# Table of contents

[Table de Figures iii](#_Toc185971322)

[Introduction 4](#_Toc185971323)

[Chapitre I : Contexte du Projet 5](#_Toc185971324)

[1. Problématique 5](#_Toc185971325)

[2. Public Cible 5](#_Toc185971326)

[3. La Solution 6](#_Toc185971327)

[4. Méthodologie de Travail 6](#_Toc185971328)

[5. Planning et Déploiement 7](#_Toc185971329)

[Chapitre II : Organisation et Développement du projet 8](#_Toc185971330)

[1. Organisation du Projet 8](#_Toc185971331)

[2. Diagramme de Cas d’Utilisation 9](#_Toc185971332)

[3. Choix Technologiques 10](#_Toc185971333)

[4. Architecture Logicielle 11](#_Toc185971334)

[5. Détails de la Mise en Œuvre 12](#_Toc185971335)

[6. Intégration d’API 13](#_Toc185971336)

[Chapitre III : Déploiement vers Azure Cloud 14](#_Toc185971337)

[1. Analyse et Conception 14](#_Toc185971338)

[2. Architecture Azure Cloud 16](#_Toc185971339)

[*2.1 Explication de l’Architecture 17*](#_Toc185971340)

[3. Justification des Choix 18](#_Toc185971341)

[*3.1 PaaS vs IaaS 18*](#_Toc185971342)

[*3.2 Avantages Clés de PaaS 19*](#_Toc185971343)

[4. Implémentation 19](#_Toc185971344)

[Chapitre IV : Evaluation et Résultat 24](#_Toc185971345)

[1. Méthodologies de Test 24](#_Toc185971346)

[2. Résultats 24](#_Toc185971347)

[3. Interfaces Utilisateur 25](#_Toc185971348)

[4. Leçons Apprises 27](#_Toc185971349)

[5. Difficultés Rencontrées 27](#_Toc185971350)

[6. Bilan du Déploiement sur Azure Cloud 28](#_Toc185971351)

[7. Défis et Pistes d’Amélioration 29](#_Toc185971352)

[Conclusion 4](#_Toc185971353)

[Bibliography 5](#_Toc185971354)

# Table de Figures

[Figure 1 - GitHub Repository for Project Management 7](#_Toc185969338)

[Figure 2 - Use Case Diagram for MSL Translation Application 8](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969339)

[Figure 3 - Project Architecture for MSL Application 10](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969340)

[Figure 4 - Services Azure Cloud 13](#_Toc185969341)

[Figure 5 - Estimation Mensuel des Services Azure 15](#_Toc185969342)

[Figure 6 - Architecture Azure Cloud 15](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969343)

[Figure 7 - Groupe de ressources 19](#_Toc185969344)

[Figure 8 - App Services 19](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969345)

[Figure 9 - Azure Registre de conteneurs 20](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969346)

[Figure 10 - Workflows CI/CD GitHub Actions 20](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969347)

[Figure 11 - Azure Storage Account 21](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969348)

[Figure 12 - Creation de Reseau Virtuel 21](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969349)

[Figure 13 - Home Page of EchoSign 24](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969350)

[Figure 14 - Sign to Text and Voice Translation Interface 25](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969351)

[Figure 15 - Text and Voice to Sign Translation Interface 26](file:///C:\Users\EL%20MOUADDIBE\Desktop\EchoSign\Report%20-%20EchoSign.docx#_Toc185969352)

# Introduction

La communication est un droit humain fondamental, pourtant de nombreuses personnes malentendantes au Maroc peinent encore à se faire comprendre au quotidien. La **Langue des Signes Marocaine (MSL)**, bien qu’elle constitue le principal mode de communication pour une large partie de la communauté sourde, demeure insuffisamment reconnue et comprise par la population en général. Cette méconnaissance alimente malentendus et isolement social, mettant en évidence la nécessité d’une solution technique pour faciliter les échanges.

Dans le cadre de notre formation d’ingénieurs, nous avons donc entrepris de développer une **application multiplateforme** permettant la traduction de la MSL en texte et en parole, et inversement. Pour atteindre cet objectif, nous nous sommes appuyés sur des outils de développement web modernes, dont **Angular**, et sur des **modèles d’apprentissage automatique** capables de fournir des traductions précises en temps réel. Nous avons également tiré parti des possibilités offertes par le **cloud**, en particulier la plateforme **Azure**, afin de déployer et de rendre notre solution disponible à grande échelle.

Notre ambition est double :

* **Réduire le fossé de communication** entre les utilisateurs de la MSL et le reste de la population, en améliorant l’accessibilité et en favorisant l’inclusion sociale.
* **Promouvoir l’apprentissage** de la MSL par la mise à disposition d’un outil pédagogique interactif et évolutif.

Ce rapport propose un aperçu complet du projet. Il débute par une analyse du contexte et des enjeux, puis décrit la conception fonctionnelle et technique de l’application. Viennent ensuite la présentation des choix méthodologiques et technologiques, suivie de la phase de développement et de **déploiement sur le cloud**. Enfin, nous mettons en lumière les principaux résultats obtenus, discutons de l’impact de l’outil et évoquons les pistes d’amélioration envisageables pour l’avenir.

# Chapitre I : Contexte du Projet

Ce chapitre expose le cadre général dans lequel notre projet a vu le jour. Il présente la problématique à laquelle nous avons souhaité répondre, décrit le public visé, puis propose la solution envisagée. Il aborde ensuite la méthodologie de travail mise en place, avant de détailler le planning qui inclut le déploiement et la migration finale vers Azure Cloud.

## Problématique

Les barrières de communication affectent significativement la vie des individus malentendants utilisant la Langue des Signes Marocaine (MSL). Ces barrières entravent leur capacité à interagir efficacement avec autrui, entraînant un isolement social et un accès limité aux services essentiels et aux informations. Les méthodes traditionnelles de communication, telles que les notes écrites ou les interprètes, sont souvent inadéquates et difficilement accessibles. Il existe donc un besoin urgent d’une solution technologique capable de combler ce fossé et de faciliter une communication fluide entre les utilisateurs de la MSL et la communauté au sens large.

## Public Cible

Le public cible principal de notre projet comprend :

* **Personnes malentendantes** : Utilisateurs de la Langue des Signes Marocaine nécessitant un outil fiable pour communiquer avec des non-utilisateurs de cette langue.
* **Famille et amis** : Personnes souhaitant améliorer leur communication avec leurs proches malentendants.
* **Éducateurs et institutions** : Écoles et organisations impliquées dans l’éducation et le soutien des malentendants, qui peuvent utiliser cet outil pour enseigner et améliorer les compétences en communication.
* **Grand public** : Personnes en interaction avec des malentendants dans divers contextes et ayant besoin d’un moyen pour mieux comprendre et communiquer avec eux.

## La Solution

La solution proposée est une application multiplateforme permettant la traduction de la Langue des Signes Marocaine vers du texte et de la voix, et inversement. Elle repose sur plusieurs technologies et méthodes :

* **Interface Intuitive** : Développée avec Angular, elle est conçue pour être à la fois réactive et facile à prendre en main.
* **Apprentissage Automatique** : Des modèles de reconnaissance et d’interprétation des gestes (vision par ordinateur et traitement du langage) assurent une traduction précise en temps réel.
* **Traduction Bidirectionnelle** : L’application convertit la MSL en texte/voix et effectue également la transformation inverse (texte/voix vers MSL).

**Les principaux objectifs de notre solution sont de :**

* Fournir un outil de communication accessible et efficace aux utilisateurs de la Langue des Signes Marocaine.
* Faciliter l'apprentissage de la MSL pour les personnes intéressées.
* Renforcer l'inclusion sociale en comblant le fossé de communication entre les utilisateurs de la langue des signes et la communauté élargie.

## Méthodologie de Travail

Nous avons adopté une démarche **agile**, privilégiant des cycles courts et des retours fréquents. Concrètement, cela s’est traduit par :

* **Brainstorming & Prototypage** : Recherche d’idées, réalisation de maquettes initiales et ajustements rapides.
* **Répartition des Tâches** : Identification des fonctionnalités majeures, puis distribution du travail entre les membres de l’équipe.
* **Intégration Continue** : Utilisation de GitHub pour centraliser le code, suivre les modifications et assurer la synchronisation entre les contributions de chacun.
* **Validation Régulière** : Tests itératifs pour corriger les anomalies, améliorer l’ergonomie et confirmer la conformité aux besoins identifiés.

## Planning et Déploiement

Afin de mener à bien le projet dans les délais, nous avons structuré les travaux en différentes phases :

1. **Analyse et Planification**
   * Définir les exigences fonctionnelles.
   * Choisir les technologies et prévoir la feuille de route.
2. **Conception**
   * Élaborer des wireframes et des diagrammes d’architecture.
   * Déterminer la structure du frontend (Angular) et du backend (Flask, Python, Machine Learning).
3. **Développement**
   * Implémenter le frontend et le backend, y compris l’intégration des modèles d’apprentissage automatique.
   * Effectuer des tests unitaires et d’intégration à mesure que les fonctionnalités sont développées.
4. **Tests & Validation**
   * Apporter les correctifs nécessaires pour garantir la stabilité et la fiabilité de l’application.
5. **Déploiement Local**
   * Mettre en place l’application sur un environnement local pour valider la configuration et la performance.
   * Effectuer une phase de monitoring afin d’identifier d’éventuels ajustements.
6. **Migration vers Azure Cloud**
   * Analyser les besoins (CPU, mémoire, stockage) et préparer l’infrastructure adéquate (Instances de calcul, Conteneurs, etc.).
   * Procéder au déploiement sur Azure App Service et d’autre solution PaaS, en assurant la sécurité et la scalabilité.
   * Vérifier la bonne accessibilité de l’application et mettre en place un plan de maintenance pour garantir sa disponibilité.

