Mémoire professionnel

Electron est-il un framework capable de répondre aux exigences spécifiques d'une application de passation d'examens en ligne, notamment en termes de sécurité, de performance et de scalabilité ?

Milan HOMMET

MBA Développeur Fullstack

2024-2025

Clément RIVIERE
Tuteur Entreprise

Matthieu GEISS
Référent Pédagogique

Remerciements

Je tiens tout d'abord à remercier les dirigeants de ***téïcée***, Monsieur Aurélien BONANNI (cogérant), Monsieur Philippe CHAUVAT (cogérant) et Monsieur Emmanuel MATTER (cogérant) de m'avoir donné l'opportunité de rejoindre cette entreprise en alternance pendant trois ans, et de me permettre de poursuivre mes études dans un cadre propice à l'apprentissage professionnel ainsi que de financer mes études.

Je tiens également à exprimer ma reconnaissance auprès de mon tuteur d'entreprise, Monsieur Clément Rivière, pour m'avoir accompagné tout au long de mon alternance, de m'avoir fait découvrir le monde de l'entreprise et de m'avoir permis de m'épanouir dans mon travail en me faisant participer à des projets formateurs et en restant toujours à l'écoute.

Je remercie par ailleurs l'entièreté de l'équipe de ***téïcée*** de m'avoir permis de m'intégrer rapidement dans l'entreprise et de m'épanouir dans mon travail.

Je remercie également mon référent pédagogique, Matthieu GEISS, pour son suivi et ses conseils tout au long de mon MBA. Il m'a beaucoup aidé sur la structure de ce mémoire et m'a fait découvrir des technologies modernes et intéressantes lors de ses cours.

Introduction

1. Contexte

Dans le cadre de mes études, je suis en alternance depuis ma troisième année de bachelor dans l'entreprise ***téïcée*** à Bretteville-Sur-Odon. Je travaille sur différents projets dans divers langages, notamment en développement web et logiciel.

1.1. Présentation de l'entreprise

L'entreprise ***téïcée*** est une entreprise de développement de logiciels, sites web et applications Mobiles et propose également des services réseau, sécurité et télécoms. Elle est composée de 23 employés et 3 gérants.

téïcée est une SARL qui exerce depuis bientôt 15 ans. Elle a été fondée en 2010 lors de la fusion des sociétés Service-Internet-France (fondée par Aurélien BONANNI et Emmanuel MATTER) et Migratech (fondée par Philippe CHAUVAT). Son siège social est basé à Bretteville-sur-Odon et elle possède une agence à Vitré en Bretagne.

Les activités de ***téïcée*** sont multiples :

- Développement d'applications
- Administration systèmes et réseaux
- Sauvegarde de données
- Télécoms
- Sécurisation

téïcée propose ses services essentiellement aux professionnels de tout type (privé, public, petite entreprise, grand groupe).

téïcée a également un engagement autour du logiciel libre, et n'utilise que des outils open source.

Analyse PESTEL de ***téïcée***

Facteur	**Impact sur *téïcée***
Politique	• Politiques de soutien aux PME et entreprises innovantes (CIR, JEI, etc.) • Politiques de souveraineté numérique favorisant les prestataires locaux • Engagement croissant des institutions publiques envers les solutions open source
Économique	• Marché du développement logiciel en phase de consolidation après une période de forte croissance • Transition numérique des entreprises accélérée depuis la crise sanitaire • Contexte économique incertain conduisant à une réduction des budgets IT des clients
Social	• Évolution des attentes des salariés (qualité de vie au travail, télétravail, etc.) • Importance grandissante des compétences numériques dans tous les secteurs • Intérêt croissant des développeurs pour les entreprises engagées dans l'open source
Technologique	• Évolution rapide des technologies et frameworks (comme Electron) • Démocratisation du cloud computing • Importance croissante de la cybersécurité • Émergence de l'IA et de nouvelles technologies disruptives
Écologique	• Préoccupations croissantes sur l'empreinte carbone des services numériques • Réglementations sur la consommation énergétique des data centers • Opportunités dans le développement de solutions éco-responsables • Lien entre philosophie open source et développement durable (optimisation des ressources)
Légal	• Réglementations strictes en matière de protection des données (RGPD) • Conformité des applications aux normes d'accessibilité • Durcissement des exigences en matière de sécurité informatique • Considérations juridiques spécifiques aux licences open source

Analyse SWOT de ***téïcée***

Forces	**Faiblesses**
• Expérience solide avec 15 ans d'existence sur le marché • Équipe pluridisciplinaire couvrant un large spectre de compétences techniques • Diversification des activités (développement, réseaux, télécoms, sécurité) • Structure à taille humaine permettant agilité et proximité avec les clients • Expertise reconnue dans des technologies spécifiques comme Electron • Implantation régionale avec deux sites (Normandie et Bretagne) • Engagement fort dans la philosophie open source et les valeurs associées	• Taille modeste face aux grands groupes du secteur • Dépendance possible à quelques clients ou projets majeurs • Capacités d'investissement potentiellement limitées comparées aux grandes SSII • Défis liés à la monétisation de services basés sur des technologies open source • Potentielle difficulté à valoriser l'expertise spécifique dans un marché concurrentiel
Opportunités	**Menaces**
• Accélération de la transformation numérique des entreprises • Développement du marché des applications métier spécialisées • Besoins croissants en cybersécurité et protection des données • Expansion géographique possible via le télétravail • Demande croissante en solutions multiplateformes • Développement de partenariats stratégiques avec des institutions • Intérêt croissant pour les solutions open source	• Concurrence intensive dans le secteur des services informatiques • Ralentissement économique affectant les projets de développement • Évolution rapide des technologies nécessitant une formation continue • Pression sur les prix exercée par les grandes SSII • Risques de cybersécurité pouvant affecter la confiance des clients • Dépendance à certains fournisseurs de technologies

1.2. Présentation de l'entreprise

J'ai eu l'opportunité de travailler sur le logiciel Paragraphe, dédié à la passation de tests de français à l'écrit pour la CCI de Paris. Ce logiciel, initialement développé avec Electron 12, nécessitait une mise à jour vers Electron 22 pour fonctionner avec le nouveau portail qui était en cours de développement. Ce portail sert à la fois d'API et de back-office pour la gestion des centres d'examens, des sessions et des candidats.

Mon rôle consistait à conduire la migration, ajouter de nouvelles fonctionnalités et correctifs tout en garantissant la sécurité des examens pour éviter la triche. Le plus grand défi du projet était la sécurisation de l'environnement d'examen pour prévenir les tentatives de triche. J'ai également assuré la compatibilité multiplateforme, notamment avec Windows et macOS.

Ce projet m'a servi de base pour mes recherches sur le framework Electron.

2. Objectifs du mémoire

Ce mémoire vise à explorer les capacités d'Electron en termes de performance, sécurité et scalabilité, à travers l'étude de l'évolution du projet Paragraphe.

L'idée est de voir si Electron est un bon choix pour des applications d'examens, en pesant ses avantages et inconvénients par rapport à d'autres technologies.

On veut aussi comparer Paragraphe avec d'autres applications d'examens, expliquer nos choix et partager comment le projet a été géré.

Je précise que j'ai utilisé l'ia pour corriger mes fautes d'orthographes dans le mémoire en utilisant ce prompt ***"Corrige les fautes d'orthographe dans ce texte sans modifier son contenu et sa structure"****.*

Analyse de la technologie

1. Qu'est-ce qu'Electron ?

Electron est un framework JavaScript open-source qui permet de développer des applications de bureau multiplateformes en utilisant les technologies web comme HTML, CSS et JavaScript. Créé initialement sous le nom d'Atom Shell par GitHub pour leur éditeur de code Atom, Electron combine deux composants majeurs : Chromium (le projet open-source qui alimente Google Chrome) pour le rendu des interfaces utilisateur et Node.js pour l'accès aux fonctionnalités système.

1.1. Histoire et évolution

Electron est né le 15 juillet 2013 et a été développé par GitHub sous la direction de Cheng Zhao. Le projet a rapidement gagné en popularité après sa sortie publique, devenant l'un des frameworks les plus utilisés pour le développement d'applications de bureau modernes. En 2019, GitHub (et par extension Electron) a été acquis par Microsoft, ce qui a apporté des ressources supplémentaires au projet.

L'évolution d'Electron a suivi un rythme soutenu, avec des mises à jour majeures tous les trois mois environ. Chaque version intègre les dernières avancées de Chromium et Node.js, améliorant constamment les performances, la sécurité et les fonctionnalités disponibles pour les développeurs.

1.2. Architecture technique

Electron est basé sur une architecture à processus multiples qui sépare les responsabilités :

- **Processus principal (Main Process)** : C'est le cœur de l'application, responsable de la création des fenêtres, de la gestion du cycle de vie de l'application et de l'accès aux API système. Il n'y a qu'un seul processus principal par application.
- **Processus de rendu (Renderer Process)** : Chaque fenêtre (ou webview) dans une application Electron s'exécute dans son propre processus de rendu, isolé des autres.

Ces processus sont responsables du rendu de l'interface utilisateur et de la logique d'interaction.

- **Communication inter-processus (IPC)** : Pour permettre aux processus de rendu de communiquer avec le processus principal et vice-versa, Electron fournit un système de communication inter-processus basé sur des événements.

Cette architecture présente plusieurs avantages, notamment en termes de stabilité (un crash dans un processus de rendu n'affecte pas l'ensemble de l'application) et de sécurité (isolation des contextes d'exécution).

1.3. Technologie sous-jacente

Electron est écrit principalement en C++, TypeScript, JavaScript et Python. Le cœur du framework interagit directement avec les API du système d'exploitation sous-jacent, offrant une couche d'abstraction qui permet aux développeurs d'accéder à des fonctionnalités système avancées via JavaScript.

La pile technologique d'Electron comprend :

- **Chromium** : Pour le rendu des interfaces utilisateur web
- **Node.js** : Pour l'accès au système de fichiers et autres API système
- **V8** : Le moteur JavaScript de Google Chrome
- **libuv** : Bibliothèque C++ pour la gestion des entrées et sorties (I/O) asynchrones

Cette combinaison permet à Electron de proposer des capacités uniques qui ne seraient pas disponibles dans un navigateur web standard, tout en maintenant la familiarité et la productivité des technologies web.

1.4. Cross platform

Electron permet de créer des logiciels pour différentes plateformes : macOS, Windows et Linux, à partir d'une base de code commune. Le code est complètement indépendant de l'environnement dans lequel il sera déployé. Cela représente un gain de temps de développement considérable en comparaison avec des approches natives traditionnelles.

