Myriam Assam 2047939

Technologies émergentes

420 5G4 MO, groupe 01

FRAMEWORK ELECTRON

Analyse détaillée du logiciel

Travail présenté a

Sylvain Labranche

Département des Techniques de l’informatique

Collège Montmorency

16 décembre 2024

1. **Introduction du logiciel Electron**

Le logiciel Electron est un environnement qui permet de développer des applications multi-plateformes de bureau avec JavaScript, HTML et CSS. Il a été créé en 2013. L’infrastructure est codée en Node.js et l’interface est bâtie sur les outils Chromimum (partie open source de Google Chrome). La dernière version est 32.0.0 et elle est sortie le 14 octobre 2024. C’est un logiciel libre open source développé par Git Hub. Les logiciels et applications célèbres qui ont été développé à l’aide d’Electron sont les éditeurs de texte libres Atom, Visual Studio Code et l’application Discord :

« Since its release, the Electron framework has won the hearts of all web and desktop developers. Many popular applications have been built with Electron that you may have used in the past or are using daily, such as Skype, Slack, WhatsApp, Discord, Signal, Visual Studio Code, Microsoft Teams, Keybase, and many others. »[[1]](#footnote-1)

1. **Fonctionnement du logiciel**

Les navigateurs web exercent de nombreuses responsabilités secondaires, telles que la gestion de plusieurs fenêtres (ou onglets) et le chargement d’extensions tierces. Avant, les navigateurs utilisaient généralement un processus unique pour toutes ces fonctionnalités. C’est moins de frais, mais quand un site Web plantait ou se bloquait, cela pouvait affecter l’ensemble du navigateur. Pour résoudre ce problème, Chrome a décidé que chaque onglet serait rendu dans son propre processus. Cela limite les dommages que le code bogué ou malveillant d’une page Web pourrait causer à l’application. Un seul processus de navigateur contrôle alors ces processus, ainsi que le cycle de vie de l’application dans son ensemble. Les applications Electron sont structurées de manière très similaire. En tant que développeur on contrôle deux types de processus. Le **«** MainProcess **»** (processus de principal) contrôlele cycle de vie del’application et le **«** Renderer Process **»** (processus de rendu distinct)gèrel’interface utilisateur.

**2.1 Processus principal**

L’objectif du processus principal est de créer et de gérer les fenêtres avec le module **«** BrowserWindow **»**. Chaque instance de la classe **«** BrowserWindow **»** crée une fenêtre d’application qui charge une page Web dans un processus de rendu distinct. On peut interagir avec ce contenu Web à partir du processus principal a l’aide de l’objet **«** webContents ». Le processus contrôle également le cycle de vie de l’application avec le module **«** app ». Le **«** MainProcess **»** ajoute également des API pour interagir avec les systèmes d’exploitation de l’utilisateur. Electron a de plus, plusieurs modules qui contrôlent les fonctionnalités natives du bureau, comme les menus, les boite de dialogue et les icones de la barre d’état.

**2.2 Processus de rendu distinct**

Le processus de rendu distinct est responsable du rendu du contenu web. Le code exécuté dans les processus de rendu doit se comporter conformément aux normes web. Les interfaces utilisateur et fonctionnalités d’application au sein d’une seule fenêtre de navigateur doivent être écrites avec les mêmes outils qu’on utilise sur le web. Certaines spécification web sont : Un fichier HTML est le point d’entrée pour le processus de rendu, le style de l’interface utilisateur est ajouté via des fichiers CSS et du code JavaScript exécutable peut être ajouté. Il n’y a pas de moyen d’importer les scripts de contenu d’Electron, donc on utilise les scripts de préchargement. Ils contiennent du code qui s’exécute dans un processus de rendu avant que son contenu Web ne commence à se charger. Ils s’exécutent dans le contexte du moteur de rendu. Ils ont accès aux API Node.js. Le script sert à améliorer le moteur de rendu. Ils exposent des API arbitraires et le contenu Web peut ensuite les consommer.

1. **Installation**

Pour l’installation, on doit installer Node.js. Il faut créer un dossier et initialiser un package npm. Ensuite, pour exécuter Electron, il faut activer la commande npm start. Electron s’exécuter à la racine du projet. Lors de l’exécution, Electron va chercher script **«** main». C’est ce script qui contrôle le processus principal.Il faut donc initialiser le script **«** main». Il faut créer un fichier vide nommé **«** main.js » dans le dossier racine du projet.

