DevSoft

Application Arcadia

Documentation Technique

Nicolas Frotey
10-29-2024

Contents

Avant-Propos

Le projet de développement de l'application Arcadia représente une initiative d'une ampleur modérée, mais avec des enjeux significatifs. Conçue pour répondre aux besoins spécifiques d'un parc, elle doit pouvoir gérer un volume potentiellement élevé de connexions simultanées, provenant à la fois des visiteurs et des employés du site. La robustesse et la sécurité de cette application constituent des exigences primordiales, car les données auxquelles les employés peuvent accéder sont particulièrement sensibles. Par exemple, des informations relatives à la santé des animaux pourraient, si elles tombaient entre de mauvaises mains, engendrer des conséquences graves pour le bien-être de ces derniers ou pour le fonctionnement global du parc. Ce projet, bien que d'une complexité modérée, demande une attention particulière à ces aspects pour garantir une expérience utilisateur fiable et protéger les ressources sensibles.

Réflexion Techniques Initiales

Choix d'architecture

Plusieurs types d'architecture possibles sont envisageables à ce stade pour concevoir l'application ; monolithique, en couches, microservices… Il est important d'aborder ce point avec deux critères essentiels ; le temps de développement, et le besoin du client. Dans le cas présent, une architecture en microservices semble largement exagérée au vu des besoins, et induirait une complexité non négligeable dans le projet, amenant nécessairement un temps de développement élevé. Dans la mesure où Arcadia est un premier projet, un premier contact avec l'entreprise avec la présence numérique en ligne, il peut être bon d'intégrer la notion d'évolutions du projet ; le client souhaitera peut-être, par exemple, disposer d'une application mobile dédiée ! Ces considérations m'ont donc amené à favoriser une architecture en couches, au détriment de l'architecture monolithique. L'architecture en couches permet se séparer les responsabilités (accès aux données, logique métier, présentation…), tout en apportant moins de complexité que les microservices. Cette approche permet d'envisager une scalabilité et une évolutivité correcte.

Choix des technologies

Le web dispose de très nombreuses technologies, backend, frontend, parfois même fullstack. Le critère de choix le plus évident se base sur les connaissances de l'équipe de développement ; en effet, dans une optique de rentabilité et d'optimisation du temps de développement, la possibilité de se passer d'une phase d'apprentissage d'un langage ou framework est un enjeu majeur.

D'autre part, chaque technologie ayant ses spécificités, ses use cases, ses avantages et ses inconvénients, il est bien entendu nécessaire de s'assurer que celle-ci est adaptée au projet.

Enfin, considération trop souvent négligée, les modalités de déploiement influenceront également le choix de la technologie.

Etat des lieux des connaissances de développement

Pour ce qui concerne le développement back end, le développeur en charge du projet maîtrise PHP et C#, mais aucun framework (Symfony, Laravel ou ASP .NET). Les technologies de base du frontend (HTML, CSS, JavaScript) sont également maitrisées, mais le développeur en charge du projet n'a aucune expérience dans aucun framework JavaScript (React, Vue, Angular, Svelte etc…). Le framework CSS Bootstrap est cependant plutôt bien maîtrisé. Il maîtrise SQL, mais n'a pas de connaissances dans les BDD NoSQL.

Le choix se pose donc sur :

- L'apprentissage d'un framework PHP ou d'ASP .NET
- Le choix d'un framework JavaScript
- Le choix d'un SGBD SQL et NoSQL

Le Backend

Si les PHP et C# sont tous deux des langages très polyvalents et developer-friendly, en termes de performances C# semble se détacher, en particulier avec le framework ASP .NET. Si ASP .NET présente une courbe d'apprentissage importante pour un débutant, cette dernière se trouve cependant largement adoucie par la connaissance préalable du langage C# et de .NET.

La configuration actuelle de l'équipe penche vers le choix d'une WebAPI ASP .NET 8. A noter que .NET 8 est une LTS et apporte ainsi une stabilité accrue et pérenne. Le déploiement est de plus relativement aisé sur les services d'Azure. D'autre part, l'ORM Entity Framework Core d'ASP .NET Core apporte de nombreuses features relatives à la sécurité et à la l'évolutivité du projet, comme les Migrations, qui permettent une mise à jour incrémentielle du schéma de la base de données, la certitude de de la synchronisation entre la BDD et le modèle de données de l'application, et la préservation des données existantes dans la base de données.

