

Élaboration d'une progressive web app : « Scidélice », carnet de cuisine intelligent

Travail de fin d'études réalisé en vue de l'obtention du diplôme de
Bachelier en Techniques graphiques

Par Théophile Desmedt
Promoteur externe: Auguste Hélène
Promoteur interne : Ivan Miller



Année académique 2022 - 2023

Table des matières

2 – Abstract
3 – Cahier des charges
3 – Objectifs
4 – Choix d'un framework
6 – Accessibilité
6 – Préparations
11 – Codage avec Angular et Ionic
20 – Conclusion
21 – Annexes

Abstract

English

In the midst of the COVID pandemic, against a backdrop of health and psychological crises, I chose this project. Like many, I realized that our physical and mental health is always hanging by a thread. It requires care, and one aspect of that is nutrition.

With this in mind, I developed Scidélice. Its goal is to facilitate the learning of healthy and delicious cooking by combining culinary and scientific knowledge and enabling the sharing of this knowledge.

Scidélice is an ever-evolving, intelligent recipe book that offers various practical tools such as a meal tracker to avoid waste, dietary guidelines, and a means to manage and search recipes. These features will allow users to manage their diet more effectively and assist them in making culinary choices based on their needs and preferences.

This project allowed me to enhance my front-end development skills while conducting an in-depth study of the mobile development market, fully realizing my strengths and weaknesses. Moreover, by contributing to a project of public interest, I was able to enrich my portfolio with an application that is close to my heart and aims to improve the health and well-being of everyone.

Français

C'est au plein cœur du COVID, dans un contexte de crises sanitaire et psychologique que j'ai fixé mon choix sur ce sujet. Comme beaucoup, j'ai pu m'en rendre compte personnellement: notre santé tant physique que mentale est somme toute toujours sur le fil. Il faut en prendre soin, et cela passe entre autres par l'alimentation.

C'est pour cela que j'ai développé Scidélice. Son but est de faciliter l'apprentissage d'une cuisine saine et délicieuse en combinant les connaissances culinaires et scientifiques, et en facilitant le partage de ces connaissances.

C'est un carnet de cuisine intelligent, en évolution constante qui se propose de mettre à disposition divers outils pratiques tels qu'un traqueur de repas pour éviter les gaspillages, des indications diététiques et un moyen de gérer et chercher ses recettes. Ces fonctionnalités permettent aux utilisateurs de gérer leur alimentation de manière plus efficace et de les assister dans leurs choix culinaires en fonction de leurs besoins et de leurs préférences.

Ce projet m'a permis de renforcer mes compétences en développement front-end, tout en réalisant une étude approfondie du marché du développement mobile, et de pleinement réaliser mes forces et mes faiblesses. De plus, en contribuant à un projet d'intérêt général,

j'ai pu enrichir mon portfolio avec une application qui me tient à cœur et qui vise à améliorer la santé et le bien-être de ses utilisateurs.

Cahier des charges

Voici les productions qui étaient prévues pour ce projet:

- Création et design d'une "Progressive Web App" CRUD
- Apprentissage de deux frameworks : Angular et Ionic
- Base de données MySQL et api PHP pour gérer les données nutritionnelles
- Stockage des données utilisateurs avec IndexedDB
- Travail de recherche sur l'état actuel des applications mobile

Objectifs

Mon objectif premier était de pouvoir toucher le plus grand public possible, pour aider le plus grand nombre de personnes. J'ai donc décidé que Scidélice soit une application axée sur les valeurs du logiciel libre et open-source, pour faciliter davantage de futures collaborations. La vie privée et le contrôle des données personnelles est un enjeu qui tient à cœur de nombreux utilisateurs, aussi je voulais permettre aux utilisateurs de pouvoir partager leurs recettes sans dépendre de grandes plateformes telles que Google ou autres géants de la technologie.

Pour garantir l'accessibilité de Scidélice, il était essentiel de développer une application pouvant fonctionner hors ligne, sans dépendre d'une connexion Internet.

Initialement, j'envisageais de créer une application native android en apprenant à coder en Kotlin et en Java. J'ai même commencé une formation dans ce sens, mais j'ai rapidement réalisé que cela demanderait un investissement de temps considérable pour combler mes lacunes en développement. En tant qu'infographiste, et comme j'ai pu m'en rendre compte lors de mon stage chez Bizzdev, mon rôle est de concevoir des interfaces et de présenter rapidement des informations et des concepts.

C'est pourquoi, après de nombreuses et longues recherches, j'ai opté pour le développement d'une Progressive Web App (PWA). Ce choix m'a permis d'utiliser et de consolider mes compétences en développement acquises lors de ma formation d'infographiste (JavaScript, HTML, CSS). De plus, cela a rendu l'application accessible à un public beaucoup plus large grâce à la compatibilité multiplateforme des technologies web.

Le déploiement de l'application en a également été (relativement) simplifié, car je n'ai pas du me préoccuper de passer par l'App Store ou le Play Store. GitHub Page permet par exemple de déployer l'application très facilement, et comme le projet est open source, n'importe qui peut déployer son propre carnet, gratuitement.

Choix d'un framework

Bien qu'il soit possible de développer une application sans framework, en utilisant les technologies web "vanilla" ces derniers offrent des avantages non négligeables :

- Valorisation sur le marché du travail
- Rapidité de développement
- Evolutivité

J'ai pris en compte plusieurs points pour choisir mon framework (ou stack).

Je voulais un framework qui me valorise sur le marché du travail, avec une documentation fournie et mise à jour, supportée par une grande entreprise pour garantir sa pérennité et son évolutivité. La courbe d'apprentissage devait idéalement être accessible, les performances bonnes sur la majorité des plateformes. J'ai aussi pris en compte l'extensibilité, les fonctionnalités, la performance, la philosophie, le support et le type de licence de chaque option existantes.

Un framework avec une grande communauté n'est pas nécessairement gage de qualité, mais c'est idéal pour un débutant car cela augmente les probabilités de pouvoir recevoir de l'aide en cas de besoin, et cela signifie également que de nombreux projets open source sont probablement disponibles et pourraient servir de référence. C'est une leçon importante que j'ai retenu de ce travail: les repo open source tels que github ou gitlab sont une mine d'or d'informations. Pour déterminer avec le plus d'objectivité possible la popularité des divers frameworks existants, après de nombreuses recherches, j'ai retenu ces sites comme sources d'information particulièrement pertinentes:

- <https://stateofjs.com>: un site web qui propose une enquête annuelle sur l'état de l'écosystème JavaScript. C'est une ressource populaire pour les développeurs JavaScript désirant suivre les tendances, les préférences et les opinions au sein de la communauté JavaScript.
- <https://survey.stackoverflow.co>: dans la même idée, un sondage organisé par l'une des ressources d'aide aux développeurs la plus populaire au monde.
- <https://npm trends.com>: npm (Node Package Manager), le gestionnaire de paquets par défaut pour l'écosystème JavaScript de Node.js. Il s'agit d'un outil en ligne de commande qui facilite l'installation, la gestion et la mise à jour des bibliothèques et des dépendances nécessaires à un projet JavaScript. Ce site m'a permis de voir quels sont les packages les plus téléchargés et ainsi de me faire une idée de la popularité des frameworks.

