Haute Ecole de Gestion & Tourisme
Hochschule für Wirtschaft & Tourismus

Exécution côté client d'une application de Machine Learning avec WebAssembly

Travail de Bachelor 2021

Rendu le 12 août 2021

Réalisé par Yevheniy Zaychenko

Professeur responsable : Henning Müller

HES-SO Valais/Wallis

www.hevs.ch

Filière : Informatique de gestion

Résumé

De nos jours, la performance au niveau des vitesses d'exécution est un facteur important dans le milieu informatique. Les applications nécessitant d'importants volumes de calculs se heurtent à de multiples difficultés à ce niveau. Pour remédier à cela, il est concevable de réduire l'utilisation des ressources côté serveur pour en augmenter celles du côté client. Afin d'y parvenir, il est nécessaire d'opérer certains calculs directement dans le navigateur de l'utilisateur.

Pour réussir ce défi, plusieurs étapes doivent être achevées dans ce travail.

Premièrement, il est impératif d'analyser convenablement la technologie permettant une telle exécution. Dans notre cas, c'est le nouveau standard pour du code binaire, se nommant WebAssembly. Pour ce faire, des recherches doivent être menées pour le situer sur le marché actuel. Les éléments fondamentaux de l'état de l'art sont la détermination de son fonctionnement, ses atouts, ses solutions alternatives ainsi que plusieurs autres facteurs.

Ensuite, pour mettre en pratique ces recherches, un prototype d'application est développé avec cette nouvelle technologie. Celui-ci se doit d'être élémentaire car l'objectif est de vérifier la faisabilité d'une telle solution et d'y voir les difficultés.

La prochaine phase de ce projet est d'inspecter les résultats au niveau de la vitesse d'exécution entre une application avec la nouvelle technologie implémentée et celle avec une technologie plus traditionnelle. Nous procédons également à une comparaison avec les mêmes sujets mais sur des critères différents comme leur structure d'application. De plus, une synthèse des différents problèmes rencontrés lors de la phase de développement est détaillée. Dans laquelle, nous présentons la principale difficulté qui nous a contraint à adapter le concept de l'application. À la suite de quoi, notre prototype a perdu sa fonctionnalité principale. Celle de s'exécuter entièrement sur la machine du client.

Avant de conclure le document avec les potentielles améliorations, nous formulons un bilan général de ce travail de Bachelor.

Mots-clés : WebAssembly – Blazor – Machine Learning – Classification d'images

Avant-propos

Ce travail clôture une formation à la HES-SO Valais pour l'obtention d'un Bachelor en informatique de Gestion. Il est réalisé durant le dernier semestre d'un cursus de quatre ans à temps partiel. L'objectif de ce dernier est d'évaluer les compétences acquises par les futurs diplômés durant leur formation.

Le sujet de ce travail a été soumis par le Dr. Prof. Henning Müller en collaboration avec un institut de recherche en informatique de la HES-SO Valais. La mission est de découvrir et de prendre connaissance d'une nouvelle technologie en se basant sur différentes recherches et en fournissant notre propre analyse complémentaire. L'institut de recherche pourra décider s'il souhaite travailler et implémenter cette nouvelle solution dans leurs futurs projets. Pour cela, le prototype d'une application doit être développé et un rapport doit être formulé pour illustrer les résultats obtenus.

Ce rapport est rédigé sur la base de la norme de mise en forme APA 6 (American Psychological Association).

Afin de faciliter la lecture de ce document, la forme masculine sera la seule employée dans les textes.

Remerciements

Je tiens à remercier chaleureusement toutes les personnes qui m'ont aidé et soutenu durant ma formation ainsi que lors de la réalisation de ce travail de Bachelor.

Des remerciements particuliers à M. Henning Müller, le professeur responsable de ce projet, pour m'avoir aiguillé et accompagné dans les meilleures conditions. M. Ivan Eggel pour son expérience et ses conseils sur le plan technique de ce travail. Tous deux ont fait preuve d'une grande disponibilité et implication.

Un grand merci également à M. Alain Duc pour m'avoir partagé ses connaissances sur le langage de programmation et l'API choisi dans ce projet.

Pour finir, je remercie toutes les personnes qui m'ont accordé de leur temps pour la relecture et la correction de ce document.

Table des matières

Liste des tableaux et liste des figures

Liste des abréviations

AJAX	Asynchronous JavaScript XML
AOT	Ahead-Of-Time
APA	American Psychological Association
API	Application Programming Interface
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CUDA	Compute Unified Device Architecture
CUDNN	CUDA Deep Neural Network
DLL	Dynamic Link Library
DOM	Document Object Model
GPU	Graphical Processing Unit
HTML	HyperText Markup Langage
HTTP	HyperText Transfert Protocol
IoT	Internet of Things
IT	Technologie de l'information
JS	JavaScript
LLVM	Low Level Virtual Machine
MVP	Minimum Viable Product
PWA	Progressive Web Application
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3	World Wide Web
WA	WebAssembly
WASI	WebAssembly System Interface
WASM	WebAssembly
WWW	World Wide Web

Introduction

Contexte

Le World Wide Web (Web, WWW, W3 ou encore toile mondiale en français) est un ensemble de pages web publiques connectées entre elles grâce à internet. Ce dernier est omniprésent sur la plupart des appareils possédant une connexion à internet (Eberhardt, 2021). Grâce à ce système et à internet, nous pouvons accéder à plus de 1,84 milliards de sites web à travers le monde (Smaïl, 2021).

Les premières pages web se reposent sur le protocole HTTP (HyperText Transfert Protocol). Celui-ci simplifie le partage de données entre serveur - client, et l'HTML (HyperText Markup Langage) représente les informations textuelles ou/et graphiques d'un site. Néanmoins, ces premiers sites web ne permettent pas d'être interactif. Chaque visiteur possède la même expérience, comme celle d'un livre (Eberhardt, History of the Web: 1980s-1990s, 2021).

Au fil des années, le web prend en maturité. Il facilite les échanges de données entre serveur - client ou encore entre différentes plateformes avec l'aide de plusieurs facteurs. Tout d'abord, les API (Application Programming Interface) et le langage JavaScript propose une expérience un peu plus interactive. Ensuite, le concept d'AJAX (Asynchronous JavaScript + XML) laisse les applications JavaScript interagirent avec le serveur de manière asynchrone. Grâce à cette architecture, nous pouvons développer des applications web dynamiques. C'est-à-dire, nous modifions le contenu partiel d'une page sans devoir naviguer sur un URL (Uniform Resource Locator) différent (GAUTRON, 2014).

Problématique

Au cours des dernières années, le W3 a constamment évolué. C'est pourquoi, JavaScript permet de réaliser des applications web complètes et interactives.

Les navigateurs web s'équipent de moteurs JavaScript puissants en utilisant des méthodes de compilation comme le « Just-In-Time[1] ». Celles-ci permettent d'avoir des vitesses

[1] Compilation juste-à-temps en français. Améliore la performance en traduisant du bytecode en code machine natif.

d'exécution impressionnantes dans les navigateurs. En revanche, sa façon d'être envoyé et exécuté par le client n'est pas optimal. Le chemin pour y parvenir a peu évolué depuis l'invention du protocole HTTP (Eberhardt, LFD133x - Introduction to WebAssembly - So Why Do We Need WebAssembly ?, 2021).

Dans une application web, le code JavaScript est regroupé et est réduit en un seul fichier .js (fichier JavaScript). Celui-ci est transmis en texte brute à l'aide du protocole HTTP au navigateur du client, il est ainsi analysé en un arbre syntaxique. Puis, compilé en bytecode et finalement exécuté par un interpréteur (Eberhardt, LFD133x - Introduction to WebAssembly - So Why Do We Need WebAssembly ?, 2021). Ce chemin ne pose pas de problème lors de l'exécution d'applications simples. Cependant, les applications qui demandent des performances accrues comme l'édition de vidéo, le Machine Learning ou encore l'intelligence artificielle, cela devient un obstacle conséquent.

Pour remédier à ces difficultés, les créateurs de langages ainsi que les développeurs cherchent et proposent des alternatives pour intégrer des langages autres que le JavaScript.

L'alternative à laquelle nous allons nous intéresser est le WebAssembly. Il est destiné à devancer les performances de JavaScript dans un navigateur pour des applications à forte intensité de calcul.

Motivation

Ces derniers temps, l'optimisation de l'expérience client sur les applications devient une nécessité inévitable. La performance est l'un des principaux facteurs pour le confort de l'utilisateur. C'est la raison pour laquelle de nouvelles solutions sont recherchées et développées pour augmenter les différentes vitesses d'exécution des applications. La motivation principale de ce travail est de réussir à réduire la quantité des calculs opérés sur le serveur.

Structure du travail

Pour commencer, nous effectuons l'état de l'art du WASM. Après diverses recherches sur le sujet, les éléments fondamentaux comme son concept, son fonctionnement et ses diverses

particularités sont détaillés. Évidemment, nous allons également lister les langages et les applications pouvant être compatibles avec le WebAssembly.

Ensuite, afin d'approfondir et mettre en pratique ce nouveau standard, un prototype d'application est développé permettant de faire une classification d'images. Les projets de Machine Learning nécessitent d'importants volumes de calculs pour exécuter les algorithmes. C'est donc un exemple particulièrement intéressant, du moins du point de vue de la performance. En parallèle, le choix de l'environnement de développement et la méthodologie de travail sont présentés ainsi que les différents aspects de l'application.

Grâce à ce développement, nous pouvons effectuer diverses comparaisons comme la communication, la structure, la performance ou encore l'initialisation d'un projet WA. Nous énumérons également les différents problèmes auxquels nous sommes confrontés ainsi que les alternatives utilisées pour parvenir à l'aboutissement d'une application fonctionnelle.

Avant de terminer, nous effectuons le bilan du travail en répondant à la question si le WebAssembly demeure une solution adéquate pour réduire la charge de travail d'un serveur. Et pour clore ce travail, nous n'omettons pas de citer de potentielles améliorations pour des travaux futurs.

1. Méthodes

1.1. Définition WebAssembly

Le WebAssembly également connu sous WASM ou WA est un langage bytecode de bas niveau. Il a été conçu par une équipe composée d'ingénieurs de Mozilla, Google, Apple, Microsoft et divers autres acteurs du monde informatique. Le bytecode de celui-ci est habituellement généré en compilant un langage de programmation de plus haut niveau. Il est destiné à exécuter des applications côté client dans un navigateur Web ou en dehors (WebAssembly Overview, s.d.).

1.2. Historique

Historiquement, le JavaScript était le seul langage de programmation destiné pour les navigateurs Web comme précisé dans la section *Problématique* de l'introduction. Néanmoins, au fil des années, il a démontré qu'il possédait des difficultés au niveau de la performance avec les applications nécessitant des calculs complexes.

Une des alternatives pour améliorer ces performances était le projet asm.js. Il a vu le jour en 2013 grâce à l'équipe de Mozilla. Cette réalisation est un compilateur C++ qui génère un sous-ensemble de JavaScript en optimisant et supprimant des étapes d'exécution de JavaScript. En revanche, il garde une grande contrainte du JavaScript dont le fait est de fournir un fichier en texte brut (Carlos, 2017).

À la suite de son succès, le projet Asm.js est devenu une preuve de concept pour le WASM. De plus, lors du lancement du projet WebAssembly en 2015, de nombreux principes de conception d'Asm.js sont réutilisés.

À partir de 2017, les représentants de tous les principaux navigateurs (Chrome, Firefox, Safari et Edge) participent à la définition de la norme WA et de son architecture (Ball, 2017).

Pour finir, le 5 décembre 2019, WASM Core 1.0 est devenu officiellement un standard du Web avec HTML, CSS (Cascading Style Sheets) et JavaScript (Hiel, 2019).

1.3. Caractéristiques de développement

Lors de la mise en place du projet, les développeurs mettent une série de caractéristiques en place. Cela permet de définir les éléments fondamentaux de ce standard qui sont le MVP (Minimum Viable Product), la performance, la portabilité, la sécurité et un code de bas niveau.

1.3.1. Minimum viable produit

Le développement de nouveaux concepts web demande beaucoup d'implication et de temps pour transiter de l'idée rudimentaire à celle du déploiement. Pour réduire cette durée, le projet initial de WebAssembly est un MVP. Dès le départ, des cas d'utilisation très

spécifiques sont définis comme le traitement d'images, les jeux vidéo et d'autres applications demandant de grande capacité de calcul (Clark, 2018).

La définition de cet objectif permet à WA d'être mis à disposition des développeurs rapidement. En revanche, des ajouts de fonctionnalités doivent être faits au fil du temps, pour rendre le WebAssembly utile à plus large échelle.

1.3.2. Performance

Grâce à son format binaire WASM réussit à prendre en charge le décodage de plusieurs threads en similitude et d'instanciation en continu, contrairement à JavaScript qui est au format texte. Cette conception lui permet d'être chargé, décodé et exécuté très rapidement, mais surtout d'être proche d'une vitesse de code machine natif (WebAssembly Overview, s.d.).

1.3.3. Portabilité

Les technologies du web doivent être développées dans un souci de portabilité. C'est-à-dire réussir à être exécuté de manière identique peu importe le navigateur ou le système d'exploitation de l'utilisateur. Le WebAssembly peut être utilisé sur les quatre grands navigateurs ; Firefox, Chrome, Safari et Microsoft Edge (Guilloux, 2017).

Les modules du WebAssembly peuvent également être hébergés dans d'autres environnements comme ; le cloud, l'IoT[2] (Internet of Things ou internet des objets en français) ou encore la blockchain (Eberhardt, LFD133x - Introduction to WebAssembly - Portability, 2021).

1.3.4. Sécurité

Le WebAssembly s'exécute dans un navigateur. Il est donc dans un environnement hostile du point de vue de la sécurité. De ce fait, il se rallie aux mêmes règles de sécurité que son

[2] Interconnexion entre des objets et internet.

environnement hôte. Une page web n'autorise pas le chargement ni le lancement d'un élément malveillant d'une origine étrangère.

De plus, le WebAssembly comme le JavaScript se repose sur le principe de SandBox[3]. Il s'agit d'une caractéristique primordiale pour les applications basées sur un navigateur. De cette manière, il est impossible d'accéder à l'environnement hôte (système d'exploitation de la machine de l'utilisateur, mémoire interne du navigateur), sauf à l'aide d'API JavaScript (Phelps, 2019).

1.3.5. Code de bas niveau

En théorie, WA n'est pas destiné à être un langage écrit à la main mais plutôt une cible de compilation. Dans son principe, il doit être codé dans un langage de programmation écrit par l'homme et compilé en code binaire lisible par la machine.

Figure 1 : Code en bytecode. https://www.javascriptjanuary.com/blog/webassembly-neither-web-nor-assembly-but-revolutionary

Le bytecode est de très bas niveau. Il se repose sur le même principe que le code des ordinateurs natifs. Il optimise les performances et il n'est pas nécessaire de se soucier du type de CPU (Central Processing Unit ou Unité central de traitement en français) de l'utilisateur. En effet, les navigateurs soumettent une machine virtuelle intégrée pour le WebAssembly (Phelps, 2019).

1.4. Fichier .wasm

WA possède un format de fichier binaire très compact et efficace. Cela signifie que son code est moins volumineux qu'un code JavaScript similaire et donc se télécharge plus

[3] Bac à sable en français. Principe d'isolation de composants informatiques.

rapidement. C'est le résultat d'une compilation en code machine durant son téléchargement. Nous appelons cela la compilation en continu (Communauté Mozilla, s.d.).

Cependant, avec une extension .wasm le code n'est pas lisible par un humain. Pour y remédier, le code de WebAssembly peut être proposé au format texte avec l'extension .wat (Ramesh, 2019).

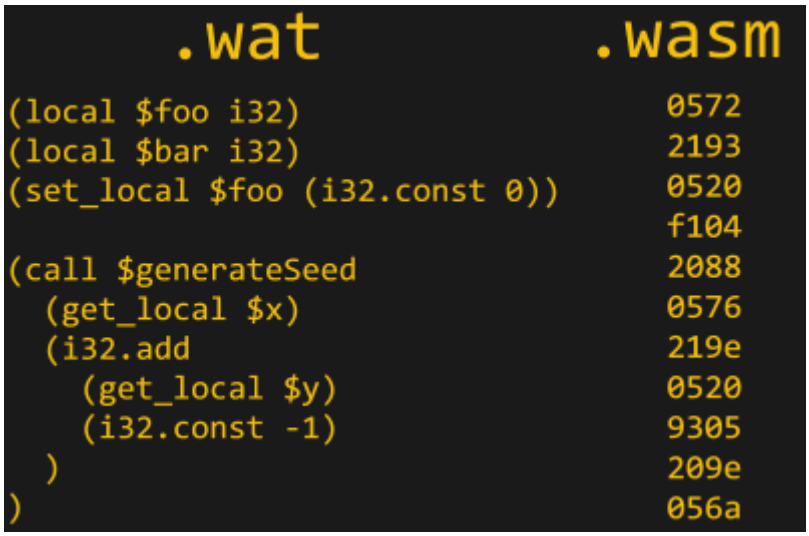


Figure 2 : Code avec extension .wat VS .wasm
https://hackaday.com/2019/04/04/webassembly-what-is-it-and-why-should-you-care/

WebAssembly est pourvu d'un système de seulement 4 types très simples, tous numériques. Il a deux entiers 32 et 64 bits et deux flottants 32 et 64 bits. En comparaison avec JavaScirpt, celui-ci ne prend en charge que les nombres en 32 bits. À cause de cela, ses capacités pour certaines tâches mathématiques sont limitées (Communauté WebAssembly).

1.5. WASI

Depuis mars 2019, un projet nommé WASI (WebAssembly System Interface) est en développement afin de permettre à WASM d'accéder à des ressources systèmes tels que les fichiers et les connexions réseau.

En effet, comme précisé dans la partie *1.3.4 Sécurité* des caractéristiques de développement, dans sa version actuelle WASM ne permet pas d'accéder à l'environnement de son hôte. Mais avec WASI, il sera possible d'interagir avec le système hôte pour lire et écrire des fichiers, créer des sockets réseau et bien plus encore. Tout cela avec la même portabilité,

sécurité et format binaire compact que nous obtenons avec WebAssembly actuellement (Yegulalp, 2019).

Concernant la sécurité pour les accès fichiers, le principe de bac à sable reste le même pour WASI. Avant de pouvoir lire ou écrire un fichier, l'application doit d'abord disposer des autorisations nécessaires pour accéder au répertoire en question. Les permissions peuvent aussi être limitées sur des modules distincts.

1.6. Langage

WebAssembly est en constante évolution. Les langages qui peuvent le prendre en charge augmentent régulièrement. Actuellement, les langages disponibles selon le site officiel sont le C/C++, Rust, C#, F#, Go, Kotlin, Swift, D, Pascal, Zig ou encore AssemblyScript (Getting Started, s.d.).

Cependant, en effectuant des recherches sur le Web, nous dénichons différents projets open-source permettant d'intégrer le WebAssembly avec d'autres langages. Ces solutions sont développées par diverses communautés sur des plateformes comme Github.com.

De plus, l'environnement de chacun de ces langages est très différent. WASM s'intègre de manière distincte à chacun d'eux. En revanche, il est important de préciser que les langages C/C++, Rust et C# restent pour le moment les plus matures pour être utilisés sur des projets en WebAssembly.

1.6.1. C/C++ avec Emscripten

Le langage C/C++ est très utilisé dans le monde de la programmation. Grâce à sa popularité, il possède énormément de documentations et une grande communauté. De ce fait, il bénéficie de nombreuses bibliothèques et extensions. À l'aide de ces dernières et de sa rapidité, le C/C++ réalise une grande panoplie d'application diverses et performantes. (Nebra & Schaller, 2021).

Emscripten est la première chaîne d'outils pour la création d'application WebAssembly. Il est Open Source. Son fonctionnement est relativement simple. Il prend un langage de haut niveau comme le C, le C++ ou un autre langage qui est basé sur la chaîne d'outils de

compilation LLVM[4] (Low Level Virtual Machine) et le compile en WebAssembly. Le principe de LLVM permet de simplifier l'architecture pour qu'Emscripten génère du code petit et rapide. De plus, lors de la compilation, Emscripten optimise automatiquement le code de l'application (Emscripten Contributors, s.d.).

Emscripten fait ses preuves depuis plusieurs années. D'ailleurs, des applications connues l'utilisent telles que :

- Google Earth : pour ne pas devoir migrer le code source C++ vers JavaScript pour rendre l'application de bureau Google Earth accessible sur le web, les développeurs ont utilisé le compilateur Emscripten (Mears, 2019).
- Bibliothèque TensorFlow : afin de remédier à la lenteur de JavaScript en arrière-plan, l'équipe a ajouté un backend en WASM. Le noyau principal de TensorFlow est en C++.

1.6.2. Rust

Rust est un nouveau langage de programmation développé par l'équipe de Mozilla. Il se base sur le principe du C/C++ mais offre des atouts supplémentaires et intéressants. Il est avant tout utilisé pour le développement d'applications multithreads performantes et propose davantage de sécurité. Il emploi également la chaîne d'outils LLVM pour la compilation vers le WebAssembly. Grâce à sa popularité, il possède une communauté dynamique. Celle-ci crée des outils pour faciliter son utilisation (Développement web, 2020).

1.6.3. C# avec Blazor

Le langage C# est introduit par Microsoft. Il s'inspire plus du langage Java que du C ou du C++. Avec l'avancement dynamique du langage, il sait se différencier de Java avec ses spécificités. Pour profiter pleinement de toutes ses ressources, il est nécessaire de l'associer au .NET Framework ou au .NET Core. À l'aide de ces outils, il crée des applications lourdes mais performantes avec des fonctionnalités tels que des accès réseau, l'appel à des bases de données ou encore l'utilisation de webservices. C'est un langage moderne avec une

[4] Machine virtuelle de bas niveau en français. Permet d'optimiser du code lors d'une compilation.

bibliothèque riche et des outils de développement avancés qui facilitent son utilisation. Appartenant au milieu Microsoft, il profite de la présence d'une très grande communauté active sur différents forums et possède énormément de documentations (Redkire, 2020).

Blazor est un environnement de développement basé sur le .NET de Microsoft et le C#. Il se repose sur la syntaxe Razor[5] et d'une interface utilisateur basée sur des composants. Il n'est pas seulement un compilateur C# vers WebAssembly mais aussi un outil complet pour le développement d'applications web. Blazor dispose d'une forte communauté active sur les forums. De plus, ils produisent un grand nombre de tutoriels ou d'applications tests, afin de faciliter son utilisation et le rendre de plus en plus populaire (Communauté Microsoft, s.d.).

Blazor peut aisément appeler des API et des bibliothèques JavaScript. De cette manière, il est possible d'utiliser ces bibliothèques pour l'interface utilisateur côté client (Dulanga, 2020).

En outre, Blazor permet aussi de développer des applications serveurs. À l'inverse d'une application en WebAssembly, le code C# est exécuté sur le serveur. La machine du client ne comporte que de l'HTML, CSS et du JavaScript. Les échanges entre serveur et client se réalisent via une connexion SignalR[6] (Bernard, 2020). En théorie, il est relativement aisé de passer d'une application serveur à une WA avec Blazor et inversement. Pour cela, il faut ajouter/ supprimer des classes de connexion et des références à l'application.

1.6.4. AssemblyScript

AssemblyScript se différencie des autres langages au niveau de son fonctionnement avec le WebAssembly qui a été spécialement conçu pour lui.

