

NextStop

Rapport de projet
À l'attention de M. Grunenwald

Projet P3 - N° 204
Fleury Anthony
Semestre d'automne 2017 - 2018

ABSTRACT

Le projet P3 donne aux étudiants de troisième année d'études informatiques la possibilité d'approfondir leurs connaissances, mais également de les préparer au travail de Bachelor. Le projet P3 n'étant pas un travail de groupe, il permet d'avoir une meilleure approche des différentes phases constituant un projet lorsque le projet est réalisé en solitaire.

L'objectif de ce projet est la réalisation d'une application web respectant les principes imposés par les PWA (Progressive Web App).

Ce rapport décrit et explique le développement, les choix et les fonctionnalités de l'application *NextStop* réalisés durant le premier semestre. Il expose les réflexions et les raisonnements qui ont permis d'aboutir aux décisions finales présentées ci-dessous.

Afin d'organiser au mieux le temps à disposition, une planification des tâches a été mise en place sous la forme d'un diagramme de Gantt. Les tâches ont donc pu être réparties dans le temps en fonction de leur ampleur et criticité. Ce document reflète le travail réalisé par l'étudiant durant le projet.

TABLE DES MATIÈRES

<u>1 INTRODUCTION</u>	1
<u>2 CONCEPT</u>	2
<u>3 PLANIFICATION</u>	3
3.1 POINTS CHAUDS	3
3.2 DIAGRAMME DE GANTT	3
<u>4 CONVENTIONS DE NOMMAGE</u>	4
4.1 HTML	4
4.2 JAVASCRIPT	4
4.3 CSS	4
<u>5 PROGRESSIVE WEB APP</u>	5
<u>6 OUTILS DE TRAVAIL</u>	6
6.1 ENVIRONNEMENT DE DÉVELOPPEMENT	6
6.2 FRAMEWORKS JAVASCRIPT FRONTEND	7
6.2.1 POURQUOI UN FRAMEWORK JS ?	7
6.2.2 LES DIFFÉRENTS FRAMEWORKS JS	7
6.2.3 SÉLECTION	8
6.2.4 VUE.JS	9
6.2.5 RETOUR SUR LE CHOIX DE VUE.JS	9
6.3 NPM	10
6.3.1 VUE-CLI	10
6.3.2 NGROK	11
6.3.3 SERVE	11
6.4 WEBPACK	12
6.4.1 DEVELOPPER RUN	13
6.4.2 PRODUCTION RUN	13
6.5 MATERIAL DESIGN	13
6.6 INDEXEDDB	14
6.6.1 LIMITES DE STOCKAGE DES DONNÉES	15
6.7 CACHE API	15
6.8 TRANSPORT API	15
6.9 GOOGLE LIGHTHOUSE	17

7 ARCHITECTURE DE L'APPLICATION	18
 7.1 VUES	18
7.1.1 HOME PAGE	19
7.1.2 AJOUT DE TRAJETS	19
 7.2 MANIFESTE	20
 7.3 SERVICE WORKERS	21
7.3.1 CYCLE DE VIE GÉNÉRAL	21
7.3.2 CYCLE DE VIE AU SEIN DU PROJET	23
7.3.3 SW-PRECACHE ET SW-TOOLBOX	24
7.3.4 STRATÉGIES UTILISÉES	26
7.3.5 MISE À JOUR DES TRAJETS	28
7.3.6 NOTIFICATIONS	28
8 INTERFACE UTILISATEUR	29
9 TESTS ET VALIDATION	29
 9.1 TESTS MANUELS	29
 9.2 TESTS UTILISATEURS	30
10 PROBLÈMES ET AMÉLIORATIONS	30
 10.1 PROBLÈMES	30
 10.2 AMÉLIORATIONS	31
11 CONCLUSION	31
12 CRÉDITS	31
13 BIBLIOGRAPHIE	31
14 ANNEXES	31

1 Introduction

NextStop est une application web, respectant les critères PWA et permettant à l'utilisateur de gérer ses trajets ferroviaires CFF favoris. Une fois sélectionnés, ceux-ci seront accessibles en tout temps et ce même sans connectivité Internet. Les informations des trajets sont ensuite mises à jour selon les perturbations du réseau ferroviaire et l'utilisateur en est avisé par le biais de notifications Push.

La planification du projet et son déroulement sont exposés et analysés dans un chapitre spécifique. Il permet avant tout de se rendre compte des enjeux de la planification, en indiquant les étapes ayant été correctement évaluées et celles qui auraient mérité plus de prudence, et d'éviter à l'avenir de reproduire les mêmes erreurs.

La suite de ce document présente l'architecture de l'application, ses composants et leur utilité/interaction ainsi que les différentes fonctionnalités qui en découlent afin de bien cerner la structure du projet dans sa globalité.

S'en suit la liste des tests et des vérifications menés de manière à valider le comportement conforme de l'application. Les améliorations éventuelles réalisables et les points nécessitant encore quelques heures de travail sont également décrits dans la dernière partie de ce rapport technique.

Finalement, le document est clôturé par une conclusion générale sur le déroulement du projet dans son ensemble.

2 Concept

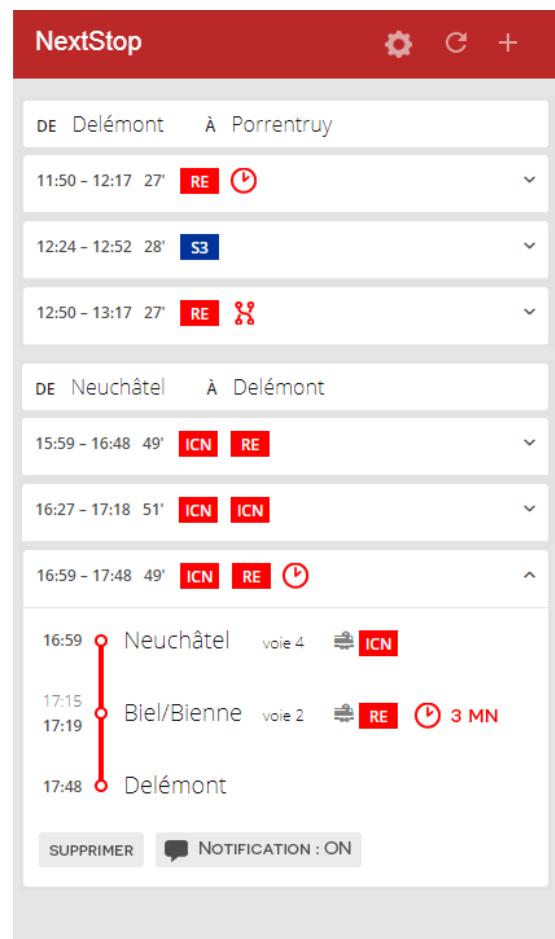
L'application web permettra aux utilisateurs de s'abonner à un certain nombre de trajets ferroviaires CFF et d'être informé grâce à des notifications push lors de perturbations (retards, changements de voie...).

Cette application fonctionnera sur tout type de périphérique (Responsive Design), mais sera avant tout prévue pour une utilisation mobile. Celle-ci sera installable et disposera de temps de réponse rapide grâce à la mise en cache d'informations ainsi qu'une optimisation en ce sens. Le système de cache permettra également de consulter les différents trajets auxquels l'utilisateur est abonné sans connexion Internet.

L'abonnement à un trajet permet de suivre son évolution (selon les informations fournies par les CFF), mais également de disposer de toutes les informations nécessaires pour voyager hors connexion. L'utilisateur aura la possibilité d'activer ou désactiver l'envoi de notifications par trajet enregistré.

Le système sera renseigné à partir du web service mis en place par search.ch donnant accès à un service de requête permettant d'obtenir les informations voulues sur les trajets désirés au format Json (via l'API Transport - <https://transport.opendata.ch/>).

Le visuel devra donner l'impression de se trouver sur une application native et d'utiliser une ergonomie permettant à l'utilisateur de se familiariser très rapidement avec les fonctionnalités proposées.



3 Planification

Avant de commencer la réalisation du projet en tant que telle, une ébauche du cahier des charges a été élaborée en listant les principales fonctionnalités que la future application se devait d'avoir. De cette liste, les spécifications ont été répertoriées en détaillant le cahier des charges. Les points chauds, ces tâches qui représentent un potentiel danger pour la fluidité du projet de par leur caractère inconnu et complexe ont ensuite été identifiés. Le cahier des charges, la liste des spécifications et la planification sont disponibles sur le Wiki du projet GitHub.

3.1 Points chauds

D'une manière générale, ce projet comportait pour moi un grand nombre d'inconnues autant au niveau des outils (bibliothèques, API, Frameworks, Webpack, ...) que des langages (JavaScript haut niveau, ...) ou des principes à utiliser (PWA, Services workers, Database locale,...).

Ceci n'a donc pas rendu la tâche facile lors de la planification, puisque je ne maîtrisais pas du tout une grande partie du projet. Je me suis donc documenté et formé au mieux de manière à éclaircir ma vision de la situation et pouvoir établir une planification « réaliste ».