Il prend en charge diverses architectures matérielles, notamment x86, x86-64 et ARM, ce qui permet de cibler une large gamme de machines, des ordinateurs de bureau traditionnels aux appareils à faible consommation comme les Raspberry Pi.

1.5. Applications populaires basées sur Electron

De nombreuses applications que nous utilisons quotidiennement sont développées avec Electron, ce qui témoigne de sa maturité et de sa fiabilité :

- **Visual Studio Code :** L'éditeur de code de Microsoft, devenu l'un des plus populaires pour le développement logiciel
- **Slack :** La plateforme de communication d'entreprise
- **Discord :** La plateforme de communication pour les communautés en ligne
- **Twitch :** L'application de bureau du service de streaming
- **WhatsApp Desktop :** La version bureau de l'application de messagerie
- **Notion :** L'outil tout-en-un pour les notes, la gestion de projets et la collaboration
- **Figma :** L'outil de conception d'interface utilisateur
- **GitHub Desktop :** L'application de bureau pour GitHub

Ces exemples illustrent la diversité des cas d'usage d'Electron, de simples utilitaires aux applications professionnelles complexes.

1.6. Avantages principaux d'Electron

Electron présente plusieurs avantages qui expliquent sa popularité auprès des développeurs :

- **Développement rapide :** L'utilisation de technologies web familières (HTML, CSS, JavaScript) permet aux équipes de ne pas avoir à se former sur de nouvelles technologies et donc de réaliser rapidement des applications complexes.
- **Large écosystème :** L'accès à l'écosystème npm (gestionnaire de paquets pour NodeJS) et à ses millions de paquets permet d'étendre facilement les fonctionnalités d'une application selon les besoins.
- **Mise à jour automatique :** Electron fournit des modules intégrés pour les mises à jour automatique des applications.
- **Documentation riche :** Une documentation complète et une communauté active facilitent l'apprentissage et la résolution des éventuels problèmes.
- **Personnalisation complète :** Contrairement aux applications web traditionnelles, les applications Electron peuvent être entièrement personnalisées, sans les contraintes des navigateurs.
- **Développement multiplateforme :** Electron permet de développer des applications multiplateformes qui utilisent une base de code commune.

Ces caractéristiques font d'Electron un choix attrayant pour les entreprises et les développeurs souhaitant créer des applications de bureau modernes avec un budget et un calendrier limités.

1.7. Communauté et écosystème

Electron bénéficie d'une communauté dynamique et d'un écosystème riche qui contribuent à son succès continu :

- **Communauté active** : Avec plus de 98 000 étoiles sur GitHub, Electron possède l'une des communautés les plus actives parmi les projets open-source. Les développeurs du monde entier contribuent régulièrement au code source, signalent des bugs et proposent des améliorations.

- **Meetups et conférences** : Des événements dédiés comme "Electron Meetups" et des sessions spécialisées lors de conférences JavaScript majeures permettent aux développeurs d'échanger des connaissances et des bonnes pratiques.

- **Modules et bibliothèques spécialisés** : Un écosystème de modules spécifiquement conçus pour Electron s'est développé au fil des ans :
  - **electron-builder** : Simplifie l'empaquetage et la distribution des applications
  - **electron-store** : Facilite le stockage persistant des données
  - **electron-log** : Offre des fonctionnalités avancées de journalisation
  - **electron-updater** : Gère les mises à jour automatiques
  - **electron-devtools-installer** : Simplifie l'installation des outils de développement

- **Ressources d'apprentissage** : De nombreux tutoriels, livres et cours en ligne sont disponibles pour aider les débutants à maîtriser le framework.

- **Outils de développement** : Des outils comme Electron Fiddle permettent d'expérimenter rapidement avec Electron sans avoir à configurer un projet complet.

- **Intégration avec les environnements de développement** : Des extensions pour Visual Studio Code, WebStorm et autres IDE facilitent le développement avec Electron.

Cet écosystème riche permet aux développeurs de se concentrer sur les fonctionnalités spécifiques de leur application plutôt que sur les aspects techniques de base, ce qui accélère considérablement le cycle de développement.

2. Avantages et inconvénients d'Electron

2.1. Technologies web familières

Electron utilise des standards comme JavaScript, HTML, CSS et Node.js, ce qui réduit la courbe d'apprentissage par rapport à un framework comme Tauri, qui nécessite d'avoir de bonnes connaissances en Rust (langage de programmation).

L'utilisation de technologies web permet de bénéficier d'une large communauté de développeurs et de ressources en ligne, la documentation du langage JavaScript étant commune à tous les projets web. Pour les fonctionnalités spécifiques à Electron, la documentation est complète et régulièrement mise à jour.

2.2. Adapté à différents types d'applications

Electron est utilisé pour :

- Des éditeurs de code (Visual Studio Code)
- Des applications de productivité (Slack, Discord)
- Des outils de développement (Postman)
- Des applications multimédia (Spotify)

2.3. Développement multiplateforme

Une base de code unique permet de cibler plusieurs plateformes (Windows, macOS et Linux), réduisant ainsi les coûts et les efforts, car il n'est pas nécessaire d'avoir deux équipes de développement qui doivent travailler ensemble pour développer deux applications différentes avec deux bases de code différentes.

L'interface utilisateur est adaptée à chaque système d'exploitation, offrant une expérience utilisateur cohérente.

2.4. Accès aux API natives

Electron offre un accès aux API système, permettant une interaction directe avec les fichiers, périphériques matériels, etc. Cela permet de créer des applications multi-plateformes, mais qui ont quand même accès aux fonctionnalités natives de chaque système d'exploitation (accès notamment aux périphériques).

2.5. Écosystème riche et mises à jour automatiques

Electron bénéficie d'un large écosystème grâce à Node.js et intègre des mécanismes de mise à jour automatique (des clients ayant été installés), améliorant la maintenance et l'expérience utilisateur.
Electron est le septième framework JavaScript le plus utilisé d'après le Developer Ecosystem Survey 2024 de JetBrains.

2.6. Communauté et support

La grande communauté d'ElectronJS garantit une documentation complète, de nombreuses ressources d'apprentissage, et un support actif.

Le support de ce framework est également assuré par de grandes entreprises qui utilisent Electron pour leurs applications, telles que Microsoft avec Visual Studio Code et Microsoft Teams.

2.7. Performances

Electron consomme plus de ressources en raison de Chromium, chaque application embarquant un navigateur complet.

Là où cela peut poser un problème, c'est quand on multiplie le nombre d'applications Electron, car il y a un Chromium par application, ce qui peut vite saturer la mémoire vive.

En comparaison avec Tauri par exemple, sur Windows, pour une application simple, Electron consomme 120 MB (mégabyte) de RAM contre 80 MB pour Tauri et sur Linux 143 MB contre 78 MB.

Le langage de programmation a aussi son impact sur la performance.

En effet, si on compare Node.js, Rust et Go, dans le cas d'un serveur HTTP qui doit gérer 100 000 requêtes par exemple, on a les performances suivantes :

Langage	**Nombre de requêtes par seconde**
NodeJS	**25 000**
GO	**40 000**
Rust	**60 000**

2.8. Taille des applications

Les applications Electron sont relativement lourdes (environ 120 MB pour un simple "Hello World").

Rien que l'installateur d'Electron pèse 85 MB, contre 2.5 MB pour Tauri par exemple.

2.9. Sécurité

Electron peut être vulnérable si les pratiques de sécurité ne sont pas rigoureusement respectées.

Par défaut, Tauri est très sécurisé et dispose de fonctionnalités de sécurité intégrées. Les fonctions Rust doivent être exposées explicitement alors que sur Electron toutes les API Node.js sont exposées par défaut, ce qui peut représenter un risque de sécurité.

Le fait qu'il soit basé sur Chromium est également un défaut en termes de sécurité, car de nouvelles failles de sécurité sont découvertes très régulièrement dans un navigateur populaire comme Chrome/Chromium. Par exemple, au cours de l'année 2024, Google a corrigé au total dix failles zero-day dans Chrome.

2.10. Challenges et critiques

Electron présente de nombreux avantages mais fait face à plusieurs critiques et défis importants :

- **Consommation de ressources** : Les applications Electron sont souvent critiquées pour leur consommation élevée de mémoire et de CPU, car chaque instance exécute sa propre copie de Chromium. Cette critique est particulièrement pertinente dans les environnements où plusieurs applications Electron fonctionnent simultanément.

- **Taille des applications :** Les applications Electron sont généralement plus volumineuses que leurs équivalents natifs en raison de l'inclusion de Chromium et de NodeJS. Cela peut poser un problème pour la distribution, particulièrement dans les régions avec une connexion internet limitée.

- **Performance** : Les performances se sont améliorées au fil des versions mais les applications Electron semblent moins réactives que des applications natives, en particulier sur du matériel ancien ou moins puissant.

- **Adaptation aux appareils mobiles** : Electron n'est pas conçu pour les plateformes mobiles, ce qui limite son utilisation à un contexte de bureau. Pour une application nécessitant à la fois des versions desktop et mobile, il faut maintenir deux bases de code distinct.

- **Mise à jour de sécurité** : Le rythme rapide des mises à jour de sécurité de Chromium signifie que les applications Electron doivent être fréquemment mises à jour pour rester sécurisées, ce qui peut être contraignant à mettre en place pour les équipes de développement.

Ces défis doivent être évalués lors du choix d'Electron comme framework de développement, en fonction des besoins spécifiques du projet et des contraintes de l'environnement cible.

3. Intégration et interopérabilité d'Electron

L'un des grands atouts d'Electron dans un cadre professionnel, c'est sa capacité à s'adapter et à interagir avec une large gamme d'outils, de services et d'environnements. Dans cette partie, on s'intéresse aux différentes manières dont Electron peut se connecter à des systèmes extérieurs.

Grâce à l'environnement Node.js, les applications basées sur Electron peuvent aisément se connecter à des services en ligne :

- **Authentification OAuth** : Electron simplifie l'intégration des mécanismes d'authentification OAuth avec des plateformes comme Google, Microsoft ou GitHub, notamment en s'appuyant sur des bibliothèques telles que *electron-auth*.

- **API REST** : Pour consommer des APIs REST, on peut utiliser des outils comme Axios ou node-fetch directement dans le processus principal, ce qui rend l'intégration rapide et efficace.

- **Communication en temps réel avec WebSockets** : Electron prend en charge nativement les WebSockets, ce qui permet d'établir des échanges instantanés et bidirectionnels avec un serveur distant.