AssemblyScript se repose sur la chaîne LLVM en utilisant l'outil Binaryen[7]. Il s'appuie sur la syntaxe de TypeScript[8], il est même un sous-ensemble de ce langage qu'il compile en WebAssembly. Il s'utilise pour les applications avec un background en TypeScript et des API

[5] Une syntaxe de balisage

[6] Cadre de messagerie en temps réel.

[7] C'est un compilateur. Il cherche à rendre la compilation vers WebAssembly rapide, facile et efficace.

[8] Sur-ensemble de JavaScript avec un typage statique.

JavaScript standard pour compiler en WA. De ce fait, il n'est pas nécessaire de basculer entre différents langages (Guilloux, 2017).

AssemblyScript est très similaire syntaxiquement à JavaScript. Il permet aux développeurs habitués à JavaScript de le comprendre et l'utiliser rapidement.

La différence entre AssemblyScript et TypeScript est au niveau de la compilation. TypeScript s'exécute en JavaScript, alors que AssemblyScript en WebAssembly. Il cible la totalité des fonctionnalités de WASM et donne un contrôle de bas niveau sur le code. Les développeurs de JavaScript et TypeScript obtiennent donc les spécifications et les avantages de WebAssembly, avec son atout principal ; la performance (Eberhardt, LFD133x - Introduction to WebAssembly - AssemblyScript, 2021).

AssemblyScript est un langage naissant avec peu d'utilisateurs. C'est pourquoi, il ne possède pas une très grande communauté. De plus, il est relativement difficile de trouver des informations intéressantes et complètes sur lui.

1.6.5. Python

Comme mentionné dans l'abstract de ce chapitre, il existe maintes collectivités impatientes d'expérimenter le WebAssembly avec leur langage de prédilection. C'est d'ailleurs le cas avec Python.

Pour commencer, ce langage est l'un des plus populaires dans le milieu des développeurs Web. Il possède une impressionnante communauté active. Laquelle propose une multitude de documentations diverses. Ensuite, il donne accès à une pléthore de bibliothèques. Et pour finir, son apprentissage est relativement aisé.

Il est cohérant à la vue de sa notoriété, que des idées de concepts pour son intégration avec WA allait émerger.

Pyodide est un projet open-source indépendant dirigé par plusieurs contributeurs sur Github.com. Ses développeurs sont des membres de la communauté active de Python. Il fonctionne entièrement dans le navigateur et permet l'accès aux différents API Web. Cependant, son concept est récent. Ce n'est que dans le courant du mois de mai 2021, qu'il a

été officiellement annoncé par Mozilla comme une solution visant à compiler du code Python en WASM.

En outre, Pyodide détient plusieurs environnements expérimentaux de calcul scientifique interactif pour le Web. Cette collection scientifique de Python comprend (Couriol, 2021) :

- NumPy : pour les calculs scientifiques.
- Pandas : pour les analyses de données.
- Matplotlib : pour les bibliothèques de traçages.
- SciPy : pour les calculs scientifiques et techniques.
- Scikit-learn pour l'apprentissage automatique.
- 75 autres paquets.

Concernant sa mise en place dans un projet concret, un script est disponible sur la plateforme Github.com. Celui-ci, permet l'initialisation et le chargement des principaux modules. Ensuite, pyodide.org, le site officiel liste différentes publications qui permettent de mieux comprendre son fonctionnement ainsi que comment l'implémenter.

À l'heure actuelle, Pyodide ne bénéficie pas d'une importante communauté. De ce fait, il est relativement compliqué de s'informer correctement sur le sujet et réaliser un apprentissage de qualité.

Pour terminer, d'après son site officiel, les performances de Pyodide doivent encore être améliorées. Ils citent dans leur feuille de route : « Dans les benchmarks, Pyodide est actuellement 3 à 5 fois plus lent que le langage Python natif » (Contributeurs du projet Pyodide, s.d.).

1.7. Applications utilisant WebAssembly

Les caractéristiques et les avantages de WebAssembly incitent abondamment les développeurs à expérimenter le concept et à l'adopter. Et cela, grâce à ses deux principaux avantages ; la simplicité et la rapidité.

Pour commencer, WA est un moteur simple et de bas niveau comme précisé dans le sous-chapitre *1.3 Caractéristiques de développement*. Il ne bénéficie pas d'approche au niveau de

la gestion de la mémoire, ni de ses propres API pour le système. Créant ainsi un moteur d'exécution léger.

En ce qui concerne sa rapidité, il est conçu pour être principalement distribué sur le Web. Là, où le temps de téléchargement et de compilation est autant important que les performances d'exécution. Il contribue à des performances presque natives lors de la navigation.

Conçu initialement pour des applications spécifiques demandant beaucoup de performances dans le navigateur, WebAssembly ne reste pas dépendant à lui-même. Dès le départ, il est conçu pour éviter toutes dépendance à l'environnement du navigateur.

D'ailleurs, son adoption d'applications hors du navigateur est relativement important. Nous allons aborder quelques exemples d'applications qui peuvent tirer des avantages conséquents avec WA.

Mais avant cela, il est important de relever que le standard de WebAssembly s'adapte avec la plupart des projets se trouvant sur le Web et s'exécutant dans un navigateur.

1.7.1. Blockchain

Au cours des dernières années, la blockchain (chaîne de blocs en français) suscite énormément d'enthousiasme et d'investissements. Dans une vidéo de TEDx, Claire Balva (CEO de Blockchain Partner et cofondatrice de Blockchain France) définit la blockchain comme : « une technologie de stockage et de transmission d'informations, transparente, sécurisée, et fonctionnant sans organe central de contrôle » (Balva, 2017). Grâce à cela, il est possible d'exécuter des transactions de cryptomonnaie selon une combinaison d'algorithmes.

Aujourd'hui, la cryptomonnaie la plus connue est le Bitcoin. Afin de promouvoir la simplicité et la sécurité dans l'exécution des transactions, elle est construite grâce à un langage de script très simple. Néanmoins, cela inflige des contraintes au niveau de la complexité des transactions (Reiff, 2020).

À l'inverse, l'Ethereum, une autre monnaie virtuelle également très connue, utilise un langage beaucoup plus puissant pour exécuter les transactions et créer des applications plus

complexes. Ces dernières sont développées avec Solidity, un langage orienté objet propre à Ethereum. Malgré cela et à sa rapide popularité, cette cryptomonnaie fait face à des défis techniques. Le plus conséquent est la lenteur du réseau. Le temps d'exécution des transactions varie de quelques secondes à plusieurs minutes.

Pour remédier à ce problème, l'équipe d'Ethereum a testé un moteur d'exécution basé sur WebAssembly, afin d'en tirer ses avantages. Les premiers atouts sont l'indépendance envers l'hôte, le principe de SandBox pour la sécurité et finalement sa simplicité. Puis, les développeurs peuvent choisir parmi un large choix de langages, plutôt que Solidity. Convaincus par son efficacité, ils ont introduit WA dans leur feuille de route pour l'Ethereum v2.0 (Eberhardt, WebAssembly on the Blockchain and JavaScript Smart Contracts, 2019).

Le potentiel de WebAssembly permet d'augmenter les performances et devient une solution de choix pour la communauté de la blockchain.

1.7.2. Serverless

La technologie serverless qui signifie sans serveur en français est relativement nouvelle. Le principe est simple : exécuter le code d'une application sans devoir configurer soi-même les serveurs physiques ou machines virtuelles. Pour cela, il suffit de déployer le code dans le cloud. Ce sont les fournisseurs du cloud qui s'occupent de toute l'infrastructure IT en assurant l'intendance des machines (Biseul, 2018).

La sécurité, la rapidité d'exécution et l'isolation sont d'une importance primordiale pour les fournisseurs de cloud. Il est possible d'améliorer leurs fonctionnements avec les différents atouts du WebAssembly.

Avec le principe de SandBox et son isolation, il est possible d'interpréter un plus grand nombre de modules WebAssembly dans le même processus. Ainsi, ces fournisseurs réduisent le temps de lancement des applications ou encore les latences au niveau des requêtes (Aboukhalil, 2019).

1.7.3. Jeux vidéo

Depuis plusieurs années, l'industrie des jeux vidéo augmente considérablement. Selon une étude de Newzoo[9], il estime le nombre de joueurs de jeux vidéo à plus de 2.8 milliards dans le monde (Takahashi, 2021).

Afin de satisfaire la demande, cette industrie possède un grand nombre de maisons d'éditions de jeux vidéo. Entre lesquelles règnent une rude concurrence. Pour se démarquer, ils donnent une grande importance aux performances des jeux vidéo. Il est envisageable de se démarquer sur ce point avec WA.

WebAssembly peut apporter son aide à des petits jeux, des jeux AAA[10] ou encore des moteurs de jeux en ligne. Grâce à lui les jeux sur console ou sur ordinateur peuvent être exécutés dans un navigateur. De ce fait, ils ne comportent pas de limitation imposée par les systèmes d'exploitations ou par les fabricants des consoles (Bryant, 2019).

1.7.4. Machine Learning

Un autre thème omniprésent dans notre société ces dernières années, est l'intelligence artificielle ou plutôt sa sous-catégorie le Machine Learning (apprentissage automatique en français). Ce domaine scientifique laisse des algorithmes détecter des patterns[11]. Ils analysent de manière autonome un ensemble de données comme : les chiffres, les mots, les images et toutes les données qui peuvent être stockées numériquement. Au fil des analyses, l'algorithme s'entraine à améliorer ses performances afin de déceler des patterns dans de nouvelles données (Bastien, 2020).

Nous retrouvons du Machine Learning dans énormément de services modernes connectés comme les systèmes de recommandations de contenu (Netflix, Spotify, les publicités ciblées, les fils d'actualité de réseaux sociaux, etc.), les assistants vocaux (Siri, Alexa), ainsi que les moteurs de recherche web tels que (Google, Baidu) et pleins d'autres encore. Pour réussir à

[9] Société de conseil en analyse de jeu.

[10] Jeux avec de gros budgets.

[11] Motifs récurrents.

prédire au mieux des recommandations, il faut nourrir l'algorithme avec des données. C'est la raison pour laquelle les géants du web collectent nos données personnelles (Yu, 2019).

Il existe plusieurs utilisations du Machine Learning et de sous-catégories dont une relativement connue, le Deep Learning (apprentissage profond en français). Cette technique se base sur un large nombre de couches de nœuds de calcul constituant un réseau. Le réseau peut être comparé à un cerveau humain et les nœuds de calculs à des neurones. Le Deep Learning possède une grande aptitude à résoudre les patterns même les plus subtiles (Thomas, 2020).

L'aboutissement de cette méthode consiste à fournir une prédiction après avoir analysé une image. Elle est en mesure d'identifier des lieux, des personnes, des objets et tout autre élément au sein d'une image ou d'une vidéo. Dans son fonctionnement, cette détection d'images analyse chaque pixel afin de prélever des informations.

Cependant, que cela soit le Machine Learning ou le Deep Learning, tous deux nécessitent beaucoup de ressources de calcul pour s'exécuter. De ce fait, la vitesse d'exécution de l'algorithme peut-être relativement lente.

Le WebAssembly est nativement multithread. Ce qui donne une meilleure vitesse d'exécution pour les applications lourdes comme le Deep Learning. De plus, après son téléchargement sur la machine du client, l'application réduit son utilisation des ressources côté serveur. C'est pourquoi, les calculs sont en principe directement exécutés dans le navigateur du client.

1.8. Différences au niveau de la communication

Dans ce chapitre et le suivant, nous découvrons les différences de communication et celles au niveau de la structure d'une application WA et standard. Pour cela, nous nous appuyons sur un projet Blazor WebAssembly et Blazor Server. Dans la partie *1.14 Choix environnement de développement* de ce rapport, nous détaillons les raisons pour lesquelles nous avons décidé de choisir le C# et Blazor pour le développement de notre prototype.

Dans un projet Blazor WA, toute l'application tourne directement sur la machine du client. Le code C# est compilé dans une DLL (Dynamic Link Library ou bibliothèque de liens dynamiques en français) et envoyé depuis le serveur d'hébergement sur l'ordinateur du client qu'une seule fois. Lors de la première utilisation de l'application, il est nécessaire de patienter un court instant durant le téléchargement de celle-ci. De plus, avec ce principe, ce sont les ressources matérielles du client qui sont utilisées et non celle du serveur lors de l'exécution.

Au contraire, une application Blazor Server ou une application standard va tourner sur un serveur ainsi que son code C#. Le client ne contient que du HTML, CSS et JavaScript. Une communication avec des échanges est effectuée pour profiter des fonctionnalités de l'application. À l'inverse de WA, ce sont les ressources du serveur qui sont utilisés avec ce principe.

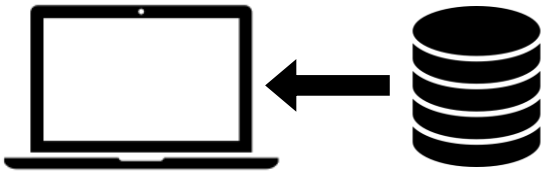
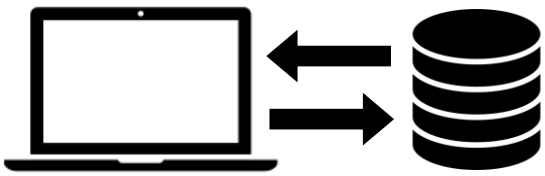
Blazor - WEBASSEMBLY	Blazor - SERVER
	
Code C# exécuté sur la machine du client	**Code C# exécuté sur le serveur**

Figure 3 : Différence au niveau de la communication (Illustration réalisée par l'auteur de ce document)

Ce principe de communication pour le WebAssembly permet de réaliser une application fonctionnant entièrement hors ligne et déconnectée du serveur. En revanche, ce schéma s'applique uniquement pour des applications n'ayant pas besoin d'échanges de données ou d'accès à des fichiers. Étant donné qu'elle est exécutée sur la machine du client, WASM ne

permet pas d'avoir de logique métier ou d'accès aux données à cause de la sécurité. Pour remédier à cela, il est nécessaire de mettre en place une API. Les données sont récupérées sur le serveur via cette API et une communication avec des échanges avec celui-ci est maintenu.

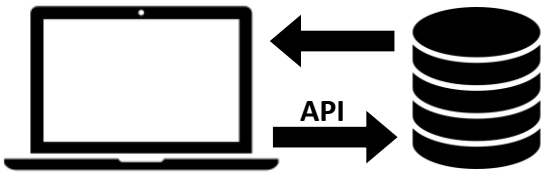
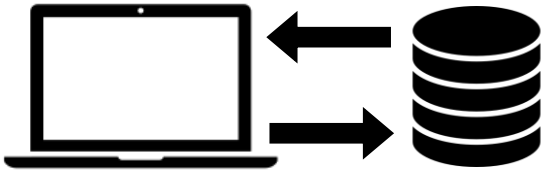
Blazor - WebAssembly	Blazor - Server
	
Code C# exécuté sur la machine du client	**Code C# exécuté sur le serveur**

Figure 4 : Différence au niveau de la communication avec API (Illustration réalisée par l'auteur de ce document)

1.9. Différences au niveau de la structure

Un projet Blazor WebAssembly et Server partagent une structure plutôt identique mais avec certaines différences qu'il est intéressant de relever.

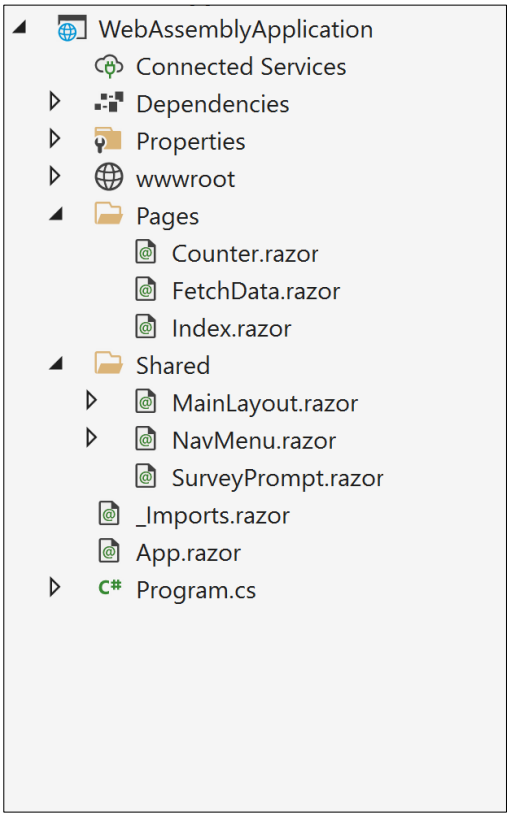


Figure 5 : Structure de l'application WASM (Illustration réalisée par l'auteur de ce document)

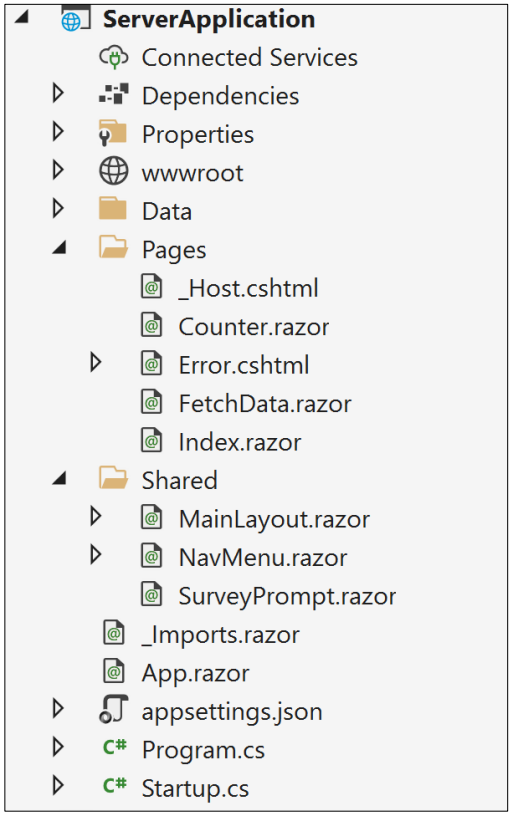


Figure 6 : Structure de l'application Server (Illustration réalisée par l'auteur de ce document)

1.9.1. Fichier projet .csproj

Nous remarquons que le contenu du fichier projet de l'application WASM comporte des références et que la cible du projet est différente de celle du serveur.

```
<Project Sdk="Microsoft.NET.Sdk.BlazorWebAssembly">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly" Version="5.0.6" />
    <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly.DevServer" Version="5.0.6" PrivateAssets="all" />
    <PackageReference Include="System.Net.Http.Json" Version="5.0.0" />
  </ItemGroup>

</Project>
```

Figure 5 : Fichier projet .csproj WASM (Illustration réalisée par l'auteur de ce document)

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
  </PropertyGroup>

</Project>
```

Figure 6 : Fichier projet .csproj Server (Illustration réalisée par l'auteur de ce document)

Tout d'abord, la cible est différente entre les deux applications. Le projet WA s'exécute dans le navigateur sur un runtime .NET de WebAssembly. C'est pourquoi, il cible « Microsoft.NET.Sdk.BlazorWebAssembly » et non « Microsoft.NET.Sdk.Web » comme le projet serveur.

Ensuite, nous retrouvons plusieurs packages références dans le projet WebAssembly que l'application serveur ne possède pas. C'est grâce à ces références que nous pouvons développer une application côté client.

1.9.2. Program.cs

Les deux projets contiennent une méthode Main() qui correspond au point d'entrée de l'application. Ces classes créent et exécutent une instance d'hôte web qui gère le cycle de vie de l'application.

Dans la méthode Main() du projet WebAssembly, nous voyons que les différents services y sont déjà déclarés. Ils permettent de faire des configurations, des injections de dépendances et tous autres paramétrages.

```
namespace WebAssemblyApplication
{
    1 reference
    public class Program
    {
        0 references
        public static async Task Main(string[] args)
        {
            var builder = WebAssemblyHostBuilder.CreateDefault(args);
            builder.RootComponents.Add<App>("#app");

            builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });

            await builder.Build().RunAsync();
        }
    }
}
```

Figure 9 : Program.cs WASM (Illustration réalisée par l'auteur de ce document)

```
namespace ServerApplication
{
    1 reference
    public class Program
    {
        0 references
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }
        1 reference
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Figure 10 : Program.cs Server (Illustration réalisée par l'auteur de ce document)

1.9.3. Startup.cs

La classe Startup n'existe que dans le projet Server. Elle contient la logique de démarrage de l'application. Nous voyons également qu'à l'inverse d'un projet WASM les services utilisés dans l'application sont déclarés dans cette classe et non dans Program.cs. Elle est appelée depuis la méthode Main() de la classe Program.

Illustration du code disponible sous l'annexe I.

1.9.4. App.razor

Tout d'abord, les fichiers avec une extension .razor définissent les composants caractérisant l'interface utilisateur. Ils gèrent les différents rendus, les méthodes de cycle de vie, la gestion des événements et d'autres logiques. Dans la plupart des cas, ces fichiers sont identiques dans un projet WebAssembly et server. Lors de l'exécution du projet, ces fichiers sont compilés dans une seule classe .NET.

Ce fichier est similaire dans les deux projets. Il permet de gérer le routage de l'application. La propriété « Found » est utilisée pour trouver et afficher les pages demandées. Si aucune correspondance existe, la propriété « NotFound » affiche un message d'erreur.

```
<Router AppAssembly="@typeof(Program).Assembly" PreferExactMatches="@true">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
    </Found>
    <NotFound>
        <LayoutView Layout="@typeof(MainLayout)">
            <p>Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>
```

Figure 7 : App.razor WASM (Illustration réalisée par l'auteur de ce document)

1.9.5. _Imports.razor

Ce fichier à l'aide de @using déclare un ensemble de directives à importer dans tous les composants razor.

```
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using WebAssemblyApplication
@using WebAssemblyApplication.Shared
```

Figure 12 : _Imports.razor WASM (Illustration réalisée par l'auteur de ce document)

```
@using System.Net.Http
@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.JSInterop
@using ServerApplication
@using ServerApplication.Shared
```

Figure 13 : Imports.razor Server (Illustration réalisée par l'auteur de ce document)

1.9.6. Appsettings.json

Ce fichier n'est présent que dans le projet Server. Il est utilisé pour sauvegarder les configurations de l'application.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

Figure 8 : Appsettings.json WASM (Illustration réalisée par l'auteur de ce document)

1.9.7. wwwroot

Le dossier wwwroot contient les éléments statiques de l'application comme les images, les fichiers CSS, l'HTML et bien d'autres. Ces fichiers sont adressables par le web, cela signifie qu'on peut y accéder à l'aide d'un lien URL sur internet.