Après avoir fait mes recherches, j'ai pu déterminer que les frameworks ayant le plus de part du marché lorsque j'ai commencé le TFE étaient React, Angular et Vue.js.

Ils sont tous open-source et sous license MIT :

- <https://github.com/angular/angular/blob/master/LICENSE>
- <https://github.com/facebook/react/blob/main/LICENSE>
- <https://github.com/vuejs/vuejs.org/blob/master/LICENSE>

React est de loin le framework le plus populaire, de par sa facilité d'apprentissage et ses performances. Il est aussi développé par facebook, ce qui est un argument de poids et permet d'être relativement sûr qu'il continuera d'être maintenu pour de longues années à venir. J'ai cependant arrêté mon choix sur Angular et Ionic:

Ionic est une library de composants open source utilisée pour développer des applications mobiles multiplateformes. Il offre un outil pour créer des applications mobiles natives à l'aide des technologies web courantes telles que HTML, CSS et JavaScript via capacitor. Ionic utilise Angular comme principal framework JavaScript pour la création des composants et la logique de l'application.

Angular est un framework open-source développé par Google. Il propose une architecture basée sur les composants, précurseur des WebComponents, ce qui permet de diviser l'application en blocs réutilisables et autonomes. Angular propose de nombreuses fonctionnalités pour échanger les données entre les diverses parties d'une app, ajoute des directives qui permettent d'étendre l'HTML de fonctionnalités supplémentaires. Il utilise TypeScript, un superset typé de JavaScript, ce qui permet de renforcer la rigueur du code et de réduire les erreurs, facilitant ainsi le processus de débogage dès la phase de compilation. Il intègre également un système de routage qui simplifie la navigation entre les vues et les composants de l'application, facilitant ainsi la création de liens entre les pages et la mise en place de fonctionnalités avancées. C'est un framework performant, riche, réputé dans l'industrie, stable, bien documenté, avec une communauté importante et il est activement maintenu.

De plus, je bénéficie d'une expérience avec Angular grâce au stage que j'ai effectué à BizzDev, ainsi que de l'aide d'un développeur Angular (Guillaume Flament) qui a accepté de répondre à mes questions et de guider mes choix.

A l'heure où j'écris ce rapport, React connaît une popularité encore plus grande, grâce à sa facilité d'apprentissage, au fait qu'il est principalement une bibliothèque de rendu qui se concentre sur la construction d'interfaces utilisateur réactives et grâce à son approche virtuelle du DOM qui minimise les mises à jour du DOM réel, coûteuses.

Cependant, les différences de performances sont minimales (comme on peut le voir sur ce benchmark comparant les frameworks javascript:

<https://krausest.github.io/js-framework-benchmark/>), et Angular reste un

framework très demandé en 2023, surtout en Belgique. Avec l'arrivée des signaux et des composants Standalone, Angular devient de plus en plus accessible. Et, il est probable qu'Angular retrouve un regain marqué de popularité.

Accessibilité

J'ai entrepris beaucoup de recherches lors de ce TFE sur l'accessibilité et l'expérience utilisateur. Ce sont des sujets très complexes qu'il n'a pas été possible d'approfondir durant les cours, mais qui sont, je pense, essentiels à connaître pour l'avenir de tout étudiant en bachelier infographie. Mes recherches m'ont mené à trouver de nombreuses ressources utiles, qui me serviront sans doute de référence pour le reste de ma carrière :

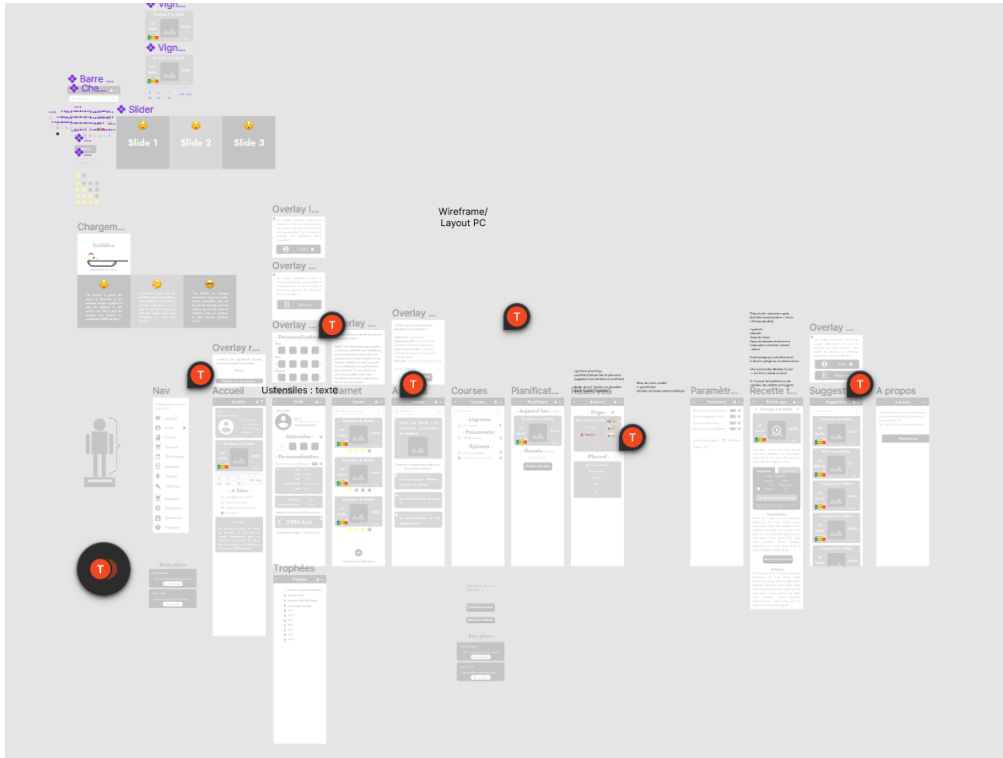
- <https://codelabs.developers.google.com/color-contrast-accessibility>
- <https://www.nngroup.com>
- <https://www.allyproject.com>
- <https://www.w3.org/WAI/ARIA/apg/patterns>

Préparations

Pour enrichir mes connaissances en UX, j'ai suivi la formation Google UX Design Certificate. J'ai appris par ce cours les étapes principales de développement d'une application. Comme vu au cours de communication, on commence par créer des personas, puis on avance en créant des wireframes, des layouts, et enfin on passe à la phase de développement.