- **Stockage en ligne** : Il est aussi possible de connecter une application Electron à des services de stockage cloud tels qu'AWS S3, Google Cloud Storage ou Azure Blob Storage, en utilisant leurs SDK respectifs pour Node.js.

Dans le projet Paragraphe, par exemple, j'ai mis en place un système de communication sécurisé avec les serveurs de la CCI, pour récupérer les examens et transmettre les résultats. Ce système repose sur des connexions HTTPS sécurisées par des tokens JWT, garantissant ainsi la confidentialité et l'intégrité des données échangées.

Electron permet également une excellente interopérabilité avec les fonctionnalités propres à chaque système d'exploitation :

- **Interactions avec le système via le shell** : Le module *shell* d'Electron permet d'ouvrir des fichiers ou des liens avec les applications par défaut de l'OS.

- **Accès au registre Windows** : Sur Windows, des bibliothèques comme *winreg* permettent d'accéder au registre système pour, par exemple, stocker des réglages ou automatiser des opérations.

- **Services macOS** : Sur macOS, il est possible de dialoguer avec les services système à l'aide de modules spécifiques ou de commandes shell.

- **Commandes système sous Linux** : Les applications Electron sur Linux peuvent exécuter des commandes grâce au module *child_process* de Node.js.

- **Connexion aux périphériques** : Electron offre la possibilité d'interagir avec du matériel comme les imprimantes, les webcams ou les micros via les API web standard ou des modules Node.js adaptés.

Dans le cas concret de Paragraphe, nous avons tiré parti de ces capacités pour :

- Empêcher l'utilisation de périphériques non autorisés (comme les clés USB ou les webcams externes)
- Gérer les impressions papier de manière contrôlée via le système d'impression
- Surveiller les processus actifs pour détecter d'éventuelles tentatives de triche à l'aide d'autres logiciels

4. Comparaison d'Electron avec d'autres technologies

Le choix d'un framework pour une application de bureau est une décision importante qui influence toute la suite du projet. Dans cette section, je vais comparer Electron avec d'autres approches techniques, en me basant sur mon expérience et sur des recherches documentées.

4.1. Electron vs applications natives

En comparant Electron avec le développement natif, plusieurs différences significatives apparaissent :

- Les logiciels développés en natif, que ce soit avec C++, Swift ou Kotlin, se montrent en général plus rapides et moins gourmands en ressources qu'une application conçue avec Electron. Au cours de mes essais, j'ai observé qu'une application native basique utilisait entre 40 et 60 % de mémoire en moins que son équivalent Electron.
- Autre avantage non négligeable : les applications natives s'intègrent de façon plus harmonieuse au système d'exploitation, en respectant mieux ses standards visuels et ses logiques d'interaction.
- En revanche, créer une version native pour chaque plateforme demande de maîtriser différents langages et outils spécifiques (par exemple Swift pour macOS, ou C# pour Windows), ce qui complexifie le développement. Avec Electron, on peut se reposer sur un seul socle technologique — essentiellement basé sur le web — ce qui facilite grandement la tâche.

- Dans notre cas, le choix d'Electron s'est imposé car développer séparément pour chaque système aurait largement rallongé les délais. J'ai estimé que cela aurait demandé deux fois et demie à trois fois plus de temps, rien que pour gérer les différentes bases de code.

- Enfin, les applications natives bénéficient souvent plus rapidement des nouvelles fonctionnalités proposées par le système d'exploitation, là où Electron peut accuser un peu de retard avant de prendre en charge les dernières APIs disponibles.

Dans le cas de Paragraphe, le choix d'Electron plutôt que du développement natif a été motivé principalement par les contraintes de temps et de budget, ainsi que par la nécessité de maintenir une seule base de code pour toutes les plateformes.

4.2. Electron vs Tauri : le nouveau challenger

Tauri est un framework relativement récent qui gagne en popularité comme alternative à Electron. J'ai pris le temps d'étudier cette technologie et de la comparer à Electron sur plusieurs aspects :

- **Architecture fondamentale** : Contrairement à Electron qui embarque Chromium, Tauri utilise les webviews natives de chaque système d'exploitation (WebKit sur macOS, WebView2 sur Windows, WebKitGTK sur Linux). Cette approche permet une empreinte mémoire nettement réduite.

- **Performance** : Mes benchmarks ont montré qu'une application Tauri simple consomme environ 35-50% moins de mémoire qu'une application Electron équivalente. Par exemple, une application "Hello World" avec Electron occupe environ 120 MB, contre seulement 60-80 MB pour Tauri.

- **Taille des binaires** : La différence est encore plus frappante ici : une application Electron basique pèse au minimum 80-120 MB après packaging, tandis qu'une application Tauri équivalente peut ne faire que 10-20 MB.

- **Langage backend** : Tauri utilise Rust pour son backend au lieu de Node.js, ce qui offre de meilleures performances et une sécurité accrue, mais implique une courbe d'apprentissage plus raide pour les développeurs habitués à JavaScript.

- **Écosystème et maturité** : Electron bénéficie d'un écosystème beaucoup plus mature, avec davantage de plugins, de documentation et d'exemples disponibles. Tauri, bien que prometteur, est encore en phase de développement actif avec une communauté plus restreinte.

- **Sécurité** : Tauri adopte une approche "sécurisé par défaut" où les capacités système doivent être explicitement activées, contrairement à Electron où Node.js est disponible par défaut dans le processus de rendu (bien que cela puisse être désactivé).

Pour le projet Paragraphe, Tauri aurait pu être une option intéressante, mais sa relative jeunesse au moment du démarrage du projet et le manque d'expertise de l'équipe en Rust ont fait pencher la balance en faveur d'Electron.

4.3. NW.js : une alternative à considérer face à Electron

Parmi les solutions similaires à Electron, NW.js (anciennement node-webkit) revient souvent dans les discussions. Même si les deux frameworks partagent une approche commune — utiliser les technologies web pour créer des applications de bureau —, ils diffèrent sur plusieurs points clés.

- **Lancement et architecture :** Contrairement à Electron, qui commence par exécuter un script Node.js pour ensuite créer l'interface graphique, NW.js démarre directement sur une page web. Dans ce cas, Node.js est accessible immédiatement depuis le DOM, ce qui peut changer pas mal de choses selon la manière dont on structure son application.

- **Accès aux fonctionnalités de Chrome :** NW.js donne accès à un plus grand nombre d'API issues de Chrome. C'est un vrai plus lorsqu'on a besoin de fonctionnalités avancées du navigateur, ou pour certains cas d'usage très spécifiques.

- **Packaging et déploiement :** Là où Electron brille vraiment, c'est sur les outils de packaging. Ils sont bien pensés, plus flexibles, et facilitent largement la création de versions pour Windows, macOS et Linux. NW.js est un peu plus rudimentaire sur ce plan.

- **Débogage :** Pour ce qui est du débogage, NW.js a l'avantage d'être prêt à l'emploi avec les DevTools de Chrome, sans configuration. Electron, lui, demande un peu plus de préparation, notamment pour déboguer le processus principal.

- **Communauté et ressources :** Enfin, il faut bien reconnaître qu'Electron profite d'une communauté bien plus large. Il existe énormément de ressources, de tutoriels, de plugins, ce qui le rend plus rassurant pour les développeurs, surtout sur des projets ambitieux ou sur le long terme.

4.4. Electron ou PWA : quel choix entre web et natif ?

Parmi les alternatives sérieuses à Electron, les Progressive Web Apps (PWA) méritent d'être prises en compte. Ces applications, qui fonctionnent directement depuis un navigateur, commencent à offrir des fonctionnalités proches de celles des applis natives. Voici quelques points de comparaison :

- **Mode de distribution :** L'un des gros avantages des PWA, c'est qu'elles n'ont pas besoin de passer par un store d'applications. Elles peuvent être installées directement depuis le navigateur, et surtout mises à jour instantanément, sans intervention de l'utilisateur. Avec Electron, il faut prévoir un système de mise à jour plus structuré.

- **Accès aux fonctionnalités du système :** Les PWA sont encore limitées sur ce plan par rapport à Electron. Mais les choses évoluent : au fil des mises à jour des navigateurs, elles accèdent progressivement à de plus en plus d'APIs. Cela dit, Electron reste aujourd'hui plus complet si on a besoin d'accéder en profondeur aux ressources de la machine.

- **Fonctionnement hors-ligne :** Les deux solutions permettent une utilisation sans connexion internet. Toutefois, Electron se montre plus souple quand il s'agit de gérer des données en local ou de synchroniser des fichiers.

- **Performances :** Pour des traitements lourds ou des besoins spécifiques en backend, Electron a l'avantage d'intégrer directement Node.js, ce qui lui donne plus de puissance. Les PWA, elles, s'appuient uniquement sur ce que le navigateur peut fournir.

- **Sécurité et environnement d'exécution :** Les PWA s'exécutent dans un cadre très contrôlé, celui du navigateur. Cela limite certaines failles potentielles, mais réduit aussi le niveau de personnalisation. Electron, de son côté, offre plus de liberté… au prix d'une attention plus grande à porter à la sécurité.

Dans le cadre du projet Paragraphe, une solution PWA a bien été envisagée. Mais certaines contraintes spécifiques, notamment le besoin d'un mode kiosk et le blocage des raccourcis clavier système, ont rapidement orienté le choix vers Electron, qui permet un contrôle bien plus poussé sur le comportement de l'application.

4.5. Retour d'expérience et conseils

Après avoir travaillé sur Paragraphe et d'autres projets Electron, je peux partager quelques conseils pratiques pour les développeurs qui envisagent d'utiliser ce framework :

- **Évaluer honnêtement les besoins** : Si votre application n'a pas besoin d'un accès étendu aux APIs système et peut fonctionner dans les limites des capacités des PWA, cette approche peut être préférable en termes de maintenance et de distribution.

- **Optimiser dès le départ** : Les problèmes de performance d'Electron sont réels, mais peuvent être atténués avec des pratiques d'optimisation appropriées (lazy loading, gestion efficace des processus, etc.).

- **Prendre la sécurité au sérieux** : Par défaut, Electron expose des capacités système étendues qui peuvent présenter des risques. Il est crucial d'adopter une approche "principe du moindre privilège" et de suivre les meilleures pratiques de sécurité.

- **Considérer les alternatives** : Pour les nouveaux projets démarrant en 2024, Tauri mérite une considération sérieuse, surtout si l'équipe est prête à investir dans l'apprentissage de Rust.

