

SAE501 : Création d'une application multimodale

Groupe F Cerberus:

Mohamed RAHMANI

Yannick KHAMIS

Patrick CHEN

Vithurzen Vilvarajah

Marie-Éva Le Saunier

Jean-Christophe Dubacq

Davide Hébert

Gaël Guibon

Sommaire

1 - Organisation du travail d'équipe.....	2
2 - Ingénierie logicielle.....	2
3 - Optimisation.....	6
4 - Aspects communicationnels.....	8
Optimisation.....	12

1 - Organisation du travail d'équipe

Notre équipe Cerberus a opté pour l'utilisation d'un serveur discord comme plateforme de communication interne, afin de gérer les annonces, les choix de direction, l'organisation des sessions et les réunions. Une fois le cahier des charges défini, nous avons décidé d'utiliser un système de planification comme Trello pour organiser et suivre l'avancement du projet. Grâce à Trello, nous avons pu visualiser clairement l'état d'avancement de chaque partie du projet et d'identifier rapidement les tâches qui prennent le plus de temps à être réalisées. Le responsable d'équipe et manager, Mohamed a également joué un rôle clé en organisant les phases de développement et en guidant les actions de chaque membre. Il fonctionnait comme une sorte de « main invisible », s'assurant que les buts soient atteints tout en permettant à chaque membre d'exécuter ses tâches en toute autonomie.

Concernant la gestion du code et le développement collaboratif, nous avons structuré notre organisation autour de GitHub via une organisation. Tout le groupe avait un accès en écriture et lecture sur toutes les branches des différents projets sauf sur la branche main des projets qui était hébergée sur le GitHub de l'organisation par le responsable d'équipe. Nous avons aussi maintenu une communication régulière grâce à des mises au point quasi hebdomadaires, notamment à travers les séances de SAE que nous avons en commun au moins une fois par semaine. De plus, notre fréquence de travail et nos échanges quotidiens facilitent une coordination fluide et continue entre tous les membres.

2 - Ingénierie logicielle

Nous avons fait le choix de partir sur une application de bureau qui serait cross-platform et pour ça, nous avons choisi le framework **Electron.js** qui est un framework qui permet de développer des applications multiplateformes de bureau avec

des technologies web comme Javascript, HTML, CSS. Nous avons dû choisir entre Electron.js et Tauri qui est aussi un framework pour créer des applications de bureau mais étant un framework utilisant Rust et Javascript, nous avons fait le choix d'Electron.js pour la rapidité de développement des technologies web que l'ensemble du groupe maîtrise même si au final l'application sera un peu plus lourde car Electron.js embarque avec l'application, un navigateur web entier.

Pour ce qui est du frontend de l'application, nous avons choisi la librairie javascript **React** pour 2 raisons, la première est que React propose une architecture basée sur des composants, où chaque partie de l'interface utilisateur(UI) peut être encapsulée dans des composants autonome et réutilisables sur toute la code base et deuxième raison, React utilise un DOM virtuel pour minimiser les manipulations directes du DOM dans le navigateur, ce qui peut être coûteux en terme de performances et ce qui va permettre de mettre à jour efficacement l'interface de l'application.

