Cahier des charges technique

S5.A.01 Développement avancé

Présentation des choix techniques et de l'architecture pour notre SAÉ.

HERRY Matteo - HERRY Elwyn

UNIVERSITÉ DE LORRAINE | IUT Saint-Dié-des-Vosges

15/10/25

Problématique

1

Description du besoin réel

gestion des livraisons, dépôts, tournées, adhérents

2

Contraintes

Utilisateurs peu qualifiés, interface simple, maintenance légère, open-source

3

Indicateurs de succès

fiabilité, accessibilité, rapidité, évolutivité

Objectifs et Résultats Clés

1

Création d’une application

Créer, **en équipe**, une application en suivant une démarche itérative ou incrémentale.

2

Développement Ergonomique

Utiliser des **technologies avancées** et s'inscrire dans une démarche d'**amélioration continue**.

3

Plateforme Technique Robuste

Déployer une architecture dockerisée moderne, basée sur Node.js et MySQL.

Identification des Besoins et Contraintes

1

Besoin de Cartographie Précise et calcul d'itinéraires efficaces

Nécessité d'intégrer une solution de cartographie open source (Leaflet + OpenStreetMap) pour visualiser les arrêts et les dépôts.

Exigence d'un moteur de routage rapide et fiable (OSRM) pour calculer les chemins optimaux.

2

Utilisation d'une API, en lien avec une BDD et un serveur

L'application devra utiliser des API complexes, interroger une base de données mais également s'appuyer sur un serveur pour la gestion des requêtes et des données

3

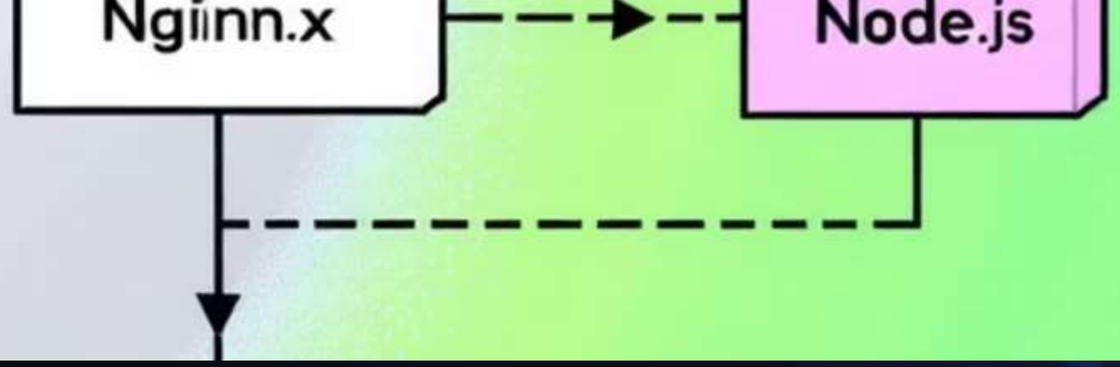
Architecture Modulaire

Le système doit être facilement maintenable et évolutif grâce à une conteneurisation complète (Docker).

4

Sécurité des Données

Toutes les informations (données clients, dépôts, tournées) doivent être stockées dans un environnement MySQL sécurisé.



Architecture Globale : Le Modèle Docker Conteneurisé

L'application est structurée autour de quatre conteneurs Docker indépendants, garantissant isolation et portabilité.

Web Server (Nginx)

Point d'entrée unique, gère les requêtes statiques (Front-end) et agit comme proxy inverse vers l'API.

Application (Node.js/Express)

Le cœur de la logique métier, responsable de l'API REST et des interactions avec la base de données.

Base de Données (MySQL)

Stockage persistant et sécurisé de toutes les données opérationnelles (dépôts, tournées, arrêts).

Administration (phpMyAdmin)

Interface graphique pour la gestion et le suivi simplifiés de la base de données (pour les développeurs et administrateurs).

Flux de Requêtes : De l'Utilisateur à la Base de Données

Chaque requête suit un chemin précis pour assurer la sécurité et le bon traitement des données.

Client **Nginx** **API Express** **MySQL**

1

Client

Envoie une requête (comme charger une tournée) via le navigateur.

2

Nginx

Reçoit la requête et la transmet à l'API Express.

3

Express API

Traite la logique métier et interroge la base de données MySQL.

4

MySQL

Fournit les données demandées (ex: liste des dépôts).

La Stack Logicielle : Technologies Clés

Nous avons sélectionné des technologies éprouvées et performantes, majoritairement Open Source.

- **Front-end :** HTML5, Bootstrap 5 (pour la réactivité), Leaflet.js (Cartographie).
- **Back-end :** Node.js et framework Express (API REST rapide et légère).
- **Base de Données :** MySQL (robustesse et performance pour les données structurées).

Le choix de Node.js permet une unification du langage (JavaScript) sur le client et le serveur, simplifiant le développement.