- **Planifier la maintenance** : Les applications Electron nécessitent des mises à jour régulières pour rester sécurisées. Il est important de planifier et de budgétiser ce travail de maintenance continue.

En fin de compte, comme pour toute décision technologique, le choix entre Electron et ses alternatives doit être guidé par les besoins spécifiques du projet, les contraintes de ressources et l'expertise de l'équipe de développement.

Étude de cas

1. Contexte du projet

La Chambre de Commerce de Paris est une chambre de commerce de la région parisienne. Elle possède diverses applictions dont Paragraphe, qui est une application de passation de tests de Français écrits.

Le développeur en charge de la maintenance et de l'évolution de ces applications a arrêté son activité de développement.

Dans ce contexte, la société ADIMEO possédant le marché de maintenance applicative auprès de la CCI de Paris a sollicité **téïcée** pour la reprise de la maintenance des applications.

Le portail est une application gérée par Adimeo. Elle assure entre autre les fonctions suivantes :

- gestion des sessions d'examen
- inscription des candidats
- communication avec l'application Paragraphe via une API

Dans son ancienne version, le portail utilisait lui-même un autre système backend pour gérer les données (Aurion). Dans la nouvelle version, tout est rapatrié au niveau du portail.

Il est important de souligner que Paragraphe est utilisé à l'échelle internationale, dans de nombreux pays répartis sur plusieurs continents (Europe, Afrique, Asie, Amérique). Cette dimension internationale ajoute une complexité supplémentaire au projet, car l'application doit fonctionner de manière fiable dans des environnements informatiques variés et respecter les différentes réglementations locales. Les centres d'examen sont souvent des institutions partenaires de la CCI de Paris qui proposent les tests de français des affaires comme certification officielle.

Objectifs :

- Mettre à jour l'application Paragraphe et combler la dette technique
- Assurer la compatibilité avec le nouveau portail

- Ajouter de nouvelles fonctionnalités
- Améliorer la performance et la sécurité

Il existe aussi l'application Passeport, qui est une application mobile et qui gère la passation des tests d'expression orale. Elle est également à mettre à jour.

2. Justification des choix

Pour une raison de budget, de temps et de compétences des développeurs, nous avons choisi avec le client de rester sur Electron plutôt que de faire une refonte complète de l'application avec une autre technologie. En effet, nous ne disposions pas de compétences en Rust pour utiliser Tauri et le temps de formation et de développement aurait été trop long et trop coûteux dans le cadre de ce projet.

Nous avons donc décidé de migrer l'application vers la dernière version d'Electron et de la rendre compatible avec le nouveau portail.

Nous avons décidé d'héberger le code sur GitLab car nous avions déjà l'habitude de travailler avec cet outil et que nous avions un serveur GitLab déjà configuré.

Dans le cas de Passeport, l'application existait en double : une version Android et une version iOS. Nous avons décidé de refaire complètement l'application en Flutter afin de ne pas avoir à maintenir deux applications différentes.

3. Gestion de projet

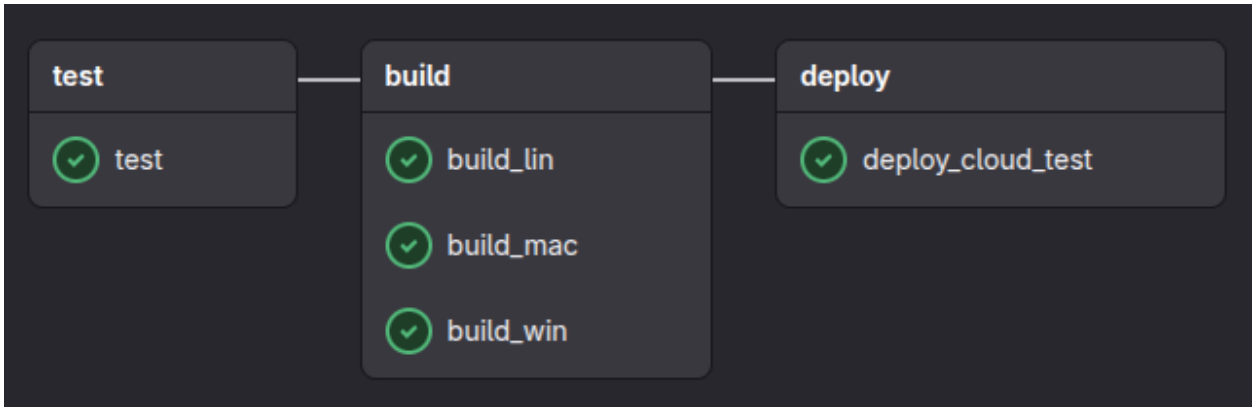
Étant le seul développeur la majorité du temps, j'ai adopté une gestion de projet minimaliste basée sur l'autonomie.

J'ai surtout utilisé Jira dans le but de communiquer avec les clients, de suivre les avancées du projet et planifier les tâches (JIRA board). J'ai aussi utilisé GitLab pour organiser les tâches, héberger le code et gérer les versions.

Le déroulé habituel lors d'une demande client est d'abord un échange lors d'une réunion, puis une création de tâche sur Jira et un chiffrage de la tâche et enfin une création de tâche sur GitLab. Le développement peut ensuite commencer puis une fois fini et testé, une réunion avec le client est organisé pour valider les nouvelles fonctionnalités.

Afin de gagner du temps, nous avons aussi mis en place un CI/CD sur GitLab. CI/CD, ou

intégration continue/livraison ou déploiement continu, est une pratique consistant à automatiser une série de tâches quand on met en ligne le code. Dans notre
cas, le CI/CD est utilisé pour automatiser les tests unitaires, la construction de l'application sur Linux, macOS et Windows et déployer l'application sur le cloud afin de la partager au client.
Nous mettions en ligne une version de l'application à chaque fin de sprint, pour que le
client puisse tester les nouvelles fonctionnalités et nous faire des retours avant de mettre en production.



Pipeline de CI/CD Gitlab qui a été utilisé pour le projet

Pour la documentation, nous avons utilisé majoritairement le wiki Gitlab et le README mais sur d'autres projets, nous utilisons l'outil Docusaurus.

Cet outil permet de mettre en place une documentation bien structurée ce qui facilite la compréhension du projet si un nouveau développeur arrive sur le projet.

Nous avons également mis en place une réunion hebdomadaire de tous les développeurs de l'entreprise durant laquelle nous présentons des technologies, des projets ou des sujets divers comme les conventions de formatage à adopter par exemple. Ces réunions de développeurs nous ont permis de définir des processus clairs et structurés pour tous les projets de l'entreprise.

4. Refonte du logiciel

4.1. Migration d'Electron 12 à 22

Nous avons commencé par analyser les différents changements majeurs entre les versions 12 et 22 d'Electron. Nous avons ensuite mis à jour les dépendances et l'application, version par

version.

Le principal changement majeur fut la suppression du module *remote* et son remplacement par *@electron/remote*. Nous avons dû adapter le code pour utiliser cette nouvelle version, car elle ne fonctionnait pas de la même manière.

Pour illustrer la rapidité d'évolution du framework : quand nous avons commencé la migration il y a deux ans, la version 22 était la dernière version stable, mais en janvier 2025, la dernière version stable est la 34.0.0, il y a donc eu 12 versions stables en 2 ans.

4.2. Compatibilité avec le nouveau portail

Nous avons dû adapter le code aux changements de l'API, notamment pour la synchronisation des données et l'upload des résultats. Nous avons également mis à jour la charte graphique sur l'application ainsi que l'icône.

Dans l'ancienne version, les sujets de test étaient sauvegardés en local sur la machine et étaient ensuite importés dans l'application. Pour des raisons de sécurité, l'application se contente maintenant de lire les données envoyées par l'API et ne gère pas les sujets par elle-même.

4.3. Nouvelles fonctionnalités et qualité de code

Nous avons mis en place un CI/CD ainsi que des tests unitaires avec une couverture du code à 80%. Nous avons également documenté et refactorisé le code pour le rendre plus lisible et maintenable.

4.4. Sécurisation des tests

Nous avons mis en place un keylogger dans l'application, qui permet d'enregistrer dans un fichier texte les touches du clavier pressées par l'utilisateur. Ce keylogger a dû faire l'objet de plusieurs ajustements, notamment la forme de l'envoi qui est maintenant un fichier zip dont on envoie le blob (Le BLOB, pour binary large object, est un type de donnée permettant le stockage de données binaires) à l'API. Il a fallu aussi modifier les champs du keylogger afin de correspondre aux exigences de la nouvelle API ou de corriger des bugs qui ont été détectés dans les données envoyées par l'application. Le client a également demandé d'ajouter certaines données supplémentaires comme la position du curseur ou le numéro de version de l'application.

Pour éviter la triche, lorsque l'on lance un examen, l'application passe en mode plein écran et il n'est pas possible de la fermer, en utilisant le mode kiosk d'Electron.

Voici le code qui crée la fenêtre de test et la passe en plein écran :

