# *Get Started*

# *Bắt đầu*

## *Introduction*

## *Giới thiệu*

*What is Electron?*

Electron là gì?

*Electron is a framework for building desktop applications using JavaScript, HTML, and CSS. By embedding Chromium and Node.js into its binary, Electron allows you to maintain one JavaScript codebase and create cross-platform apps that work on Windows, macOS, and Linux — no native development experience required.*

Electron là một khung làm việc (framework) để phát triển ứng dụng desktop bằng JavaScript, HTML và CSS. Bằng cách tích hợp Chromium và Node.js vào tệp nhị phân của mình, Electron cho phép bạn duy trì một mã nguồn JavaScript duy nhất và tạo ra các ứng dụng đa nền tảng hoạt động trên Windows, macOS và Linux — không yêu cầu kinh nghiệm phát triển bản địa.

*Getting started*

Bắt đầu

*We recommend you to start with the tutorial, which guides you through the process of developing an Electron app and distributing it to users. The examples and API documentation are also good places to browse around and discover new things.*

Chúng tôi khuyên bạn nên bắt đầu với hướng dẫn, giúp bạn thực hiện quy trình phát triển ứng dụng Electron và phân phối nó cho người dùng. Các ví dụ và tài liệu API cũng là những nguồn tài liệu hữu ích để bạn tham khảo và khám phá những điều mới mẻ.

*Running examples with Electron Fiddle*

Chạy các ví dụ với Electron Fiddle

*Electron Fiddle is a sandbox app written with Electron and supported by Electron's maintainers. We highly recommend installing it as a learning tool to experiment with Electron's APIs or to prototype features during development.*

Electron Fiddle là một ứng dụng sandbox được phát triển bằng Electron và được hỗ trợ bởi các nhà phát triển chính thức của Electron. Chúng tôi khuyến khích bạn cài đặt ứng dụng này như một công cụ học tập để thử nghiệm các API của Electron hoặc để tạo mẫu các tính năng trong quá trình phát triển.

*Fiddle also integrates nicely with our documentation. When browsing through examples in our tutorials, you'll frequently see an "Open in Electron Fiddle" button underneath a code block.*

Fiddle cũng tích hợp mượt mà với tài liệu của chúng tôi. Khi duyệt qua các ví dụ trong các hướng dẫn của chúng tôi, bạn sẽ thường thấy nút "Mở trong Electron Fiddle" bên dưới một khối mã.

*If you have Fiddle installed, this button will open a fiddle.electronjs.org link that will automatically load the example into Fiddle, no copy-pasting required.*

Nếu bạn đã cài đặt Fiddle, nút này sẽ mở liên kết fiddle.electronjs.org, tự động tải ví dụ vào Fiddle mà không cần sao chép và dán.

// main.js

const { app, BrowserWindow } = require('electron/main')

const path = require('node:path')

function createWindow () {

const win = new BrowserWindow({

width: 800,

height: 600,

webPreferences: {

preload: path.join(\_\_dirname, 'preload.js')

}

})

win.loadFile('index.html')

}

app.whenReady().then(() => {

createWindow()

app.on('activate', () => {

if (BrowserWindow.getAllWindows().length === 0) {

createWindow()

}

})

})

app.on('window-all-closed', () => {

if (process.platform !== 'darwin') {

app.quit()

}

})

// preload.js

window.addEventListener('DOMContentLoaded', () => {

const replaceText = (selector, text) => {

const element = document.getElementById(selector)

if (element) element.innerText = text

}

for (const type of ['chrome', 'node', 'electron']) {

replaceText(`${type}-version`, process.versions[type])

}

})

//index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Hello World!</title>

<meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline';" />

</head>

<body>

<h1>Hello World!</h1>

<p>

We are using Node.js <span id="node-version"></span>,

Chromium <span id="chrome-version"></span>,

and Electron <span id="electron-version"></span>.

</p>

</body>

</html>

*What is in the docs?*

Trong tài liệu có gì?

*All the official documentation is available from the sidebar. These are the different categories and what you can expect on each one:*

Tất cả tài liệu chính thức đều có sẵn trong thanh bên. Dưới đây là các danh mục khác nhau và những gì bạn có thể mong đợi ở từng danh mục:

* *Tutorial: An end-to-end guide on how to create and publish your first Electron application.*
* Hướng dẫn chi tiết: Hướng dẫn từng bước về cách tạo và phát hành ứng dụng Electron đầu tiên của bạn.
* *Processes in Electron: In-depth reference on Electron processes and how to work with them.*
* Các quy trình trong Electron: Tài liệu tham khảo chi tiết về các quy trình trong Electron và cách làm việc với chúng.
* *Best Practices: Important checklists to keep in mind when developing an Electron app.*
* Thực hành tốt nhất: Danh sách kiểm tra quan trọng cần lưu ý khi phát triển ứng dụng Electron.
* *Examples: Quick references to add features to your Electron app.*
* Ví dụ: Các tham chiếu nhanh để thêm tính năng vào ứng dụng Electron của bạn.
* *Development: Miscellaneous development guides.*
* Phát triển: Các hướng dẫn phát triển khác nhau.
* *Distribution: Learn how to distribute your app to end users.*
* Phân phối: Học cách phân phối ứng dụng của bạn đến người dùng cuối.
* *Testing And Debugging: How to debug JavaScript, write tests, and other tools used to create quality Electron applications.*
* Kiểm thử và gỡ lỗi: Cách gỡ lỗi JavaScript, viết các bài kiểm thử và các công cụ khác được sử dụng để phát triển ứng dụng Electron chất lượng.
* *References: Useful links to better understand how the Electron project works and is organized.*
* Tham khảo: Các liên kết hữu ích để hiểu rõ hơn về cách dự án Electron hoạt động và được tổ chức.
* *Contributing: Compiling Electron and making contributions can be daunting. We try to make it easier in this section.*
* Đóng góp: Việc biên dịch Electron và đóng góp có thể gây khó khăn. Chúng tôi cố gắng làm cho quá trình này dễ dàng hơn trong phần này.

*Getting help*

Nhận sự trợ giúp

*Are you getting stuck anywhere? Here are a few links to places to look:*

Bạn có gặp khó khăn ở đâu không? Dưới đây là một số liên kết đến các nguồn thông tin hữu ích:

*If you need help with developing your app, our community Discord server is a great place to get advice from other Electron app developers.*

Nếu bạn cần hỗ trợ trong việc phát triển ứng dụng của mình, máy chủ Discord của cộng đồng chúng tôi là nơi tuyệt vời để nhận lời khuyên từ các nhà phát triển ứng dụng Electron khác.

*If you suspect you're running into a bug with the electron package, please check the GitHub issue tracker to see if any existing issues match your problem. If not, feel free to fill out our bug report template and submit a new issue.*

Nếu bạn nghi ngờ rằng mình đang gặp phải lỗi với gói Electron, vui lòng kiểm tra trình theo dõi vấn đề trên GitHub để xem liệu có vấn đề nào hiện có trùng khớp với vấn đề của bạn hay không. Nếu không, hãy thoải mái điền vào mẫu báo cáo lỗi của chúng tôi và gửi một vấn đề mới.

## *Why Electron*

## *Tại sao là Electron?*

*Electron is a framework enabling developers to build cross-platform desktop applications for macOS, Windows, and Linux by combining web technologies (HTML, JavaScript, CSS) with Node.js and native code. It is open-source, MIT-licensed, and free for both commercial and personal use. In this document, we’ll explain why companies and developers choose Electron.*

Electron là một khung công cụ cho phép các nhà phát triển tạo ra các ứng dụng desktop đa nền tảng cho macOS, Windows và Linux bằng cách kết hợp các công nghệ web (HTML, JavaScript, CSS) với Node.js và mã nguồn gốc. Nó là mã nguồn mở, được cấp phép theo giấy phép MIT và miễn phí cho cả mục đích thương mại và cá nhân. Trong tài liệu này, chúng tôi sẽ giải thích lý do tại sao các công ty và nhà phát triển lựa chọn Electron.

*We can split up the benefits of Electron in two questions: First, why should you use web technologies to build your application? Then, why should you choose Electron as the framework to do so?*

Chúng ta có thể chia lợi ích của Electron thành hai câu hỏi: Thứ nhất, tại sao bạn nên sử dụng công nghệ web để phát triển ứng dụng của mình? Thứ hai, tại sao bạn nên chọn Electron làm khung làm việc để thực hiện điều đó?

*If you’re already using web technologies for your application, you can skip straight to the Why Electron? section below.*

Nếu bạn đã sử dụng công nghệ web cho ứng dụng của mình, bạn có thể bỏ qua phần "Tại sao lại là Electron?" bên dưới.

*Why choose web technologies*

Tại sao nên chọn công nghệ web?

*Web technologies include HTML, CSS, JavaScript, and WebAssembly. They’re the storefront of the modern Internet. Those technologies have emerged as the best choice for building user interfaces — both for consumer applications as well as mission-critical business applications. This is true both for applications that need to run in a browser as well as desktop applications that are not accessible from a browser. Our bold claim here is that this isn’t just true for cross-platform applications that need to run on multiple operating systems but true overall.*

Công nghệ web bao gồm HTML, CSS, JavaScript và WebAssembly. Chúng là giao diện chính của Internet hiện đại. Những công nghệ này đã trở thành lựa chọn hàng đầu để phát triển giao diện người dùng — cả cho ứng dụng tiêu dùng lẫn ứng dụng doanh nghiệp quan trọng. Điều này đúng cả với các ứng dụng cần chạy trong trình duyệt lẫn ứng dụng desktop không thể truy cập qua trình duyệt. Lời khẳng định táo bạo của chúng tôi ở đây là điều này không chỉ đúng đối với các ứng dụng đa nền tảng cần chạy trên nhiều hệ điều hành mà còn đúng một cách tổng thể.

*As an example, NASA’s actual Mission Control is written with web technologies. The Bloomberg Terminal, the computer system found at every financial institution, is written with web technologies and runs inside Chromium. It costs $25,000 per user, per year. The McDonald’s ordering kiosk, powering the world’s biggest food retailer, is entirely built with Chromium. The SpaceX’s Dragon 2 space capsule uses Chromium to display its interface. You get the point: web technologies are a great tech stack to build user interfaces.*

Ví dụ, Trung tâm Điều khiển Nhiệm vụ thực tế của NASA được phát triển bằng công nghệ web. Hệ thống máy tính Bloomberg Terminal, có mặt tại mọi tổ chức tài chính, cũng được phát triển bằng công nghệ web và chạy trên nền tảng Chromium. Chi phí cho mỗi người dùng là $25,000 mỗi năm. Kiosk đặt hàng của McDonald’s, hệ thống hỗ trợ chuỗi bán lẻ thực phẩm lớn nhất thế giới, được xây dựng hoàn toàn bằng Chromium. Tàu vũ trụ Dragon 2 của SpaceX sử dụng Chromium để hiển thị giao diện người dùng. Bạn đã hiểu ý: công nghệ web là một bộ công cụ công nghệ tuyệt vời để phát triển giao diện người dùng.

*Here are the reasons we, the Electron maintainers, are betting on the web.*

Dưới đây là những lý do tại sao chúng tôi, các nhà phát triển Electron, tin tưởng vào web.

*Versatility*

Đa năng

*Modern versions of HTML and CSS enable your developers and designers to fully express themselves. The web’s showcase includes Google Earth, Netflix, Spotify, Gmail, Facebook, Airbnb, or GitHub. Whatever interface your application needs, you will be able to express it with HTML, CSS, and JavaScript.*

Các phiên bản hiện đại của HTML và CSS cho phép các nhà phát triển và thiết kế của bạn thể hiện ý tưởng một cách đầy đủ. Các ứng dụng tiêu biểu trên web bao gồm Google Earth, Netflix, Spotify, Gmail, Facebook, Airbnb hoặc GitHub. Dù ứng dụng của bạn cần giao diện như thế nào, bạn đều có thể thể hiện nó bằng HTML, CSS và JavaScript.

*If you want to focus on building a great product without figuring out how you can realize your designer’s vision in a specific UI framework, the web is a safe bet.*

Nếu bạn muốn tập trung vào việc phát triển một sản phẩm tuyệt vời mà không cần phải lo lắng về cách hiện thực hóa tầm nhìn của nhà thiết kế trong một khung giao diện người dùng cụ thể, thì web là một lựa chọn an toàn.

*Reliability*

Độ tin cậy

*Web technologies are the most-used foundation for user interfaces on the planet. They have been hardened accordingly. Modern computers have been optimized from the CPU to the operating system to be good at running web technologies. The manufacturers of your user’s devices—be that an Android phone or the latest MacBook—will ensure that they can visit websites, play videos on YouTube, or display emails. In turn, they’ll also ensure that your app has a stable foundation, even if you have just one user.*

Công nghệ web là nền tảng được sử dụng phổ biến nhất cho giao diện người dùng trên toàn cầu. Chúng đã được tối ưu hóa tương ứng. Các máy tính hiện đại đã được tối ưu hóa từ CPU đến hệ điều hành để có thể chạy các công nghệ web một cách hiệu quả. Các nhà sản xuất thiết bị của người dùng - dù đó là điện thoại Android hay MacBook mới nhất - sẽ đảm bảo rằng họ có thể truy cập trang web, xem video trên YouTube hoặc hiển thị email. Đổi lại, họ cũng sẽ đảm bảo rằng ứng dụng của bạn có một nền tảng ổn định, ngay cả khi bạn chỉ có một người dùng.

If you want to focus on building a great product without debugging a weird quirk that nobody has found before, the web is a safe bet.

Nếu bạn muốn tập trung vào việc phát triển một sản phẩm tuyệt vời mà không phải mất thời gian khắc phục một lỗi kỳ lạ mà chưa ai phát hiện ra trước đó, thì web là một lựa chọn an toàn.

*Interoperability*

Khả năng tương tác

*Whatever provider or customer data you need to interact with, they will have probably thought of an integration path with the web. Depending on your technology choice, embedding a YouTube video either takes 30 seconds or requires you to hire a team devoted to streaming and hardware-accelerated video decoding. In the case of YouTube, using anything other than the provided players is actually against their terms and conditions, so you’ll likely embed a browser frame before you implement your own video streaming decoder.*

Dù bạn cần tương tác với dữ liệu của nhà cung cấp hay khách hàng nào, họ có thể đã nghĩ đến một phương án tích hợp với web. Tùy thuộc vào lựa chọn công nghệ của bạn, việc nhúng video YouTube có thể chỉ mất 30 giây hoặc yêu cầu bạn thuê một đội ngũ chuyên trách về phát trực tuyến và giải mã video bằng phần cứng. Trong trường hợp của YouTube, việc sử dụng bất kỳ trình phát nào khác ngoài trình phát được cung cấp thực sự vi phạm điều khoản và điều kiện của họ, vì vậy bạn có thể sẽ nhúng một khung trình duyệt trước khi triển khai trình giải mã phát trực tuyến video của riêng mình.