Choix : Web API ASP .NET 8 avec Entity Framework Core

Les SGBD

Au vu du choix de technologie backend, le SGBD SQL qui semble s'imposer est SQL Server, pour son intégration simple avec ASP .NET et sa facilité de déploiement sur Azure.

Le choix du SGBD non-relationnel se portera sur MongoDB pour sa large communauté et sa simplicité de mise en place et de déploiement avec les services d'Atlas.

Choix : SQL Server et MongoDB

Le Client (frontend)

La variété des frameworks frontend rend la décision difficile, et nous nous contenterons d'évoquer les 3 frameworks JavaScript majeurs, React, Vue et Angular, et les deux frameworks les plus répandus, Boostrap et Tailwind.

Le développeur en charge du projet dispose des acquis de base des technologies frontend, affectionne particulièrement les langages fortement typés et les frameworks très structurés. Dans la mesure où il sera nécessaire d'attribuer un temps d'apprentissage pour un framework frontend, Angular semble tout indiqué pour ce projet ; favorisant Typescript, Angular est un framework dogmatique (opinionated) développé par Google réputé pour sa robustesse, sa scalabilité, et ses conventions structurelles strictes. Particulièrement stable, Angular propose la quasi-totalité des composants et librairies nécessaires au développent d'applications de grande envergure, évitant les potentielles difficultés de compatibilité lors des montées de version par exemple.

S'il est évident que le projet ne saurait se contenter d'un style prédéfini, le temps gagné par l'utilisation d'un framework CSS l'est tout autant. Tailwind apporte une personnalisation supérieure à Bootstrap, qui lui privilégie rapidité et simplicité. Comme mentionné plus haut, le framework CSS que nous devons choisir ne constituera pas l'essentiel du style de l'application, mais contribuera à la rapidité du développement, et en cela Bootstrap semble tout indiqué.

Choix : Angular 18 et Bootstrap 5

La Sécurité

Les Enjeux :

La sécurité est un élément essentiel de toute application web aujourd'hui, et les menaces sont multiples ; utilisateur mal intentionné, pirate, utilisateur négligent… Plusieurs leviers sont à notre disposition pour tenter d'y répondre au mieux.

Authentification et Autorisation

Le rempart le plus évident dans le cas d'une application qui a besoin d'une partie avec accès restreint est l'authentification. Dans notre cas, l'utilisateur entre son adresse mail (identifiant) et son mot de passe dans le formulaire de connexion du client prévu à cet effet, client qui envoie via le protocole HTTPS (nous y reviendrons) les informations à l'API. Notons que le mot de passe doit comporter un minimum de 12 caractères, des chiffres, des minuscules, des majuscules, et des caractères spéciaux. L'API renvoie alors un JWT (JSON WebToken) qui, si la connexion est réussie, renvoie l'email de l'utilisateur et ses rôles. Le JWT est sécurisé par une signature obtenue par hashage et qui le rend irrecevable si la moindre modification y a été apportée. Le JWT dispose également d'une durée de vie, qui permet de l'invalider automatiquement après une durée définie (60 min dans le cas de notre application).

Ainsi, le client décode le JWT et stocke les informations de celui-ci dans un cookie, cookie qui s'efface à la fermeture de l'onglet ou à la déconnexion de l'utilisateur.

Une fois authentifié, l'utilisateur se voir attribuer les rôles qui lui ont été donnés par l'administrateur à la création de son compte. Ces rôles restreignent l'accès aux ressources à la fois du client et de l'API. Le front, écrit avec Angular, dispose d'une technologie appelée Guard qui va lui permettre de vérifier à chaque routage vers une autre partie de l'application si l'utilisateur actuellement connecté dispose du rôle requis pour cette action.

Ainsi, pour récapituler, l'application dispose d'un premier niveau de protection avec l'authentification par JWT, et d'un second avec les autorisations basées sur les rôles.

Sécurisation des Inputs

Les inputs utilisateurs doivent faire l'objet d'une attention toute particulière. Un utilisateur mal intentionné aurait tôt fait d'utiliser des failles dans la vérification des inputs pour, par exemple, injecter du code JavaScript ou SQL malveillant.

Le premier pas est la restriction du type des champs en HTML ; si j'attends un nombre entre 0 et 5, mon champ devrait être de type number avec une limite basse à 0 et une limite haute à 5.

