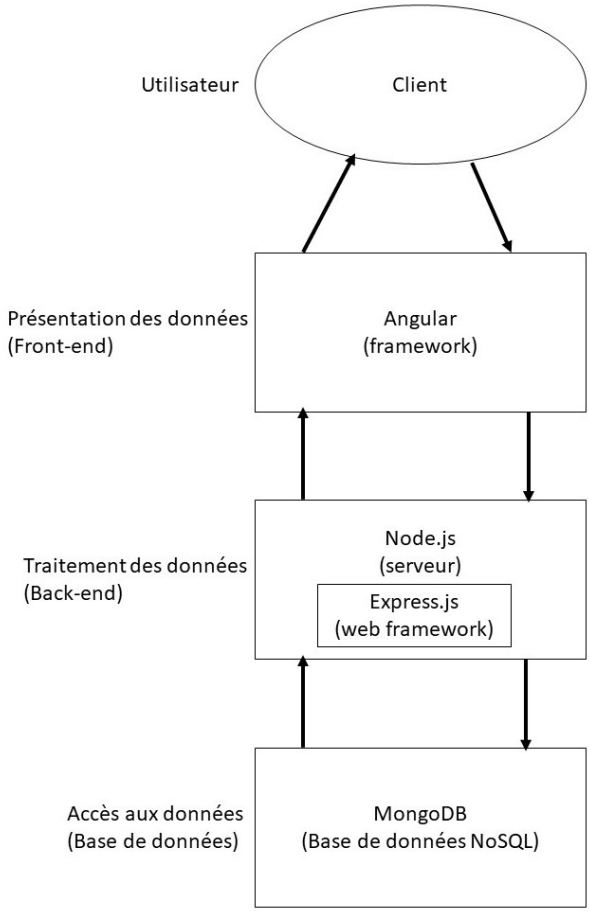

FIGURE 3.1. – Architecture à trois niveaux

Utiliser le stack MEAN offre plusieurs avantages. Il utilise le JavaScript comme seul langage de programmation. De plus, le frontend et le backend sont clairement séparés. Il y a quelques années, c'était difficile de programmer indépendamment un backend et un frontend car ils étaient tout le temps reliés. Grâce au stack MEAN, il est possible de modifier facilement l'un sans modifier l'autre. Un autre avantage est le JSON pour les objets. C'est un langage utilisé par la plupart des applications web qui est léger et facile à utiliser. Finalement, la forte communauté autour de ce stack permet de demander et de recevoir de l'aide en cas de problème.

Cependant le stack MEAN a aussi un désavantage inhérent à JavaScript. JavaScript est un langage typé dynamiquement, c'est-à-dire qu'il n'a pas de types fixes, et cela peut créer des problèmes. A noter que pour contrer ce genre d'erreurs, Angular a intégré le langage très typé Typescript.

Le code 3.1 ci-dessous illustre une erreur avec les langages typés dynamiquement. A la première ligne, on déclare une variable sans type. Ensuite on lui attribue un entier (integer). Puis, on attribue à cette variable une chaîne de caractères (string). On oublie qu'on a un string pour la variable x et on veut imprimer une opération. Lors du lancement du programme (au runtime), une erreur NaN (Not a Number) est levée. Bien sûr ici il est facile de détecter l'erreur, mais cela devient plus problématique dans un code avec des milliers de lignes.

```
1 var x;
```

```
2 x = 12;
3 x = "Bonjour";
4 console.log(x -1);
```

Listing 3.1 – Exemple de problème dans un code typé dynamiquement

3.1.1. MongoDB

MongoDB est un SGBD (Système de Gestion de Base de Données) orienté document. Un SGBD est un logiciel qui permet de créer, de gérer et d'utiliser une base de données et qui est souvent juste abrégé base de données[32]. Grâce à MongoDB, on va pouvoir stocker des données tels qu'un certain produit ou une liste de courses de manière permanente. MongoDB se différencie des SGBD traditionnels car il est NoSQL, c'est-à-dire qu'il n'utilise pas le langage SQL pour manipuler les données et qu'il n'utilise pas un schéma relationnel. Relationnel veut dire que les données sont stockées dans plusieurs tables et qu'elles sont connectées entre elles via des clés primaires et des clés étrangères. En NoSQL, les données sont stockées sans schéma fixe et en langage BSON qui est du binary JSON. Cependant, on communique avec la base de données avec des objets JSON. [6] MongoDB offre plusieurs avantages. Il n'y a pas de schéma et donc ça offre une grande adaptativité, ce qui est un avantage notamment avec les développements agiles actuels (scrum). De plus la base de données est rapide : il y a une réponse rapide aux requêtes car elle ne repose pas sur des joints complexes qu'on retrouve dans une base de données relationnelle. Cette absence de jointures complexes rend également MongoDB facile à comprendre pour les novices[23].

Pour faire un parallèle avec les SGBD relationnelles populaires tels que MySQL ou PostgreSQL, une table est une collection dans MongoDB et un enregistrement (soit une ligne dans des tables) est un document dans cette base de données NoSQL.

3.1.2. Express.js

Express.js est un framework permettant de créer des applications web basées sur Node.js. C'est le framework de développement de serveur par défaut de Node.js. Il est minimaliste mais il est possible d'étendre les fonctionnalités grâce à des plugins[35]. Il permet de construire des fonctions communes aux applications web tel que le routing qui décrit comment l'application répond aux requêtes URL du client basé sur les endpoints. Un client et un serveur communiquent en utilisant le protocole standard HTTP et un endpoint consiste en une URL et une méthode dont les plus communes sont les GET, POST, PUT, DELETE.[13]

3.1.3. Angular

Angular est un framework frontend gratuit développé principalement par Google[4]. Un framework, comme son nom l'indique, est un cadre (frame) de travail (work), qui facilite le travail des développeurs en ayant déjà implémenté les fondations. Le frontend est toute la partie visible d'un site web ou d'une application du point de vue de l'utilisateur, son interface utilisateur (UI). Par exemple un bouton ou un formulaire est du frontend et plus généralement tout élément visuel avec lequel l'utilisateur interagit est du frontend[15].

Angular est une réécriture d'Angular js, qui utilise le langage Typescript, très typé et similiaire dans sa syntaxe à Java. De plus, lors de la compilation, Angular convertit automatiquement le code Typescript en JavaScript. Angular se divise en composants, ce qui est pratique car chaque petit composant est facile à construire et à maintenir. Chaque composant comporte une classe du composant en Typescript, une template HTML et un fichier CSS tous reliés entre eux : Le fichier Typescipt gère les données et les fonctionnalités (la logique), le fichier HTML affiche le contenu du fichier Typescript sur le navigateur (il détermine l'UI) et le fichier CSS embellit la page, il définit le look and feel[5]. Un autre avantage de Angular, mais qui ne sera pas utilisé dans ce travail, est son module de test intégré. Il va utiliser Jasmine, un framework, pour tester et karma pour faire marcher le test.

