P_Cloud 346

Alban Segalen – mid3b
Yosef Nademo – cid3b
Vennes
32p
Gaël Sonney

Table des matières

1 SPÉCIFICATIONS

1.1 Titre

Cloud Webapp

1.2 Description

Déployer l'application « Passion Lecture » sur Azure.

1.3 Matériel et logiciels à disposition

- PC ETML
- Groupe de ressource sur Azure
- Code initial de l'application sur GitHub

1.4 Prérequis

- Avoir suivi le module 346 « Concevoir et réaliser des solutions cloud »

1.5 Cahier des charges

1.5.1 Fonctionnalités requises (du point de vue de l'utilisateur)

L'application « Passion Lecture » permet à chaque utilisateur, d'ajouter, de modifier et de supprimer des livres. Il est possible de commenter et de noter chaque livre.

1.5.2 Travail à réaliser par l'apprenti

- Déployer l'application sur Azure en utilisant des services PaaS
- Modifier l'application pour intégrer l'authentification MSAL
- Migrer une fonctionnalité de l'application en fonction serverless
- Ajouter un pipeline de déploiement automatique

1.5.3 Si le temps le permet …

- Mettre en place un environnement de staging
- Stocker les secrets dans Azure Key Vault
- Mettre en place une sauvegarde automatisé de la base de données
- Configurer un CDN
- Ajouter un Application Gateway ou Azure Front Door

1.6 Les points suivants seront évalués

- Un application web hébergée sur Azure
- Un rapport répondant à ces questions / thèmes
- Une présentation en classe du travail en fin de projet
- Le code source documenté et présent sur Github
- Le site doit aussi pouvoir être lancé en local (npm run start)

1.7 Validation et conditions de réussite

- Compréhension du travail
- Possibilité de transmettre le travail à une personne extérieure pour le terminer, le corriger ou le compléter
- Etat de fonctionnement du produit livré

2 PLANIFICATION INITIALE

Au début du projet, nous avons organisé une séance de planification afin de définir clairement les objectifs et la répartition du travail.

Nous avons choisi ensemble une application créée lors d'un projet précédent, et nous avons réparti les tâches nécessaires pour ce nouveau mandat. La première étape consistait simplement à faire fonctionner l'application en local, afin de s'assurer que tout marchait correctement avant de penser au déploiement.

Une fois cela fait, nous avons commencé à travailler en parallèle sur plusieurs points : préparer le déploiement sur Azure en utilisant des services PaaS, mettre en place la base de données centralisée utilisée par toute la classe, et commencer à comprendre comment fonctionne l'authentification MSAL. Nous devions également préparer un petit schéma explicatif pour notre enseignant avant d'intégrer réellement MSAL dans l'application.

Nous avions aussi prévu de migrer une fonctionnalité en mode serverless et d'ajouter un pipeline de déploiement automatique basé sur des déclencheurs GitHub (push). Pour suivre l'avancement, nous faisions des réunions courtes de type Daily Scrum, où chacun expliquait ce qu'il avait fait, les problèmes rencontrés et ce qu'il comptait faire ensuite. Grâce à ces échanges, nous avons ajusté le planning quand c'était nécessaire. Toutes les mises à jour ont été notées et ont donné plusieurs versions du planning initial.

3 RÉALISATION

3.1 Problèmes rencontrés

Etape	Problème	Solution
Création de la DB	Pas possible de créer des utilisateurs depuis le GUI	Se connecter en ligne de commande
	Pas possible de se connecter en ligne de commande car la DB est isolée dans son réseau	Créer une VM sur Azure dans le même sous-réseau que la DB
	Mauvais hôte pour les utilisateurs mysql : user@localhost	Renommer les utilisateurs en user@% pour qu'ils puissent se connecter depuis n'importe quel hôte
MSAL	**Gestion de deux types de tokens (standard + MSAL)** : L'application devait supporter à la fois les utilisateurs classiques avec token JWT et les utilisateurs Microsoft OAuth2 via MSAL.	Création d'un fichier centralisé auth.js avec des variables réactives (token, msJwt, currentUserId) et des computed properties (activeToken, isLoggedIn, userId, authHeader) pour gérer de manière uniforme l'état de connexion.