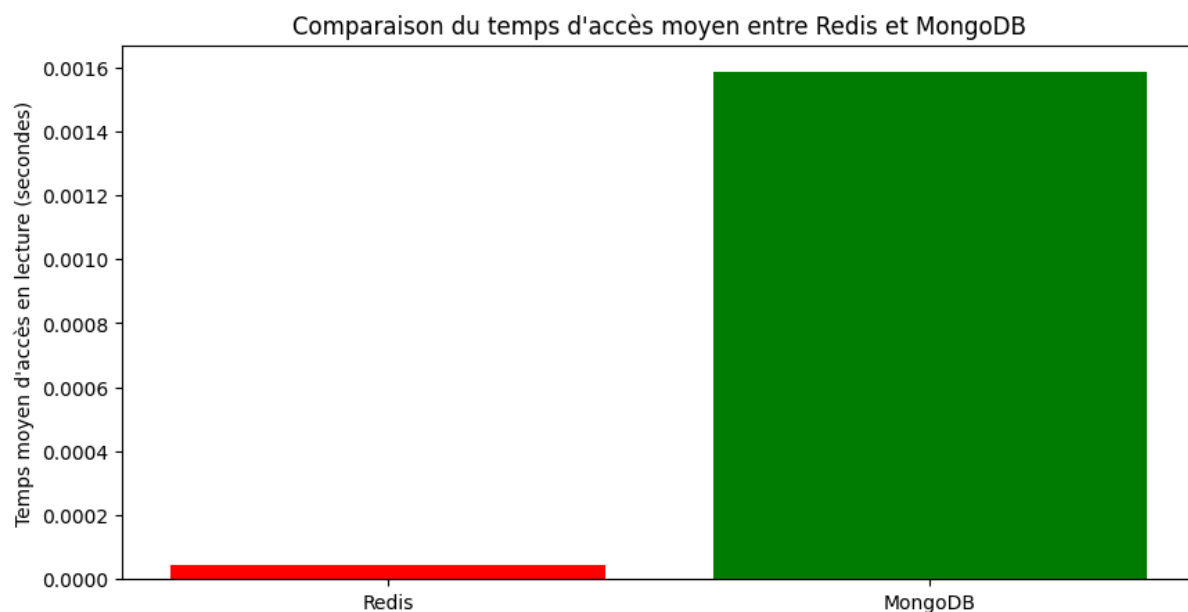
Pour ce qui est du backend, si on peut appeler ça un backend, nous sommes partis sur une API Rest en **Java Spring Boot**. Pour réaliser l'api, nous avons longuement hésités entre Spring Boot et FastAPI mais voici une liste des arguments qui nous a départagés:

- Premier point, Spring Boot est un framework en Java qui est un langage de programmation orienté objet et donc fortement typé, permettant ainsi de contrôler presque parfaitement les données en entrées et sorties de l'api contrairement à Python qui ne l'est pas.
- Deuxièmement point, pour des charges intensives ou simultanées, Python sera limité en terme de performances à cause d'overhead due à la GIL(Global Interpreter Lock) tandis que Java offrira des performances plus stable grâce à sa compilation et son exécution sur la JVM(Java Virtual Machine) qui

permet une exécution plus rapide et une meilleure gestion des threads.

L'API Rest sera construite sur Java Spring Boot avec des dépendances à jour et sans vulnérabilités comme Spring Web, Spring Data JPA, Spring Dev Tools et MongoDB Driver. Oui, en effet, le SGBD NoSQL qu'on a choisi pour l'application est **MongoDB** mais pour aboutir à ce choix nous avons comparé MongoDB et Redis dans notre cas d'usage. La base de données de l'application sera accessible uniquement par des opérations de lecture car il s'agit seulement d'un sélecteur multimodal donc nous avons comparé les temps d'accès en lecture entre la structure de données probabiliste bloom filter en Redis et une collection indexée en MongoDB. La collection indexée sera insérée avec 100 000 documents idem pour le bloom filter qui aura un ensemble de 100 000 données mais en réalité notre base de données métier de voitures ne comporte seulement que 47 000 éléments mais dans le cas où la base de données de l'application augmente significativement avec le temps, nous avons fait le choix de prendre exprès 100 000 éléments.

Voici donc un tableau qui compare les temps d'accès moyen entre Redis et MongoDB dans les conditions défini juste avant:

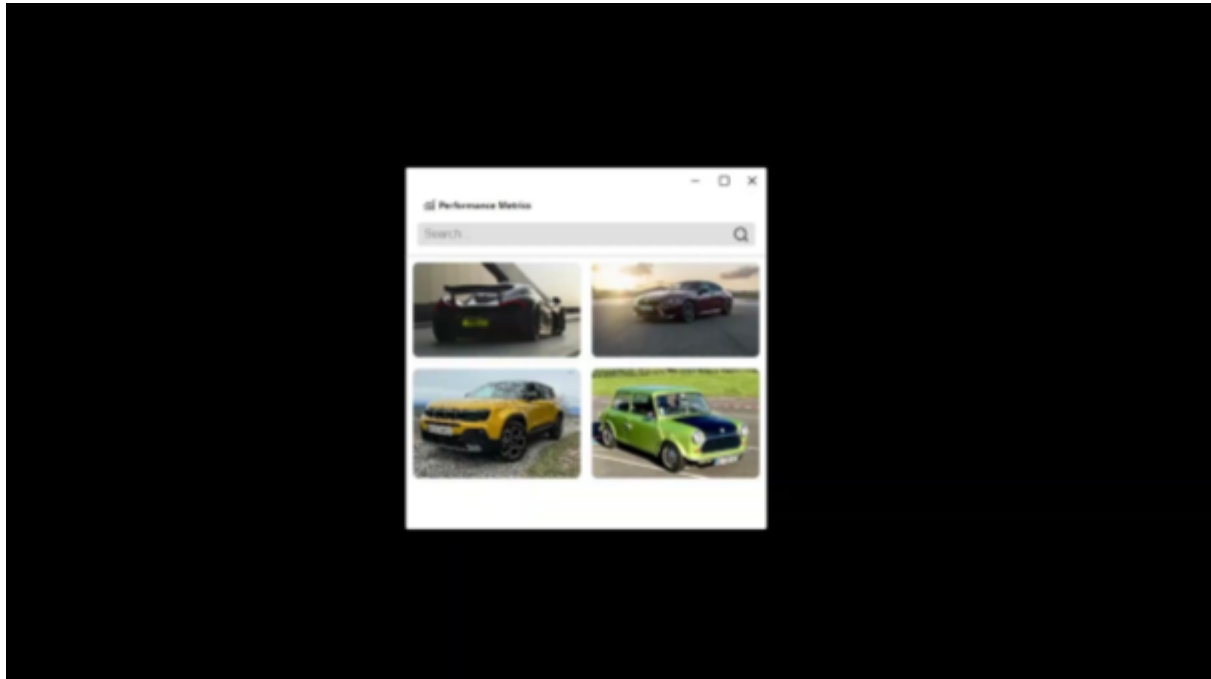



Redis est sans aucun doute un SGBD NoSQL qui propose des performances élevées dans l'accès en lecture à la base de données car il est orienté en mémoire mais surtout qu'il peut stocker les requêtes en cache ce qui est un gain de temps non négligeable. Notre application a besoin de scalabilité car chaque année plusieurs modèles de véhicule peuvent être commercialisés et malheureusement Redis n'est pas fait pour stocker une grande quantité de données tandis que MongoDB est fait pour ça. MongoDB permet aussi de stocker les données sous forme de documents JSON, qui est un format idéal pour représenter les metadata des voitures mais ce format permet aussi d'avoir des données assez structurées mais surtout flexibles par rapport à Redis qui est assez strict sur ses structures de données. Le seul inconvénient de MongoDB est d'être moins performant que Redis mais qui reste tout de même performant avec en moyenne environ **0.0016 secondes** pour des opérations de lecture dans une collection indexée de **100 000 documents**.

Pour le déploiement de l'application web, qui permettra d'installer l'application desktop, nous avons utilisé l'hébergeur Vercel avec un nom de domaine qu'on a pu acquérir sur le site IONOS. Pour l'application desktop, nous avons déjà dans un premier temps dû héberger l'API Rest Spring Boot et pour se faire, nous avons pris une machine virtuelle sur AWS(Amazon Web Services) via le service Amazon EC2 qui propose une instance gratuite pour 750h avec 1 CPU et 1 GiB de Mémoire. Nous avons donc mis en place l'environnement pour que l'API Rest puisse être accessible et fonctionnel c'est-à-dire création d'une image docker mongodb avec la base de données de l'application et une image de l'API Rest avec toutes les dépendances nécessaire et l'orchestration des conteneurs pour que l'API et la base de données puisse communiquer sur les bons ports. Bien sûr, l'API sera accessible sur une seule origine, l'application desktop, défini dans ses propriétés pour des raisons de sécurité évidentes comme par exemple éviter que des bots sur internet accèdent à notre API Spring Boot.