*There will be virtually no platform where your app cannot run if you build it with web technologies. Virtually all devices with a display—be that an ATM, a car infotainment system, a smart TV, a fridge, or a Nintendo Switch—come with means to display web technologies. The web is safe bet if you want to be cross-platform.*

Nếu bạn phát triển ứng dụng bằng công nghệ web, gần như không có nền tảng nào mà ứng dụng của bạn không thể chạy được. Hầu hết các thiết bị có màn hình — dù đó là máy ATM, hệ thống giải trí trên xe hơi, tivi thông minh, tủ lạnh hay Nintendo Switch — đều được trang bị khả năng hiển thị công nghệ web. Web là lựa chọn an toàn nếu bạn muốn phát triển ứng dụng đa nền tảng.

*Ubiquity*

Sự phổ biến

*It’s easy to find developers with experience building with web technologies. If you’re a developer, it’ll be easy to find answers to your questions on Google, Stack Overflow, GitHub, or a coding AI of your choice. Whatever problem you need to solve, it’s likely that somebody has solved it well before—and that you can find the answer to the puzzle online.*

Dễ dàng tìm được các nhà phát triển có kinh nghiệm trong việc phát triển ứng dụng web. Nếu bạn là một nhà phát triển, bạn có thể dễ dàng tìm thấy câu trả lời cho các câu hỏi của mình trên Google, Stack Overflow, GitHub hoặc công cụ AI lập trình mà bạn lựa chọn. Bất kỳ vấn đề nào bạn cần giải quyết, rất có thể đã có ai đó giải quyết thành công trước đó — và bạn có thể tìm thấy câu trả lời cho bài toán đó trực tuyến.

*If you want to focus on building a great product with ample access to resources and materials, the web is a safe bet.*

Nếu bạn muốn tập trung vào việc phát triển một sản phẩm xuất sắc với nguồn lực và tài nguyên dồi dào, internet là một lựa chọn an toàn.

*Why choose Electron*

Tại sao nên chọn Electron?

*Electron combines Chromium, Node.js, and the ability to write custom native code into one framework for building powerful desktop applications. There are three main reasons to use Electron:*

Electron kết hợp Chromium, Node.js và khả năng viết mã gốc tùy chỉnh thành một khung làm việc duy nhất để phát triển các ứng dụng desktop mạnh mẽ. Có ba lý do chính để sử dụng Electron:

*Enterprise-grade*

Dành cho doanh nghiệp

*Electron is reliable, secure, stable, and mature. It is the premier choice for companies building their flagship product. We have a list of some of those companies on our homepage, but just among chat apps, Slack, Discord, and Skype are built with Electron. Among AI applications, both OpenAI’s ChatGPT and Anthropic’s Claude use Electron. Visual Studio Code, Loom, Canva, Notion, Docker, and countless other leading developers of software bet on Electron.*

Electron là một công cụ đáng tin cậy, an toàn, ổn định và đã được kiểm chứng. Đây là lựa chọn hàng đầu cho các công ty đang phát triển sản phẩm chủ lực của mình. Chúng tôi có danh sách một số công ty đó trên trang chủ của mình, nhưng chỉ riêng trong lĩnh vực ứng dụng chat, Slack, Discord và Skype đều được xây dựng bằng Electron. Trong lĩnh vực ứng dụng trí tuệ nhân tạo (AI), cả ChatGPT của OpenAI và Claude của Anthropic đều sử dụng Electron. Visual Studio Code, Loom, Canva, Notion, Docker và vô số nhà phát triển phần mềm hàng đầu khác đều tin tưởng vào Electron.

*We did make it a priority to make Electron easy to work with and a delight for developers. That’s likely the main reason why Electron became as popular as it is today — but what keeps Electron alive and thriving is the maintainer’s focus on making Electron as stable, secure, performant, and capable of mission-critical use cases for end users as possible. We’re building an Electron that is ready to be used in scenarios where unfixable bugs, unpatched security holes, and outages of any kind are worst-case scenarios.*

Chúng tôi đã ưu tiên việc làm cho Electron dễ sử dụng và mang lại trải nghiệm tuyệt vời cho các nhà phát triển. Đó có lẽ là lý do chính khiến Electron trở nên phổ biến như ngày nay — nhưng điều gì khiến Electron tiếp tục phát triển và thịnh vượng chính là sự tập trung của các nhà phát triển vào việc làm cho Electron trở nên ổn định, an toàn, hiệu suất cao và có khả năng đáp ứng các trường hợp sử dụng quan trọng cho người dùng cuối. Chúng tôi đang phát triển một phiên bản Electron sẵn sàng cho các tình huống mà các lỗi không thể khắc phục, lỗ hổng bảo mật chưa được vá và các sự cố ngừng hoạt động là những tình huống tồi tệ nhất.

*Mature*

*Our current estimation is that most desktop computers on the planet run at least one Electron app. Electron has grown by prioritizing talent in its maintainer group, fostering excellent and sustainable engineering practices in managing the ongoing maintenance, and proactively inviting companies betting on Electron to directly contribute to the project. We’re an impact project with the OpenJS foundation, which is itself a part of the Linux foundation. We share resources and expertise with other foundation projects like Node.js, ESLint, Webpack - or the Linux Kernel or Kubernetes.*

Ước tính hiện tại của chúng tôi cho thấy hầu hết các máy tính để bàn trên hành tinh này đều chạy ít nhất một ứng dụng Electron. Electron đã phát triển bằng cách ưu tiên nhân tài trong nhóm duy trì, thúc đẩy các thực hành kỹ thuật xuất sắc và bền vững trong việc quản lý việc duy trì liên tục, và chủ động mời các công ty đặt cược vào Electron tham gia trực tiếp vào dự án. Chúng tôi là một dự án có tác động thuộc Quỹ OpenJS, vốn là một phần của Quỹ Linux. Chúng tôi chia sẻ tài nguyên và chuyên môn với các dự án khác của quỹ như Node.js, ESLint, Webpack - hoặc Nhân Linux hay Kubernetes.

*What does all of that mean for you, a developer, in practice?*

Điều đó có ý nghĩa gì đối với bạn, một nhà phát triển, trong thực tế?

* *Reliable release schedule: Electron will release a new major version in lockstep with every second major Chromium release, usually on the same day as Chromium. A lot of work, both in the form of building processes and tools, but also in terms of raw invested hours every week, has to go into making that happen.*
* Lịch trình phát hành đáng tin cậy: Electron sẽ phát hành một phiên bản chính mới đồng bộ với mỗi phiên bản chính thứ hai của Chromium, thường vào cùng ngày với Chromium. Để thực hiện điều này, cần phải đầu tư rất nhiều công sức, cả về quy trình xây dựng và công cụ, cũng như về số giờ làm việc thực tế mỗi tuần.
* *No dictators: Sometimes, betting on a technology also requires you to bet on a single person or company. In turn, it requires you to trust that the person or company never has a breakdown, starts fighting you directly, or does anything else drastic that’ll force you rethink your entire tech stack. Electron is maintained by a diverse set of companies (Microsoft, Slack/Salesforce, Notion, and more) and will continue to welcome more companies interested in ensuring their “seat at the decision-making table”.*
* Không có nhà độc tài: Đôi khi, đầu tư vào một công nghệ cũng đồng nghĩa với việc bạn phải đặt cược vào một cá nhân hoặc công ty cụ thể. Điều này đòi hỏi bạn phải tin tưởng rằng cá nhân hoặc công ty đó sẽ không bao giờ gặp sự cố, không đối đầu trực tiếp với bạn hoặc làm bất cứ điều gì khác có thể buộc bạn phải xem xét lại toàn bộ hệ thống công nghệ của mình. Electron được duy trì bởi một nhóm đa dạng các công ty (Microsoft, Slack/Salesforce, Notion và nhiều công ty khác) và sẽ tiếp tục chào đón thêm các công ty quan tâm đến việc đảm bảo "ghế của họ tại bàn quyết định".

*Stability, security, performance*

Ổn định, an toàn, hiệu suất

*Electron delivers the best experience on all target platforms (macOS, Windows, Linux) by bundling the latest version of Chromium, V8, and Node.js directly with the application binary. When it comes to running and rendering web content with upmost stability, security, and performance, we currently believe that stack to be “best in class”.*

Electron mang đến trải nghiệm tốt nhất trên tất cả các nền tảng mục tiêu (macOS, Windows, Linux) bằng cách tích hợp trực tiếp phiên bản mới nhất của Chromium, V8 và Node.js vào tệp nhị phân ứng dụng. Khi nói đến việc chạy và hiển thị nội dung web với độ ổn định, bảo mật và hiệu suất cao nhất, chúng tôi hiện tin rằng bộ công cụ này là "tốt nhất trong phân khúc".

*Why bundle anything at all*

Tại sao lại gói gọn bất cứ thứ gì?

*You might wonder why we bundle Chromium’s web stack with our apps when most modern operating systems already ship a browser and some form of web view. Bundling doesn’t just increase the amount of work for Electron maintainers dramatically, it also increases the total disk size of Electron apps (most apps are >100MB). Many Electron maintainers once developed applications that did make use of embedded web views — and have since accepted the increased disk size and maintainer work as a worthy trade-off.*

Bạn có thể thắc mắc tại sao chúng tôi lại tích hợp bộ công cụ web của Chromium vào các ứng dụng của mình trong khi hầu hết các hệ điều hành hiện đại đã tích hợp sẵn trình duyệt và một số dạng xem web. Việc tích hợp không chỉ làm tăng đáng kể khối lượng công việc cho các nhà phát triển Electron mà còn làm tăng kích thước đĩa tổng thể của các ứng dụng Electron (hầu hết các ứng dụng đều có kích thước trên 100MB). Nhiều nhà phát triển Electron từng phát triển các ứng dụng sử dụng trình duyệt web nhúng — và sau đó đã chấp nhận việc tăng kích thước đĩa và công việc duy trì như một sự đánh đổi đáng giá.

*When using an operating system's built-in web view, you're limited by the browser version included in the oldest operating system version you need to support. We have found the following problems with this approach:*

Khi sử dụng trình duyệt web tích hợp sẵn trong hệ điều hành, bạn sẽ bị giới hạn bởi phiên bản trình duyệt được tích hợp trong phiên bản hệ điều hành cũ nhất mà bạn cần hỗ trợ. Chúng tôi đã phát hiện ra các vấn đề sau đây với phương pháp này:

* *Stability: The modern web technology stack is complex, and as a result, you’ll sooner or later encounter bugs. If you use the operating system’s web view, your only recourse will be to ask your customers to upgrade their operating system. If no upgrade is available for that machine (because of no ability to upgrade to the latest macOS or Windows 11), you’ll have to ask them to buy a new computer. If you’re unlucky, you’re now losing a major customer because they will not upgrade their entire fleet of thousands of machines just because one team wanted to try your startup’s app. You have no recourse in this situation. Even the risk of that happening is unacceptable to the companies that employ the Electron maintainers.*
* Ổn định: Hệ thống công nghệ web hiện đại rất phức tạp, và do đó, bạn sẽ sớm muộn gì cũng gặp phải lỗi. Nếu bạn sử dụng trình duyệt web tích hợp sẵn trong hệ điều hành, giải pháp duy nhất của bạn là yêu cầu khách hàng nâng cấp hệ điều hành của họ. Nếu không có bản nâng cấp khả dụng cho máy tính đó (vì không thể nâng cấp lên phiên bản macOS hoặc Windows 11 mới nhất), bạn sẽ phải yêu cầu họ mua máy tính mới. Nếu bạn không may mắn, bạn đang mất đi một khách hàng lớn vì họ sẽ không nâng cấp toàn bộ đội máy tính hàng nghìn chiếc chỉ vì một nhóm muốn thử ứng dụng của startup của bạn. Bạn không có cách nào để giải quyết tình huống này. Ngay cả rủi ro đó cũng là không thể chấp nhận được đối với các công ty sử dụng Electron.
* *Security: Similar to how you can fix stability bugs by releasing an app update, you can also release security fixes to your application without asking your customer to upgrade their operating system. Even if operating system providers prioritize updates to their built-in browser, we have not seen them reliably update the built-in web views with similar urgency. Bundling a web renderer gives you, the developer, control.*
* Bảo mật: Tương tự như cách bạn có thể khắc phục các lỗi ổn định bằng cách phát hành bản cập nhật ứng dụng, bạn cũng có thể phát hành các bản vá bảo mật cho ứng dụng của mình mà không cần yêu cầu khách hàng nâng cấp hệ điều hành. Mặc dù các nhà cung cấp hệ điều hành ưu tiên cập nhật trình duyệt tích hợp sẵn, chúng tôi chưa thấy họ cập nhật các trình xem web tích hợp sẵn với mức độ khẩn cấp tương tự. Việc gói gọn trình hiển thị web cho phép bạn, nhà phát triển, kiểm soát.
* *Performance: For simple HTML documents, a built-in web view will sometimes use fewer resources than an app with a bundled framework. For bigger apps, it is our experience that we can deliver better performance with the latest version of Chromium than we can with built-in web views. You might think that the built-in view can share a lot of resources with other apps and the operating system— but for security reasons, apps have to run in their own sandboxes, isolated from each other. At that point, the question is whether the OS’ web view is more performant than Chromium. Across many apps, our experience is that bundling Chromium and Node.js enables us to build better and more performant experiences.*
* Hiệu suất: Đối với các tài liệu HTML đơn giản, trình duyệt web tích hợp có thể sử dụng ít tài nguyên hơn so với ứng dụng có khung gầm tích hợp. Đối với các ứng dụng lớn hơn, kinh nghiệm của chúng tôi cho thấy chúng tôi có thể đạt được hiệu suất tốt hơn với phiên bản mới nhất của Chromium so với trình duyệt web tích hợp. Bạn có thể nghĩ rằng trình duyệt web tích hợp có thể chia sẻ nhiều tài nguyên với các ứng dụng khác và hệ điều hành — nhưng vì lý do bảo mật, các ứng dụng phải chạy trong các hộp cát riêng biệt, cách ly với nhau. Lúc đó, câu hỏi là liệu trình duyệt web của hệ điều hành có hiệu suất cao hơn Chromium hay không. Qua nhiều ứng dụng, kinh nghiệm của chúng tôi cho thấy việc tích hợp Chromium và Node.js giúp chúng tôi xây dựng các trải nghiệm tốt hơn và hiệu suất cao hơn.

*Why bundle Chromium and Node.js*

Tại sao lại gói Chromium và Node.js lại với nhau?