	Synchronisation avec le stockage local (localStorage) : Les tokens MSAL et classiques devaient être synchronisés avec l'état réactif de Vue pour que les composants (Header, Sidebar, Views) puissent se mettre à jour automatiquement.	Implémentation de fonctions initAuth, setToken, setMsalToken et logout pour initialiser et mettre à jour l'état réactif depuis localStorage. L'état réactif est désormais utilisé par tous les composants.
	Flux de connexion MSAL backend : Nécessité d'échanger le code Microsoft contre un token, de créer l'utilisateur MSAL dans la base de données si inexistant, et de générer un JWT utilisable dans l'application.	Création de routes dédiées /auth/msal/login et /auth/msal/callback dans le backend Express. Les utilisateurs MSAL sont créés avec un mot de passe aléatoire hashé pour respecter la contrainte allowNull: false. Un JWT est généré et renvoyé au frontend pour l'authentification.
	Compatibilité avec les vues existantes : Toutes les vues de l'application (ex. RegisterView.vue, ProfileView.vue, MyBooksView.vue, CreateBookView.vue, EditBookView.vue, BookDetailsView.vue) devaient fonctionner avec les deux types de token sans rupture de logique.	Refactorisation de chaque vue pour utiliser les helpers du fichier auth.js (getToken(), getCurrentUserId(), isLoggedIn) afin d'assurer que les requêtes HTTP et les composants s'adaptent automatiquement au type de token présent.
	Détection et affichage cohérent dans les composants : Les composants globaux comme Header.vue et Sidebar.vue devaient refléter correctement si un utilisateur était connecté via MSAL ou token classique, et proposer les bonnes options (login, logout, profil).	Adaptation de tous les composants pour utiliser isLoggedIn et userId du helper. Header.vue affiche dynamiquement les liens Login/Register ou Profil/Logout selon le type de token. La fonction logout() supprime tous les tokens et met à jour l'état réactif, garantissant que l'UI se synchronise immédiatement.
Déploiement sur App Service	Pas possible de se connecter à la DB	Créer un sous-réseau dans le réseau privé de la DB et l'intégrer dans l'app service
	Pas possible de se connecter à la DB	Enlever les credentials en dur du backend et utiliser des variables d'environnements
	Pas possible de se connecter à la DB	Ajouter le certificat ssl pour se connecter à la DB : Azure demande un CA pour se connecter à un serveur mysql flexible
	Pas possible d'intégrer un sous-réseau	Passer du plan F1 au plan B1

	L'application ne démarrait pas	Uploader seulement le dossier de build (/dist) vers /site/wwwroot et changer la commande de démarrage de l'application à npx serve -s