1. **Création d’une application simple**

On va commencer par créer une application qui juste afficher **«** hello world ». Il faut commencer par créer un nouveau dossier (par exemple **«** ma-premiere-app **»**). On peut utiliser la commande **«** mkdir ma-premiere-app **»**. Ensuite on navigue vers celui-ci avec la commande **«** cd ma-premiere-app **»**. On initialise le projet avec **«** npm init **»**. Il y’aura des questions posées et il faut le répondre. Plus tard, on peut utiliser la commande **«** npm init -y **»** pour que les anciennes valeurs soient utilisées. Cet outil va générer un nouveau fichier appelé **«** package.json **»**. Le fichier **«** main **»** du projet sera **«** index.js **»**. On doit d’abord exécuter cette commande **«** npm i -D electron ». Il faut créer maintenant le fichier **«** index.js **»** dans le dossier racine du projet. On doit d’abord importer les objets et classes qu’on a besoin. Il faut référencer à un objet ou au type **«** BrowserWindow». On attend ensuite que l’application soit prête et crée une petite fenêtre. Après le chargement, ça doit afficher le contenu du fichier **«** index.html ». On doit définir le contenu principal dans le format d’une page HTML. Il faut créer un fichier index.html a côté du fichier **«** package.json » et **«** index.js ». On insère **«** Hello World! » dans le fichier HTML. Il faut mettre à jour les scripts. Ajoutez l’entrée start dans le fichier package.json. Pour lancer l’application il reste juste à exécuter la commande **«** npm start ». Pour l’arreter il faut faire Ctrl + C.

**4.1 Faire fonctionner l’application dans plusieurs plateformes : macOs**

Pour lancer l’application dans plusieurs plateformes. On va utiliser electron-builder pour faire cela. On l’installe avec la commande **«** npm i -D electron-builder ». On va faire un exemple avec macOS. Pour macOS, si on veut publier notre application dans l’App Store, il faut mettre un ID pour l’application dans le fichier **«** package.json ». Il faut mettre aussi l’entrée build : macos. Pour lancer l’application, on peut exécuter **«** npm run build : macos ». Après quelques secondes dans le dossier dist/mac on peut voir l’application, Il faut juste faire un double-clique qui va lancer l’application. Il reste juste à ajouter un dernier script nécessaire. Il sert pour l’empaquetage de l’application. Après avoir lancé le script, il y’aura différent fichier dans le dossier dist. Il y’a le fichier **«** ma-premiere-app-1.0.0.dmg » qui est un **«** installer **»** pour macOS. Le fichier .zip et le fichier de l’application qui est dans le dossier mac. Après avoir lancer le fichier .dmg, on devrait voir l’ «installer **»** macOS.

**4.2 Faire fonctionner l’application dans plusieurs plateformes : Windows**

Pour Windows, les scripts sont: « "build:windows": "electron-builder --win --dir" **»** et « "dist:windows": "electron-builder --win" ». On ouvre le temrnial et execute cette commande « npm run build:windows ». On clicque ensuite sur le fichier .exe pour lancer l’application. On doit de plus utiliser build:windows pour créer un package de distribution pour les tests. On exécute la commande « npm run dist:windows » et on peut voir dans le dossier dist l’ «installer » .exe et le dossier win-unpacked. On clique deux fois sur le fichier installer et l’application devrait fonctionner.

**4.3 Faire fonctionner l’application dans plusieurs plateformes : Ubuntu**

On commence par mettre une section **«** linux **»** dans le fichier package.json. On peut mettre des paramètres pour plusieurs plateformes en même temps ce qui est utile quand on veut utiliser plusieurs plateformes. Comme avec Windows, on met les scripts **«** "build:linux": "electron-builder --linux --dir" » et **«** "dist:linux": "electron-builder --linux" ». On lance la commande **«** npm run build» dans le terminal. Dans le dossier racine, il devrait y avoir linux-unpacked qui contient plusieurs artefacts **«** build ». Maintnant pour construire des packages pour la distribution, on exécute la commande **«** npm run dist : linux ». Il y’auras plusieurs autres packages. Il y a un fichier de fichier AppImage qui est un format universel pour GNU/Linux, Il y a aussi un fichier snap pour les applications sandbox. Dans le dossier linux-unpacked il y a le fichier ma-premiere-app qui est un **«** build » pour les tests et les distributions. Il faut cliquer deux fois sur le fichier AppImage pour lancer l’application. Normalement, l’application devrait maintenant marcher dans Linux.