*Electron aims to enable the apps it supports to deliver the best possible user experience, followed by the best possible developer experience. Chromium is currently the best cross-platform rendering stack available. Node.js uses Chromium’s JavaScript engine V8, allowing us to combine the powers of both.*

Electron nhằm mục tiêu giúp các ứng dụng mà nó hỗ trợ cung cấp trải nghiệm người dùng tốt nhất có thể, tiếp theo là trải nghiệm phát triển tốt nhất có thể. Chromium hiện là bộ công cụ hiển thị đa nền tảng tốt nhất hiện có. Node.js sử dụng máy biên dịch JavaScript V8 của Chromium, cho phép chúng ta kết hợp sức mạnh của cả hai.

*Developer experience*

Trải nghiệm của nhà phát triển

*To summarize, we aim to build an Electron that is mature, enterprise-grade, and ready for mission-critical applications. We prioritize reliability, stability, security, and performance. That said, you might also choose Electron for its developer experience:*

Tóm lại, mục tiêu của chúng tôi là xây dựng một Electron trưởng thành, đạt tiêu chuẩn doanh nghiệp và sẵn sàng cho các ứng dụng quan trọng. Chúng tôi ưu tiên tính tin cậy, ổn định, bảo mật và hiệu suất. Tuy nhiên, bạn cũng có thể lựa chọn Electron vì trải nghiệm phát triển của nó:

* Powerful ecosystem: Anything you find on npm will run inside Electron. Any resource available to you about how to work with Node.js also applies to Electron. In addition, Electron itself has a thriving ecosystem — including plenty of choices for installers, updaters, deeper operating system-integration, and more.
* Hệ sinh thái mạnh mẽ: Bất kỳ thứ gì bạn tìm thấy trên npm đều có thể chạy trong Electron. Tất cả tài nguyên có sẵn cho bạn về cách làm việc với Node.js cũng áp dụng cho Electron. Ngoài ra, chính Electron cũng có một hệ sinh thái phát triển mạnh mẽ — bao gồm nhiều lựa chọn cho trình cài đặt, trình cập nhật, tích hợp sâu hơn với hệ điều hành và nhiều tính năng khác.
* *Plenty of built-in capabilities: Over the last ten years, Electron’s core has gained plenty of native capabilities that you might need to build your application. Written in C++ and Objective-C, Electron has dozens of easy-to-use APIs for deeper operating-system integration — like advanced window customization for transparent or oddly shaped widgets, receiving push notifications from the Apple Push Notification Network, or handling a custom URL protocol for your app.*
* Nhiều tính năng tích hợp sẵn: Trong vòng mười năm qua, lõi của Electron đã được trang bị nhiều tính năng tích hợp sẵn mà bạn có thể cần để phát triển ứng dụng của mình. Được viết bằng C++ và Objective-C, Electron cung cấp hàng chục giao diện lập trình ứng dụng (API) dễ sử dụng để tích hợp sâu hơn với hệ điều hành — như tùy chỉnh cửa sổ nâng cao cho các widget trong suốt hoặc có hình dạng đặc biệt, nhận thông báo đẩy từ Mạng Thông báo Đẩy của Apple (Apple Push Notification Network), hoặc xử lý giao thức URL tùy chỉnh cho ứng dụng của bạn.
* *Open source: The entire stack is open source and open to your inspection. This ensures your freedom to add any feature or fix any bug you might encounter in the future.*
* Nguồn mở: Toàn bộ hệ thống là nguồn mở và sẵn sàng cho bạn kiểm tra. Điều này đảm bảo quyền tự do của bạn trong việc thêm bất kỳ tính năng nào hoặc sửa bất kỳ lỗi nào mà bạn có thể gặp phải trong tương lai.
* *Native code when you need it: It bears repeating that Electron allows you to mix and match web technologies and C++, C, Objective-C, Rust, and other native languages. Whether it be SQLite, a whole LLM, or just the ability to call one specific native API, Electron will make it easy.*
* Mã nguồn gốc khi bạn cần: Cần nhấn mạnh rằng Electron cho phép bạn kết hợp các công nghệ web với C++, C, Objective-C, Rust và các ngôn ngữ gốc khác. Dù đó là SQLite, một mô hình ngôn ngữ lớn (LLM) hay chỉ khả năng gọi một API gốc cụ thể, Electron sẽ giúp bạn thực hiện điều đó một cách dễ dàng.

*Why choose something else*

Tại sao lại chọn thứ khác?

*As outlined above, the web is an amazing platform for building interfaces. That doesn’t mean that we, the maintainers, would build everything with HTML and CSS. Here are some notable exceptions:*

Như đã nêu ở trên, web là một nền tảng tuyệt vời để xây dựng giao diện. Điều đó không có nghĩa là chúng tôi, những người duy trì, sẽ xây dựng mọi thứ bằng HTML và CSS. Dưới đây là một số trường hợp ngoại lệ đáng chú ý:

*Resource-Constrained Environments and IoT: In scenarios with very limited memory or processing power (say, one megabyte of memory and 100MHz of processing power on a low-powered ARM Cortex-M), you will likely need to use a low-level language to directly talk to the display to output basic text and images. Even on slightly higher-powered single-chip devices you might want to consider an embedded UI framework. A classic example is a smart watch.*

Môi trường có tài nguyên hạn chế và IoT: Trong các tình huống có bộ nhớ hoặc sức mạnh xử lý rất hạn chế (ví dụ: 1 megabyte bộ nhớ và 100MHz sức mạnh xử lý trên một thiết bị ARM Cortex-M có công suất thấp), bạn có thể cần sử dụng ngôn ngữ cấp thấp để giao tiếp trực tiếp với màn hình nhằm hiển thị văn bản và hình ảnh cơ bản. Ngay cả trên các thiết bị đơn chip có sức mạnh xử lý cao hơn một chút, bạn cũng nên xem xét sử dụng khung giao diện người dùng nhúng. Một ví dụ điển hình là đồng hồ thông minh.

*Small Disk Footprint: Zipped Electron apps are usually around 80 to 100 Megabytes. If a smaller disk footprint is a hard requirement, you’ll have to use something else.*

Dung lượng ổ đĩa nhỏ: Các ứng dụng Electron nén thường có dung lượng khoảng 80 đến 100 megabyte. Nếu dung lượng ổ đĩa nhỏ là yêu cầu bắt buộc, bạn sẽ phải sử dụng giải pháp khác.

*Operating System UI Frameworks and Libraries: By allowing you to write native code, Electron can do anything a native application can do, including the use of the operating system’s UI components, like WinUI, SwiftUI, or AppKit. In practice, most Electron apps make rare use of that ability. If you want the majority of your app to be built with operating system-provided interface components, you’ll likely be better off building fully native apps for each operating system you’d like to target. It’s not that it’s impossible with Electron, it’ll just likely be an overall easier development process.*

Khung giao diện người dùng (UI) và thư viện của hệ điều hành: Bằng cách cho phép bạn viết mã gốc, Electron có thể thực hiện mọi chức năng mà một ứng dụng gốc có thể làm, bao gồm việc sử dụng các thành phần giao diện người dùng của hệ điều hành như WinUI, SwiftUI hoặc AppKit. Trên thực tế, phần lớn các ứng dụng Electron hiếm khi sử dụng khả năng này. Nếu bạn muốn phần lớn ứng dụng của mình được xây dựng bằng các thành phần giao diện do hệ điều hành cung cấp, bạn có thể sẽ tốt hơn khi xây dựng các ứng dụng bản địa hoàn toàn cho từng hệ điều hành mà bạn muốn nhắm đến. Không phải là điều đó không thể thực hiện được với Electron, nhưng quá trình phát triển tổng thể có thể sẽ dễ dàng hơn.

*Games and Real-Time Graphics: If you're building a high-performance game or application requiring complex real-time 3D graphics, native frameworks like Unity, Unreal Engine, or DirectX/OpenGL will provide better performance and more direct access to graphics hardware. Web fans might point out caveats, like the fact that even Unreal Engine ships with Chromium — or that WebGPU and WebGL are developing rapidly and many game engines, including the ones listed here, can now output their games in a format that runs in a browser. That said, if you asked us to build the next AAA game, we’d likely use something else than just web technologies.*

Trò chơi và đồ họa thời gian thực: Nếu bạn đang phát triển một trò chơi hoặc ứng dụng đòi hỏi đồ họa 3D thời gian thực phức tạp, các khung nền tảng gốc như Unity, Unreal Engine hoặc DirectX/OpenGL sẽ cung cấp hiệu suất tốt hơn và truy cập trực tiếp hơn vào phần cứng đồ họa. Người hâm mộ web có thể chỉ ra một số hạn chế, chẳng hạn như việc Unreal Engine được tích hợp sẵn với Chromium — hoặc rằng WebGPU và WebGL đang phát triển nhanh chóng và nhiều engine trò chơi, bao gồm cả những engine được liệt kê ở đây, hiện có thể xuất trò chơi của mình dưới định dạng chạy trên trình duyệt. Tuy nhiên, nếu bạn yêu cầu chúng tôi phát triển trò chơi AAA tiếp theo, chúng tôi có thể sẽ sử dụng công nghệ khác ngoài công nghệ web.

*Embedding Lightweight Websites: Electron apps typically are mostly web apps with native code sprinkled in where useful. Processing-heavy Electron applications tend to write the UI in HTML/CSS and build the backend in Rust, C++, or another native language. If you’re planning to build a primarily native application that also wants to display a little website in a specific view, you might be better off using the OS-provided web view or something like ultralight.*

Tích hợp trang web nhẹ: Ứng dụng Electron thường chủ yếu là ứng dụng web với mã nguồn gốc được tích hợp ở những vị trí cần thiết. Các ứng dụng Electron đòi hỏi nhiều tài nguyên xử lý thường sử dụng HTML/CSS để xây dựng giao diện người dùng và phát triển phần backend bằng Rust, C++ hoặc ngôn ngữ gốc khác. Nếu bạn đang có kế hoạch phát triển một ứng dụng chủ yếu là ứng dụng gốc nhưng vẫn muốn hiển thị một trang web nhỏ trong một giao diện cụ thể, bạn có thể cân nhắc sử dụng trình duyệt web tích hợp sẵn của hệ điều hành hoặc các giải pháp nhẹ như ultralight.

## *Tutorial*

## Hướng dẫn

### *Prerequisites*

### Điều kiện tiên quyết

*Electron is a framework for building desktop applications using JavaScript, HTML, and CSS. By embedding Chromium and Node.js into a single binary file, Electron allows you to create cross-platform apps that work on Windows, macOS, and Linux with a single JavaScript codebase.*

Electron là một khung làm việc (framework) để phát triển ứng dụng desktop bằng JavaScript, HTML và CSS. Bằng cách tích hợp Chromium và Node.js vào một tệp nhị phân duy nhất, Electron cho phép bạn tạo ra các ứng dụng đa nền tảng hoạt động trên Windows, macOS và Linux chỉ với một mã nguồn JavaScript duy nhất.

*This tutorial will guide you through the process of developing a desktop application with Electron and distributing it to end users.*

Hướng dẫn này sẽ hướng dẫn bạn qua quy trình phát triển ứng dụng desktop bằng Electron và phân phối nó cho người dùng cuối.

*Goals*

Mục tiêu

*This tutorial starts by guiding you through the process of piecing together a minimal Electron application from scratch, then teaches you how to package and distribute it to users using Electron Forge.*

Hướng dẫn này bắt đầu bằng việc hướng dẫn bạn từng bước tạo ra một ứng dụng Electron tối giản từ đầu, sau đó hướng dẫn bạn cách đóng gói và phân phối ứng dụng đó cho người dùng thông qua Electron Forge.

*If you prefer to get a project started with a single-command boilerplate, we recommend you start with Electron Forge's create-electron-app command.*

Nếu bạn muốn bắt đầu một dự án với một mẫu mã nguồn mở đơn giản, chúng tôi khuyên bạn nên sử dụng lệnh create-electron-app của Electron Forge.

*Assumptions*

Giả định

*Electron is a native wrapper layer for web apps and is run in a Node.js environment. Therefore, this tutorial assumes you are generally familiar with Node and front-end web development basics. If you need to do some background reading before continuing, we recommend the following resources:*

Electron là một lớp bọc gốc cho ứng dụng web và được chạy trong môi trường Node.js. Do đó, hướng dẫn này giả định rằng bạn đã quen thuộc với Node và các kiến thức cơ bản về phát triển web phía trước. Nếu bạn cần tìm hiểu thêm trước khi tiếp tục, chúng tôi khuyến nghị các tài nguyên sau:

* [*Getting started with the Web (MDN Web Docs)*](https://developer.mozilla.org/en-US/docs/Learn/)
* Bắt đầu với Web (MDN Web Docs)
* [*Introduction to Node.js*](https://nodejs.dev/en/learn/)
* Giới thiệu về Node.js

*Required tools*

Các công cụ cần thiết

*Code editor*

Trình soạn thảo mã nguồn

*You will need a text editor to write your code. We recommend using Visual Studio Code, although you can choose whichever one you prefer.*

Bạn sẽ cần một trình soạn thảo văn bản để viết mã nguồn. Chúng tôi khuyến nghị sử dụng Visual Studio Code, mặc dù bạn có thể chọn bất kỳ trình soạn thảo nào mà bạn ưa thích.

*Command line*

Dòng lệnh

*Throughout the tutorial, we will ask you to use various command-line interfaces (CLIs). You can type these commands into your system's default terminal:*

Trong suốt quá trình hướng dẫn, chúng tôi sẽ yêu cầu bạn sử dụng các giao diện dòng lệnh (CLI) khác nhau. Bạn có thể nhập các lệnh này vào terminal mặc định của hệ thống:

* *Windows: Command Prompt or PowerShell*
* Windows: Command Prompt hoặc PowerShell
* macOS: Terminal
* *Linux: varies depending on distribution (e.g. GNOME Terminal, Konsole)*
* Linux: tùy thuộc vào phiên bản phân phối (ví dụ: GNOME Terminal, Konsole)

*Most code editors also come with an integrated terminal, which you can also use.*

Hầu hết các trình soạn thảo mã nguồn đều đi kèm với một terminal tích hợp, mà bạn cũng có thể sử dụng.

*Git and GitHub*

Git và GitHub

*Git is a commonly-used version control system for source code, and GitHub is a collaborative development platform built on top of it. Although neither is strictly necessary to building an Electron application, we will use GitHub releases to set up automatic updates later on in the tutorial. Therefore, we'll require you to:*

Git là hệ thống kiểm soát phiên bản phổ biến cho mã nguồn, và GitHub là nền tảng phát triển hợp tác được xây dựng trên nền tảng đó. Mặc dù cả hai đều không bắt buộc phải có để phát triển ứng dụng Electron, chúng ta sẽ sử dụng GitHub Releases để thiết lập cập nhật tự động sau này trong hướng dẫn. Do đó, chúng ta yêu cầu bạn:

* *Create a GitHub account*
* Tạo tài khoản GitHub
* *Install Git*
* Cài đặt Git

*If you're unfamiliar with how Git works, we recommend reading GitHub's Git guides. You can also use the GitHub Desktop app if you prefer using a visual interface over the command line.*

Nếu bạn chưa quen với cách hoạt động của Git, chúng tôi khuyên bạn nên đọc các [hướng dẫn Git](https://github.com/git-guides/) của GitHub. Bạn cũng có thể sử dụng ứng dụng [GitHub Desktop](https://desktop.github.com/) nếu bạn thích sử dụng giao diện đồ họa thay vì dòng lệnh.