Voici le lien github de l'organisation : Vizucar

Vidéo de présentation de l'application



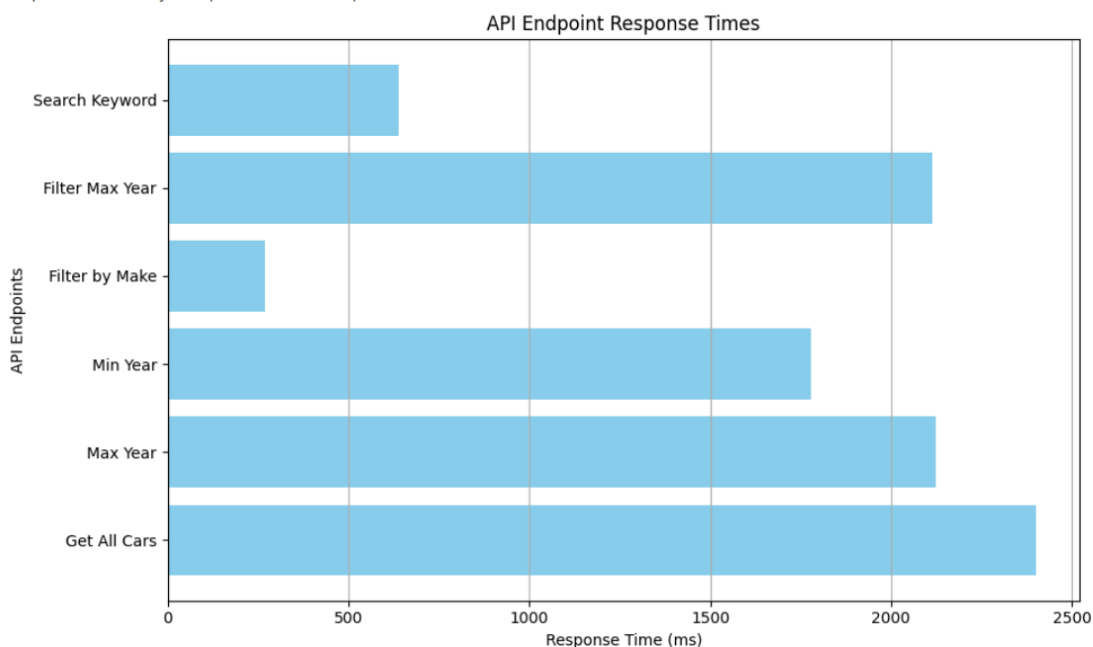
Lien de la vidéo:  vizucar.mkv

3 - Optimisation

Pour notre projet, nous avons utilisé une recherche linéaire simple comme algorithme de sélection. Cette approche consiste à parcourir chaque document de la base de données pour vérifier s'il correspond aux critères spécifiés par l'utilisateur, comme la marque, le modèle, ou d'autres attributs. L'objectif était de permettre à l'utilisateur de rechercher des voitures de manière efficace. Les avantages de la recherche linéaire sont la simplicité et la flexibilité. L'algorithme de recherche linéaire est facile à comprendre, implémenter et déboguer. Il permet de parcourir les documents de manière séquentielle, ce qui fonctionne pour tout type de filtre ou de critère, sans nécessiter de configuration particulière.

Pour évaluer les performances de notre application, nous avons réalisé des tests sur les principaux endpoints de l'API avec un script python. Ces tests ont permis de mesurer les temps de réponse pour différents scénarios d'utilisation avec une base de données contenant 47 000 documents. Les résultats sont présentés sous forme de graphiques et ils montrent des variations significatives selon les types de requêtes effectuées.

Endpoint: Get All Cars | Time: 2401.83 ms | Status Code: 200
Endpoint: Max Year | Time: 2122.81 ms | Status Code: 200
Endpoint: Min Year | Time: 1777.89 ms | Status Code: 200
Endpoint: Filter by Make | Time: 270.58 ms | Status Code: 200
Endpoint: Filter Max Year | Time: 2115.51 ms | Status Code: 200
Endpoint: Search Keyword | Time: 639.86 ms | Status Code: 200



L'endpoint Get All Cars qui récupère l'ensemble des documents de la base de données, est celui qui affiche le temps de réponse le plus élevé avec une moyenne de 2400 ms. Les endpoints Max Year et Min Year, utilisés pour identifier respectivement l'année la plus récente et la plus ancienne parmi les voitures ont des temps de réponse moyens de 2100 ms et 1700 ms. Ces opérations nécessitent un parcours linéaire de l'ensemble des documents pour effectuer une comparaison sur le champ year. L'endpoint Filter by Make, qui filtre les voitures par marque, affiche un temps de réponse beaucoup plus rapide avec une moyenne de 270 ms. Cela s'explique par l'utilisation d'index par MongoDB qui permet de limiter le nombre de

documents parcourus lors de la recherche. Cela démontre l'importance des index dans l'amélioration des performances. La recherche de Search Keyword avec un temps de réponse moyen de 600 ms, utilise une expression régulière pour parcourir les champs. L'endpoint Filter Max Year qui combine un filtre sur une marque avec la recherche de l'année maximale, présente un temps moyen de 2100 ms. Ce temps élevé reflète la complexité des requêtes combinées.