# Chapitre II : Organisation et Développement du projet

Une organisation, une planification et une mise en œuvre efficaces ont été essentielles au succès de notre application de traduction de la Langue des Signes Marocaine (MSL). Ce chapitre décrit l'approche structurée que nous avons adoptée, y compris l'utilisation de diagrammes UML pour définir les cas d'utilisation, la conception architecturale de notre application, les technologies utilisées pour le prototypage et développement, ainsi que les détails de la mise en œuvre

## Organisation du Projet

Afin d'assurer une communication efficace et une progression fluide, nous avons utilisé plusieurs outils et méthodes :

1. **Outils de Collaboration**

**GitHub :** utilisé pour le contrôle de version et la collaboration. Cette plateforme nous a permis de gérer efficacement notre base de code, de suivre les modifications et de collaborer de manière productive. Elle a facilité :

* **Le contrôle de version** : Suivi de l’historique des modifications et possibilité de revenir à des versions antérieures.
* **La collaboration** : Travail en parallèle des membres de l'équipe avec intégration fluide grâce aux pull requests et aux revues de code.

**A screenshot of a computer

Description automatically generated**

Figure 1 - GitHub Repository for Project Management

## Diagramme de Cas d’Utilisation

Afin de déterminer et de prioriser les fonctionnalités de notre application, nous avons conçu un diagramme de cas d’utilisation. Celui-ci met en évidence les principales interactions entre les utilisateurs et le système, et s’articule autour de la fonctionnalité centrale de traduction de la MSL.

* **Cas d'utilisation principal**

Le cas d’utilisation principal identifié pour notre projet est de faciliter la traduction de la Langue des Signes Marocaine (MSL) en texte et en parole, et inversement. Le diagramme met en évidence les principales interactions qu’un utilisateur peut avoir avec le système.

Figure 2 - Use Case Diagram for Moroccan Sign Language Translation Application

**Fonctionnalités principales incluses dans le diagramme :**

 Transformation **du Texte et de la Voix en Langue des Signes**

* L’utilisateur saisit un texte ou enregistre une voix ; le système le convertit en gestes MSL.
* Utile pour les personnes ne maîtrisant pas la MSL.

 Transformation **de la MSL en Texte et en Parole**

* Acquisition d’images via webcam, prétraitement, extraction de caractéristiques et reconnaissance via des modèles d’apprentissage automatique.
* Permet de traduire un geste MSL en texte ou en parole.

 Traduction **des Résultats en Anglais et en Français**

* Possibilité de proposer le texte reconnu dans différentes langues (ex. anglais, français et arabe), afin de rendre l’application accessible à un plus large public.

 Contact **avec les Administrateurs**

* Les utilisateurs peuvent contacter l’équipe administrative pour obtenir de l’aide, signaler un bug ou partager des suggestions.

## Choix Technologiques

Après avoir identifié les fonctionnalités requises, nous avons sélectionné un ensemble de technologies assurant robustesse, évolutivité et efficacité.

**a) Développement Frontend**

* **Angular**
  + Puissant framework JavaScript permettant la création d’applications monopages (SPA) et de Progressive Web Apps (PWA).
  + Architecture modulaire, gestion du routage et des formulaires, intégration facile de services partagés, etc.
* **Tailwind CSS**
  + Framework CSS utilitaire facilitant la création de designs personnalisés et responsive.
  + Permet d’harmoniser rapidement l’interface tout en restant flexible sur la personnalisation.

**b) Développement Backend et Machine Learning**

* **Python** : Simple d’emploi, avec un écosystème très complet pour la science des données et le machine learning.
* **Flask**
  + Framework léger pour créer des API RESTful et communiquer facilement avec le frontend.
  + Mise en place de points de terminaison pour gérer les requêtes et intégrer les modèles d’IA.
* **TensorFlow et Keras**
  + Bibliothèques conçues pour développer et entraîner des modèles de reconnaissance gestuelle, indispensables dans notre projet.
* **OpenCV**
  + Bibliothèque de vision par ordinateur nécessaire à la capture et au prétraitement des images.
  + Sert à extraire les caractéristiques des gestes MSL pour la classification.
* **OpenAI API** : Utilisée pour la synthèse vocale. Elle convertit le texte généré par les modèles en parole naturelle.

## Architecture Logicielle

L'architecture de notre application est un schéma complet qui décrit l'interaction entre les différents composants et services. Ce design garantit un système évolutif, maintenable et efficace. L'architecture est divisée en trois couches principales : le **frontend**, le **backend** et le **modèle de machine learning**.

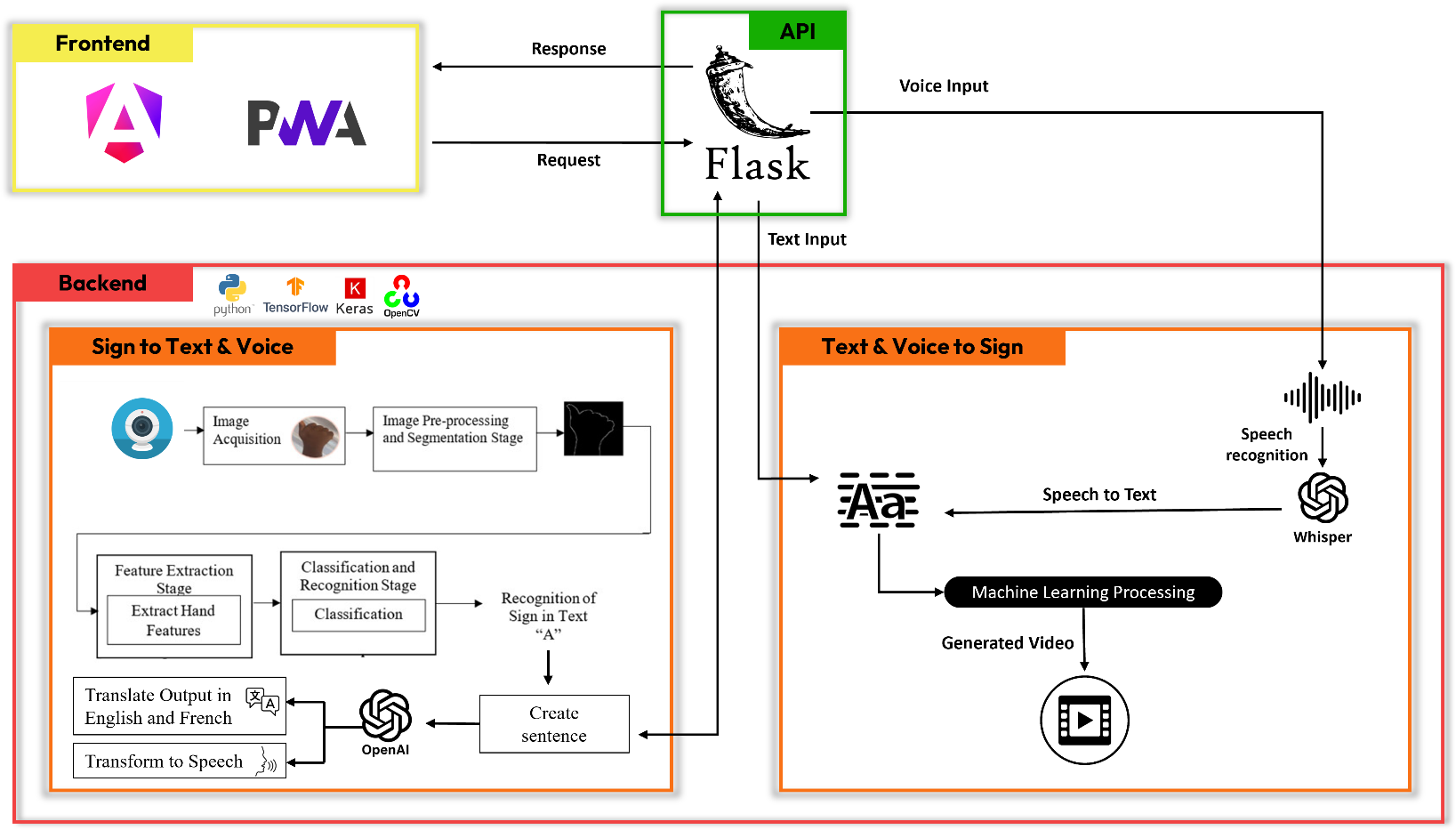


Figure 3 - Project Architecture for Moroccan Sign Language Translation Application



1. **Frontend**
   * Développé sous Angular pour offrir une interface dynamique et réactive.
   * Conçu également comme une PWA pour fonctionner sur différents appareils (smartphones, tablettes, PC) avec une expérience quasi-native.
2. **Backend**
   * Constitué d’une API RESTful créée avec Flask.
   * Gère la logique métier, orchestre les appels aux modèles de machine learning et transmet les résultats au frontend via des requêtes API.
3. **Machine Learning**
   * Entraîne, héberge et exécute les modèles de reconnaissance des gestes (TensorFlow + Keras).
   * S’appuie sur OpenCV pour la capture et la pré-analyse des images.

**Flux de Données**

1. **Interaction Utilisateur** : L’utilisateur fournit un texte, une voix ou un geste MSL via l’interface Angular.
2. **API Requests** : Le frontend envoie une requête HTTP (REST) au backend, qui encapsule les informations reçues.
3. **Traitement Machine Learning** : Le backend sollicite les modèles IA pour reconnaître les signes ou générer des gestes correspondants.
4. **Retour des Résultats** : La réponse (texte, voix ou vidéo de geste) est renvoyée au frontend, où elle est présentée à l’utilisateur.