*We recommend that you create a local Git repository and publish it to GitHub before starting the tutorial, and commit your code after every step.*

Chúng tôi khuyến nghị bạn tạo một kho lưu trữ Git cục bộ và đăng tải nó lên GitHub trước khi bắt đầu hướng dẫn, và commit mã nguồn sau mỗi bước.

*INSTALLING GIT VIA GITHUB DESKTOP*

CÀI ĐẶT GIT THÔNG QUA GITHUB DESKTOP

*GitHub Desktop will install the latest version of Git on your system if you don't already have it installed.*

GitHub Desktop sẽ cài đặt phiên bản mới nhất của Git trên hệ thống của bạn nếu bạn chưa cài đặt nó.

*Node.js and npm*

Node.js và npm

*To begin developing an Electron app, you need to install the Node.js runtime and its bundled npm package manager onto your system. We recommend that you use the latest long-term support (LTS) version.*

Để bắt đầu phát triển ứng dụng Electron, bạn cần cài đặt môi trường chạy Node.js và trình quản lý gói npm đi kèm vào hệ thống của mình. Chúng tôi khuyến nghị bạn sử dụng phiên bản hỗ trợ dài hạn (LTS) mới nhất.

*TIP*

LƯU Ý

*Please install Node.js using pre-built installers for your platform. You may encounter incompatibility issues with different development tools otherwise. If you are using macOS, we recommend using a package manager like Homebrew or nvm to avoid any directory permission issues.*

Vui lòng cài đặt Node.js bằng các trình cài đặt sẵn có cho nền tảng của bạn. Nếu không, bạn có thể gặp phải các vấn đề không tương thích với các công cụ phát triển khác. Nếu bạn đang sử dụng macOS, chúng tôi khuyến nghị sử dụng trình quản lý gói như [Homebrew](https://brew.sh/) hoặc [nvm](https://github.com/nvm-sh/nvm) để tránh các vấn đề liên quan đến quyền truy cập thư mục.

*To check that Node.js was installed correctly, you can use the -v flag when running the node and npm commands. These should print out the installed versions.*

Để kiểm tra xem Node.js đã được cài đặt đúng cách hay chưa, bạn có thể sử dụng tùy chọn -v khi chạy các lệnh node và npm. Các lệnh này sẽ hiển thị phiên bản đã cài đặt.

$ node -v

v16.14.2

$ npm -v

8.7.0

*CAUTION*

CẢNH BÁO

*Although you need Node.js installed locally to scaffold an Electron project, Electron does not use your system's Node.js installation to run its code. Instead, it comes bundled with its own Node.js runtime. This means that your end users do not need to install Node.js themselves as a prerequisite to running your app.*

Mặc dù bạn cần cài đặt Node.js trên máy tính cục bộ để tạo khung cho một dự án Electron, Electron không sử dụng bản cài đặt Node.js của hệ thống để chạy mã của nó. Thay vào đó, nó được tích hợp sẵn với runtime Node.js riêng của mình. Điều này có nghĩa là người dùng cuối không cần phải cài đặt Node.js như một yêu cầu tiên quyết để chạy ứng dụng của bạn.

*To check which version of Node.js is running in your app, you can access the global process.versions variable in the main process or preload script. You can also reference* [*https://releases.electronjs.org/releases.json*](https://releases.electronjs.org/releases.json)*.*

Để kiểm tra phiên bản Node.js đang chạy trong ứng dụng của bạn, bạn có thể truy cập biến toàn cục process.versions trong quá trình chính hoặc tệp preload. Bạn cũng có thể tham khảo <https://releases.electronjs.org/releases.json>.

### *Building your First App*

### Xây dựng ứng dụng đầu tiên của bạn

*Learning goals*

Mục tiêu học tập

*In this part of the tutorial, you will learn how to set up your Electron project and write a minimal starter application. By the end of this section, you should be able to run a working Electron app in development mode from your terminal.*

Trong phần này của hướng dẫn, bạn sẽ học cách thiết lập dự án Electron và viết một ứng dụng khởi đầu tối thiểu. Sau khi hoàn thành phần này, bạn sẽ có thể chạy một ứng dụng Electron hoạt động ở chế độ phát triển từ terminal của mình.

*Setting up your project*

Thiết lập dự án của bạn

*AVOID WSL*

TRÁNH SỬ DỤNG WSL

*If you are on a Windows machine, please do not use Windows Subsystem for Linux (WSL) when following this tutorial as you will run into issues when trying to execute the application.*

Nếu bạn đang sử dụng máy tính Windows, vui lòng không sử dụng Hệ thống con Linux cho Windows (WSL) khi thực hiện theo hướng dẫn này, vì bạn sẽ gặp vấn đề khi cố gắng chạy ứng dụng.

*Initializing your npm project*

Khởi tạo dự án npm

*Electron apps are scaffolded using npm, with the package.json file as an entry point. Start by creating a folder and initializing an npm package within it with npm init.*

Các ứng dụng Electron được tạo khung bằng npm, với tệp package.json làm điểm vào. Bắt đầu bằng cách tạo một thư mục và khởi tạo gói npm bên trong nó bằng lệnh npm init.

mkdir my-electron-app && cd my-electron-app

npm init

*This command will prompt you to configure some fields in your package.json. There are a few rules to follow for the purposes of this tutorial:*

Lệnh này sẽ yêu cầu bạn cấu hình một số trường trong tệp package.json của bạn. Có một số quy tắc cần tuân theo cho mục đích của hướng dẫn này:

* *entry point should be main.js (you will be creating that file soon).*
* Điểm vào nên là main.js (bạn sẽ tạo tệp đó trong thời gian tới).
* *author, license, and description can be any value, but will be necessary for packaging later on.*
* Tác giả, giấy phép và mô tả có thể là bất kỳ giá trị nào, nhưng sẽ cần thiết cho quá trình đóng gói sau này.

*Then, install Electron into your app's devDependencies, which is the list of external development-only package dependencies not required in production.*

Sau đó, cài đặt Electron vào danh sách devDependencies của ứng dụng, đây là danh sách các gói phụ thuộc phát triển bên ngoài chỉ được sử dụng trong quá trình phát triển và không cần thiết trong môi trường sản xuất.

*Why is Electron a devDependency?*

Tại sao Electron lại là một phụ thuộc phát triển (devDependency)?

*This may seem counter-intuitive since your production code is running Electron APIs. However, packaged apps will come bundled with the Electron binary, eliminating the need to specify it as a production dependency.*

Điều này có thể khiến bạn cảm thấy khó hiểu vì mã sản phẩm của bạn đang sử dụng các API của Electron. Tuy nhiên, các ứng dụng được đóng gói sẽ đi kèm với tệp nhị phân Electron, loại bỏ nhu cầu phải chỉ định nó là một phụ thuộc sản phẩm.

npm install electron --save-dev

*Your package.json file should look something like this after initializing your package and installing Electron. You should also now have a node\_modules folder containing the Electron executable, as well as a package-lock.json lockfile that specifies the exact dependency versions to install.*

Tệp package.json của bạn nên trông giống như sau sau khi khởi tạo gói và cài đặt Electron. Bạn cũng nên có một thư mục node\_modules chứa tệp thực thi Electron, cùng với tệp khóa package-lock.json xác định chính xác các phiên bản phụ thuộc cần cài đặt.

{

"name": "my-electron-app",

"version": "1.0.0",

"description": "Hello World!",

"main": "main.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Jane Doe",

"license": "MIT",

"devDependencies": {

"electron": "23.1.3"

}

}

*ADVANCED ELECTRON INSTALLATION STEPS*

CÁC BƯỚC CÀI ĐẶT NÂNG CAO CHO ELECTRON

*If installing Electron directly fails, please refer to our Advanced Installation documentation for instructions on download mirrors, proxies, and troubleshooting steps.*

Nếu việc cài đặt Electron trực tiếp không thành công, vui lòng tham khảo tài liệu Hướng dẫn [cài đặt nâng cao](https://www.electronjs.org/docs/latest/tutorial/installation) của chúng tôi để biết hướng dẫn về các máy chủ tải xuống, proxy và các bước khắc phục sự cố.

*Adding a .gitignore*

Thêm tệp .gitignore

*The .gitignore file specifies which files and directories to avoid tracking with Git. You should place a copy of GitHub's Node.js gitignore template into your project's root folder to avoid committing your project's node\_modules folder.*

Tệp .gitignore xác định các tệp và thư mục cần loại trừ khỏi quá trình theo dõi của Git. Bạn nên sao chép [mẫu .gitignore của Node.js](https://github.com/github/gitignore/blob/main/Node.gitignore) từ GitHub vào thư mục gốc của dự án để tránh commit thư mục node\_modules của dự án.

*Running an Electron app*

Chạy ứng dụng Electron

*FURTHER READING*

ĐỌC THÊM

*Read Electron's process model documentation to better understand how Electron's multiple processes work together.*

Đọc tài liệu [mô hình quy trình của Electron](https://www.electronjs.org/docs/latest/tutorial/process-model) để hiểu rõ hơn về cách các quy trình đa nhiệm của Electron hoạt động cùng nhau.

*The main script you defined in package.json is the entry point of any Electron application. This script controls the main process, which runs in a Node.js environment and is responsible for controlling your app's lifecycle, displaying native interfaces, performing privileged operations, and managing renderer processes (more on that later).*

Script main mà bạn đã định nghĩa trong package.json là điểm vào của bất kỳ ứng dụng Electron nào. Script này kiểm soát quá trình chính, chạy trong môi trường Node.js và chịu trách nhiệm kiểm soát vòng đời của ứng dụng, hiển thị các giao diện gốc, thực hiện các thao tác có quyền truy cập đặc quyền và quản lý các quá trình renderer (sẽ được đề cập chi tiết hơn sau).

*Before creating your first Electron app, you will first use a trivial script to ensure your main process entry point is configured correctly. Create a main.js file in the root folder of your project with a single line of code:*

Trước khi tạo ứng dụng Electron đầu tiên, bạn sẽ sử dụng một script đơn giản để đảm bảo điểm vào của quá trình chính được cấu hình đúng cách. Tạo tệp main.js trong thư mục gốc của dự án với một dòng mã duy nhất:

console.log('Hello from Electron 👋')

*Because Electron's main process is a Node.js runtime, you can execute arbitrary Node.js code with the electron command (you can even use it as a REPL). To execute this script, add electron . to the start command in the scripts field of your package.json. This command will tell the Electron executable to look for the main script in the current directory and run it in dev mode.*

Vì quá trình chính của Electron là một môi trường chạy Node.js, bạn có thể thực thi mã Node.js tùy ý bằng lệnh electron (bạn thậm chí có thể sử dụng nó như một REPL). Để thực thi script này, hãy thêm electron . vào lệnh start trong trường scripts của tệp package.json. Lệnh này sẽ thông báo cho trình thực thi Electron tìm kiếm script chính trong thư mục hiện tại và chạy nó ở chế độ phát triển.

{

"name": "my-electron-app",

"version": "1.0.0",

"description": "Hello World!",

"main": "main.js",

"scripts": {

"start": "electron .",

"test": "echo \"Error: no test specified\" && exit 1"

},

"author": "Jane Doe",

"license": "MIT",

"devDependencies": {

"electron": "23.1.3"

}

}

npm run start

*Your terminal should print out Hello from Electron 👋. Congratulations, you have executed your first line of code in Electron! Next, you will learn how to create user interfaces with HTML and load that into a native window.*

Máy tính của bạn sẽ hiển thị thông báo Hello from Electron 👋. Chúc mừng, bạn đã thực thi dòng code đầu tiên trong Electron! Tiếp theo, bạn sẽ học cách tạo giao diện người dùng bằng HTML và tải nó vào cửa sổ gốc.

*Loading a web page into a BrowserWindow*

Tải trang web vào cửa sổ trình duyệt

*In Electron, each window displays a web page that can be loaded either from a local HTML file or a remote web address. For this example, you will be loading in a local file. Start by creating a barebones web page in an index.html file in the root folder of your project:*

Trong Electron, mỗi cửa sổ hiển thị một trang web có thể được tải từ một tệp HTML cục bộ hoặc một địa chỉ web từ xa. Trong ví dụ này, bạn sẽ tải một tệp cục bộ. Bắt đầu bằng cách tạo một trang web cơ bản trong tệp index.html ở thư mục gốc của dự án:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<!-- https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP -->

<meta

http-equiv="Content-Security-Policy"

content="default-src 'self'; script-src 'self'"

/>

<meta

http-equiv="X-Content-Security-Policy"

content="default-src 'self'; script-src 'self'"

/>

<title>Hello from Electron renderer!</title>

</head>

<body>

<h1>Hello from Electron renderer!</h1>

<p>👋</p>

</body>

</html>

*Now that you have a web page, you can load it into an Electron BrowserWindow. Replace the contents of your main.js file with the following code. We will explain each highlighted block separately.*

Bây giờ bạn đã có một trang web, bạn có thể tải nó vào cửa sổ trình duyệt Electron. Thay thế nội dung của tệp main.js bằng mã sau. Chúng tôi sẽ giải thích từng khối được đánh dấu riêng biệt.

const { app, BrowserWindow } = require('electron')

const createWindow = () => {

const win = new BrowserWindow({

width: 800,

height: 600

})

win.loadFile('index.html')

}

app.whenReady().then(() => {

createWindow()

})

*Importing modules*

Nhập các mô-đun

const { app, BrowserWindow } = require('electron')

*In the first line, we are importing two Electron modules with CommonJS module syntax:*

Trong dòng đầu tiên, chúng ta đang nhập hai mô-đun Electron bằng cú pháp mô-đun CommonJS:

* *app, which controls your application's event lifecycle.*
* app, quản lý chu kỳ sự kiện của ứng dụng của bạn.
* *BrowserWindow, which creates and manages app windows.*
* BrowserWindow, thành phần tạo và quản lý các cửa sổ ứng dụng.