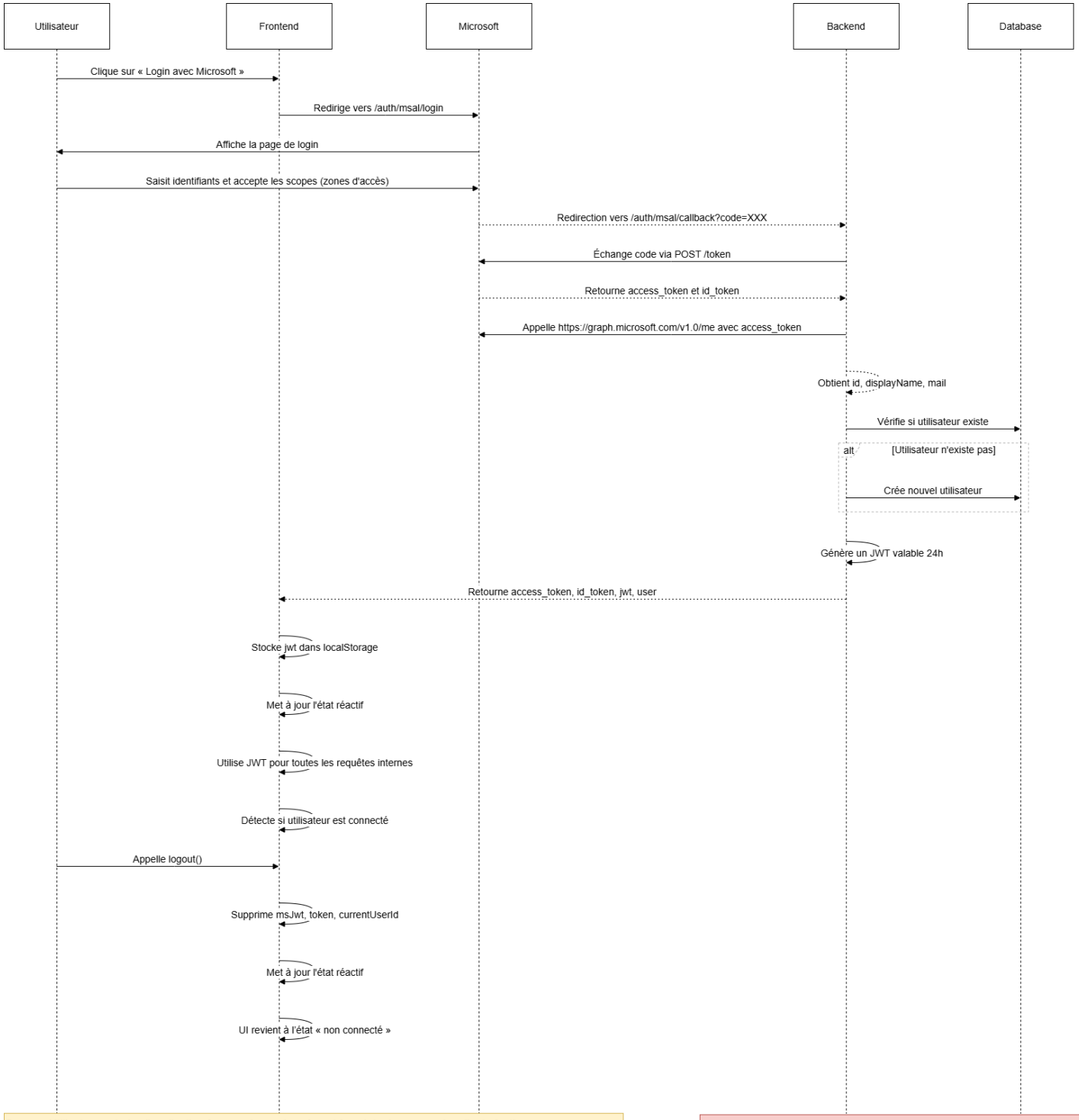
3.2 Connaissances acquises

Ce projet nous a permis d'améliorer nos compétences en technologies cloud, en déployant notre application sur des App Services et pas simplement sur des VMs. Nous avons également compris le fonctionnement de l'authentification MSAL en l'intégrant dans notre application. Nous avons également appris comment intégrer une fonction serverless dans notre projet.

3.3 Répartition des tâches

Yosef : déploiement local, authentification MSAL, intégration de MSAL dans chaque fonctionnalité
Alban : déploiement sur Azure, création de la DB

3.4 Fonctionnement de l'authentification MSAL



Scopes: Utilisateur autorise l'application à recevoir ces données et à maintenir une session d'autorisation.

MSAL (Microsoft Authentication Library) : Bibliothèque utilisée pour gérer la connexion via Microsoft OAuth2. Elle facilite la récupération des tokens, la gestion des sessions et l'intégration avec Azure AD.

LocalStorage synchronisé avec Vue : Mécanisme permettant de stocker les tokens côté client et de mettre à jour automatiquement les composants Vue lorsque l'état de connexion change.

Access Token : Token temporaire utilisé pour authentifier les requêtes de l'application vers l'API. Il est fourni par Microsoft après l'autorisation de l'utilisateur.

Flux OAuth2 / MSAL :

Processus en plusieurs étapes :

1. Redirection vers la page Microsoft login.
2. Saisie des identifiants et acceptation des scopes.
3. Retour au backend avec un code d'autorisation.
4. Échange du code contre un Access Token et un ID Token.
5. Création de l'utilisateur MSAL en base si inexistant.
6. Génération d'un JWT interne pour authentification dans l'application.