ASP .NET effectue également dans l'API et de manière transparente des vérifications visant à « assainir » l'input et à « désarmer » d'éventuels caractères qui pourraient représenter une menace. C'est ce que fait PHP par exemple avec tous les filtres FILTER_SANITIZE (email, quotes, numbers, special_chars…). L'utilisation de LINQ et d'Entity Framework pour les requêtes permet également de les protéger par défaut des injections SQL. ASP.NET Web API inclut également une validation des paramètres d'URL et des requêtes, qui filtre automatiquement les requêtes invalides ou mal formées ; les types forts dans les contrôleurs permettent de même de filtrer les entrées incorrectes avant que le traitement de la logique métier n'ait lieu.

Enfin ASP .NET permet l'utilisation d'annotations qui renforcent la sécurité sur certains champs qui peuvent être requis ou restreints en nombre de caractères par exemple.

Sécurisation de la communication API / Client

Il est également important de soigner la sécurité de la communication entre le back et le front, tout particulièrement dans le cas d'une API séparée du client.

Le protocole HTTPS utilise TLS/SSL pour chiffrer la connexion entre le client et le serveur, rendant ainsi bien plus ardue la lecture des données qui transitent par celle-ci.

Le déploiement sur deux serveurs distincts du front et du back impose également de prêter attention aux politiques CORS (Cross Origin Ressource Sharing) ; ces dernières interdisent par défaut tout site qui ne serait pas déployé sur le même serveur d'accéder aux ressources de l'API. Leur paramétrage permet de limiter l'accès à certains sites identifiés, et limiter ainsi grandement les interactions tierces, et nécessairement les risques.

Prévention client

En dehors de l'application elle-même, la prévention auprès de l'humain est également essentielle. Rappeler de ne jamais partager ses mots de passe, de changer son mot de passe après sa première connexion à l'application, suggérer l'utilisation de coffres forts comme Bitwarden sont autant de gestes qui seront à la fois bien perçus par le client et lui apporteront des réflexes salutaires en matière de cybersécurité. D'autre part, un générateur de mots de passe sécurisés propose un mot de passe qui suit les recommandations de cybersécurité à chaque création de compte, changement ou réinitialisation de mot de passe.

Bonnes pratiques du développeur

J'ai pu m'apercevoir au cours du développement de l'importance de certaines bonnes pratiques de développement en matière de cybersécurité. En effet, à l'ativation de la licence GitGuardian offerte dans le GitHub Student Pack, j'ai pu constater que certaines données confidentielles avaient été push sur GitHub (mots de passe, clé secrète JWT, ConnectionString…). J'ai donc immédiatement modifié tous ces éléments et retiré les fichiers incriminés du tracking du repository. Un rappel efficace de la nécessité d'utiliser correctement le gitignore et de ne JAMAIS push le moindre secret.

Un allié de taille, l'OWASP

L'Open Application Security Project (OWASP) est une organisation internationale à but non lucrative qui se consacre à la sécurité des application web. Elle met à disposition des outils comme ZAP qui permettent de tester la sécurité de son application en simulant une attaque. Notre application a d'ailleurs passé avec succès ce test.

D'autre part, l'OWASP publie régulièrement le top 10 des menaces de cybersécurité qui pèsent sur les applications web. Parmi elles, l'injection SQL (gérée avec ASP, LINQ et Entity Framework), XSS ou Cross-Site Scripting (gérée avec SameSite = Strict sur le cookie d'Authorization), ou le Broken Access Control, faille numéro 1 d'après le top 10 2021 (gérée également avec une attribution méticuleuse des rôles dans l'API).

Ressources

- Documentation Microsoft pour ASP.NET et SQL Server (Microsoft Learn : Développer des compétences qui ouvrent de nouvelles opportunités de carrière)
- Documentation Angular pour le client (Home • Angular)
- Plateforme de cours Dyma (Dyma)
- Plateforme de cours Udemy (Cours en ligne : apprenez ce que vous voulez, à votre rythme | Udemy)
- Plateforme de cours Studi (STUDI - Connexion)
- Plateforme de cours OpenClassrooms (https://openclassrooms.com/)
- Plateforme de cours Grafikart (Tutoriels et Formations vidéos sur le développement web | Grafikart)
- Cours Angular 18 de Simon Dieny (Tableau de bord - Angular SENIOR)
- Stackoverflow (Stack Overflow - Where Developers Learn, Share, & Build Careers)
- ChatGPT (ChatGPT), Claude AI (https://claude.ai/), Gemini (https://gemini.google.com/)