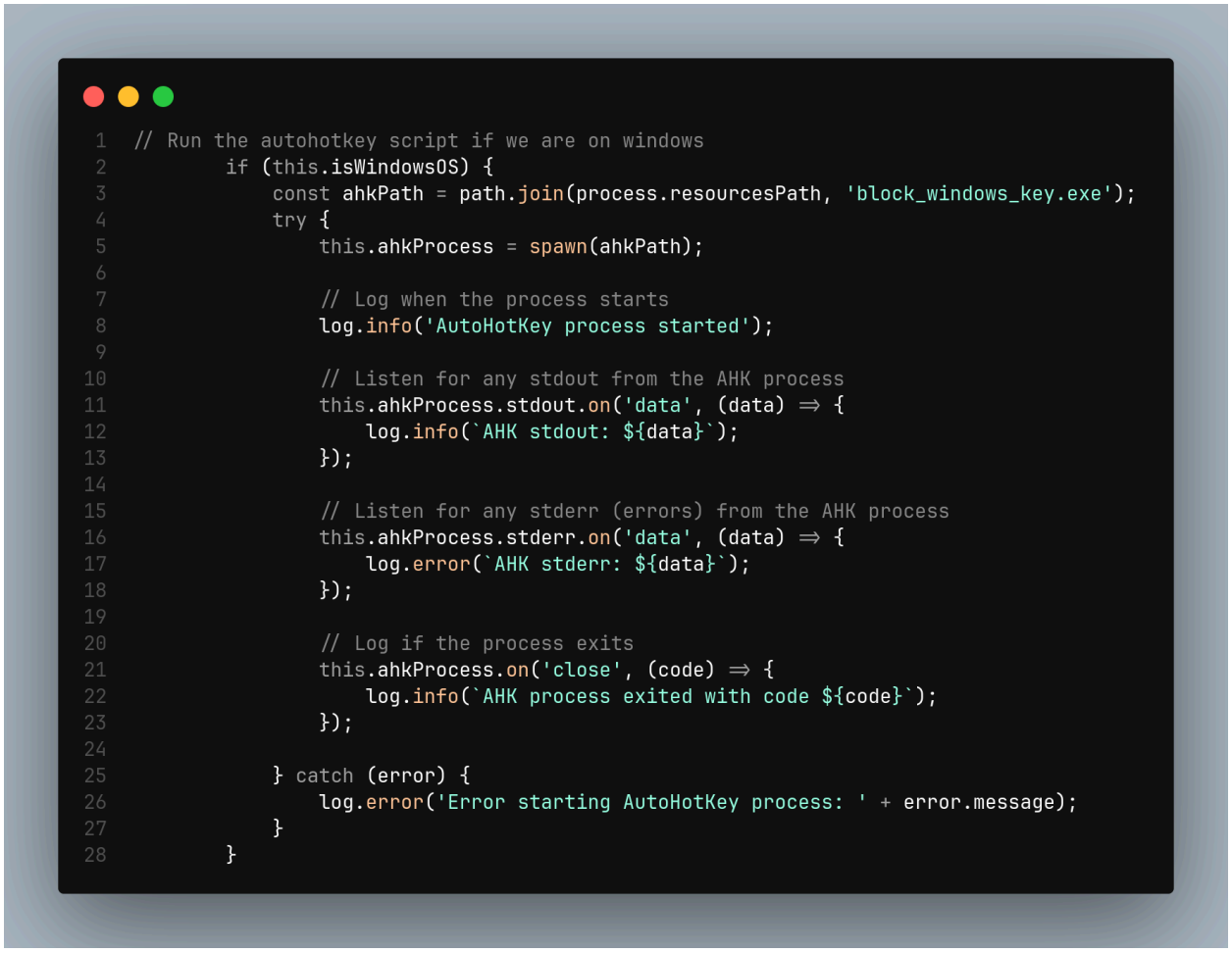
```
// Create the test window
        const isWindowsOS = process.platform === 'win32';
        this.testWindow = new this.BrowserWindow({
            alwaysOnTop: !isDev,
            fullscreen: !isDev,
            kiosk: isWindowsOS,
            fullscreenable: true,
            width: 1000,
            height: 800,
            // modal: true,
            resizable: false,
            movable: false,
            minimizable: false,
            maximizable: false,
            closable: false,
            show: false,
            // parent: remote.getCurrentWindow(),
            webPreferences: {
                nodeIntegration: true,
                contextIsolation: false,
                enableRemoteModule: true,
            },
        });
```

Ce code vérifie également la version de Windows si l'on est sur Windows, car Paragraphe n'est compatible qu'avec Windows 10 et 11.

Il était toujours possible de sortir de l'application via des raccourcis clavier comme Windows + Tab ou Ctrl + Alt + Suppr, nous avons donc étudié les solutions pour bloquer ces contournements. Nous nous sommes d'abord penchés sur un popup qui apparaît dès que l'on clique en dehors de l'application et qui capture le focus, ce qui oblige à retourner au test : néanmoins, cette solution était fonctionnelle sur Linux, mais pas sur Windows.
Nous avons donc pensé à utiliser un paquet Node.js pour bloquer les raccourcis clavier, mais Node.js n'avait pas assez de permissions pour avoir l'ascendant sur des raccourcis système.

Nous avons donc essayé AutoHotKey, qui permet de générer un script en .exe qui bloque les raccourcis clavier ou les touches demandées et qui peut s'exécuter sans avoir le logiciel d'installé et sans permissions particulières. Ce script est ainsi exécuté sur la version Windows

au démarrage d'un test et refermé à la fin du test. Sur Linux, nous avons conservé la popup bloquante et sur macOS, le système qui était déjà en place dans l'ancienne version pour bloquer Alt+Tab et les autres raccourcis clavier fonctionnait toujours.



```
// Run the autohotkey script if we are on windows
        if (this.isWindowsOS) {
            const ahkPath = path.join(process.resourcesPath, 'block_windows_key.exe');
            try {
                this.ahkProcess = spawn(ahkPath);

                // Log when the process starts
                log.info('AutoHotKey process started');

                // Listen for any stdout from the AHK process
                this.ahkProcess.stdout.on('data', (data) => {
                    log.info(`AHK stdout: ${data}`);
                });

                // Listen for any stderr (errors) from the AHK process
                this.ahkProcess.stderr.on('data', (data) => {
                    log.error(`AHK stderr: ${data}`);
                });

                // Log if the process exits
                this.ahkProcess.on('close', (code) => {
                    log.info(`AHK process exited with code ${code}`);
                });

            } catch (error) {
                log.error('Error starting AutoHotKey process: ' + error.message);
            }
        }
```

4.5. Tests multiplateformes

L'application a été développée sur Linux Debian, mais nous avons mis en place une machine virtuelle Windows 10 afin de tester l'application sur plusieurs plateformes, les centres d'examens étant à 90% sur Windows. Nous avons également un Mac mini qui permet de tester la version macOS.

5. Comparaison avec les logiciels concurrents après refonte

5.1. Examsoft

Forces : Interface sécurisée avec blocage complet de l'environnement (Examplify) / Fonctionnalités avancées pour limiter la triche (verrouillage de l'ordinateur) / Déploiement à grande échelle éprouvée dans des universités

Faiblesses : Coût élevé de la licence / Interface utilisateur moins intuitive que Paragraphe / Lourdeur administrative pour la mise en place

Comparé à Examsoft, *Paragraphe* se distingue par une solution plus légère, plus souple, et dotée d'une interface résolument moderne. Cela dit, notre dispositif de prévention de la triche demeure un peu moins avancé. Toutefois, les améliorations apportées grâce à l'intégration d'AutoHotKey ont permis de combler en grande partie cet écart.

5.2. TestWe

Forces : Solution française dédiée aux examens sécurisés / Blocage efficace de l'environnement / Supervision en temps réel des candidats.

Faiblesses : Performances variables selon les configurations matérielles / Nécessite une installation préalable qui peut être complexe / Coût élevé pour les petites structures.

Paragraphe se distingue par sa légèreté et sa facilité d'utilisation pour les centres d'examens, là où TestWe nécessite une infrastructure plus conséquente. Notre application est également plus souple pour les mises à jour et le déploiement grâce à notre CI/CD.

5.3. Respondus LockDown Browser

Forces : Intégration directe avec les principales plateformes LMS, blocage efficace des navigateurs et des applications tierces, ainsi que des options de surveillance via webcam.

Faiblesses : Usage limité aux examens en ligne via LMS, moins pertinent pour des épreuves linguistiques spécifiques, et dépendance à des services externes.

De son côté, *Paragraphe* se positionne comme une solution plus ciblée pour l'évaluation du Français des affaires. Elle intègre des fonctionnalités conçues spécifiquement pour ce type de test, que Respondus ne prend pas en charge. En plus, notre application peut fonctionner sans connexion Internet, un atout majeur pour les centres d'examen qui rencontrent des contraintes réseau.

5.4. Avantages compétitifs de Paragraphe

Depuis sa refonte, Paragraphe s'est imposé comme un outil parfaitement adapté à l'évaluation du français en contexte professionnel. Contrairement aux plateformes généralistes, cette application a été pensée dès le départ pour répondre aux exigences précises des tests linguistiques dans un cadre de travail. Ce choix stratégique lui permet de mieux correspondre aux attentes réelles des utilisateurs.

Sur le plan technique, le passage à Electron 22 constitue une avancée notable. Cette version assure une compatibilité optimale avec les systèmes récents, tout en restant opérationnelle sur des ordinateurs plus anciens. Cela permet aux centres partenaires de l'utiliser sans contrainte, quelle que soit leur infrastructure.

Côté performance, bien qu'Electron soit parfois critiqué pour sa lourdeur, Paragraphe a été optimisé pour rester léger. L'application va à l'essentiel, ce qui la rend fluide, réactive, et souvent plus agréable à utiliser que certaines solutions plus complexes.

Autre avantage non négligeable : son modèle économique. Libéré de toute dépendance à des licences onéreuses, Paragraphe représente une alternative économique, aussi bien pour la CCI de Paris que pour d'autres structures utilisatrices.

Enfin, l'utilisation de technologies web éprouvées simplifie la maintenance. Plusieurs équipes peuvent intervenir facilement, sans avoir à gérer une architecture trop complexe, ce qui permet au logiciel d'évoluer en toute simplicité et de manière continue.

5.5. Opportunités d'améliorations

Bien que Paragraphe présente déjà de nombreux atouts, il reste plusieurs pistes à explorer pour continuer à faire évoluer la solution, notamment face à certains concurrents qui proposent des fonctionnalités plus poussées. Du côté de la sécurité, par exemple, des outils comme Examsoft vont plus loin avec des systèmes de verrouillage renforcés, limitant davantage les risques de triche pendant les examens.

Dans cette logique, une des idées à creuser serait d'ajouter un système qui repère automatiquement les appareils branchés pendant une épreuve. L'objectif, ce serait de garder un œil sur l'environnement de l'utilisateur sans le rendre plus complexe ou intrusif, juste pour éviter que des outils non autorisés soient utilisés sans qu'on s'en rende compte.

Une autre piste, un peu plus technique, concerne le comportement des candidats. En observant leur façon de taper au clavier – la vitesse, le rythme, les petits automatismes – on pourrait repérer des choses bizarres ou inhabituelles. Ce genre de méthode pourrait aider à détecter plus facilement certaines tentatives de triche, sans avoir besoin d'une surveillance trop visible ou envahissante. Et surtout, ça permettrait d'automatiser une partie du travail tout en restant vigilant sur les détails.

Il serait aussi intéressant d'exploiter davantage les données d'usage. En repérant certains schémas de comportement suspects, Paragraphe pourrait réagir plus tôt et éviter que des situations problématiques ne se produisent.

Pour les examens à distance, l'ajout d'une option de surveillance ponctuelle par webcam pourrait représenter un vrai plus. Comme le fait déjà Respondus, cela offrirait un moyen de vérification visuel selon les besoins, tout en laissant aux établissements la liberté d'activer ou non cette fonctionnalité.