Étapes principales du flux :

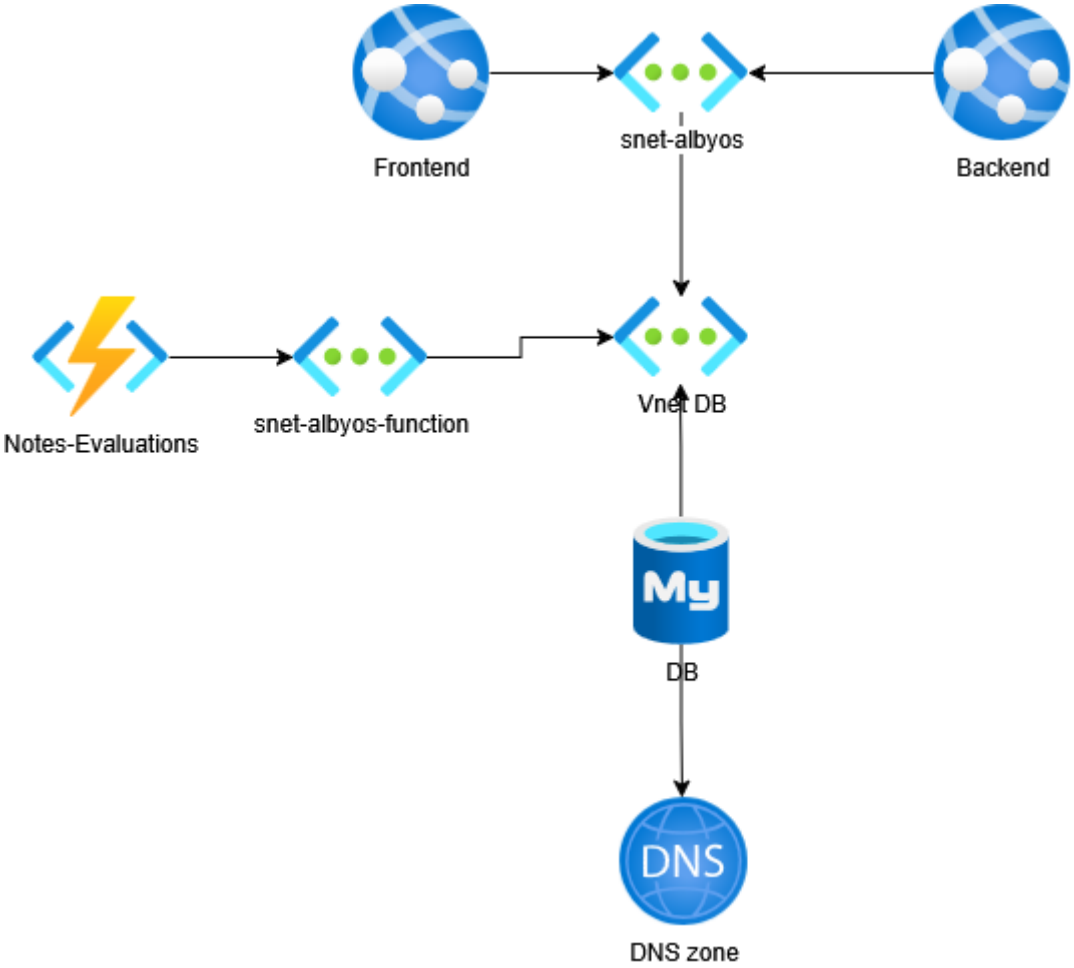
1. **Utilisateur initie la connexion Microsoft depuis le frontend**
   - L'utilisateur clique sur « Login avec Microsoft » dans LoginView.vue.
   - Le navigateur est redirigé vers l'URL d'autorisation Microsoft (/auth/msal/login).
2. **Redirection vers Microsoft OAuth2**
   - Microsoft affiche la page de login et demande l'autorisation à l'utilisateur.
   - L'utilisateur saisit ses identifiants Microsoft et accepte les scopes demandés (openid profile email offline_access).
3. **Retour du code d'autorisation au backend**
   - Après succès, Microsoft redirige vers /auth/msal/callback?code=XXX.
   - Le backend Express récupère ce **code d'autorisation**.
4. **Échange du code contre un token Microsoft**
   - Backend fait une requête POST vers Microsoft (/token) avec :
     - client_id, client_secret
     - redirect_uri
     - code reçu
     - grant_type=authorization_code
   - Microsoft retourne un **access_token** et un **id_token**.
5. **Récupération des informations utilisateur via Microsoft Graph**
   - Backend utilise access_token pour appeler https://graph.microsoft.com/v1.0/me.
   - Obtention des informations : id, displayName, mail.
6. **Création ou récupération de l'utilisateur dans la base de données**
   - Si l'utilisateur n'existe pas (ms_id absent), créer un nouvel utilisateur avec :
     - username = displayName
     - email = mail
     - ms_id = id
     - hashed_password généré aléatoirement pour respecter allowNull: false.
7. **Génération d'un JWT interne pour l'application**
   - Backend génère un JWT (jwt.sign({ userId }, OIDC_SECRET)) valable 24h.
   - Ce JWT sera utilisé pour authentifier l'utilisateur dans les vues Vue.js.
8. **Retour du JWT et des informations utilisateur au frontend**
   - Frontend reçoit :
     - access_token (MS)
     - id_token (MS)
     - jwt (application interne)
     - user (infos utilisateur)
   - jwt est stocké dans localStorage via setMsalToken().
   - L'état réactif de Vue (auth.js) est mis à jour : msJwt, currentUserId, isLoggedIn.
9. **Utilisation du JWT pour les requêtes internes**
   - Toutes les requêtes HTTP depuis Vue utilisent authHeader pour inclure le JWT.
   - Les composants (MyBooksView, ProfileView, Header) détectent automatiquement si l'utilisateur est connecté et MSAL ou classique.
10. **Déconnexion**
    - La fonction logout() supprime msJwt, token et currentUserId de localStorage et met à jour l'état réactif.
    - L'UI revient à l'état « non connecté ».