3.1.4. Node.js

Node.js est un environnement d'exécution gratuit utilisant le JavaScript. Concrètement il permet aux développeurs d'écrire et de faire marcher du code en Javascript sur un serveur. Il est asynchrone, ce qui veut dire qu'il peut gérer plusieurs requêtes en parallèle sans attendre et n'attend pas non plus qu'une méthode ait fini de s'exécuter pour passer à la suivante[26]. Il est adapté aux applications en temps réel et sera utilisé pour créer notre serveur dans le backend (utilisant le même langage que le frontend Angular : JavaScript). Le backend est la partie invisible du code pour l'utilisateur d'une application et qui permet de réaliser des actions[15]. Un avantage de Node.js est qu'il est construit dans le gestionnaire de paquets NPM. NPM permet d'ajouter facilement du code écrit par d'autres développeurs dans le projet[27]. Il évite ainsi de réinventer la roue. Un des grands avantages de NPM est qu'il va spécifier la version et les paquets à installer. Ainsi, si on passe le code source à un autre développeur (sans le dossier npm modules), cet autre développeur aura uniquement à taper `npm install` dans l'invité de commande pour que NPM installe toutes les dépendances spécifiées pour le projet mais sans erreur (bug de changement de système d'exploitation, problèmes de versions).

3.2. Logo

FIGURE 3.2. – Logo

Le logo, élément essentiel pour toute application, a été pensé pour représenter au mieux une application web de liste de courses. A l'arrière plan du logo, il y a une fenêtre de navigateur internet qui représente une application web. Au premier plan il y a un caddie, qui incarne le shopping. Dans ce caddie se trouve une liste de vérification. Ensemble, le

caddie et la liste de vérification illustrent une liste de courses. Les couleurs jaunes et violet ont été choisies car elles se marient bien entre elles.

3.3. Mescourses en pratique

L'application Mescourses a été mise en œuvre en mettant un point d'honneur à la clarté du code. C'est pourquoi, on peut retrouver un backend et un frontend, le pattern modèle-vue-contrôleur, une API REST, des noms explicites pour les variables et pour les fonctions, des commentaires dans le code et un typage fort grâce au langage Typescript. Le code source de l'application se trouve sur Github [1].

3.3.1. Modèle-vue-contrôleur

Le modèle-vue-contrôleur ou sous son abréviation MVC, est un pattern très populaire pour les applications web. Il sépare une architecture logicielle en trois modules[24].

Le **modèle** contient les données à afficher. Dans l'application Mescourses, le modèle inclut l'interface de l'entité avec tous ses attributs requis ainsi que le schéma utilisé par la base de données MongoDB pour enregistrer l'entrée. Pour éviter les redondances avec les différentes entités de l'application, les exemples de code s'appuient sur l'entité produit (product en anglais) qui est la plus importante. (voir 3.2).

```
export interface IProduct {
  product_check? : boolean;
  product_name : string;
  product_quantity? : number;
  product_shop? : IShop;
  product_category? : ICategory;
  product_budget? : number;
  product_price? : number;
  product_description? : string;
  _main_list_id : String;
}

const productSchema = new Schema<IProduct>({
    product_check : { type: Boolean, default:false, required:false },
    product_name: { type: String, required: true },
    product_quantity : { type: Number, required:false },
    product_shop : { type : Schema.Types.ObjectId , default : null, ref: 'Shop',
        required:false},
    product_category : { type : Schema.Types.ObjectId , default : null, ref: 'Category
        ', required:false},
    product_budget : { type: Number, required:false },
    product_price : { type: Number, required:false },
    product_description : { type: String, required:false },
    _main_list_id : {type: mongoose.Types.ObjectId, ref: 'Mainlist', required:true},
  },
  {versionKey : false}
);

```

[1] https ://github.com/Plancherelo/Mescourses

```
27 export default mongoose.model<IProduct>('Product', productSchema)
```

Listing 3.2 – Modèle produit

La **vue** est ce que l'utilisateur voit. Dans Mescourses, la vue est dans la partie frontend, plus précisément dans le fichier products.component.html d'Angular.

```
1  <table mat-table [dataSource]="dataSource" class="mat-elevation-z8 demo-table" matSort
       >
2      ...
3    <!-- Colonne Nom -->
4    <ng-container matColumnDef="product_name">
5      <th mat-header-cell *matHeaderCellDef mat-sort-header>Nom</th>
6      <td mat-cell *matCellDef="let row"> {{row.product_name}} </td>
7      <td mat-footer-cell *matFooterCellDef></td>
8    </ng-container>
9    ...
10   <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
11   <tr mat-row *matRowDef="let row; columns: displayedColumns;" [ngStyle]="{'background
         -color':row.product_check ? 'lime' : ''}"></tr>
12   <tr mat-footer-row *matFooterRowDef="displayedColumns"></tr>
13
14 </table>
```

Listing 3.3 – Extrait de la vue d'un produit en html

Le **contrôleur** contient la logique entre le modèle et la vue. Dans Mescourses, il s'agit des opérations CRUD d'un produit dans la base de données (voir 3.4).

```
1     /** Creer un produit. */
2     public static async createProduct(req : Request, res : Response, next:
          NextFunction): Promise<any>{ ... };
3
4     /** Lire un produit. */
5     public static async readProduct(req : Request, res : Response, next: NextFunction)
          : Promise<any>{ ... };
6
7     /** Lire des produits. */
8     public static async readAllProduct(req : Request, res : Response, next:
          NextFunction): Promise<any>{ ... };
9
10    /** Mettre a jour un produit. */
11    public static async updateProduct(req : Request, res : Response, next:
          NextFunction): Promise<any>{ ... };
12
13    /** Supprimer un produit. */
14    public static async deleteProduct(req : Request, res : Response, next:
          NextFunction): Promise<any>{ ... };
```

Listing 3.4 – Contrôleur produit définition de méthodes

3.3.2. Mongoose et MongoDB

L'application Mescourses utilise la librairie Mongoose pour connecter MongoDB et le serveur. Au point 2.9, le modèle entité-association de l'application était établi et il a fallu l'implémenter concrètement. Premièrement, l'anglais est utilisé à la place du français dans

le code car c'est une norme et ça évite ainsi les problèmes avec les accents tel que "é", "è", "à". Deuxièmement, toutes les cardinalités entre les entités reliées sont de l'ordre 1-n. De plus, il s'agit de la notation Chen (et non pas la notation min-max) ce qui veut dire que la clé étrangère doit être incluse chez l'entité du côté n [1]. Étant donné qu'on utilise MongoDB, on parlera de référence au lieu de clé étrangère. C'est l'entité *produit* qui contiendra toutes les id de *catégorie*, *magasin* et *liste de courses* étant du côté n de chaque entité à laquelle elle est reliée. Dans le code 3.2, les attributs *product_shop* (l.17), *product_category* (l.18) et *_main_list_id* (l.22) sont tous de type id. Ils réfèrent les entités *magasin*, *catégorie* et *liste de courses* respectivement et seront correctement appelés lorsque l'utilisateur voudra lire le produit.

3.3.3. Architecture REST

Mescourses utilise une architecture REST (**Re**presentational **S**tate **T**ransfer) ou RESTful. Rest n'est pas un protocole mais un ensemble de contraintes à respecter. Ces conventions sont les suivantes[31][28] :

— Tout est ressource et est accédé via des URI (Uniform Ressource Identifier). Ce terme désigne un élément permettant d'identifier une ressource, son nom pour une personne ou son ISBN pour un livre par exemple[11]. Plus précisément, une URL (Uniform Resource Locator) est utilisée pour définir une ressource dans une architecture REST. Une URL est une URI qui contient en plus du nom le chemin pour accéder à la ressource. Ci-dessous (voir 3.5), l'URL de la ressource "Coop".

```
1 http://localhost:1992/api/v1/shops/62d7d92fc0f92ce95e4fa0d0
```

Listing 3.5 – URL d'une ressource REST dans Mescourses

— Une architecture client/serveur. Le côté serveur qui attend les requêtes et le côté client qui émet les requêtes au client ne sont pas dépendants l'un de l'autre, sont séparés, et peuvent donc être développés indépendamment. C'est le cas dans Mescourses. Il est possible que de multiples clients envoient des requêtes aux mêmes endpoints (voir 3.3.4) et obtiennent les mêmes réponses.