*Module capitalization conventions*

Quy ước viết hoa cho các mô-đun

*You might have noticed the capitalization difference between the app and BrowserWindow modules. Electron follows typical JavaScript conventions here, where PascalCase modules are instantiable class constructors (e.g. BrowserWindow, Tray, Notification) whereas camelCase modules are not instantiable (e.g. app, ipcRenderer, webContents).*

Bạn có thể đã nhận thấy sự khác biệt về cách viết hoa giữa các mô-đun app và BrowserWindow. Electron tuân theo các quy ước thông thường của JavaScript ở đây, trong đó các mô-đun viết hoa theo kiểu PascalCase là các hàm tạo lớp có thể khởi tạo (ví dụ: BrowserWindow, Tray, Notification), trong khi các mô-đun viết hoa theo kiểu camelCase không thể khởi tạo (ví dụ: app, ipcRenderer, webContents).

*Typed import aliases*

Các biệt danh nhập khẩu được định nghĩa

*For better type checking when writing TypeScript code, you can choose to import main process modules from electron/main.*

Để kiểm tra kiểu dữ liệu tốt hơn khi viết mã TypeScript, bạn có thể chọn nhập các mô-đun quá trình chính từ electron/main.

const { app, BrowserWindow } = require('electron/main')

*For more information, see the* [*Process Model docs*](https://www.electronjs.org/docs/latest/tutorial/process-model#process-specific-module-aliases-typescript)*.*

Để biết thêm thông tin, hãy tham khảo [tài liệu về Mô hình Quá trình](https://www.electronjs.org/docs/latest/tutorial/process-model#process-specific-module-aliases-typescript).

*ES MODULES IN ELECTRON*

CÁC MÔ-ĐUN ES TRONG ELECTRON

*ECMAScript modules (i.e. using import to load a module) are supported in Electron as of Electron 28. You can find more information about the state of ESM in Electron and how to use them in our app in our ESM guide.*

Các mô-đun ECMAScript (tức là sử dụng lệnh import để tải mô-đun) được hỗ trợ trong Electron kể từ phiên bản Electron 28. Bạn có thể tìm thêm thông tin về tình trạng hỗ trợ ESM trong Electron và cách sử dụng chúng trong ứng dụng của chúng tôi trong hướng dẫn ESM của chúng tôi.

*Writing a reusable function to instantiate windows*

Viết một hàm có thể tái sử dụng để khởi tạo cửa sổ

*The createWindow() function loads your web page into a new BrowserWindow instance:*

Hàm createWindow() tải trang web của bạn vào một đối tượng BrowserWindow mới:

const createWindow = () => {

const win = new BrowserWindow({

width: 800,

height: 600

})

win.loadFile('index.html')

}

*Calling your function when the app is ready*

Gọi hàm của bạn khi ứng dụng đã sẵn sàng

app.whenReady().then(() => {

createWindow()

})

*Many of Electron's core modules are Node.js event emitters that adhere to Node's asynchronous event-driven architecture. The app module is one of these emitters.*

Nhiều mô-đun lõi của Electron là các trình phát sự kiện Node.js tuân thủ kiến trúc sự kiện không đồng bộ của Node. Mô-đun ứng dụng là một trong số các trình phát sự kiện này.

*In Electron, BrowserWindows can only be created after the app module's ready event is fired. You can wait for this event by using the app.whenReady() API and calling createWindow() once its promise is fulfilled.*

Trong Electron, các cửa sổ trình duyệt (BrowserWindows) chỉ có thể được tạo sau khi sự kiện ready của mô-đun ứng dụng được kích hoạt. Bạn có thể đợi sự kiện này bằng cách sử dụng API app.whenReady() và gọi hàm createWindow() sau khi lời hứa (promise) của nó được hoàn thành.

*INFO*

THÔNG TIN

*You typically listen to Node.js events by using an emitter's .on function.*

Bạn thường lắng nghe các sự kiện Node.js bằng cách sử dụng hàm .on của emitter.

+ app.on('ready', () => {

- app.whenReady().then(() => {

createWindow()

})

*However, Electron exposes app.whenReady() as a helper specifically for the ready event to avoid subtle pitfalls with directly listening to that event in particular. See* [*electron/electron#21972*](https://github.com/electron/electron/pull/21972) *for details.*

Tuy nhiên, Electron cung cấp phương thức app.whenReady() như một hàm trợ giúp dành riêng cho sự kiện ready để tránh những rủi ro tiềm ẩn khi trực tiếp lắng nghe sự kiện đó. Xem electron/electron#21972 để biết chi tiết.

*At this point, running your Electron application's start command should successfully open a window that displays your web page!*

Tại thời điểm này, việc chạy lệnh khởi động của ứng dụng Electron của bạn sẽ mở thành công một cửa sổ hiển thị trang web của bạn!

*Each web page your app displays in a window will run in a separate process called a renderer process (or simply renderer for short). Renderer processes have access to the same JavaScript APIs and tooling you use for typical front-end web development, such as using webpack to bundle and minify your code or React to build your user interfaces.*

Mỗi trang web mà ứng dụng của bạn hiển thị trong cửa sổ sẽ chạy trong một quy trình riêng biệt được gọi là quy trình renderer (hoặc đơn giản là renderer). Các quy trình renderer có quyền truy cập vào các API JavaScript và công cụ tương tự như bạn sử dụng trong phát triển front-end web thông thường, chẳng hạn như sử dụng webpack để gói và nén mã nguồn hoặc React để xây dựng giao diện người dùng.

*Managing your app's window lifecycle*

Quản lý vòng đời cửa sổ của ứng dụng

*Application windows behave differently on each operating system. Rather than enforce these conventions by default, Electron gives you the choice to implement them in your app code if you wish to follow them. You can implement basic window conventions by listening for events emitted by the app and BrowserWindow modules.*

Các cửa sổ ứng dụng hoạt động khác nhau trên mỗi hệ điều hành. Thay vì áp dụng các quy ước này theo mặc định, Electron cho phép bạn lựa chọn triển khai chúng trong mã ứng dụng của mình nếu bạn muốn tuân theo. Bạn có thể triển khai các quy ước cơ bản của cửa sổ bằng cách lắng nghe các sự kiện được phát ra bởi các mô-đun app và BrowserWindow.

*PROCESS-SPECIFIC CONTROL FLOW*

LUỒNG ĐIỀU KHIỂN CỤ THỂ CHO QUY TRÌNH

*Checking against Node's process.platform variable can help you to run code conditionally on certain platforms. Note that there are only three possible platforms that Electron can run in: win32 (Windows), linux (Linux), and darwin (macOS).*

Kiểm tra biến [process.platform](https://nodejs.org/api/process.html#process_process_platform) của Node.js có thể giúp bạn chạy mã một cách có điều kiện trên các nền tảng cụ thể. Lưu ý rằng Electron chỉ có thể chạy trên ba nền tảng sau: win32 (Windows), linux (Linux) và darwin (macOS).

*Quit the app when all windows are closed (Windows & Linux)*

Đóng ứng dụng khi tất cả các cửa sổ đã được đóng (Windows & Linux)

*On Windows and Linux, closing all windows will generally quit an application entirely. To implement this pattern in your Electron app, listen for the app module's window-all-closed event, and call app.quit() to exit your app if the user is not on macOS.*

Trên Windows và Linux, việc đóng tất cả các cửa sổ thường sẽ thoát hoàn toàn ứng dụng. Để triển khai mẫu này trong ứng dụng Electron của bạn, hãy lắng nghe sự kiện window-all-closed của mô-đun ứng dụng và gọi app.quit() để thoát ứng dụng nếu người dùng không sử dụng macOS.

app.on('window-all-closed', () => {

if (process.platform !== 'darwin') app.quit()

})

*Open a window if none are open (macOS)*

Mở cửa sổ nếu không có cửa sổ nào đang mở (macOS)

*In contrast, macOS apps generally continue running even without any windows open. Activating the app when no windows are available should open a new one.*

Ngược lại, các ứng dụng macOS thường tiếp tục chạy ngay cả khi không có cửa sổ nào mở. Khi kích hoạt ứng dụng mà không có cửa sổ nào sẵn có, ứng dụng sẽ mở một cửa sổ mới.

*To implement this feature, listen for the app module's activate event, and call your existing createWindow() method if no BrowserWindows are open.*

Để triển khai tính năng này, hãy lắng nghe sự kiện activate của mô-đun ứng dụng và gọi phương thức createWindow() hiện có nếu không có cửa sổ trình duyệt nào đang mở.

*Because windows cannot be created before the ready event, you should only listen for activate events after your app is initialized. Do this by only listening for activate events inside your existing whenReady() callback.*

Vì cửa sổ không thể được tạo ra trước sự kiện ready, bạn chỉ nên lắng nghe sự kiện activate sau khi ứng dụng của bạn đã được khởi tạo. Để làm điều này, hãy chỉ lắng nghe sự kiện activate bên trong hàm callback whenReady() hiện có của bạn.

app.whenReady().then(() => {

createWindow()

app.on('activate', () => {

if (BrowserWindow.getAllWindows().length === 0) createWindow()

})

})

*Final starter code*

Mã khởi động cuối cùng

const { app, BrowserWindow } = require('electron/main')

const createWindow = () => {

const win = new BrowserWindow({

width: 800,

height: 600

})

win.loadFile('index.html')

}

app.whenReady().then(() => {

createWindow()

app.on('activate', () => {

if (BrowserWindow.getAllWindows().length === 0) {

createWindow()

}

})

})

app.on('window-all-closed', () => {

if (process.platform !== 'darwin') {

app.quit()

}

})

*Optional: Debugging from VS Code*

Tùy chọn: Gỡ lỗi từ VS Code

*If you want to debug your application using VS Code, you need to attach VS Code to both the main and renderer processes. Here is a sample configuration for you to run. Create a launch.json configuration in a new .vscode folder in your project:*

Nếu bạn muốn gỡ lỗi ứng dụng của mình bằng VS Code, bạn cần kết nối VS Code với cả quá trình chính và quá trình renderer. Dưới đây là một cấu hình mẫu để bạn chạy. Tạo một tệp cấu hình launch.json trong thư mục .vscode mới trong dự án của bạn:

{

"version": "0.2.0",

"compounds": [

{

"name": "Main + renderer",

"configurations": ["Main", "Renderer"],

"stopAll": true

}

],

"configurations": [

{

"name": "Renderer",

"port": 9222,

"request": "attach",

"type": "chrome",

"webRoot": "${workspaceFolder}"

},

{

"name": "Main",

"type": "node",

"request": "launch",

"cwd": "${workspaceFolder}",

"runtimeExecutable": "${workspaceFolder}/node\_modules/.bin/electron",

"windows": {

"runtimeExecutable": "${workspaceFolder}/node\_modules/.bin/electron.cmd"

},

"args": [".", "--remote-debugging-port=9222"],

"outputCapture": "std",

"console": "integratedTerminal"

}

]

}

*The "Main + renderer" option will appear when you select "Run and Debug" from the sidebar, allowing you to set breakpoints and inspect all the variables among other things in both the main and renderer processes.*

Tùy chọn "Main + renderer" sẽ xuất hiện khi bạn chọn "Run and Debug" từ thanh bên, cho phép bạn đặt điểm dừng và kiểm tra tất cả các biến trong cả quá trình chính và quá trình renderer, cùng với các tác vụ khác.

*What we have done in the launch.json file is to create 3 configurations:*

Trong tệp launch.json, chúng tôi đã tạo 3 cấu hình:

* *Main is used to start the main process and also expose port 9222 for remote debugging (--remote-debugging-port=9222). This is the port that we will use to attach the debugger for the Renderer. Because the main process is a Node.js process, the type is set to node.*
* Main được sử dụng để khởi động quá trình chính và cũng mở cổng 9222 cho việc gỡ lỗi từ xa (--remote-debugging-port=9222). Đây là cổng mà chúng ta sẽ sử dụng để kết nối trình gỡ lỗi cho Renderer. Vì quá trình chính là một quá trình Node.js, loại được đặt là node.
* *Renderer is used to debug the renderer process. Because the main process is the one that creates the process, we have to "attach" to it ("request": "attach") instead of creating a new one. The renderer process is a web one, so the debugger we have to use is chrome.*
* Renderer được sử dụng để gỡ lỗi quá trình renderer. Vì quá trình chính là quá trình tạo ra quá trình renderer, chúng ta phải "kết nối" với nó ("request": "attach") thay vì tạo một quá trình mới. Quá trình renderer là một quá trình web, do đó trình gỡ lỗi chúng ta phải sử dụng là Chrome.
* *Main + renderer is a* [*compound task*](https://code.visualstudio.com/Docs/editor/tasks#_compound-tasks) *that executes the previous ones simultaneously.*
* Main + renderer là một tác vụ phức hợp thực thi đồng thời các tác vụ trước đó.

*CAUTION*

LƯU Ý

*Because we are attaching to a process in Renderer, it is possible that the first lines of your code will be skipped as the debugger will not have had enough time to connect before they are being executed. You can work around this by refreshing the page or setting a timeout before executing the code in development mode.*

Vì chúng ta đang gắn vào một quy trình trong Renderer, có thể các dòng code đầu tiên của bạn sẽ bị bỏ qua vì trình gỡ lỗi chưa kịp kết nối trước khi chúng được thực thi. Bạn có thể khắc phục điều này bằng cách làm mới trang hoặc đặt thời gian chờ trước khi thực thi code trong chế độ phát triển.

*FURTHER READING*

ĐỌC THÊM

*If you want to dig deeper in the debugging area, the following guides provide more information:*

Nếu bạn muốn tìm hiểu sâu hơn về việc gỡ lỗi, các hướng dẫn sau đây cung cấp thêm thông tin:

* [*Application Debugging*](https://www.electronjs.org/docs/latest/tutorial/application-debugging)
* Gỡ lỗi ứng dụng
* [*DevTools Extensions*](https://www.electronjs.org/docs/latest/tutorial/devtools-extension)
* Phần mở rộng DevTools

*Summary*

Tóm tắt

*Electron applications are set up using npm packages. The Electron executable should be installed in your project's devDependencies and can be run in development mode using a script in your package.json file.*

Ứng dụng Electron được thiết lập bằng cách sử dụng các gói npm. Tệp thực thi Electron nên được cài đặt trong phần devDependencies của dự án và có thể được chạy ở chế độ phát triển bằng cách sử dụng một script trong tệp package.json của bạn.

*The executable runs the JavaScript entry point found in the main property of your package.json. This file controls Electron's main process, which runs an instance of Node.js and is responsible for your app's lifecycle, displaying native interfaces, performing privileged operations, and managing renderer processes.*

Tệp thực thi sẽ chạy điểm vào JavaScript được tìm thấy trong thuộc tính main của tệp package.json của bạn. Tệp này kiểm soát quá trình chính của Electron, chạy một phiên bản của Node.js và chịu trách nhiệm về vòng đời ứng dụng của bạn, hiển thị các giao diện gốc, thực hiện các thao tác có quyền truy cập đặc quyền và quản lý các quá trình renderer.