3.5 .env

Voici une liste des variables d'environnement qu'utilise notre application.

Variable d'environnement	Utilité	Environnement
DB_HOST	L'adresse de notre base de données	Local/Prod
DB_NAME	Le nom de la base de données	Local/Prod
DB_PASSWORD	Le mot de passe pour se connecter à la DB	Local/Prod
DB_PORT	Le port de la base de données	Local/Prod
DB_USERNAME	L'utilisateur utilisé pour se connecter à la DB	Local/Prod
WEBSITES_INCLUDE_CLOUD_CERTS	Inclue les certificats pour se connecter à la DB	Prod

3.6 Architecture de l'application



Il y a 2 App Services, 1 pour le backend et un autre pour le frontend. Un sous-réseau est intégré dans l'App Service pour la connexion à la base de donnée, qui est dans un réseau privé sans accès public. Une fonction Serverless qui gère les notes et commentaires est également connectée à la DB.

4 CONCLUSION

4.1 Bilan personnel

4.1.1 Yosef

La réalisation de ce projet m'a permis de développer trois compétences techniques majeures et complémentaires.

Premièrement, j'ai mené à bien la migration du module "Notes et Évaluations" vers une architecture Serverless. Cette tâche m'a appris à déployer une Azure Function et à résoudre les défis liés à la connectivité avec une base de données MySQL au travers d'un réseau virtuel (VNet).

Deuxièmement, l'intégration de la bibliothèque MSAL m'a permis de maîtriser les standards actuels de sécurité. J'ai pu mettre en pratique les flux d'authentification modernes en déléguant la gestion des identités à Microsoft, garantissant ainsi un accès sécurisé à l'application.

Enfin, ce projet a considérablement renforcé mon aisance générale avec le Cloud Azure. La gestion des ressources, la configuration des variables d'environnement et l'administration via le portail font désormais partie de mes acquis.

Ces trois volets — architecture logicielle, sécurité et infrastructure Cloud — constituent ensemble un socle solide pour mon parcours professionnel futur.

4.1.2 Alban

J'ai apprécié ce projet car j'ai appris à utilisé des App Services et à intégrer des sous-réseaux, ce qui m'a permis de découvrir plus de fonctionnalités d'Azure tout en me familiarisant avec des problèmes réels. En effet, je pense que la mise en production est l'étape la plus dure du développement d'une application. Bien que le temps de déploiement du pipeline soie relativement long, avoir un pipeline ci/cd permet d'avoir des déploiements fiables en réduisants les erreurs.

Si je devais refaire ce projet, j'explorerai sans doute une autre solution, par exemple avec des conteneurs, ou migrer l'application entièrement en fonction Serverless : les possibilité que les technologies Clouds offrent ne manquent pas.

5 ANNEXES

GitHub - ASETML/P_Cloud346: Deploy a web app to azure

6 WEBOGRAPHIE

- MSAL in 10min : https://www.youtube.com/watch?v=wbSrmeCvTC4