Rapport de Stage — BTS SIO SLAM

Option SLAM · 1ère année · Semaines 2 à 4

Hidaoui Mohamed Amine — Planisphère Info

2e Mission — Développement d'une interface de consultation Karlia

Présentation de l'entreprise

Planisphère Info est une entreprise spécialisée dans le développement de solutions logicielles et de connecteurs entre logiciels de gestion. Elle développe principalement des applications web en PHP et des solutions e-commerce sous Prestashop, et utilise un serveur GitLab interne pour la gestion du code source.

Contexte

À la suite de la première mission de migration GitLab, l'entreprise m'a confié le développement, depuis zéro, d'une application web interne destinée à consulter et gérer les données du CRM Karlia : entreprises, contacts et tickets. L'interface officielle de Karlia étant jugée trop lente et peu intuitive pour les besoins quotidiens de l'équipe, l'objectif était de fournir un outil plus rapide et plus clair.
Cette mission a couvert l'ensemble du cycle de développement web : appels d'API REST, développement PHP et MySQL côté serveur, JavaScript côté navigateur, gestion de sessions sécurisées et optimisation des performances face aux contraintes de l'API Karlia.

Contraintes principales

- Volume de données : 1 676 entreprises clientes, leurs contacts associés et plusieurs milliers de tickets.
- Rate limit de l'API : 100 requêtes par minute maximum. Tout dépassement provoque une erreur HTTP 429, comptée dans le quota même en cas d'échec.
- Environnement de développement : WAMP 64 (Apache, PHP, MySQL) en local sous Windows.
- Projet mené en cinq versions successives (V1 à V5), suivies d'une refonte architecturale (V6).

Semaine 2 — Prototype JavaScript (V1 et V2)

Contexte

La semaine 2 a été consacrée à la compréhension du fonctionnement de l'API REST Karlia et à la réalisation d'un premier prototype fonctionnel en JavaScript (Node.js). L'objectif initial était d'afficher la liste des 1 676 entreprises clientes dans un tableau HTML consultable.

Méthode utilisée

Le prototype V1 repose sur deux fichiers Node.js distincts :

- server.js : serveur proxy Express.js écoutant sur localhost:3000. Il contourne la restriction CORS du navigateur en servant d'intermédiaire, et ajoute le token d'authentification Bearer à chaque requête vers l'API.
- karlia-contacts.js : script exécuté en ligne de commande, qui appelle l'API de façon paginée (50 entreprises par requête, soit environ 34 appels pour 1 676 entreprises) et génère un fichier HTML statique.

Les appels à l'API suivent la structure suivante : base URL https://karlia.fr/api/, authentification par header Authorization: Bearer, format REST/JSON, et pagination via les paramètres limit et offset.

Difficultés rencontrées et solutions

Crash des serveurs Karlia (HTTP 429)

Sans délai entre les requêtes, le quota de 100 requêtes par minute était dépassé en quelques secondes lors du chargement des contacts, bloquant temporairement la clé API. Deux solutions ont été appliquées en V2 : un délai de 700 ms entre chaque requête paginée (soit environ 85 req/min, avec une marge de sécurité pour les autres utilisateurs partageant le quota), et un traitement par lots de 10 contacts en parallèle avec une pause de 7 secondes entre chaque lot.

Temps de génération prohibitif

Le chargement séquentiel des contacts (un appel par entreprise) demandait jusqu'à 33 minutes. La solution adoptée est le lazy loading par Intersection Observer : les contacts ne sont chargés qu'au défilement, pour les lignes visibles à l'écran uniquement. La génération initiale est ainsi tombée à environ 30 secondes.

Dépendance au serveur Node.js

L'interface ne s'affichait que si server.js était actif en arrière-plan. À la fermeture du terminal, tous les appels échouent. Ce problème a motivé la migration vers PHP en semaine 3.

Résultat

Le prototype V2 affiche les entreprises avec un premier design (en-têtes colorés, lignes alternées, distinction des numéros fixes et mobiles), respecte le rate limit de l'API, et réduit le temps de génération de 33 minutes à 30 secondes. Sa principale limite reste la dépendance au serveur Node.js.

Semaine 3 — Migration PHP et authentification (V3 et V4)

Contexte

La semaine 3 visait deux objectifs : éliminer la dépendance au serveur Node.js en portant le projet vers PHP sur WAMP, puis ajouter un système d'authentification et de traçabilité pour réserver l'accès à l'équipe de Planisphère.

Méthode utilisée — Migration PHP (V3)

Les deux fichiers JavaScript (server.js et karlia-contacts.js) ont été fusionnés en un unique fichier karlia.php. PHP s'exécutant côté serveur, il n'y a plus de problème de CORS ni de proxy à maintenir, et la clé API n'est plus exposée au navigateur.