3.2 Diagramme de Gantt

Une fois les spécifications décrites et les points chauds identifiés, un diagramme de Gantt a été mis en place. La planification du diagramme s'étale sur quatre aspects: la documentation, la phase de recherche et formation, le développement, et enfin les tests et de validation. L'étape du développement étant elle-même découpée en plusieurs sous-étapes permettant de scinder le travail de manière stratégique et donc déterminer des Milestones. Le diagramme de Gantt du projet dans son ensemble est présenté ci-contre :

... [Gannt]

L'évolution du diagramme de Gantt a été tenue à jour au fil de la réalisation du projet. Celle-ci a permis de garder une stabilité dans le travail fourni en ayant une vue globale de ce qui était fait et de ce qu'il restait à faire.

Les échéances des tâches ont été respectées dans l'ensemble, mais la charge de travail que constitue également les projets en parallèle de celui-ci n'a malheureusement pas permis de se conformer entièrement à la planification prévue. Les tâches en retard ont donc été réalisées durant les vacances de Noël afin de revenir le plus rapidement possible au déroulement espéré.

... [Mot sur la réussite ou non du projet]

4 Conventions de nommage

Ce projet regroupant plusieurs langages différents fait donc usage de différentes conventions selon ceux-ci. L'utilisation de l'anglais plutôt que le français a été préférée.

4.1 HTML

Les identifiants respectent la convention camelCase. Cette dernière consiste à commencer les noms par des minuscules et si le nom est une composition de plusieurs mots, une majuscule est utilisée comme séparateur.

Exemple: submitSearch

4.2 Javascript

Les variables ainsi que les méthodes utilisent la convention camelCase (Voir explications sous HTML).

Les verbes sont privilégiés lors du nommage de méthodes par souci de clarté.

Exemple: onConnectionSubmit()

4.3 CSS

L'utilisation de la convention kebab-case a été privilégiée. Celle-ci sépare les mots écrits entièrement en minuscules par des traits d'union.

Exemple: mdl-button

5 Progressive Web App

Les Progressive Web App (PWA) sont des sites Web classiques tirant partie des nouvelles fonctionnalités proposées par les navigateurs modernes de manière à combiner les meilleurs aspects des applications web et natives.

Une liste de dix critères définissant les PWAs a été établie en 2015 :

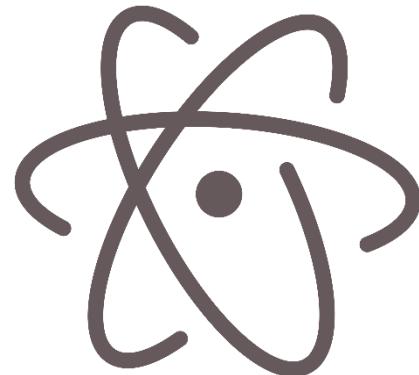
1. **Progressive** : Fonctionne pour tous les utilisateurs, quel que soit le choix du navigateur car l'application est construite avec l'idée du « progressive enhancement » en tant que principe de base.
2. **Responsive** : S'ajuster n'importe quel écran : bureau, mobile, tablette, etc.
3. **Connectivity independant** : Les service workers permettent le travail hors ligne ou sur des réseaux de faible qualité.
4. **App-like** : Ressemble à une application native avec un style et une navigation correspondante
5. **Fresh** : Toujours à jour grâce au processus de mise à jour des service workers.
6. **Safe** : Servie via HTTPS pour empêcher la surveillance et s'assurer que le contenu n'a pas été falsifié.
7. **Discoverable** : Identifiable comme des « applications » grâce aux manifestes du W3C et services workers.
8. **Re-engageable** : Facilite le réengagement de l'utilisateur grâce à des fonctionnalités telles que les notifications Push.
9. **Installable** : Permet aux utilisateurs de « conserver » les applications qu'ils trouvent les plus utiles sur leur écran d'accueil sans les problèmes des app store.
10. **Linkable** : Facilement partageable via une URL et ne nécessite pas d'installation complexe.

(Traduction de Wikipedia)

6 Outils de travail

6.1 Environnement de développement

Parmi les nombreux IDE et éditeurs de code proposés de nos jours, j'ai choisi d'utiliser pour la première fois Atom pour la réalisation de ce projet. Atom est un éditeur de code gratuit et entièrement modulable, il permet donc une grande personnalisation ainsi qu'un ajout de bon nombre d'outils très pratiques pour le développement. Une bonne configuration combinée à des modules plus qu'utiles permet à Atom de devenir un vrai IDE et donc de rivaliser avec VisualStudio ou NetBeans par exemple. Cette façon de voir les choses est vraie pour le développement Web indépendant, une grande structure (entreprise) aura cependant peut-être plus d'avantages à travailler à avec un IDE reconnu tel que VisualStudio, mais d'une façon générale, le Web laisse plus de liberté dans le choix de son environnement de développement que d'autres langages (comme Java ou C#).



Pour ma part, ayant déjà fait du développement Web avec NetBeans et Notepad++ par le passé, ce projet est donc une bonne opportunité d'évaluer Atom par moi-même et de me forger mon propre avis.

Avis final : Atom est une très bonne surprise, j'ai grandement apprécié travailler sur cet éditeur. J'ai été impressionné par ses possibilités de personnalisation, mais également par sa capacité à s'adapter à toutes les situations. Atom est définitivement un logiciel que je continuerai à utiliser à l'avenir.

Quelques packages qui m'ont été particulièrement utiles durant ce projet :

- **Atom-ternjs** : Support pour ES5, ES6, jQuery, Node.js, ...
- **Language-vue** : Support Vue
- **Vue2-autocomplete** : Support Vue2
- **Ide-typescript** : Support JavaScript
- **Minimap** : Un aperçu global du fichier en cours
- **Color-picker** : Pour choisir directement des couleurs

6.2 Frameworks JavaScript Frontend

6.2.1 Pourquoi un Framework JS ?

Une application Web monopage (Single Page Application – SPA) est un site Web ne comportant qu'une seule page. Son contenu change de façon dynamique selon les actions de l'utilisateur et ce, sans recharge de la page. Ce fonctionnement permet de fluidifier les actions de l'utilisateur et donc de donner une impression d'application.

Ce comportement est exactement celui recherché pour l'application développée dans ce projet et sera donc mis en place par divers moyens.

Un de ces moyens consiste à utiliser AJAX (Asynchronous Javascript And XML), système permettant d'effectuer des demandes au serveur Web et de recevoir en retour des informations au format JSON. Le navigateur modifie ensuite dynamiquement le contenu de la page affichée en fonction des données reçues, ceci évitant d'avoir à effectuer un recharge complet d'une page lors d'une action de l'utilisateur par exemple.

Toutefois il existe un outil regroupant bon nombre des fonctionnalités utiles pour le développement de SPA : le framework JS frontend. Bien qu'il en existe plusieurs à l'heure actuelle, tous vont permettre de se faciliter le travail lors du développement de tel type de page. N'ayant cependant aucune expérience dans l'utilisation de tels outils, j'ai donc tenté d'étudier ce que proposent ces frameworks JS et de sélectionner celui semblant le plus adapté à la situation de ce projet.

6.2.2 Les différents Frameworks JS

Après quelques recherches sur la compatibilité des PWAs et des frameworks JS, j'ai pris la décision d'utiliser un de ces fameux outils afin de découvrir une nouvelle facette du développement Web tout en étoffant le plus possible mon application.

Il est important de savoir que la réalisation d'une PWA n'oblige en rien l'utilisation d'un framework JS ([My First Progressive Web App](#) en est un bon exemple). Cependant, un tel outil, une fois maîtrisé, permet d'automatiser et d'améliorer significativement le processus de développement de ces dernières.

Comme indiqué auparavant, il existe un grand nombre de frameworks JS ce qui implique donc une sélection. Faire un choix entre ceux-ci est une manœuvre plutôt complexe, car pour ce faire il est avant tout nécessaire de bien identifier les besoins de l'application à produire de manière à sélectionner le framework proposant des fonctionnalités se rapprochant le plus de ceux-ci.

Ne bénéficiant personnellement d'aucune expérience avec ces outils et d'une connaissance minime du fonctionnement des SPAs et PWAs, je me suis ainsi renseigné et ai fait un choix sur la base des informations les plus pertinentes que j'ai pu rassemblées. Celui-ci aura impact essentiel sur le développement de ce projet et jouera un rôle primordial dans mon apprentissage de la mise en place d'une PWA ainsi que la rapidité/facilité avec laquelle je pourrai implémenter les fonctionnalités prévues. Il est donc indispensable que ce choix soit le fruit d'une réflexion mûrement menée.