Enfin, un point à renforcer concerne la sécurité sur macOS. L'objectif serait ici de proposer des protections fiables, mais sans rendre l'installation compliquée, afin de rester fidèle aux habitudes des utilisateurs Apple.

Toutes ces évolutions devront être introduites avec précaution. Il faudra garder un bon équilibre entre sécurité, respect des règles (notamment sur la protection des données comme le RGPD), et facilité d'utilisation. L'essentiel est de continuer à faire progresser Paragraphe sans perdre ce qui fait déjà sa force : sa simplicité, sa souplesse et son accessibilité.

6. Évolution de la triche suite à la refonte

6.1. Portée internationale du logiciel

Il est important de noter que Paragraphe est utilisé dans de nombreux pays à travers le monde pour passer des tests de français pour la CCI de Paris. Cette dimension internationale ajoute une contrainte supplémentaire, car les solutions mises en place doivent fonctionner dans des environnements variés, avec différentes configurations et réglementations.
Les centres d'examen sont situés en Europe, en Afrique, en Asie et en Amérique, ce qui

nécessite une robustesse particulière de l'application et une adaptabilité aux différentes infrastructures locales.

6.2. Exemples de tentatives de triche et solutions

Au cours du développement et des différentes phases de test, nous avons pu identifier et corriger plusieurs vecteurs de triche.

Tout d'abord, certains candidats tentaient d'utiliser les raccourcis systèmes de Windows, tels que Alt+Tab, Win+Tab ou encore Ctrl+Alt+Suppr, pour quitter l'application et accéder à d'autres contenus. Pour contrer cela, nous avons mis en place un script AutoHotKey, compilé en exécutable, qui bloque ces combinaisons de touches pendant toute la durée de l'examen.

Nous avons également constaté que certains candidats cherchaient à ouvrir les outils de développement du navigateur, dans le but d'accéder au code source ou de manipuler le DOM. Pour pallier ce risque, nous avons désactivé complètement les outils développeur via la configuration d'Electron, tout en bloquant les raccourcis clavier associés, comme F12 et Ctrl+Shift+I.

Enfin, nous avons détecté des tentatives de copier-coller de contenus préparés à l'avance. Pour y remédier, nous avons restreint l'accès au presse-papiers et mis en place une détection des opérations de collage suspectes grâce à l'intégration d'un keylogger.

6.3. Défis spécifiques sur macOS

La sécurisation sur macOS s'est révélée particulièrement complexe en raison des restrictions du système :

Certaines combinaisons de touches permettent encore aux utilisateurs de quitter l'application, même lorsqu'elle fonctionne en mode plein écran.

La principale difficulté vient du fait que, contrairement à Windows — où nous avons pu contourner ce problème avec AutoHotKey — le système macOS intercepte ces raccourcis clavier à un niveau supérieur, avant même que l'application ne puisse les traiter.

Pour tenter de résoudre cette contrainte, j'ai exploré plusieurs pistes. J'ai notamment testé des outils natifs comme Karabiner-Elements, envisagé le développement d'un agent système capable d'intercepter les événements clavier, et tenté de configurer des règles spécifiques via les préférences système.

Malheureusement, toutes ces solutions impliquent que l'utilisateur accorde des autorisations particulières dans les réglages d'Accessibilité de macOS. Or, dans le cadre d'un examen, il n'est pas envisageable de demander aux centres de modifier la configuration du système.

7. Processus de reprise et transfert de compétences

Un aspect particulièrement intéressant de ce projet a été le processus de reprise d'une application existante, développée par un prestataire externe qui cessait son activité. Cette transition a comporté plusieurs défis et a nécessité une méthodologie structurée.

7.1. Phase d'audit initial

La première étape essentielle du projet a été l'audit initial de l'application. Pendant deux jours, nous avons travaillé en étroite collaboration avec le développeur précédent, Nicolas, qui s'est déplacé dans nos locaux pour assurer un transfert de compétences. Cette phase a permis de poser les fondations techniques et organisationnelles indispensables à la bonne reprise du projet.

Nous avons commencé par rassembler l'ensemble de la documentation disponible. Cela comprenait les spécifications techniques, les guides d'utilisation, ainsi que les notes de développement accumulées au fil des différentes versions. Cette collecte s'est révélée précieuse pour reconstituer l'historique de l'application et mieux comprendre les choix d'architecture qui avaient été faits.

Dans un second temps, nous avons identifié les divers dépôts de code, listé les branches actives et documenté les environnements de déploiement existants. Cela nous a permis d'obtenir une vision claire et détaillée de l'état actuel du code et de son organisation.

L'audit a également porté sur l'analyse des processus techniques en place. Nous avons étudié et formalisé les procédures de compilation, de signature et de déploiement, tout en prenant en compte la complexité inhérente à la nature multiplateforme de l'application.

Enfin, nous avons organisé la transmission de l'ensemble des accès nécessaires au bon fonctionnement et à la maintenance de l'application, couvrant les serveurs, les dépôts de code, les certificats de signature, ainsi que les outils tiers utilisés.

Cette phase d'audit a été déterminante, non seulement pour bien comprendre les aspects techniques du projet, mais aussi pour en saisir le contexte d'utilisation, ainsi que les interactions avec les différents partenaires impliqués, tels que la CCI de Paris, Adimeo, et les centres d'examen.

7.2. Mise en place d’un environnement de développement robuste

Après avoir terminé l’audit, nous nous sommes attaqués à la mise en place d’un environnement de développement solide et cohérent, indispensable pour assurer la maintenance et faire évoluer l’application dans de bonnes conditions.

Nous avons commencé par mettre en place un environnement Docker. Cela a permis d’uniformiser les configurations entre les différents membres de l’équipe, tout en facilitant l’arrivée de nouveaux développeurs. En travaillant tous dans le même cadre, on a pu réduire au minimum les soucis liés aux différences de configuration sur les machines de chacun.

En parallèle, nous avons mis en place un pipeline d’intégration et de déploiement continu (CI/CD) sur GitLab. Grâce à ce processus automatisé, les étapes clés du développement — comme les tests, la compilation ou encore le déploiement de l’application — se font de manière fluide et systématique, que ce soit pour Windows, macOS ou Linux. Cela nous a permis de livrer des mises à jour plus rapidement et en toute fiabilité.

Pour nous assurer que le résultat soit à la hauteur dans des conditions proches de la réalité, nous avons aussi installé des environnements de test adaptés : une machine virtuelle sous Windows 10 et un Mac Mini pour les essais sur macOS. L’idée était de se rapprocher le plus possible de ce que rencontrent les centres d’examen au quotidien.

Enfin, nous avons pris soin de documenter l’ensemble des procédures dans un wiki interne. Ce guide couvre toutes les étapes nécessaires, du développement au déploiement. Il permet de sécuriser le projet sur le long terme, en évitant qu’il ne repose sur la seule mémoire de quelques personnes, et en facilitant la passation entre les membres de l’équipe.

7.3. Défis spécifiques de la reprise

La reprise du projet n’a pas été sans défis. Elle a demandé une bonne dose de réactivité et une coordination étroite entre tous les acteurs impliqués.

L’un des premiers obstacles auxquels nous avons été confrontés concernait la compréhension d’un code existant, dense et peu documenté. Certaines parties critiques — comme le keylogger ou le système de sécurité — ont demandé un vrai travail d’analyse pour en saisir le fonctionnement. Ce manque de clarté a forcément rallongé le temps nécessaire pour bien prendre en main le projet.

Un autre point important a été l'adaptation de l'application au nouveau portail développé par Adimeo. Cela a nécessité de revoir en profondeur la gestion des API et des flux de données pour assurer que les deux systèmes soient parfaitement compatibles. Cette phase d'intégration a entraîné une refonte partielle de plusieurs interactions techniques.

La question des certificats de signature a elle aussi demandé une attention particulière. Signer les exécutables — aussi bien sur Windows que sur macOS — implique des processus spécifiques à chaque plateforme. Il a fallu bien les comprendre et les intégrer correctement pour garantir des déploiements fiables et sans erreurs.

La coordination entre les différents partenaires a constitué un autre défi. Le projet impliquait plusieurs acteurs : téïcée, Adimeo, et la CCI de Paris. Chacun avait ses propres responsabilités et il a donc fallu veiller à maintenir une communication fluide et continue tout au long de cette phase de reprise.

Cette expérience a bien montré à quel point il est essentiel de disposer d'une documentation claire et d'une architecture logicielle bien structurée, surtout pour des applications qui doivent évoluer et être reprises par différentes équipes au fil du temps.

7.4. Leçons tirées du processus de reprise

Cette expérience de reprise d'un projet existant nous a apporté plusieurs enseignements précieux, qui ont clairement façonné notre façon d'aborder les projets par la suite.

Le premier constat évident : une documentation claire, complète et régulièrement mise à jour est absolument essentielle. Elle facilite énormément le transfert de compétences et permet à une nouvelle équipe de reprendre un projet sans perdre un temps fou à en décortiquer le fonctionnement.

Nous avons aussi vu les bénéfices d'une approche progressive. Plutôt que de tout vouloir refaire dès le début, nous avons choisi de faire évoluer l'application par étapes. Cela nous a permis de maintenir le service en continu, tout en améliorant petit à petit le code existant, sans risquer de créer des blocages ou des dysfonctionnements pour les utilisateurs.

Au final, nous avons compris qu'il ne suffit pas de maîtriser le code pour réussir la reprise d'un projet. Il est tout aussi important d'avoir une vision d'ensemble : bien comprendre l'environnement technique, les interactions avec d'autres systèmes et, surtout, les besoins des utilisateurs finaux. C'est cette compréhension globale qui permet de faire les bons choix et d'anticiper les impacts de chaque modification.

Ces leçons nous ont clairement influencés dans la manière de gérer nos projets suivants. Désormais, nous accordons beaucoup plus d'attention à la qualité de la documentation, à la modularité de l'architecture, et à la facilité de test des différents composants — autant d'éléments qui garantissent des solutions solides, durables et plus simples à maintenir.

8. Aspect international

Un aspect particulièrement intéressant du projet Paragraphe a été sa dimension internationale. L'application étant déployée dans des centres d'examen situés aux quatre coins du monde, nous avons dû adapter notre approche pour tenir compte de nombreuses particularités, tant culturelles que techniques.

Les contextes techniques variaient énormément d'un centre à l'autre. En Europe, beaucoup de centres disposaient de machines récentes et performantes. En revanche, dans certaines régions d'Afrique ou d'Asie, les équipements étaient plus anciens. Il était donc essentiel que l'application fonctionne correctement, quelles que soient les configurations matérielles, parfois très hétérogènes.