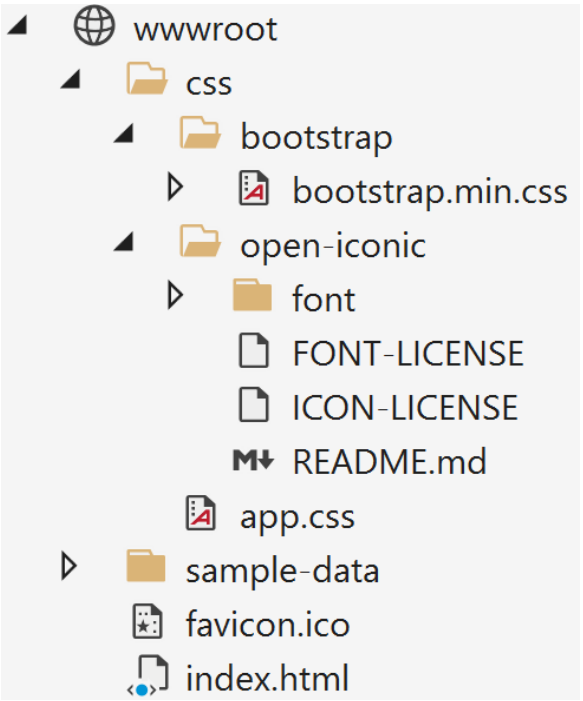


Figure 15 : Dossier wwwroot WASM (Illustration réalisée par l'auteur de ce document)

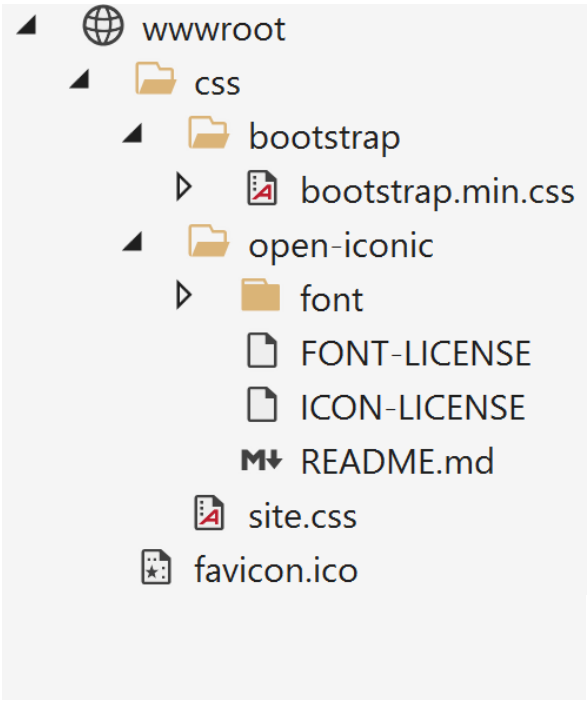


Figure 16 : Dossier wwwroot Server (Illustration réalisée par l'auteur de ce document)

Nous remarquons qu'il existe deux fichiers aux contenus identiques mais l'appellation demeure différente. Il s'agit du fichier app.css sur le projet WebAssembly et site.css sur celui de Server. Grâce au WASM, il est possible d'avoir une application web progressive. C'est-à-dire qu'en plus d'accéder à l'application par le Web, nous pouvons la télécharger directement sur notre machine et en faire une application de bureau. À l'inverse, avec un projet Server il n'est pas possible d'utiliser cette fonctionnalité.

Ensuite, nous observons que le projet WA possède un fichier index.html. Lors de la première requête, c'est cette page qui est affichée en retour. À l'intérieur de celle-ci, nous retrouvons les balises « head » et « body » dans lesquelles se trouvent les différents liens vers les fichiers du dossier wwwroot. À la fin de ce fichier, nous remarquons un script qui pointe vers un fichier JavaScript pour le framework de Blazor WebAssembly. Il permet de télécharger l'application compilée, les dépendances et les environnements .NET pour l'exécution dans le navigateur.

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
    <title>WebAssemblyApplication</title>
    <base href="/" />
    <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
    <link href="css/app.css" rel="stylesheet" />
    <link href="WebAssemblyApplication.styles.css" rel="stylesheet" />
</head>

<body>
    <div id="app">Loading...</div>

    <div id="blazor-error-ui">
        An unhandled error has occurred.
        <a href="" class="reload">Reload</a>
        <a class="dismiss">✕</a>
    </div>
    <script src="_framework/blazor.webassembly.js"></script>
</body>

</html>
```

Figure 9 : index.html WASM (Illustration réalisée par l'auteur de ce document)

Ce fichier existe également dans le projet Server mais il est localisé dans un autre répertoire avec une appellation différente. Celui-ci se nomme _Host.cshtml et possède les mêmes éléments que le fichier index.html du projet WASM. En revanche, le script est différent car il tend vers un JavaScript du framework Blazor Server. Par conséquent, son rôle n'est pas le

même. Dans une application serveur, ce fichier JavaScript est responsable de la gestion de la connexion entre le navigateur et le serveur pour leurs échanges d'informations.

1.9.8. Pages

Le dossier pages contient les différentes pages avec l'extension .razor qui forment l'application. Un exemple de la structure d'un de ces fichiers est présenté dans le chapitre *1.14 Choix de l'environnement de développement*.

Nous pouvons remarquer que le projet Server possède deux fichiers en plus. Le premier est _Host.cshtml que nous avons déjà présenté dans le sous-chapitre précédent. Et le deuxième, le fichier Error.razor est utilisé si une exception non gérée se produit dans l'application.

1.9.9. Shared

Dans ce dossier nous retrouvons d'autres composants de l'interface utilisateur. Ceux-ci servent à gérer la disposition principale de l'application ainsi que le menu de navigation.

1.9.10. Structure avec une API

Lors de la création d'un projet WebAssembly, nous avons accès à une option nommée « ASP.NET Core hosted » (Hébergement ASP.NET Core en français). Si celle-ci est activée, elle va créer un projet prenant en charge une API et une structure différente que celle présentée précédemment.

Nous remarquons que le projet est partagé en trois partie distinctes :

- **WebAssemblyApplication.Client :** Il s'agit d'un projet web Blazor contenant le code public. C'est lui qui va être téléchargé et exécuté dans le navigateur du client. Il possède exactement la même structure que celle analysée dans les sous-chapitres précédents.
- **WebAssemblyApplication.Server :** Dans ce projet nous retrouvons le code privé qui ne sera pas téléchargé sur la machine du client et qui de ce fait restera du côté serveur. Celui-ci contient l'API REST qui va permettre de fournir un échange de

données au projet du client. La structure ressemble à celle d'un projet Blazor Server sans les éléments pour le Web et avec un dossier « Controllers » pour gérer l'API.

- **WebAssemblyApplication.Shared :** Ce dernier projet renferme des classes qui sont partagées entre le projet du client et celle du serveur. Il contient les modèles pour les objets utilisés dans les deux projets.

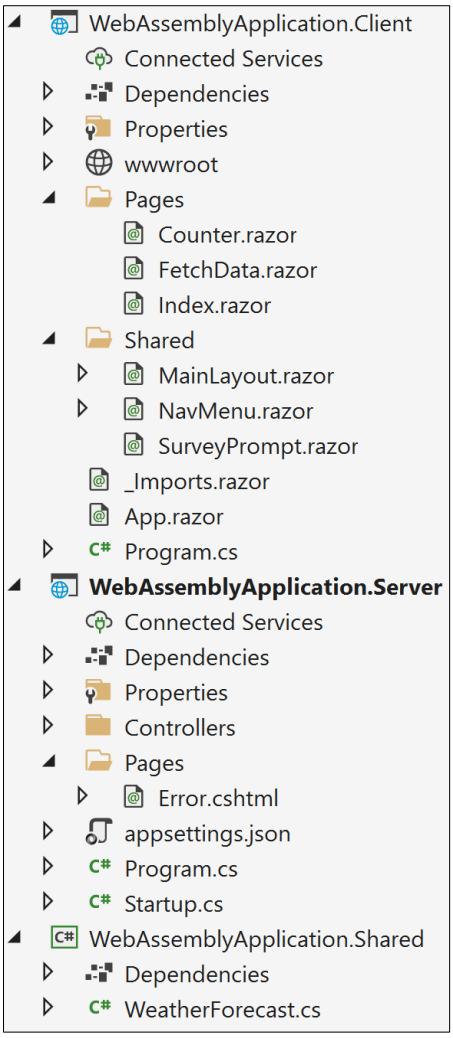


Figure 10 : Structure application avec API WASM (Illustration réalisée par l'auteur de ce document)

1.10. Convertir Blazor Server vers WebAssembly

Si dès le départ, l'application est développée correctement en respectant les règles et les logiques de Blazor alors il est relativement facile de convertir une application Blazor Server vers WebAssembly et inversement.

Les différences se trouvent au niveau des structures des fichiers des projets qui possèdent quelques différences dans leurs contenus.

Pour réussir le processus de conversion, il suffit de s'appuyer sur le chapitre *1.9 Différences au niveau de la structure*. Il faut réaliser les différents changements entre les deux structures d'applications comme ci-dessous :

- **Fichier projet .csproj** : changer la cible du projet et les packages de références.
- **Program.cs** : modifier le contenu pour changer le mode de démarrage de l'application.
- **Startup.cs** : supprimer ce fichier.
- **_Imports.razor** : modifier le contenu pour utiliser les bons composants razor.
- **Appsettings.json** : supprimer ce fichier.
- **wwwroot** : ajouter le fichier index.html en copiant le contenu de la *figure 17* du sous-chapitre *1.9.7 wwwroot*.
- **Pages** : supprimer le fichier « _Host.cshtml » et « Error.cshtml ».

1.11. Progressive web application

1.11.1. Définition

Une application web progressive (PWA) est une application utilisant les nouvelles capacités des navigateurs pour se comporter comme une application de bureau. Une PWA peut tout autant bien fonctionner dans un navigateur Web qu'en tant qu'application de bureau dans sa propre fenêtre. Les PWA tournent sur n'importe quel appareil pouvant être connecté au Web et cela avec la même base de code.

1.11.2. Fonctionnalités

Les applications Web progressives bénéficient de plusieurs avantages et de fonctionnalités intéressantes (Communauté Microsoft, 2021) :

- Chargement instantané de l'application indépendamment de la vitesse du réseau et celle d'internet.
- Capable de fonctionner hors ligne si aucunes fonctionnalités nécessitent de connexion réseau.
- L'application peut recevoir des notifications push d'un serveur même si l'utilisateur n'utilise pas l'application.
- Lorsqu'une mise à jour est disponible, l'application l'installe automatiquement. Si elle est en mode hors ligne, la mise à jour s'installera lorsque l'application sera de nouveau connectée à internet.
- Démarrage de l'application depuis le menu de démarrage du système d'exploitation ou depuis un raccourci sur l'écran d'accueil.
- L'application s'exécute dans sa propre fenêtre et pas seulement dans une fenêtre de navigateur.

1.11.3. Mise en place PWA lors de la création

La mise en place d'une PWA lors de la création d'un nouveau projet est relativement simple et rapide. Il suffit de cocher l'option « Progressive Web Application » dans la fenêtre de création d'un nouveau projet sur le logiciel Visual Studio. De cette manière, les fichiers nécessaires vont être automatiquement générées.

Quatre nouveaux éléments sont créés dans le dossier wwwroot. Les deux premiers fichiers service-worker.js et service-worker.published.js gèrent différents paramétrages comme le mode hors ligne ou encore la gestion du cache de l'application. D'ailleurs l'option hors ligne est désactivée lorsque l'application est en développement afin d'éviter des conflits si des modifications sont apportées lorsque l'application est hors ligne.

Le deuxième fichier est le manifest.json qui permet de personnaliser le titre de la fenêtre d'application, la palette de couleurs, l'icône et d'autres détails.

```
{
  "name": "Progressive Web Application",
  "short_name": "PWA",
  "start_url": "./",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#03173d",
  "icons": [
    {
      "src": "icon-512.png",
      "type": "image/png",
      "sizes": "512x512"
    }
  ]
}
```

Figure 11 : manifest.json WASM (Illustration réalisée par l'auteur de ce document)

Pour finir, nous trouvons une image qui servira d'icône pour représenter l'application sur l'écran d'accueil de la machine du client.

1.11.4. Mise en place PWA après la création

Il est toujours possible de rendre une application Web progressive par la suite même en n'ayant pas cocher l'option lors de la création. Pour cela, il est nécessaire de rajouter certaines références et de créer quelques fichiers.

Ajouter les deux références suivantes dans le fichier .csproj du projet :

```
...
  <ServiceWorkerAssetsManifest>service-worker-assets.js</ServiceWorkerAssetsManifest>
</PropertyGroup>
```

```
<ItemGroup>
  <ServiceWorker Include="wwwroot\service-worker.js" PublishedContent="wwwroot\service-worker.published.js" />
</ItemGroup>
```

Figure 12 : Mise en place PWA - fichier .csproj (Illustration réalisée par l'auteur de ce document)

Ajouter les trois références suivantes dans le fichier wwwroot/index.html :

```
<link href="manifest.json" rel="manifest" />
<link rel="apple-touch-icon" sizes="512x512" href="icon-512.png" />
```

```
    ...
    <script>navigator.serviceWorker.register('service-worker.js');</script>
</body>
```

Figure 13 : Mise en place PWA - fichier index.html (Illustration réalisée par l'auteur de ce document)

Pour terminer la configuration, il est fondamental d'ajouter les quatre fichiers qui sont créés si l'option PWA est activée, lors de la création du projet que nous avons vu dans le sous-chapitre précédent. Pour cela, il suffit de copier les fichiers « icon-512.png », « manifest.json », « service-worker.js » et « service-worker.published.js » dans le dossier wwwroot sous le lien suivant et les coller dans le dossier wwwroot de notre projet : https://github.com/dotnet/aspnetcore/tree/main/src/ProjectTemplates/Web.ProjectTemplates/content/ComponentsWebAssembly-CSharp/Client/wwwroot

1.11.5. Utilisation

Lorsque nous exécutons l'application et qu'elle s'affiche dans le navigateur, nous avons une option à droite de l'URL pour l'installer comme une application Web progressive.

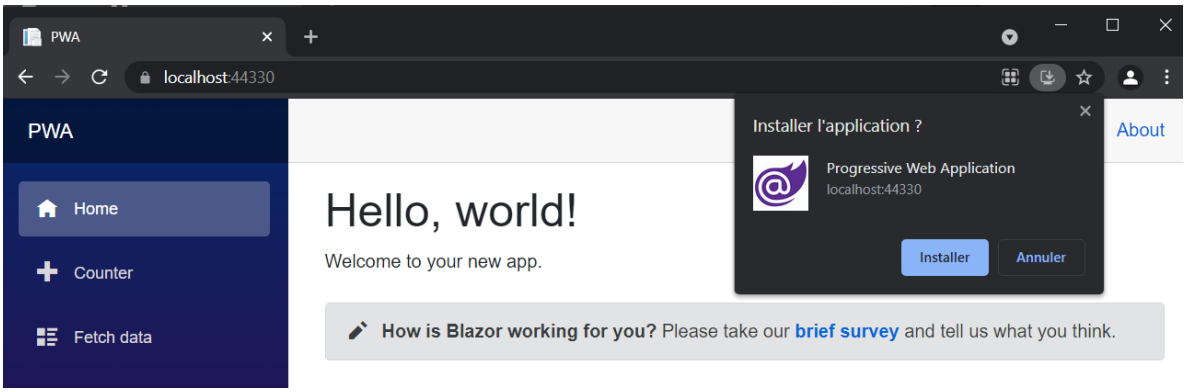


Figure 14 : PWA dans le navigateur (Illustration réalisée par l'auteur de ce document)

Une fois le téléchargement terminé, nous retrouvons notre application dans le menu démarrage du sytème d'exploitation de la machine et un raccourci est également crée sur l'écran d'accueil.

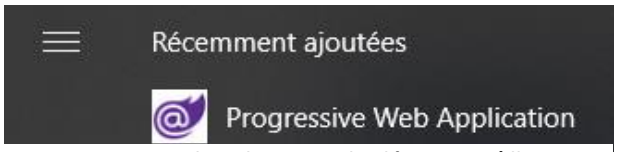


Figure 23 : PWA dans le menu de démarrage (Illustration réalisée par l'auteur de ce document)

Elle est à présent accessible depuis notre bureau avec sa propre fenêtre d'application. En appuyant sur les trois points en haut à droite de l'écran, nous y trouvons affichés des informations sur l'application dont son certificat, les cookies utilisés, une option pour copier l'URL et pour la désinstaller.

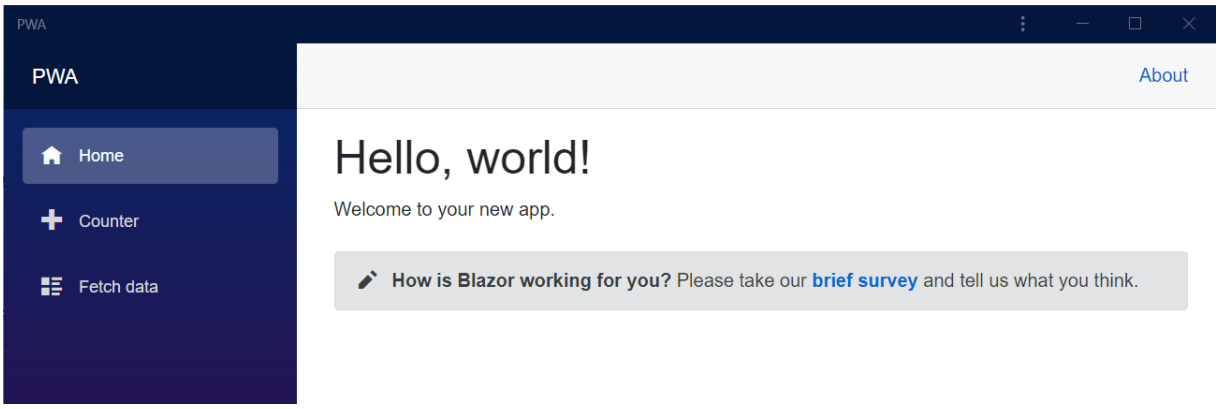


Figure 24 : PWA depuis le bureau (Illustration réalisée par l'auteur de ce document)

1.12. Avantages et inconvénients

Blazor - WebAssembly	
Avantages	**Inconvénients**
▪ Connexion au serveur active non requise (sauf avec API) ▪ Ressources et capacités du client utilisé ▪ Interaction plus rapide dans l'application ▪ Disponible hors ligne ▪ Transformer en application de bureau	▪ Premier lancement de l'application plus long ▪ Limité aux capacités du navigateur ▪ Le matériel du client doit être performant ▪ Une API est nécessaire pour un accès à des données ▪ Taille des fichiers à télécharger importante

Figure 15 : Avantages et inconvénients WASM (Illustration réalisée par l'auteur de ce document)

Blazor - Server	
Avantages	**Inconvénients**
▪ L'application se charge plus rapidement ▪ Peut tirer pleinement des capacités du serveur ▪ Le client n'a besoin que d'un navigateur ▪ Meilleure sécurité pour le code ▪ Accès à des ressources sécurisées	▪ Un serveur est requis ▪ Une connexion active est requise avec le serveur ▪ Latence plus élevée en raison d'aller-retour avec le serveur ▪ Pas de possibilité d'application hors ligne

Figure 16 : Avantages et inconvénients Server (Illustration réalisée par l'auteur de ce document)

1.13. Technologies alternatives

Le WebAssembly n'est pas l'unique solution qui propose une application côté client. Il existe d'autres technologies qui le permettent comme Angular, React ou encore Vue. Toutes ces solutions peuvent offrir des avantages considérables mais différents selon l'application que nous souhaitons développer. Ces technologies ne bénéficient pas d'une communauté égale car ils ne fournissent ni les mêmes fonctionnalités, ni les mêmes performances et divergent sur différents autres aspects. Il est donc impératif de définir le cadre du projet et sa complexité pour effectuer son choix.

1.13.1. Angular – Framework JavaScript

Le premier est un framework JavaScript du nom d'Angular. Celui-ci permet de développer et d'exécuter des projets Web clients complexes dans le navigateur. L'application est constituée de différents composants écrit en JavaScript ou TypeScript. Lors de l'exécution du code dans le navigateur, celui-ci est transmis vers un compilateur AOT[12] (Ahead-Of-Time ou compilation anticipé en français). Ce mécanisme se déclenche pendant l'implémentation de

[12] Traduction du code en langage machine lors de l'exécution.

l'application avant même que cette dernière soit téléchargée ou exécutée dans le navigateur. De ce fait, Angular procure un temps de chargement rapide et une sécurité accrue.

Ce framework est disponible depuis plusieurs années à l'inverse du WASM. Il est donc relativement populaire et possède une plus grande quantité de support. Nous trouvons du contenu sous forme de cours, de livres, de blogs, de vidéos et sous multiple autre forme.

Cependant, Angular se repose sur un concept propre à lui-même avec des terminologies spécifiques. Cela rend son apprentissage plus complexe. C'est pour cette raison qu'il ne demeure pas une solution de premier choix pour le développement d'application Web « simple ».

En outre, comme son rival WASM, il permet de développer des applications Web progressives (Beres, Blazor vs Angular Comparison in 2021, 2021).

1.13.2. React – Bibliothèque JavaScript

Nous présentons une bibliothèque JavaScript du nom de React, pour la deuxième alternative au WA. C'est une solution adaptable, efficace et déclarative. Elle permet de créer des applications modernes côté client en s'appuyant sur le vaste écosystème de JS comme le WebAssembly.

React bénéficie du principe du DOM (Document Object Model) virtuel. Cet atout permet d'identifier facilement un changement de données et de modifier le contenu au moment opportun. L'interface utilisateur réagit de manière plus rapide lors de la navigation sur l'application ainsi les performances du projet sont donc augmentées.

De plus, une application React montre de meilleurs résultats lors de l'initialisation. Celle-ci fournit des fichiers statiques empaquetés contrairement à WASM qui télécharge la totalité des bibliothèques DLL sur la machine du client.

À l'inverse d'Angular, cette solution ne bénéficie pas d'une grande communauté. Par conséquent, il est relativement difficile de trouver de la documentation officielle ou un support de qualité (Beres, Blazor vs React Comparison 2021, 2021).

1.13.3. Vue – Framwork JavaScript

La dernière solution présentée est un Framwork JavaScript portant le nom de Vue. C'est une technologie Open Source comme Angular et React. L'une de ses forces principales est sa légèreté par rapport aux autres alternatives. Elle permet d'être mise en place soit sur une application déjà existante pour une amélioration progressive soit pour être développée à partir de zéro. Comme pour la plupart des autres Frameworks JavaScript, un projet avec Vue se construit avec des composants. D'ailleurs, un outil du nom de « CLI » est disponible pour faciliter la création d'applications plus complexes avec cette technologie.

Cependant, Vue à l'image de React ne possède pas une communauté très développée. Cela rend son apprentissage plus complexe (Hilton, 2020).

Pour finir, Vue ainsi que les autres solutions mentionnées sont étroitement liées à l'écosystème JS. Ceci oblige les développeurs à posséder de bonnes connaissances sur ce langage pour profiter pleinement des atouts de ces technologies alternatives au WebAssembly.

1.14. Choix environnement de développement

Selon la donnée de ce travail de Bachelor, les spécificités quant aux choix de l'environnement de développement sont à la charge de l'étudiant.

Afin de pouvoir tirer les meilleures analyses et les avantages du WebAssembly, il est primordial de choisir le langage de programmation le plus adapté. Pour cela, il est important de faire un comparatif parmi les différentes solutions. Pour mener à bien cette comparaison les critères suivants sont choisis :

- Compatibilité avec WebAssembly
- Communauté active pour le WebAssembly
- Documentation disponible sur WebAssembly
- Support / help en ligne pour le WebAssembly
- Exemple d'application WebAssembly
- Extension pour le Machine Learning
- Connaissance personnelle

Les points sont attribués de 1 à 5, avec un 1 pour signifier que le critère ne répond pas aux attentes et un 5 pour stipuler que le critère correspond parfaitement aux espérances.

	Compatiblité	Communauté	Documentation	Support	Exemple	Extension	Connaissance	TOTAL
C/C++	5	4	3	3	3	4	2	24
RUST	4	3	2	2	2	2	1	16
C#	5	4	4	4	3	4	4	28
AssemblyScript	5	2	3	2	2	3	1	18

Figure 17 : Tableau comparatif des langages (Illustration réalisée par l'auteur de ce document)

1.14.1. C#

Le langage de programmation choisi pour développer l'application est le C#. Tout d'abord, c'est un des langages les plus matures au niveau de l'intégration avec WASM.

Ensuite, il bénéfice d'une grande communauté très active. De ce fait, il existe un grand nombre de documentations et plusieurs supports d'aide pour le WebAssembly. Nous pouvons également trouver quelques exemples d'applications en C# et WA en naviguant sur le Web.

Pour concevoir une application de classification d'images, il est nécessaire d'exécuter différents algorithmes qui vont se charger de la prédiction. Pour faciliter cette partie, il existe différentes bibliothèques de Machine Learning mise à disposition que nous pouvons utiliser dans ce genre d'application. Et le C# en possède plusieurs qui nous conviennent.

En définitive, comme mentionné à plusieurs reprises, ce travail de Bachelor a comme objectif principal de démontrer les performances de WebAssembly et ses avantages. Afin de se concentrer au mieux sur cette partie, il est intéressant de choisir un langage de haut niveau avec lequel nous avons déjà travaillé et dont nous possédons certaines connaissances. Ce qui est notre cas. C'est également un langage qui reste facilement abordable pour des personnes ayant des connaissances en Java car de nombreuses similarités se retrouvent.

1.14.2. .NET 5

Afin de tirer pleinement des avantages du C#, il est intéressant de l'associer à un framework qui n'est ni plus ni moins une collection de code prêt à l'usage. Il permet aux développeurs de réutiliser du code existant. Tout d'abord, nous allons bénéficier d'un gain de temps. Et ensuite, nous avons une standardisation entre les applications développées en C#.

Le framework .net 5 est une multiplateforme, cela signifie qu'en plus de créer et d'exécuter des applications sous Windows nous pouvons le faire sous Linux et macOS (Chiarelli, 2020).

De nombreuses nouveautés sont introduites dans la nouvelle version 5. L'objectif principal est de faciliter le développement et d'unifier un grand nombre d'application comme celle de bureau, Web, cloud, mobile, Machine Learning et l'intelligence artificielle.

1.14.3. Blazor

Comme présenté dans la partie *1.6.3 C# avec Blazor*, celui-ci permet de créer des applications WA en utilisant .NET et C#. Cela signifie que nous pouvons utiliser l'écosystème existant de ces bibliothèques même pour le développement Web.

Il se base sur un modèle de composants puissants et flexibles afin de développer des interfaces utilisateurs web interactives. Nous devons pour cela implémenter les composants en utilisant une combinaison de code .NET et de syntaxe Razor (mélange de code C# et HTML).

