|  |
| --- |
| DevSoft |
| Application Arcadia |
| Documentation Technique |

|  |
| --- |
| Nicolas Frotey  10-29-2024 |

Contents

[Avant-Propos 2](#_Toc181951472)

[Réflexion Techniques Initiales 3](#_Toc181951473)

[Choix d’architecture 3](#_Toc181951474)

[Choix des technologies 3](#_Toc181951475)

[Etat des lieux des connaissances de développement 3](#_Toc181951476)

[Le Backend 3](#_Toc181951477)

[Les SGBD 4](#_Toc181951478)

[Le Client (frontend) 4](#_Toc181951479)

[La Sécurité 4](#_Toc181951480)

[Authentification et autorisation 4](#_Toc181951481)

[CORS 5](#_Toc181951482)

[HTTPS 5](#_Toc181951483)

[Stockage des données sensibles 5](#_Toc181951484)

[Ressources 5](#_Toc181951485)

[Environnement de travail 6](#_Toc181951486)

[BackEnd 6](#_Toc181951487)

[Bases de Données 6](#_Toc181951488)

[Client 6](#_Toc181951489)

[Graphismes 6](#_Toc181951490)

[Modèle Conceptuel de Données 7](#_Toc181951491)

[Diagrammes 8](#_Toc181951492)

[Déploiement 9](#_Toc181951493)

[API et Base De Données : Azure & Atlas (MongoDB) 9](#_Toc181951494)

[Créer les ressources 9](#_Toc181951495)

[Déployer depuis Visual Studio 9](#_Toc181951496)

[Déployer la base de données MongoDB sur Atlas 9](#_Toc181951497)

[Le Client Angular : Firebase 9](#_Toc181951498)

# Avant-Propos

Le projet de l’application Arcadia est un projet d’amplitude moyenne, qui implique potentiellement de nombreuses connexions simultanées de visiteurs et potentiellement d’employés du parc. Elle doit être robuste et sécurisée car les données accessibles par les employés sont de nature à compromettre, par exemple, la santé des animaux, entre de mauvaises mains.

# 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

La sécurité est un élément essentiel dans le développement d’une application.

### Authentification et autorisation

Le mécanisme d’authentification choisi est le JWT, ou JSON Web Token. Délivré par l’API avec une durée de validité déterminée, il permet au client authentifié d’accéder aux ressources si son niveau d’autorisation le lui permet. L’autorisation sera basée sur le principe simple de rôle, transmis dans le JWT. L’outil ASP .NET Core Identity permet de gérer simplement les identités et les rôles des utilisateurs, et sera utilisé dans le développement du projet.

### CORS

Dans une architecture avec une API et un client séparés et déployés sur des serveurs et des hébergeurs différents, il est essentiel de ne pas oublier le paramétrage CORS (Corss-Origin Ressource Sharing). Le bon paramétrage des requêtes CORS permet de n’accorder l’accès aux ressources de l’API qu’à des clients identifiés dans une liste d’URL autorisées, et ainsi d’éviter des interactions avec des clients potentiellement malveillants. Nous reparlerons de la mise en place de cet outil dans la partie déploiement.

### HTTPS

Le protocole HTTPS sécurise les échanges de données à l’aide de la technologie SSL/TLS.

### Stockage des données sensibles

Une application web a besoin de plusieurs identifiants, mots de passe, clés, qui doivent rester secrets sous peine de compromission plus ou moins sérieuse de la sécurité de l’application. Par exemple, la clé utilisée pour générer les JWT permettrait, entre de mauvaises mains, de générer à loisir des JWT valides avec le rôle Admin, et avec une validité infinie. Il est donc impératif de ne pas stocker ces secrets en dur dans le code.

En ASP .NET, comme dans de nombreux autres langages, il est possible de stocker ces secrets dans un fichier de configuration, ici appsettings.json. Lors du développement, j’ai par exemple commis l’erreur de laisser ce fichier dans le tracking du repo, et donc de le commit et de le push régulièrement. À la mise en production du projet, j’ai donc modifié toutes les clefs avant de retirer du tracking le fichier en question, m’assurant ainsi de la confidentialité la plus totale de ces secrets.