**5. Sécurité Dans Electron**

Les risques de sécurité sont généralement faibles, mais l'utilisation d'un environnement qui combine des technologies web (HTML, CSS, JavaScript) avec l'accès à des fonctionnalités systèmes via Node.js peut introduire des vulnérabilités si certaines pratiques ne sont pas suivies. Voici une analyse plus détaillée des risques et des mesures de sécurité à mettre en place :

**5.1 Périls liés aux privilèges supplémentaires**

Les risques de sécurité augmentent avec les pouvoirs supplémentaires accordés au code, tels que l'accès direct au système de fichiers ou aux processus via Node.js. Cette combinaison de privilèges expose l'application à des attaques potentielles, telles que l’exécution de code arbitraire. Coline Mignot dit dans son article publié dans Ippon positive technology :

« Le développement via Electron enjoint à être plus prudent d’un point de vue sécurité, car Electron n’est pas un navigateur web. Ici, on peut accéder au système de fichiers, au shell utilisateur, etc. Il est donc important de veiller à ce que votre application n’exécute pas du code d’une source externe non fiable ».[[2]](#footnote-2)

Il peut aussi avoir des attaques de type Cross-Site Scripting (XSS) : L'utilisation d'une interface utilisateur basée sur HTML expose l'application aux vulnérabilités XSS.

**5.2 Nécessité de mises à jour régulières**

Les vulnérabilités critiques sont normalement corrigées quand on met à jour le Framework vers la dernière version. Les vulnérabilités critiques identifiées sont régulièrement corrigées dans les nouvelles versions. Cela inclut, les mises à jour de Chromium, qui réparent les failles du moteur web et les correctifs de Node.js, qui comblent les vulnérabilités au niveau du back-end.

**5.3 Sécurité des bibliothèques**

L'installation de bibliothèques externes peut introduire des risques supplémentaires. Il est crucial de faire attention aux bibliothèques qu’on installe. Il faut donc vérifier les sources des bibliothèques pour s’assurer qu’elles viennent d’auteurs fiables. De plus il faut aussi examiner régulièrement les dépendances pour identifier les versions obsolètes ou vulnérables. On peut utiliser des outils comme npm audit pour détecter les failles dans les modules installés.

**5.4 Bonnes pratiques de configuration**

Pour réduire les risques, il est recommandé de suivre ces bonnes pratiques de configuration. En premier lieu, on peut désactiver « Node Integration ». L'intégration de Node.js dans les processus Renderer est une source majeure de vulnérabilités. La désactiver empêche l'exécution de commandes Node.js dans l'interface utilisateur. En deuxième lieu, activer le « Context Isolation » : Cette fonctionnalité empêche les scripts malveillants de modifier des objets globaux partagés entre le Renderer et le Main Process. En troisième lieu, utilisez un « Content Security Policy (CSP) » : Le CSP définit les sources autorisées pour les scripts et les ressources, limitant ainsi les risques d’exploitation. En dernier lieu, restreindre les permissions. Limitez les privilèges accordés à chaque processus Renderer. Par exemple, évitez d’autoriser l'accès au système de fichiers sauf si cela est absolument nécessaire.

**5.5 Communication sécurisée entre processus**

L’interaction entre le Main Process et les Renderer Processes via l’API IPC doit être soigneusement contrôlée. Validez toutes les données échangées entre les processus. Utilisez des canaux sécurisés et évitez d’exposer des API sensibles via IPC.

**5.6 Gestion des ressources tierces**

Si votre application charge du contenu web (par exemple, des API tierces ou des URL externes), il faut s’assurer d'utiliser des connexions HTTPS pour garantir la confidentialité et l'intégrité des données. C’est important aussi de valider et de filtrer le contenu avant de l'afficher dans une « WebView » ou un « iframe ». Finalement, il est utile de mettre en place des audits réguliers pour s'assurer que l'application respecte les normes de sécurité. C’est-à-dire utiliser des outils comme « Snyk » ou « OWASP Dependency-Check » pour identifier les vulnérabilités potentielles. C’est important d’effectuer des tests pénétration pour détecter les failles exploitables.

1. **Avantages et inconvénients**

Ce logiciel utilise les langages pour la programmation web. Ce qui est plus facile pour les personnes qui maitrises seulement HTML, CSS et JavaScript. Un autre avantage, c’est que ce Framework peut être utilisé dans plusieurs plateformes. Les applications peuvent fonctionner dans Windows, Linux et Mac. Pour les inconvénients, Les applications qu’on crées sont lourdes à cause de Chromimum et Node. Elles peuvent contenir beaucoup de package et de dépendances en plus. Une application Electron peut dépasser les 200-300 Mo. Elles prennent de plus beaucoup de mémoire.