```
@page "/counter"

<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```

Figure 18 : Page Blazor WASM (Illustration réalisée par l'auteur de ce document)

1.14.4. ML.NET

Dans le but d'utiliser l'algorithme de classification d'images de notre application, nous allons employer la bibliothèque ML.NET. C'est une solution open-source proposée par Microsoft et multiplateforme pour le Machine Learning.

Cette bibliothèque propose un vaste choix de scénarios différents comme l'analyse de sentiments, la recommandation de produits, la prédiction de prix, la détection d'objets et de fraudes ou encore la classification d'images (Communauté Microsoft, s.d.).

Pour exécuter un algorithme de classification d'images, il faut des données et plus précisément un modèle de données déjà étiqueté. C'est-à-dire que nous possédons déjà une série d'images classée correctement selon ce qu'elles représentent. Grâce à ce modèle, lorsque nous allons fournir à notre application une nouvelle image, l'algorithme va pouvoir comparer avec les données existantes et faire une prédiction pour la nouvelle image.

Dans le but de créer ces modèles de données, ML.NET met à disposition un outil d'interface utilisateur nommé Model Builder. Celui-ci automatise la création, l'entraînement et le déploiement de ces modèles personnalisés. Il suffit de charger les données et l'outil s'occupe du reste. En plus de créer le modèle entrainé, ML.NET génère le code qui sert à charger le modèle. Celui-ci est sous la forme d'un fichier .zip (Communauté Microsoft, s.d.).

Pour finir, ML.NET est compatible avec TensorFlow, l'un des frameworks les plus prisé pour la construction des modèles de Machine Learning. Il est donc également possible d'utiliser un modèle TensorFlow existant pour démarrer une application de prédiction (Microsoft, 2021).

1.14.5. Model Builder

Comme présenté dans le sous-chapitre précédent, nous avons accès grâce à la bibliothèque ML.NET à un outil d'interface pour générer automatiquement un modèle. Les étapes pour y parvenir sont simples et rapides.

Nous avons le choix entre plusieurs scénarios pour créer un modèle.

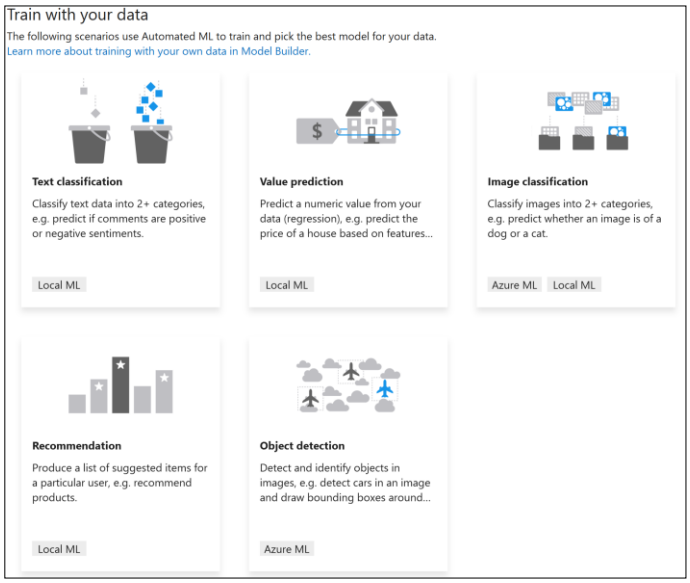


Figure 19 : Model Builder - Choix scénarios (Illustration réalisée par l'auteur de ce document)

La prochaine étape consiste à décider si nous souhaitons entraîner notre modèle de manière local à notre machine ou bien dans le cloud avec Azure[13]. Si nous optons pour la solution locale, nous devons choisir entre deux types de microprocesseurs, le CPU[14] (Central Processing Unit) ou bien le GPU[15] (Graphical Processing Unit). Le chapitre *1.16 CPU ou GPU* est dédié dans ce document à la découverte de leurs différences. Pour choisir l'option avec le GPU, notre ordinateur doit remplir plusieurs critères :

- La compatibilité CUDA[16] (Compute Unified Device Architecture) pour le GPU (Liste des compatibilités : https://developer.nvidia.com/cuda-gpus)
- La version 10.1 de CUDA
- La version 7.6.4 de cuDNN[17] (CUDA Deep Neural Network)

[13] Plate-forme de Microsoft pour gérer le stockage via Internet (cloud en anglais).

[14] Processeur central de l'ordinateur.

[15] Processeur équipant la carte graphique.

[16] Framework pour exploiter la puissance du GPU pour des calculs en parallèle.

[17] Outil indispensable pour utiliser le GPU lors de l'entrainement des réseaux de neurones.

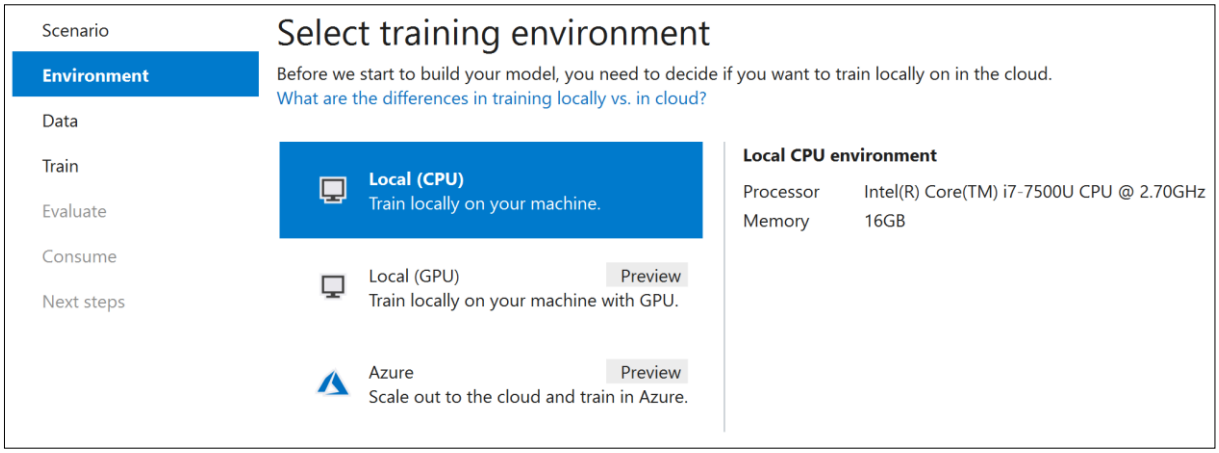


Figure 20 : Model Builder - Choix environnement (Illustration réalisée par l'auteur de ce document)

Ensuite, la partie la plus importante du processus est de fournir des exemples de données. Il suffit pour cela de créer une arborescence précise de dossiers avec à l'intérieur des images. Plus nous pouvons fournir d'images par catégories et plus nous allons obtenir un modèle de qualité.

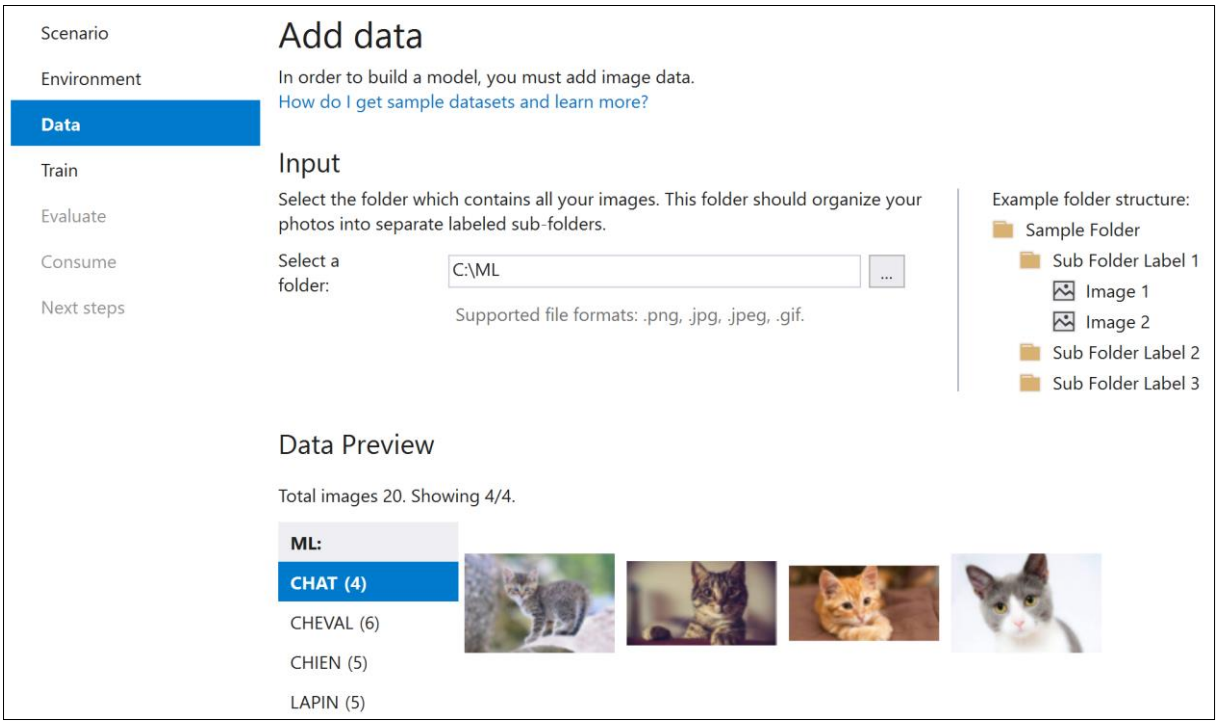


Figure 21 : Model Builder - Choix données (Illustration réalisée par l'auteur de ce document)

Nous avons une confirmation que l'entraînement de notre modèle est terminé.

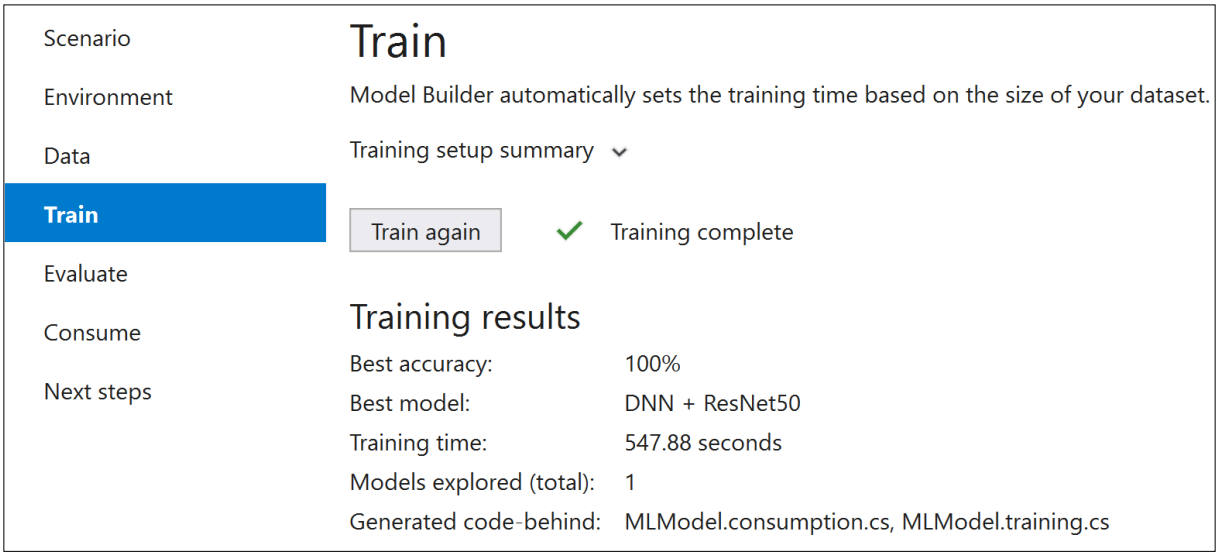


Figure 22 : Model Builder - Entrainement du modèle (Illustration réalisée par l'auteur de ce document)

Avant la dernière étape, nous pouvons directement dans les étapes de création du modèle, l'évaluer. Pour cela, il suffit d'insérer une image pour le tester.

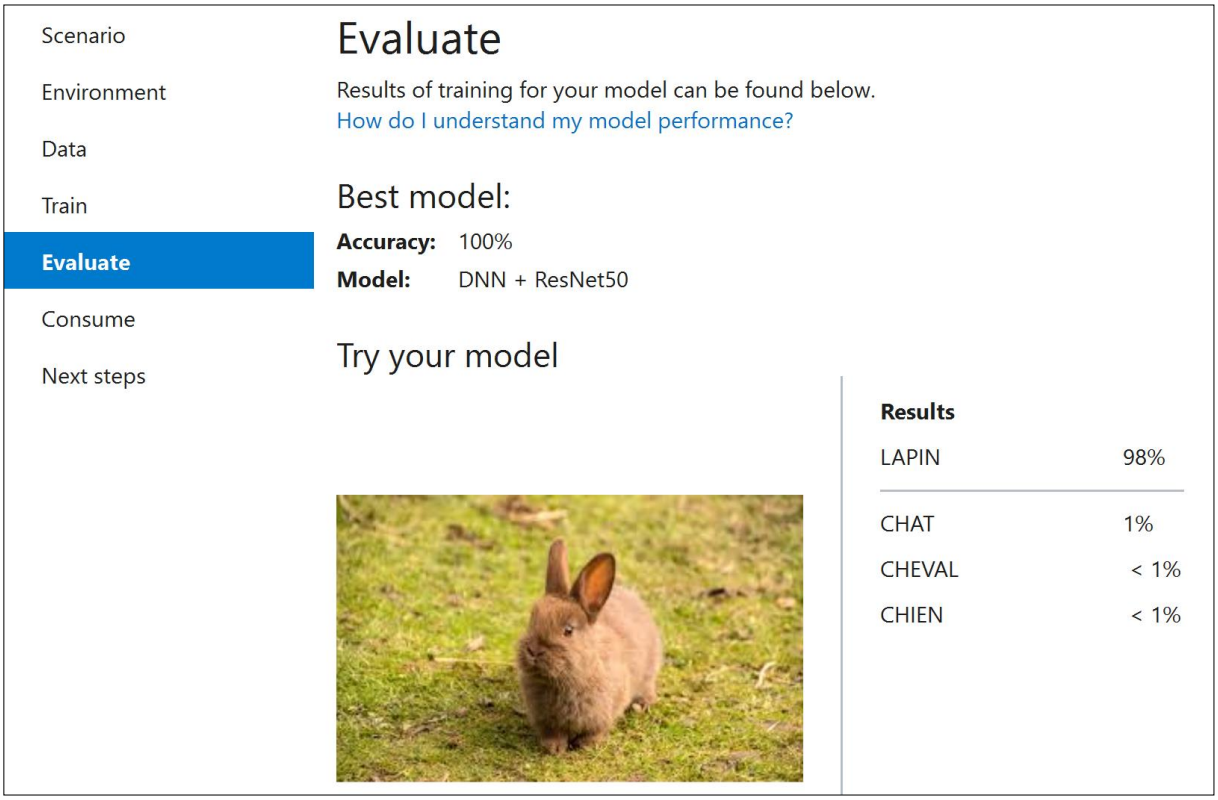


Figure 23 : Model Builder - Evaluation du modèle (Illustration réalisée par l'auteur de ce document)

Dernière étape du processus de création, nous avons la possibilité de générer automatiquement du code pour utiliser le modèle. Nous pouvons choisir entre une application console ou une application Web.

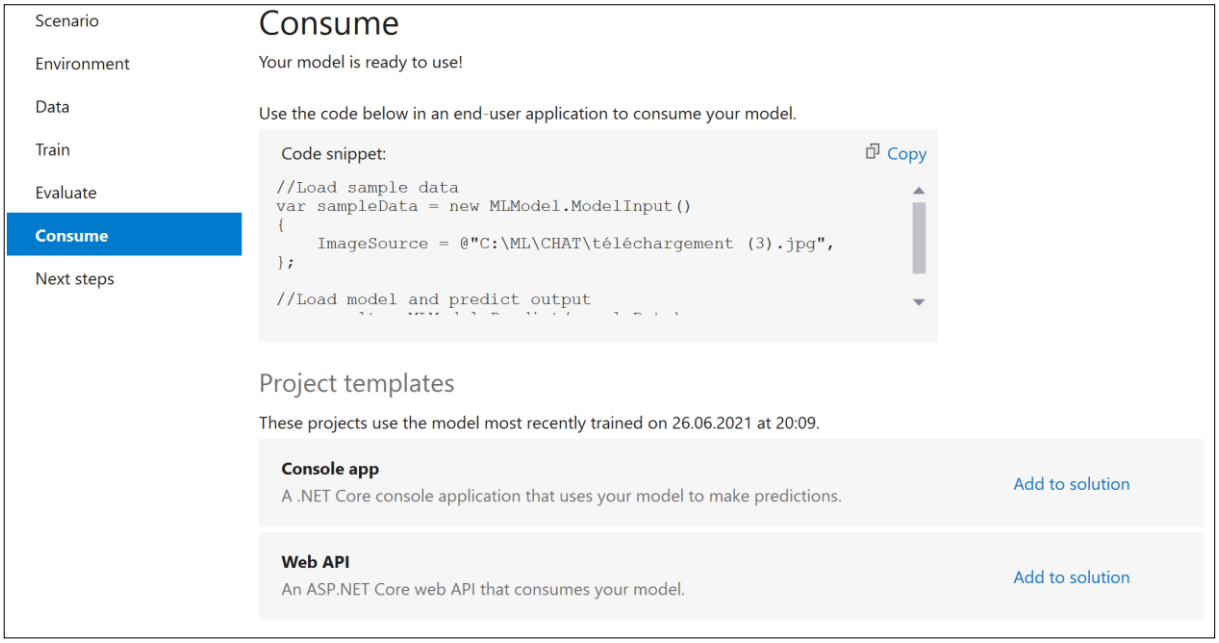


Figure 24 : Model Builder - Génération du code (Illustration réalisée par l'auteur de ce document)

Le code généré correspond à la logique pour accéder au modèle et l'utiliser. Néanmoins, il reste à développer l'API pour les échanges entre l'application client et le serveur ainsi que mettre en place l'interface de l'application avec les différentes pages Web.

1.14.6. Windows Server 2019

Pour déployer l'application à la fin de son développement, la HES-SO Valais a mis à notre disposition une machine virtuelle. Celle-ci tourne sous le système d'exploitation « Windows Server 2019 ». C'est la solution la plus cohérente étant donné que le langage C# fait partie de l'écosystème de Windows.

Cependant, grâce au framework .net 5 avec lequel notre application est réalisée, nous pouvons déployer notre prototype sous plusieurs système d'exploitation. En effet, nous pouvons le faire également avec Linux et macOS.

1.15. Choix méthodologie de travail

Concernant la méthodologie de travail, elle est libre et au choix de l'étudiant. Afin que nous puissions optimiser la gestion de ce projet et de pratiquer les nouvelles connaissances apprises durant cette formation. Il est décidé qu'une version allégée et simplifiée de la méthodologie Agile – SCRUM sera mise en place. De plus, étant donné qu'il s'agit d'un travail réalisé seul, il n'est pas possible de l'appliquer telle quelle dans sa totalité.

Pour suivre et gérer l'évolution du développement de l'application, un Product Backlog (Annexe V) est réalisé au lancement du projet avec le professeur responsable de ce travail de Bachelor, le répondant technique et l'étudiant. Dans celui-ci, nous retrouvons les fonctionnalités qui doivent être implémentées et les buts à atteindre sous forme d'user stories. Lesquels sont soit des objectifs primaires (must have) soit secondaires (nice-to-have).

Ensuite, pour organiser et structurer la partie écrite du travail, un document (Annexe VI) se basant sur l'idée d'un Product Backlog est créé. Cette fois-ci, il comporte les différents éléments et sujets que le rapport doit aborder.

Durant tout le projet et cela chaque deux semaines, une séance en visioconférence est organisée avec le professeur répondant et le répondant technique. L'objectif est de les tenir informés de l'avancement du projet. Ils sont également disponibles pendant et hors ces séances pour amener des éclaircissements sur des sujets pouvant être confus et de ce fait apporter leurs aides en cas de difficultés rencontrées. D'ailleurs, ces réunions ont donné lieu à des discussions très intéressantes qui ont permis de mieux cerner l'objectif de ce travail tout en gardant la bonne direction de ce dernier. Toutes les informations de ces échanges sont répertoriées dans une feuille Excel (Annexe VII) avec les sujets traités et les tâches à effectuer pour la prochaine rencontre.

Pour finir, un journal de bord est régulièrement mis à jour par l'étudiant. De façon qu'une vision globale et chronologique du déroulement de ce travail soit apportée. Les tâches effectuées, les difficultés rencontrées et le temps consacré sont listés.

1.16. CPU ou GPU

Le Machine Learning requiert une immense puissance de calcul comme mentionné dans le chapitre *1.7 Applications utilisant WebAssembly*. Par le passé, les différents workflows[18] se reposaient uniquement sur le CPU pour le traitement des données. Celui-ci n'est pas optimal pour la manipulation de volumes massifs de données.

Grâce au parallélisme, le GPU surpasse aisément le CPU lors de calculs simultanés. Un cœur CPU est plus performant qu'un cœur GPU, mais ce dernier peut être constitué de plusieurs dizaines de milliers de cœurs, par conséquent, il est plus adapté au calcul intensif. À l'inverse, le CPU est limité à quelques dizaines de cœurs.