Environnement de travail

BackEnd

- Microsoft Visual Studio 2022

Visual Studio est l'IDE de Microsoft dédié aux langages de Microsoft. Il gère nativement le C#, le F#, mais aussi C/C++, Python, Javascript et Typescript. Il embarque une IA puissante capable de suggérer des propositions pertinentes d'auto-complétion au fil de la saisie. Gratuit dans sa version Community, cet IDE est extrêmement complet.

- Swagger

L'outil Swagger permet de documenter facilement les API, mais également de les tester, en envoyant des requêtes avec les paramètres attendus. Il est également particulièrement pratique pour débugger les types de paramètres attendus dans les requêtes.

Bases de Données

- SQL Server Management Studio 20

SQL Server Management Studio (SSMS) et le logiciel de Microsoft pour gérer les bases de données SQL Server. C'est un équivalent à WorbBench pour MySQL.

- Extension MongoDB pour VS Code

Voir la présentation de l'IDE VS Code dans la partie Client.

Client

- JetBrains Webstorm

JetBrains WebStorm, récemment devenu gratuit pour tous les projets non commerciaux, est un IDE de l'entreprise JetBains, créateurs de l'IDE IntelliJ. C'est un IDE particulièrement polyvalent et puissant qui permet de créer simplement des applications web.

- Visual Studio Code

L'Editeur de Texte Visual Studio Code, édité par Microsoft, est réputé pour sa flexibilité et sa polyvalence, reposant sur un magasin pléthorique d'extensions et une communauté nombreuse et active. Adapté pour tout type de projet, il utilise massivement la CLI pour la plupart de ses features.

Graphismes

Toute la partie graphique de ce repo (Charte Graphique, Mockups, Wireframes) a été réalisée avec Adobe Photoshop CS6. Le MCD, le diagramme de séquence et le diagramme de cas d'utilisation ont été créés avec Drawio.