## Détails de la Mise en Œuvre

**a) Sign to Text & Voice**

1. **Acquisition d’images** : Utiliser la webcam pour capturer le geste MSL.
2. **Prétraitement & Segmentation** : Nettoyage, recadrage et normalisation de l’image.
3. **Extraction de Caractéristiques** : Identification des informations pertinentes (mouvements, formes de la main, etc.).
4. **Classification & Reconnaissance** : Le modèle ML associe les caractéristiques extraites à un signe MSL connu.
5. **Traduction & Synthèse Vocale** : Le signe reconnu est affiché en texte et converti en parole via l’API OpenAI.

**b) Text & Voice to Sign**

1. **Traitement du Langage Naturel** : Interpréter la phrase entrée ou la voix enregistrée pour en extraire le sens.
2. **Génération de Signes** : Afficher une séquence vidéo représentant les gestes MSL correspondants.

## Intégration d’API

La couche API joue un rôle central, facilitant et sécurisant la communication entre le frontend et le backend. Elle :

* Gère les requêtes HTTP entrants (POST, GET, etc.) depuis Angular.
* Interagit avec les modèles de ML pour effectuer les opérations de reconnaissance ou de génération de signes.
* Retourne les informations au frontend dans un format exploitable (JSON, texte, fichiers audio ou vidéo).

# Chapitre III : Déploiement vers Azure Cloud

Dans ce chapitre, nous décrivons en détail le processus de déploiement de notre application **EchoSign** sur la plateforme **Microsoft Azure**. Nous revenons d’abord sur l’analyse et la conception nécessaires à la migration, puis nous détaillons l’architecture envisagée, nos justifications techniques et l’implémentation étape par étape. Enfin, nous concluons sur l’évaluation de la solution, en soulignant les avantages, les défis rencontrés et les pistes d’amélioration.

## Analyse et Conception

Avant toute mise en place, il convenait d’évaluer précisément les dépendances de **EchoSign** et d’estimer les ressources nécessaires sur Azure. Dans cette optique, nous avons utilisé l’outil [Pricing Calculator](https://azure.microsoft.com/en-us/pricing/calculator/) qui permet de simuler les coûts mensuels/annuels des différents services utilisés (App Services, Container Registry, Storage, etc.).

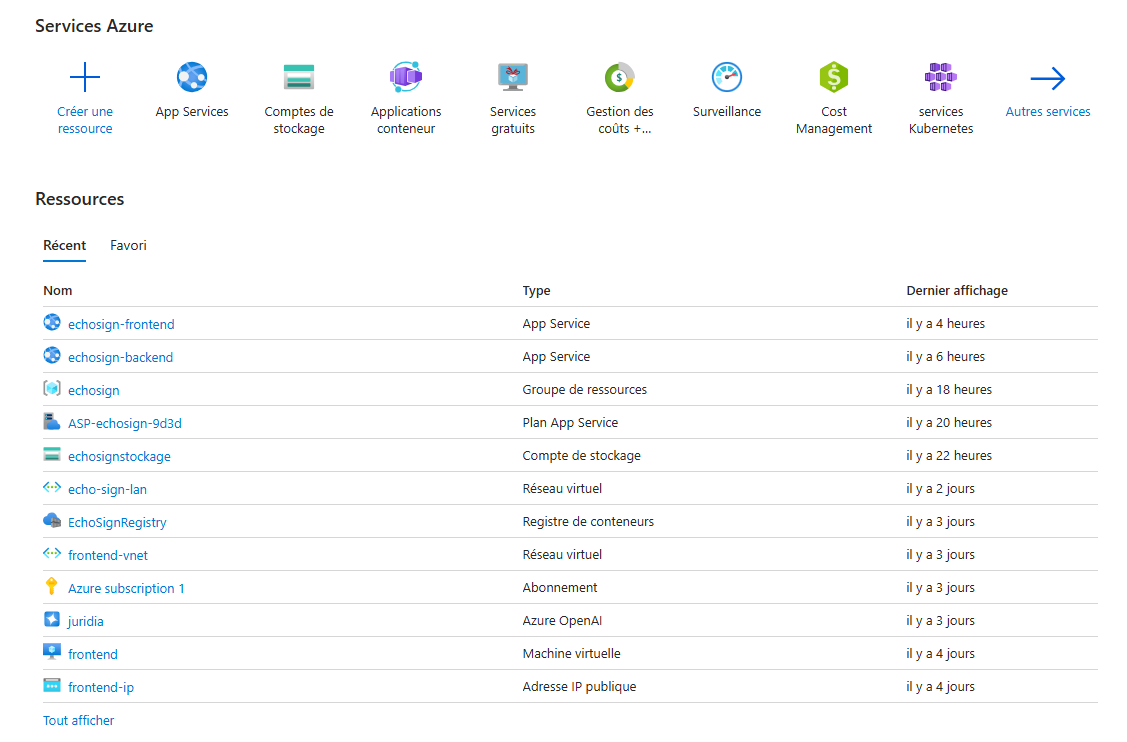


Figure 4 - Services Azure Cloud

* 1. **Identification des Dépendances**
* **Frontend**
  + **Node.js** : Exécution du projet Angular en mode production.
  + **Angular** : Framework pour construire une application monopage (SPA) et progressive (PWA).
* **Backend & Machine Learning**
  + **Python** : Langage principal pour la logique métier et l’appel aux modèles ML.
  + **Flask** : Framework léger facilitant la création d’API RESTful.
  + **TensorFlow et OpenCV** : Bibliothèques pour la reconnaissance des gestes (MSL) et le traitement d’images.
  + **Docker** : Conteneurisation des différents services pour plus de fiabilité et de portabilité.
* **Stockage**
  + Fichiers multimédias (images, vidéos) et logs d’application, nécessitant un service de type **Blob Storage**.
* **Réseau et Sécurité**
  + Besoin d’un réseau virtuel (VNet) pour isoler les services et d’un mécanisme de protection DDoS.

**1.2 Évaluation des Ressources**

* **CPU/RAM** : Les modèles de machine learning demandent une puissance de calcul suffisante (surtout lors des pics d’activité).
* **Stockage** : Volume modulable pour héberger les ressources (images, éventuels logs).
* **Haute Disponibilité** : Nécessité de répliquer les instances (autoscaling), d’avoir un mécanisme de répartition de la charge (load balancer) et d’assurer la tolérance aux pannes.

Grâce au **Pricing Calculator** [***(****https://azure.microsoft.com/en-us/pricing#Pricing-by-product),*](https://azure.microsoft.com/en-us/pricing#Pricing-by-product)nous avons simulé nos choix de configuration (App Services, Container Registry, etc.) et obtenu une estimation chiffrée. Cette démarche nous a permis de prévoir un budget prévisionnel et d’optimiser le dimensionnement des services.

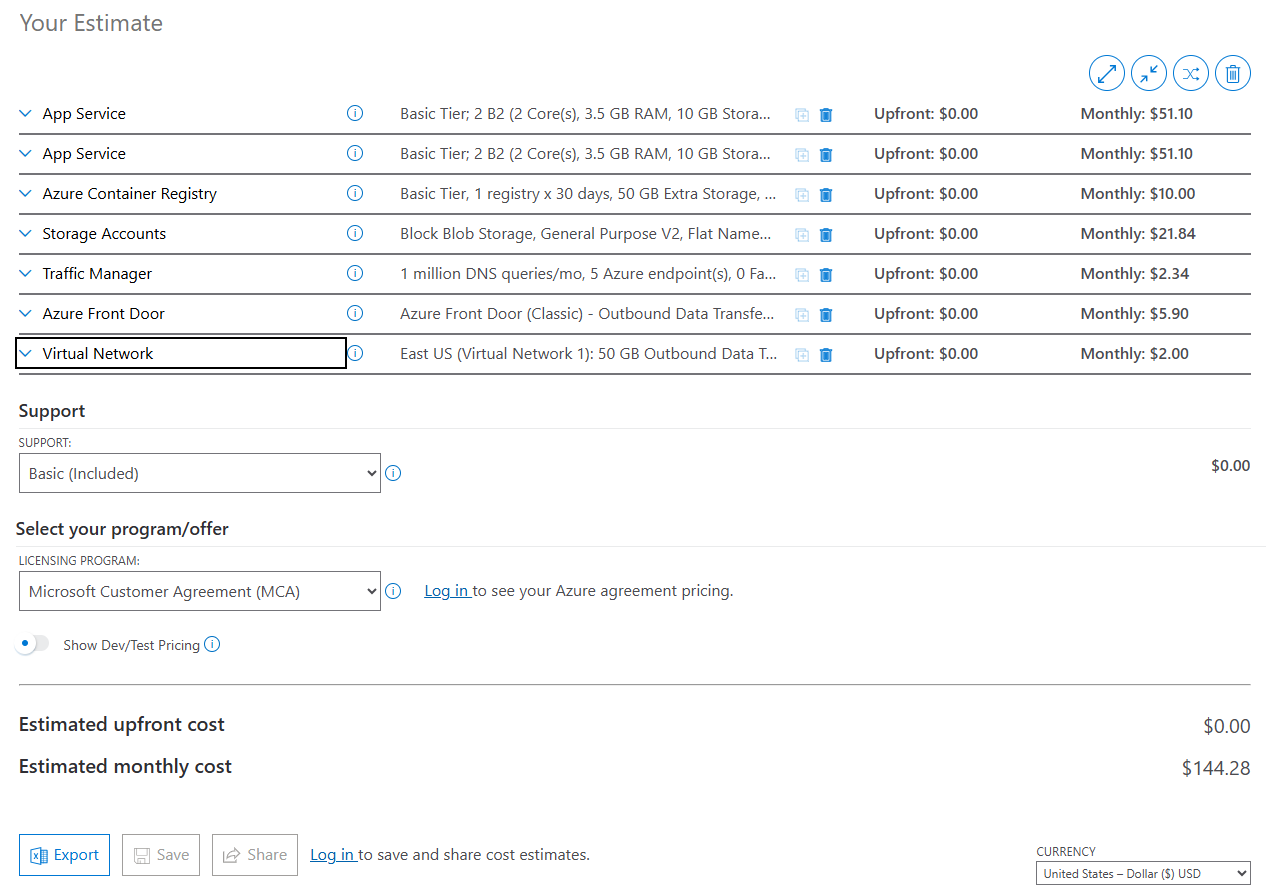


Figure 5 - Estimation Mensuel des Services Azure

## Architecture Azure Cloud

Après avoir collecté toutes les informations nécessaires, nous avons conçu une **architecture cloud** intégrant tous les composants de **EchoSign** et répondant aux exigences de performance, de sécurité et de résilience.