Afin de démontrer la différence de performance entre CPU et GPU, un article est disponible sur le site elbruno.com. Dans lequel, l'auteur utilise le Model Builder pour entraîner son modèle de données, la première fois avec le CPU et ensuite avec le GPU de sa machine (Bruno, 2020).

Pour le test de performance, 22'424 images sont utilisées pour l'entrainement du modèle de données, ce qui représente 1GB en volume de stockage.

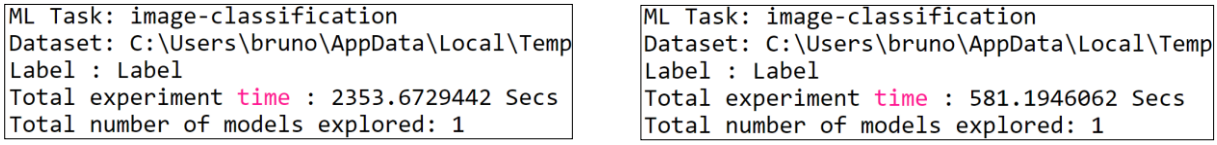


Figure 25 : Comparaison CPU vs GPU
https://elbruno.com/2020/09/09/vs2019-ml-net-model-builder-gpu-vs-cpu-test-4-times-faster/

Nous voyons avec ces résultats qu'en utilisant le GPU nous obtenons une performance quatre fois meilleures qu'avec le CPU.

[18] Processus séquentiel pour accomplir une tâche.

2. Résultats

2.1. Concept de l'application

Comme mentionné dans le chapitre *1.14 Choix environnement de développement*, l'outil Blazor permet de développer des applications WebAssembly et la bibliothèque ML.NET donne les outils nécessaires à l'intégration du Machine Learning dans ces dernières.

Dans la première version du projet, un modèle TensorFlow devait être utilisé pour procéder à la prédiction. Cette solution donnait accès à un modèle déjà entraîné et étiqueté pour la classification de plus de mille images différentes. Afin de l'utiliser, un fichier compressé (.zip) avec toute la logique et un fichier texte (.txt) devait être employé directement par l'application. Avec un tel concept, le fichier texte comporte la liste des différentes catégories d'images que l'algorithme est en mesure de classifier. Nous avons constaté durant le déroulement du projet que cette solution n'est malheureusement pas réalisable. La contrainte des accès fichiers du WASM et la limitation du choix des paramètres d'entrée de la méthode pour exécuter la classification empêche le succès de cette approche. Nous revenons en détail sur les différents problèmes rencontrés dans le chapitre *3 Discussion*.

Pour réussir à illustrer ce travail avec un prototype fonctionnel, le choix d'un changement de concept et la mise en place d'une API est fait. Le modèle de données utilisé dans la deuxième version du projet est entrainé directement par l'outil nommé Model Builder faisant partie de la bibliothèque ML.NET. Le déroulement de son entrainement et son intégration au prototype est détaillé avec des illustrations dans la partie *1.14.5 Model Builder*.

En revanche, avec la mise en place d'une API pour la classification notre application perd le concept principal du WASM. Comme mentionné à plusieurs reprises, le WebAssembly permet d'exploiter des applications qui sont téléchargées et exécutées entièrement du côté client sans connexion constante avec un serveur. L'objectif est que ça soit les ressources de la machine de l'utilisateur qui sont utilisés et non celle du serveur. Avec notre deuxième concept, le processus de classification ne se déroule pas sur l'ordinateur du client mais sur le serveur. De plus, lorsqu'une API est utilisée, une connexion ainsi que des communications ont lieu entre la machine du client et le serveur.

Le principe de l'application est basique. Il permet de faire une classification d'images. Une illustration numérique est choisie par l'utilisateur depuis ses fichiers locaux de son ordinateur. Celle-ci est alors transmise à l'aide de l'API au serveur sur lequel une méthode exécute la classification. À la fin de ce processus, lorsque l'image est attribuée à la bonne catégorie, le résultat est envoyé sur la machine du client. Celui-ci est alors affiché sur l'application.

2.2. Fonctionnement de l'application

Nous avons été contraints de modifier le concept du prototype pendant la phase de développement de celui-ci comme détaillé dans le sous-chapitre ci-dessus. Dans un premier temps, nous allons présenter les principes de fonctionnement de la première version de l'application et ensuite la deuxième.

2.2.1. Première version

Nous avons révélé dans la partie *1.10 Convertir Blazor Server vers WebAssembly* lorsqu'une application est développée en respectant la logique de Blazor, il est facile de la convertir vers du WASM. D'ailleurs, cette procédure peut s'effectuer dans les deux sens sans grande difficulté.

À la suite de nos recherches, nous découvrons un article de M. Daniel Jimenz Garciais, développeur de logiciels, dans lequel il décrit les différentes étapes nécessaires à la réalisation d'une application Blazor serveur. Nous décidons dans un premier temps, de suivre ses recommandations dans le but de développer une application Blazor serveur et de la convertir par la suite.

Le fonctionnement de cette version se devait simple et basique. L'utilisateur choisit une image qu'il souhaite classifier dans une des multiples catégories disponibles dans le modèle TensorFlow. Celle-ci est sélectionnée localement à partir des différents dossiers de son ordinateur. Avant de procéder à la classification, l'illustration numérique est convertie en base64[19] afin de pouvoir l'afficher dans l'interface de l'application. Ensuite, le contenu de l'image est téléchargé dans un flux de mémoire. Cela facilite son transport à travers les

[19] Type d'encodage pour les données binaires.

différentes couches de l'application. Lorsque ce flux de mémoire parvient jusqu'à la méthode de classification, ce dernier est converti en Bitmap et l'algorithme peut s'exécuter. Pour finir, la probabilité la plus plausible est récupérée et affichée sur l'application.

Dans la version serveur, l'accès aux fichiers .zip et .txt est effectué à l'aide d'une classe. À l'intérieure de cette dernière, une méthode récupère les informations nécessaires des répertoires dont les chemins d'accès à ces deux fichiers. Pour rappel, avec une application classique, il est permis d'accéder aux ressources fichiers du système.

Pour approfondir le principe de fonctionnement de cette première version de notre prototype, nos différentes explications sont illustrées à l'aide de code dans l'annexe III.

Lorsque nous avons terminé le développement de l'application serveur, nous avons procédé à la conversion de celle-ci en WebAssembly. Durant laquelle, plusieurs problèmes et défis nous ont finalement contraint à modifier le concept du prototype. Ces difficultés sont détaillées dans les différents sous-chapitres de la partie *3.2 Problèmes* de ce document.

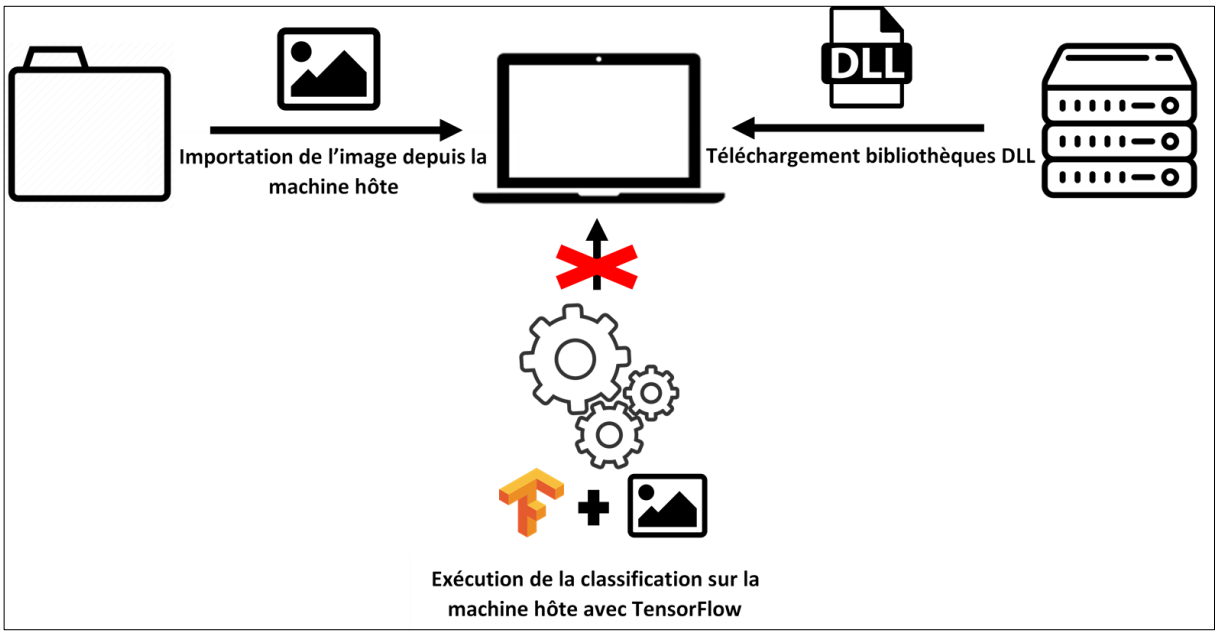


Figure 26 : Architecture du système de l'application - Première version (Illustration réalisée par l'auteur de ce document)

2.2.2. Deuxième version

Dans la deuxième version de l'application une API est mise en place pour contrer la contrainte de WA quant à l'accès fichier nécessaire à la classification. De plus, nous décidons d'utiliser le Model Builder pour la création du modèle entrainé. Pour terminer, nous commençons à développer le prototype directement en WASM.

Pour nous aider dans la réalisation de cette application, nous nous documentons sur ce nouveau concept à l'aide de deux vidéos proposées par le développeur Biswa Ranjan. Ces dernières sont disponibles sur la plateforme YouTube. Dans la première, il enseigne comment créer l'interface nécessaire à importer une image dans l'application par l'utilisateur. Dans la seconde, il présente le déroulement pour la création d'un projet Blazor serveur avec une API.

Au niveau de son fonctionnement, des similitudes avec la première version subsiste. En effet, avant d'être transmise au serveur, les images sont redimensionnées et converties en base64. Ensuite, elles peuvent être envoyées à l'aide d'une requête « POST» [20] d'API vers le serveur. Les méthodes pour la classification possèdent des limitations quant à leurs paramètres d'entrées pour s'exécuter. Effectivement, ces dernières exigent un chemin d'accès pour employer les images. Il est donc nécessaire de les stocker le temps de la procédure sur le serveur pour débuter la classification. À la fin de celle-ci, le résultat est renvoyé à l'aide de l'API sur la machine du client pour être affiché sur l'application.

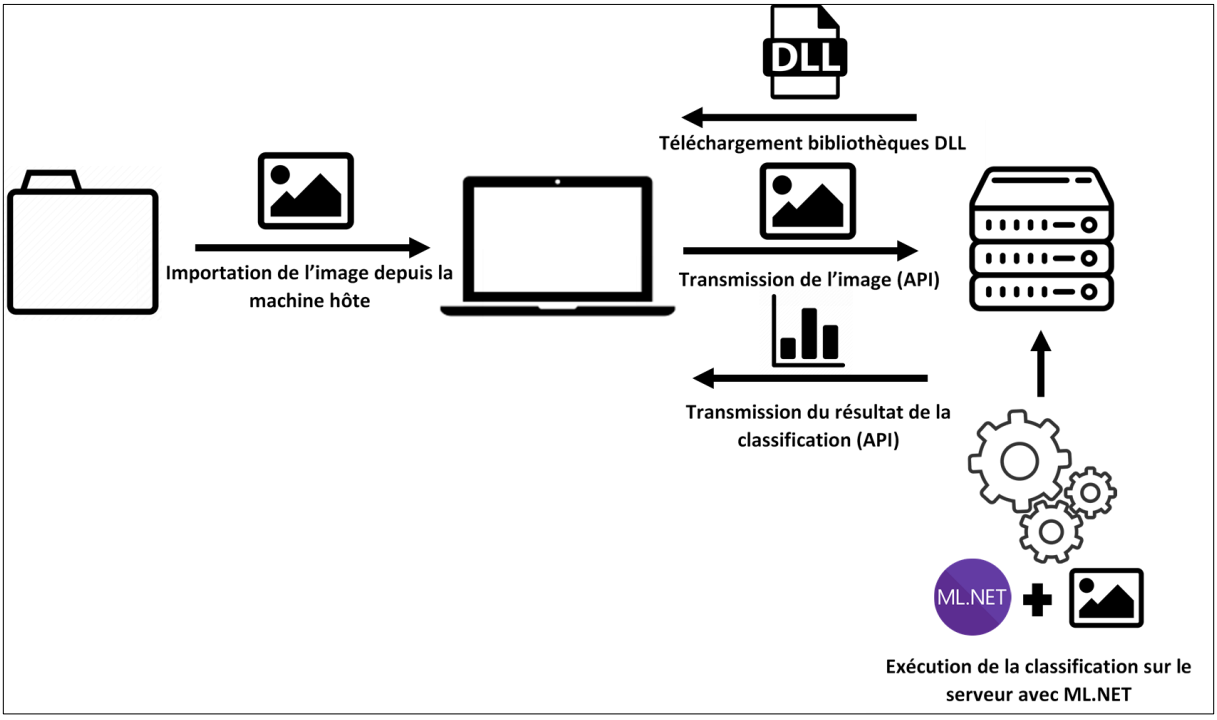


Figure 27 : Architecture du système de l'application - Deuxième version (Illustration réalisée par l'auteur de ce document)

Concernant les accès aux fichiers de cette deuxième version, ils sont créés automatiquement par le Model Builder à la fin du processus de création du modèle entrainé.

[20] Requête permettant d'envoyer des données au serveur.

L'annexe IV est à disposition à la fin de ce rapport dans le but d'enrichir nos propos de ce sous-chapitre.

2.3. Visuel de l'application

Comme mentionné à plusieurs reprises, le prototype se doit d'être simple. C'est pour cela, que nous retrouvons deux pages basiques sur l'application. La première est une page d'accueil sur laquelle nous retrouvons des indications brèves sur le projet. Et sur la deuxième, nous pouvons procéder à une classification d'images. Dans laquelle, à la fin du processus, la prédiction est affichée.

Figure 38 : Page accueil de l'application (Illustration réalisée par l'auteur de ce document)

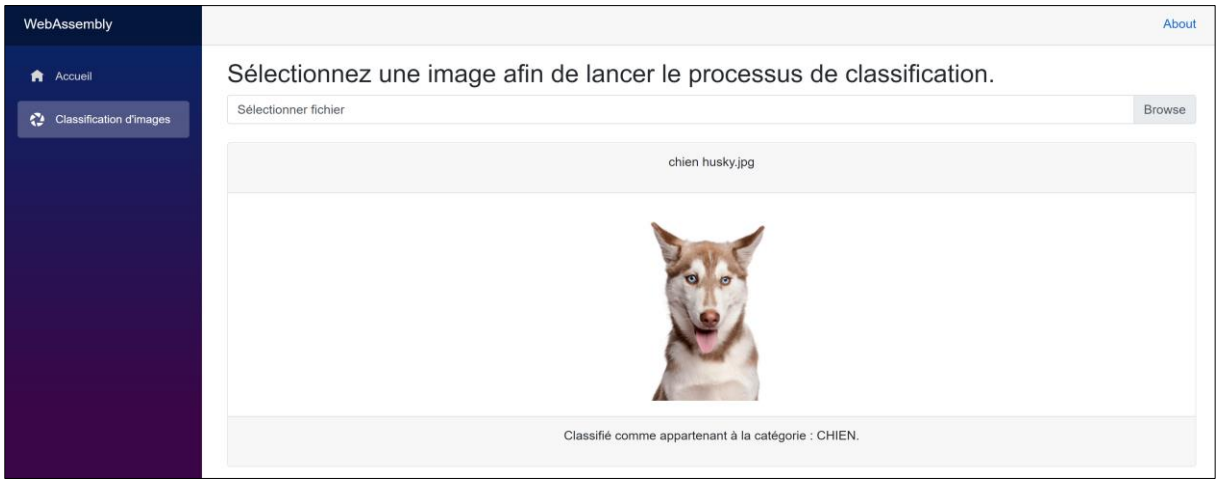


Figure 39 : Page classification d'images (Illustration réalisée par l'auteur de ce document)

2.4. Temps d'initialisation

Une application Blazor WebAssembly et une Blazor Server se distinguent au niveau de la vitesse d'initialisation dans le navigateur. Nous allons observer et relever les nuances lors du démarrage de celles-ci. Pour cela, deux prototypes sont développés avec les différentes fonctionnalités par défaut proposé lors de la création avec Blazor.

2.4.1. Blazor server

Name	Stat...	Type	Initiator	Size	Time
site.css	200	stylesheet	(index)	672 B	61 ms
ServerApplication.styles.css	200	stylesheet	(index)	1.2 kB	61 ms
open-iconic.woff	200	font	open-iconic-bootstrap.min.css	15.0 kB	16 ms
open-iconic-bootstrap.min.css	200	stylesheet	site.css	2.6 kB	48 ms
negotiate?negotiateVersion=1	200	fetch	blazor.server.js:1	349 B	13 ms
localhost	200	document	Other	1.7 kB	28 ms
favicon.ico	200	x-icon	Other	5.5 kB	41 ms
bootstrap.min.css	200	stylesheet	(index)	35.5 kB	178 ms
blazor.server.js	200	script	(index)	96.7 kB	126 ms
_blazor?id=uRV4xl4KAV4eTpa1p1-Q5g	101	websocket	blazor.server.js:1	0 B	Pending

10 requests | 159 kB transferred | 468 kB resources | Finish: 384 ms

Figure 28 : Initialisation application Server (Illustration réalisée par l'auteur de ce document)

Lorsque nous démarrons l'application Blazor Server nous téléchargeons une série de composants depuis le serveur. D'abord, nous avons des fichiers CSS, HTML et des icônes pour gérer le rendu visuel. Ensuite, nous avons un fichier JavaScript qui va initialiser et s'occuper de la connexion SignalR pour les échanges. Finalement, nous observons un WebSocket[21] qui est en attente pour procéder à des échanges de messages binaires. À chaque interaction d'un utilisateur sur l'interface graphique de l'application un message est envoyé au serveur et un résultat est retourné.

Nous constatons qu'un total de 10 requêtes HTTP pour 468 Kilobyte et un temps de 384 millisecondes sont nécessaires pour lancer une application serveur. À chaque démarrage, les mêmes composants que ceux présentés ci-dessous sont téléchargés sur la machine du client.

[21] Protocole réseau qui définit la manière dont les données sont échangées dans le Web.

2.4.2. Blazor WebAssembly

Entre le premier démarrage et les suivants d'une application WebAssembly, ce ne sont pas les mêmes fichiers qui sont téléchargés. Pour commencer, lors du premier lancement nous retrouvons des fichiers CSS et HTML, ainsi que des fichiers Javascript qui gèrent les dépendances de l'application et l'initiation du runtime. Ensuite, nous avons un fichier .json dans lequel nous retrouvons les références pour l'ensemble des bibliothèques de l'application à télécharger. Finalement, nous avons une série de ces bibliothèques en question, avec l'extension .dll.

À l'opposé d'une application serveur, il convient d'utiliser une plus grande quantité de ressources à l'initialisation. Un total de 203 requêtes HTTP pour 22.8 Mégabyte et un temps de 3.7 secondes sont indispensables pour démarrer une application WA. Il est important de relever que plus l'application est complexe et grande, plus les ressources et le temps de lancement sont élevés.

Name	Status	Type	Initiator	Size	Time
localhost	200	document	Other	643 B	87 ms
bootstrap.min.css	200	stylesheet	(index)	35.5 kB	105 ms
app.css	200	stylesheet	(index)	698 B	97 ms
WebAssemblyApplication.styles.css	200	stylesheet	(index)	1.2 kB	97 ms
blazor.webassembly.js	200	script	(index)	27.9 kB	99 ms
open-iconic-bootstrap.min.css	200	stylesheet	app.css	2.6 kB	15 ms
blazor.boot.json	200	fetch	blazor.webassembly.js:1	19.4 kB	10 ms
favicon.ico	200	x-icon	Other	5.5 kB	18 ms
dotnet.5.0.6.js	200	script	blazor.webassembly.js:1	63.0 kB	73 ms
Microsoft.AspNetCore.Authorization.dll	200	fetch	blazor.webassembly.js:1	21.1 kB	682 ms
Microsoft.AspNetCore.Components.dll	200	fetch	blazor.webassembly.js:1	75.3 kB	1.39 s
203 requests | 9.2 MB transferred | 22.8 MB resources | Finish: 3.70 s					

Figure 29 : Temps de la première initialisation application WASM (Illustration réalisée par l'auteur de ce document)

En revanche, lors du second et suivant démarrage de l'application sur la même machine du client, la collection de bibliothèques n'est pas retéléchargée. En effet, celles-ci sont sauvegardées dans la mémoire cache (cache storage en anglais) du navigateur lors du premier lancement et sont réutilisées lors des prochaines exécutions. Grâce à cela, nous distinguons une diminution des requêtes HTTP, taille de fichiers téléchargés et temps d'initiation.

Name	Status	Type	Initiator	Size	Time
app.css	200	stylesheet	(index)	698 B	95 ms
blazor.boot.json	200	fetch	blazor.webassembly.js:1	19.4 kB	44 ms
blazor.webassembly.js	200	script	(index)	27.9 kB	124 ms
bootstrap.min.css	200	stylesheet	(index)	35.5 kB	126 ms
dotnet.5.0.6.js	200	script	blazor.webassembly.js:1	63.0 kB	38 ms
favicon.ico	200	x-icon	Other	5.5 kB	41 ms
localhost	200	document	Other	686 B	71 ms
open-iconic-bootstrap.min.css	200	stylesheet	app.css	2.6 kB	40 ms
open-iconic.woff	200	font	open-iconic-bootstrap.mi...	15.1 kB	7 ms
WebAssemblyApplication.styles.css	200	stylesheet	(index)	1.2 kB	107 ms

10 requests | 172 kB transferred | 528 kB resources | Finish: 1.79 s

Figure 30 : : Temps de la deuxième initialisation de l'application WASM (Illustration réalisée par l'auteur de ce document)

Cependant, il est important de relever que lorsque nous naviguons dans l'interface et interagissons avec les composants de l'application, les temps d'exécution sont plus rapides que ceux sur une application serveur. Ce gain de temps est le résultat du téléchargement de toutes les bibliothèques de l'application sur la machine du client. De ce fait, aucun échange avec le serveur n'est nécessaire avec une application basique WASM et donc, aucune latence.

2.5. Performance

La présentation des résultats de notre prototype n'est malheureusement pas pertinente. En effet, l'utilisation d'une API pour la classification a éteint le principe du WebAssembly et fausse ses résultats. Afin de démontrer les performances d'une application WA, nous allons exploiter et détailler les résultats de M. David Grace dans son article « How Blazor Performs Against Other Frameworks » sur le site Web « Progress Telerik » (2021).

Les performances des applications ASP.NET Core MVC, Blazor Server et Blazor WebAssembly sont évaluées. Dans l'objectif d'avoir une comparaison pertinente, ces applications sont développées et implémentées avec des fonctionnalités similaires et avec la même base de données. L'expérience se concentre sur le temps d'initialisation des applications ainsi que celui pour recevoir et afficher des données. Pour cela, une base de données est mise en place avec dix milles articles dont dix qui sont chargés au démarrage de chaque application et dix autres par la suite en naviguant sur cette dernière.