C'est un cycle, plusieurs essais sont nécessaires et il faut souvent recommencer le design par le début, en se reposant sur les connaissances obtenues au fil des itérations successives. J'ai commencé par faire mes personas (voir annexes), puis j'ai réalisé des wireframes sur figma:

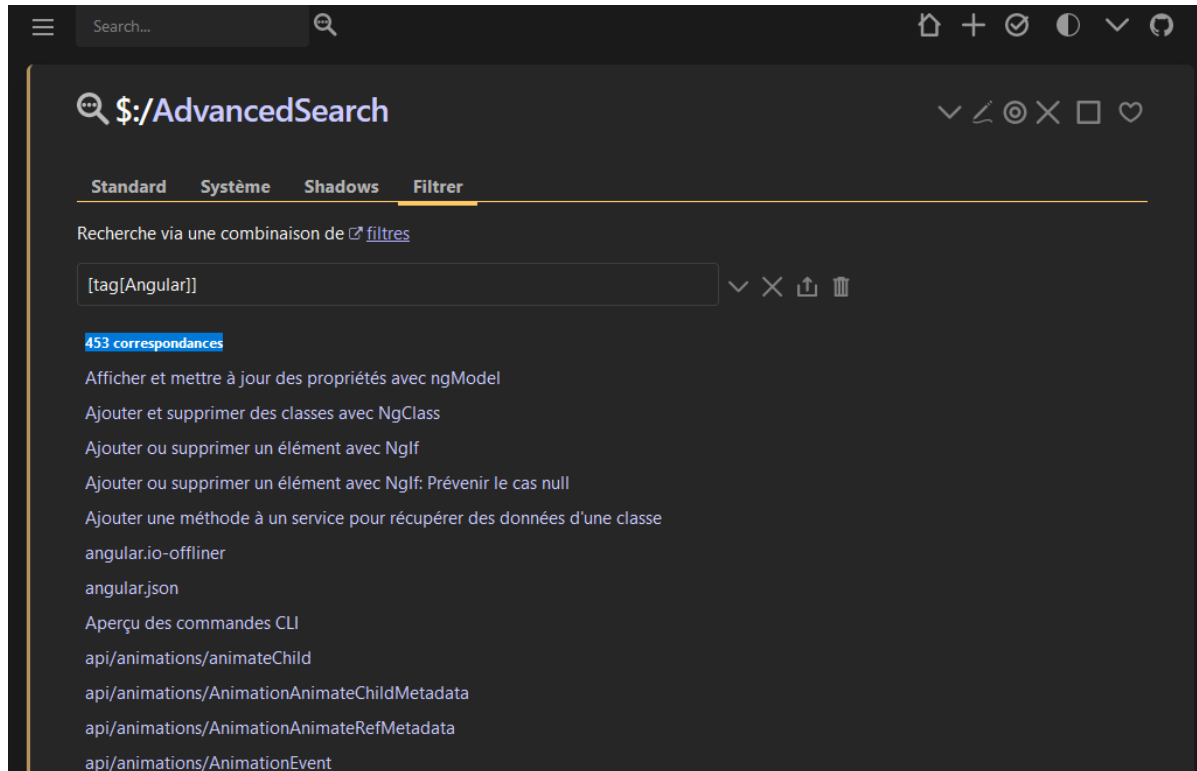
« Scidélíce », carnet de cuisine intelligent



<https://www.figma.com/file/rshXLVpBiMOjt12dbSey7n/Scid%C3%A9lice---Wireframe>

J'ai également essayé le logiciel *Penpot*, logiciel concurrent de Figma que je trouvais attractif pour son modèle gratuit et open source, mais j'ai vite pu me rendre compte des différences de performance, passé un certain nombre d'éléments sélectionnés, il devient beaucoup trop lent (vidéo: <https://youtu.be/p5R1q8qII0U>).

Lors du codage, j'ai rapidement été confronté aux limites de mes connaissances avec Angular. J'ai d'abord commencé par étudier la documentation en prenant des notes dans tiddlywiki, un outil de gestion des connaissances contenu dans un fichier html. J'ai donc entrepris d'étudier méthodiquement la documentation angular, page après page :

