Merimi Ayat & Mehiaoui Mohamed

BUT Informatique année 3

SAE5: Développement avancé

Proposition technique

UNIVERSITÉ DE LORRAINE | IUT Saint-Dié

Année 2025/2026

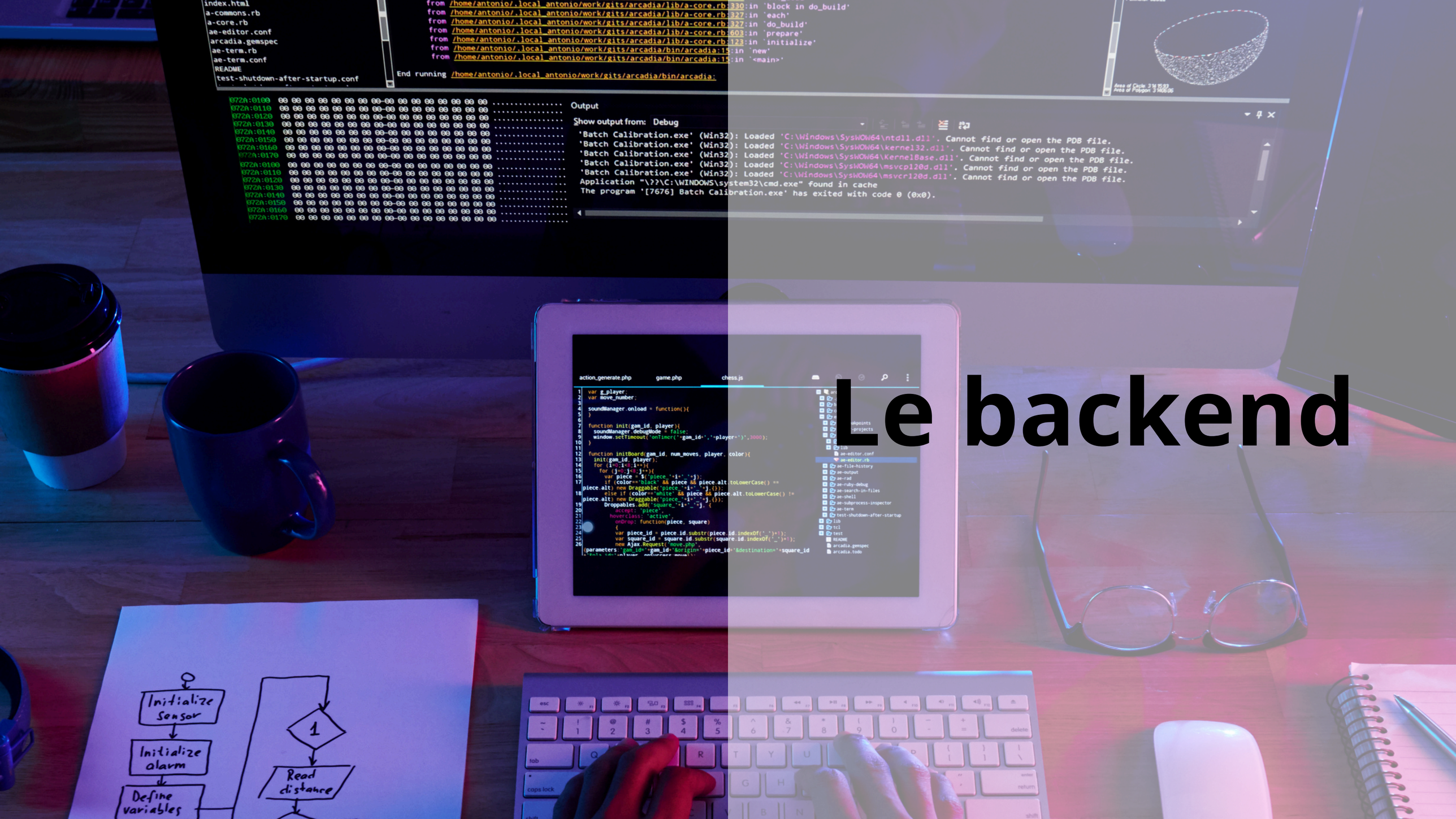
L’équipe de développement

Développeuse Frontend : Merimi Ayat
Contribution : Création des interfaces utilisateur, intégration des cartes interactives, gestion des vues et des formulaires.

Développeur Backend : Mehiaoui Mohamed
Contribution : Conception de l’API, gestion de la base de données, sécurisation des routes, déploiement via Docker.