Modèle Conceptuel de Données

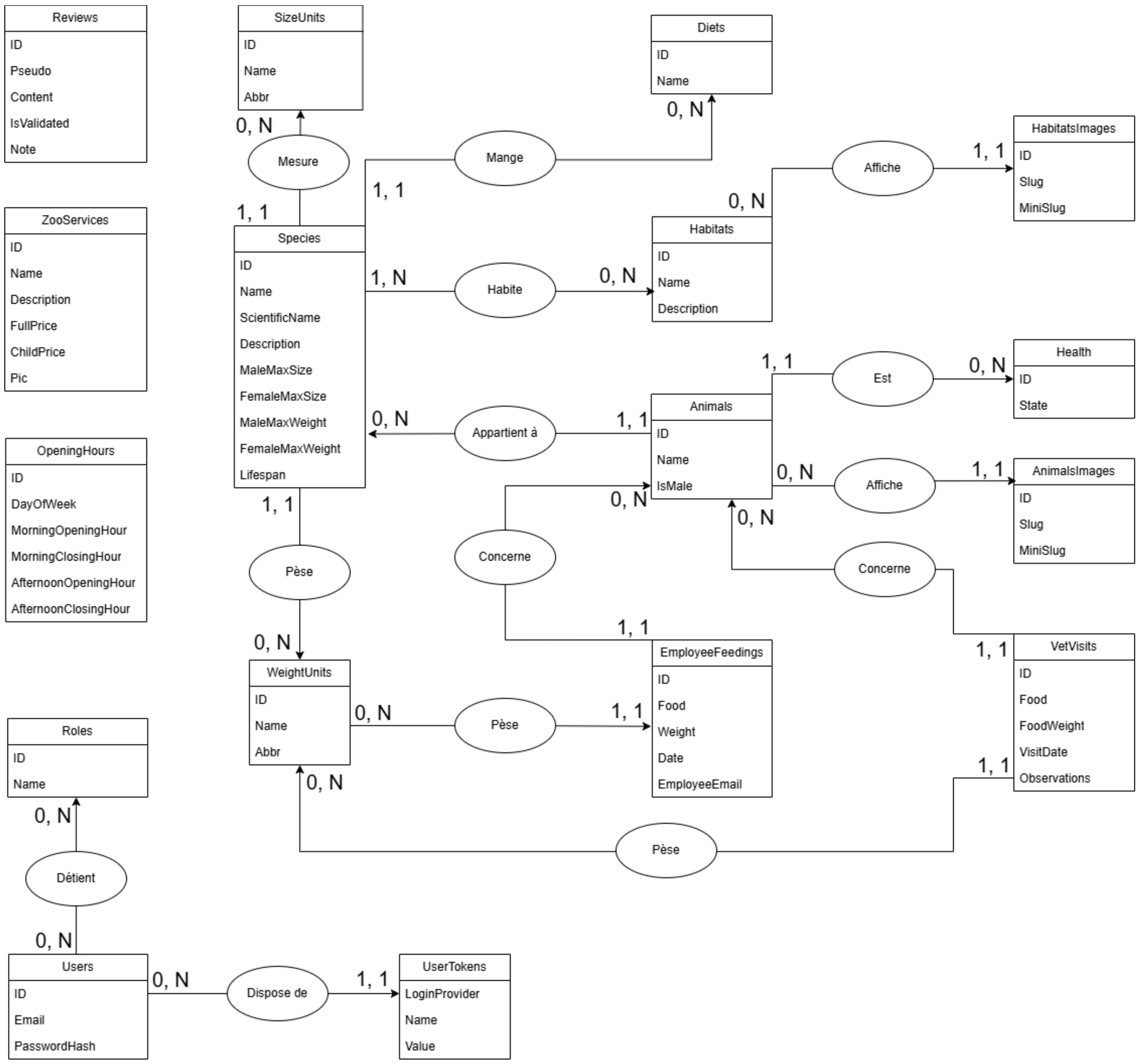

Figure 1. MCD

Diagrammes

Diagramme de Cas d'Utilisation

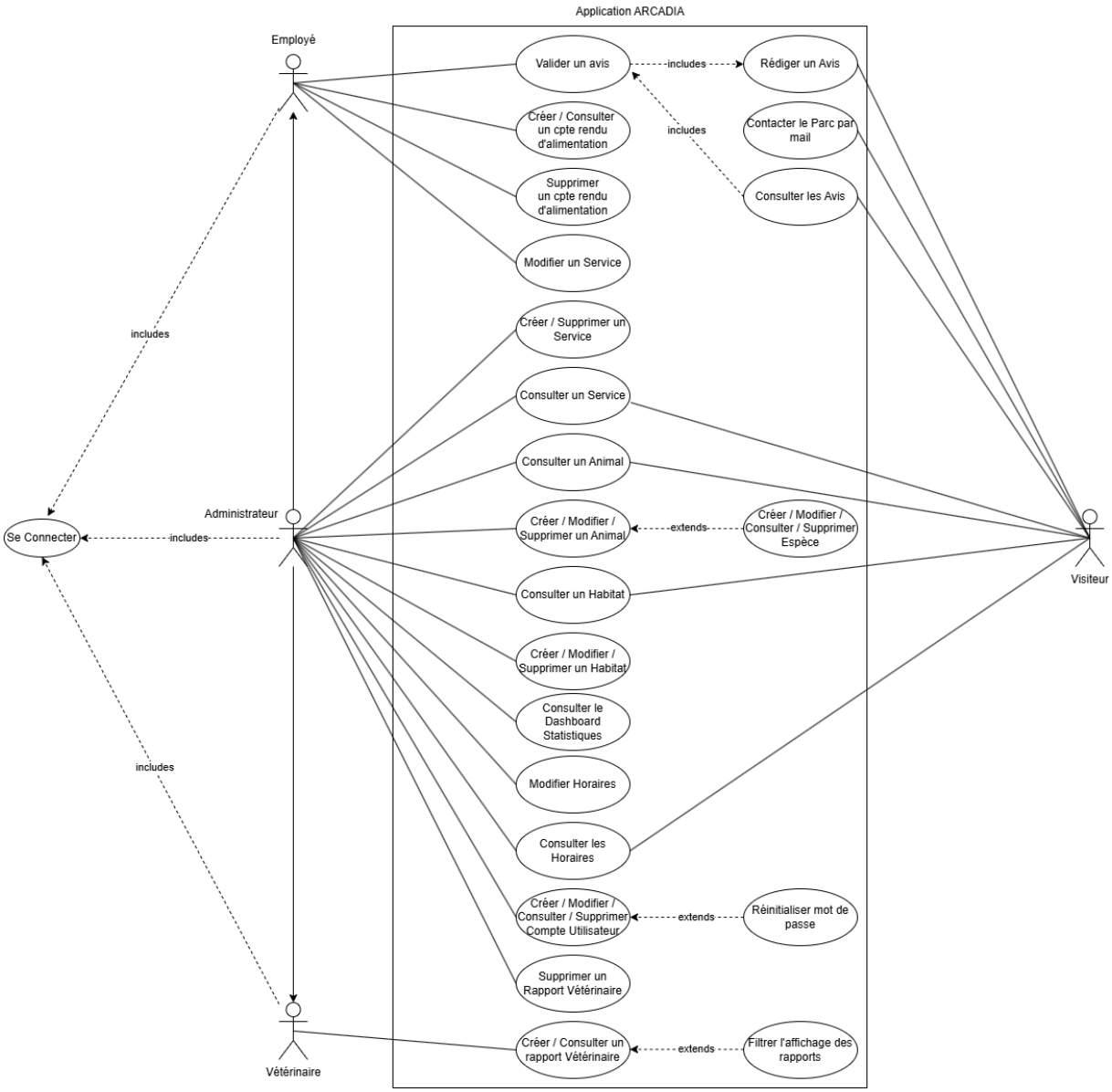

Figure 2: Diagramme de cas d'utilisation