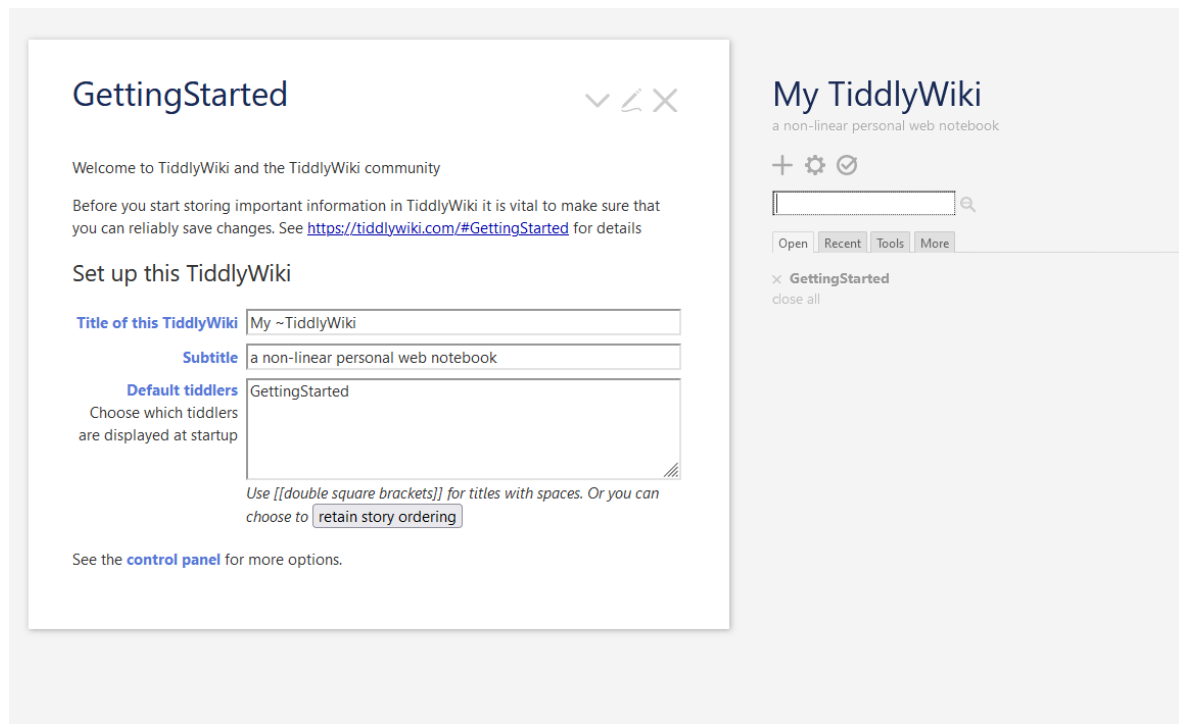


Notes personnelles de mon apprentissage d'Angular sur TiddlyWiki

Après avoir accumulé plus de 300 notes sur le sujet (comme illustré ci-dessus), je me suis retrouvé dans une impasse, je ne me sentais pas plus à l'aise dans ma maîtrise du framework. J'ai cependant réussi à tirer parti de mes connaissances acquises dans TiddlyWiki en utilisant mes compétences développées lors de mon apprentissage pour créer un prototype interactif de mon application, dans tiddlywiki lui-même.

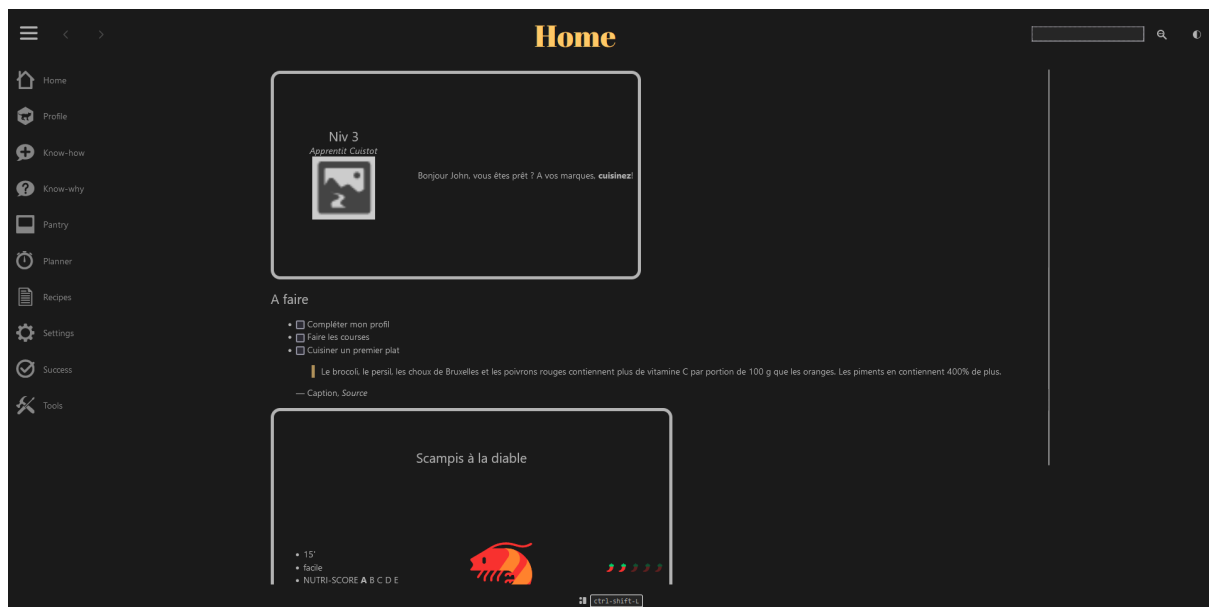
Cela m'a permis d'obtenir une vision plus claire de la structure du projet et de rendre le travail beaucoup plus abordable. TiddlyWiki est un outil intéressant car, en plus de ses capacités de prise de notes lors d'un projet nécessitant l'apprentissage de nombreuses technologies (TypeScript, Angular, Ionic, etc.), il permet de coder rapidement des interfaces en HTML/CSS sans aucun outil particulier, en offrant un retour quasi instantané. Il présente des fonctionnalités qui rappellent beaucoup Angular, comme le fait d'être une application monopage avec des composants (ou widgets dans le vocabulaire de TiddlyWiki), et une philosophie de composants réutilisables sous la forme de tiddlers, etc.

De base, Tiddlywiki ressemble à ceci:



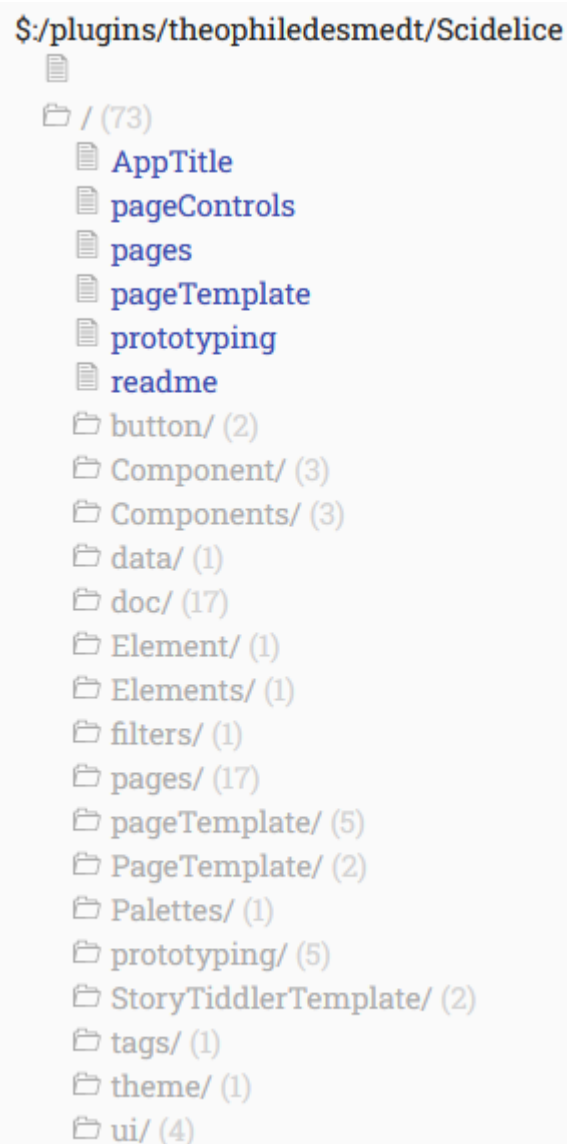
<https://tiddlywiki.com/prerelease/empty.html>

Mais on peut complètement changer son interface si on le souhaite, par exemple j'ai créé ce wireframe interactif pour prototyper l'application:



<https://theophiledesmedt.github.io/Scidelice/prototype>

Cela m'a permis de concevoir une vue sous forme d'arborescence et de commencer à créer le style des différents composants sans être dépassé par l'ampleur du projet. J'ai suivi une approche similaire au concept de Storybook, utilisé par des grandes sociétés comme Microsoft pour documenter leurs composants en isolation. Comme il s'agit d'un projet avec un seul développeur, il m'a semblé excessif d'ajouter une couche supplémentaire comme Storybook, car il n'y a pas de besoin immédiat de partager et de collaborer avec d'autres personnes. Créer une version simplifiée avec Tiddlywiki était, je pense, un bon compromis.



```
$:/plugins/theophiledesmedt/Scidelice
├── / (73)
│   ├── AppTitle
│   ├── pageControls
│   ├── pages
│   ├── pageTemplate
│   ├── prototyping
│   └── readme
│   ├── button/ (2)
│   ├── Component/ (3)
│   ├── Components/ (3)
│   ├── data/ (1)
│   ├── doc/ (17)
│   ├── Element/ (1)
│   ├── Elements/ (1)
│   ├── filters/ (1)
│   ├── pages/ (17)
│   ├── pageTemplate/ (5)
│   ├── PageTemplate/ (2)
│   ├── Palettes/ (1)
│   ├── prototyping/ (5)
│   ├── StoryTiddlerTemplate/ (2)
│   ├── tags/ (1)
│   ├── theme/ (1)
│   └── ui/ (4)
```

<https://theophiledesmedt.github.io/Scidelice/prototype/#Hiérarchie>

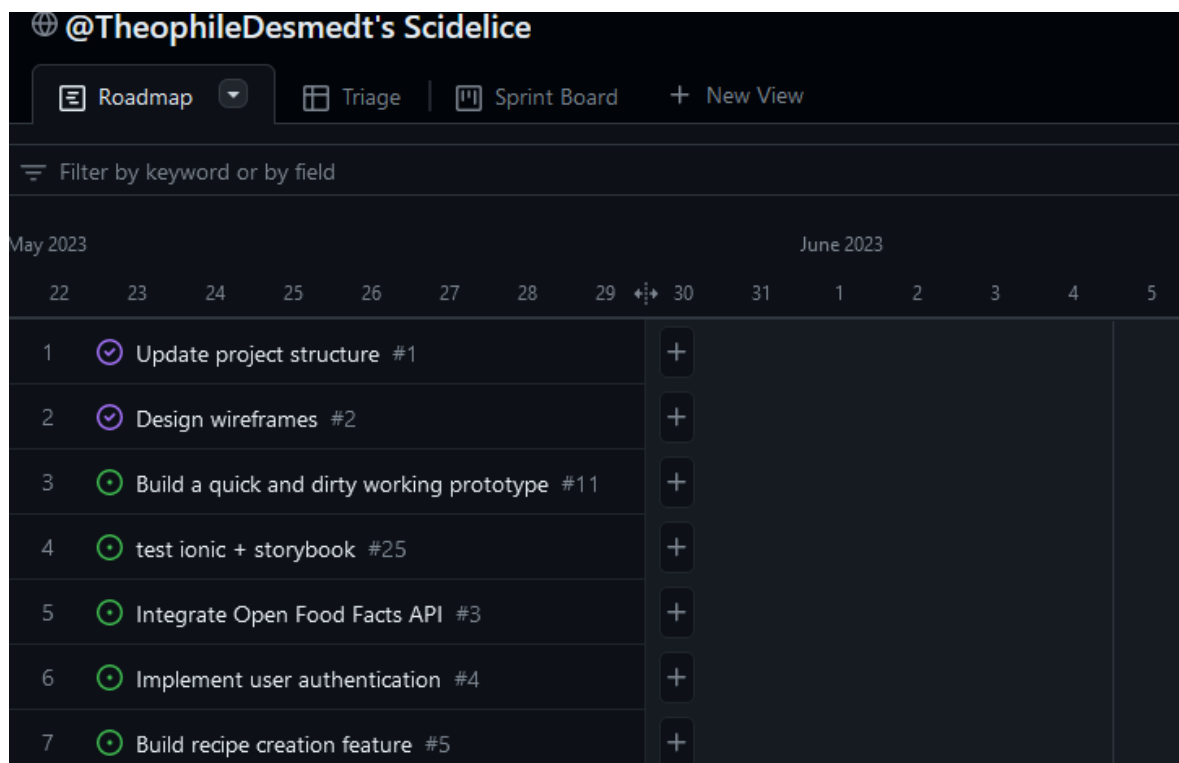
Après cela, je suis finalement passé à l'étape de développement final avec Ionic.

Codage avec Angular et Ionic

Suivi du projet avec Git et GitHub

Pour coder l'application efficacement, j'ai appris à utiliser Git et la ligne de commande, ce qui m'a permis de pouvoir garder une trace documentée de l'évolution de mon projet. Git est un outil très pratique car il permet d'avoir un historique documenté de toutes les manipulations qui ont été effectuées sur le code, permettant de facilement revenir en arrière au besoin. L'historique complet de mon projet ionic/Angular est disponible publiquement a cette adresse: <https://github.com/TheophileDesmedt/Scidelice>

Github propose entre autre de suivre les tâches du projet avec Github Project:



<https://github.com/users/TheophileDesmedt/projects/2>

Ce qui est très pratique, c'est que chaque tâche peut être associée à une "issue", c'est-à-dire à un post détaillant un problème spécifique. Chaque "issue" peut ensuite être liée à une section particulière du code, ce qui permet de suivre facilement son évolution.

Ces derniers mois, Angular et Ionic ont reçus de nombreuses nouveautés facilitant grandement la rapidité de développement:

- **Standalone component:** Des composants pouvant être créés et utilisés sans être déclarés dans un NgModule, simplifiant la construction d'applications Angular en réduisant la dépendance aux NgModules. Les composants autonomes spécifient directement leurs dépendances ce qui rend le code beaucoup plus lisible et modulaire (source: <https://angular.io/guide/standalone-components>)
- **Signals:** fonctionnalité introduite dans la version 16 d'Angular. Ils permettent de définir des valeurs réactives et d'exprimer des dépendances entre ces valeurs. Ils sont utilisés pour gérer les états, déclencher des effets, gérer les dépendances et calculer des valeurs dérivées dans une application. Ils représentent des informations qui peuvent changer au fil du temps et permettent aux parties de l'application de réagir en conséquence. (source: <https://angular.fr/signals/#pourquoi-angular-contient-desormais-la-notion-des-signaux>)
- **Ionic VS Code Extension:** Un addons très pratique pour Ionique qui permet de lancer rapidement des scripts pour développer son application ionique au lieu d'écrire les commandes manuellement.