*Renderer processes (or renderers for short) are responsible for displaying graphical content. You can load a web page into a renderer by pointing it to either a web address or a local HTML file. Renderers behave very similarly to regular web pages and have access to the same web APIs.*

Các quy trình hiển thị (hoặc gọi tắt là renderer) chịu trách nhiệm hiển thị nội dung đồ họa. Bạn có thể tải một trang web vào renderer bằng cách chỉ định địa chỉ web hoặc tệp HTML cục bộ. Các renderer hoạt động tương tự như các trang web thông thường và có quyền truy cập vào các API web tương tự.

*In the next section of the tutorial, we will be learning how to augment the renderer process with privileged APIs and how to communicate between processes.*

Trong phần tiếp theo của hướng dẫn, chúng ta sẽ tìm hiểu cách mở rộng quá trình renderer bằng các API có quyền truy cập đặc quyền và cách giao tiếp giữa các quá trình.

***Using Preload Scripts***

**Sử dụng các tập lệnh tải trước**

*Learning goals*

Mục tiêu học tập

*In this part of the tutorial, you will learn what a preload script is and how to use one to securely expose privileged APIs into the renderer process. You will also learn how to communicate between main and renderer processes with Electron's inter-process communication (IPC) modules.*

Trong phần này của hướng dẫn, bạn sẽ tìm hiểu về kịch bản preload là gì và cách sử dụng nó để an toàn phơi bày các API có quyền truy cập đặc quyền vào quá trình renderer. Bạn cũng sẽ học cách giao tiếp giữa quá trình chính và quá trình renderer bằng các mô-đun giao tiếp giữa các quá trình (IPC) của Electron.

*What is a preload script?*

Preload script là gì?

*Electron's main process is a Node.js environment that has full operating system access. On top of Electron modules, you can also access Node.js built-ins, as well as any packages installed via npm. On the other hand, renderer processes run web pages and do not run Node.js by default for security reasons.*

Quá trình chính của Electron là một môi trường Node.js có quyền truy cập đầy đủ vào hệ điều hành. Ngoài các mô-đun của Electron, bạn cũng có thể truy cập các hàm tích hợp sẵn của Node.js cũng như bất kỳ gói nào được cài đặt thông qua npm. Mặt khác, các quá trình renderer chạy các trang web và không chạy Node.js theo mặc định vì lý do bảo mật.

*To bridge Electron's different process types together, we will need to use a special script called a preload.*

Để kết nối các loại quy trình khác nhau của Electron lại với nhau, chúng ta sẽ cần sử dụng một kịch bản đặc biệt gọi là preload.

*Augmenting the renderer with a preload script*

Tăng cường trình hiển thị bằng một preload script

*A BrowserWindow's preload script runs in a context that has access to both the HTML DOM and a limited subset of Node.js and Electron APIs.*

Script tải trước của BrowserWindow chạy trong một bối cảnh có quyền truy cập vào cả HTML DOM và một tập con giới hạn của các API Node.js và Electron.

*PRELOAD SCRIPT SANDBOXING*

TÍNH NĂNG CÁCH LY SANDBOX CHO PRELOAD SCRIPT

*From Electron 20 onwards, preload scripts are sandboxed by default and no longer have access to a full Node.js environment. Practically, this means that you have a polyfilled require function that only has access to a limited set of APIs.*

Từ Electron 20 trở đi, các script khởi động trước (preload scripts) được cách ly trong môi trường sandbox theo mặc định và không còn truy cập được vào môi trường Node.js đầy đủ. Trên thực tế, điều này có nghĩa là bạn có một hàm require được polyfill chỉ truy cập được vào một tập hợp giới hạn các API.

|  |  |
| --- | --- |
| Available API | Details |
| Electron modules | Renderer process modules |
| Node.js modules | [events](https://nodejs.org/api/events.html), [timers](https://nodejs.org/api/timers.html), [url](https://nodejs.org/api/url.html) |
| Polyfilled globals | [Buffer](https://nodejs.org/api/buffer.html), [process](https://www.electronjs.org/docs/latest/api/process), [clearImmediate](https://nodejs.org/api/timers.html#timers_clearimmediate_immediate), [setImmediate](https://nodejs.org/api/timers.html#timers_setimmediate_callback_args) |

*For more information, check out the Process Sandboxing guide.*

Để biết thêm thông tin, hãy tham khảo hướng dẫn về [quy trình Sandboxing](https://www.electronjs.org/docs/latest/tutorial/sandbox).

*Preload scripts are injected before a web page loads in the renderer, similar to a Chrome extension's content scripts. To add features to your renderer that require privileged access, you can define global objects through the contextBridge API.*

Các preload script được chèn vào trình hiển thị trước khi trang web được tải, tương tự như các script nội dung của tiện ích mở rộng Chrome. Để thêm các tính năng vào trình hiển thị yêu cầu quyền truy cập đặc quyền, bạn có thể định nghĩa các đối tượng global thông qua API contextBridge.

*To demonstrate this concept, you will create a preload script that exposes your app's versions of Chrome, Node, and Electron into the renderer.*

Để minh họa khái niệm này, bạn sẽ tạo một script tải trước (preload script) để hiển thị các phiên bản Chrome, Node và Electron của ứng dụng vào trình hiển thị (renderer).

*Add a new preload.js script that exposes selected properties of Electron's process.versions object to the renderer process in a versions global variable.*

Thêm một tệp script mới có tên preload.js để phơi bày các thuộc tính được chọn của đối tượng process.versions của Electron cho quá trình renderer thông qua biến toàn cục versions.

// preload.js

const { contextBridge } = require('electron')

contextBridge.exposeInMainWorld('versions', {

node: () => process.versions.node,

chrome: () => process.versions.chrome,

electron: () => process.versions.electron

// we can also expose variables, not just functions

})

*To attach this script to your renderer process, pass its path to the webPreferences.preload option in the BrowserWindow constructor:*

Để gắn kịch bản này vào quá trình renderer, hãy truyền đường dẫn của nó vào tùy chọn webPreferences.preload trong hàm tạo BrowserWindow:

// main.js

const { app, BrowserWindow } = require('electron')

const path = require('node:path')

const createWindow = () => {

const win = new BrowserWindow({

width: 800,

height: 600,

webPreferences: {

preload: path.join(\_\_dirname, 'preload.js')

}

})

win.loadFile('index.html')

}

app.whenReady().then(() => {

createWindow()

})

***INFO***

**THÔNG TIN**

*There are two Node.js concepts that are used here:*

Có hai khái niệm Node.js được sử dụng ở đây:

* *The \_\_dirname string points to the path of the currently executing script (in this case, your project's root folder).*
* Chuỗi \_\_dirname trỏ đến đường dẫn của tệp kịch bản đang được thực thi (trong trường hợp này là thư mục gốc của dự án của bạn).
* *The path.join API joins multiple path segments together, creating a combined path string that works across all platforms.*
* Phương thức path.join kết hợp nhiều đoạn đường dẫn lại với nhau, tạo ra một chuỗi đường dẫn kết hợp có thể sử dụng trên tất cả các nền tảng.

*At this point, the renderer has access to the versions global, so let's display that information in the window. This variable can be accessed via window.versions or simply versions. Create a renderer.js script that uses the document.getElementById DOM API to replace the displayed text for the HTML element with info as its id property.*

Tại thời điểm này, renderer đã có quyền truy cập vào biến global versions, vì vậy, hãy hiển thị thông tin đó trong cửa sổ. Biến này có thể được truy cập thông qua window.versions hoặc đơn giản là versions. Hãy tạo một script renderer.js sử dụng DOM API document.getElementById để thay thế văn bản hiển thị cho phần tử HTML có thuộc tính id là info.

// renderer.js

const information = document.getElementById('info')

information.innerText = `This app is using Chrome (v${versions.chrome()}), Node.js (v${versions.node()}), and Electron (v${versions.electron()})`

*Then, modify your index.html by adding a new element with info as its id property, and attach your renderer.js script:*

Sau đó, chỉnh sửa tệp index.html bằng cách thêm một phần tử mới có thuộc tính id là info, và gắn tệp renderer.js vào:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<meta

http-equiv="Content-Security-Policy"

content="default-src 'self'; script-src 'self'"

/>

<meta

http-equiv="X-Content-Security-Policy"

content="default-src 'self'; script-src 'self'"

/>

<title>Hello from Electron renderer!</title>

</head>

<body>

<h1>Hello from Electron renderer!</h1>

<p>👋</p>

<p id="info"></p>

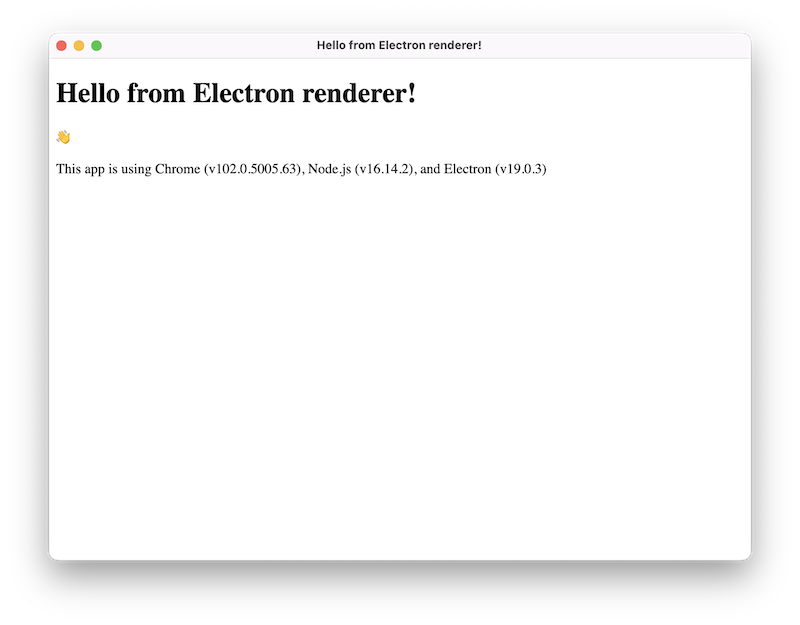
</body>

<script src="./renderer.js"></script>

</html>

*After following the above steps, your app should look something like this:*

Sau khi thực hiện các bước trên, ứng dụng của bạn sẽ trông giống như sau:



*And the code should look like this:*

Và mã nguồn nên trông như sau:

// main.js

const { app, BrowserWindow } = require('electron/main')

const path = require('node:path')

const createWindow = () => {

const win = new BrowserWindow({

width: 800,

height: 600,

webPreferences: {

preload: path.join(\_\_dirname, 'preload.js')

}

})

win.loadFile('index.html')

}

app.whenReady().then(() => {

createWindow()

app.on('activate', () => {

if (BrowserWindow.getAllWindows().length === 0) {

createWindow()

}

})

})

app.on('window-all-closed', () => {

if (process.platform !== 'darwin') {

app.quit()

}

})

// preload.js

const { contextBridge } = require('electron/renderer')

contextBridge.exposeInMainWorld('versions', {

node: () => process.versions.node,

chrome: () => process.versions.chrome,

electron: () => process.versions.electron

})

// index.html

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8" />

<meta

http-equiv="Content-Security-Policy"

content="default-src 'self'; script-src 'self'"

/>

<meta

http-equiv="X-Content-Security-Policy"

content="default-src 'self'; script-src 'self'"

/>

<title>Hello from Electron renderer!</title>

</head>

<body>

<h1>Hello from Electron renderer!</h1>

<p>👋</p>

<p id="info"></p>

</body>

<script src="./renderer.js"></script>

</html>

// renderer.js

const information = document.getElementById('info')

information.innerText = `This app is using Chrome (v${window.versions.chrome()}), Node.js (v${window.versions.node()}), and Electron (v${window.versions.electron()})`

*Communicating between processes*

Giao tiếp giữa các tiến trình

*As we have mentioned above, Electron's main and renderer process have distinct responsibilities and are not interchangeable. This means it is not possible to access the Node.js APIs directly from the renderer process, nor the HTML Document Object Model (DOM) from the main process.*

Như chúng ta đã đề cập ở trên, main process và renderer process của Electron có những trách nhiệm riêng biệt và không thể hoán đổi cho nhau. Điều này có nghĩa là không thể truy cập trực tiếp các Node.js API từ renderer process, cũng như không thể truy cập HTML Document Object Model (DOM) từ main process.

*The solution for this problem is to use Electron's ipcMain and ipcRenderer modules for inter-process communication (IPC). To send a message from your web page to the main process, you can set up a main process handler with ipcMain.handle and then expose a function that calls ipcRenderer.invoke to trigger the handler in your preload script.*

Giải pháp cho vấn đề này là sử dụng các module ipcMain và ipcRenderer của Electron cho giao tiếp giữa các tiến trình (IPC). Để gửi một tin nhắn từ trang web của bạn đến main process, bạn có thể thiết lập một handler ở main process bằng ipcMain.handle và sau đó expose một hàm gọi ipcRenderer.invoke để kích hoạt handler trong preload script của bạn.

*To illustrate, we will add a global function to the renderer called ping() that will return a string from the main process.*

Để minh họa, chúng ta sẽ thêm một hàm toàn cục vào trình hiển thị có tên là ping(), hàm này sẽ trả về một chuỗi ký tự từ quá trình chính.

*First, set up the invoke call in your preload script:*

Đầu tiên, thiết lập lệnh invoketrong file preload.js của bạn:

// preload.js

const { contextBridge, ipcRenderer } = require('electron')

contextBridge.exposeInMainWorld('versions', {

node: () => process.versions.node,

chrome: () => process.versions.chrome,

electron: () => process.versions.electron,

ping: () => ipcRenderer.invoke('ping')

// we can also expose variables, not just functions

})

***IPC SECURITY***

*Notice how we wrap the ipcRenderer.invoke('ping') call in a helper function rather than expose the ipcRenderer module directly via context bridge. You never want to directly expose the entire ipcRenderer module via preload. This would give your renderer the ability to send arbitrary IPC messages to the main process, which becomes a powerful attack vector for malicious code.*

Lưu ý cách chúng ta wrap lệnh gọi ipcRenderer.invoke('ping') trong một helper function thay vì expose trực tiếp module ipcRenderer qua context bridge. Bạn không bao giờ muốn expose toàn bộ module ipcRenderer trực tiếp qua preload. Điều này sẽ cung cấp cho renderer khả năng gửi các thông điệp IPC tùy ý đến main process, trở thành một attack vector mạnh mẽ cho mã độc.