**6.1 Comparaison avec Flutter**

Si on compare Electron avec Flutter par exemple, les deux sont multi-plateformes et open-source. Les deux sont rapides, mais Electron, mais il faut plus de configurations à appliquer. C’est mieux d’utiliser Electron pour faire de applications de bureau et Flutter pour les applications mobiles. Les applications Flutter utilisent moins de mémoire.

**6.2 Comparaison avec tauri**

Maintenant, si on veut comparer avec Tauri. On peut utiliser n’importe framework front-end qui compile avec HTML, JavaScript et CSS. Levente Lőrik, un ingénieur dit dans son blog :

« Electron is being replaced? Yes, Tauri is way better, but it still misses a lot. In a couple of years, I'm sure the Tauri team will catch app to Electron. »[[3]](#footnote-3)

Le developpeur que j’ai contacté sur Reddit Anupam Chakrawarti explique :

« There are alternatives to Electron like Tauri which is Rust based also has a port for JS. It uses the existing WebView for the respective platforms (Google it). The binary file is small, and the memory management is also good. »[[4]](#footnote-4)

On peut voir qu’Electron peut être utile dans plusieurs cas, mais il y a d’autres framework plus performants et moins lourds qu’Electron qui peuvent utile dans d’autres cas. Si on ne connait que les languages de programmation JavaScript, HTML et CSS, Electron est ce qu’il faut choisir, sinon Tauri si on connait Rust.

**7. Applications crées avec Electron**

Plusieurs applications ont été créés avec ce Framework. C’est le cas de Visual Studio code, Skype, Discord, Slack et plus. Visual Studio Code est l’application la plus réussite faite avec Electron. Elle est multiplateforme et utilise des technologies comme HTML, CSS et JavaScript qui sont familière. De plus, il y a plusieurs extensions qui peuvent être utilisé facilement. Le problème est que l’application utilise beaucoup de mémoire, car c’est fait avec Electron. C’est la même chose avec Slack qui est un outil de communication d’équipe. L’interface ressemble à celle de Discord.

**8. Amélioration de la performance**

On a vu donc qu’une application Electron peut être assez lourde. Pour garantir une meilleure expérience, on peut améliorer la performance d’une application Electron avec quelques pratiques bonnes pratiques.

**8.1 Examiner les modules**

Une première bonne pratique est d’examiner un module Node.js avant de l’installer dans l’application. Il faut examiner le nombre de dépendances que ce module inclut, les ressources nécessaires pour le charger avec require() et les ressources nécessaires pour effectuer ce qu’on veut.

* 1. **Prioriser les opérations importantes.**

Au lieu terminer les opérations tout de suite après le démarrage de l’application. Il ne faut pas non plus lancer des opérations qui ne sont pas essentielles trop rapidement. Si on prend pour exemple Visual Studio Code, quand on ouvre un fichier, le code n’est pas mis en surbrillance. Il priorise notre habileté à interagir avec le texte. Après cela, le code est mis en couleur. C’est préférable de charger les ressources uniquement quand c’est nécessaire.

* 1. **Bloquer le processus principal et le processus Renderer**

Il ne faut absolument pas bloquer le processus principal. C’est lui qui s’occupe les fenêtres, les interactions et la communication entre les différents components dans l’application. Il ne faut non plus bloquer le processus renderer qui gère l’interface utilisateur.

**9. Conclusion**

Pour conclure, Electron est un très bon outil pour développer des applications de bureau. On a pu voir que certaines personnes apprécient ce logiciel et d’autre préfère d’autres alternatives. Les causes sont que les applications Electron utilisent beaucoup de mémoire et il y a des inconvénients en termes de performances comparativement à Tauri. Cependant, il utilise des languages web comme HTML, CSS et JavaScript pour faire des applications ce qui peut être utile pour ceux qui connaissent ses langages seulement. En plus, les applications peuvent êtres fonctionnels dans plusieurs plateformes comme MacOS, Linux et Windows.