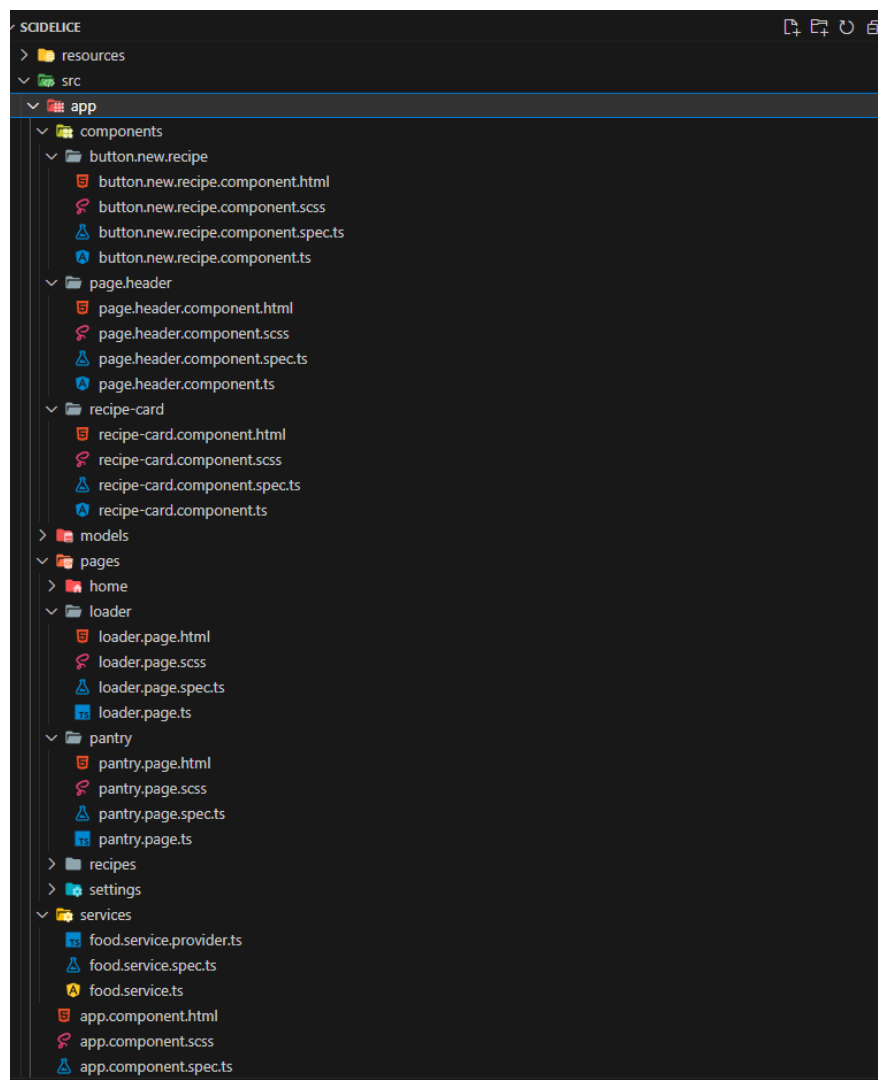
Au vu de ces développements, j'ai décidé de reprendre de zéro mon projet et de tout refaire en utilisant les composants `Standalone`, les `Signals` et l'addon pour VSCode.

Cela m'a permis d'avoir un projet beaucoup plus agréable et facile à coder, et en quelques semaines j'ai pu gagner des mois de travail. Je me suis également servi de Chatgpt et de Github copilot pour créer les textes placeholders dans mon application. Ces outils sont terriblement efficaces et je pense qu'ils deviendront indispensables aux infographistes pour gagner en productivité et efficacité.

Pour les images placeholder, je les ai générées en utilisant les intelligences artificielles DALL·E 2 de Open AI et thispersondoesnotexist.com, pour éviter les problèmes éventuels de copyright. Les contenus créés par IA ne sont pas copyrightables, mais dans mon cas ce n'est pas un souci car mon projet est open source et à des fins pédagogiques plutôt que commerciales. Pour un client, en revanche, une licence Adobe photo serait plus appropriée.

Structure du projet

Une application Angular est structurée en composants, qui sont ensuite assemblés pour créer des vues. On peut naviguer dans ces vues grâce à des routeurs. Les services permettent de passer des données d'un composant à un autre, tandis que les interfaces permettent de typer ces données de manière cohérente. J'ai mis en œuvre l'ensemble de ces concepts, en m'efforçant d'adopter une approche de développement orientée tests pour valider mes formulaires.



Aperçu de l'architecture du projet

Dans Angular, chaque composant a un template (.html), une feuille de style (.scss avec Ionic) et un fichier typescript (.ts) associés. Pour éviter d'alourdir inutilement ce rapport de TFE, tout ne sera pas expliqué en détail, car cela ferait trop d'informations. Le code sur Github est cependant agrémenté d'explications en commentaire.

Voici le détail de quelques-uns de ces composants. Les balises avec le préfix `<ion- . . >` sont des composants Ionic.

recipe-card.component

Ce composant représente une description de recette. Il peut afficher ou masquer certaines informations selon le contexte dans lequel on l'utilise.

J'utilise la projection de contenu avec `<ng-content>` pour insérer du contenu à l'intérieur de la carte. J'utilise deux sélecteurs : `[header]` et `[footer]`. Le contenu qui correspond à ces sélecteurs sera affiché respectivement avant et après le contenu principal de la carte.

Ensuite, le composant définit une section pour l'en-tête de la carte avec `<ion-card-header>`. À l'intérieur de cette section, il y a un titre `<ion-card-title>` et un sous-titre `<ion-card-subtitle>`. La visibilité de ces éléments dépend de certaines conditions vérifiées par des directives structurales `*ngIf` (par exemple, `*ngIf="showTitle"`).

Le contenu principal de la carte est placé dans `<ion-card-content>`. Il comprend une image de recette, des informations sur la difficulté, le temps de préparation, les calories et le score nutritionnel de la recette. Ces informations sont affichées à l'aide de balises `<ion-item>` contenant les icônes et labels correspondants.

Le composant gère également la possibilité d'afficher des notes sur la recette. Si des notes sont disponibles et que l'affichage des notes est activé (`showNotes` est vrai), elles seront affichées à l'intérieur d'un élément `<ion-item>`.

Il offre la possibilité d'insérer un bouton pour ouvrir la recette complète (`<ion-button>`). La visibilité de ce bouton dépend de la valeur de `isPreview`, et il déclenche une action définie par `(click)="goToRecipe()"`.