— La communication entre le client et le serveur est stateless (sans état) et est faite via le protocole HTTP. Il n'y a aucune session qui est gardée en mémoire par le serveur ce qui implique que le client envoie toute les informations au serveur dans chaque requête et que le serveur traite chaque requête comme étant indépendante. En d'autres termes, la requête est auto-suffisante.

— Le client et le serveur échangent une représentation des resources. En effet, ce ne sera jamais la resource elle-même qui sera envoyée mais une copie de celle-ci.

— Les ressources sont décomposées et accessibles en plusieurs formats tels que le XML, le JPEG et le JSON. Dans Mescourses, le format JSON est utilisé.

```
1 {
2     "_id": "62d7d92fc0f92ce95e4fa0d0",
3     "shop_name": "Coop",
4     "shop_description": "Coop de la gare Fribourg"
5 }
```

Listing 3.6 – Une ressource au format JSON dans Mescourses

— L'utilisation d'une interface uniforme. En effet, il faut définir une interface claire à laquelle chaque serveur doit se conformer, ceci afin d'éviter que le client ait à s'adapter à chaque serveur auquel il émet une requête. Pour Mesccourses, qui utilise le protocole HTTP, cette interface comprend les méthodes [GET, HEAD, PUT, POST, DELETE], des URI et un échange de représentation de ressources.

3.3.4. REST API et Endpoints

Afin de mettre en pratique les contraintes d'une architecture REST, il faut créer une API REST. Une API est une interface de programmation d'application. L'API permet au client de communiquer avec la machine sans que ce dernier ne connaisse les détails de l'implémentation[29]. La communication se fait au moyen d'endpoints (point de terminaison d'API) qui est une URL, qui, pour rappel fournit l'emplacement d'une ressource sur le serveur[30]. En plus de l'URL, une méthode HTTP est envoyée. Ci-dessous les méthodes HTTP les plus utilisées et qui sont utilisées dans Mescourses.

POST (Créer) :

POST crée une ressource à cette localisation. Les données se trouvent dans le body de la requête. Dans Mescourses, pour créer un produit on applique la méthode POST à l'adresse : *http ://localhost :1992/api/v1/mainlist/ :_ main_ list_ id/products*. Cette URL donne plein d'informations. Elle indique que le protocole utilisé est le HTTP, que le serveur est localhost (donc sur la machine actuelle) et que la communication se fait sur le port 1992. Api/v1 est le préfixe de l'API. Ceci est une convention afin de garantir une maintenance dans le temps. En effet, si on change l'API à la v2, les clients utilisant la v1 pourront continuer d'utiliser l'application. */mainlist/ :_ main_ list_ id/products* est le chemin d'accès de la ressource. En français, c'est la liste de produits qui se trouve dans une liste de courses spécifique qui elle est dans une liste de liste de courses. Le code ci-dessous (3.7) montre comment un produit est créé via l'API. A la ligne 2, l'adresse à laquelle il faut faire un POST est spécifié. Pour créer un produit, on récupère d'abord tous les champs qui sont fournis dans le corps de la requête ; ensuite on crée un nouveau produit à la ligne 8 puis on le sauve dans la base de donnée à la ligne 23. Si l'opération est un succès, on obtient le code création 201, sinon une erreur serveur 500.

```
1  /** Dans Product_routes.ts */
2  router.post(/api/v1/mainlist/:_main_list_id/products, productController.createProduct)
3
4  /** Dans productController.ts */
5  public static async createProduct(req : Request, res : Response, next: NextFunction):
       Promise<any>{
6      const { product_check, product_name, product_quantity, product_shop,
           product_category, product_budget, product_price, product_description } = req.
           body;
7
8      const product = new Product({
9          _product_id : new mongoose.Types.ObjectId(),
10         product_check,
11         product_name,
12         product_quantity,
13         product_shop,
14         product_category,
15         product_budget,
```

```
        product_price,
        product_description,
        _main_list_id : req.params._main_list_id

    });

    return product
        .save()
        .then((product) => res.status(201).json( { product }))
        .catch((error) => res.status(500).json( { error } ));
};
```

Listing 3.7 – POST produit

GET (Lire) :

GET demande une représentation de la/les ressource(s) spécifiée(s). Dans Mescourses, pour obtenir tous les produits qui se trouvent dans une certaine liste de courses, il faut utiliser la méthode GET à l'adresse :
http ://localhost :1992/api/v1/mainlist/ :_main_list_id/products. La requête va donc renvoyer une représentation de tous les produits qui se trouvent dans cette liste de courses spécifique. Ci-dessous (3.8), le code pour GET les produits d'une certaine liste de courses. Pour les obtenir, on va rechercher dans la base de données et retourner tous les produits qui contiennent l'id de la liste de course désirée. Populate sert à retourner les objets magasins et catégories qui sont uniquement enregistrés en tant qu'id dans le document produit.

```
/** Dans Product_routes.ts */
router.get(/api/v1/mainlist/:_main_list_id/products, productController.readAllProduct)

/** Dans productController.ts */
public static async readAllProduct(req : Request, res : Response, next: NextFunction):
     Promise<any>{
    return Product.find( {_main_list_id: req.params._main_list_id} ).populate('
        product_shop').populate('product_category').exec()
        .then((products) => res.status(200).json( { products }))
        .catch((error) => res.status(500).json({ error }));
    };
```

Listing 3.8 – GET produits

PUT (Mettre à jour toute la ressource) :

PUT met à jour toutes les informations de la ressource, tout en gardant la même id. Cette méthode n'est pas utilisée dans l'application.

PATCH (Mettre à jour certains champs) :

PATCH met à jour uniquement certains champs de la ressource. Cette méthode a été choisie dans Mescourses à la place de PUT car elle est plus confortable pour l'utilisateur qui pourra uniquement modifier les champs voulus. Lors d'un PATCH, Mongoose va rechercher l'id du produit. Si le produit est trouvé, alors celui-ci est mis à jour avec la méthode *set()* et est enregistré dans MongoDB au moyen de la méthode *save()*. Lorsque l'opération est bien terminée, le code 201 est retourné. Si, le produit n'est pas trouvé, une erreur 404 est levée. Si quelque chose s'est mal passé, une erreur 500 apparaîtra.

```
/** Dans Product_routes.ts */
```

```
router.patch(/api/v1/mainlist/:_main_list_id/products/:_product_id", productController
    .updateProduct);

/** Dans productController.ts */
public static async updateProduct(req : Request, res : Response, next: NextFunction):
    Promise<any>{
    const product_id = req.params._product_id;

    return Product.findByIdAndUpdate(product_id)
        .then((product) => {
            if (product) {
                product.set(req.body).save(),
                res.status(201).json({ product })
            } else {
                res.status(404).json({ message : 'Product not found' });
            }
        })
        .catch((error) => res.status(500).json({ error }));
};
```

Listing 3.9 – PATCH produit

DELETE (Supprimer) :