Modèle de données et init

1

Scripts d’initiation

Exemples de scripts : 01-depots.sql, 02-tournees.sql

2

Jeu de données initial

5 tournées, liste de dépôts

3

Encodage

utf8mb4

API REST : Les Routes CRUD Principales

L'API Express expose des points d'accès clairs pour gérer les ressources de l'application.

Ressource	Méthode	Description
/api/tournees	GET	Récupérer la liste de toutes les tournées (avec filtres optionnels).
/api/tournees	POST	Créer une nouvelle tournée avec ses arrêts initiaux.
/api/tournees/{id}	PUT/PATCH	Modifier les détails d'une tournée existante (date, statut, etc.).
/api/tournees/{id}	DELETE	Supprimer une tournée.
/api/tournees/{id}/stops	POST	Ajouter un arrêt à une tournée spécifique.

Les routes d'API sont versionnées pour garantir la compatibilité future.

Robustesse : Validation des Données et Gestion des Erreurs

La fiabilité du système passe par une vérification rigoureuse des données entrantes et une gestion claire des erreurs pour le client.

Validation Côté Serveur

Toutes les données (ex: coordonnées géographiques, format de date) sont validées par l'API avant l'écriture en base. Utilisation de schémas Joi.

Codes de Statut HTTP

En cas d'échec, l'API retourne des codes HTTP standard (ex: 400 Bad Request, 404 Not Found, 500 Internal Server Error) pour une interprétation facile.

Feedback Utilisateur

Les messages d'erreur sont concis et traduits, permettant à l'utilisateur de corriger rapidement ses actions.

Sécurité et Conformité

CORS configuré

Secrets dans .env

Logs d’accès et erreurs

Mention future : authentification JWT / session

Compte MySQL applicatif à privilèges réduits

Solution d’hébergement

Deux options principales ont été étudiées :

Critère	**Scaleway (France)**	**OVHcloud (France)**
Localisation	Paris / Amsterdam	Roubaix / Gravelines
Type d’offre	Instances cloud (VPS ou “Scaleway Elements”)	VPS Cloud / Public Cloud
Compatibilité	Docker, Docker Compose natif	Compatible Docker via SSH
Sauvegardes	Snapshots et volumes persistants	Backups automatiques
RGPD	Conforme (hébergement UE)	Conforme (hébergement UE)
Avantage principal	Simplicité et prix bas	Fiabilité et support reconnu

Solution d’hébergement

Après comparaison de 2 solutions cloud européennes (Scaleway, OVHcloud), l’équipe a retenu **OVHcloud** pour l’hébergement de l’application web.

Raisons du choix :

- **Fiabilité reconnue** : OVHcloud est un acteur français historique, utilisé dans de nombreux environnements professionnels.
- **Hébergement 100 % en France** : garantit la conformité RGPD et la souveraineté des données.
- **Compatibilité Docker complète** : permet de déployer notre stack
- **Stockage persistant et sauvegardes automatiques** : idéal pour sécuriser la base MySQL.
- **Rapport qualité/prix** : offre VPS abordable (≈ 5 €/mois) parfaitement adaptée à un projet académique et évolutif.

Solution de tests

Garantir la **fiabilité**, la **stabilité** et la **cohérence des données** de l'application à chaque itération de développement.

Type de test	Objectif	Outils / Méthodes
Tests unitaires	Vérifier le bon fonctionnement des routes API (GET, POST, PUT, DELETE).	Jest + Supertest (Node.js)
Tests d'intégration	Contrôler la communication entre Express, MySQL et le front.	Docker Compose en environnement de test
Tests manuels / exploratoires	Vérifier le rendu visuel, les interactions carte / calendrier.	Navigateur + console développeur
Validation continue	Lancer automatiquement les tests à chaque	GitLab CI/CD pipeline

Présentation de l’équipe

Nous sommes une équipe de 2 développeurs juniors (étudiants) full-stack, un binôme polyvalent.

Rôle	Nom
Matteo Herry	Développement full-stack : conception de l’API (Node.js / Express), intégration du front (Bootstrap / Leaflet / FullCalendar), gestion Docker et base MySQL.
Elwyn Herry	Développement full-stack : création des interfaces, gestion des routes API, modélisation des données, intégration des solutions Carte / Itinéraire / Calendrier.

Déploiement et Configuration d'Environnement (.env)

Le déploiement est simplifié grâce à Docker Compose, et la configuration utilise des variables d'environnement.

Déploiement via Docker Compose

Un seul fichier `docker-compose.yml` permet de démarrer l'ensemble des quatre conteneurs, assurant une installation cohérente.

Variables d'Environnement (.env)

Les informations sensibles (mots de passe, clés API, ports) sont stockées dans un fichier `.env` et ne sont jamais codées en dur.

Séparation des Environnements

Configuration séparée pour les environnements de développement, test et production (ex: `NODE_ENV=production`).