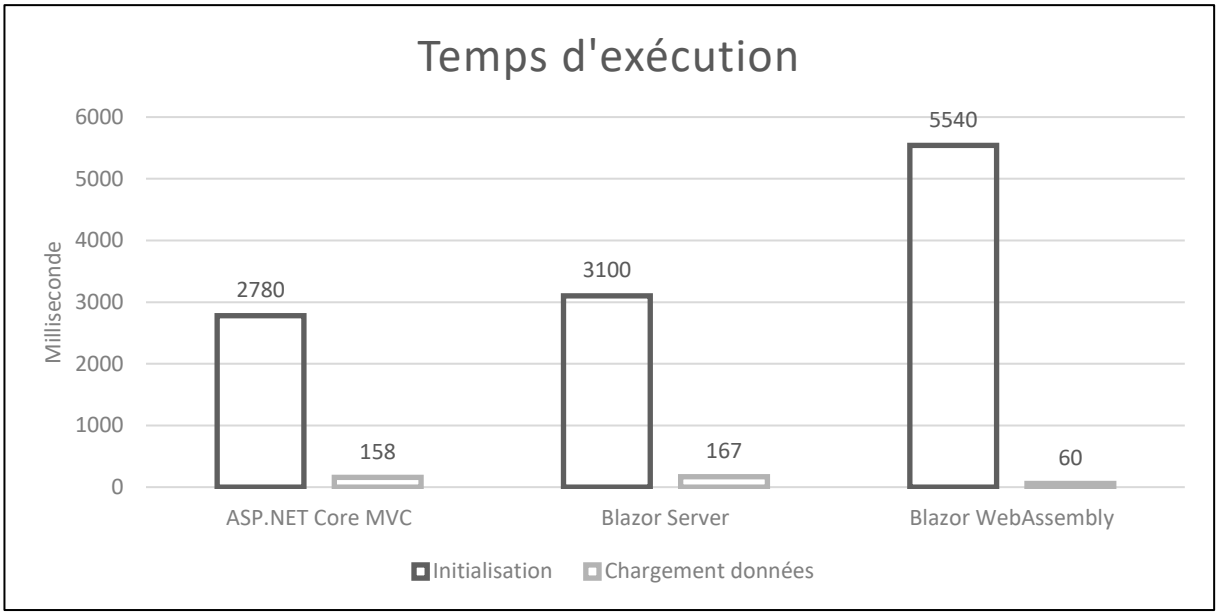


Figure 31 : Comparatif de performance (Illustration réalisée par l'auteur de ce document)

La première application, ASP.NET Core MVC, met moins de 2.8 secondes pour s'initialiser en chargeant des fichiers CSS et HTML nécessaires pour l'interface utilisateur. Le temps de chargement pour les données est nettement plus rapide, il suffit de 150 millisecondes pour effectuer cette tâche.

Le temps pour initialiser l'application Blazor Server est légèrement plus long, il est nécessaire d'attendre 3.1 secondes. Cette différence est due à l'obligation d'établir une connexion WebSocket entre le navigateur et le serveur pour la communication. Cela réduit considérablement le temps de chargement lors de l'utilisation de l'application et la quantité de bande passante employée. Concernant le temps de chargement des données, il est délicat d'obtenir une estimation précise. En effet, nous n'utilisons plus de requêtes supplémentaires pour recevoir des données mais bel et bien le WebSocket qui fournit une connexion constante entre le navigateur et le serveur. Afin de présenter une approximation du temps de chargement dans l'expérience, une class « Stopwatch » de .NET est mise en place pour mesurer ce temps. Celui-ci se monte à 167 millisecondes.

Pour terminer, le temps d'initialisation de l'application WASM est de 5.54 secondes. Dans l'intention de récupérer les articles depuis la base de données, une API est mise en place dans la démonstration de M. Grace. Avec elle, le temps de chargement pour les articles est de 60 millisecondes, un temps bien meilleur que ses concurrents.

Pour consulter les détails d'exécution des applications, voir annexe IX.

3. Discussion

3.1. Interprétation des résultats

Les performances d'applications présentés dans ce document et celles trouvées sur le Web peuvent varier à la suite de plusieurs conditions.

Premièrement, le nombre de requêtes HTTP ou la taille de celles-ci jouent un rôle sur la rapidité de l'application. Ces dernières augmentent ou diminuent selon le type et la complexité de l'application. En effet, un projet WA regroupant d'importantes quantités de fonctionnalités voit son temps d'initialisation s'accroître. Cela fait suite à la quantité de fichiers .dll téléchargée.

Ensuite, comme indiqué, les applications WebAssembly tournent sur l'ordinateur du client. Les performances varient selon la puissance de la machine de l'utilisateur. C'est pourquoi, une divergence de vitesse peut déjà apparaître sur la même application WASM mais utilisée sur deux terminaux différents. La puissance du processeur ainsi que les différents composants de l'ordinateur jouent un rôle sur la rapidité des exécutions. De plus, les mêmes nuances apparaissent sur un serveur. Les performances de celui-ci varient selon sa puissance. Toutes ces diversités doivent être considérées lors d'une comparaison de performance.

Pour la dernière divergence, cette étude démontre la compétitivité des applications pour une seule initialisation et un seul chargement de données. Dans le cas où l'utilisateur effectue plusieurs chargements de données à la suite, les résultats vont évoluer. Un autre cas peut même intervertir le classement si un utilisateur relance plusieurs fois de suite l'application. En effet, comme expliqué dans le chapitre *2.4 Temps d'Initialisation*, à partir de la seconde fois une application WASM réduit son temps d'initialement. À l'inverse, les autres applications conservent le même temps d'exécution.

Pour conclure ces comparaisons, nous constatons que les vitesses d'exécution et les performances générales d'un projet WA fluctuent rapidement à la suite de plusieurs critères. Les ressources matérielles et la façon d'utiliser l'application jouent un rôle primordial dans ces différences. Avant de trancher pour une solution WASM, il est donc impératif de prendre en considération ces différentes nuances.

3.2. Problèmes

Durant la réalisation de ce travail de Bachelor, plusieurs difficultés se sont présentées à nous. Les plus conséquentes ont émergé lors du développement du prototype. Certaines ont été faciles à résoudre et d'autres ont nécessitées des recherches plus conséquentes. Le dernier problème mentionné dans les sous-chapitres suivants, nous a même obligé à modifier le concept de l'application

3.2.1. Compatibilité ML.NET

Afin d'utiliser pleinement les fonctionnalités de la bibliothèque ML.NET, il est impératif de veiller à la compatibilité de plusieurs éléments.

Premièrement, il doit être associable au bon framework. Le langage C# a la possibilité de cibler plusieurs bibliothèques . Nous avons le choix entre .NET standard 2.1 avec ses versions supérieurs et .NET Core 3 ou sa nouvelle mise à jour .NET 5. Pour avoir accès à la totalité des fonctionnalités et méthodes mises à disposition par ces bibliothèques, nous devons adopter le deuxième choix. Dans notre projet, nous optons pour la nouvelle version .NET 5. Elle présente plusieurs avantages que nous avons déjà détaillés dans la partie *1.14.2 .NET 5*.

Ensuite, l'application doit obligatoirement être paramétrée pour pointer explicitement une architecture de processeur en x64 ou x86. Si nous laissons le paramétrage par défaut qui correspond à la valeur « Any CPU » l'application n'est pas en mesure de compiler.

Les deux éléments présentés dans cette partie ont levé des exceptions lors du lancement de l'application. Il est donc judicieux de veiller à la bonne compatibilité des différentes bibliothèques et outils utilisés.

3.2.2. Incompatibilité Bitmap

Dans la première version de l'application, il était primordial de procéder à quelques manipulations de l'image avant de l'utiliser. En effet, pour arriver à exécuter l'algorithme de classification, la valeur d'entrée dans notre cas l'image doit obligatoirement correspondre à une taille en pixel bien précise. Ce critère est défini lors de la création du modèle. Vu que nous utilisions un modèle TensorFlow déjà entrainé et préparé, nous devions redimensionner celle-

ci à 224x224 pixel. Étant donné que nous recevions l'image choisit par l'utilisateur comme une instance de mémoire, nous avons décidé dans un premier temps de la transformer en un Bitmap. C'est la solution la plus répandue et la plus couramment utilisée.

Malheureusement, Blazor WebAssembly n'arrive pas à gérer un Bitmap. Pour remédier à ce problème, il est nécessaire d'utiliser une autre bibliothèque graphique. Il en existe une multitude sur le marché compatible avec C# et avec un projet WA. Dans notre cas, nous avons choisi ImageSharp. Elle possède toutes les fonctionnalités nécessaires au traitement d'images.

3.2.3. Injection du service pour la classification

Dans le but de pouvoir exécuter une classification dans la première version de notre prototype, il était impératif de mettre en place un « Prediction Engine Pool ». En d'autres termes, cela correspond à une injection de dépendances. Pour y parvenir, nous devons employer des services.

Un service constitue en un composant réutilisable. Celui-ci est déclaré dans la méthode « main » de la classe « Program ». Ce sont les premiers éléments qui sont exécutés en arrière-plan lors du démarrage du prototype. Enfin, leurs objectifs sont d'offrir diverses fonctionnalités accessibles depuis l'ensemble de l'application.

Dans notre cas, le service est affecté avec la méthode AddPredictionEnginePool. Cette dernière doit recevoir plusieurs éléments en paramètre d'entrée. D'abord, nous spécifions quelles classes doivent être utilisées comme éléments d'entrée et de sortie pour l'algorithme de classification. Ceux-ci correspondent à l'image pour le paramètre d'entrée et le résultat de la classification pour celle de sortie. Pour finir, nous devons spécifier l'emplacement dans lequel la méthode va pouvoir trouver le modèle entrainé TensorFlow. Celui-ci équivaut à notre fichier .zip.

Afin de fournir ce fichier à la méthode, nous avons la possibilité entre seulement deux solutions. Ces dernières sont des extensions de méthode se nommant « FromFile » et « FromUri ». La première requière un chemin du système de fichier local. C'est-à-dire, d'accéder directement au fichier se trouvant dans les répertoires de la machine du client ou celui du serveur dans une application classique. La deuxième alternative est de fournir l'accès avec une

URI[22] (Uniform Resource Identifier). Avec ce choix, le fichier .zip doit obligatoirement être une ressource intégrée. Son emplacement doit se situer dans le dossier wwwroot de l'application.

Malheureusement, lors du démarrage de l'application WASM des erreurs sont décelées et une exception se lève sur la déclaration de ce service en question. Comme présenté dans la partie *1.3.4 Sécurité* du chapitre *1.3 Caractéristiques de développement*, WA reste limité pour l'instant quant aux accès de fichiers. Le prototype n'arrive pas à y accéder. Spécifiant alors que le chemin précisé est introuvable. À la suite de quoi, plusieurs DLL ne parviennent pas à être téléchargées. Pour rappel, lors de l'exécution d'un projet WASM, toutes les bibliothèques doivent être chargées correctement sur la machine du client.

De multiples recherches ont été menées, en vain, afin de contourner l'obligation d'utiliser un « Prediction Engine Pool » pour une classification d'images avec un modèle de données TensorFlow.

3.3. Critiques personnelles

Le dernier problème mentionné dans le sous-chapitre ci-dessus reste la plus grande difficulté de ce travail. Il nous a obligé à trouver une alternative pour le développement de l'application. Suite à cela, notre prototype a perdu le concept principal du WebAssembly. En effet, la classification ne s'exécute plus sur la machine du client comme souhaité mais bel et bien sur le serveur.

Le WebAssembly reste une technologie récente. Certes, il possède une grande quantité de documentation et une communauté active mais son concept demeure relativement théorique sur les différents forums. À cause de cela, il est délicat de se procurer du contenu de qualité pour faciliter le développement d'une application. Cette contrainte greffe une difficulté conséquente pour la réussite d'un projet WASM.

De plus, un prototype pour la classification d'images reste une réalisation complexe avec un modèle de conception particulier. En effet, l'application possède une logique distincte comme l'accès à un fichier .zip pour son bon fonctionnement. De ce fait, les différents

[22] C'est un identifiant pour une ressource se trouvant sur un réseau informatique.

contenus trouvés pour notre prototype doivent correspondre à la logique WA et celle du Machine Learning.

Toutes ces particularités ajoutent de la délicatesse au bon déroulement d'un projet WASM.

Conclusion

Synthèse générale

L'idée initiale de ce travail consistait à découvrir le standard WebAssembly. Ensuite, nous devions évaluer sa faisabilité à s'intégrer dans une application de classification d'images. Sans oublier que celle-ci se devait de s'exécuter entièrement sur la machine du client. En effet, c'est la fonctionnalité principale du WASM.

Dans un premier temps, nous avons procédé à de multiples recherches afin de dresser un état de l'art sur le sujet. Cela nous a permis de comprendre son concept ainsi que ses différentes caractéristiques. Nos analyses quant aux choix de l'environnement de développement ainsi que notre enquête sur les différentes applications utilisant WASM, nous ont démontré l'étendu de sa popularité. Celle-ci est relativement bonne même si les supports disponibles sur le plan pratique sont plus rares. Ensuite, durant la phase de développement, les sous-chapitres dédiés aux différences entre un projet Blazor Server et WebAssembly nous ont été d'une aide précieuse. En effet, nous connaissions au préalable les particularités à prendre en considération lors de la création du prototype.

Ensuite, les différentes difficultés rencontrées pendant le développement de l'application, nous ont conduit à approfondir d'avantages nos connaissances sur le plan pratique du WASM. De plus, nous avons également pu définir plus aisément les limites actuelles d'implémentation de ce standard dans des projets concrets. Effectivement à l'heure actuelle, l'utilisation de WA est surtout intéressante avec des applications basiques. Plus ces dernières ne nécessitent pas une logique particulière comme des accès fichiers par exemple et plus sa réalisation sera concevable. Cependant, il est important de relever que le WebAssembly est en constante évolution. En effet, différentes recherches sont menées pour améliorer et faciliter son fonctionnement ainsi que son utilisation dans des projets plus complexes.

Pour finir, notre analyse des résultats sur les différentes vitesses d'exécutions, nous a permis de découvrir dans quel scénario le WebAssembly est bénéfique. Contraint de constater, que plusieurs facteurs doivent être pris en considération pour cela. D'abord, vu le temps d'initialisation d'un prototype WA lors du premier lancement, il est essentiel que l'utilisateur emploi l'application à maintes reprises. Ensuite, ce dernier doit également exécuter de multiples calculs et requêtes car c'est seulement durant la navigation que les performances sont meilleures sur un projet WASM. Pour finir et sans oublier, qu'il est intéressant que la machine sur laquelle l'application est téléchargée possède des composants performants dans le but d'optimiser l'expérience utilisateur. Tous ces différents paramètres doivent être appliqués afin de tirer pleinement avantages du WebAssembly.

Retour d'expérience

Grâce à nos différentes recherches et notre retour d'expérience sur le développement d'un projet WASM, nous estimons que ce standard est une solution prometteuse. Cependant, il est préférable que celle-ci mûrisse d'avantages pour palier à ces différentes limitations.

Dans notre cas, lors du développement de notre prototype, nous devions veiller à ce que plusieurs facteurs soient compatibles entre eux. D'abord, le principe d'une application de Machine Learning pour une classification d'images devait être faisable en WASM. Ensuite, nous étions obligés de vérifier que le langage choisit permettait de respecter ces différents principes. De ce fait, les contraintes liées à la logique du Machine Learning, du WA et du langage réduisait considérablement le contenu pour notre apprentissage disponible sur le Web ainsi que la faisabilité du prototype.

Pour répondre à la question si le WebAssembly réduit la charge de calcul sur un serveur, nous estimons qu'il le fait. À conditions, que l'application soit téléchargée et exécutée entièrement sur la machine du client. Avec notre prototype, cette condition n'est pas respectée. Effectivement, nous avons le désavantage du WA avec une initialisation longue et des communications avec le serveur pour exécuter la classification d'images.

Améliorations

Dans l'état actuel, les calculs de notre prototype ne sont pas exécutés sur l'ordinateur de l'utilisateur. Pour remédier à cela, il peut être intéressant de recourir au projet WASI lors de son officialisation. En effet, celui-ci autorise les accès fichiers dans une application WebAssembly.

Pour conclure, il est envisageable de se tourner vers un langage de programmation autre que le C# pour la réalisation d'un projet de classification d'images en WA. À la suite de la récente présentation de Pyodide, solution permettant de compiler une application dans le navigateur en Python. Cette alternative semble être plus adéquate à la réalisation d'un projet d'apprentissage automatique. Car effectivement, le langage Python semble faire l'unanimité auprès des développeurs d'applications de Machine Learning.

Références

Aboukhalil, R. (2019, Août 28). *Beyond The Browser: Getting Started With Serverless WebAssembly*. Récupéré sur Smashing magazine: https://www.smashingmagazine.com/2019/08/beyond-browser-serverless-webassembly/

B, T. (2020, Juin 4). *Tout savoir sur le Deep Learning*. Récupéré sur DataScientest: https://datascientest.com/deep-learning

Ball, K. (2017, Décembre 5). *WebAssembly est maintenant supporté sur tous les navigateurs*. Récupéré sur Info Q: https://www.infoq.com/fr/news/2017/12/webassembly-browser-support/

Balva, C. (2017, Février 22). *La Blockchain: réinventer les rapports de confiance | Claire BALVA | TEDxLyon*. Récupéré sur YouTube: https://www.youtube.com/watch?v=JID9c-MABis&t=67s

Bastien, L. (2020, Novembre 18). *Machine Learning : Définition, fonctionnement, utilisations*. Récupéré sur DataScientest: https://datascientest.com/machine-learning-tout-savoir#:~:text=Le%20Machine%20Learning%20ou%20apprentissage,dans%20les%20ensembles%20de%20donn%C3%A9es.

Beres, J. (2021, Février 4). *Blazor vs Angular Comparison in 2021*. Récupéré sur Infragistics: https://www.infragistics.com/community/blogs/b/jason_beres/posts/blazor-vs-angular

Beres, J. (2021, Mai 11). *Blazor vs React Comparison 2021*. Récupéré sur Infragistics: https://www.infragistics.com/community/blogs/b/jason_beres/posts/blazor-vs-react-comparison

Bernard, S. (2020, Septembre 2). *Blazor WebAssembly : Les différences avec Blazor server en un coup d'oeil* . Récupéré sur SQLI Digital Experience: https://www.technologies-ebusiness.com/solutions/blazor-webassembly-server

Biseul, X. (2018, Avril 09). *Plus flexible, moins cher : le serverless est la prochaine étape du cloud*. Récupéré sur Journal du net: https://www.journaldunet.com/solutions/cloud-computing/1207866-serverless-la-prochaine-revolution-du-cloud/

Bruno, E. (2020, Septembre 9). *ML.NET Model Builder GPU vs CPU test: 4 times faster !* Récupéré sur elbruno.com: https://elbruno.com/2020/09/09/vs2019-ml-net-model-builder-gpu-vs-cpu-test-4-times-faster/

Bryant, D. (2017, Mars 7). *Why WebAssembly is a game changer for the web — and a source of pride for Mozilla and Firefox*. Récupéré sur Mozilla Tech: https://medium.com/mozilla-tech/why-webassembly-is-a-game-changer-for-the-web-and-a-source-of-pride-for-mozilla-and-firefox-dda80e4c43cb#.8r1vu8w4l

Carlos. (2017, Mars 29). *Asm.js and Web Assembly*. Récupéré sur The Publishing Project: https://publishing-project.rivendellweb.net/asm-js-and-web-assembly/

CERN Blog. (s.d.). *Brève histoire du Web*. Récupéré sur CERN: https://home.cern/fr/science/computing/birth-web/short-history-web

Chiarelli, A. (2020, Novembre 10). *Five Things You Should Know About .NET 5*. Récupéré sur Auth0: https://auth0.com/blog/dotnet-5-whats-new/

Clark, L. (2018, Octobre 22). *WebAssembly's post-MVP future: A cartoon skill tree*. Récupéré sur Hacks Mozilla: https://hacks.mozilla.org/2018/10/webassemblys-post-mvp-future/

Communauté Microsoft. (2021, 11 01). *Build Progressive Web Applications with ASP.NET Core Blazor WebAssembly*. Récupéré sur Microsoft: https://docs.microsoft.com/en-us/aspnet/core/blazor/progressive-web-app?view=aspnetcore-5.0&tabs=visual-studio

Communauté Microsoft. (s.d.). *Blazor*. Récupéré sur Microsoft: https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor

Communauté Microsoft. (s.d.). *ML.NET*. Récupéré sur Microsoft: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet

Communauté Microsoft. (s.d.). *ML.NET Model Builder*. Récupéré sur Microsoft: https://dotnet.microsoft.com/apps/machinelearning-ai/ml-dotnet/model-builder

Communauté Mozilla. (s.d.). *Comprendre le format texte de WebAssembly*. Récupéré sur Developper Mozilla: https://developer.mozilla.org/fr/docs/WebAssembly/Understanding_the_text_format

Communauté Mozzila. (s.d.). *Converting WebAssembly text format to wasm*. Récupéré sur Developper Mozilla: https://developer.mozilla.org/en-US/docs/WebAssembly/Text_format_to_wasm

Communauté WebAssembly. (s.d.). *Types*. Récupéré sur WebAssembly Github: https://webassembly.github.io/spec/core/syntax/types.html

Contributeurs du projet Pyodide. (s.d.). *Roadmap*. Récupéré sur pyodide.org: https://pyodide.org/en/stable/project/roadmap.html

Couriol, B. (2021, Mai 31). *Pyodide Brings Python and Its Scientific Stack to the Browser with WebAssembly*. Récupéré sur InfoQ: https://www.infoq.com/news/2021/05/pyodide-python-webassembly/

Développement web. (2020, Février 07). *HTML5 c'est quoi ?* Récupéré sur Digital Guide IONOS: https://www.ionos.fr/digitalguide/sites-internet/developpement-web/html5-cest-quoi/

Développement web. (2020, Septembre 16). *Programmer en langage Rust : présentation de ce langage de programmation moderne*. Récupéré sur Digital Guide IONOS: https://www.ionos.fr/digitalguide/sites-internet/developpement-web/rust-langage-de-programmation/

Dulanga, C. (2020, Août 24). *Blazor VS JavaScript*. Récupéré sur Bits and Pieces: https://blog.bitsrc.io/frontend-dev-blazor-vs-javascript-35f27e0fd618

Eberhardt, C. (2019, Novembre 26). *WebAssembly on the Blockchain and JavaScript Smart Contracts*. Récupéré sur Scott Logic: https://blog.scottlogic.com/2019/11/26/webassembly-on-the-blockchain.html

Eberhardt, C. (2021, Janvier). *History of the Web: 1980s-1990s*. Récupéré sur EDX courses: https://learning.edx.org/course/course-v1:LinuxFoundationX+LFD133x+1T2021/block-v1:LinuxFoundationX+LFD133x+1T2021+type@sequential+block@5765346b183a4b4cae1e1de9314040f6/block-v1:LinuxFoundationX+LFD133x+1T2021+type@vertical+block@7209595d5fd244b8a74b0e9c

Eberhardt, C. (2021, Janvier). *LFD133x - Introduction to WebAssembly - A Brief History*. Récupéré sur EDX courses: https://learning.edx.org/course/course-v1:LinuxFoundationX+LFD133x+1T2021/block-v1:LinuxFoundationX+LFD133x+1T2021+type@sequential+block@5765346b183a4b4cae1e1de9314040f6/block-v1:LinuxFoundationX+LFD133x+1T2021+type@vertical+block@006bbd0e5ace43abab8c8fe7

Eberhardt, C. (2021, Janvier). *LFD133x - Introduction to WebAssembly - AssemblyScript*. Récupéré sur EDX courses: https://learning.edx.org/course/course-v1:LinuxFoundationX+LFD133x+1T2021/block-v1:LinuxFoundationX+LFD133x+1T2021+type@sequential+block@32ae0801418a441ca8278dc065464255/block-v1:LinuxFoundationX+LFD133x+1T2021+type@vertical+block@755a3975d2534e8b8b94a914