Ensemble, nous couvrons l’intégralité du cycle de développement : du backend sécurisé à l’interface utilisateur interactive, pour offrir une application SaaS fonctionnelle et intuitive pour Jardin de Cocagne.

Le backend


index.html
a-commons.rb
a-core.rb
ae-editor.conf
arcadia.gemspec
ae-term.rb
ae-term.conf
README
test-shutdown-after-startup.conf
Output
Show output from: Debug
'Batch Calibration.exe' (Win32): Loaded 'C:\Windows\SysWOW64\ntdll.dll'. Cannot find or open the PDB file.
'Batch Calibration.exe' (Win32): Loaded 'C:\Windows\SysWOW64\kernel32.dll'. Cannot find or open the PDB file.
'Batch Calibration.exe' (Win32): Loaded 'C:\Windows\SysWOW64\KernelBase.dll'. Cannot find or open the PDB file.
'Batch Calibration.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msvcp120d.dll'. Cannot find or open the PDB file.
'Batch Calibration.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msvcr120d.dll'. Cannot find or open the PDB file.
Application "\??\C:\WINDOWS\system32\cmd.exe" found in cache
The program '[7676] Batch Calibration.exe' has exited with code 0 (0x0).
Initialize Sensor
Initialize alarm
Define variables
Read distance

La base de données

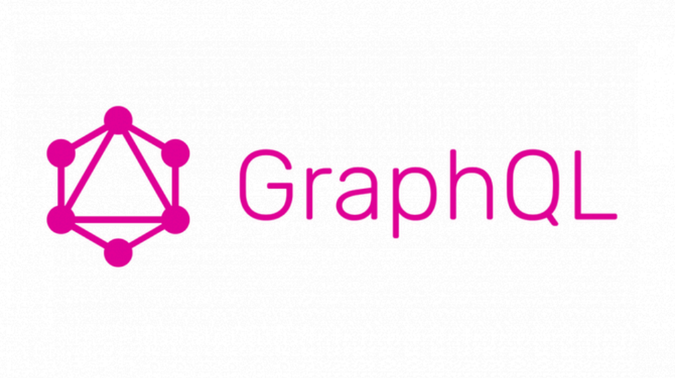
Faite avec : PostgreSQL

Pourquoi ?

- Stable et fiable
- Supporte des types de données variés : JSON, XML, UUID, ARRAY
- Permet d’ajouter des extensions comme PostGIS pour la gestion de données géographiques
- Fonctionne sur tous les principaux OS : Linux, Windows, macOS et compatible avec la plupart des langages de programmation (PHP, Python, Java, Node.js...).

Le serveur backend

Le serveur backend gère toute la logique métier de l’application : il centralise les données, assure leur sécurité, et fournit une interface de communication entre la base de données et le front-end via une API GraphQL.



Node JS

Le cœur logique de l’application. Il gère les données, la sécurité et les échanges entre le front-end et la base de données.

Node.js repose sur le moteur V8 de Chrome et adopte un modèle asynchrone et non-bloquant, idéal pour gérer un grand nombre de requêtes simultanées.
Il sert de base à de nombreux frameworks (comme Express) et permet de construire des serveurs performants.

Pour créer un server clair et simple, avec express.js, node js est parfait : rapide, facilité de prise en main, et possède un écosystème riche (npm, nodemon,path)

Express JS

Permet de créer facilement un serveur HTTP pour gérer les routes, les requêtes et la logique métier.

Express repose sur Node.js et simplifie la création d'un serveur web.
Il permet d'ajouter des middlewares (authentification, logs, CORS, etc.) et de gérer des routes vers les endpoints de ton API.
Il est souvent utilisé comme base pour GraphQL ou REST API.

Comme j'utilise nodejs, alors il fallait pour bien comprendre la création de route api, un framework simple d'utilisation. J'ai donc opté pour express js .

Graphql

Un langage de requête pour interagir avec les données du serveur, plus flexible et précis que REST

GraphQL permet au client de demander exactement les données nécessaires, ni plus ni moins.
Il centralise toutes les ressources dans une seule requête.
Avec nodeJS il s’intègre parfaitement à Express pour exposer des queries, mutations et types définis dans le schéma

J’ai opté pour GraphQL, car c’est un type d’API très utilisé dans le monde professionnel.
De plus, il permet de sélectionner précisément les données à renvoyer, ce qui le rend plus sécurisé et efficace que les API REST.
Grâce à un seul point d’entrée et des schémas définis, il assure un échange clair et structuré des données.