Difficultés de la migration

- Erreur SSL sous WAMP : PHP ne disposait pas du bundle de certificats racines pour vérifier les connexions HTTPS. Résolu en téléchargeant cacert.pem et en le déclarant dans php.ini (curl.cainfo et openssl.cafile).
- Encodage JSON : les noms d'entreprises contenant une apostrophe (ex. « L'Atelier ») cassaient la chaîne JavaScript injectée par PHP. Résolu avec les options JSON_HEX_TAG, JSON_HEX_APOS et JSON_HEX_QUOT de json_encode().
- URL des appels AJAX : les requêtes échouaient car le chemin de base était mal résolu. Corrigé en utilisant $_SERVER['PHP_SELF'], qui renvoie le chemin exact du fichier en cours d'exécution.

Réalisation — Fonctionnalités complètes (V4)

La V4 est la version la plus dense en fonctionnalités. Elle introduit un système d'authentification complet, une interface d'administration et un onglet de gestion globale des tickets.

Système d'authentification

- Deux tables MySQL : users (identifiants, mot de passe hashé en bcrypt, rôle, état actif) et logs (utilisateur, action, cible, IP, horodatage).
- Protection contre la fixation de session (régénération de l'identifiant après connexion) et expiration du cookie à la fermeture du navigateur.
- Requêtes préparées PDO contre l'injection SQL, et journalisation des tentatives de connexion échouées avec leur adresse IP.

Interface d'administration

- Gestion des comptes : création d'utilisateurs et activation/désactivation, un compte administrateur ne pouvant pas en désactiver un autre.
- Modification de son propre mot de passe avec vérification de l'ancien.
- Consultation des journaux d'activité, filtrables par utilisateur, action et date.

Onglet Tickets global

Cet onglet offre une vue transversale sur tous les tickets de toutes les entreprises, fonctionnalité difficile d'accès dans l'interface officielle Karlia. Il propose le tri par colonne, des filtres combinables (recherche, statut, priorité), des actions groupées de changement de statut, la création de tickets via un panneau latéral avec autocomplétion sur le nom d'entreprise, et des notifications en temps réel (vérification toutes les 60 secondes).

Résultat

À l'issue de la V4, l'application est autonome (WAMP seul), sécurisée par authentification, et permet la consultation comme la gestion des tickets. La latence lors du chargement des contacts au défilement reste le principal point à optimiser.

Semaine 4 — Optimisation et architecture structurée (V5 et V6)

Contexte

En sortie de V4, la latence demeurait le problème dominant : une session complète de consultation pouvait cumuler jusqu'à 9 700 appels à l'API (chargement des entreprises, puis un appel de contacts par ligne au défilement, plus les vérifications en temps réel). L'objectif de la semaine 4 était de rendre la consultation nettement plus rapide que l'interface officielle, et de restructurer le code pour le rendre maintenable.

Réalisation — Corrections et refonte (V5)

Bugs JavaScript bloquants

- Apostrophe non échappée dans deux chaînes contenant « l'onglet », provoquant une erreur de syntaxe qui bloquait tout le chargement des contacts.
- Autocomplétion d'entreprise : guillemets mal fermés dans un attribut HTML. Résolu en attachant un véritable écouteur d'événement (addEventListener) plutôt qu'un gestionnaire inline, robuste pour tous les noms d'entreprises.

- Bouton « Retour » de la fenêtre de détail d'un ticket : il appelait une fonction supprimée lors d'une refactorisation. Le bouton a été retiré, la croix de fermeture étant conservée.

Affichage des statuts et priorités

L'API renvoie les identifiants numériques des statuts et priorités, mais pas toujours leur libellé. Les colonnes affichaient donc un tiret. La solution consiste à charger au préalable les listes de référence (statuts, priorités, pipelines) et à résoudre le libellé à partir de l'identifiant au moment de l'affichage.

Chargement progressif et refonte du design

La pagination numérotée a été remplacée par un chargement progressif : le premier lot de tickets s'affiche immédiatement, les suivants se chargent en arrière-plan avec un indicateur discret. Le design a été refondu autour d'un système cohérent (police Inter, variables CSS, badges de statut colorés, notifications).

Bilan

Version	Technologie	Apport principal	Limite restante
V1	Node.js (2 fichiers)	Premier affichage fonctionnel	Crash API, lenteur, dépendance
V2	Node.js optimisé	Rate limit, lazy loading	Dépendance à server.js
V3	PHP (1 fichier)	Indépendance vis-à-vis de Node.js	Latence, bugs SSL et JSON
V4	PHP et MySQL	Authentification, admin, tickets	Latence au défilement
V5	PHP et design system	Correction des bugs, design	Architecture monolithique

Compétences développées

- API REST : authentification, pagination, gestion du rate limit, reprise automatique sur erreur 429.
- PHP 8 : appels HTTPS via cURL, PDO/MySQL avec requêtes préparées, bcrypt, sessions sécurisées.
- JS : chargement à la demande, parallélisation des appels, temps réel et manipulation du DOM.
- MySQL : conception d'un schéma relationnel et requêtes avec jointures.
- WAMP et Apache : configuration de php.ini, fichiers .htaccess, débogage via les outils du navigateur.
- Sécurité web : prévention des injections SQL et des failles XSS, protection des fichiers sensibles.

Ce Projet va se conclure par la v6 ou v7 durant la semaine 5 soit la dernière du stage