Voici donc une liste des différentes possibilités s'offrant à moi et comment je les ai perçues :

Framework	Parution	Avantages	Inconvénients	Conclusion
AngularJS	2009	Largement répandu et utilisé pour les SPAs	L'un des plus lents	
Angular 2	2016			
React	2013	L'un des plus rapides,	<ul style="list-style-type: none">• Nécessite d'apprendre JSX• Pas simple à prendre en main,	Bibliothèque faites pour les grandes applications avec des données changeant souvent
Vue.js	2014	<ul style="list-style-type: none">• Facile à prendre en main• Rapide• Tutoriels vidéo sur Laracasts.com• Template PWA existant• Très bonne documentation		Mix des bonnes choses entre React et Angular
Ember.js	2011			

6.2.3 Sélection

Pour la réalisation de ce projet, j'ai pris la décision d'utiliser Vue.js au détriment des autres frameworks JS plus connus comme AngularJS ou React. Malgré les nombreux points communs partagés par ces outils, j'ai préféré me pencher sur Vue.js pour plusieurs raisons ayant leur importance à mes yeux :

- Documentation très fournie et disponible en français
- Réputé simple à prendre en main (pas de langage spécifique à connaître hormis JS) et dispose d'une courbe d'apprentissage idéale
- Framework le plus associé au terme PWA selon mes recherches
- Laracasts.com dispose d'une série de vidéos très détaillée sur l'utilisation de Vue.js
- Réputé pour regrouper les aspects positifs d'Angular et React
- Existence de templates et tutoriels associant Vue.js et PWA

N'ayant pas encore l'expérience nécessaire pour juger les différences techniques entre ces différents outils, j'ai simplement choisi celui qui m'inspirait le plus selon mes propres critères. Une fois le projet

avancé, je reviendrai sur la pertinence de ma décision et donnerai mon ressenti après une utilisation plus approfondie.

6.2.4 Vue.js

Dans le cadre de ce projet, Vue gérera entièrement le front-end de l'application puisque tout sera affiché à partir de JavaScript, mais cela tombe bien puisqu'il a été conçu en partie pour répondre à ce genre de besoins.



Vue fonctionne avec un système de composants et permet de générer des pages entières ou simplement d'insérer des composants dans des pages existantes. Dans ce projet, seul le premier cas nous intéresse, car les vues seront des composants à part entière.

Vue met à disposition un grand nombre d'éléments et d'attributs lui étant spécifiques afin de gérer dynamiquement le contenu HTML de la page générée à partir de JavaScript et des données du composant. Lorsque le composant change, la page est directement modifiée en live ce qui donne lieu à de nombreuses possibilités puisque tout est rendu dynamiquement. Voici un exemple de rendu conditionnel possible ([documentation officielle](#)) :

```
<div v-if="Math.random() > 0.5">
    Maintenant vous me voyez
</div>
<div v-else>
    Maintenant vous ne me voyez pas
</div>
```

(Exemple de vuejs.org)

Ce simple code permet selon une condition de générer deux sorties HTML différentes, à partir de ceci, il est simple d'imaginer le nombre de possibilités sachant que la condition pourrait être une variable et être modifiée de manière dynamique.

L'utilisation de la bibliothèque officielle vue-router ([documentation officielle](#)) permet de gérer le routing facilement au sein d'une application monopage comme celle-ci. C'est pourquoi cette dernière est primordiale dans ce projet.

Vue est un outil très puissant, mais ce seront là ses principales fonctionnalités au sein de ce projet.

6.2.5 Retour sur le choix de Vue.js

...

6.3 NPM

NPM est un gestionnaire de dépendances (modules/paquets) JavaScript pour Node.js, il est comparable à Composer pour PHP, mais dans une version JS. Ce qui est très utile puisque ce projet est essentiellement constitué de JavaScript.

NPM va donc permettre d'inclure directement des packages à notre en projet par le biais de commandes.

Listes des dépendances ajoutées (en plus du template utilisé) :

- Dexie : API pour IndexedDB
- Material Design Lite : Pour le visuel
- Moment.js : Gestion des dates en JS
- Vue : Vue.js core
- Vue-idb : Pour travailler avec IDB dans Vue
- Vue-router : Pour le routing des pages en mode SPA
- Vue2-autocomplete-js : Composant d'autocomplétion pour Vue

6.3.1 Vue-CLI

Vue-CLI est un module Node.js permettant comme son nom l'indique de bénéficier d'une interface en ligne de commande pour interagir avec Vue. Elle permet de démarrer des projets sur la base de templates existants. Ce qui est particulièrement intéressant puisqu'il existe un template PWA.

Grâce à la commande suivante, Vue-CLI va créer un nouveau projet sur la base du template PWA comprenant VueJS, Webpack, Vue-loader (avec hot reload), un manifeste valide ainsi qu'un service worker basique.

```
vue init pwa project_name
```

Il est ensuite possible de paramétriser quelque peu l'installation, ci-dessous un aperçu des options utilisées durant ce projet. ESLint est d'ailleurs très intéressant à activer puisqu'il va permettre de corriger en live le code produit et de vérifier qu'il soit écrit de manière uniforme.

```
C:\wamp64\www>vue init pwa project_name
? Project name project_name
? Project short name: Fewer than 12 characters to not be truncated on homescreens (default: same as name)
? Project description A Vue.js project
? Author
? Vue build standalone
? Install vue-router? Yes
? Use ESLint to lint your code? Yes
? Pick an ESLint preset Standard
? Setup unit tests with Karma + Mocha? Yes
? Setup e2e tests with Nightwatch? No
vue-cli · Generated "project_name".
To get started:
  cd project_name
  npm install
  npm run dev
Documentation can be found at https://vuejs-templates.github.io/webpack
```

Dès lors, le projet peut commencer puisque ceci a permis de mettre en place une base saine pour la suite du développement.

Il est d'ailleurs très simple de le vérifier avec Google Lighthouse une fois l'application buildée et déployée (avec serve) :



6.3.2 Ngrok

Ngrok permet de créer un tunnel à partir d'Internet vers un port local sur la machine. C'est donc un moyen simple pour mettre en ligne temporairement le site en développement et pouvoir y accéder depuis n'importe quel périphérique relié à Internet.

Dans le cadre de ce projet, Ngrok a été installé avec NPM et sert à tester l'application sur les téléphones mobiles sans difficulté.

Une fois Ngrok installé et l'application démarrée sur un port (avec « npm run dev » par exemple), il est possible d'utiliser :

```
ngrok http 8080
```

Et d'ainsi démarrer les services d'Ngrok sur le port 8080 qui fournira toutes les informations nécessaires pour l'accès depuis Internet.

```
ngrok by @inconshreveable

Session Status          online
Version                 2.2.8
Region                  United States (us)
Web Interface           http://127.0.0.1:4040
Forwarding              http://d7c553fa.ngrok.io -> localhost:8080
                        https://d7c553fa.ngrok.io -> localhost:8080

Connections             ttl     opn      rt1      rt5      p50      p90
                        0       0       0.00    0.00    0.00    0.00
```

6.3.3 Serve

Serve est un package NPM permettant de servir un site statique au travers du réseau local. Ce qui est très pratique dans le cas de ce projet puisqu'une fois notre application buildée (avec « npm run build »), un dossier avec un version statique du site est créé dans une dossier nommé « dist ».

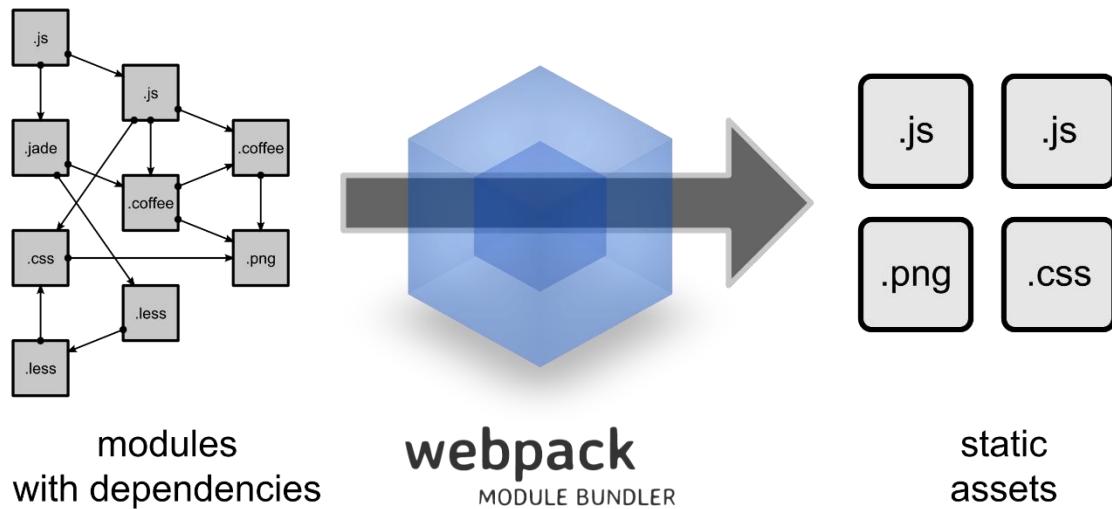
Ce qui permet de lancer la commande suivante et de pouvoir accéder localement à notre site comme il le serait en production :

```
serve dist/
```

```
C:\wamp64\www\project_name>serve dist/  
  
Serving!  
- Local: http://localhost:5000  
- On Your Network: http://192.168.178.46:5000  
  
Copied local address to clipboard!
```

6.4 Webpack

Webpack est ce qu'on appelle en anglais un « module bundler ». Il permet de prendre en entrée des modules avec des dépendances et de générer en sortie des assets statiques. Ce genre d'outil est souvent utilisé avec Vue et NPM, car il permet d'avoir une approche modulaire pour le développement d'applications.