Leaft.js

Leaflet permet de créer des cartes interactives, d’ajouter des marqueurs, des popups, et de gérer des événements utilisateur comme les clics ou les déplacements.
Elle fonctionne parfaitement avec Node.js côté frontend pour afficher des cartes dynamiques et réactives.

🌍Open-source et légère (pas lourde comme Google Maps)
⚡Rapide et performante pour des cartes interactives
Simple à intégrer avec HTML, CSS et JS
📌Supporte les marqueurs, popups et couches personnalisées
🔄Compatible avec tous les navigateurs modernes


//set close by state
//set close by "esc"
const closeOnEscapeKeyDown = (e) => {
if (e.keyCode === 27) {
(arguments);}
use effect to detect click "esc"
useEffect(() => {
document.body.addEventListener
return function cleanup() {
document.body.removeEventListener
}, []);
.App {
text-align

Le Frontend

Un server côté frontend !

Ce serveur gère la partie visible du site : il distribue les pages web, tout en contrôlant l'accès selon l'authentification et le rôle de l'utilisateur.

Contrairement au serveur backend (qui gère les données et l'API), ce serveur Express côté frontend s'occupe de :

- Distribuer les pages HTML au navigateur
- Contrôler l'accès à certaines routes selon la session utilisateur
- Gérer la sécurité (sessions, cookies, HTTPS, redirections)

Express JS

Architecture Front-End

Assurer la conception et le développement de l’interface utilisateur de l’application du réseau Cocagne, en garantissant une expérience fluide, responsive et conforme aux standards de qualité ergonomique et logicielle.

Le Front-End s’appuie sur une architecture modulaire :

- Chaque composant visuel (formulaire, tableau, carte, navigation, etc.) est isolé dans une structure propre (SCSS + HTML).
- Les éléments réutilisables sont factorisés afin de réduire la redondance du code.
- Les fichiers SCSS sont organisés par modules et importés dans une feuille principale

SCSS

Organisation du code SCSS

- **scss/**
  - **base/ → variables, couleurs, typographies**
  - **components/ → styles des boutons, formulaires, cartes, menus**
  - **layout/ → grid, header, footer, responsive**
  - **pages/ → pages spécifiques (dashboard, profil, accueil…)**
  - **main.scss → fichier principal compilé en CSS**

Cette organisation suit une logique modulaire pour permettre aux membres de l'équipe de travailler en parallèle sur différents éléments de l'interface.

- Permet de mieux structurer et organiser les feuilles de style en les découpant par composants ou par pages.
- Facilite la maintenance et l'évolution du design dans le temps.
- Offre des fonctionnalités avancées (variables, mixins, nesting…) pour une meilleure productivité.

Responsivité et accessibilité

- Utilisation du système de grille Bootstrap pour adapter le design à toutes les résolutions.
- Composants testés sur mobile, tablette et desktop.
- Respect des bonnes pratiques d’accessibilité (contraste, tailles, alternatives textuelles).

Déploiement

Docker

Docker encapsule chaque partie de l’application (frontend et backend) dans un conteneur indépendant.
Chaque conteneur a son environnement propre, ses dépendances, et peut tourner de manière identique en développement et en production.

Cela permet de :

- **Déployer le projet sur n’importe quelle machine ou serveur**
- **Éviter les conflits entre versions de Node.js, packages, ou bases de données**
- **Avoir des environnements reproductibles et stables**

💡 **Pourquoi Docker pour ce projet :**

1. ✅ **Technologie connue → Nous maîtrisons déjà Docker, ce qui facilite le déploiement.**
2. ✅ **Isolation des environnements → frontend et backend tournent séparément, sans interférer l’un avec l’autre.**
3. ✅ **Portabilité → le projet fonctionne exactement pareil sur ton PC, sur un serveur cloud, ou sur une VM.**

Hébergement

Render

Frontend, backend et base de données dans des conteneurs isolés, pour un déploiement simple, sécurisé et reproductible.

Render permet de déployer tous les composants Docker de ton projet :

- **Frontend Node.js + Express**
- **Backend Node.js + Express + GraphQL**
- **Base de données PostgreSQL**
- **Chaque composant est isolé dans un conteneur indépendant, communique via des ports configurés, et fonctionne de manière identique en dev et en prod.**