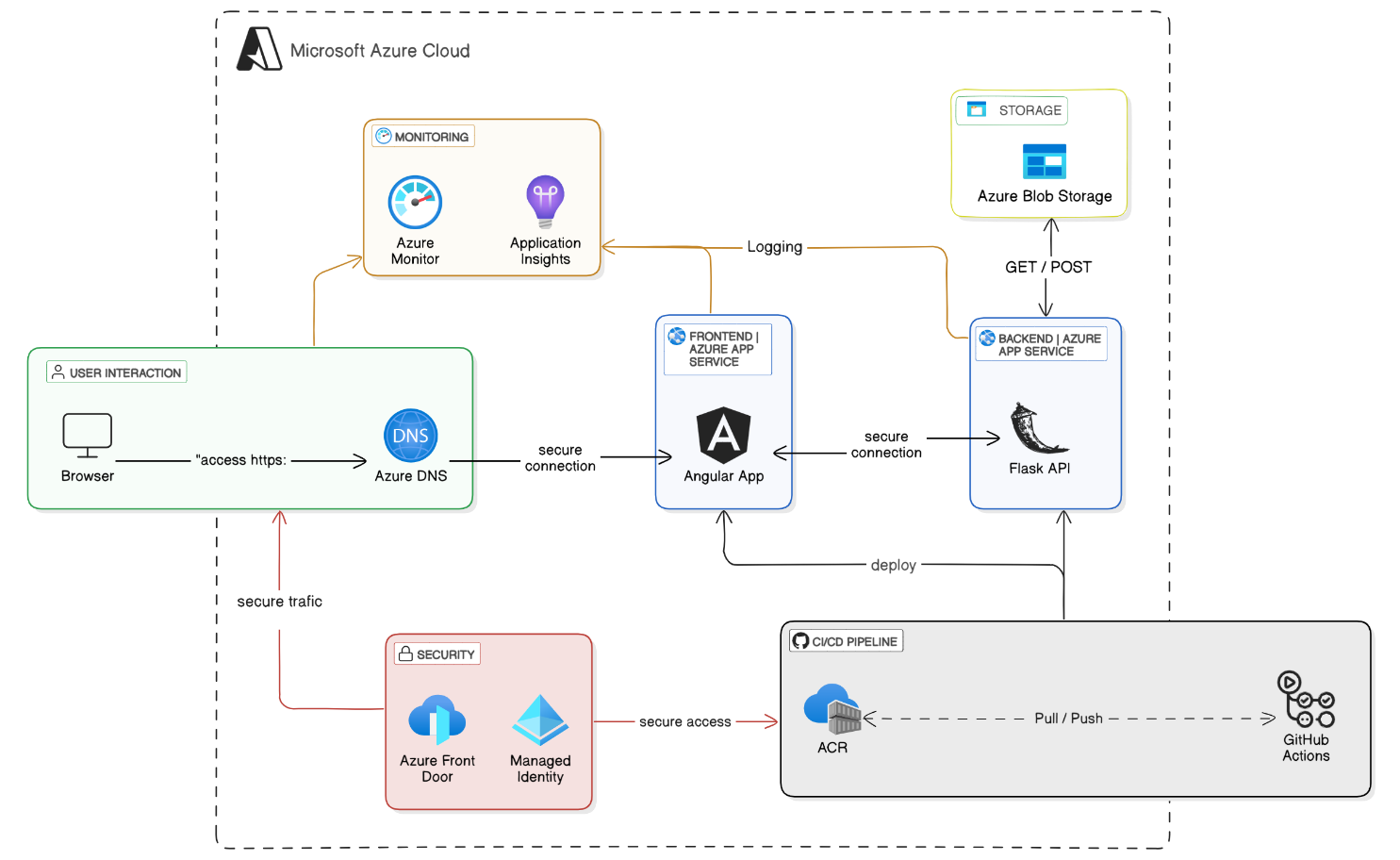
****

Figure 6 - Architecture Azure Cloud

### 2.1 Explication de l’Architecture

1. **Utilisateur**
   * L’utilisateur accède à **EchoSign** via un navigateur (URL : **echo-sign.tech**).
   * Les requêtes sont résolues par **Azure DNS** et envoyées vers la couche de sécurité et de répartition du trafic.
2. **Sécurité et Répartition du Trafic**
   * **Azure Front Door** assure la **répartition intelligente du trafic** entre les App Services (frontend et backend), propose un **load balancing global** et une protection DDoS au niveau applicatif.
   * L’**autoscaling** est configuré pour créer ou supprimer des instances supplémentaires selon la charge, garantissant ainsi la **scalabilité** et un temps de réponse optimal.
   * Les **Managed Identities** sécurisent l’accès entre les services internes (ex. communication avec le Container Registry).
3. **Frontend : Azure App Service**
   * Le code Angular est déployé sur une App Service (plan PaaS).
   * Avantages :
     + **Scalabilité automatique** : possible d’augmenter le nombre d’instances en cas de forte demande.
     + **Facilité de mise à jour** : un simple push du code ou de l’image Docker.
     + **Séparation claire** entre l’application Angular et l’OS sous-jacent (gestion prise en charge par Azure).
4. **Backend : Azure App Service**
   * L’API **Flask** (Python) et les modèles de machine learning (TensorFlow/OpenCV) tournent dans une autre App Service pour mieux séparer la logique métier et améliorer la maintenance.
   * **Load Balancer** intégré : répartit la charge entre plusieurs instances de l’API si nécessaire.
   * Communication sécurisée avec le **Storage Account** pour stocker/récupérer les données (Blob Storage).
5. **Azure Container Registry (ACR)** & **CI/CD**
   * Les images Docker des services (frontend, backend) sont stockées dans l’**ACR**.
   * Un pipeline **GitHub Actions** est mis en place pour (1) builder, (2) tester et (3) pousser automatiquement ces images dans l’ACR.
   * L’App Service peut ensuite les récupérer pour procéder au déploiement automatique.
6. **Azure Storage Account**
   * Le **Blob Storage** sert d’espace de stockage pour les fichiers d’images, les vidéos, les modèles entraînés (optionnel) et les logs.
   * Gestion de l’authentification via des clés d’accès sécurisées ou via **Managed Identities** pour éviter le partage de secrets.
7. **Réseau Virtuel (VNet)** & **Sécurité**
   * **echo-sign-lan** : Un **VNet** isole le trafic entre les App Services et les autres ressources (ACR, Storage).
   * **Pare-feu Azure** et **Protection DDoS** : Renforcent la sécurité et permettent une protection au niveau de la couche réseau.
8. **Monitoring & Logging**
   * **Azure Monitor** et **Application Insights** collectent en continu les métriques, traces et événements.
   * Des alertes sont configurées pour détecter une hausse anormale de la latence ou des erreurs HTTP 5xx.

## Justification des Choix

### 3.1 PaaS vs IaaS

* **IaaS (Infrastructure as a Service)**
  + Offre un contrôle complet (choix du système d’exploitation, configuration réseau avancée, etc.).
  + En contrepartie, **complexité plus élevée** (maintenance des VM, correctifs système, sécurité étendue).
* **PaaS (Platform as a Service)**
  + Gère automatiquement les contraintes d’infrastructure (mises à jour, patchs, autoscaling).
  + Permet de **se concentrer** sur le code et la logique métier.
  + **Coûts maîtrisés** grâce à la facturation à l’usage (calcul, stockage, bande passante).

En cohérence avec nos besoins en disponibilité, en flexibilité et en scalabilité, nous avons opté pour des **App Services PaaS**, combinés à un **ACR** et à un **Storage Account** géré. Cela nous libère des tâches d’administration complexes et offre une montée en charge quasi transparente pour l’utilisateur.

### 3.2 Avantages Clés de PaaS

* **Load Balancing Automatique** : Azure Front Door, intégré avec App Service, répartit la charge pour éviter tout engorgement sur un nœud particulier.
* **Scalabilité Simplifiée** : Possibilité de créer de nouvelles instances en quelques clics ou automatiquement grâce aux règles d’autoscaling.
* **Gestion du Trafic** : Azure DNS et Front Door assurent un routage intelligent et une protection DDoS.
* **CI/CD Intégré** : Les déploiements sont fluides grâce au pipeline GitHub Actions connectant ACR et App Services.
* **Monitoring et Diagnostics** : Azure Monitor et Application Insights offrent une visibilité en temps réel sur la santé de l’application.

### 4. Implémentation

Le déploiement de l’infrastructure **EchoSign** s’est déroulé en plusieurs phases :

1. **Création du Resource Group**
   * Définition du groupe echosign pour regrouper et gérer tous les éléments (App Services, ACR, VNet, etc.).