DELETE supprime la ressource spécifiée. Avec Mongoose, il suffit d'utiliser la fonction *findByIdAndDelete()* qui trouve une ressource grâce à son id et la supprimera. Dans Mescourses, lors de la suppression d'une liste de courses, tous les produits que la liste contenait sont également supprimés de la base de données. Cela évite une base de données encombrée par plein d'enregistrements inutilisables. Lors de la suppression d'un magasin ou d'une catégorie, toutes les références sont supprimées dans les produits. C'est-à-dire que les produits qui contenaient une de ces entités ont maintenant un champ null, qu'il est possible de réaffecter à un autre magasin ou catégorie au moyen d'un PATCH par exemple.

```
/** Dans Product_routes.ts */
router.delete(/api/v1/mainlist/:_main_list_id/products/:_product_id",
    productController.deleteProduct);

/** Dans productController.ts */
public static async deleteProduct(req : Request, res : Response, next: NextFunction):
    Promise<any>{
    const product_id = req.params._product_id;

    return Product.findByIdAndDelete(product_id)
        .then((product) => {
            if (product) {
                res.status(200).json({ message: 'Product deleted' })
            } else {
                res.status(404).json({ message : 'Product not found' })
            }
        })
        .catch((error) => res.status(500).json({ error }));
};
```

Listing 3.10 – DELETE produit

3.3.5. Diverses fonctionnalités supplémentaires

Mescourses adopte un design responsive. C'est-à-dire que les différentes pages s'adaptent automatiquement à la machine sur laquelle elles sont affichées. La commande *ng generate @angular/material :navigation <component-name>*, trouvable dans la documentation d'Angular, a permis de générer une base responsive qu'il a fallu ensuite adapter[25]. Ci-dessous (voir 3.3), le design sur un ordinateur et sur un smartphone.

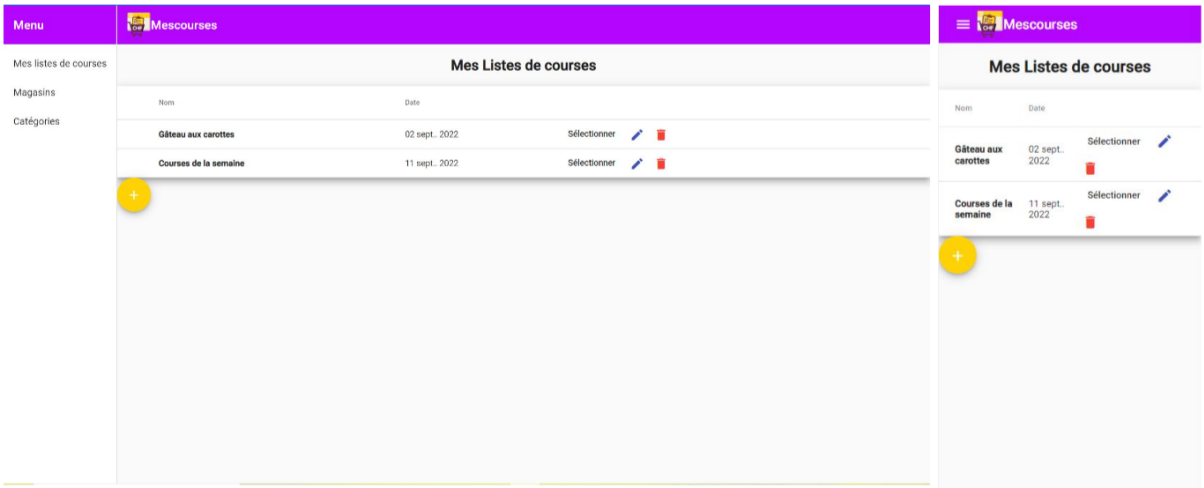

FIGURE 3.3. – Page d'acceuil Mescourses sur ordinateur (à gauche) et smartphone (à droite).

Une autre fonctionnalité est la possibilité de rechercher un produit ou d'appliquer un filtre. Par exemple, on veut voir tous les produits qu'on achètera chez Coop. Bien que l'implémentation pour filtrer des chaînes de caractères soit founie par Angular, le challenge a été de faire fonctionner le filtre pour les objets tels que les magasins ou les catégories.

L'application permet également de trier les colonnes des différentes entités par nom alphabétique ou nombre croissant/décroissant. Il peut être pratique de trier les listes de courses par date ou de trier les produits par magasins par exemple.

4

Utilisation de l'application

4.1. Utilisation de l'application

Cette section applique le scénario lister les ingrédients (voir 2.3.2). Le but est de se rendre compte que l'application fonctionne bien.

4.1.1. Liste de courses

Jeanne prévoit de faire un gâteau aux carottes pour sa famille. Elle ouvre son frigo et remarque qu'il lui manque des carottes, des oeufs, du sucre et des citrons. Elle doit aller faire des courses, et avec l'inflation actuelle doit faire attention à ses dépenses. Elle en vient à télécharger sur sa tablette l'application Mescourses depuis le Google Play Store. Une fois l'application lancée, elle arrive sur la page d'accueil. Tout de suite, le design de Mescourses lui plaît. En effet, elle comprend intuitivement comment utiliser l'application.

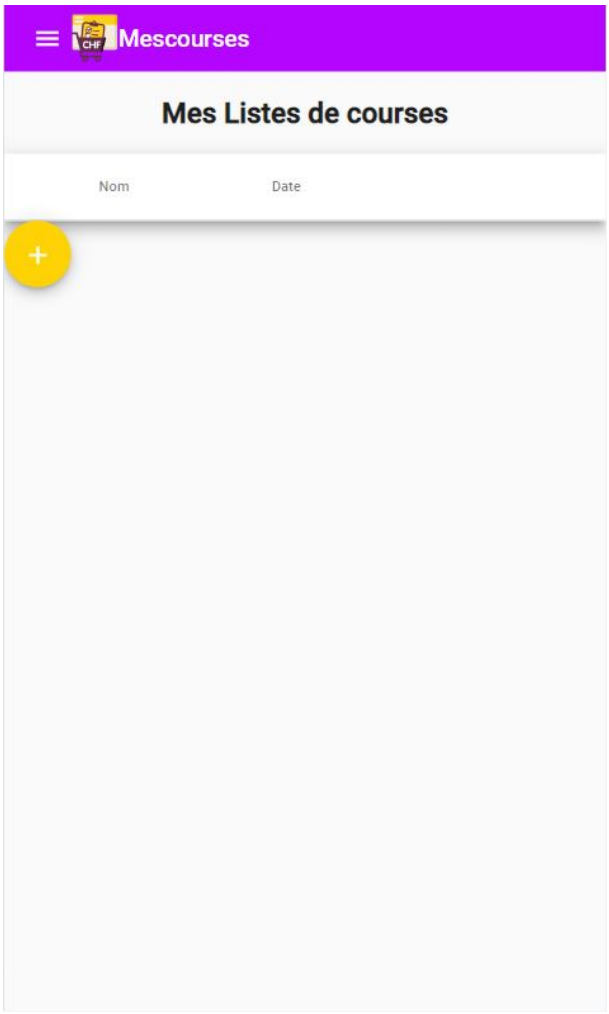

FIGURE 4.1. – Page d'accueil de Mescourses

Après avoir appuyé sur le "Plus" jaune (cf. bord inférieur gauche de la figure 4.1), elle crée la liste de courses comme illustré par les figures 4.2 et 4.3. Elle prévoit de faire ses courses aujourd'hui, le 2 septembre 2022.