Eberhardt, C. (2021, Janvier). *LFD133x - Introduction to WebAssembly - Portability*. Récupéré sur EDX courses: https://learning.edx.org/course/course-v1:LinuxFoundationX+LFD133x+1T2021/block-v1:LinuxFoundationX+LFD133x+1T2021+type@sequential+block@76dba8164bf34bfcaf2be25e37a13bf9/block-v1:LinuxFoundationX+LFD133x+1T2021+type@vertical+block@8522fb318c71415b81340a51

Eberhardt, C. (2021, Janvier). *LFD133x - Introduction to WebAssembly - So Why Do We Need WebAssembly?* Récupéré sur EDX courses: https://learning.edx.org/course/course-v1:LinuxFoundationX+LFD133x+1T2021/block-v1:LinuxFoundationX+LFD133x+1T2021+type@sequential+block@5765346b183a4b4cae1e1de9314040f6/block-v1:LinuxFoundationX+LFD133x+1T2021+type@vertical+block@3c7ffb295205444cbdfd11d5

Emscripten Contributors. (s.d.). *About Emscripten*. Récupéré sur Emscripten: https://emscripten.org/docs/introducing_emscripten/about_emscripten.html

Garcia, D. J. (2020, Juillet 05). *Using Blazor, Tensorflow and ML.NET to Identify Images*. Récupéré sur dotnetcurry.com: https://www.dotnetcurry.com/aspnet-core/1537/blazor-ml-dotnet

GAUTRON, N. (2014, Mai 10). *Le principe de l'AJAX*. Récupéré sur Informatix: http://www.informatix.fr/tutoriels/javascript/le-principe-de-l-ajax-88

Getting Started. (s.d.). Récupéré sur WebAssembly: https://webassembly.org/getting-started/developers-guide/

Grace, D. (2021, Janvier 18). *How Blazor Performs Against Other Frameworks*. Récupéré sur Progress Telerik: https://www.telerik.com/blogs/how-blazor-performs-against-other-frameworks

Guilloux, M. (2017, Septembre 12). *Le projet AssemblyScript compile un sous-ensemble de TypeScript en WebAssembly*. Récupéré sur Developpez.com: https://www.developpez.com/actu/159837/Le-projet-AssemblyScript-compile-un-sous-ensemble-de-TypeScript-en-WebAssembly-il-est-open-source-et-disponible-sous-licence-Apache-2-0/

Guilloux, M. (2017, Novembre 14). *Le support de WebAssembly est désormais disponible dans tous les principaux navigateurs*. Récupéré sur Developpez.com: https://www.developpez.com/actu/173213/Le-support-de-WebAssembly-est-desormais-disponible-dans-tous-les-principaux-navigateurs-Safari-et-Microsoft-Edge-emboitent-le-pas-a-Firefox-et-Chrome/#:~:text=Le%20support%20de%20WebAssembly%20est,pas%20%C3%A0%20Fire

Hiel, A. (2019, Décembre 5). *World Wide Web Consortium (W3C) brings a new language to the Web as WebAssembly becomes a W3C Recommendation*. Récupéré sur W3: https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en

Hilton, J. (2020, Août 18). *Blazor vs Vue*. Récupéré sur Progress Telerik: https://www.telerik.com/blogs/blazor-vs-vue-web-developers

La Rédaction JDN. (2019, Février 14). *HTML5 (HyperText Markup Language 5) : définition de ce langage informatique*. Récupéré sur Journal du net: https://www.journaldunet.fr/web-tech/dictionnaire-du-webmastering/1203257-html5-hypertext-markup-langage5-definition-traduction/

Mears, J. (2019, Juin 20). *How we're bringing Google Earth to the web*. Récupéré sur web.dev: https://web.dev/earth-webassembly/

Microsoft, C. (2021, Avril 13). *Train an ML.NET classification model to categorize images*. Récupéré sur Microsoft: https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/image-classification

Nebra, M., & Schaller, M. (2021, Mars 22). *Programmez avec le langage C++*. Récupéré sur Openclassrooms: https://openclassrooms.com/fr/courses/1894236-programmez-avec-le-langage-c/1894377-quest-ce-que-le-c

Phelps, J. (2019, Janvier 02). *WebAssembly: Neither Web, Nor Assembly, but Revolutionary*. Récupéré sur JavaScript January: https://www.javascriptjanuary.com/blog/webassembly-neither-web-nor-assembly-but-revolutionary

Ramesh, P. (2019, Janvier 3). *The elements of WebAssembly – Wat and Wasm*. Récupéré sur Packt: https://hub.packtpub.com/the-elements-of-webassembly-wat-and-wasm-explained-tutorial/

Ranjan, B. (2020, Juillet 23). File Uploads in Blazor C# .Net Core(3.1) Session-7. Récupéré sur https://www.youtube.com/watch?v=clPew1ciR6g

Ranjan, B. (2020, Septembre 2020). Image Classification(Deep Learning) with ML.NET and Blazor Server App (.NET Core). Récupéré sur https://www.youtube.com/watch?v=EWOVAETLIRs&t=650s

Redkire. (2020, Février 14). *Pourquoi apprendre C#*. Récupéré sur Le blog du codeur: https://leblogducodeur.fr/pourquoi-apprendre-c-2/

Reiff, N. (2020, Juin 16). *Bitcoin vs. Ethereum: What's the Difference?* Récupéré sur Investopedia: https://www.investopedia.com/articles/investing/031416/bitcoin-vs-ethereum-driven-different-purposes.asp

Smaïl. (2021, Mars 8). *Nombre de sites web en 2021 : combien existe-t'il de sites internet dans le monde ?* Récupéré sur EMarketerz: https://www.emarketerz.fr/nombre-de-sites-web-internet-monde/#:~:text=de%20la%20publicit%C3%A9).-,Mise%20%C3%A0%20jour%20%3A%20au%2008%20mars%202021%2C%20on%20comptabilise%201,web%20%C3%A0%20l'%C3%A9chelle%20mondiale

Takahashi, D. (2020, Juin 25). *Newzoo: There will be over 3 billion gamers by 2023*. Récupéré sur Venture Beat: https://venturebeat.com/2020/06/25/newzoo-over-3-billion-gamers-by-2023/

WebAssembly Overview. (s.d.). Récupéré sur Site officiel de WebAssembly: https://webassembly.org/

Yegulalp, S. (2019, Avril 9). *WASI de Mozilla porte WebAssembly au-delà du navigateur*. Récupéré sur Le monde informatique: https://www.lemondeinformatique.fr/actualites/lire-wasi-de-mozilla-porte-webassembly-au-dela-du-navigateur-74932.html

Yu, A. (2019, Février 27). *How Netflix Uses AI, Data Science, and Machine Learning — From A Product Perspective*. Récupéré sur Becoming Human: https://becominghuman.ai/how-netflix-uses-ai-and-machine-learning-a087614630fe

Annexe I : Consigne travail de Bachelor

HES-SO Valais

FEE	FIG	FTO
	X	

Information regarding Bachelor's thesis

FO.2.2.02.28.EC
mob/30/10/2017

Degree programme: **BUSINESS INFORMATION TECHNOLOGY**

Confidential ☐

Student **SURNAME:** **ZAYCHENKO** Name: Yevheniy **Mobile:** **+41 79 845 89 91**	**Year** 2021
Submitted by: **Professor:** Henning Müller	**Language** ☒ French ☒ German ☒ English

Title: Limiting server resource usage via client-side processing using WebAssembly

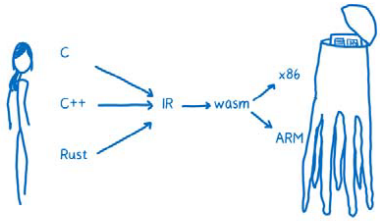
Description:

Context (company research project, etc.)

At the Institute of Information Systems of HES-SO Valais in Sierre research is conducted in the domain of e-health among other topics. An increasingly important paradigm in IT in general and especially e-health is Artificial Intelligence (AI) using Machine Learning (ML) techniques for predictions, such as cancer detection in images or prediction of response to treatment. As computing efficiency is an important aspect for such applications due to limited resources, we are looking for a way to reduce resource usage on the server side by running some of the computations directly in the client browser.

With the creation of WebAssembly[1] a few years ago, it is now possible to run heavier computations directly in the browser. WebAssembly provides a binary instruction format for the Browser's Virtual Machine and aims to execute at native speed by taking advantage of common hardware capabilities. A WebAssembly module can officially be compiled from C/C++, Rust and AssemblyScript (and a few other languages).

The protoypes developed in the proposed bachelor project should provide us with enough information to decide whether WebAssembly is a technology that we want to examine further and possibly use in the context of other research projects.



https://miro.medium.com/max/5200/0*E2D6rYKafJBRuIK9.png

What concrete issues does the project deal with?

Machine Learning uses many computing resources, which can easily overload servers (CPU, GPU, RAM). We would like to be able to reduce the server resource usage by increasing client-side resource utilization via the browser. Every instance of the application running in the browser should be able to partially replace specific server tasks. This project should address these problems by examining the WebAssembly technology and its ability to use client CPU and possibly GPU. A prototype application of medical image classification will showcase these capabilities.

[1] http://www.webassembly.org

HES-SO Valais

FEE	FIG	FTO
	X	

Information regarding Bachelor's thesis

FO.2.2.02.28.EC
mob/30/10/2017

Planned project stages:

- Comparison of WebAssembly and similar approaches in a state of the art
- Creation of the project backlog with the mandorty and nice-to-have items
- Regular physical or virtual meetings with supervisor
- Decisions on the environment to use for WebAssembly development for a simple image classification task
- Implementation of a sample WebAssembly application (programming language to be decided by the student)
- Writing of the final report

Objectives:

- Analysis of the state of the art regarding WebAssembly
- Decision taking regarding porgamming languages to use and the exact development environment
- Implementation of a WebAssembly demo application
- Written final report with conclusion that highlights the advantages and inconveniences of WebAssembly compared to similar approaches

Comments by the professor:

Signature

Head degree programme Business Information Tech.

.....

Professor:

.....

Student:

.....

Key dates

Attribution of subject:

22.02.2021

Due date of report:

13.08.2021 12.00

Public exhibition of Bachelor's theses:

To be defined

Annexe II : Classe startup – Blazor Server

```
namespace ServerApplication
{
    2 references
    public class Startup
    {
        0 references
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        1 reference
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        // For more information on how to configure your application,
        // visit https://go.microsoft.com/fwlink/?LinkID=398940
        0 references
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();
            services.AddServerSideBlazor();
            services.AddSingleton<WeatherForecastService>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        0 references
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this for production scenarios,
                // see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapBlazorHub();
                endpoints.MapFallbackToPage("/_Host");
            });
        }
    }
}
```

Annexe III : Code première version du prototype

Code permettant de sauvegarder l'image dans un flux de mémoire pour l'afficher sur l'application.

```
public class SelectedImage
{
    private IFileListEntry _file;
    3 references | yevhe, 36 days ago | 1 author, 1 change
    public string Base64Image { get; private set; }
    6 references | yevhe, 36 days ago | 1 author, 1 change
    public ImageClassificationResult ClassificationResult { get; set; }
    2 references | yevhe, 36 days ago | 1 author, 1 change
    public string Name => _file.Name;
    2 references | yevhe, 36 days ago | 1 author, 1 change
    public double UploadedPercentage => 100.0 * _file.Data.Position / _file.Size;

    1 reference | yevhe, 36 days ago | 1 author, 1 change
    public SelectedImage(IFileListEntry file)
    {
        _file = file;
    }

    1 reference | yevhe, 36 days ago | 1 author, 1 change
    public async Task<MemoryStream> Upload(Action OnDataRead)
    {
        if (_file.Data.Position > 0) throw new InvalidOperationException("Already uploaded");

        EventHandler eventHandler = (sender, eventArgs) => OnDataRead();
        _file.OnDataRead += eventHandler;

        // Download File contents into a memory stream, so we can later hand it over to the ML.NET service
        var fileStream = new MemoryStream();
        await _file.Data.CopyToAsync(fileStream);

        // Get a base64 so we can render an image preview
        Base64Image = Convert.ToBase64String(fileStream.ToArray());

        _file.OnDataRead -= eventHandler;
        return fileStream;
    }
}
}
```

Code permettant de procéder à la classification avec la conversion de l'image en Bitmap.

```
public class ImageClassificationService
{
    private string[] _labels;
    private PredictionEnginePool<ImageInputData, ImageLabelPredictions> _predictionEnginePool;

    0 references | yevhe, 36 days ago | 1 author, 1 change
    public ImageClassificationService(PredictionEnginePool<ImageInputData, ImageLabelPredictions> predictionEnginePool)
    {
        _predictionEnginePool = predictionEnginePool;
        string labelsFileLocation = PathUtilities.GetPathFromBinFolder(Path.Combine("TFInceptionModel",
            "imagenet_comp_graph_label_strings.txt"));
        _labels = System.IO.File.ReadAllLines(labelsFileLocation);
    }

    1 reference | yevhe, 36 days ago | 1 author, 1 change
    public ImageClassificationResult Classify(MemoryStream image)
    {
        // Convert to Bitmap
        Bitmap bitmapImage = (Bitmap)Image.FromStream(image);

        // Set the specific image data into the ImageInputData type used in the DataView
        ImageInputData imageInputData = new ImageInputData { Image = bitmapImage };

        // Predict code for provided image
        ImageLabelPredictions imageLabelPredictions = _predictionEnginePool.Predict(imageInputData);

        // Predict the image's label (The one with highest probability)
        float[] probabilities = imageLabelPredictions.PredictedLabels;
        var maxProbability = probabilities.Max();
        var maxProbabilityIndex = probabilities.AsSpan().IndexOf(maxProbability);
        return new ImageClassificationResult()
        {
            Label = _labels[maxProbabilityIndex],
            Probability = maxProbability
        };
    }
}
}
```

Code permettant de récupérer le chemin d'accès des fichiers .zip et .txt (fonctionne seulement dans la version Blazor Server).

```
public class PathUtilities
{
    1 reference | yevhe, 36 days ago | 1 author, 1 change
    public static string GetPathFromBinFolder(string relativePath)
    {
        FileInfo _dataRoot = new FileInfo(typeof(Program).Assembly.Location);
        string assemblyFolderPath = _dataRoot.Directory.FullName;

        string fullPath = Path.Combine(assemblyFolderPath, relativePath);

        return fullPath;
    }
}
}
```

Annexe IV : Code deuxième version du prototype

Code permettant de sauvegarder l'image dans un flux de mémoire pour l'afficher sur l'application, conversion en Base64 et appel de l'API.

```
@code {
    List<ImageInputData> filesBase64 = new List<ImageInputData>();
    bool isDisabled = false;
    string result = string.Empty;
    string header = string.Empty;


    //Manipulation de l'image transmise par l'utilisateur
    async Task OnChange(InputFileChangeEventArgs e)
    {
        isDisabled = true;
        filesBase64.Clear();
        result = string.Empty;
        var files = e.GetMultipleFiles();
        foreach (var file in files)
        {
            var resizedFile = await file.RequestImageFileAsync(file.ContentType, 640, 480);
            var buf = new byte[resizedFile.Size];
            using (var stream = resizedFile.OpenReadStream())
            {
                await stream.ReadAsync(buf);
            }
            header = file.Name;
            filesBase64.Add(new ImageInputData { base64data = Convert.ToBase64String(buf) });
        }
        await Upload();
    }


    //Méthode pour envoyer et recevoir le résultat de la classification
    async Task Upload()
    {
        using (var msg = await Http.PostAsJsonAsync<List<ImageInputData>>("/api/imageclassification", filesBase64, System.Threading.CancellationToken.None))
        {
            var data = await msg.Content.ReadFromJsonAsync<ImageInputData>();
            result = $"Classifié comme appartenant à la catégorie : {data.result}.";
            isDisabled = false;
        }
    }
}
```

API permettant de procéder à la classification d'images.

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class ImageClassificationController : ControllerBase
{
    private readonly IWebHostEnvironment env;
    0 references
    public ImageClassificationController(IWebHostEnvironment env)
    {
        this.env = env;
    }

    //Api qui reçoit les images pour la classification et retourne le résultat
    [HttpPost]
    0 references
    public async Task<ImageInputData> Post([FromBody] ImageInputData[] files)
    {
        ImageInputData obj = new ImageInputData();
        foreach (var file in files)
        {
            var buf = Convert.FromBase64String(file.base64data);
            obj.filePath = (env.ContentRootPath + "\\" + "Images" + "\\" +  "-" + file.fileName);
            await System.IO.File.WriteAllBytesAsync(obj.filePath, buf);
            obj.result = ImageDefectionFromML(obj.filePath);
            System.IO.File.Delete(obj.filePath);
        }
        return obj;
    }