**Annexe : veille technologique**

1. **Professionnel humain**

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

**Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police, document

Description générée automatiquement**

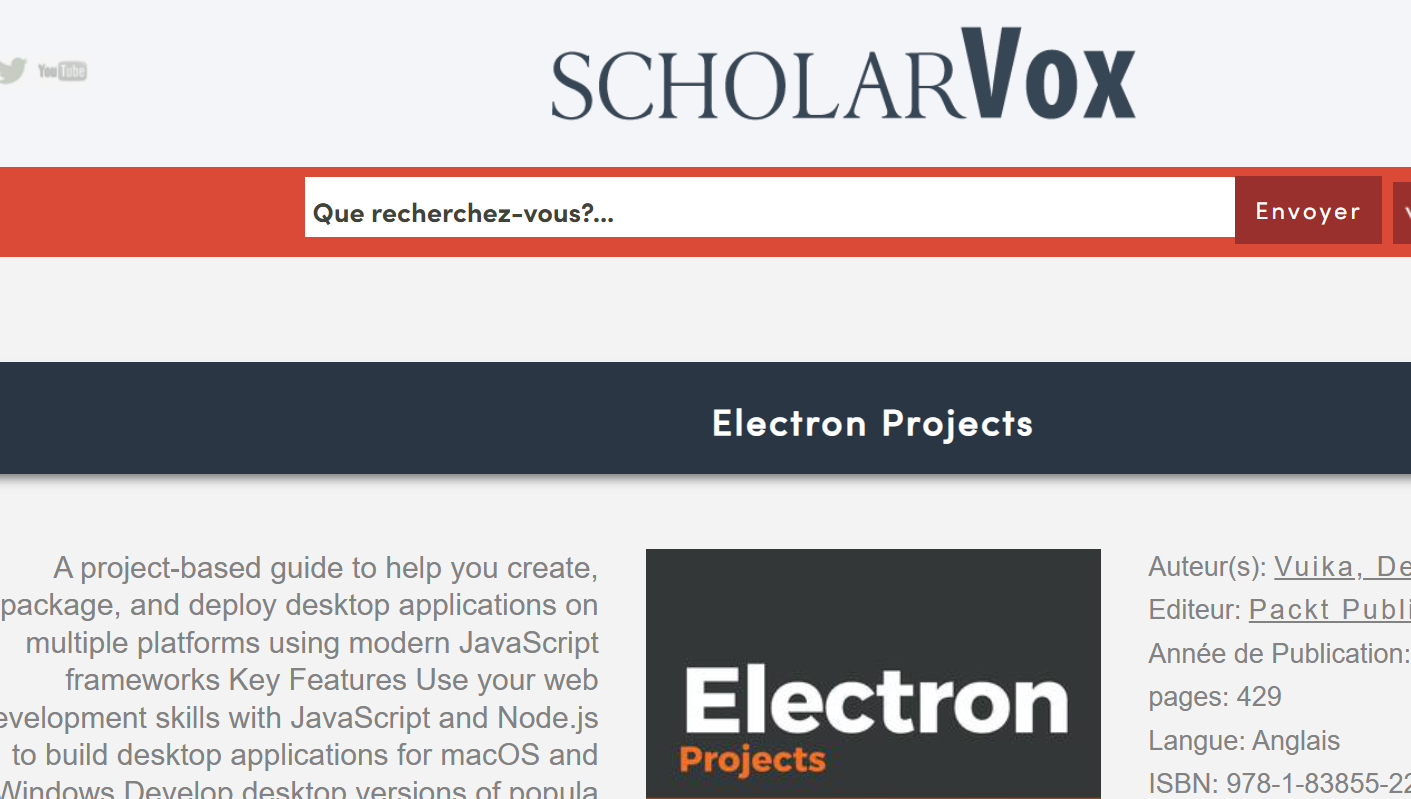
**J’ai fait un poste sur une communauté reddit r/Electronjs et des développeurs qui ont de l’experience avec ce logiciel m’ont répondu. Le compte CURVX est un developpeur Frontend et son nom est Anupam Chakrawarti Je l’ai cité dans mon travail.**

**Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement**

1. **Moteurs de recherche spécialisés**

**ScholarVox :**

****

**Je l’ai trouvé dans la section catalogue bibliothèque de l’école et j’ai trouvé un livre utile pour ce projet.**

**Semantic scholar : Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement**

**J’ai utilisé Semantic Scholar pour trouver des articles et d’autres informations pertinentes.**

**WorldCat: Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement**

**Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement**

**J’ai aussi essayé WorldCat, mais je n’ai trouvé grand-chose.**

**Google Scholar :**

**Une image contenant texte, capture d’écran, Police, Page web

Description générée automatiquement**

**J’ai aussi utilisé Google Scholar pour d’autres ouvrages académiques.**

1. **Suivi de mots-clés avec Google Alertes**

**Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement**

**Les mots clés utilisés sont Electron updates, Cross-platform apps Electron, Electron Node.js, Electron security et Electron framework pour avoir les mises à jour et nouveautés de ce logiciel.**