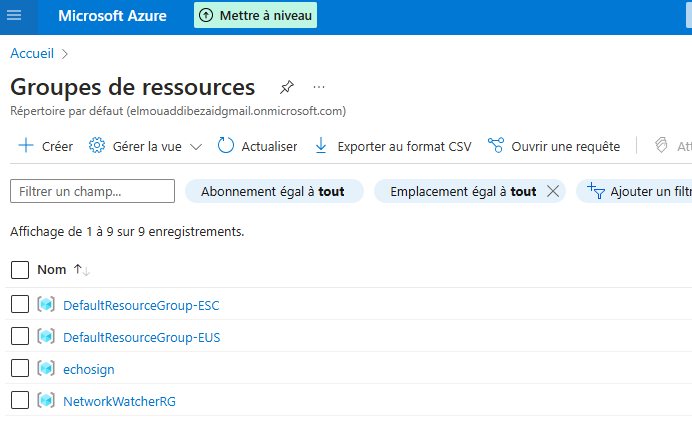


Figure 7 - Groupe de ressources

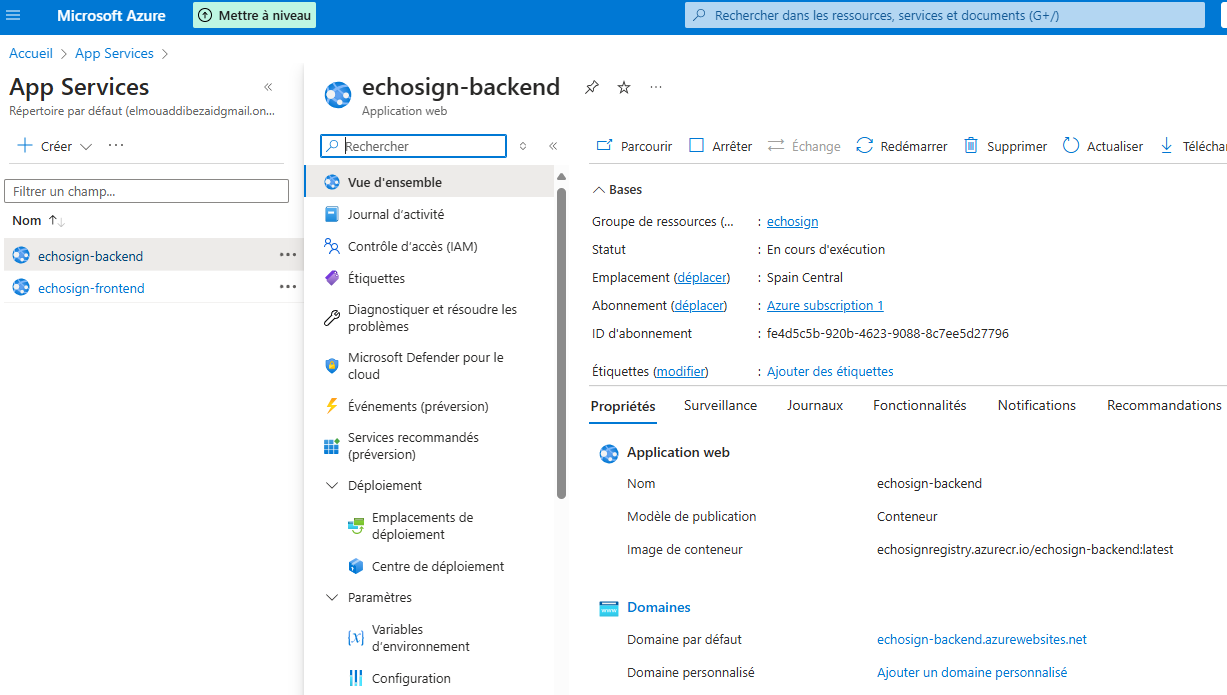
1. **Déploiement des App Services**
   * **echosign-frontend** : Héberge l’application Angular.
   * **echosign-backend** : Héberge l’API Flask et les modèles ML.

Figure 8 - App Services

1. **Configuration d’Azure Container Registry (ACR)**
   * Mise en place de l’ACR EchoSignRegistry.

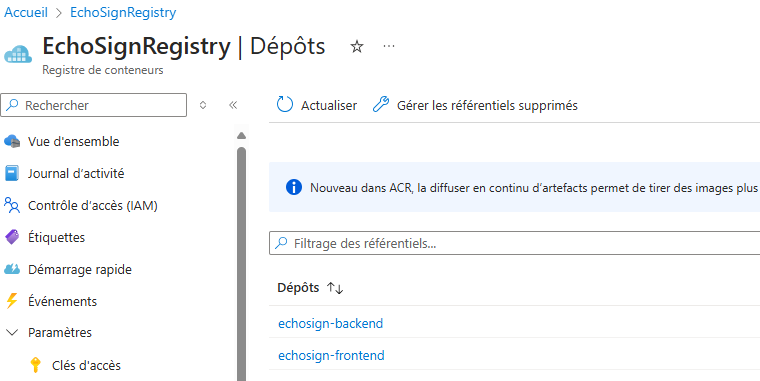


Figure 9 - Azure Registre de conteneurs

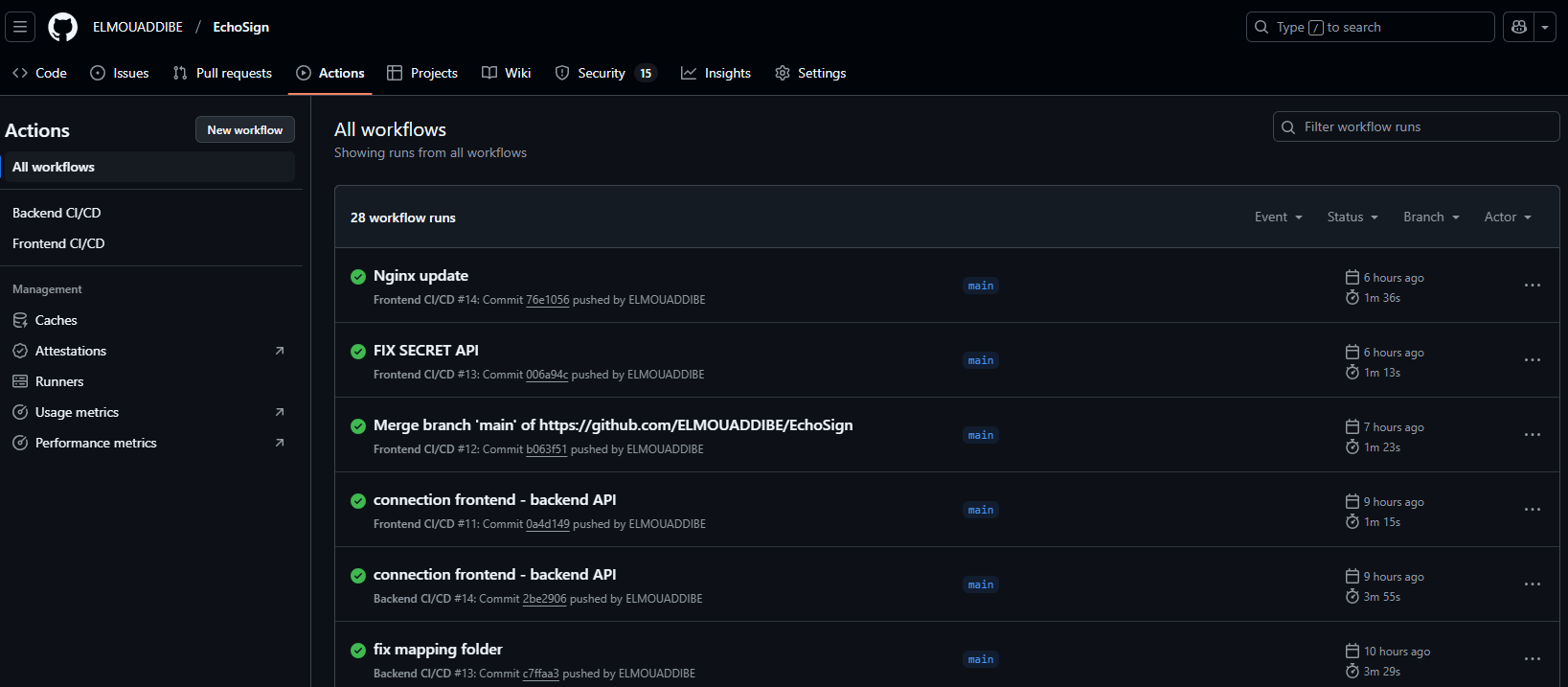
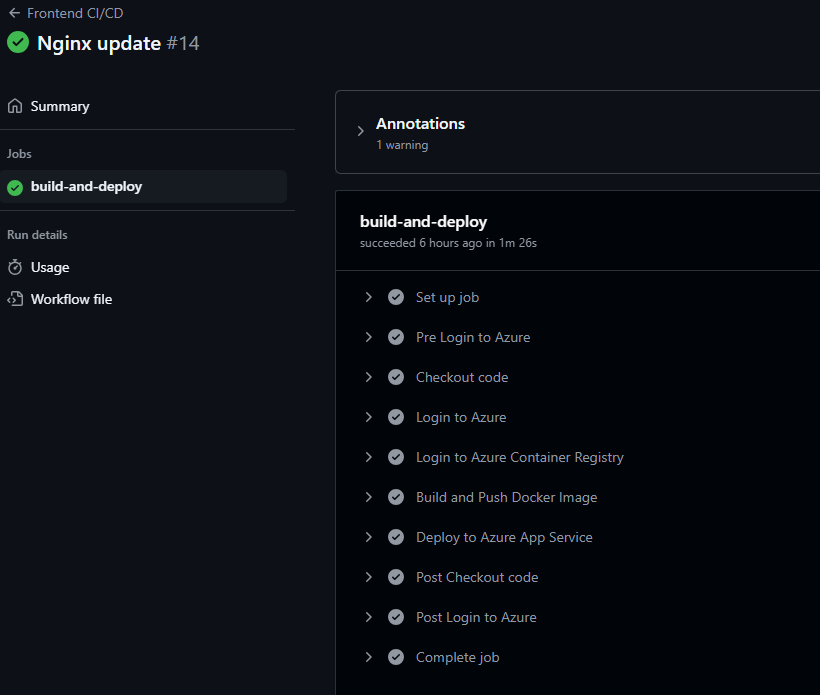
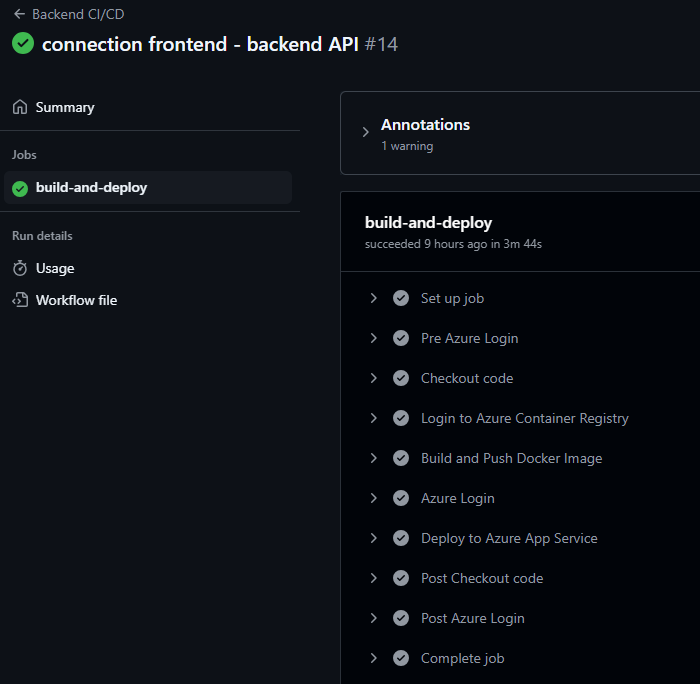
**Pipeline GitHub Actions :** build/test des images Docker → push vers l’ACR → pull automatique côté App Services.