FIGURE 4.2. – Ajout liste de courses

FIGURE 4.3. – Lister liste de courses

4.1.2. Menu latéral

Elle appuie sur l'icône du menu (cf. bord supérieur gauche de la figure 4.1) et le menu latéral s'ouvre comme dans la figure 4.4. Elle remarque des onglets magasins et catégories puis presse sur magasins.

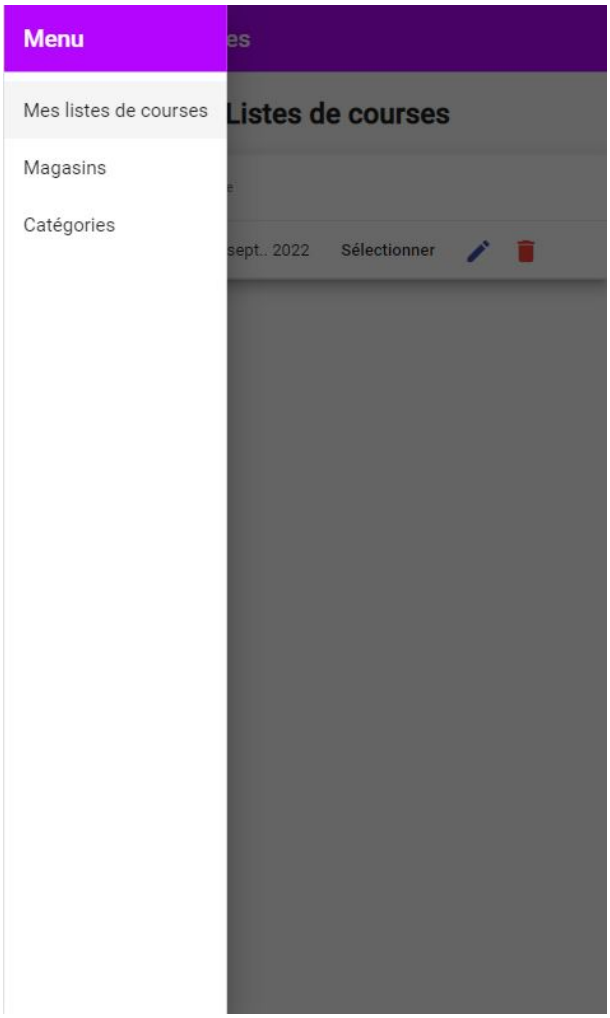

FIGURE 4.4. – Menu latéral

4.1.3. Magasins

Jeanne atterrit sur la page « Magasins » comme illustrée sur la figure 4.5. Elle se réjouit car il est possible d'ajouter des magasins. Dans son scénario elle désirait pouvoir ajouter des magasins qui lui permettront de savoir dans quel commerce trouver ses ingrédients. Elle s'empresse d'ajouter ses magasins favoris au moyen du formulaire 4.6 et elle les voit apparaître comme dans la figure 4.7.