1. **Flux RSS**

**Une image contenant texte, logiciel, Page web, Icône d’ordinateur

Description générée automatiquement**

**J’ai utilisé Freedly pour les Flux RSS avec Electron js, Smashing magazine et Dev.to.**

**Une image contenant texte, logiciel, capture d’écran, Page web

Description générée automatiquement**

**J’ai aussi utilisé Inoreader, car je n’étais pas sur si Feedly marchait. Je me suis abonnée au r/electronjs de Reddit, le GitHub officiel d’Electron et d’autres articles et blogs qui parle/ont parlé d’Electron.**

1. **Médias sociaux**

**Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement**

**Je me suis abonnée sur twitter au compte officiel Electron.**

**Une image contenant texte, capture d’écran, logiciel, multimédia

Description générée automatiquement**

**C’est le compte de Node.js. Surveiller des nouveautés sur Node.js qui est utilisé sur Electron.**

**Une image contenant texte, capture d’écran, logiciel, multimédia

Description générée automatiquement**

**Frontend Masters : c’est un compte de cours et d’informations crées par des experts**

1. **Surveillance de pages Web**

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

**Node.js Blog**

Une image contenant texte, logiciel, Page web, Site web

Description générée automatiquement

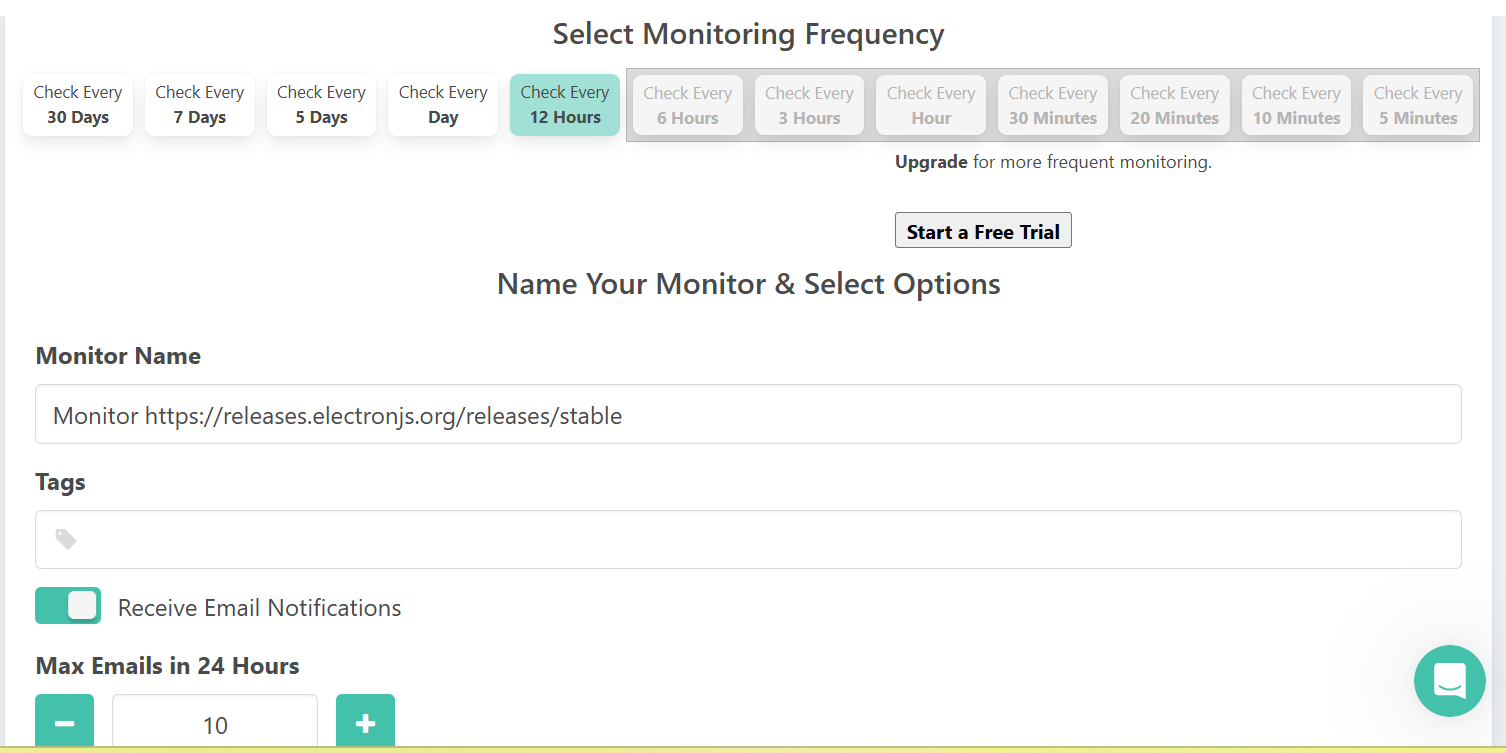
Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