Figure 10 - Workflows CI/CD GitHub Actions

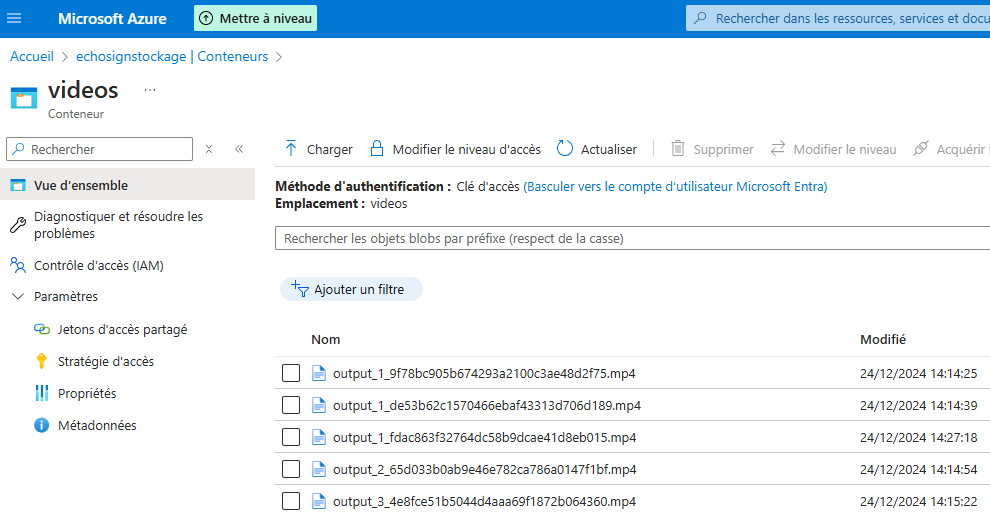
1. **Stockage (Azure Storage Account)**
   * Création de echosignstockage (Blob Storage) pour les ressources multimédia, logs et backups éventuels.
   * Intégration via une **Managed Identity** pour éviter la gestion manuelle des clés d’accès.

Figure 11 - Azure Storage Account

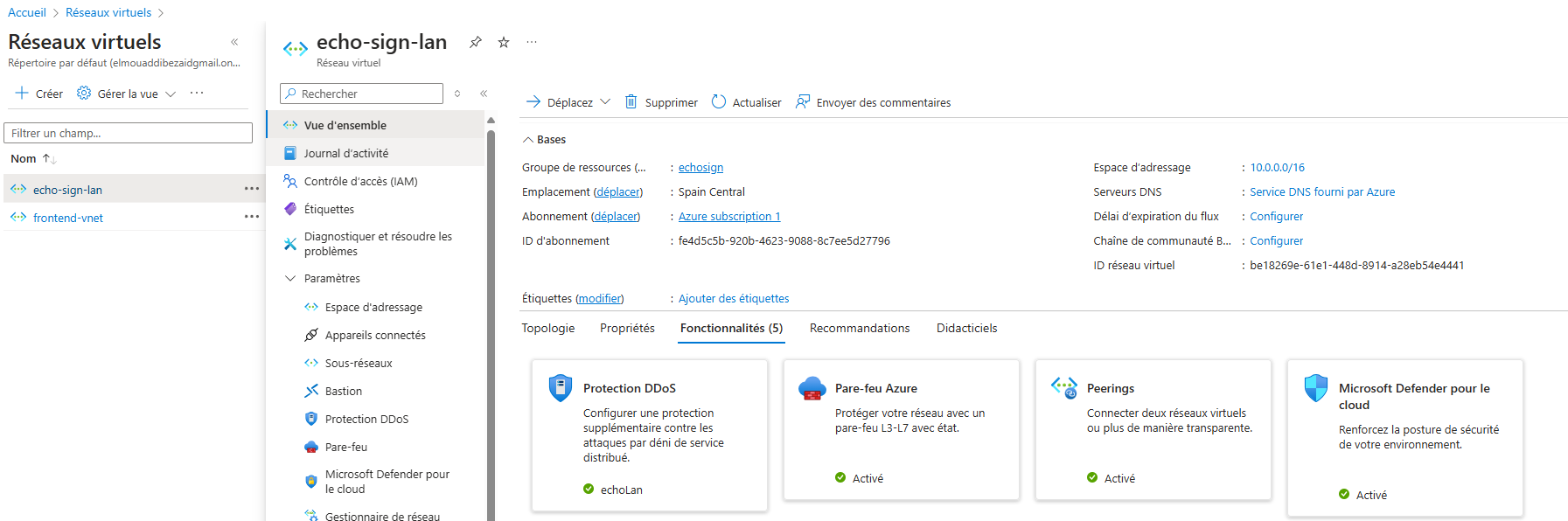
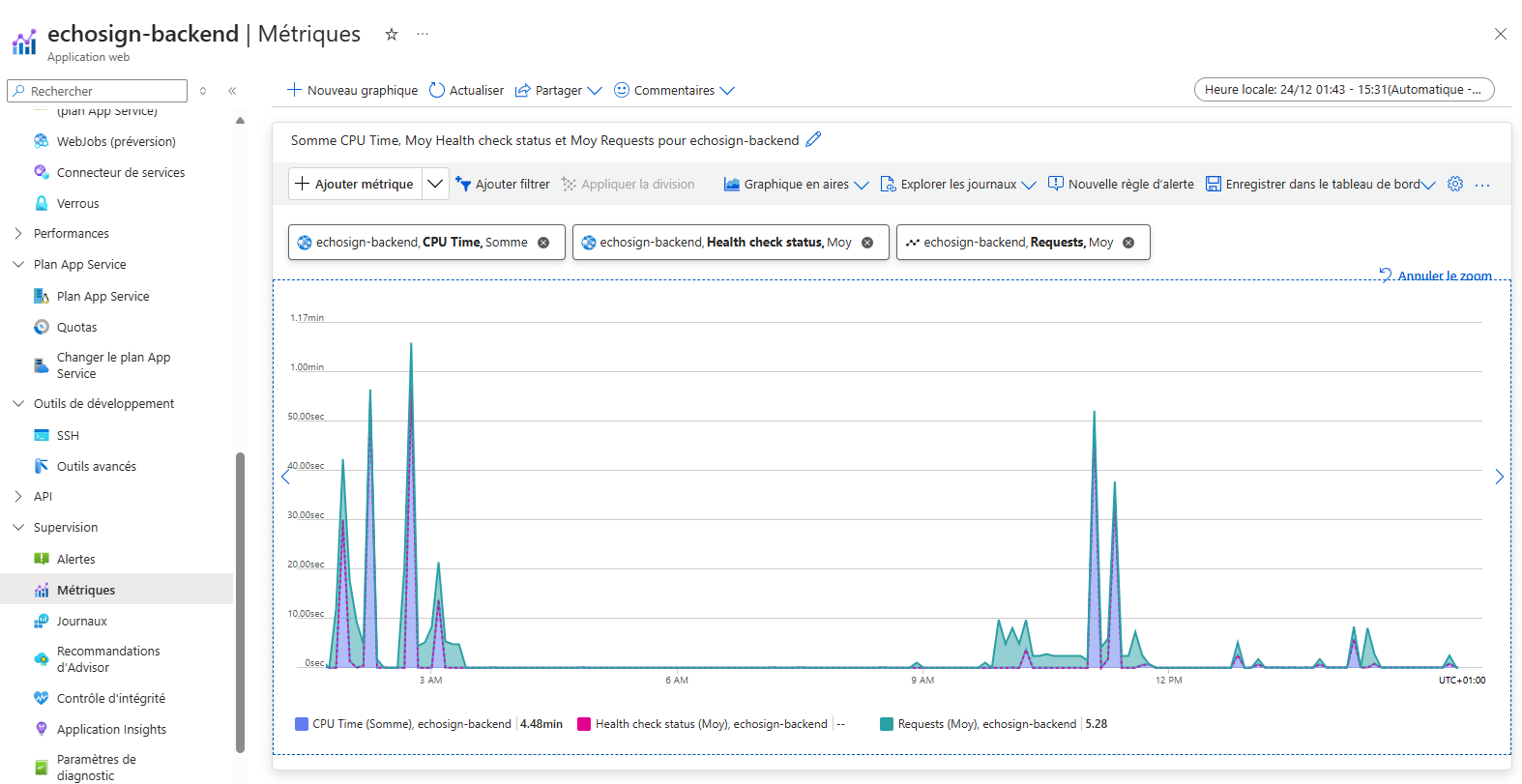
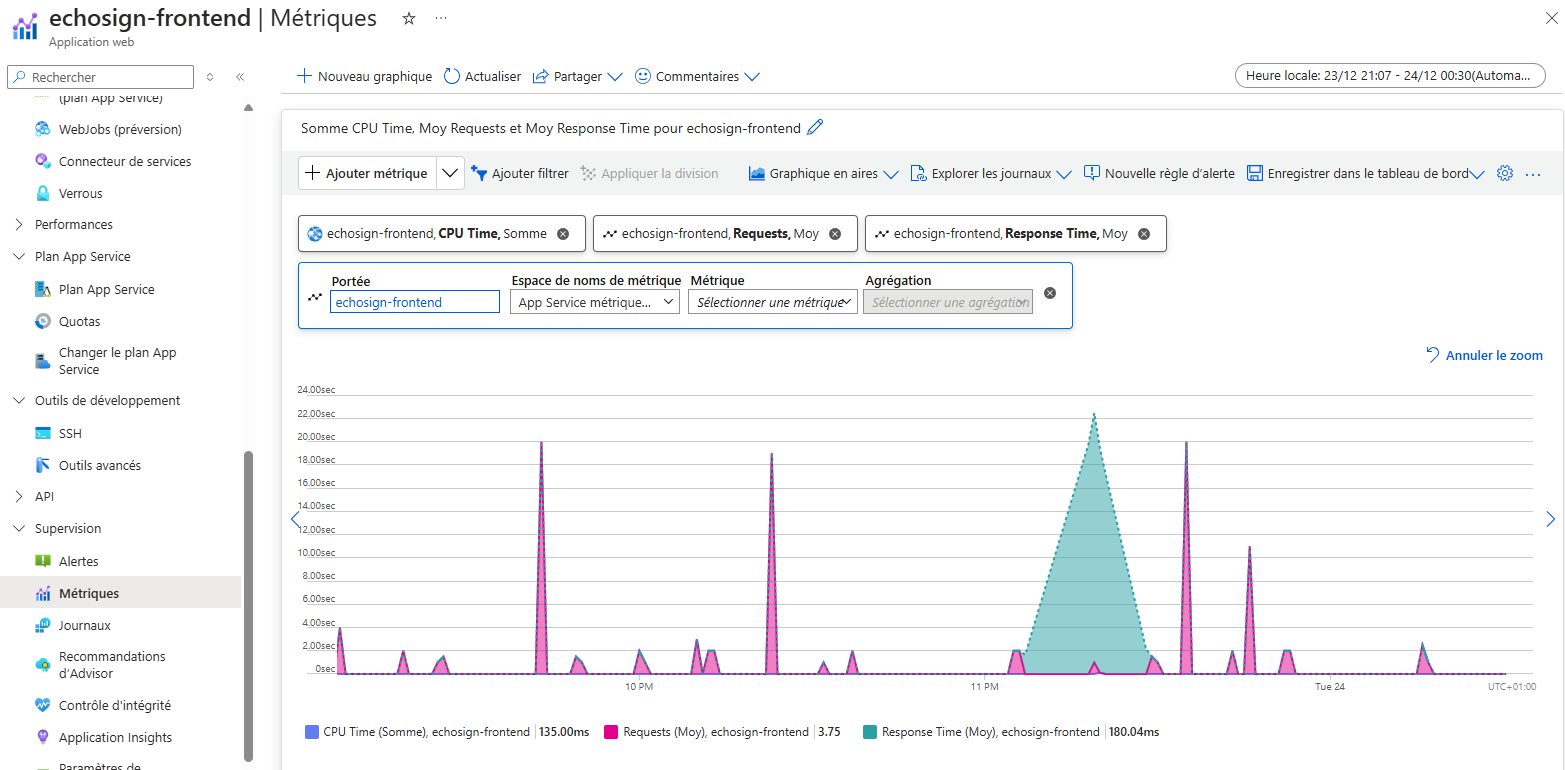
1. **Réseau Virtuel (echo-sign-lan)**
   * Configuration d’un sous-réseau isolé pour limiter les accès publics non autorisés.
   * Activation de la protection DDoS au niveau du réseau.