Quelques points intéressants pour le style: ionic utilise sass (Syntactically Awesome Style Sheet), qui permet entre autres d'écrire des règles CSS imbriquées les unes dans les autres.

Pour afficher exactement trois ligne de texte, j'utilise la propriété line-clamp, qui est expérimentale mais correctement supportée sur tous les navigateurs modernes :

```
-webkit-box-orient: vertical;
-webkit-line-clamp: 3;
//
https://developer.mozilla.org/fr/docs/Web/CSS/-webkit-line-clamp
display: -webkit-box;
overflow: hidden;
```

Pour le nutriscore, j'utilise les CSS container queries pour pouvoir adapter la taille des lettres relativement à la largeur du container. J'utilise des variable CSS pour rendre le code plus facile à modifier le cas échéant:

```
[nutriscore] {
  //width approx equal to 5 capital letters, if there is no more
  space, shrink down
  flex-basis: calc(5 * 3ch);
  padding: 0;
  //reduce the font size as the width of the container shrinks
  container-type: inline-size;
  display: flex;
  justify-content: center;
  font-family: arial;
  font-weight: bold;
  place-items: center;
  --radius: 0.6em;
  color: rgba(255, 255, 255, 0.5);
  --score-A-color: rgb(3, 129, 65);
  --score-B-color: rgb(133, 187, 47);
  --score-C-color: rgb(254, 203, 2);
  --score-D-color: rgb(238, 129, 0);
  --score-E-color: rgb(230, 62, 17);
}

[nutriscore] li {
  font-size: 10cqw;
  list-style: none;
  padding: 0.5em 0.5em;
}

[nutriscore] :nth-child(1) {
  background: var(--score-A-color);
  border-radius: var(--radius) 0 0 var(--radius);
}
```



```
[nutriscore] :nth-child(2) {
  background: var(--score-B-color);
}
..
}
[nutriscore] :nth-child(5) {
  background: var(--score-E-color);
  border-radius: 0 var(--radius) var(--radius) 0;
}
[nutriscore="A"] :nth-child(1),
[nutriscore="B"] :nth-child(2),
[nutriscore="C"] :nth-child(3),
[nutriscore="D"] :nth-child(4),
[nutriscore="E"] :nth-child(5) {
  border: 2pt solid white;
  border-radius: 0.5em;
  color: white;
  font-size: calc(10cqw + 3pt);
  margin: 0 -0.3em;
  padding: 0.7em 0.5em;
  z-index: 1;
}
```

Ce composant a un test unitaire (Jasmine) qui lui est associé, c'est l'approche préconisée par le *test driven development*. Il va vérifier le comportement du composant lorsqu'on clique sur le bouton "open recipe".

On utilise `describe` pour déclarer un bloc de tests, puis les variables `component`, `fixture` et `router` sont déclarées pour stocker les instances du composant, `componentFixture` et du `Router`.

On utilise la fonction `beforeEach` pour configurer chaque test, elle utilise `waitForAsync` pour s'assurer que les opérations asynchrones sont terminées avant de poursuivre.

`compileComponents` est appelée pour compiler les composants du module de test, ensuite `createComponent` crée une instance `RecipeCardComponent`, qui est ensuite assignée à la variable `fixture`.

`Inject` est utilisée pour injecter une instance du `Router` dans la variable `router`, il déclare une spécification de test. Ici le test vérifie que cliquer sur le bouton "open recipe" appelle `goToRecipe` et que le `Router` navigue vers la page de la recette correspondante. `SpyOn` est utilisée pour surveiller `navigate` et permet ainsi de vérifier plus tard si cette méthode a été appelée avec les bons paramètres, puis `goToRecipe` est appelée et `expect` est utilisée pour vérifier si `navigate` a été appelée avec les paramètres attendus suite à l'appel de la fonction `goToRecipe`, en l'occurrence `[component.recipeName]`.

FoodService

`FoodService` gère les opérations de stockage et de manipulation des données relatives aux aliments en stockage: ajout, obtention et suppression des aliments. J'utilise le service `Storage` de `@ionic/storage-angular` pour stocker les données localement, et je rend des méthodes publiques pour permettre à d'autres parties de l'application d'interagir avec ces données.

La classe `FoodService` est marquée avec `Injectable` pour la marquer comme étant injectable, c'est-à-dire qu'elle peut être injectée en tant que dépendance dans d'autres classes. Elle possède une propriété publique `allFood` qui est un tableau de type `Food` (un modèle défini ailleurs dans le code pour spécifier la forme de ce type de donnée). Cette propriété stocke la liste de tous les aliments.

Le constructeur reçoit une instance de `Storage` via l'injection de dépendances. Lorsque le service est construit, la méthode `initStorage()` est appelée pour initialiser le stockage et afficher les données stockées au démarrage. Cette méthode est une méthode privée asynchrone qui crée une instance de stockage avec `storage.create`. Ensuite, elle récupère les aliments stockés avec `storage.get('allFood')`. Si des aliments sont effectivement stockés, ils seront assignés à la propriété `allFood`.

Publication automatique sur GitHub Page avec GitHub actions

GitHub Actions est une fonctionnalité de GitHub permettant d'automatiser des tâches de développement, un peu comme un CI/CD. J'ai pu publier mon application de deux façons: sous forme d'APK, pour installer l'appli sur un téléphone android, et sous forme d'app webs, en utilisant GitHub Page et GitHub Actions.

Pour la première méthode, j'ai simplement compilé l'app, puis dans le CLI:

```
npx cap copy && npx cap sync
```

Ce qui permet de copier tous les asset web et de synchroniser les changements éventuels fait aux plugins. On ouvre le projet dans android studio avec

```
npx cap open android
```

Il ne reste plus ensuite qu'à signer l'APK:

- ouvrir le menu `Build`

- choisir `generate signed bundle / APK`
- suivre les indications pour exporter l'app signée

Pour publier l'app sous forme de PWA sur GitHub Pages, j'ai réalisé les manipulations suivantes:

Ajouter `angular-cli-ghpages` au projet:

```
ng add angular-cli-ghpages
```

A l'heure actuelle, cela donne une erreur

```
NOT SUPPORTED: keyword "id", use "$id" for schema ID:
```

Pour résoudre cette erreur, j'ai dû désinstaller le package, réinstaller avec `npm` au lieu de `ng` et enfin installer avec `ng add`.