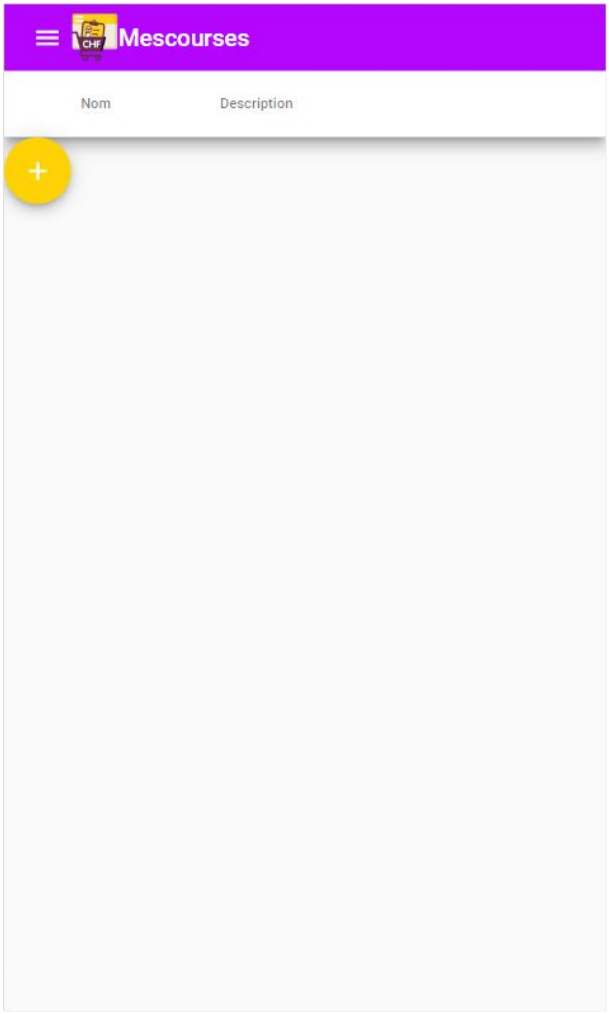

FIGURE 4.5. – Page magasins

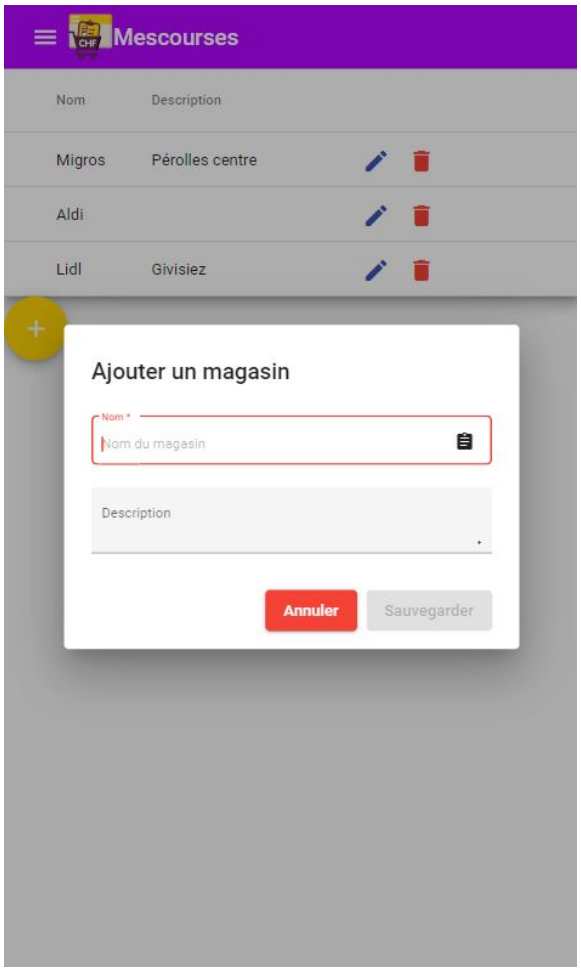

FIGURE 4.6. – Ajouter un magasin

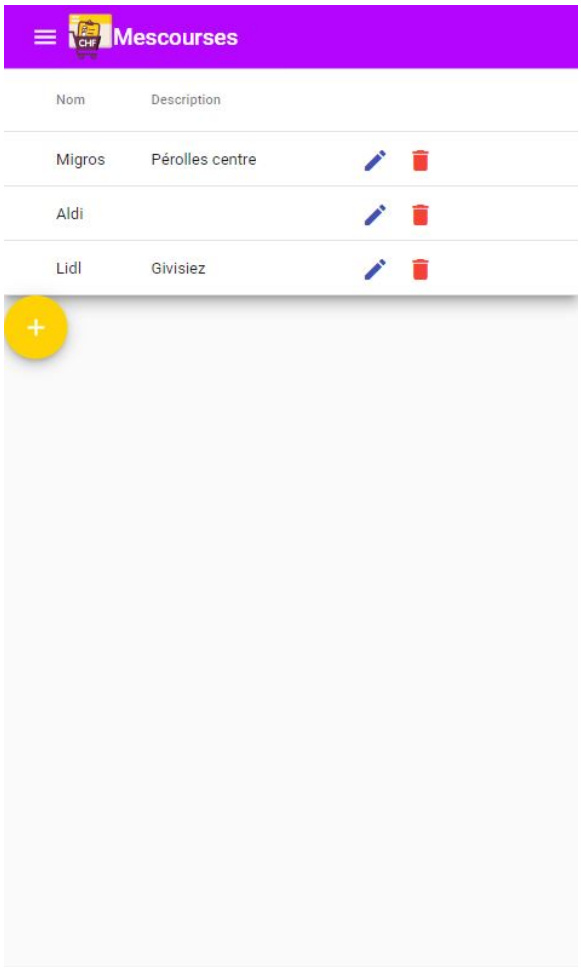

FIGURE 4.7. – Lister les magasins

4.1.4. Catégories

Jeanne retourne dans le menu en appuyant à nouveau sur l'icône menu et va dans la rubrique « Catégories ». La page, vierge pour l'instant, est illustrée à la figure 4.8. Elle crée les catégories dont elle a besoin en entrant le nom de celles-ci au moyen du formulaire de la figure 4.9 et les voit toutes listées comme dans la figure 4.10.

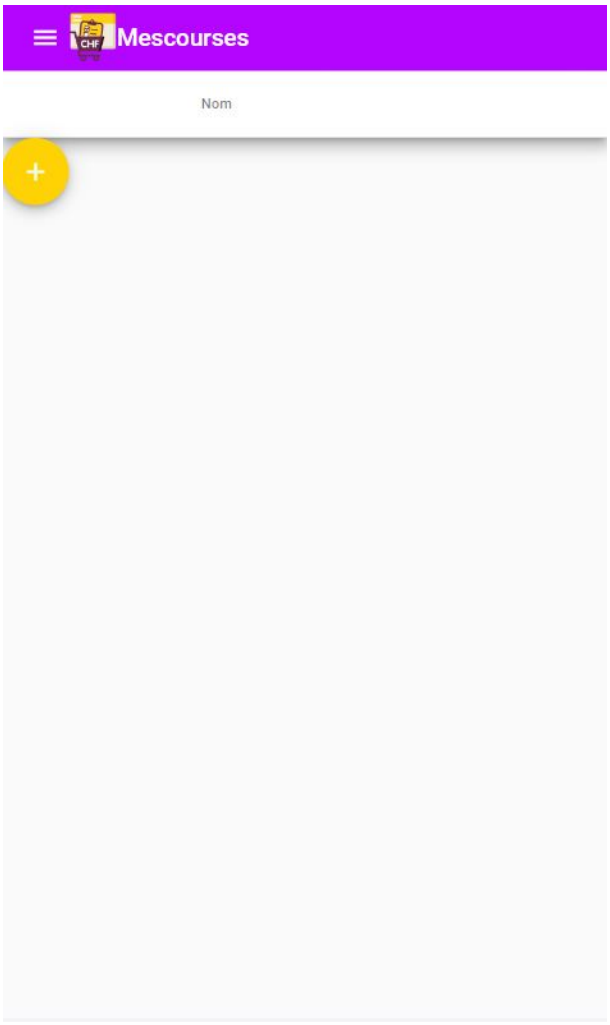

FIGURE 4.8. – Page catégories

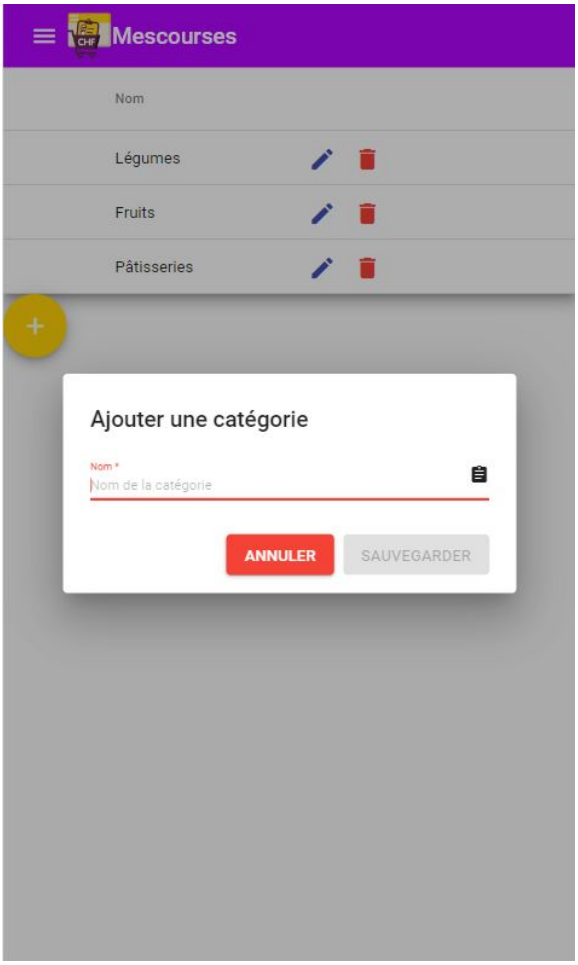

FIGURE 4.9. – Ajouter une catégorie

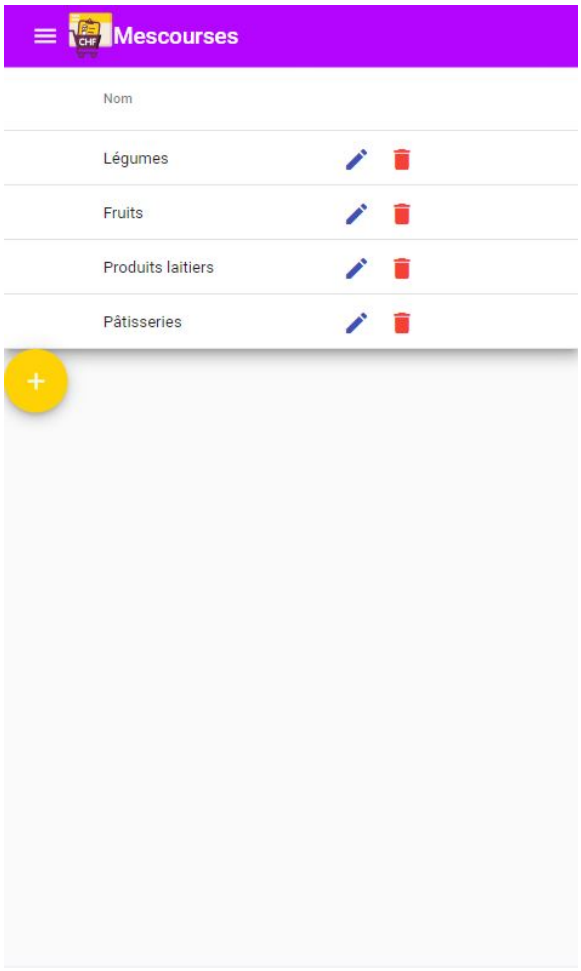

FIGURE 4.10. – Lister les catégories

4.1.5. Produits

Jeanne revient sur la page d'accueil, où sont listées ses listes de courses (cf 4.1) en pressant sur l'icône de l'application qui se trouve dans l'en-tête. Elle sélectionne sa liste de courses qui l'intéresse, en l'occurrence « Gâteau aux carottes ». Elle arrive dans l'interface de cette liste de courses comme illustré à la figure 4.11 et ajoute ensuite les produits qui lui manquent au moyen du formulaire 4.12. Elle précise également un budget qu'elle souhaite allouer à chaque produit. Dans la figure 4.13, elle vient d'effectuer une partie de ses achats à la Migros, étant donnée qu'ils sont colorés en vert et elle se dirige vers Aldi.