L'algorithme de recherche repose sur une méthode linéaire, complétée par des requêtes MongoDB pour des filtres simples et des expressions régulières. La recherche linéaire est utilisée pour des opérations nécessitant de parcourir l'intégralité des documents afin d'appliquer une condition particulière, comme dans les cas des endpoints Max Year et Min Year. Cet algorithme examine chaque document un par un pour identifier celui qui correspond au critère. La complexité est $O(n)$, chaque document est visité une seule fois, ce qui rend l'algorithme simple et efficace pour des collections de taille modérée. Pour des recherches simples, comme le filtrage par marque, MongoDB peut tirer parti des index appliqués sur certains champs. Ces requêtes sont rapides car MongoDB n'a pas besoin de parcourir tous les documents. La complexité est $O(\log n)$ grâce à l'utilisation d'index. Cette approche est beaucoup plus rapide que la recherche linéaire. Dans l'application, les images associées aux voitures sont stockées sous forme d'URL dans le champ `image_url` de chaque document MongoDB. Ces URLs permettent au client de charger les images directement via HTTP.

4 - Aspects communicationnels

Les premières étapes de notre collaboration sur ce projet ont été dédiées à la recherche d'un nom pour notre équipe. Pour cela nous nous sommes concertés via nos différents moyens de communications (notamment le serveur discord) dans le but d'établir une certaine productivité selon une méthode de création. Nous avons donc analysé les critères énoncés pour

ensuite échanger sur les idées de tous ceux présents dans le groupe. Le nom "Cerberus", soumis par le membre de notre équipe Yannick Khamis, est une référence à la mythologie grecque. Le chien à trois têtes gardien de la porte des enfers semblait être adapté à la phase deux de ce projet : Cyber[EDU]. Cet emblème de notre groupe est d'ailleurs affilié au github du sélecteur multimodal.

En dehors de la confection du groupe, la première tâche de ce dernier a été la réalisation d'une vidéo de sensibilisation à la cybersécurité. Dans le cadre de cette production la vision du résultat final, de cette projection dans le futur, était unanime. l'ensemble des idées ont pu être réalisés dans le délai imparti. De part cette symbiose sur la réalisation et les différents points abordés dans la vidéo, nous étions plutôt tournés vers un management participatif. Aucun d'entre nous n'a pris en charge seul une partie de ce projet. Bien que nos ressources ne nous permettait pas un logiciel performant de montage vidéo nous avons pu nous coordonner afin d'avoir une qualité vidéo correspondante à nos objectifs initiaux. Le montage a été produit par Vithurzen Vilvarajah qui a mis l'ensemble des vidéos prises lors des tournages sur un google drive afin que Yannick Khamis puisse peaufiner le résultat final. Les risques liés à l'environnement externe tels que la météo ou l'emploi du temps de chacun à pu être évité.

Cependant, on peut relever un manque d'inclusion. Notre vidéo est plus démonstrative qu'explicative, ce qui permet à tous ceux qui peuvent la voir de comprendre et ce peu importe leurs langues de communication (française, anglaise...). Sauf que les trente dernières secondes s'adressent à un public francophone. C'est donc avec cet élément en tête et sur notre cohésion d'équipe que nous avons abordé la phase 3.

La conception du logo de notre sélecteur multimodal à suivi la même méthode que précédemment. Nos recherches étant plus en lien avec notre subjectivité et notre choix de sujet nous avons échangés plusieurs idées. Plusieurs maquettes de

logos ont été abordées et désignées malgré le peu de compétence design dans notre équipe. C'est lorsque nous avons pensé à garder une éthique ainsi qu'un style simple, minimaliste et épuré pour l'entièreté du projet que les éléments commençaient à coïncider. Un logo simple mais efficace est plus facilement reconnaissable. S'ajoute à cela la touche minimaliste qui peut ajouter une touche d'élégance. Pour cela le choix des couleurs à son importance. Nous savons (grâce aux notions de communications) que ces dernières peuvent évoquer ou sont généralement associés à des émotions.



Logo de l'application, Vizucar

C'est pour cette raison que selon notre direction artistique nous avons opté pour du blanc sur du noir avec une touche de rouge pour le logo. Le noir apporte une sobriété qui contrasté avec le blanc permet une élégance en plus d'apporter une lisibilité claire de par le fait qu'il s'agisse de "couleurs opposées". Le logo représentant le nom de l'application étant écrit en blanc relève le minimalisme recherché par notre équipe. De plus, ce contraste peut souligner un lien avec le raffinement et le luxe, ce qui se joint à l'une des notions des voitures (de luxe), et par extension en devient l'une de ses notions de notre sélecteur. La touche de rouge permet d'éviter un aspect froid, austère et distant puisqu'il s'agit d'une couleur intense évoquant la passion. L'ajout de cette couleur seulement sur la première lettre empêche le spectateur de se sentir agressé et peut lui

apporter un dynamisme. Avec cette charte graphique les maquettes progressèrent rapidement permettant, une fois la forme et l'environnement externe du projet établi, de se concentrer sur notre application: Vizucar.