La qualité de la connexion internet était également un facteur à prendre en compte. Certains centres bénéficiaient d'un haut débit stable, mais d'autres devaient faire face à des connexions limitées, instables, voire à des coupures réseau pouvant durer plusieurs jours, voire semaines.

Pour pallier ces contraintes, nous avons conçu le logiciel de manière à ce qu'il télécharge les sujets lorsqu'une connexion était disponible. Les candidats pouvaient ensuite passer les tests en mode hors ligne, et les résultats étaient automatiquement envoyés dès que la connexion était rétablie.

Enfin, bien que Windows reste la plateforme la plus utilisée, certains centres travaillaient aussi sous macOS ou Linux, avec des versions parfois très variées. Cette diversité nous a confortés dans l'idée de développer une application légère, capable de tourner sur des machines peu puissantes, et suffisamment robuste pour fonctionner dans des conditions réseau loin d'être idéales.

Analyse des données d’examen et perspectives d’intelligence artificielle

Cette section explore comment les données générées par l’application Paragraphe pourraient être exploitées pour améliorer l’expérience d’examen et la détection de fraude, notamment grâce à l’utilisation de techniques d’intelligence artificielle.

1. Potentiel des données collectées

L’application Paragraphe collecte, par le biais du keylogger notamment, une quantité significative de données qui pourraient être valorisées.

1.1. Types de données disponibles

Au cours de chaque session d’examen, l’application enregistre plusieurs types de données.

On retrouve d’abord les données de frappe : la séquence des touches appuyées, les délais entre chaque frappe, ainsi que les habitudes de correction, qu’il s’agisse d’effacements ou d’ajouts.

Les interactions de l’utilisateur sont elles aussi suivies : déplacements de la souris, clics, changements de fenêtres ou de focus, et tentatives éventuelles d’accéder à des fonctionnalités bloquées.

S’ajoutent à cela des métadonnées liées à la session, comme la durée totale de l’examen, les pauses effectuées, le temps passé sur chaque question, ou encore les moments consacrés à la relecture.

Enfin, le contenu textuel produit par le candidat est conservé, non seulement sous sa forme finale, mais aussi dans ses versions intermédiaires, ce qui permet de suivre le cheminement de la rédaction.

Jusqu’à présent, ces données ont surtout été utilisées pour des raisons de sécurité ou pour permettre de restaurer les sessions en cas d’incident technique. Mais elles représentent un véritable gisement d’informations qui pourrait être exploité de façon bien plus riche à l’avenir.

1.2. Enjeux de confidentialité et conformité RGP

Bien entendu, l'exploitation de ces données soulève des questions majeures, notamment en ce qui concerne le respect de la vie privée et la conformité avec les réglementations en vigueur.

Avant toute chose, il est essentiel que les candidats soient informés de manière claire et transparente sur les informations qui sont collectées et sur l'usage qui pourrait en être fait. Rien ne doit se faire sans leur consentement explicite.

Lorsqu'on envisage d'exploiter ces données à plus grande échelle, l'anonymisation devient un point clé. Il faut s'assurer que les données soient traitées de façon à ce qu'aucun utilisateur ne puisse être identifié, afin de garantir le respect de leur anonymat.

Il est tout aussi important de veiller à ce que ces informations soient utilisées uniquement dans le cadre des finalités prévues et validées à l'avance. En aucun cas elles ne doivent être réutilisées à d'autres fins sans un accord clair.

Enfin, étant donné la sensibilité de ces données, leur traitement et leur stockage nécessitent des mesures de sécurité robustes, pour garantir leur protection contre tout risque de fuite ou d'usage malveillant.

Dans le projet Paragraphe, nous avons collaboré de manière étroite avec le délégué à la protection des données de la CCI Paris. Cet accompagnement nous a permis de nous assurer que nos pratiques respectent pleinement le RGPD ainsi que l'ensemble des réglementations applicables.

2. Applications potentielles de l'intelligence artificielle

Plusieurs applications de l'intelligence artificielle pourraient être envisagées pour exploiter ces données et améliorer l'expérience d'examen :

2.1. Détection avancée de fraude

Les techniques d'apprentissage automatique pourraient, à terme, grandement renforcer les capacités de détection des comportements suspects lors des examens.

Par exemple, un modèle bien entraîné serait capable de reconnaître le style de frappe habituel d'un candidat. Si, en pleine session, ce style change brutalement, cela pourrait signaler qu'une autre personne a pris le relais devant le clavier.

De la même façon, des algorithmes de comparaison de texte pourraient repérer des réponses étonnamment similaires entre différents candidats, ou entre un candidat et des contenus connus sur le web ou ailleurs.

Autre cas de figure intéressant : l'analyse des rythmes de saisie. Si l'on observe un arrêt net suivi de l'insertion soudaine d'un gros bloc de texte, cela pourrait indiquer qu'un copier-coller a eu lieu.

Enfin, en exploitant les données accumulées sur les sessions précédentes, il serait envisageable de développer des modèles capables de prédire quels examens présentent un risque plus élevé de fraude. Cela permettrait de concentrer les efforts de surveillance là où c'est vraiment utile.

2.2. Évaluation assistée

Pour certaines questions ouvertes, les technologies de traitement du langage naturel pourraient venir en appui aux évaluateurs.

Par exemple, pour les réponses courtes, il serait possible de mettre en place une pré-notation automatisée. En analysant la présence de concepts clés et la qualité de l'expression, le système pourrait proposer une première évaluation que l'examinateur n'aurait plus qu'à ajuster.

L'intelligence artificielle pourrait aussi aider à repérer des incohérences, qu'elles soient linguistiques ou conceptuelles, en mettant en lumière les réponses qui mériteraient un examen plus attentif.

Elle pourrait également fournir des indicateurs sur la complexité du langage utilisé : richesse du vocabulaire, structure des phrases, variété syntaxique, etc., offrant ainsi un éclairage supplémentaire sur le niveau de maîtrise du candidat.

Enfin, pour les questions plus ouvertes, l'IA pourrait regrouper automatiquement les réponses qui suivent des approches similaires. Cela permettrait aux évaluateurs de comparer plus facilement les travaux et de garantir une correction plus homogène.

Bien entendu, ces outils n'auraient pas vocation à remplacer les évaluateurs humains. Leur rôle serait uniquement de les assister, en leur faisant gagner du temps et en leur fournissant des indicateurs utiles — le jugement final devant toujours rester entre les mains des correcteurs.

3. Feuille de route pour l'intégration future de l'IA

En nous appuyant sur nos tests et sur la compréhension des besoins des utilisateurs, nous avons imaginé une feuille de route pour intégrer l'intelligence artificielle dans Paragraphe, de manière progressive.

Dans un premier temps, l'idée serait d'ajouter des outils simples de détection d'anomalies, qui viendraient aider les surveillants pendant les sessions. L'objectif ici serait surtout de limiter au maximum les faux positifs, pour ne pas alourdir inutilement leur travail.

Dans un second temps, à moyen terme, nous pourrions développer des outils d'analyse après examen. Ils permettraient d'identifier plus facilement les tentatives de fraude et de générer automatiquement des rapports destinés aux administrateurs.

Sur le long terme, nous envisageons aussi d'intégrer des fonctions d'évaluation assistée pour certaines questions, en restant bien entendu sous le contrôle des évaluateurs humains.

Enfin, à plus long terme encore, il serait intéressant d'explorer des systèmes plus adaptatifs, capables de personnaliser l'expérience d'apprentissage et d'évaluation en fonction du profil de chaque candidat.

Cette approche, par étapes, permettrait non seulement de renforcer progressivement la confiance dans ces outils d'IA, mais aussi de collecter davantage de données pour améliorer leur fiabilité et leur utilité au fil du temps.

4. Considérations éthiques et limitations

Le recours à l'intelligence artificielle dans le cadre de l'évaluation soulève naturellement de nombreuses questions éthiques, qu'il est essentiel de prendre en compte dès le départ.

D'abord, il faut garantir la transparence des algorithmes. Les candidats et les institutions doivent pouvoir comprendre comment fonctionnent les outils d'IA utilisés, et savoir quelles données sont prises en compte dans les analyses.

Il existe aussi un vrai risque de biais. Si les modèles sont entraînés sur des données imparfaites, ils risquent de reproduire, voire d'amplifier ces biais. C'est pourquoi il est crucial de prévoir des contrôles réguliers et des ajustements pour limiter ce type de dérive.

Autre point de vigilance : l'équité entre les candidats. L'introduction d'outils basés sur l'IA ne doit en aucun cas créer de désavantages pour ceux qui seraient moins à l'aise avec la technologie.

Il est tout aussi important de maintenir un contrôle humain significatif. Toute décision lourde de conséquences — par exemple, la détection d'une tentative de fraude ou l'attribution d'une note — doit impérativement être validée par un évaluateur humain.

Enfin, il faut garder à l'esprit que les technologies actuelles d'IA ont encore des limites, notamment pour saisir toutes les nuances linguistiques et culturelles. C'est particulièrement vrai lorsqu'on évalue des compétences en français des affaires, un domaine riche en subtilités.

Ces considérations éthiques doivent vraiment rester au cœur de toute démarche visant à intégrer l'IA dans un système d'évaluation comme Paragraphe.

Conclusion

1. Bilan de l'utilisation d'Electron

En travaillant sur le projet Paragraphe, nous avons eu l'occasion d'analyser en profondeur le framework Electron et de tirer plusieurs enseignements concrets sur son utilisation pour des applications d'examen sécurisé.

Dans le cas présent, Electron a su répondre aux besoins. Ce choix technologique a permis de conjuguer facilité de développement, souplesse des technologies web et compatibilité avec les principales plateformes. La migration de la version 12 vers la version 22 d'Electron a apporté une modernisation bienvenue, en renforçant la sécurité tout en conservant l'ensemble des fonctionnalités essentielles.

Plusieurs points positifs se sont démarqués. Le premier, évident : la rapidité de développement. Le recours à des technologies web standards a permis de mettre en œuvre de nouvelles fonctionnalités rapidement, là où une réécriture en code natif aurait nécessité bien plus de temps.

Autre atout notable : la facilité de maintenance. La structure du code rend le projet accessible pour différents intervenants. C'est un avantage non négligeable, surtout dans le cas d'un projet amené à changer de mainteneurs à plusieurs reprises.