## Ressources

* Documentation Microsoft pour ASP.NET et SQL Server ([Microsoft Learn : Développer des compétences qui ouvrent de nouvelles opportunités de carrière](https://learn.microsoft.com/fr-fr/))
* Documentation Angular pour le client ([Home • Angular](https://angular.dev/))
* Plateforme de cours Dyma ([Dyma](https://dyma.fr/developer/accueil))
* Plateforme de cours Udemy ([Cours en ligne  : apprenez ce que vous voulez, à votre rythme | Udemy](https://www.udemy.com/))
* Plateforme de cours Studi ([STUDI - Connexion](https://app.studi.fr/v3/login))
* Plateforme de cours OpenClassrooms (https://openclassrooms.com/)
* Plateforme de cours Grafikart ([Tutoriels et Formations vidéos sur le développement web | Grafikart](https://grafikart.fr/))
* Cours Angular 18 de Simon Dieny ([Tableau de bord - Angular SENIOR](https://angularsenior.fr/b-account))
* Stackoverflow ([Stack Overflow - Where Developers Learn, Share, & Build Careers](https://stackoverflow.com/))
* ChatGPT ([ChatGPT](https://chatgpt.com/)), 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

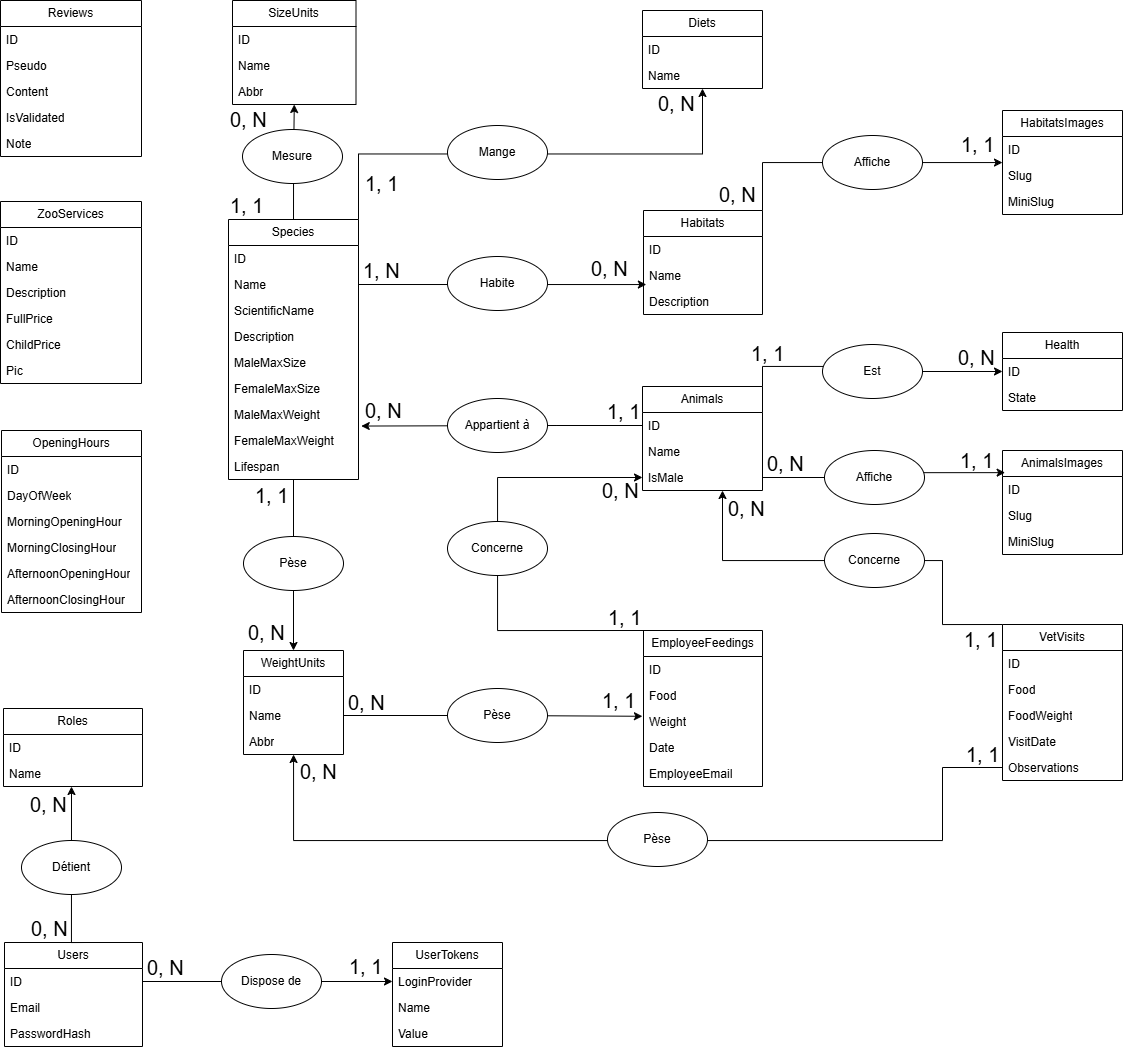


Figure . MCD

# Diagrammes

TODO

* Séquence
* Utilisation

# 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

Depuis VS blablabla

TODO

### Déployer la base de données MongoDB sur Atlas

TODO

## 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 ».