Figure 12 - Creation de Reseau Virtuel

1. **Azure Front Door & DNS**
   * Configuration de **Azure Front Door** pour répartir le trafic et protéger le site contre les attaques de type DDoS.
   * Association du nom de domaine echo-sign.tech (via **Azure DNS**) à l’App Service (frontend).
2. **Monitoring et Logging**
   * **Application Insights** : collecte et analyse des métriques (CPU, mémoire) et des logs (erreurs, temps de réponse).
   * **Azure Monitor** : définition de règles d’alerte (ex. taux d’erreur élevé, saturation CPU).



# Chapitre IV : Evaluation et Résultat

Dans ce chapitre, nous présentons l’évaluation de notre application **EchoSign**, qui traduit la Langue des Signes Marocaine (MSL) en texte et en parole (et inversement). Nous revenons sur nos méthodes de test, les performances constatées, les retours utilisateurs, ainsi que l’impact global du déploiement sur **Azure Cloud**. Enfin, nous mettons en avant les pistes d’amélioration identifiées.

### 1. Méthodologies de Test

Pour garantir la qualité et la fiabilité de **EchoSign**, plusieurs types de tests ont été mis en œuvre :

1. **Tests Unitaires**
   * **Frontend (Angular)** : Vérification du rendu des composants et de la bonne gestion des interactions utilisateur.
   * **Backend (Flask API)** : Validation du traitement des requêtes, des réponses, et de la gestion des erreurs.
   * **Modèles de Machine Learning** : Évaluation de la précision, du rappel et de la robustesse dans la reconnaissance et la traduction des gestes MSL.
2. **Tests d’Intégration**
   * **API** : Validation de la circulation des données entre le frontend et le backend via les appels REST.
   * **Modèles** : Vérification du flux complet, depuis la capture vidéo jusqu’à l’affichage des résultats (texte/voix/gestes).

### 2. Résultats

**2.1 Précision**

* **Reconnaissance des Gestes** : Nos modèles ont atteint un taux de précision moyen de 91%, permettant une traduction en temps réel fiable.
* **Synthèse Vocale** : Les retours indiquent une voix naturelle et claire, améliorant grandement l’expérience utilisateur.

**2.2 Utilisabilité**

* **Interface Utilisateur** : L’usage combiné d’**Angular** et de **Tailwind CSS** offre une interface réactive et ergonomique.

**2.3 Performance**

* **Temps de Réponse** : Les appels API et le traitement des modèles ont été optimisés pour fournir des traductions quasi instantanées.
* **Fiabilité** : Les tests de montée en charge révèlent une grande stabilité, sans pannes majeures. L’application gère efficacement plusieurs utilisateurs simultanés.

**2.4 Impact**

* **Inclusion Sociale** : EchoSign contribue à réduire l’isolement des personnes malentendantes en favorisant une communication fluide et accessible.
* **Outil Éducatif** : L’application est également utilisée comme support pédagogique pour apprendre la MSL, tant par les étudiants que par les enseignants.

### Interfaces Utilisateur

Notre application propose plusieurs interfaces pour répondre aux besoins variés des utilisateurs :

1. **Home Page**
   * Accès rapide aux fonctionnalités de traduction (MSL ↔ Texte/Voix).
   * Menu de navigation (Accueil, À propos, Contact, etc.) pour une prise en main simple.