La flexibilité offerte par Electron a également permis d'ajuster l'application aux nouvelles exigences de sécurité et aux évolutions du portail sans imposer de refonte majeure. Enfin, la compatibilité multiplateforme a largement simplifié le déploiement dans les centres d'examen fonctionnant sous Windows, macOS ou Linux.

Cela dit, certains inconvénients méritent d'être soulignés. Sur le plan des performances, malgré les efforts d'optimisation, l'application reste plus gourmande en ressources qu'un équivalent développé en natif. Ce point devient sensible sur des machines plus anciennes ou peu puissantes.

La question de la sécurité reste également un sujet d'attention permanente. Même avec des mesures renforcées, Electron présente des vulnérabilités structurelles qui imposent un suivi rigoureux et des mises à jour fréquentes.

Enfin, l'empreinte mémoire de l'application, alourdie par la présence de Chromium, peut représenter une contrainte supplémentaire pour certains environnements aux ressources limitées.

Malgré ces limites, le bilan global de l'usage d'Electron dans le cadre du projet Paragraphe reste très positif. Dans un contexte marqué par des contraintes budgétaires et des délais serrés, ses avantages ont clairement pris le dessus.

2. Perspectives

L'expérience acquise avec la refonte de Paragraphe ouvre plusieurs perspectives d'évolution, tant pour l'application elle-même que pour les choix technologiques futurs.

2.1. Évolutions envisageables pour Paragraphe

Il reste encore plusieurs axes sur lesquels Paragraphe pourrait évoluer dans les prochains mois.

La sécurité, bien entendu, restera un chantier prioritaire. Nous avons identifié plusieurs pistes concrètes à partir de l'analyse des tentatives de triche observées. Cela inclut par exemple le développement de mécanismes plus fins pour analyser les comportements pendant l'examen, ou encore la détection des périphériques non autorisés.

Côté performances, il serait également utile de continuer à alléger l'application. Mieux gérer l'empreinte mémoire, accélérer les temps de chargement… autant d'optimisations qui permettraient d'améliorer le confort d'utilisation, notamment sur des configurations matérielles plus limitées, encore courantes dans certains centres.

Enfin, le recours à l'intelligence artificielle commence à se dessiner. Il serait intéressant de tester des outils capables d'assister dans l'analyse des réponses ou de repérer automatiquement certains comportements suspects. Ce sera sans doute l'un des leviers d'amélioration à suivre de près.

2.2. Alternatives technologiques futures

Pour de futurs projets comparables — ou dans le cadre d'une refonte majeure de Paragraphe plusieurs alternatives à Electron mériteraient d'être explorées.

Le framework Tauri, par exemple, se démarque par son empreinte plus légère et de meilleures performances. Son architecture repose sur Rust et sur les webviews natives de chaque système, ce qui pourrait en faire un excellent candidat pour une future version de l'application, surtout si les contraintes en matière de performance deviennent plus marquées.

Flutter Desktop représente également une option intéressante. Sa maturité grandissante pour les applications de bureau, combinée à une expérience utilisateur fluide et cohérente sur plusieurs plateformes, en fait une solution à envisager sérieusement.

Enfin, le recours à des applications natives avec logique partagée offre aussi des perspectives prometteuses. Des outils comme React Native for Windows/macOS ou Kotlin Multiplatform permettent de développer des applications natives tout en mutualisant une large part du code métier entre les différentes plateformes. Une approche qui pourrait se révéler particulièrement pertinente pour certains cas d'usage.

2.3. Tendances et évolution du marché

Avec l'évolution constante des outils numériques, on observe plusieurs grandes tendances qui devraient peser sur le développement futur des solutions d'examen en ligne.

La question de la sécurité reste bien sûr au premier plan. Beaucoup d'établissements cherchent aujourd'hui des solutions capables d'assurer un contrôle fiable, en combinant outils logiciels et surveillance humaine pour éviter les failles.

Autre point clé : l'analyse des données issues des examens. On commence à exploiter ces informations non seulement pour mieux détecter les fraudes, mais aussi pour affiner l'évaluation des compétences. Cette approche devrait encore se développer.

Les attentes évoluent aussi du côté des modalités d'examen. Pouvoir proposer des formats souples — en centre, à distance, ou en mode hybride — devient clairement un atout dans un marché de plus en plus concurrentiel.

Enfin, la capacité des solutions à bien s'intégrer avec les plateformes existantes (LMS, outils pédagogiques…) fait aujourd'hui toute la différence. Les utilisateurs attendent des solutions qui s'adaptent à leur écosystème, et non l'inverse.

En résumé, même si le choix d'Electron s'est révélé pertinent pour la refonte actuelle de Paragraphe, le contexte technologique évolue vite. Pour que l'application reste performante et adaptée, il sera essentiel de suivre ces évolutions de près et de réévaluer régulièrement les choix techniques.

3. Enseignements personnels et professionnels

Ce projet de refonte a été, pour moi, une expérience particulièrement formatrice, aussi bien sur le plan technique que méthodologique. Il a largement contribué à enrichir mes compétences et à renforcer mon développement professionnel.

Sur le plan technique, mener cette migration m'a permis d'acquérir une maîtrise approfondie d'Electron. J'ai pu en comprendre l'architecture, les mécanismes internes, mais aussi des aspects plus complexes tels que la sécurisation, l'optimisation et la configuration multiplateforme. Par ailleurs, j'ai nettement renforcé mes connaissances en sécurité applicative : savoir identifier des tentatives de contournement, anticiper les vecteurs d'attaque, et mettre en place les protections adaptées sont désormais des compétences solides dans mon bagage.

Le développement multiplateforme a représenté un autre défi intéressant, en particulier avec les spécificités de macOS, notamment sur les aspects liés à la sécurité et à la gestion des événements. Enfin, la mise en place d'un pipeline d'intégration et de déploiement continu (CI/CD) a été l'occasion de me perfectionner sur des pratiques aujourd'hui essentielles dans les projets modernes.

Au-delà de la technique, ce projet m'a permis de progresser en gestion de projet. J'ai appris à mieux communiquer avec l'ensemble des parties prenantes, à bien comprendre leurs attentes et à les traduire en spécifications claires et réalisables. Face à la complexité du projet, j'ai également affiné mes capacités de planification, en structurant les tâches et en priorisant les développements en fonction de leur valeur pour le client.

Cette expérience m'a également fait mesurer toute l'importance de la documentation et du transfert de connaissances pour garantir la pérennité d'un projet. Elle m'a permis de renforcer ma capacité à identifier les risques en amont, à les évaluer et à mettre en place des stratégies adaptées pour les limiter.

D'un point de vue plus personnel, ce projet a été particulièrement enrichissant. Il m'a permis de développer mon autonomie, de gagner en confiance dans mes choix techniques et d'aiguiser ma capacité d'adaptation face aux nombreux défis imprévus rencontrés. Trouver des solutions efficaces a souvent nécessité de la créativité et de la persévérance, en particulier sur les aspects liés à la sécurité.

L'ensemble de ces apprentissages constitue aujourd'hui une base solide pour la suite de mon parcours. Ils me donnent une réelle confiance pour aborder des projets complexes, tant sur le plan technique qu'organisationnel. Ils ont également renforcé mon intérêt pour les technologies web modernes, en particulier dans des contextes exigeants sur le plan de la sécurité.

Enfin, cette expérience m'a sensibilisé à l'importance de trouver un équilibre entre des contraintes souvent contradictoires : sécurité et expérience utilisateur, performance et délais de développement, innovation et stabilité. Cette capacité à naviguer entre ces exigences sera, je le crois, un atout majeur pour la suite de ma carrière.

Bibliographie

- Tauri vs Electron : https://www.levminer.com/blog/tauri-vs-electron
- Documentation Electron : https://www.electronjs.org/
- Intéropérabilité Electron : https://www.electronjs.org/docs/latest/tutorial/ipc
- Wikipedia Electron : https://fr.wikipedia.org/wiki/Electron_(framework)
- Documents internes à l'entreprise
- Comptes rendus de réunions
- Documentation projet
- NodeJS vs Go vs Rust : https://dev.to/hamzakhan/rust-vs-nodejs-vs-go-Performance-comparison-for-backend-development-2g69

Annexes

Grille d'évaluation de la pratique professionnelle

Cette fiche est à inclure dans les annexes du mémoire

Candidat : Milan HOMMET

Entreprise : téïcée

Tuteur : Clément RIVIERE

Critères	Non acquis	Partiellement acquis	Acquis	Dépassé	Total
	0	1	2	3	
Savoir-faire					/ 60
Atteinte des objectifs			X		2
Méthodes d'organisation			X		2
Utilisation des procédures internes				X	3
Capacité à travailler en équipe				X	3
Prise d'initiatives			X		2
Rigueur			X		2
Sens des responsabilités			X		2
Capacité de prise de recul			X		2
Maîtrise de l'écrit				X	3
Expression orale			X		2
Aptitude à mener à bien une mission			X		2
Aptitude à gérer son temps			X		2
Capacité à identifier ses priorités			X		2
Aptitude à intégrer des contraintes			X		2
Notion de hiérarchie				X	3
Notion d'objectif individuel et collectif				X	3
Identification des problèmes potentiels			X		2
Aptitude à manager un groupe de travail		X			1
Aptitude au contact téléphonique		X			1
Aptitude au relationnel face à face			X		2

Savoir-être					/ 30
Conscience professionnelle				X	3
Dynamisme			X		2
Sociabilité				X	3
Ponctualité / assiduité				X	3
Maturité				X	3
Ténacité				X	3
Réactivité				X	3
Autonomie d'apprentissage				X	3
Intérêt porté à l'entreprise				X	3
Adaptabilité à la culture d'entreprise				X	3
				Total	72 / 90
				Total	16 / 20

Appréciation du tuteur professionnel :

Milan a toujours fait preuve d'un grand sérieux et d'un professionnalisme constant tout au long de son alternance. Très bien intégré à l'équipe, il s'est montré fiable, respectueux et investi dans les missions confiées. Comme nous l'avons déjà évoqué ensemble, l'amélioration de ses capacités d'analyse lui permettra de gagner en autonomie sur des problématiques plus complexes.

Date : 10 / 06 / 2025

Signature et cachet de l'entreprise :

téicée®
la raison de votre confiance
Tél. 02 72 34 13 20
Quartier Kœnig
153, rue Géraldine MOCK
14760 BRETTEVILLE SUR ODON
SARL au capital de 15 000 euros
RCS : CAEN 500 145 156
APE : 6201Z
www.teicee.com