Diagramme de Séquence

Connexion

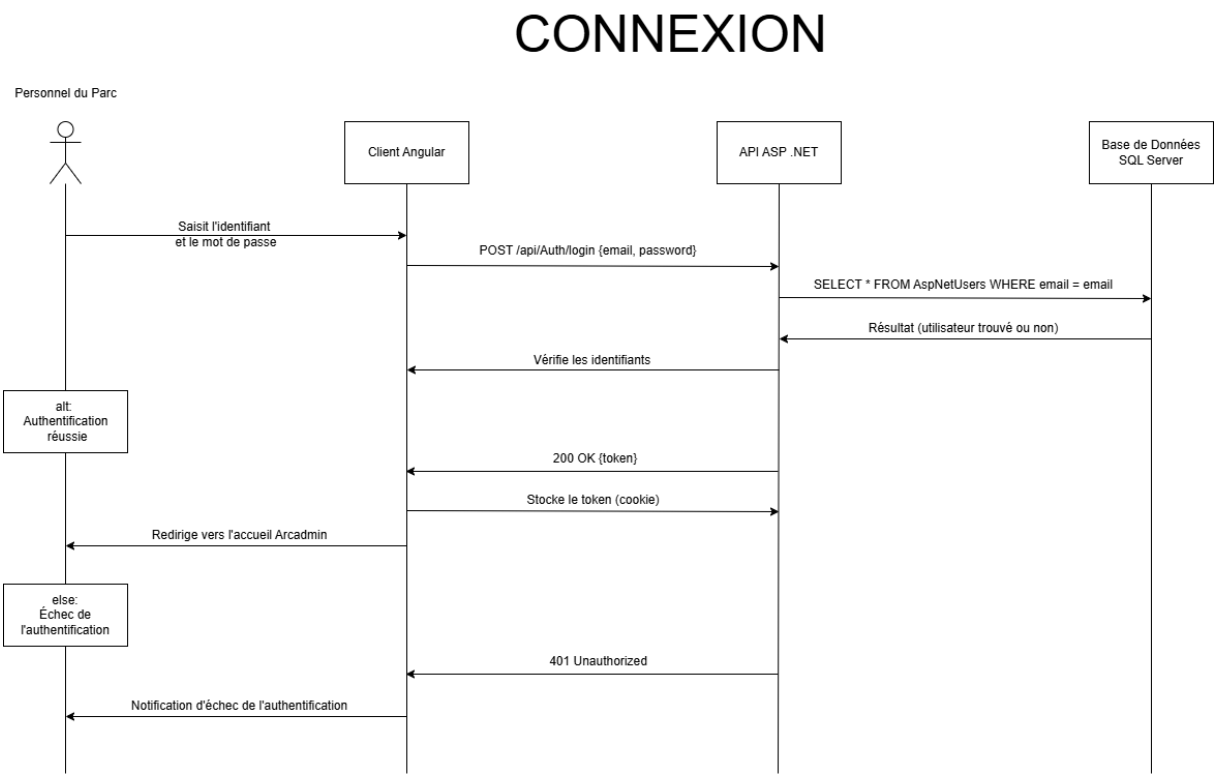

Figure 3: Diagramme de séquence de la connexion à l'application

Création d’un compte utilisateur

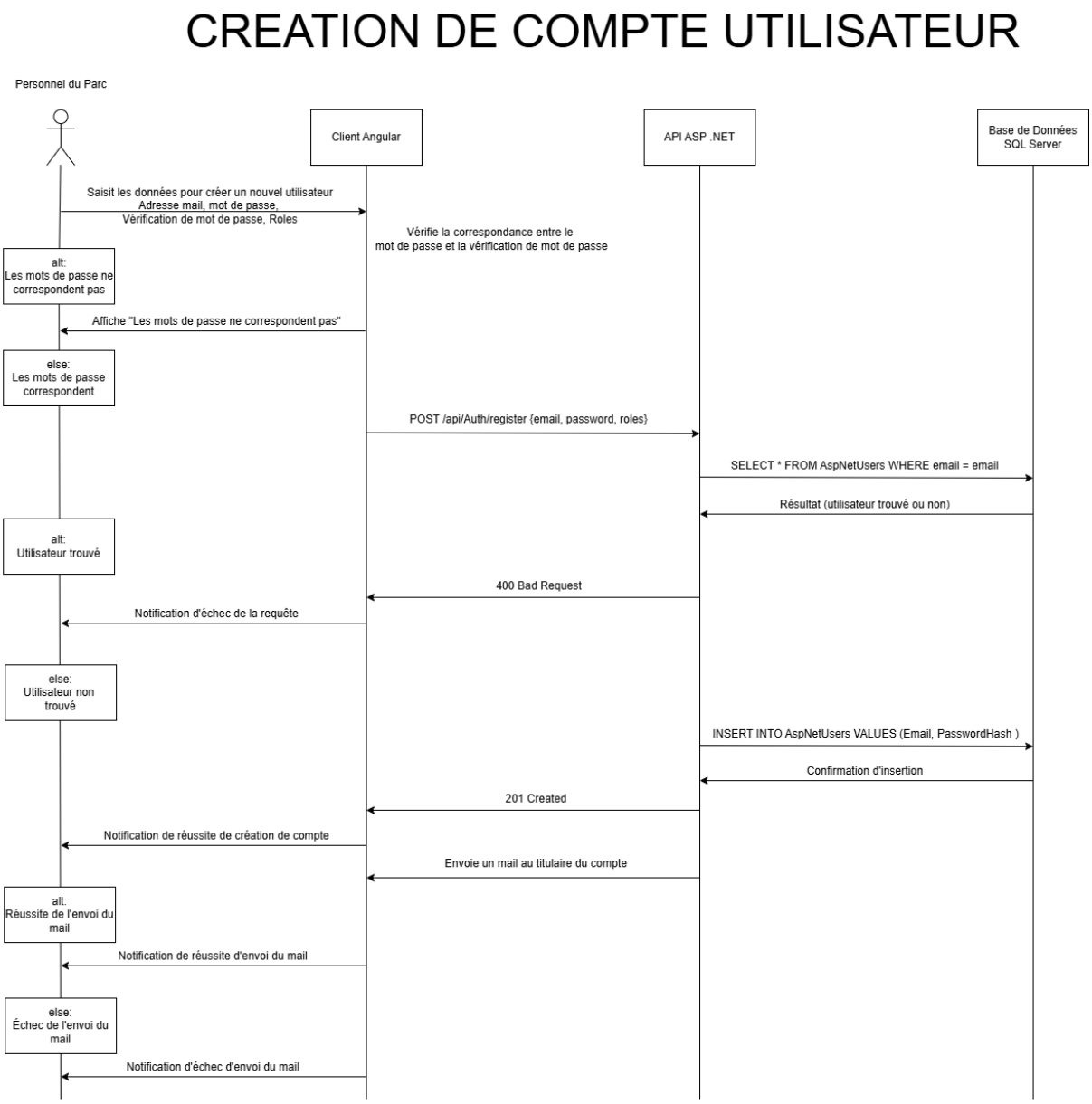

Figure 4: Diagramme de séquence de création d'un nouvel utilisateur

Déploiement

L'application se décomposant en 3 parties distinctes (base de données, API, client), le déploiement se fera également en plusieurs étapes.