Figure 13 - Home Page of EchoSign

1. **Traduction de la MSL vers Texte/Voix**
   * **Flux Webcam** : Capture en direct des gestes.
   * **Résultats** : Affichage du texte reconnu et possibilité d’écoute via un bouton “Écouter”.
   * **Options de Traduction** : Choix de la langue de sortie (Darija, français, anglais).A black and white sound waves

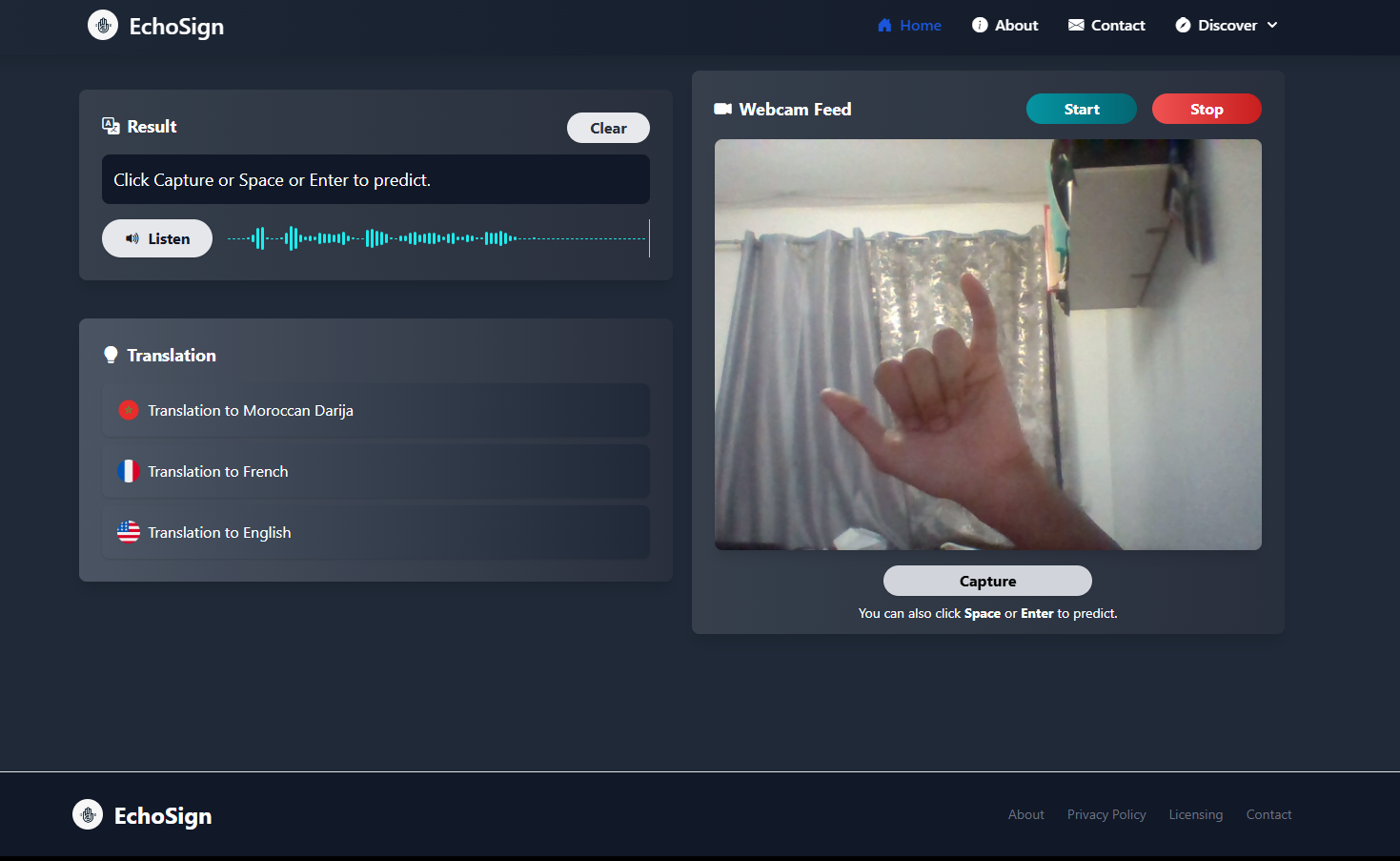
     Description automatically generated

Figure 14 - Sign to Text and Voice Translation Interface

1. **Traduction du Texte/Voix vers la MSL**

* **Saisie de Texte ou Vocale** : Conversion en gestes MSL représentés sous forme de vidéo ou d’animations.
* **Interface de Type Chat** : Permet de visualiser l’historique des entrées et les gestes correspondant à chaque message.
* A screenshot of a computer screen

  Description automatically generated**"Exprimez-vous"** : Une interface de type chat où les utilisateurs peuvent voir le texte qu’ils saisissent et soumettre leur demande de traduction. La vidéo correspondante des gestes est affichée à gauche.

Figure 15 - Text and Voice to Sign Translation Interface

### Leçons Apprises

* **Collaboration** : L’usage intensif d’outils comme WhatsApp, Google Meet et GitHub (Issues, Pull Requests) a facilité la coordination de l’équipe.
* **Intégration Technologique** : Combiner Angular, Flask, TensorFlow et OpenCV a nécessité une planification rigoureuse et des tests réguliers.
* **Focus Utilisateur** : Les retours constants ont permis d’affiner l’interface et d’améliorer la pertinence des traductions.
* **Cloud vs Local** : La migration vers Azure a simplifié la maintenance et accru la scalabilité, nous libérant des contraintes d’infrastructure gérées auparavant en local.

### Difficultés Rencontrées

* **Collecte et Prétraitement de Données :** Obtenir des jeux de gestes variés et les étiqueter efficacement a été coûteux en temps.
* **Entraînement des Modèles** : L’optimisation des hyperparamètres (TensorFlow/OpenCV) a exigé des ressources machines importantes et un suivi précis.
* **Intégration API + ML** : Assurer une latence faible et une orchestration fluide entre le frontend et le backend a nécessité des tests d’intégration approfondis.

### Bilan du Déploiement sur Azure Cloud

Le déploiement de **EchoSign** sur **Azure Cloud** a permis de répondre aux exigences de fiabilité, de scalabilité et de simplicité de maintenance :

1. **Haute Disponibilité et Scalabilité**
   * Les **App Services** s’adaptent automatiquement à la charge, créant ou libérant des ressources selon les besoins.
   * La gestion du trafic et la protection DDoS via **Azure Front Door** renforcent la résistance de l’application face aux pics d’activité et aux attaques.
2. **Sécurité et Fiabilité**
   * La combinaison d’un **VNet**, de la **protection DDoS** et de **Managed Identities** protège l’application et les données contre les tentatives d’intrusion.
3. **Coûts Optimisés**
   * Grâce au **Pricing Calculator**, nous avons pu estimer précisément le coût mensuel de l’infrastructure (App Services, Container Registry, Storage, etc.).
   * Le paiement à l’usage (pay-as-you-go) évite de supporter le coût d’un serveur inactif ou sous-utilisé.
4. **Maintenance Réduite**
   * En se libérant des tâches de configuration et de maintenance de serveurs physiques (mises à jour, patchs de sécurité, backups, etc.), l’équipe se focalise sur l’amélioration fonctionnelle et la qualité des modèles ML.
   * Les pipelines CI/CD intégrés (GitHub Actions) facilitent la mise à jour continue et la publication de nouvelles fonctionnalités.

### Défis et Pistes d’Amélioration

* **Versions de Bibliothèques ML** : Il est essentiel pour nous de maintenir une gestion rigoureuse du Dockerfile et de la documentation liée aux compatibilités (TensorFlow, OpenCV, etc.) afin de garantir la stabilité des modèles.
* **Gestion des Coûts** : Nous veillons à ajuster précisément le dimensionnement de l’App Service Plan et de l’espace de stockage, afin d’éviter tout risque de dérive budgétaire et de respecter les contraintes financières du projet.
* **Sécurité et Services Avancés** : En cas de croissance importante du trafic, nous envisageons de migrer vers **Azure Kubernetes Service (AKS)** ou d’exploiter **Azure Cognitive** **Services** afin de bénéficier de fonctionnalités ML plus poussées (reconnaissance vocale, etc.) et d’optimiser l’architecture globale.

En définitive, le passage de l’environnement local à **Azure Cloud** nous a permis de professionnaliser l’infrastructure, d’optimiser la réactivité et de renforcer la fiabilité de l’application **EchoSign**. Nous avons ainsi pu consacrer nos efforts au cœur du projet – la traduction en temps réel de la Langue des Signes Marocaine – au profit direct de nos utilisateurs finaux. Nous sommes fiers de ce que nous avons accompli et restons motivés à poursuivre son amélioration.

# Conclusion

Le développement de notre application de traduction de la Langue des Signes Marocaine (**MSL**) s’est révélé particulièrement enrichissant, tant sur le plan technique qu’humain. Grâce à une planification et à un travail d’équipe soutenu, nous avons créé une solution permettant de convertir la MSL en texte/parole et réciproquement, améliorant nettement la communication et l’inclusion de la communauté malentendante.

Le choix **d’Azure Cloud** pour le déploiement a joué un rôle déterminant : l’utilisation **d’infrastructures PaaS** et des outils de monitoring et de sécurité intégrés nous a permis de gagner en fiabilité, en scalabilité et en sérénité. Nous avons ainsi pu concentrer nos efforts sur l’amélioration constante des modèles d’apprentissage automatique et la conception d’une interface conviviale.

Si plusieurs défis ont jalonné notre parcours comme la collecte de données et l’intégration des modèles la collaboration et l’apprentissage continu nous ont permis de les surmonter avec succès. Nous prévoyons d’enrichir encore l’application en étendant le vocabulaire gestuel, en perfectionnant la précision des traductions et en intégrant de nouvelles fonctionnalités pour répondre à un plus large public.

En définitive, nous sommes fiers d’avoir conçu une application qui aide à combler le fossé de communication autour de la MSL. Notre expérience confirme l’importance du **cloud** (et notamment **d’Azure Cloud**) pour déployer rapidement des solutions innovantes, accessibles et durables. Nous restons pleinement engagés dans l’évolution de ce projet afin de rendre la traduction de la MSL toujours plus performante et inclusive.

# Bibliography

|  |  |
| --- | --- |
| [1] | Kayo Yin et Jesse Read, «Better Sign Language Translation with STMC-Transformer,» chez *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, 2020. |
| [2] | Y. X. D. L. Sijie Yan, «Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action,» 2018. |
| [3] | T. Liu, W. Zhou et H. Li, «Sign language recognition with long short-term memory,» chez *2016 IEEE International Conference on Image Processing (ICIP)*, Phoenix, AZ, USA, 2016. |
| [4] | J. W. Thad Starner, «Real-Time American Sign Language,» chez *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, Cambridge, 1998. |