Le nom de notre application est Vizucar. Il s'agit d'une contraction du mot "Visualization" et "car" étant deux mots anglais pour visualisation et voiture. Le logo comme le nom de notre application est court et reflète l'objectif de ce projet. Vizucar est un sélecteur multimodal de voiture. Il se définit comme un projet d'ingénierie. Le but est de pouvoir retrouver un modèle spécifique grâce aux spécificités propres à chacun selon une large gamme de voitures. Puisque ce sélecteur recouvre différentes marques allant de la plus conventionnelle à celles les plus luxueuses, on retrouve la notion de sophistication apportée par le logo. Nous n'avons pas peaufiné d'histoire de marque mais le but est d'aider la recherche d'une voiture selon les besoins de chaque client. C'est pourquoi notre slogan est "Visualize your dream car". Le style minimaliste se poursuit sur notre site web comme l'interface utilisateur de l'application : ce dernier cherche une voiture selon la marque, le modèle et autres caractéristiques et il lui est retourné un ensemble de photos adaptés à sa recherche. L'essentiel est fourni sans trahir l'identité du projet, proposer à l'utilisateur un ensemble de voitures à sa convenance.

Le positionnement marketing à été une notion abordée lors de quelques réunions. Tout d'abord nous savions que notre sujet pour le sélecteur n'as pas été très convoités comparés à celui des Pokémon. Nous proposons donc une solution unique pour une clientèle aussi bien francophone que anglophone.

Ensuite, pour être plus précis sur les utilisateurs potentiels de cette application, nous pensions nous adresser à un public de jeunes adultes. Cette catégorie de public, notamment les jeunes conducteurs, sont en général à la recherche d'une voiture convenant à leurs espérance ou se rapprochant des voitures utilisées lors de leurs heures de

conduite. Être capable de leurs proposer un moyen de recherche selon leurs envies a été une approche intéressante. Il s'agit d'une clientèle qui ne s'épuise pratiquement pas dans le temps. Des anciens utilisateurs peuvent même y retourner pour la recherche de leurs nouvelles voitures, éloignant ainsi l'usage unique par utilisateur de cette application. S'ajoute à cela les perspectives d'évolution de l'application, ajoutant par exemple les caractéristiques de chaque voiture. Cet ajout peut permettre la comparaison directe entre plusieurs gammes de voitures de marques différentes. Sachant que les sites de ces dernières ont tendance à n'apporter que les aspects positifs de leurs produits, Vizucar pourrait être une solution à but informatif et objectif. Cette option amène un argument supplémentaire pour une clientèle de conducteur, de futur conducteur ou encore de passionnée d'automobile.

Enfin, nos moyens de communication pour promouvoir notre application seront via les méthodes classiques. Nous utiliserons nos LinkedIn pour mettre en avant cette solution, les avantages ainsi que les technologies utilisées. De plus, nous avons développé un site web respectant le W3C (l'organisme World Wide Web Consortium) ce qui permet un meilleur référencement et une meilleure visibilité en plus de pouvoir télécharger Vizucar. Ce site web se fonde à l'identité visuelle du projet en respectant la charte graphique du projet via l'utilisation des couleurs du logo (#ff3131, #000, #fff) et une interface sobre et minimaliste. Basés sur les plans de Virthuzen, Mohamed et Yannick effectués sur Figma, ce dernier a conçu en Next.js un site web délivrant le projet à tous ceux disposant d'une connexion internet.

Conclusion

En conclusion, le projet Vizucar a été une expérience enrichissante qui a combinée des compétences en ingénierie logicielle, en gestion de projet, et en communication. En adoptant des framework modernes comme Electron.js, React, et Spring Boot, nous avons essayé de concevoir au mieux une

application performante et cross-plateforme, soutenue par une base de données optimisée grâce à MongoDB. Notre travail collaboratif, structuré autour de Trello, Discord, et GitHub, a permis une coordination fluide et autonome et une gestion efficace des tâches. Enfin, notre approche minimaliste en matière de design et de communication, alliée à un positionnement marketing réfléchi, reflète notre ambition de proposer une solution innovante et accessible, répondant aux besoins variés de nos utilisateurs et proche de ce qui se fait dans le monde professionnel.