    //Methode ML.NET pour exécuter la classification
    1 reference
    public string ImageDefectionFromML(string imageUrl)
    {
        var sampleData = new MLModel.ModelInput()
        {
            ImageSource = imageUrl,
        };
        var result = MLModel.Predict(sampleData);
        return result.Prediction.ToString();
    }
}
```

Annexe V : Product Backlog

Product Backlog

US Nr.	Thème	En tant que	Je souhaite	afin de	Critère d'acceptation	Statut	MoSCoW
1	Documentation	Développeur	Avoir une documentation sur le suivi du projet	Comprendre toutes les parties du projet	Avoir un rapport détaillé des opérations effectuées	●	Must
4	Préparation	Développeur	Installer tous les outils nécessaires pour le projet	Être prêt à commencer le travail dès le backlog accepté	Outils téléchargés et configurés	●	Must
5	Recherches et Analyses	Développeur	Prendre connaissance de la technologie WebAssembly	Définir le language à utiliser pour l'application définir l'architecture à mettre en place	Le language choisi s'intégre avec WebAssembly et l'application tourne correctement dans son environnement	●	Must
6	Recherches et Analyses	Développeur	Prendre connaissance pour l'hébergement de l'application et de ses données	Savoir ou publier l'application	Définir un lieu d'hébergement pour stocker l'outil et les données	●	Must
7	Recherches et Analyses	Développeur	Prendre connaissance d'autres technologies pour une application client-side	Comparer si une meilleure technologie existe à la place de WebAssembly	Démontrer les avantages et inconvénients	●	Must
8	Recherches et Analyses	Développeur	Prendre connaissance du language AssemblyScript	Comprendre son fonctinnement	Démontrer son utilité avec WebAssembly	●	Must
9	Application	Développeur	Exécuter un use-case de classification d'image de manière native	Exécuter la classifiction avec un training et une prédiction	L'algorithme analyse l'image et donne un résultat correct	●	Must
10	Application	Développeur	Transformer l'application pour la faire tourner en WebAssembly	La transformer d'une application native à WebAssembly	L'application s'execute en WebAssembly	●	Must
11	Application	Développeur	Importer un model entrainé existant pour faire une prédicion en WebAssembly	Utiliser des models existants	Le model est importé dans l'application et s'exécute en WebAssembly	●	Nice-to-have
12	Application	Développeur	Exporter un model entrainé depuis WebAssembly	Réutiliser le model	Le model est exporté depuis l'application et peut être réutilisé	●	Nice-to-have
13	Application	Développeur	Rendre l'application progressive	Accéder à l'application depuis mes applications de bureaux	L'application est affichée sur le bureau et est accessible en offline	●	Nice-to-have
14	Déploiement	Développeur	Déployer l'application	Avoir accès depuis internet	L'application est en ligne	●	Must
15	Analyses	Développeur	Comparer l'application serveur à celle en WebAssembly	Voir les avantages et les inconvénients Voir les différences de vitesses des executions	Les différences sont trouvées et présentées	●	Must
16	Analyses	Développeur	Explorer le capabilities hardware de WebAssembly	Voir si c'est possible d'utiliser diectement des composants hardware tel que GPU	Démontrer les possibilitées de comment accéder le hardware tel que GPU avec WebAssembly	●	Nice-to-have

Annexe VI : Idées rapport

Rapport idées

Thème	Idée	Status
Recherches	Historique du WebAssmebly	
Recherches	Principe et fonctionnement du WebAssembly	
Recherches	Caractéristiques de développement et de concept	
Recherches	Pour quelle utilisation le WebAssembly est intéressant (blockchain, machine learning, serverless, …)	
Recherches	Les principaux langages qui prennent en charge le WebAssembly (C/C++ avec Emscripten, RUST avec LLVM, C# avec Blazor)	
Recherches	Les langages qui ne prennent pas forcément encore WebAssembly.	
Recherches	Une partie sur AssemblyScript (Ivan l'a demandé)	
Recherches	Application progressive (desktop, offline)	
Analyses	Comparaison application WebAssembly et server side sur différent aspect	
Recherches	Solutions altérnatives au WebAssembly	
Analyses	Comparer les performances d'une application WebAssembly	

Annexe VII : Compte rendu des séances

Séances

Date	Sujet	à faire
26.02.2021	Présentation Discussion sur le WebAssembly Discussion sur l'idée générale du PB	Product Backlog Application basic de WebAssembly Prendre connaissance de WebAssembly
12.03.2021	Discussion sur le PB (trop de détails) Discussion sur le langage (Python, C#) Démo application basic WebAssembly Discussion Tenserflow	Modifier le PB Prendre connaissance de WebAssembly Prendre connaissance de Python et WebAssembly Prendre connaissance de tenserflow
26.03.2021	Discussion sur le langage de l'application (Python ou C#) Difficulté avec Python donc prendre C# Discussion sur les principes du machine Learning en général et son fonctionnement Discussion sur la structure du document (les éléments à mettre) Discussion de la présentation Discussion de l'évaluation	Application basic de machine learning en C# Prendre connaissance de WebAssembly Des librairies tensorflow, ML.net
09.04.2021	Démonstration de l'application server Discussion de l'avancement Discussion sur Tenserflow Discussion sur CPU et GPU	Transformer l'application vers WebAssembly Voir si compatible avec d'autres modèles de tensorflow Commencer rapport
30.04.2021	Présentation de l'avancée du rapport Discussion sur la structure du rapport Feedback sur les parties faites	Continuer le rapport
14.05.2021	Présentation de l'avancée du rapport La partie présentation de WebAssembly est terminée	Finir la partie analyse / état de l'art du sujet Structurer le rapport selon les remarques Résoudre et continuer l'application
28.05.2021	Diuscussion sur la structure que le rapport doit avoir Explication des différentes parties (Résultat, Discussion) Discussion sur comment lire un fichier	Chercher une solution pour récupérer le fichier .zip et txt pour le lire dans l'application Mettre en place cette solution
11.06.2021	Présentation de la solution pour lire les fichiers dans l'applcation --> API et calcul sur le serveur Discussion pour la machine virtuelle Discussion sur la partie du rapport pour Server VS WebAssembly	Mettre en place l'API de l'application
25.06.2021	Présentation de l'avancée du rapport Discussion sur le problème de l'accès du fichier Présentation du nouvelle solution avec Model Builder Discussion pour le déploiement	Continuer rapport Développer la nouvelle solution
08.07.2021	Présentation de l'avancée du rapport Présentation de la nouvelle solution avec une API Discussion sur la structure du rapport Discussion sur les parties à faire dans le rapport Discussion sur le temps restant et l'organisation à venir	Continuer rapport Finir la nouvelle solution Envoyer l'avancé par mail pour valider la structure
21.07.2021	Présentation de l'avancé du rapport Discussion sur le feedback reçu Noter les différentes remarques pour le document Présenter l'altérnative pour l'application Discuter des différents parties à rajouter dans le document	Continuer rapport Restructurer le document selon les remarques Voir la solution d'Ivan pour le problème avec l'app
27.07.2021	Présentation de l'avancé du rapport Discussion sur le feedback reçu Noter les différentes remarques pour le document Présenter les éléments à encore ajouter Parler du déploiement de l'app	Finir rapport Déployer l'application

Annexe VIII : Journal de bord personnel

Journal de bord

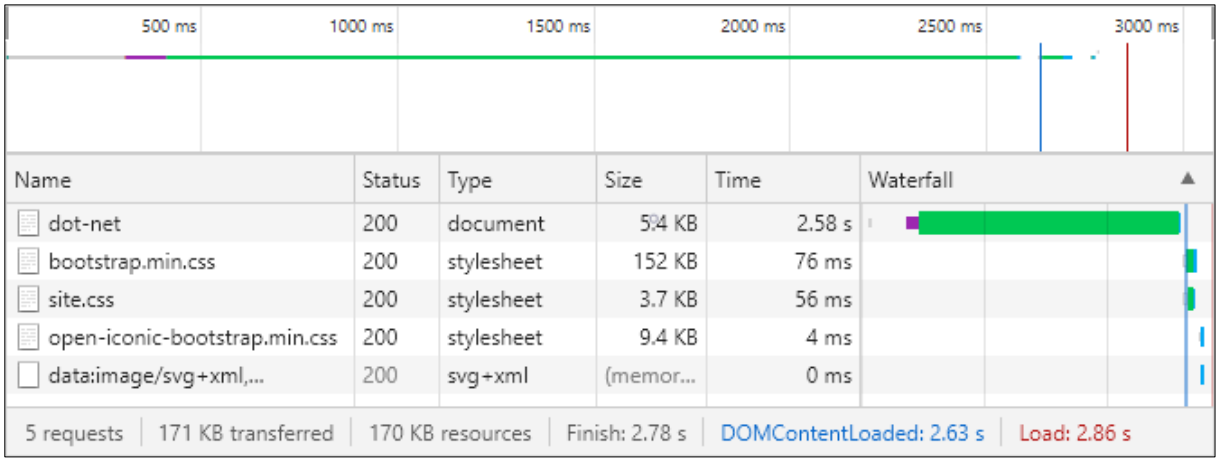
N° Semaine	Date	Tâches effectuées	Difficultés	Heures
8	22.02.21 - 28.02.21	Lecture des consignes du TB Lecture sur WebAssembly Vidéo sur WebAssembly en général	Sujet très vaste Concept très théorique, peu d'application concrète ou exemple	12
9	01.03.21 - 07.03.21	Conception du Product Backlog Lecture et vidéo sur les langages pour le WASM	PB trop détaillé Les sources sont souvent en anglais sur WebAssembly	13
10	08.03.21 - 14.03.21	Modification du Product Backlog Suivi un cours en ligne sur WebAssembly Développer une application basic avec C# Développer une application avancé selon tuto avec api, DB Vidéo sur WebAssembly et C# Recherche de sources et de documentations pour le sujet	Prendre la main avec Blazor	15
11	15.03.21 - 21.03.21	Recherche sur Python et WebAssembly Lecture sur les bases du Python Recherche sur les librairies pour le machine learning Tuto avec Tensorflow et Python Lecture sur le WebAssembly et reconnaissance d'images	Pas de connaissance de Python Comment transmettre l'application de Python à WebAssembly car très peu d'informations… Aucune informations sur le site officiel de WebAssembly	14
12	22.03.21 - 28.03.21	Recherche exemple application WebAssembly Recherche exemple application reconnaissance d'images Lecture sur le machine learning Suivi un cours sur le fonctionnememt d'une application de reconnaissances d'images Recherche, lecture et vidéo sur prise en main de Blazor avec C# Librairies pour machine learning avec C# Recherche si Tenserflow fonctionne avec C#	Pas de connaissance du machine learning Difficile de comprendre le fonctionnement des algorithmes	15
13	29.03.21 - 04.04.21	Lecture sur librairies ML.net et TensorFlow Recherche tuto application machine learning Développer application de détection d'images mais SERVEUR	Peu d'informations sur machine learning et webassembly application	16
14	05.04.21 - 11.04.21	Continuer application détection d'images SERVEUR	Différents bugs sur l'application Comprendre les différences entre les fichiers de l'application	12
15	12.04.21 - 18.04.21	Recherche sur comment transformer application SERVEUR vers WebAssembly Différentes vidéos, articles Transformer application serveur vers WebAssembly	Bug sur les versions net ou net core Bug sur les veriosn 3 et 5	16
16	19.04.21 - 25.04.21	Continuer à transformer l'application vers WebAssembly	Bug pour accès fichier de l'algo	14
17	26.04.21 - 02.05.21	Lecture d'ancien TB Planification de la structure du rapport Recherche sur les sources, les idées du rapport Début du rapport sur le web en général	Sujet très vaste du web	18
18	03.05.21 - 09.05.21	Continuer le rapport sur les concepts du WebAssembly Les différents langages Modifier le rapport selon les remarques des responsables du TB		12
19	10.05.21 - 16.05.21	Continuer la partie analyse de WebAssembly Historiques, définition, applications qui utilisent WebAssembly Avantages de WASM WASI Extension des différents fichiers de WebAssembly Finaliser la partie état de l'art du sujet Structurer le docuement selon les normes		18
20	17.05.21 - 23.05.21	Recherche sur le problème de l'upload de l'image Essayer plusieurs méhtodes Recherche sur la bonne version à avoir --> bug à cause de la version Résoudre bug Chercher une solution pour lire le fichier modèle dans l'application WebAssembly (API, HTTPClient ?)	Problème avec les différentes versions de visual studio, .net 5 et les références pour les forms WebAssembly ne permet pas de lire des fichiers car principe de sandbox… trouver un moyen	16
21	24.05.21 - 30.05.21	Continuer les recherches pour lire un fichier depuis une application WebAssembly Lire et apprendre comment mettre en place une API Se renseigner sur le déploiement de l'application		12
22	31.05.21 - 06.06.21	Supprimer les classes inutiles Nettoyer le code Mettre en place un nouveau projet qui prends en charge les APIs Résoudre les problèmes des bugs à cause de la nouvelle structure	Un projet qui gère les APIs à une autre structure donc ajuster le code et résoudre les bugs	16

23	07.06.21 - 13.06.21	Avancer sur la mise en place de l'API Recherche sur les différences entre une application Server et WebAssembly --> mettre dans le rapport		14
24	14.06.21 - 20.06.21	Écrire la partie du rapport sur le choix de l'environnement de développement et de travail Faire des recherches pour accéder aux fichiers avec la méthode FromUri Mettre en place la mérhode FromUri	La méthode FromUri ne fonctionne pas	20
25	21.06.21 - 27.06.21	Écrire la partie sur différences Blazor Server et WA, structure, avantages et inconvénients Écrire la partie sur Progressive web application Faire des recherches pour accéder aux fichiers avec embedded resource Mettre en place principe embedded resource Faire des recherches sur l'erreur avec Bitmap WA Tester des solutions pour résoudre le problème avec Bitmap WA	Faux espoir quant à la résolution du problème pour les fichiers --> embedded resource ne fonctionne pas pour finir - Problème avec la gestion des Bitmap WA	22
26	28.06.21 - 04.07.21	Écrire la partie sur Model Builder Écrire la partie sur CPU vs GPU Écrire la partie sur PWA (Progress Web Application) Mise en place d'une nouvelle solution pour les Bitmap		10
27	05.07.21 - 11.07.21	Terminer la partie sur les différences Blazor server et WA au niveau de la structure Écrire la partie sur les différences au niveau des performances Écrire la partie sur les différences Blazor server et WA au niveau de l'initialisatiion Tester une solution pour accéder aux fichiers depuis le storage du navigateur depuis l'application	Toujours le même problème pour l'accès des fichiers	35
28	12.07.21 - 18.07.21	Écrire la partie sur les technologies altérnatives Restructurer le document Recherche pour une nouvelle solution avec Model Builder Recherche pour mettre en place l'API Mise en place de la nouvelle solution	Méthode input pour exécuter l'algorithme doit avoir un chemin pour accéder à l'image	31
29	19.07.21 - 25.07.21	Écrire la partie des remerciements Écrire la partie de l'avant propos Écrire la partie du Résumé Modifier et compléter l'introduction Écrire la partie sur les résultats Écrire la partie sur la discussion Déplacer certains élements depuis méthode vers résultat Voir une altérnative d'Ivan au problème de l'application Restructurer le document Relire le document		44
30	26.07.21 - 01.08.21	Modifier le document selon le feedback Écrire la partie sur Python Compléter la partie des résultats Écrire la conclusion Relecture du document		25
31	02.08.21 - 08.08.21	Relecture du document Préparation des annexes Déploiement de l'application sur une VM Ubuntu	Problème avec le déploiement de l'application : la VM Ubuntu redirige automatiquemnt vers un lien HTTPS	20
32	09.08.21 - 13.08.21	Relecture du document Préparation du poster Déploiement de l'application sur une VM Windows		12
			Total	**452**

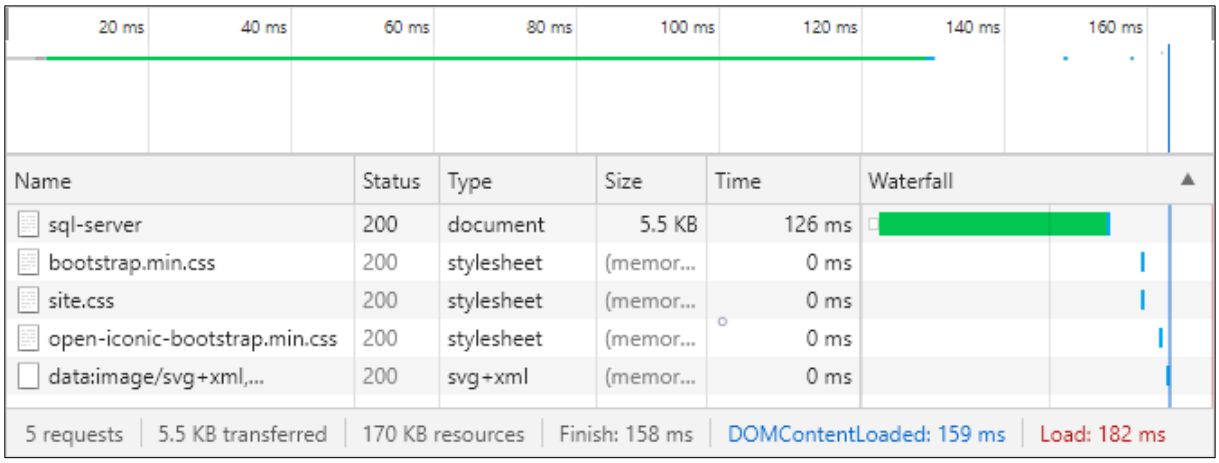
Annexe IX : Détails de l'étude utilisé dans le rapport

ASP.NET Core MVC Application

Temps pour l'initialisation de l'application

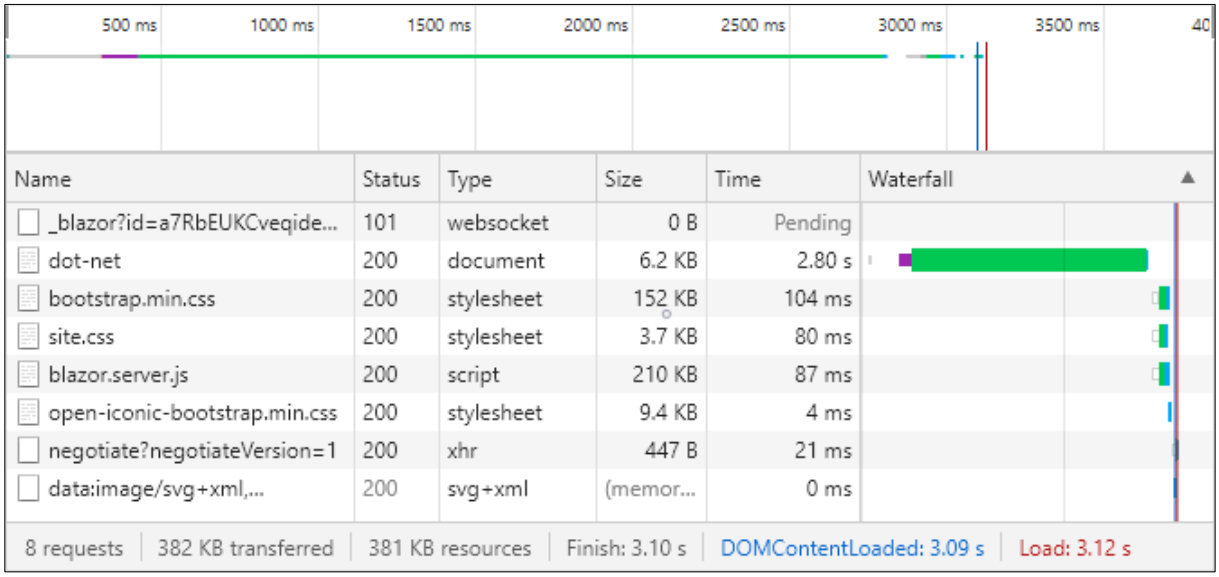


Temps pour la requête

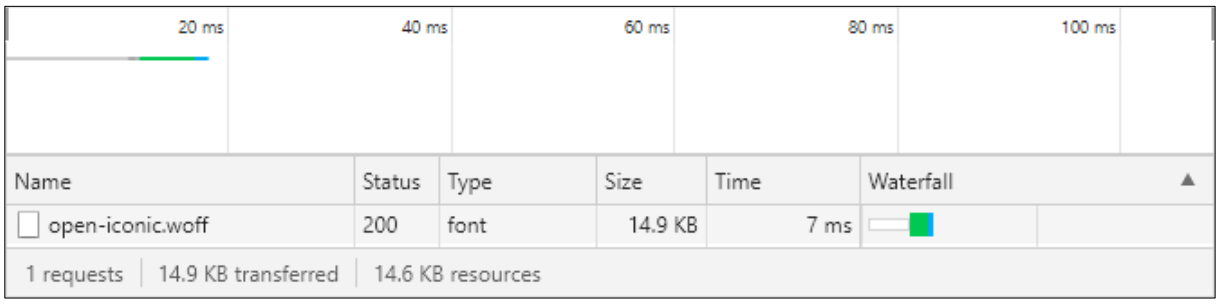


Blazor Server Application

Temps pour l'initialisation de l'application

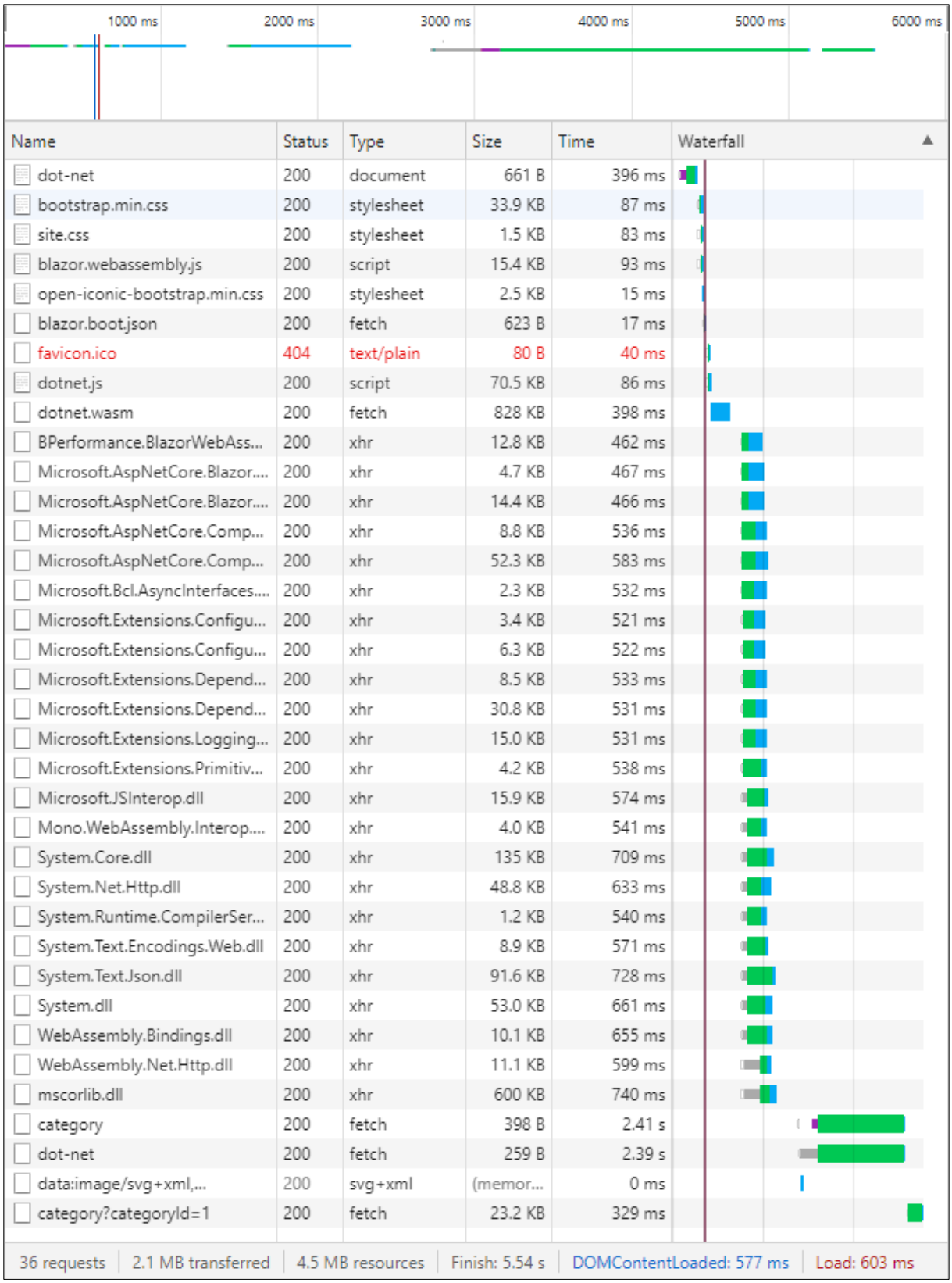


Temps pour la requête



Blazor WebAssembly Application

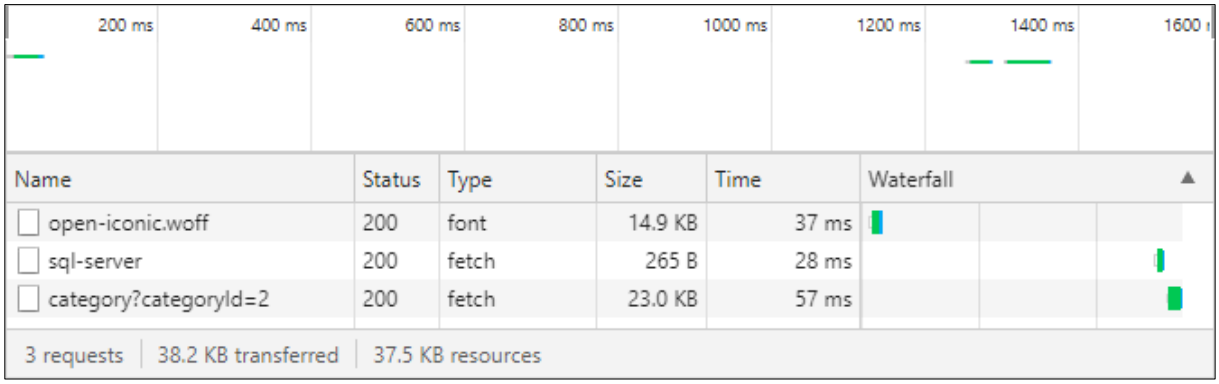
Temps pour l'initialisation de l'application



Name	Status	Type	Size	Time
dot-net	200	document	661 B	396 ms
bootstrap.min.css	200	stylesheet	33.9 KB	87 ms
site.css	200	stylesheet	1.5 KB	83 ms
blazor.webassembly.js	200	script	15.4 KB	93 ms
open-iconic-bootstrap.min.css	200	stylesheet	2.5 KB	15 ms
blazor.boot.json	200	fetch	623 B	17 ms
favicon.ico	404	text/plain	80 B	40 ms
dotnet.js	200	script	70.5 KB	86 ms
dotnet.wasm	200	fetch	828 KB	398 ms
BPerformance.BlazorWebAss...	200	xhr	12.8 KB	462 ms
Microsoft.AspNetCore.Blazor....	200	xhr	4.7 KB	467 ms
Microsoft.AspNetCore.Blazor....	200	xhr	14.4 KB	466 ms
Microsoft.AspNetCore.Comp...	200	xhr	8.8 KB	536 ms
Microsoft.AspNetCore.Comp...	200	xhr	52.3 KB	583 ms
Microsoft.Bcl.AsyncInterfaces....	200	xhr	2.3 KB	532 ms
Microsoft.Extensions.Configu...	200	xhr	3.4 KB	521 ms
Microsoft.Extensions.Configu...	200	xhr	6.3 KB	522 ms
Microsoft.Extensions.Depend...	200	xhr	8.5 KB	533 ms
Microsoft.Extensions.Depend...	200	xhr	30.8 KB	531 ms
Microsoft.Extensions.Logging...	200	xhr	15.0 KB	531 ms
Microsoft.Extensions.Primitiv...	200	xhr	4.2 KB	538 ms
Microsoft.JSInterop.dll	200	xhr	15.9 KB	574 ms
Mono.WebAssembly.Interop....	200	xhr	4.0 KB	541 ms
System.Core.dll	200	xhr	135 KB	709 ms
System.Net.Http.dll	200	xhr	48.8 KB	633 ms
System.Runtime.CompilerSer...	200	xhr	1.2 KB	540 ms
System.Text.Encodings.Web.dll	200	xhr	8.9 KB	571 ms
System.Text.Json.dll	200	xhr	91.6 KB	728 ms
System.dll	200	xhr	53.0 KB	661 ms
WebAssembly.Bindings.dll	200	xhr	10.1 KB	655 ms
WebAssembly.Net.Http.dll	200	xhr	11.1 KB	599 ms
mscorlib.dll	200	xhr	600 KB	740 ms
category	200	fetch	398 B	2.41 s
dot-net	200	fetch	259 B	2.39 s
data:image/svg+xml,...	200	svg+xml	(memor...	0 ms
category?categoryId=1	200	fetch	23.2 KB	329 ms

36 requests | 2.1 MB transferred | 4.5 MB resources | Finish: 5.54 s | DOMContentLoaded: 577 ms | Load: 603 ms

Temps pour la requête



Les images proviennent de la source suivante : https://www.telerik.com/blogs/how-blazor-performs-against-other-frameworks

Annexe X : Accéder à l'application

Afin d'utiliser l'application, nous avons le choix entre deux solutions :

1. **Localement** : Importer le code source de notre prototype dans l'application « Visual studio Entreprise 2019 ». Puis, exécuter ce dernier à l'aide du debugger « IIS Express ».
2. **Serveur** : L'application est déployée sur un machine virtuelle Windows. De cette manière, nous pouvons y accéder avec le lien suivant : http://153.109.124.171/
   Attention **:** il est nécessaire d'être connecté au réseau de la HES-SO Valais. Pour cela, nous devons utiliser une connexion VPN avec Pulse Secure ou se trouver physiquement dans l'établissement. Informations complémentaires : https://sinf.hevs.ch/Ressources/R%C3%A9seau/VPN

Déclaration de l'auteur

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autres aides que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Dr. Henning Müller
- M. Ivan Eggel

Monthey, le 12 août 2021

Yevheniy Zaychenko