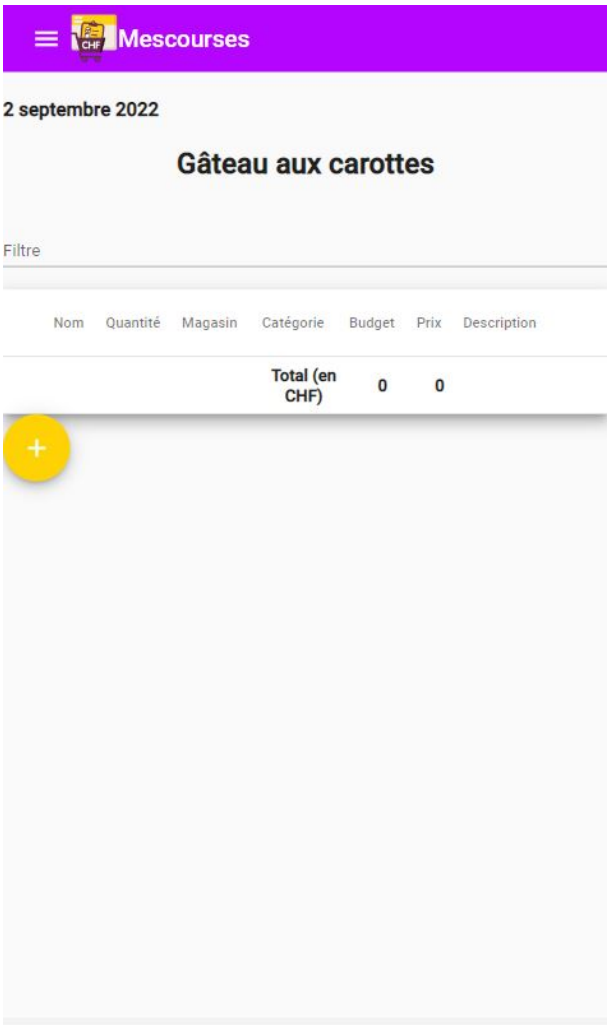

FIGURE 4.11. – Page des produits de la liste de courses "Gâteau aux carottes"

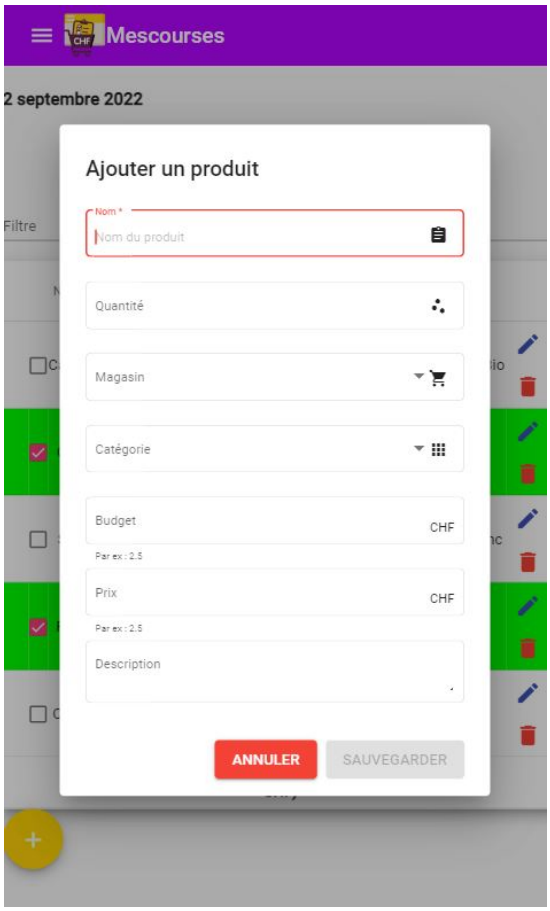

FIGURE 4.12. – Ajouter un produit

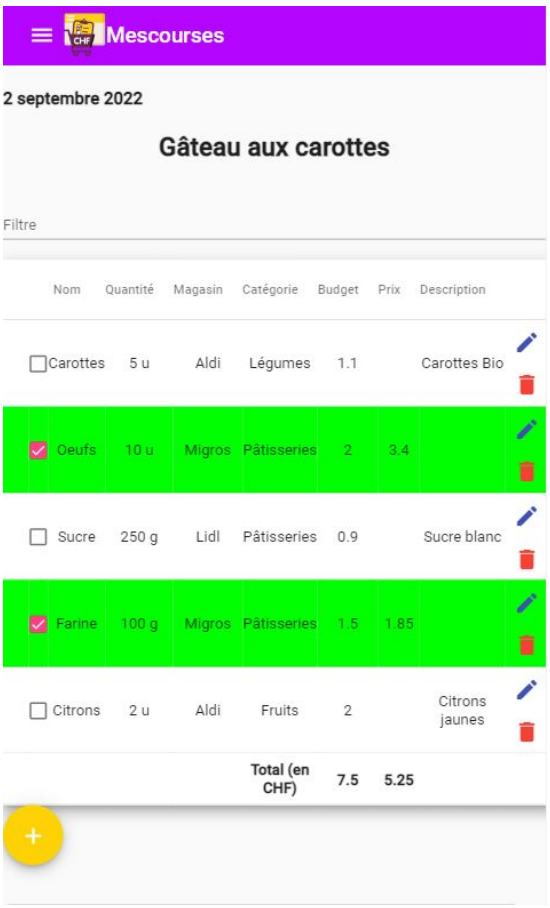

FIGURE 4.13. – Lister les produits

5
Conclusion

5.1. Résultats

Ce travail montre les différentes étapes pour la création d'une application web de gestion de listes de courses et sa mise en oeuvre. Après avoir eu l'idée de créer une application de gestion de listes de courses, il a fallu dans un premier temps définir quelles possibilités l'application offrira aux utilisateurs. La prospection de ce qui a déjà été fait ainsi qu'une approche plus formelle a été utilisée. Dans un deuxième temps, une fois les fonctionnalités clairement définies, il fallait les implémenter. Le choix pour l'implémentation s'est porté sur le stack MEAN principalement car personnellement je voulais découvrir MongoDB et car ce stack liste différents logiciels qui permettent de créer une application web client-serveur gratuitement. Finalement, toutes les fonctionnalités choisies telles que créer une liste de courses, y ajouter un produit, modifier un produit, etc... ont été implémentées correctement et fonctionnent : l'objectif est atteint.