**Electron Release Notes**

**Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement**

****

**GitHub Electron Issues**

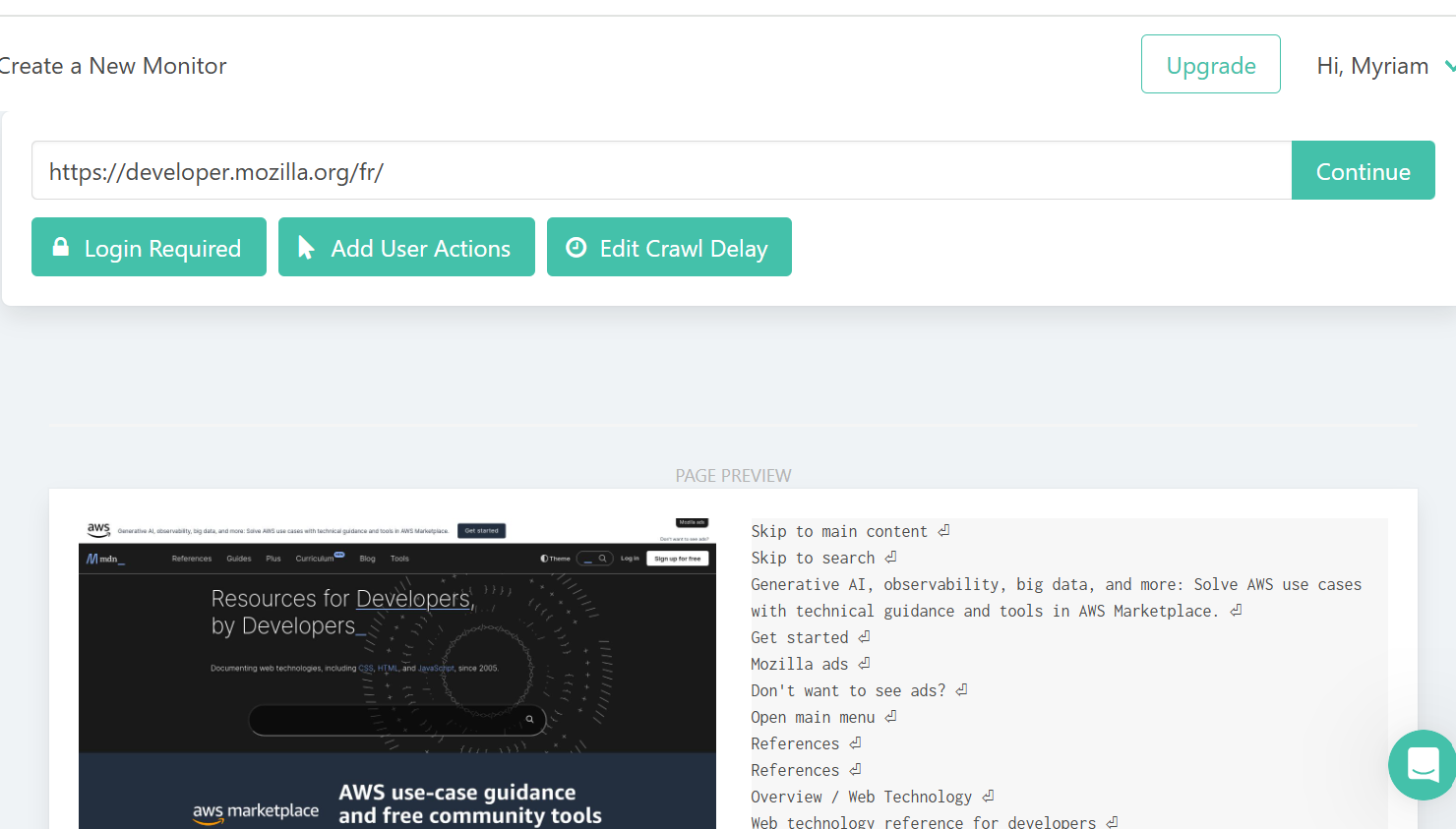
**Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement**

**Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement**

**Tutorials Electron**

****

**Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement**

**MDN Web Docs**

**Bibliographie**

ELECTRONJS, « Documentation officielle Electron », 2024, <https://www.electronjs.org/docs/latest>, consulté le 14 décembre 2024.

WIKIPEDIA, « Electron (framework) », Wikipédia, <https://fr.wikipedia.org/wiki/Electron_(framework)>, consulté le 14 décembre 2024.

FREELANCEREPUBLIK, « Electron, ou comment faire du logiciel avec du JavaScript », Talks Freelancerepublik, <https://talks.freelancerepublik.com/electron-javascript-creer-apps-desktop/#:~:text=Comme%20nous%20l'avons%20vu>, consulté le 14 décembre 2024.

VUIKA, Denys, *Electron Projects*, Packt Publishing, 2019, 429 p., <https://cyberlibris-montmorency.proxy.collecto.ca/catalog/book/88876732>, consulté le 14 décembre 2024.  
MIGNOT CAROLINE, « Retour d’expérience sur Electron », <https://blog.ippon.fr/2022/02/14/retour-dexperience-sur-electron/>, consulté le 16 décembre 2024.

ANONYME, « Flutter vs Electron: What’s the Difference? (Comparison Guide 2023) », <https://pieces.app/blog/flutter-vs-electron-whats-the-difference>, consulté le 16 décembre 2024.

LORIK LEVENTE, « Tauri VS. Electron - Real world application », <https://www.levminer.com/blog/tauri-vs-electron>, consulté le 14 décembre 2024.  
CHAKRAWARTI ANUPAM, « Réponse à une question dans le subreddit r/electronjs », Reddit, [Electron: building desktop apps with web technology](https://www.reddit.com/r/electronjs/), consulté le 14 décembre 2024.

Electron, Twitter, <https://x.com/electronjs>, consulté le 14 décembre 2024.

Node.js, Twitter, <https://x.com/nodejs>, consulté le 14 décembre 2024.

FrontEnd Masters, Twitter, <https://x.com/FrontendMasters>, consulté le 14 décembre 2024.

1. « Depuis sa sortie, le framework Electron a conquis le cœur de tous les développeurs web et desktop. De nombreuses applications populaires ont été construites avec Electron que vous avez peut-être utilisé dans le passé ou que vous utilisez quotidiennement, telles que Skype, Slack, WhatsApp, Discord, Signal, Visual Studio Code, Microsoft Teams, Keybase et bien d’autres. » [notre traduction]. VUIKA, Denys, *Electron Projects*, Packt Publishing, 2019, p.7 [↑](#footnote-ref-1)
2. MIGNOT, Coline, « Retour d’expérience sur Electron », *Ippon positive technology,* 14 avril 2022, <https://blog.ippon.fr/2022/02/14/retour-dexperience-sur-electron/>, consulté le 14 décembre 2024 [↑](#footnote-ref-2)
3. « Electron est-il en train d’être remplacé ? Oui, Tauri est bien meilleur, mais il lui manque encore beaucoup de choses. Dans quelques années, je suis sûr que l’équipe de Tauri rattrapera l’application d’Electron ». [notre traduction]. LORIK, Levente, « Tauri VS. Electron - Real world application », *levminer,* 22 août 2022, <https://www.levminer.com/blog/tauri-vs-electron>, consulté le 14 décembre 2024 [↑](#footnote-ref-3)
4. « Il existe des alternatives à Electron comme Tauri qui est basé sur Rust et qui dispose également d’un port pour JS. Il utilise le WebView existant pour les plates-formes respectives (chercher sur Google). Le fichier binaire est petit et la gestion de la mémoire est également bonne » [notre traduction]. CHAKRAWARTI, Anupam, « poste nommé project », *Commentaire sur le subreddit r/electronjs,* 14 décembre 2024, [Electron: building desktop apps with web technology](https://www.reddit.com/r/electronjs/), consulté le 14 décembre 2024 [↑](#footnote-ref-4)