Dans notre cas, il est directement inclus dans le template utilisé pour démarrer le projet et va permettre de construire au moment voulu notre application dans une version statique et déployable grâce à une simple commande.

6.4.1 Développer run

```
npm run dev
```

Permet de démarrer localement l'application en mode développement sur le port 8080, mais de cette manière les services workers ne fonctionneront pas, car c'est une fonctionnalité qui n'est ajoutée qu'en mode production.

6.4.2 Production run

```
npm run build
```

Permet de créer la version statique du site dans le dossier « dist » en mode production. Pour pouvoir ensuite consulter le site généré, il faut le servir en utilisant le package serve et la commande suivante :

```
serve dist/
```

Les services workers sont alors actifs et le site apparaîtra tel qu'il le serait en production.

6.5 Material Design

« Le Material Design est un ensemble de règles de design proposées par Google et qui s'appliquent à l'interface graphique des logiciels et applications¹. Il est utilisé notamment à partir de la version 5.0 du système d'exploitation Android. » *Wikipedia*

L'utilisation de Material Design permet aux utilisateurs d'être directement familiers avec l'interface de l'application et donc de s'y sentir à l'air dès les premiers instants.

N'étant toutefois que des règles de design, il existe de nombreuses bibliothèques proposant un style graphique respectant celles-ci. Il a donc fallu faire un choix :

- [Material.io](#): Nouvelle version par Google de Material Design, disponible pour Android, iOS et Web
- [Material Design Lite](#) (MDL) : Ancienne version très utilisée

J'ai décidé en début de projet de faire usage du second choix, me paraissant plus simple et plus documenté. Seulement, après avoir rencontrés à plusieurs reprises des problèmes avec celle-ci, je me suis rendu compte que MDL n'avait pas été créé dans une optique SPA alors que Material.io était lui mieux disposé à travailler dans cette idée.

Ayant toutefois réussi à surmonter les problèmes rencontrés, j'ai tout de même décidé de continuer le projet avec mon premier choix, mais dans le cas d'un nouveau projet, je n'hésiterais pas à utiliser Material.io à l'instar de MDL.

6.6 IndexedDB

IndexedDB est une API permettant d'utiliser une base de données locale directement embarquée dans le navigateur Internet.

Sa manière de fonctionner ressemble beaucoup à du NoSQL, car elle s'organise sous forme de collections d'objets JSON. C'est donc un système très différent d'une base de données relationnelle classique, il n'y pas de tables, lignes et colonnes, mais plutôt des collections contenant des ensembles clé-valeur. Il y aura donc une collection (object store) pour un type de données et les objets JavaScript qui y seront stockés. Chaque valeur peut-être une un objet et les clés des identifiants uniques ou des propriétés de ces objets. Il est aussi possible de créer des index depuis n'importe quelle propriété afin des objets de manière à faciliter les recherches.

Il est important de savoir qu'IDB travaille principalement de manière asynchrone. On ne stocke pas ou récupère pas une valeur directement depuis la base, mais il faut demander qu'une opération soit exécutée (« callback »). S'ensuit un événement DOM lorsque la requête est exécutée et le type de celui-ci permet de connaître l'état de réussite.

L'utilisation de ce genre de base de données est plus que pratique dans le cadre des applications qui ont un côté non-connecté. En effet, cet base située localement reste accessible en tout temps et ne nécessite donc pas de connexion Internet, ce qui est très arrangeant pour une PWA. C'est pourquoi j'ai décidé d'utiliser ce système afin de stocker les objets se rapportant aux trajets favoris.

La façon de travailler d'IDB est plutôt particulière car celle basée sur l'utilisation de requêtes propriétaires, ce qui ne plaît pas à tout le monde. C'est pourquoi il n'est pas rare d'utiliser une nouvelle couche (wrapper) permettant de travailler avec un élément JavaScript plus utilisé : Promise. C'est également le choix que j'ai fait en décidant d'utiliser Dexie.js qui permet d'utiliser IDB, mais de manière plus conventionnelle.

Seulement, utilisant Vue dans ce projet, et sachant que ce dernier utilise une sorte d'arbre de composants pour générer les pages, cela ne me permettrait pas d'utiliser directement Dexie. J'ai donc dû trouver une alternative ([vue-idb](#)) sous la forme d'un package NPM jouant le rôle de wrapper et permettant une utilisation de Dexie directement dans les composants Vue. Il a donc ainsi été possible de partager une instance de la base de données de manière centralisée à travers les composants.

Pour finir, il est très important de souligner le fait qu'IndexedDB reste très récent et requiert donc l'usage d'un navigateur lui-même à jour (<https://caniuse.com/#search=indexeddb>). Ce qui pourrait en un sens aller à l'encontre du progressive enhancement faisant lui-même partie des critères d'une PWA. Hélas, il n'y a pas vraiment d'autres alternatives pour moi dans cette situation.

Remarque : Il existe un autre moyen de stockage local appelé localStorage (DOM Storage), mais ce dernier est plutôt destiné à travailler avec des très petites quantités de données et fonctionne de manière synchrone, il est donc bloquant pour l'utilisateur, ce qui est n'est pas pratique.

6.6.1 Limites de stockage des données

Les données stockées avec IndexedDB sont dites persistantes, elles se doivent donc d'être conservées durant une longue période.

Toutefois ce genre de stockage en local côté client implique forcément une limite. Cette limite est calculée de manière dynamique selon la taille du disque dur. Firefox et Google Chrome utilisent la même politique, une fois un certain pourcentage du disque dur rempli, le navigateur commence à faire de la place en supprimant les données locales les moins utilisées. Dans le cas de ce projet, cela peut être problématique, puisque cela revient à supprimer l'entièreté des trajets favoris de l'utilisateur. Une solution serait la mise en place d'un système de sauvegarde externe (Firebase par exemple) qui permettrait en cas d'effacement des données de recharger celles-ci dans un état cohérent. Hélas par manque de temps ce système ne pourra pas être mis en place et il faudra se contenter de données susceptibles à la suppression par le navigateur.

Pour les détails concernant les politiques des navigateurs concernant leur espace de stockage, je recommande de consulter ce lien expliquant le processus en détails: [Limites de stockage du navigateur et critères d'éviction - Mozilla](#)

6.7 Cache API

L'interface Cache de l'API ServiceWorker, elle permet de stocker les paires d'objet Request / Response mises en cache depuis un service worker lors de son cycle de vie.

Elle est utilisée dans ce projet de manière à stocker les ressources statiques et dynamiques de l'application comme les fichiers CSS ou les fonts.

Le cache partage le même espace que celui utilisé par IndexedDB ou LocalStorage, il faut donc faire attention à ne pas en abuser.

6.8 Transport API

Après plusieurs recherches, j'ai pris la décision d'utiliser une API non officielle appelée [Transport API](#) permettant de récupérer toutes les informations relatives aux horaires CFF. Cette API utilise elle-même un web service mis à disposition par [search.ch](#). Transport API fonctionne à l'aide du verbe GET et d'une URL (avec paramètres) et retourne une réponse au format JSON.

J'ai fait le choix de travailler avec cette API, car ses réponses sont selon moi plus simples à travailler que celles originellement créées par search.ch. Toutes les informations sur les trajets sont accessibles facilement à l'aide de propriétés intelligemment nommées et documentées.

Search.ch indique une limite maximum à 1000 requêtes journalières, il reste à définir si cette limite est imposée par domaine ou par l'adresse IP de l'utilisateur.

La requête est entièrement paramétrable et permet de récupérer les résultats précisément souhaités.