Ensuite, j'ai utilisé ces commandes:

```
npm uninstall angular-cli-ghpages
npm i angular-cli-ghpages
ng add angular-cli-ghpages
```

On peut spécifier les paramètres de build dans `angular.json` pour ne pas devoir les indiquer dans la commande manuellement :

```
json
"deploy": {
  "builder": "angular-cli-ghpages:deploy",
  "options": {
    "baseHref": "/standalone/",
    "name": "deploy-bot",
    "email": "deploy-bot@angular"
  }
}
```

`baseHref` est nécessaire car les fichiers ne sont pas à la racine du site sur GitHub Page. Sans ça, on aurait des erreurs 404 dans la console, car les urls des divers fichiers ne pointeront pas au bon endroit.

Dans mon cas, l'url est <https://theophiledesmedt.github.io/standalone/>, il faut donc changer `baseHref` par `/standalone/`.

On commence par créer un token github avec le scope `repo`. Cela va permettre au workflow de modifier le repo. Par sécurité, on n'encode jamais les tokens dans le code publié. A la place, on crée une variable d'environnement: `GH_TOKEN`.

A noter que `GITHUB_TOKEN` est plus sécurisé que `GH_TOKEN` car limité à un seul repo, mais ne permet pas de publier des builds. Ce n'est pas vraiment un souci pour un compte github dédié à un seul projet comme le mien.

Copier le token puis aller sur le repo github de l'app et créer un secret (une variable d'environnement encrypté seulement accessible aux GitHub Actions) de nom `GH_TOKEN`, avec le token comme "value".

Il faut ensuite créer un workflow et créer un script `yaml` que j'ai du mettre à jour (original: <https://angular.schule/blog/2020-01-everything-github>)

```
main.yml
yaml
# Nom du workflow
name: Deploy to GitHub Pages via angular-cli-ghpages

# Le workflow est déclenché lorsqu'un push est effectué sur la
branche "main"
on:
  push:
    branches: [main]

# Tâches à effectuer
jobs:
  # Nom de la tâche
  build-and-deploy:
    # Système d'exploitation sur lequel la tâche est exécutée
    runs-on: ubuntu-latest

    # Étapes
    steps:
      # Clone le repo sur la machine virtuelle
      - uses: actions/checkout@v3
      # Installe Node.js 18
      - uses: actions/setup-node@v3
        with:
          node-version: 18

      # Nom de l'étape
      - name: Prepare and deploy
        # Si la branche correspond à main
        if: github.ref == 'refs/heads/main'
        # Définit les variables d'environnement
        env:
```

```
# GH_TOKEN est utilisé pour l'authentification lors du
déploiement
GH_TOKEN: ${ secrets.GH_TOKEN }
# Commandes à exécuter dans l'étape
run: |
  # Installe les dépendances du projet
  npm install
  # Déploie le projet sur GitHub Pages
  npx ng deploy
```

Conclusion

Je me suis lancé dans ce projet en sous-estimant considérablement sa complexité tant technique que organisationnelle. Cela a nécessité de nombreux sacrifices et compromis de ma part, mais m'a également apporté de nombreux apprentissages (Angular, Ionic, Git, test driven development, notions avancées d'accessibilité et d'expérience utilisateur, etc.). Je pense désormais être mieux préparé pour évaluer ce type de projet et les mener avec plus d'efficacité.

Malgré les défis rencontrés, cette expérience m'a permis de grandir professionnellement et personnellement. Je suis reconnaissant d'avoir eu l'opportunité de mener à bien ce projet et je suis confiant quant à ma capacité à aborder de nouveaux défis avec une meilleure compréhension et une plus grande efficacité, notamment chez Elia, entreprise qui m'a déjà recruté sous réserve de la validation de mon diplôme d'infographiste.

Je remercie ma famille pour son soutien inébranlable et mes professeurs pour leur écoute.

Annexes

Personnas

Michel

Chef cuisinier retraité de 65 ans, aime expérimenter avec des recettes et des techniques de cuisine différentes.

Objectifs

Il veut une application qui lui permet de stocker ses recettes personnelles et de les trier par différents critères. Il apprécie également la possibilité de partager ses recettes.

Technologie

Michel utilise un PC Windows, mais il est aussi à l'aise avec son smartphone Android.

Freins

Si l'application ne propose pas de prendre des notes sur les recettes, Michel pourrait ne pas l'adopter.

Sophie

35 ans, mère de deux enfants. Elle travaille à temps plein et aime cuisiner des repas équilibrés pour sa famille.

Objectifs

Elle a besoin d'une application qui lui permette de trouver des recettes en fonction des ingrédients qu'elle a sous la main. Elle aimerait aussi pouvoir partager ses recettes favorites avec ses amis

.

Technologie

Elle utilise principalement son smartphone pour chercher des recettes. Elle a une connaissance de base en technologie.

Freins

Si l'application est trop complexe ou si elle n'est pas capable de fonctionner hors ligne, elle pourrait ne pas l'utiliser régulièrement.

Thomas

Étudiant en informatique de 22 ans. Il est passionné par la cuisine et le bien-être.

Objectifs

Il veut une application qui lui permet de trouver facilement des recettes, des informations nutritionnelles et des astuces de cuisine. Il apprécie également le côté gamifié de l'application.

Technologie

Thomas est technophile. Il utilise un smartphone Android et a également un PC Windows pour ses études.

Freins

Si l'application n'est pas intuitive ou si les informations nutritionnelles ne sont pas précises, il pourrait être découragé.

Bibliographie et sitographie

- Ninja Squad. "Devenez un Ninja avec Angular." 2023. Consulté le 17 mai 2023. [En ligne]. Disponible sur : <https://books.ninja-squad.com/angular>.
- Angular. "Documentation officielle d'Angular." Consulté le 9 juin 2023. [En ligne]. Disponible sur : <https://angular.io/docs>.
- Ionic Framework. "Documentation d'Ionic." Consulté le 9 juin 2023. [En ligne]. Disponible sur : <https://ionicframework.com/docs>.
- Code Concept Formations Mongo Express Angular Node React Vue. "Ionic 5 par la pratique" [Cours en ligne]. Consulté le 9 juin 2023. [En ligne]. Disponible sur : <https://www.udemy.com/course/ionic-5-par-la-pratique>
- Académie Udemy. "Build Native iOS & Android as well as Progressive Web Apps with Angular, Capacitor and the Ionic Framework (Ionic 4+)." [Cours en ligne]. Consulté le 9 juin 2023. [En ligne]. Disponible sur : <https://www.udemy.com/course/ionic-2-the-practical-guide-to-building-ios-android-apps/>
- Académie Udemy. "Angular - The Complete Guide (2023 Edition)." [Cours en ligne]. Consulté le 9 juin 2023. [En ligne]. Disponible sur : <https://www.udemy.com/course/the-complete-guide-to-angular-2>