*Then, set up your handle listener in the main process. We do this before loading the HTML file so that the handler is guaranteed to be ready before you send out the invoke call from the renderer.*

Sau đó, thiết lập trình xử lý sự kiện handle trong quá trình chính. Chúng tôi làm điều này trước khi tải file HTML để đảm bảo rằng handler đã sẵn sàng trước khi bạn gửi lệnh gọi invoke từ trình renderer.

const { app, BrowserWindow, ipcMain } = require('electron/main')

const path = require('node:path')

const createWindow = () => {

const win = new BrowserWindow({

width: 800,

height: 600,

webPreferences: {

preload: path.join(\_\_dirname, 'preload.js')

}

})

win.loadFile('index.html')

}

app.whenReady().then(() => {

ipcMain.handle('ping', () => 'pong')

createWindow()

})

*Once you have the sender and receiver set up, you can now send messages from the renderer to the main process through the 'ping' channel you just defined.*

Khi đã thiết lập sender và receiver, bạn có thể gửi tin nhắn từ renderer đến main process thông qua kênh 'ping' mà bạn vừa định nghĩa.

// renderer.js

const func = async () => {

const response = await window.versions.ping()

console.log(response) // prints out 'pong'

}

func()

***INFO***

*For more in-depth explanations on using the ipcRenderer and ipcMain modules, check out the full* [*Inter-Process Communication*](https://www.electronjs.org/docs/latest/tutorial/ipc) *guide.*

Để hiểu sâu hơn về cách sử dụng module ipcRenderer và ipcMain, bạn hãy xem toàn bộ hướng dẫn về Inter-Process Communication.

***Summary***

*A preload script contains code that runs before your web page is loaded into the browser window. It has access to both DOM APIs and Node.js environment, and is often used to expose privileged APIs to the renderer via the contextBridge API.*

Một preload script chứa mã chạy trước khi trang web của bạn được tải vào cửa sổ trình duyệt. Nó có quyền truy cập cả DOM APIs và môi trường Node.js, và thường được dùng để hiển thị các API đặc quyền cho renderer thông qua contextBridge API.

*Because the main and renderer processes have very different responsibilities, Electron apps often use the preload script to set up inter-process communication (IPC) interfaces to pass arbitrary messages between the two kinds of processes.*

Vì main process và renderer process có những trách nhiệm rất khác nhau, các ứng dụng Electron thường sử dụng preload script để thiết lập giao diện IPC (inter-process communication) nhằm truyền tải các thông điệp tùy ý giữa hai loại tiến trình này.

*In the next part of the tutorial, we will be showing you resources on adding more functionality to your app, then teaching you how to distribute your app to users.*

Trong phần tiếp theo của hướng dẫn, chúng tôi sẽ giới thiệu các tài nguyên để bạn bổ sung thêm tính năng cho ứng dụng, sau đó hướng dẫn bạn cách phân phối ứng dụng đến người dùng.

***Adding Features***

**Thêm tính năng**

*Adding application complexity*

Tăng độ phức tạp của ứng dụng

*If you have been following along, you should have a functional Electron application with a static user interface. From this starting point, you can generally progress in developing your app in two broad directions:*

Nếu bạn đã theo dõi từ đầu, giờ đây bạn hẳn đã có một ứng dụng Electron hoạt động với giao diện người dùng tĩnh. Từ điểm khởi đầu này, bạn có thể phát triển ứng dụng theo hai hướng chính:

1. *Adding complexity to your renderer process' web app code*
2. Tăng độ phức tạp cho mã nguồn ứng dụng web của renderer process
3. *Deeper integrations with the operating system and Node.js*
4. Tích hợp sâu hơn với hệ điều hành và Node.js

*It is important to understand the distinction between these two broad concepts. For the first point, Electron-specific resources are not necessary. Building a pretty to-do list in Electron is just pointing your Electron BrowserWindow to a pretty to-do list web app. Ultimately, you are building your renderer's UI using the same tools (HTML, CSS, JavaScript) that you would on the web. Therefore, Electron's docs will not go in-depth on how to use standard web tools.*

Điều quan trọng là phải hiểu sự khác biệt giữa hai khái niệm rộng này. Đối với điểm đầu tiên, không cần thiết phải có các tài nguyên dành riêng cho Electron. Việc xây dựng một ứng dụng danh sách việc cần làm (to-do list) đẹp mắt trong Electron chỉ đơn giản là trỏ BrowserWindow của Electron đến một ứng dụng web to-do list đẹp mắt mà thôi. Cuối cùng, bạn đang xây dựng UI của renderer bằng chính các công cụ (HTML, CSS, JavaScript) mà bạn sẽ dùng trên web. Do đó, tài liệu của Electron sẽ không đi sâu vào cách sử dụng các công cụ web tiêu chuẩn.

*On the other hand, Electron also provides a rich set of tools that allow you to integrate with the desktop environment, from creating tray icons to adding global shortcuts to displaying native menus. It also gives you all the power of a Node.js environment in the main process. This set of capabilities separates Electron applications from running a website in a browser tab, and are the focus of Electron's documentation.*

Mặt khác, Electron cũng cung cấp một bộ công cụ phong phú cho phép bạn tích hợp với môi trường desktop, từ việc tạo các tray icon, thêm các global shortcut cho đến hiển thị các native menu. Nó cũng mang lại cho bạn toàn bộ sức mạnh của môi trường Node.js trong main process. Tập hợp các khả năng này chính là điều tách biệt các ứng dụng Electron khỏi việc chạy một trang web trong một tab trình duyệt, và là trọng tâm của tài liệu Electron.

*How-to examples*

Ví dụ hướng dẫn

*Electron's documentation has many tutorials to help you with more advanced topics and deeper operating system integrations. To get started, check out the How-To Examples doc.*

Tài liệu của Electron có rất nhiều hướng dẫn để giúp bạn với các chủ đề nâng cao hơn và tích hợp sâu hơn với hệ điều hành. Để bắt đầu, hãy xem tài liệu How-To Examples.

***LET US KNOW IF SOMETHING IS MISSING!***

**HÃY CHO CHÚNG TÔI BIẾT NẾU CÓ GÌ THIẾU SÓT!**

*If you can't find what you are looking for, please let us know on* [*GitHub*](https://github.com/electron/website/issues/new) *or in our* [*Discord server*](https://discord.gg/electronjs)*!*

Nếu bạn không tìm thấy điều mình cần, hãy cho chúng tôi biết trên GitHub hoặc trong Discord server của chúng tôi nhé!

*What's next?*

Bước tiếp theo là gì?

*For the rest of the tutorial, we will be shifting away from application code and giving you a look at how you can get your app from your developer machine into end users' hands.*

Phần còn lại của hướng dẫn này, chúng ta sẽ tạm gác lại phần code ứng dụng để xem cách bạn có thể đưa sản phẩm của mình từ máy dev đến tay người dùng cuối.

***Packaging Your Application***

**Đóng gói ứng dụng của bạn**

*Learning goals*

Mục tiêu học tập

*In this part of the tutorial, we'll be going over the basics of packaging and distributing your app with Electron Forge.*

Trong phần này của hướng dẫn, chúng ta sẽ cùng tìm hiểu những kiến thức cơ bản về cách đóng gói (packaging) và phân phối (distributing) ứng dụng của bạn bằng Electron Forge.

*Using Electron Forge*

Sử dụng Electron Forge

*Electron does not have any tooling for packaging and distribution bundled into its core modules. Once you have a working Electron app in dev mode, you need to use additional tooling to create a packaged app you can distribute to your users (also known as a distributable). Distributables can be either installers (e.g. MSI on Windows) or portable executable files (e.g. .app on macOS).*

Electron không tích hợp sẵn bất kỳ công cụ đóng gói và phân phối nào vào các module cốt lõi của nó. Khi bạn đã có một ứng dụng Electron hoạt động ở chế độ dev, bạn cần sử dụng các công cụ bổ sung để tạo một ứng dụng đã đóng gói mà bạn có thể phân phối cho người dùng của mình (còn được gọi là distributable). Các distributable này có thể là trình cài đặt (ví dụ: MSI trên Windows) hoặc các file thực thi di động (ví dụ: .app trên macOS).

*Electron Forge is an all-in-one tool that handles the packaging and distribution of Electron apps. Under the hood, it combines a lot of existing Electron tools (e.g. @electron/packager, @electron/osx-sign, electron-winstaller, etc.) into a single interface so you do not have to worry about wiring them all together.*

Electron Forge là một công cụ "tất cả trong một" chuyên xử lý việc đóng gói và phân phối các ứng dụng Electron. Về cơ bản, nó kết hợp nhiều công cụ Electron hiện có (ví dụ: @electron/packager, @electron/osx-sign, electron-winstaller, v.v.) vào một giao diện duy nhất, giúp bạn không cần phải lo lắng về việc kết nối tất cả chúng lại với nhau.

*Importing your project into Forge*

Import Project của bạn vào Forge

*You can install Electron Forge's CLI in your project's devDependencies and import your existing project with a handy conversion script.*

Bạn có thể cài đặt CLI của Electron Forge vào devDependencies trong project của mình và import project hiện có bằng một conversion script tiện lợi.

npm install --save-dev @electron-forge/cli

npx electron-forge import

*Once the conversion script is done, Forge should have added a few scripts to your package.json file.*

Khi conversion script hoàn tất, Forge sẽ thêm một vài script vào file package.json của bạn.

//...

"scripts": {

"start": "electron-forge start",

"package": "electron-forge package",

"make": "electron-forge make"

},

//...

***CLI DOCUMENTATION***

*For more information on make and other Forge APIs, check out the Electron Forge CLI documentation.*

Để biết thêm thông tin về make và các API khác của Forge, hãy xem tài liệu Electron Forge CLI.

*You should also notice that your package.json now has a few more packages installed under devDependencies, and a new forge.config.js file that exports a configuration object. You should see multiple makers (packages that generate distributable app bundles) in the pre-populated configuration, one for each target platform.*

Bạn cũng sẽ thấy rằng package.json của bạn giờ đây đã cài đặt thêm một vài gói trong devDependencies, cùng với một file mới forge.config.js dùng để export một configuration object. Bạn sẽ thấy nhiều maker (các gói tạo ra các app bundle có thể phân phối được) trong cấu hình được điền sẵn, mỗi maker tương ứng với một nền tảng mục tiêu.

*Creating a distributable*

Tạo bản phân phối (Distributable)

*To create a distributable, use your project's new make script, which runs the electron-forge make command.*

Để tạo một bản phân phối, bạn hãy sử dụng script make mới của project, script này sẽ chạy lệnh electron-forge make.

npm run make

*This make command contains two steps:*

Lệnh make này bao gồm hai bước:

1. *It will first run electron-forge package under the hood, which bundles your app code together with the Electron binary. The packaged code is generated into a folder.*
2. Đầu tiên, nó sẽ chạy electron-forge package ngầm bên dưới, công đoạn này sẽ đóng gói code ứng dụng của bạn cùng với Electron binary. Mã nguồn đã được đóng gói sẽ được tạo ra trong một thư mục.
3. *It will then use this packaged app folder to create a separate distributable for each configured maker.*
4. Sau đó, nó sẽ sử dụng thư mục ứng dụng đã đóng gói này để tạo một bản phân phối (distributable) riêng biệt cho mỗi maker đã được cấu hình.

*After the script runs, you should see an out folder containing both the distributable and a folder containing the packaged application code.*

Sau khi script chạy xong, bạn sẽ thấy một thư mục out chứa cả bản distributable và một thư mục chứa code ứng dụng đã được đóng gói.

out/

├── out/make/zip/darwin/x64/my-electron-app-darwin-x64-1.0.0.zip

├── ...

└── out/my-electron-app-darwin-x64/my-electron-app.app/Contents/MacOS/my-electron-app

*The distributable in the out/make folder should be ready to launch! You have now created your first bundled Electron application.*

Bản distributable trong thư mục out/make đã sẵn sàng để khởi chạy! Vậy là bạn đã tạo thành công ứng dụng Electron đầu tiên của mình.

*DISTRIBUTABLE FORMATS*

ĐỊNH DẠNG DISTRIBUTABLE

*Electron Forge can be configured to create distributables in different OS-specific formats (e.g. DMG, deb, MSI, etc.). See Forge's Makers documentation for all configuration options.*

Electron Forge có thể được cấu hình để tạo các bản distributable theo nhiều định dạng đặc thù của từng hệ điều hành (ví dụ: DMG, deb, MSI, v.v.). Để biết tất cả các tùy chọn cấu hình, hãy xem tài liệu Makers của Forge.

*CREATING AND ADDING APPLICATION ICONS*

TẠO VÀ THÊM BIỂU TƯỢNG ỨNG DỤNG (APPLICATION ICONS)

*Setting custom application icons requires a few additions to your config. Check out* [*Forge's icon tutorial*](https://www.electronforge.io/guides/create-and-add-icons) *for more information.*

Để đặt các biểu tượng ứng dụng tùy chỉnh, bạn cần thêm một vài điều chỉnh vào file config của mình. Hãy xem [hướng dẫn về biểu tượng của Forge](https://www.electronforge.io/guides/create-and-add-icons) để biết thêm thông tin chi tiết.

*PACKAGING WITHOUT ELECTRON FORGE*

ĐÓNG GÓI KHÔNG DÙNG ELECTRON FORGE

*If you want to manually package your code, or if you're just interested understanding the mechanics behind packaging an Electron app, check out the full Application Packaging documentation.*

Nếu bạn muốn tự đóng gói code của mình, hoặc đơn giản là bạn tò mò muốn hiểu cơ chế đằng sau việc đóng gói một ứng dụng Electron, hãy xem toàn bộ tài liệu Application Packaging nhé.

*Important: signing your code*

Quan trọng: Ký mã nguồn (Code Signing)

*In order to distribute desktop applications to end users, we highly recommend that you code sign your Electron app. Code signing is an important part of shipping desktop applications, and is mandatory for the auto-update step in the final part of the tutorial.*

Để phân phối ứng dụng desktop đến người dùng cuối, chúng tôi khuyến nghị mạnh mẽ bạn nên ký mã nguồn (code sign) cho ứng dụng Electron của mình. Ký mã nguồn là một phần quan trọng trong việc phát hành ứng dụng desktop và là bắt buộc cho bước tự động cập nhật trong phần cuối cùng của hướng dẫn này.

*Code signing is a security technology that you use to certify that a desktop app was created by a known source. Windows and macOS have their own OS-specific code signing systems that will make it difficult for users to download or launch unsigned applications.*

Ký mã nguồn (Code signing) là một công nghệ bảo mật mà bạn dùng để xác nhận rằng một ứng dụng desktop được tạo ra bởi một nguồn đáng tin cậy. Windows và macOS có các hệ thống ký mã nguồn riêng biệt, đặc thù cho từng hệ điều hành, những hệ thống này sẽ gây khó khăn cho người dùng khi tải xuống hoặc khởi chạy các ứng dụng chưa được ký.