5.2. Améliorations possibles

En l'état, Mescourses : gestionnaire de listes de courses constitue une bonne application fonctionnelle qui remplit son rôle. Elle est en effet sobre et bien exécutée. Cependant, on peut toujours rajouter certaines améliorations.

— Une bonne amélioration serait de pouvoir s'authentifier et ainsi avoir plusieurs utilisateurs.
— On pourrait utiliser le protocole sécurisé HTTPS. Pour ce faire, il faudrait ajouter un certificat SSL au protocole HTTP. Ainsi, les données échangées entre le client et le serveur seraient chiffrées et donc illisibles pour un tiers. On empêcherait, par exemple, une attaque de « l'homme du milieu » qui intercepte et lit les données échangées.
— Une autre amélioration serait de pouvoir utiliser l'application sans connexion internet active. Pour ce faire, on pourrait utiliser le stockage local du navigateur et ensuite synchroniser l'application avec le serveur lorsqu'une connexion internet active est trouvée.

Ces améliorations feraient de Mescourses une application très complète pouvant challenger les autres applications de gestion de listes de courses populaires tel que Bring ! et Listonic sur le Google Play Store et l'App Store.

A

Licence de la Documentation

Bibliographie

[1] Hans-Georg Fill. *Database*. Fill, Hans-Georg, 2020.

[2] Jacques Pasquier-Rocha. *Software Engineering I – Object-Oriented Methods*. Pasquier-Rocha, Jacques, 2022.

[3] Ian Sommerville. *Engineering Software Products - An Introduction to Modern Software Engineering*. Peason Education, 221 River Street, Hoboken, NJ, USA, 2020.

Sites Web

[4] Wikipedia - angular. https://fr.wikipedia.org/wiki/Angular (dernière consultation le July 10, 2022).

[5] Commencer avec angular. https://angular.io/start (dernière consultation le July 10, 2022).

[6] Oracle - différence sql et nosql. https://www.oracle.com/ch-fr/database/nosql/what-is-nosql/#:~:text=que%20le%20NoSQL%20?-,Une%20d%C3%A9finition%20du%20NoSQL,donn%C3%A9es%20dans%20un%20format%20diff%C3%A9rent (dernière consultation le July 9, 2022).

[7] Site officiel de bring ! https://www.getbring.com/en/about-us (dernière consultation le July 4, 2022).

[8] Site officiel de bring ! https://www.getbring.com/en/home (dernière consultation le July 4, 2022).

[9] Site officiel de bring ! https://www.getbring.com/magazin/migros-marketing-campaign (dernière consultation le July 4, 2022).

[10] Notation chen. https://vertabelo.com/blog/chen-erd-notation (dernière consultation le September 28, 2022).

[11] Différence entre url, uri et urn. https://danielmiessler.com/study/difference-between-uri-url/ (dernière consultation le September 2, 2022).

[12] Qu'est-ce qu'un diagramme entité-association ? https://www.lucidchart.com/pages/fr/diagramme-entite-association (dernière consultation le August 26, 2022).

[13] Express expliqué avec des exemples. https://www.freecodecamp.org/news/express-explained-with-examples-installation-routing-middleware-and-more (dernière consultation le July 10, 2022).

[14] Free Documentation Licence (GNU FDL). http://www.gnu.org/licenses/fdl.txt (dernière consultation le July 11, 2022).

[15] Front-end vs back-end : comprendre les différences. https://datascientest.com/front-end-vs-back-end (dernière consultation le July 10, 2022).

[16] Google play store bring ! https://play.google.com/store/apps/details?id=ch.publisheria.bring&hl=fr&gl=US (dernière consultation le July 4, 2022).

[17] Documentation d'ibm sur les diagrammes de classe. https://www.ibm.com/docs/fr/rsar/9.5?topic=diagrams-class (dernière consultation le July 12, 2022).

[18] Documentation d'ibm sur les diagrammes de cas d'utilisation. https://www.ibm.com/docs/fr/rational-soft-arch/9.5?topic=diagrams-use-case (dernière consultation le July 12, 2022).

[19] Everybodywiki - listonic. https://en.everybodywiki.com/Listonic (dernière consultation le July 5, 2022).

[20] Google play store listonic. https://play.google.com/store/apps/details?id=com.l&hl=fr_CH&gl=US (dernière consultation le July 5, 2022).

[21] Site officiel de listonic. https://listonic.com (dernière consultation le July 5, 2022).

[22] Architecture trois tiers - définition et explications. https://www.techno-science.net/definition/5266.html (dernière consultation le July 8, 2022).

[23] Site de mongodb - avantages de mongodb. https://www.mongodb.com/advantages-of-mongodb (dernière consultation le July 12, 2022).

[24] Wikipedia - modèle-vue-contrôleur. https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur (dernière consultation le August 27, 2022).

[25] Documentation angular. https://material.angular.io/guide/schematics#navigation-schematic (dernière consultation le September 3, 2022).

[26] À propos de node.js. https://nodejs.org/en/about (dernière consultation le July 11, 2022).

[27] À propos de npm. https://docs.npmjs.com/about-npm (dernière consultation le July 11, 2022).

[28] Qu'est-ce que rest. https://gayerie.dev/epsi-i4-web-services/http/rest.html (dernière consultation le September 2, 2022).

[29] Qu'est-ce qu'une api rest ? https://www.talend.com/fr/resources/rest-api/ (dernière consultation le September 3, 2022).

[30] Qu'est-ce qu'un point de terminaison d'api ? https://kinsta.com/fr/base-de-connaissances/api-endpoint/ (dernière consultation le September 3, 2022).

[31] Api rest : Comprendre et construire une api restful. https://practicalprogramming.fr/api-rest/ (dernière consultation le September 2, 2022).

[32] Explication du système de sgbd. https://www.ionos.fr/digitalguide/hebergement/aspects-techniques/systeme-de-gestion-de-base-de-donnees-sgbd (dernière consultation le July 9, 2022).

[33] Wikipedia - solution stack. https://en.wikipedia.org/wiki/Solution_stack (dernière consultation le July 8, 2022).

[34] User stories avec des exemples et un modèle. https://www.atlassian.com/fr/agile/project-management/user-stories (dernière consultation le July 7, 2022).

[35] Wikipedia - express.js. https://fr.wikipedia.org/wiki/Express.js (dernière consultation le July 9, 2022).

[36] Wikipedia - application web. https://fr.wikipedia.org/wiki/Application_web (dernière consultation le July 8, 2022).

UNI
FR

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Faculté des sciences économiques et sociales
Wirtschafts- und sozialwissenschaftliche Fakultät
Boulevard de Pérolles 90
CH-1700 Fribourg

DECLARATION

Par ma signature, j'atteste avoir rédigé personnellement ce travail écrit et n'avoir utilisé que les sources et moyens autorisés, et mentionné comme telles les citations et paraphrases.

J'ai pris connaissance de la décision du Conseil de Faculté du 09.11.2004 l'autorisant à me retirer le titre conféré sur la base du présent travail dans le cas où ma déclaration ne correspondrait pas à la vérité.

De plus, je déclare que ce travail ou des parties qui le composent, n'ont encore jamais été soumis sous cette forme comme épreuve à valider, conformément à la décision du Conseil de Faculté du 18.11.2013.

Pensier, le 30 septembre 2022

Olivier Blanchard
(signature)