Exemple de requête:

```
GET http://transport.opendata.ch/v1/connections?from=Lausanne&to=Genève
```

Exemple de réponse:

```
{
  "connections" : [
    {
      "from" : {
        "arrival" : null,
        "departure" : "2012-03-31T08:58:00+02:00",
        "platform" : "7",
        "prognosis" : {
          "platform" : null,
          "arrival" : null
          "departure" : null
          "capacity1st" : "-1",
          "capacity2nd" : "-1",
        },
        "station" : {
          "coordinate" : {
            "type" : "WGS84",
            "x" : "6629086",
            "y" : "46516785"
          },
          "id" : "008501120",
          "name" : "Lausanne",
          "score" : null
        }
      },
      "to" : {
        "arrival" : "2012-03-31T09:46:00+02:00",
        "departure" : null,
        "platform" : "2",
        "prognosis" : {
          "platform" : null,
          "arrival" : null,
          "departure" : null
          "capacity1st" : null,
          "capacity2nd" : null,
        },
        "station" : {
          "coordinate" : {
            "type" : "WGS84",
            "x" : "6142437",
            "y" : "46210217"
          },
          "id" : "008501008",
          "name" : "Genève",
          "score" : null
        }
      }
    }
}
```

```
    },  
    ],  
    // ...  
}
```

6.9 Google Lighthouse

Google Lighthouse est un outil de test remarquable permettant d'exécuter une grande quantité de tests sur une page web de manière totalement automatisée. Un rapport est ensuite généré indiquant les résultats obtenus dans différentes catégories et ce qu'il faudra faire pour corriger les problèmes rencontrés.

Cet outil m'a permis tout au long du développement de tester mon application et d'avoir un retour très rapide et détaillé. J'ai ainsi réalisé des tests avec Google Lighthouse aux différentes étapes clés du projet de manière à un suivi de l'évolution. Ces rapports sont aujourd'hui enregistrés et sont disponibles sur le Wiki du projet.

Lighthouse m'a permis de vérifier que le départ que j'avais choisi était le bon puisque j'ai pu directement jauger mon template de départ et ainsi m'assurer que la base de départ était saine.

Voici le rapport généré après quelques heures de travail sur le template de départ :



Les performances ont été quelque peu réduites dû à l'incorporation de styles, bibliothèques, etc, mais les scores restent tout de même très bons.

6.10 Chrome DevTools

Les outils nativement proposés Google Chrome se révèlent très utiles lorsque l'on sait les utiliser. En effet, le panneau « Application » faisant partie de ses outils m'a dépanné à de multiples reprises lorsque je travaillais avec les service worker ou encore IndexedDB. Ce panneau permet de consulter le manifeste et toutes ses options, les services workers associés au domaine et leur état, tous les types de stockages locaux (IndexedDB et le cache y compris), mais également d'intervenir directement de différentes manières sur tous ces différents aspects de l'application. Il est même possible désactiver virtuellement l'accès réseau à un service worker pour étudier son comportement.

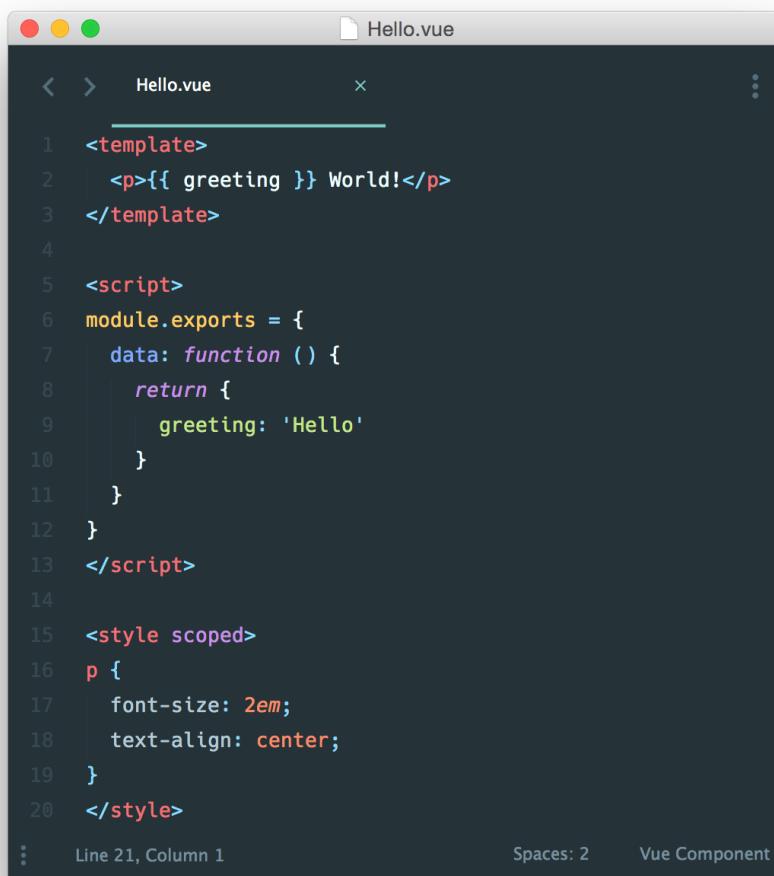
7 Architecture de l'application

L'architecture logicielle détaille les choix qui ont été pris durant la phase de développement du projet, mais également les différents composants implémentés et comment ces derniers interagissent entre eux.

7.1 Vues

Comme indiqué précédemment, Vue est basé sur l'utilisation de composants, c'est pourquoi dans cette application les vues en elles-mêmes sont des composants. Dans le cas de projets où le front-end entièrement généré par JavaScript, il est plus agréable d'utiliser ce qu'on appelle des composants monofichiers (voir [documentation officielle](#)). Ces fichiers portant l'extension « .vue » sont possibles grâce à l'utilisation de Webpack.

Voici un exemple simple d'une composant :



```
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

Line 21, Column 1 Spaces: 2 Vue Component

On peut voir qu'il y a tout d'abord une partie template qui accueillera la partie HTML, puis une partie script pour les déclarations Vue et enfin une dernière partie pour les styles qui seront propres à ce composant.

Ce format permet donc de travailler sous forme de modules, ce qui est une bonne chose dans le cas de grands projets ou simplement pour diminuer le couplage entre les composants.

7.1.1 Home page

La page d'accueil est l'endroit où sont affichés les trajets favoris de l'utilisateur. Ceux-ci sont disposés sous forme d'accordéon et sont ordonnés de manière à faciliter la vie de l'utilisateur : par tuple ville de départ-arrivée, puis par horaire de manière croissante. Lors du clic sur un trajet, celui-ci se déplie et affiche toutes les informations le concernant comme les changements de trains ou les horaires. Toutes ces informations étant stockées en local grâce à IndexedDB, celles-ci restent consultables sans connexion Internet ou avec un réseau instable. Cela revient donc pour l'utilisateur à disposer en quelques sorte d'un agenda hors ligne de ses trajets préférés. Il est bien entendu possible de supprimer un trajet ne paraissant plus intéressant.

La seconde fonctionnalité de cette page est d'actualiser ces trajets en rafraîchissant les informations stockées depuis l'API Transport. Les perturbations sur les trajets peuvent ainsi être indiquées directement à côté de ceux-ci et être signalées à l'utilisateur par le biais de notifications Push.

7.1.2 Ajout de trajets

La page d'ajout de nouveaux trajets est très simple, elle comme son nom l'indique d'ajouter un nouveau trajet favori à la liste de l'utilisateur. Pour ce faire, il suffit de remplir les champs correspondant aux villes de départ/arrivée ainsi que l'heure de départ. Une autocomplétion est présente sur les champs des locations permettant une saisie agréable à l'utilisateur. La liste des propositions est renouvelée lors de l'entrée de chaque nouveau caractère dans les champs. Les propositions proviennent directement de la base de données CFF et sont les mêmes que celle que l'on retrouve sur la recherche officielle du site CFF. Ce service transitant par Internet nécessite donc une connexion pour fonctionner, mais comme l'ajout n'est de toute manière disponible que lorsque le réseau est présent, cela n'est en réalité pas un problème.

Une fois les informations rentrées et validées, la page va effectuer une requête vers l'API Transport avec les informations indiquées et récupérer une réponse au format JSON. Celle va être traitée et affichée à l'utilisateur sous la forme d'un tableau indiquant pour chaque ligne un trajet-horaire avec une case à cocher en guise de sélection. L'utilisateur est donc libre de choisir un ou plusieurs trajets parmi ceux-là étant proposés avant de valider son choix. Les trajets cochés sont évidemment enregistrés localement grâce à IndexedDB et l'utilisateur est ensuite simplement redirigé vers la page d'accueil de l'application où ses nouveaux trajets favoris ont été triés et ajoutés à l'accordéon.

7.2 Manifeste

Le manifeste d'une application web est un document au format JSON regroupant des informations concernant celle-ci telles que son nom, sa description ou ses icônes. Le manifeste sert à signaler le site web comme PWA, mais surtout à le rendre installable sur l'écran d'accueil de l'utilisateur. C'est également grâce à l'option "display": "standalone" qu'une fois l'application ouverte à partir de l'icône présente sur l'accueil, le navigateur sera masqué et donne alors une impression d'application native. L'installation permet aussi l'envoie de notifications Push sur l'appareil (fonctionnalité pas encore disponible partout: [CanIUse – Compatibilité Web Notifications](#), voir également [Push API](#))

Exemple du manifeste de ce projet:

```
{  
  "name": "NextStop",  
  "short_name": "NextStop",  
  "icons": [  
    {  
      "src": "/static/img/icons/android-chrome-192x192.png",  
      "sizes": "192x192",  
      "type": "image/png"  
    },  
    {  
      "src": "/static/img/icons/android-chrome-512x512.png",  
      "sizes": "512x512",  
      "type": "image/png"  
    }  
  ],  
  "start_url": "/index.html",  
  "display": "standalone",  
  "background_color": "#000000",  
  "theme_color": "#bb2828"  
}
```

Afin d'indiquer l'utilisation d'un manifeste, il faut le lier en utilisant une balise link:

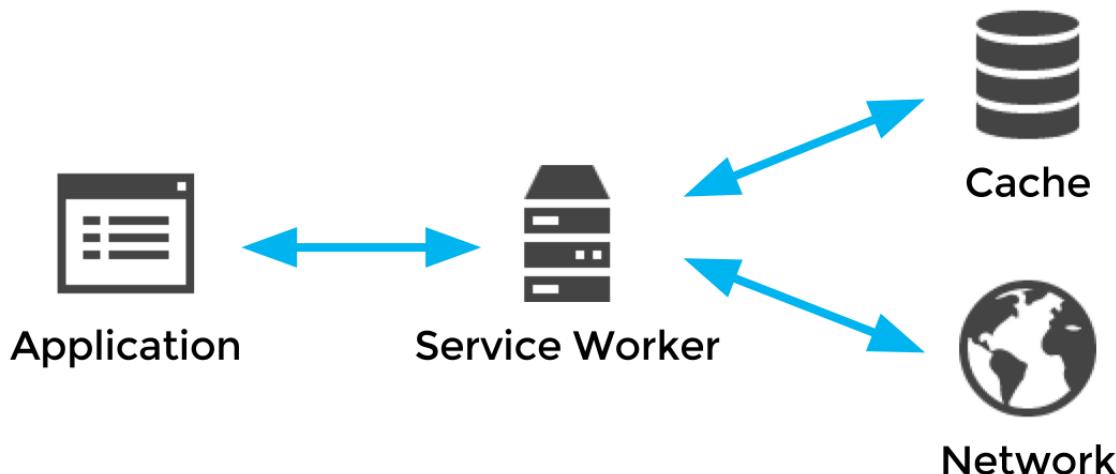
```
<link rel=manifest href=/static/manifest.json>
```

Se référer à la page web suivante pour plus de détails concernant les différents paramètres : [Manifeste des application web - Mozilla](#)

7.3 Service workers

Les service workers jouent essentiellement le rôle de serveurs proxy placés entre une application web, et le navigateur ou le réseau (lorsque disponible.) Ils sont destinés (entre autres choses) à permettre la création d'expériences de navigation déconnectée efficaces, en interceptant les requêtes réseau et en effectuant des actions appropriées selon que le réseau est disponible et que des ressources mises à jour sont à disposition sur le serveur. Ils permettront aussi d'accéder aux APIs de notifications du serveur (push) et de synchronisation en arrière-plan. (Mozilla.org)

Les service workers ont donc un rôle très important dans le cadre des Progressive Web App. Ce sont eux qui vont orchestrer, intercepter et diriger les requêtes réseau afin d'aller chercher les informations aux endroits adéquats à la situation (dans le cache ou à la source par exemple). De part ce fait, ce sont également eux qui vont donner la possibilité d'agir selon l'état du réseau et d'aborder la situation selon un plan défini.



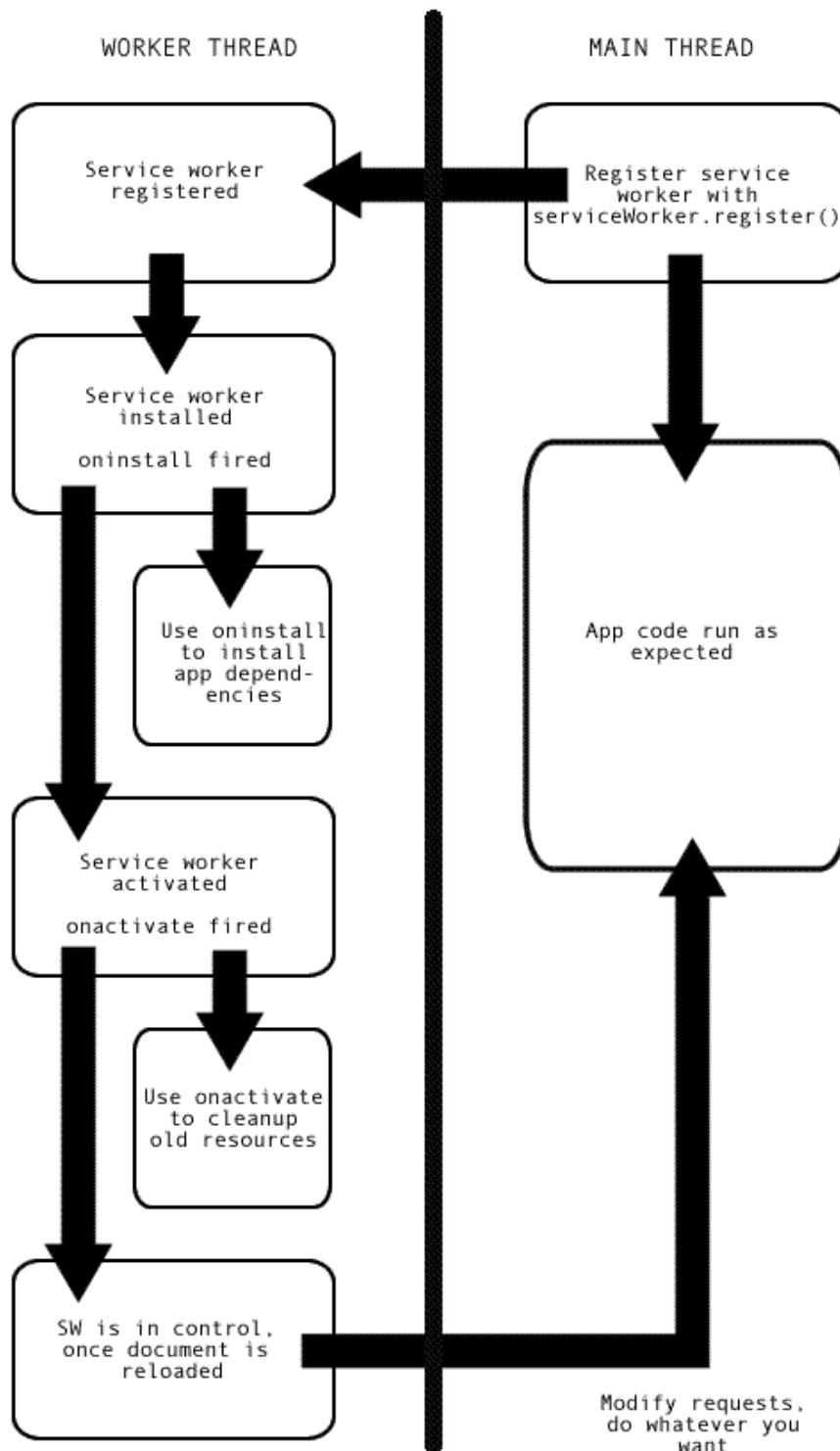
Un service worker est un fichier JavaScript contenant un worker évènementiel qui sera enregistré auprès d'une page (la racine du site dans notre cas). Il sera ainsi installé et activé aux utilisateurs consultant cette page ou ses enfants et pourra contrôler leurs requêtes réseau.

Comme les service workers sont écrits en JavaScripts (modifiables) et qu'ils ont un énorme pouvoir sur les requêtes réseau, pour des raisons de sécurité évidentes, ceux-ci ne fonctionnent qu'en HTTPS. C'est une des raisons principales pour lesquelles une PWA se doit de fonctionner en HTTPS.

7.3.1 Cycle de vie général

Les services workers ont un cycle de vie très spécifique qu'il est nécessaire de comprendre afin de maîtriser leur comportement dans toutes les situations.

Un service worker étant un worker ne dispose de son propre cycle de vie séparé de celui de l'application elle-même.



Premièrement, un service worker doit être enregistré, cette étape se fait au sein du code JavaScript de l'application. Lors de l'enregistrement une certaine portée (scope) lui est définie, c'est dans cette portée qui pourra intervenir au niveau des requêtes.

Lorsqu'un utilisateur se rendra pour la première fois sur la page (dans la portée), le service worker sera téléchargé.

Le service worker va alors débuter sa procédure d'installation (event : install) et s'activer (event : activate) directement s'il ne détecte pas d'ancienne version de lui-même en fonction, sinon il passera dans un état d'attente jusqu'à ce qu'il lui soit dit spécifiquement de continuer (skipWaiting).

C'est lors de l'événement d'installation que l'on va en général remplir le cache avec les assets statiques et lors que l'activation que l'on va profiter de nettoyer les anciennes données contenues dans le cache.

Une fois activé, le nouveau service worker est prêt à contrôler les requêtes réseau et se déclencher sur l'événement fetch par exemple.

Pour plus d'informations: [Service Worker API - Mozilla](#)

7.3.2 Cycle de vie au sein du projet

Toute la partie liée à l'enregistrement et aux états de vie du service worker s'effectue sur l'événement `window.load` dans un fichier JS nommé `service-worker-prod.js`.

Le service worker en lui-même est généré par SW-Precache (voir plus loin) selon les configurations indiquées.

L'enregistrement s'effectue seulement si le navigateur est capable de gérer les service workers et que l'on se trouve en HTTPS (ou localhost). Le service worker est alors enregistré avec la racine comme portée.

A partir de ce point plusieurs scénarios sont probables :

- L'utilisateur se rend pour la première fois sur l'application et qu'aucun service worker n'est activé
- Un service worker est déjà en cours de fonctionnement

Dans le premier cas, l'installation ainsi que l'activation du service worker sont lancés et l'événement `onupdatefound` est déclenché. Un message (snackbar) indique alors à l'utilisateur que le contenu est désormais disponible hors ligne, car le travail de mise en cache aura à ce moment été effectué du côté du service worker.

Dans le second cas, un autre service worker est déjà en cours d'utilisation, mais le nouveau s'installera tout de même et déclenchera un événement `onstatechange` sur ce dernier. Cet événement affichera un message à l'utilisateur lui indiquant qu'une nouvelle version du site est disponible, il devra alors cliquer sur le bouton « refresh » proposé par le message qui activera le nouveau service worker et remplacera l'ancien.

Si l'utilisateur ne voit pas le message ou ne presse pas le bouton « refresh » avant sa disparition, le nouveau service worker restera en état d'attente et le message reviendra lors du prochain rechargeement de la page.

7.3.2.1 Message en ligne/hors ligne

Dans ce même fichier deux autres événements sont gérés :

- Online
- Offline

Ils permettent de simplement d'afficher un message à l'utilisateur leur indiquant s'il leur état de connectivité Internet vient de changer.

7.3.3 SW-Precache et SW-Toolbox

Il faut savoir qu'il est tout à fait possible d'écrire un service worker entièrement à la main, seulement cela requiert de bonnes connaissances en JavaScript et il existe aujourd'hui des méthodes plus simples pour au mieux partir d'une bonne base.

SW-Precache est une bibliothèque permettant de générer des service worker pour la mise en cache d'assets statiques (App Shell utilisant la stratégie cache first).

SW-Toolbox, lui aussi une bibliothèque, fonctionne souvent de pair avec SW-Precache et permet cette fois-ci la mise en cache de contenu dynamique comme les images ou toutes les autres ressources nécessaires au fonctionnement de l'application.

Ces bibliothèques sont toutes deux préinstallées dans le template utilisé lors de ce projet et fonctionne donc avec l'installation Webpack existante.

Il faut savoir qu'il existe une autre bibliothèque reconnue permettant le développement de services worker. Cette dernière se nomme [Workbox](#) et est développée par Google. Après m'être renseigné, j'ai tout de même décidé de poursuivre le projet avec les outils proposés et installés nativement dans mon template. Toutefois, Workbox à l'air d'être un très bon outil et permet tout autant de chose si ce n'est plus que le combo SW-Precache + SW+Toolbox.

Comme indiqué auparavant, les service worker ne seront générés qu'en mode production. Il est donc nécessaire de build le projet afin de voir le fichier `service-worker.js` apparaître dans le dossier `dist/` contenant la version finale et déployable de l'application.

Les détails et options concernant le build sont paramétrables dans le fichier `build/webpack.prod.conf..js` sous la partie `SWPrecacheWebpackPlugin`:

```
// service worker caching
new SWPrecacheWebpackPlugin({
  cacheId: 'my-vue-app',
  filename: 'service-worker.js',
  staticFileGlobs: ['dist/**/*.{js,html,css}'],
  minify: false,
  stripPrefix: 'dist/',
  skipWaiting: false,
  importScripts:['static/js/service-worker-import.js'],
  handleFetch: true,
  runtimeCaching: [
    {
      urlPattern: /^https:\/\/fonts.googleapis\.com$/,
      handler: 'cacheFirst'
    },
    {
      urlPattern: /^https:\/\/fonts\.gstatic\.com$/,
      handler: 'cacheFirst'
    },
    {
      urlPattern: /^https:\/\/code\.getmdl\.io$/,
      handler: 'cacheFirst'
    }
  ]
})
```

C'est ici que le nom du cache (`cacheId`) utilisé va être indiqué ou encore le nom du service-worker (`filename`) qui sera créé.

La ligne `staticFileGlobs` est très importante puisque c'est elle qui va indiquer quels fichiers feront partie des assets statiques de l'application. Ici le paramètre définit que tous les fichiers JavaScript, HTML et CSS seront mis en cache peu importe où ils sont contenus dans le dossier `dist/`.

`ImportScripts` permet d'indiquer un fichier à concaténer à la fin du fichier service worker généré. Je l'utilise ici afin de pouvoir faire réagir mon service worker à l'événement personnalisé `skipWaiting` et passer le relai d'un ancien service worker vers un nouveau.

`HandleFetch` donne la possibilité de choisir si l'on souhaite que l'événement `fetch` soit traité directement dans le code généré ou si l'on souhaite s'en occuper nous-même (dans un fichier importé par exemple).

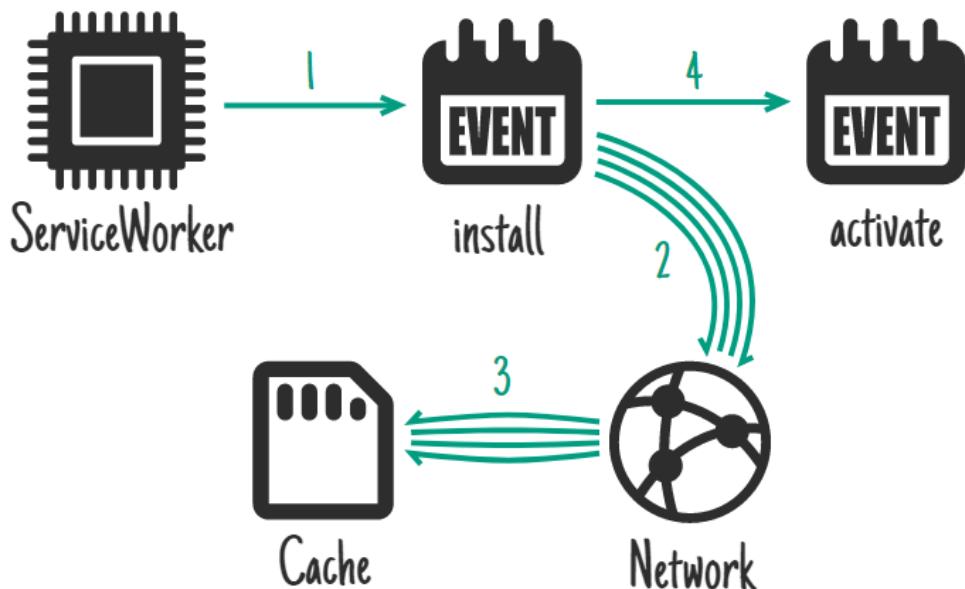
Le tableau affecté dans l'option `runtimeCaching` contient les ressources dynamiques qui seront traitées par SW-Toolbox et qui seront mises en cache. Il est obligatoire d'indiquer la stratégie de cache à leur appliquer. La méthode `cache first` essaie en priorité de servir les ressources depuis le cache avant d'essayer sur Internet si celles-ci ne s'y trouvent pas. Cette même stratégie est également appliquée à l'App Shell qui doit toujours être configuré de manière `cache first` ou `cache only`.

Pour plus d'informations sur les options de SW-Precache et SW-Toolbox : [SW-Precache GitHub](#)

7.3.4 Stratégies utilisées

Cette section vise à expliquer ce qu'il se passe concrètement au niveau du service worker par rapport aux ressources lors de différents événements et situations.

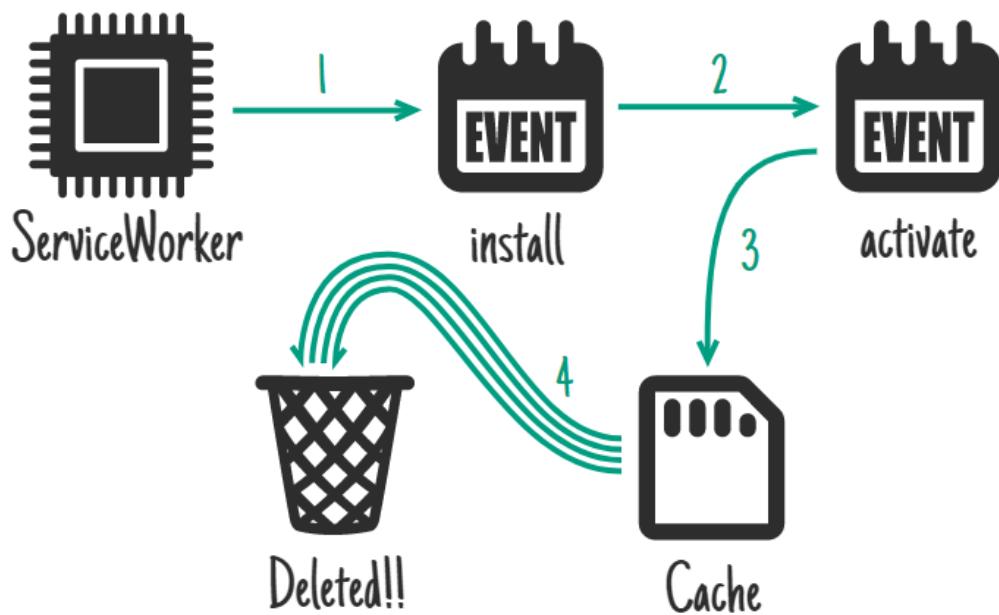
7.3.4.1 Installation du service worker



Ci-dessus est présentée la stratégie utilisée lorsqu'un nouveau service worker vient à s'installer, dans cas d'une modification de l'application par exemple. Le service worker, lors de son installation va simplement aller chercher (`fetch`) les ressources nécessaires sur Internet et les ajouter au cache.

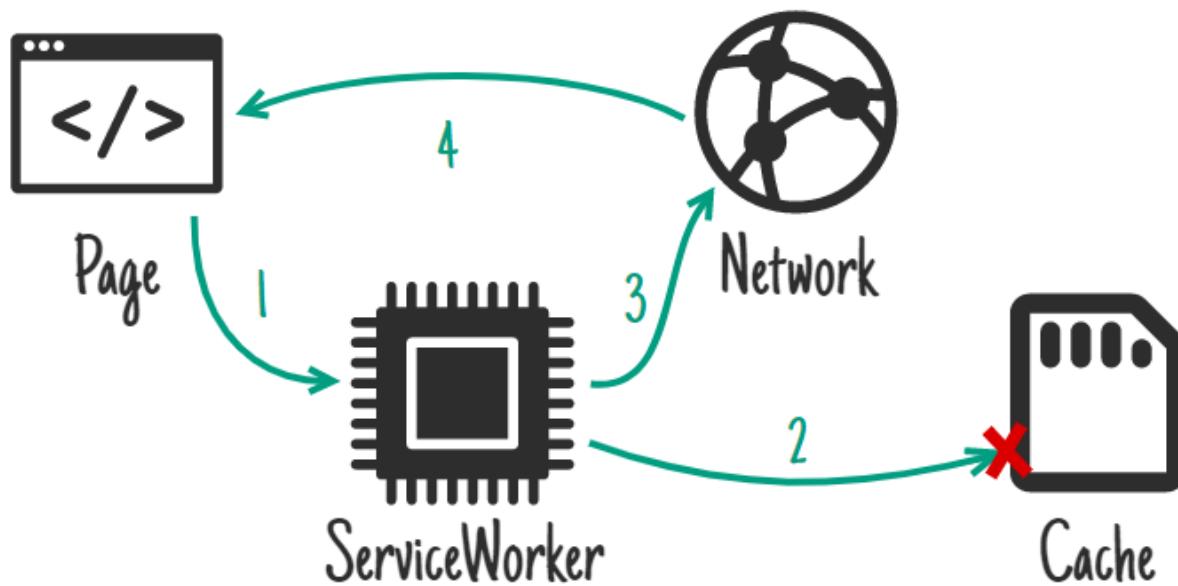
Il se produit la même chose pour les ressources dynamiques auxquelles la stratégie `cache first` a été affectée.

7.3.4.2 Activation d'un service worker



Cette stratégie est utilisée lorsqu'un nouveau service worker prend le relais et doit nettoyer le cache présent avant de continuer son travail. Une fois l'installation effectuée, les nouvelles ressources sont ajoutées dans le cache, mais il faut également supprimer celle que ne sont plus utilisées. Lors que le nouveau service worker est activé, il supprime les ressources non utilisées en effectuant une comparaison.

7.3.4.3 Cas général - Cache, falling back to network



Le service worker regarde en priorité si les ressources désirées sont présentes dans le cache, si c'est le cas il les retourne simplement. Sinon, il va les chercher sur Internet et les retourne. Cette approche peut être considérée comme Cache Only pour les ressources qui seront forcément présente dans le cache comme les assets statiques ou Network only pour les ressources qui n'y seront jamais ajoutées comme informations requises pour l'autocomplétion des destinations par exemple.

Les ressources dynamiques comme les fonts ou les glyphicons utilisent quant à elles une légère variante de la stratégie présentée ci-dessus en mettant simplement à jour le cache une fois les ressources trouvées sur Internet. Cette stratégie peut être vue comme Cache First.