API et Base De Données : Azure & Atlas (MongoDB)

Pour réduire au maximum les coûts, j'ai utilisé une solution qui n'est pas viable pour une application réelle, à savoir le Free Tier d'Azure. Ce dernier alloue un temps de disponibilité mensuel de calcul, et pour économiser au maximum ce temps monte l'API à la demande, et la démonte automatiquement après un certain temps. Si cela nous évite d'atteindre trop vite la limite mensuelle, cela implique aussi un temps de démarrage très long, car l'API est démarrée à chaque fois (situation qui n'est pas supposée arriver dans un vrai projet client).

Après avoir créé un compte, place au déploiement. Il faut créer dans notre portail Azure les ressources correspondantes, à commencer par le groupe de ressources que nous appellerons Arcadia.

Créer les ressources

- L'API

Afin de contourner le recours à un service payant, nous créerons une ressource « Application web » (et non Application web + base de données). Il est important de bien la rattacher au groupe de ressources initialement crée.

- La BDD SQL

Pour éviter de dépenser, nous utiliserons l'offre gratuite « Free Serverless Database » de la ressource intitulée « Créer une base de données SQL ». Là aussi, bien penser à rattacher la ressource au groupe de ressources initialement créé.

Il est important de bien sélectionner une méthode d'authentification qui inclut l'authentification SQL ; celle-ci sera en effet celle utilisée pour déployer depuis Visual Studio.

Déployer depuis Visual Studio

Nous devons commencer par créer un profil de publication Azure sur Visual Studio. Pour ce faire, clic droit sur le projet dans l'explorateur de fichiers à droite, puis clic sur «Publier ». On sélectionne alors Azure, puis Azure App Service (Windows) , puis Visual Studio retrouve automatiquement le service App Service qu'on a créé depuis le portail Azure. On le sélectionne avant de cliquer sur Suivant. On choisit alors de publier l'API directement depuis Visual Studio. Après un clic sur Terminer, Visual Studio crée un fichier de profil de publication avec l'extension pubxml dans le dossier Properties/PublishProfiles.

De retour sur l'écran de publication, On clique alors sur les … en face de *Base de données serveur SQL* dans Dépendance de Service. Un nouvel assistant de configuration est lancé, et on sélectionne alors Azure SQL Database. L'assistant retrouve automatiquement la base de données qu'on a créée depuis le portail Azure. Il suffit de la sélectionner et de cliquer sur suivant. À cette étape, l'assistant récupère automatiquement le nom de la chaîne de connexion dans le projet. Il faut renseigner le nom d'utilisateur et le mot de passe configurés lors de la création du serveur SQL sur Azure. On termine la configuration, et il suffit alors de cliquer sur *Publier* pour lancer le déploiement de l'API.

Déployer la base de données MongoDB sur Atlas

Atlas permet de créer sa base de données MongoDB directement en ligne, ce que nous ferons compte tenu de l'extrême légèreté de son contenu. Petite étape à ne pas négliger cependant, il est IMPÉRATIF de penser à autoriser l'URL de l'API à utiliser les ressources de la base de donnés en renseignant là ou les IP utilisées par l'API dans la catégorie Network Access.

Le Client Angular : Firebase

La solution gratuite de Google pour déployer des applications sur Firebase semble tout à fait adaptée pour ce cas d'usage. Les étapes sont simples :

- Créer un projet dans Firebase
- Installer Firebase CLI avec la commande « npm install -g firebase-tools »
- Se connecter à son compte Firebase avec la commande « firebase login »
- Configurer Firebase en local avec la commande « firebase init ». Il suffit de répondre aux questions au fur et à mesure de leur apparition. Dans le cas d'une SPA (ce qui est notre cas !) il est essentiel de bien veiller à répondre « Y » à la question « Configure as a single page app (rewrite all urls to /index.html) ? » ; cela permet de prendre en charge la redirection systématique vers l'index, et donc de ne pas aller chercher via l'URL des ressources uniquement accessibles via le module de routage.
- Build l'application avec la commande « ng build » lancée à la racine du projet.
- Déployer avec la commande « firebase deploy ».