Pour plus de détails sur les différentes stratégies existantes : [The offline cookbook – By Jake Archibald](#)

7.3.5 Mise à jour des trajets

7.3.6 Notifications

8 Interface utilisateur

L'interface utilisateur se veut très simple et agréable d'usage, c'est pourquoi la décision d'utiliser Material Design a été prise. En effet, Material Design est devenu un standard en ce qui concerne l'UI des applications, de par son élégance, sa sobriété et sa clarté d'utilisation. Pourquoi faudrait-il retravailler ce que des designers spécialisés ont mis du temps à élaborer avec soins ?

En ce qui concerne l'affichage des trajets l'idée était de rester assez proche du style et du fonctionnement de l'horaire CFF. Les utilisateurs cibles de l'applications sont probablement des habitués de l'horaire en ligne et connaissent son fonctionnement, il serait malvenu de changer leurs habitudes. Le système d'accordéon ainsi que le placement des informations sont donc proches visuellement parlant de ce que l'on peut trouver sur l'horaire officiel.

9 Tests et validation

9.1 Tests manuels

9.1.1 Responsive Design / Multi-plateforme

- Les pages du site s'adaptent bien sur mobile / tablette / PC
- L'application fonctionne sur tous les navigateurs récents

9.1.2 Navigation

- La navigation se fait de manière rapide
- Les boutons de navigations renvoient à leur page respective
- Le logo du site renvoie sur la page d'accueil

9.1.3 Service Worker

- La création de service worker se fait sans erreur
- Un nouveau service worker s'installe, s'active et prend le contrôle correctement
- Le service worker enregistre dans le cache toutes les assets statiques correctement
- Les ressources dynamiques sont intégrées au cache après le premier chargement de page
- Les ressources dynamiques et statiques sont servies correctement en mode hors ligne
- Les Snackbars avertissant le passage en ligne et hors ligne apparaissent
- La Snackbars avertissant d'un nouveau contenu apparaît
- Un nouveau service worker est mis en attente jusqu'à sa confirmation par la Snackbar
- Une fois le bouton « refresh » pressé sur la Snackbar la page est rafraîchit et le nouveau service worker prend le relai
- Un nouveau service worker ajoute les nouvelles ressources au cache et retire celles inutilisées
- Les ressources non présentent dans le cache utilisent la stratégie Network Only
- Les ressources toujours présentent dans le cache utilisent la stratégie Cache Only
- Les ressources dynamiques utilisent la stratégie Cache First

9.1.4 Recherche / Ajout de trajet

- L'autocomplétion fonctionne lors de l'ajout d'un caractère dans une des champs de localité
- La liste de proposition est fidèle aux caractères inscrits
- La saisie de l'heure est validée
- La touche entrée dans le champs heure de départ valide le formulaire
- La requête est correctement formulée avec les bons champs vers transport.ch
- La validation du formulaire affiche le tableau des résultats
- Le tableau de résultat contient des données pertinentes
- Le calcul de la durée du trajet est pertinent
- Les cases à cocher se comportent normalement
- La validation du formulaire du tableau entraîne la redirection vers l'accueil
- Après validation des choix, les trajets sélectionnés sont ajoutés dans IDB

9.1.5 Trajets favoris

- La liste des trajets favoris de l'utilisateur est chargée depuis IDB
- Les trajets sont triés correctement d'abords par paire localité départ-arrivée puis par heure de départ
- Chaque paire localité départ-arrivée dispose de son bloc d'accordéon
- Chaque heure de départ dispose de son accordéon
- Plusieurs accordéons peuvent être ouverts en même temps
- Les informations sont disposées de manière correcte une fois l'accordéon ouvert
- Il est possible de supprimer un trajet favori, il disparaît et est également supprimé de IDB

9.1.6 Mise à jour des trajets favoris

9.2 Tests utilisateurs

De manière à percevoir un avis général quant à l'application, il a été demandé à 6 personnes différentes de tester l'application sur mobile durant quelques instants.

Voici la liste des différentes constatations évoquées par les utilisateurs :

-
-

10 Problèmes et améliorations

10.1 Problèmes

10.2 Améliorations

11 Conclusion

12 Crédits

13 Bibliographie

14 Annexes