*On macOS, code signing is done at the app packaging level. On Windows, distributable installers are signed instead. If you already have code signing certificates for Windows and macOS, you can set your credentials in your Forge configuration.*

Trên macOS, việc ký mã nguồn (code signing) được thực hiện ở cấp độ đóng gói ứng dụng. Còn trên Windows, các trình cài đặt distributable sẽ được ký thay thế. Nếu bạn đã có chứng chỉ ký mã nguồn cho Windows và macOS, bạn có thể thiết lập thông tin xác thực của mình trong cấu hình của Forge.

***INFO***

**INFO**

*For more information on code signing, check out the* [*Signing macOS Apps*](https://www.electronforge.io/guides/code-signing) *guide in the Forge docs.*

Để biết thêm thông tin về ký mã nguồn (code signing), bạn hãy xem hướng dẫn Signing macOS Apps trong tài liệu của Forge nhé.

Windows

// forge.config.js

module.exports = {

// ...

makers: [

{

name: '@electron-forge/maker-squirrel',

config: {

certificateFile: './cert.pfx',

certificatePassword: process.env.CERTIFICATE\_PASSWORD

}

}

]

// ...

}

// forge.config.js

module.exports = {

packagerConfig: {

osxSign: {},

// ...

osxNotarize: {

tool: 'notarytool',

appleId: process.env.APPLE\_ID,

appleIdPassword: process.env.APPLE\_PASSWORD,

teamId: process.env.APPLE\_TEAM\_ID

}

// ...

}

}

***Summary***

**Tóm tắt**

*Electron applications need to be packaged to be distributed to users. In this tutorial, you imported your app into Electron Forge and configured it to package your app and generate installers.*

Các ứng dụng Electron cần được đóng gói (packaged) để có thể phân phối đến người dùng. Trong hướng dẫn này, bạn đã import ứng dụng của mình vào Electron Forge và cấu hình nó để đóng gói ứng dụng cũng như tạo ra các trình cài đặt (installer).

*In order for your application to be trusted by the user's system, you need to digitally certify that the distributable is authentic and untampered by code signing it. Your app can be signed through Forge once you configure it to use your code signing certificate information.*

Để ứng dụng của bạn được hệ thống của người dùng tin cậy, bạn cần chứng nhận kỹ thuật số rằng bản distributable là chính hãng và chưa bị can thiệp bằng cách ký mã nguồn (code signing) nó. Ứng dụng của bạn có thể được ký thông qua Forge ngay khi bạn cấu hình nó để sử dụng thông tin chứng chỉ ký mã nguồn của bạn.

***Publishing and Updating***

**Xuất bản và Cập nhật**

***Learning goals***

**Mục tiêu học tập**

*If you've been following along, this is the last step of the tutorial! In this part, you will publish your app to GitHub releases and integrate automatic updates into your app code.*

Nếu bạn đã theo dõi từ đầu, đây chính là bước cuối cùng của hướng dẫn! Trong phần này, bạn sẽ publish ứng dụng của mình lên GitHub Releases và tích hợp tính năng tự động cập nhật vào code ứng dụng.

***Using update.electronjs.org***

**Sử dụng update.electronjs.org**

*The Electron maintainers provide a free auto-updating service for open-source apps at https://update.electronjs.org. Its requirements are:*

Các maintainer của Electron cung cấp một dịch vụ tự động cập nhật miễn phí cho các ứng dụng mã nguồn mở tại địa chỉ <https://update.electronjs.org/>. Các yêu cầu để sử dụng dịch vụ này là:

* Your app runs on macOS or Windows
* Ứng dụng của bạn chạy trên macOS hoặc Windows
* Your app has a public GitHub repository
* Ứng dụng của bạn có một GitHub repository công khai
* Builds are published to GitHub releases
* Các bản build được publish lên GitHub Releases
* Builds are code signed (macOS only)
* Các bản build được ký mã nguồn (code signed) (chỉ áp dụng cho macOS)

*At this point, we'll assume that you have already pushed all your code to a public GitHub repository.*

Tại thời điểm này, chúng ta sẽ giả định rằng bạn đã push tất cả code của mình lên một GitHub repository công khai.

***ALTERNATIVE UPDATE SERVICES***

**CÁC DỊCH VỤ CẬP NHẬT THAY THẾ**

*If you're using an alternate repository host (e.g. GitLab or Bitbucket) or if you need to keep your code repository private, please refer to our* [*step-by-step*](https://www.electronjs.org/docs/latest/tutorial/updates) *guide on hosting your own Electron update server.*

Nếu bạn đang sử dụng một repository host khác (ví dụ: GitLab hoặc Bitbucket) hoặc nếu bạn cần giữ repository code của mình ở chế độ riêng tư, vui lòng tham khảo hướng dẫn chi tiết của chúng tôi về cách tự host server cập nhật Electron của riêng bạn.

***Publishing a GitHub release***

**Đăng tải GitHub Release**

*Electron Forge has* ***Publisher*** *plugins that can automate the distribution of your packaged application to various sources. In this tutorial, we will be using the GitHub Publisher, which will allow us to publish our code to GitHub releases.*

Electron Forge có các plugin **Publisher** có thể tự động hóa việc phân phối ứng dụng đã đóng gói của bạn lên nhiều nguồn khác nhau. Trong hướng dẫn này, chúng ta sẽ sử dụng GitHub Publisher, nó sẽ cho phép chúng ta publish code của mình lên GitHub releases.

***Generating a personal access token***

**Tạo Personal Access Token**

*Forge cannot publish to any repository on GitHub without permission. You need to pass in an authenticated token that gives Forge access to your GitHub releases. The easiest way to do this is to create a new personal access token (PAT) with the public\_repo scope, which gives write access to your public repositories.* ***Make sure to keep this token a secret****.*

Forge không thể publish lên bất kỳ repository nào trên GitHub mà không có quyền. Bạn cần truyền vào một token đã được xác thực để cấp quyền truy cập GitHub releases cho Forge. Cách dễ nhất để làm điều này là [tạo một personal access token (PAT) mới](https://github.com/settings/tokens/new) với scope public\_repo, vốn cấp quyền ghi vào các repository công khai của bạn. **Hãy đảm bảo giữ bí mật token này nhé**.

***Setting up the GitHub Publisher***

**Cài đặt GitHub Publisher**

***Installing the module***

**Cài đặt module**

*Forge's* [*GitHub Publisher*](https://www.electronforge.io/config/publishers/github) *is a plugin that needs to be installed in your project's devDependencies:*

GitHub Publisher của Forge là một plugin cần được cài đặt vào devDependencies của project bạn:

npm install --save-dev @electron-forge/publisher-github

***Configuring the publisher in Forge***

**Cấu hình Publisher trong Forge**

*Once you have it installed, you need to set it up in your Forge configuration. A full list of options is documented in the Forge's PublisherGitHubConfig API docs.*

Sau khi cài đặt xong, bạn cần thiết lập nó trong cấu hình của Forge. Danh sách đầy đủ các tùy chọn đã được tài liệu hóa trong API docs PublisherGitHubConfig của Forge.

// forge.config.js

module.exports = {

publishers: [

{

name: '@electron-forge/publisher-github',

config: {

repository: {

owner: 'github-user-name',

name: 'github-repo-name'

},

prerelease: false,

draft: true

}

}

]

}

***DRAFTING RELEASES BEFORE PUBLISHING***

**SOẠN THẢO RELEASE TRƯỚC KHI PUBLISH**

*Notice that you have configured Forge to publish your release as a draft. This will allow you to see the release with its generated artifacts without actually publishing it to your end users. You can manually publish your releases via GitHub after writing release notes and double-checking that your distributables work.*

Hãy để ý rằng bạn đã cấu hình Forge để publish release dưới dạng bản nháp (draft). Điều này sẽ cho phép bạn xem release cùng với các artifact đã được tạo mà không thực sự publish nó cho người dùng cuối. Bạn có thể tự publish release của mình thông qua GitHub sau khi viết ghi chú release và kiểm tra kỹ lại các bản distributable đã hoạt động.

***Setting up your authentication token***

**Thiết lập mã thông báo xác thực của bạn**

*You also need to make the Publisher aware of your authentication token. By default, it will use the value stored in the GITHUB\_TOKEN environment variable.*

Bạn cũng cần làm cho Publisher biết về mã thông báo xác thực của mình. Theo mặc định, nó sẽ sử dụng giá trị được lưu trữ trong biến môi trường GITHUB\_TOKEN.

***Running the publish command***

**Chạy lệnh Publish**

*Add Forge's publish command to your npm scripts.*

Hãy thêm [lệnh publish](https://www.electronforge.io/cli#publish) của Forge vào các npm scripts của bạn.

// package.json

//...

"scripts": {

"start": "electron-forge start",

"package": "electron-forge package",

"make": "electron-forge make",

"publish": "electron-forge publish"

},

//...

*This command will run your configured makers and publish the output distributables to a new GitHub release.*

Lệnh này sẽ chạy các maker đã được cấu hình của bạn và publish các bản distributable đầu ra lên một GitHub release mới.

npm run publish

*By default, this will only publish a single distributable for your host operating system and architecture. You can publish for different architectures by passing in the --arch flag to your Forge commands.*

Mặc định, lệnh này sẽ chỉ publish một bản distributable duy nhất cho hệ điều hành và kiến trúc máy chủ của bạn. Bạn có thể publish cho các kiến trúc khác bằng cách truyền cờ --arch vào các lệnh Forge của bạn.

*The name of this release will correspond to the version field in your project's package.json file.*

Tên của bản release này sẽ tương ứng với trường version trong file package.json của project bạn.

***TAGGING RELEASES***

**GẮN THẺ (TAGGING) CÁC RELEASE**

*Optionally, you can also tag your releases in Git so that your release is associated with a labeled point in your code history. npm comes with a handy npm version command that can handle the version bumping and tagging for you.*

Bạn có thể tùy chọn gắn thẻ các release của mình trong Git để release của bạn được liên kết với một điểm được gắn nhãn trong lịch sử code của bạn. npm đi kèm với một lệnh tiện lợi là npm version có thể xử lý việc tăng version và gắn thẻ giúp bạn.

***Bonus: Publishing in GitHub Actions***

**Bonus: Publish với GitHub Actions**

*Publishing locally can be painful, especially because you can only create distributables for your host operating system (i.e. you can't publish a Windows .exe file from macOS).*

Việc publish cục bộ có thể khá "đau đầu", đặc biệt là vì bạn chỉ có thể tạo các bản distributable cho hệ điều hành máy chủ của mình (tức là bạn không thể publish một file .exe của Windows từ macOS).

*A solution for this would be to publish your app via automation workflows such as* ***GitHub Actions****, which can run tasks in the cloud on Ubuntu, macOS, and Windows. This is the exact approach taken by Electron Fiddle. You can refer to Fiddle's* ***Build and Release pipeline*** *and* ***Forge configuration*** *for more details.*

Một giải pháp cho vấn đề này là publish ứng dụng của bạn thông qua các workflow tự động như **GitHub Actions**, nơi có thể chạy các tác vụ trên cloud ở các hệ điều hành Ubuntu, macOS và Windows. Đây chính xác là cách tiếp cận mà **Electron Fiddle** đã sử dụng. Bạn có thể tham khảo pipeline Build và Release cũng như **cấu hình Forge** của Fiddle để biết thêm chi tiết.

***Instrumenting your updater code***

**Triển khai code cập nhật**

*Now that we have a functional release system via GitHub releases, we now need to tell our Electron app to download an update whenever a new release is out. Electron apps do this via the* ***autoUpdater*** *module, which reads from an update server feed to check if a new version is available for download.*

Giờ đây, chúng ta đã có một hệ thống release hoạt động ổn định thông qua GitHub releases, việc cần làm tiếp theo là "bảo" ứng dụng Electron của chúng ta tự động tải xuống bản cập nhật mỗi khi có release mới. Các ứng dụng Electron thực hiện điều này thông qua module **autoUpdater**, module này sẽ đọc từ một feed của update server để kiểm tra xem có phiên bản mới nào sẵn sàng để tải về hay không.

*The update.electronjs.org service provides an updater-compatible feed. For example, Electron Fiddle v0.28.0 will check the endpoint at https://update.electronjs.org/electron/fiddle/darwin/v0.28.0 to see if a newer GitHub release is available.*

Dịch vụ update.electronjs.org cung cấp một feed tương thích với updater. Chẳng hạn, Electron Fiddle v0.28.0 sẽ kiểm tra endpoint tại https://update.electronjs.org/electron/fiddle/darwin/v0.28.0 để xem liệu có bản GitHub release nào mới hơn hay không.

*After your release is published to GitHub, the update.electronjs.org service should work for your application. The only step left is to configure the feed with the autoUpdater module.*

Sau khi bản release của bạn được publish lên GitHub, dịch vụ update.electronjs.org sẽ hoạt động cho ứng dụng của bạn. Bước cuối cùng còn lại là cấu hình feed với module autoUpdater.

*To make this process easier, the Electron team maintains the update-electron-app module, which sets up the autoUpdater boilerplate for update.electronjs.org in one function call — no configuration required. This module will search for the update.electronjs.org feed that matches your project's package.json "repository" field.*

Để làm cho quá trình này dễ dàng hơn, đội ngũ Electron duy trì module update-electron-app, module này sẽ thiết lập boilerplate của autoUpdater cho update.electronjs.org chỉ bằng một lệnh gọi hàm — không yêu cầu cấu hình. Module này sẽ tìm kiếm feed của update.electronjs.org khớp với trường "repository" trong file package.json của project bạn.

*First, install the module as a runtime dependency.*

Đầu tiên, hãy cài đặt module này như một runtime dependency.

npm install update-electron-app

*Then, import the module and call it immediately in the main process.*

Sau đó, hãy import module và gọi nó ngay lập tức trong main process.

// main.js

require('update-electron-app')()

*And that is all it takes! Once your application is packaged, it will update itself for each new GitHub release that you publish.*

Và chỉ cần vậy thôi! Khi ứng dụng của bạn đã được đóng gói, nó sẽ tự động cập nhật mỗi khi bạn publish một GitHub release mới.

***Summary***

**Tóm tắt**

*In this tutorial, we configured Electron Forge's GitHub Publisher to upload your app's distributables to GitHub releases. Since distributables cannot always be generated between platforms, we recommend setting up your building and publishing flow in a Continuous Integration pipeline if you do not have access to machines.*

Trong hướng dẫn này, chúng ta đã cấu hình GitHub Publisher của Electron Forge để tải các bản distributable của ứng dụng lên GitHub releases. Vì các bản distributable không phải lúc nào cũng có thể được tạo ra trên các nền tảng khác nhau, chúng tôi khuyến nghị nên thiết lập quy trình build và publish của bạn trong một pipeline Continuous Integration nếu bạn không có quyền truy cập vào các máy vật lý.

*Electron applications can self-update by pointing the autoUpdater module to an update server feed. update.electronjs.org is a free update server provided by Electron for open-source applications published on GitHub releases. Configuring your Electron app to use this service is as easy as installing and importing the update-electron-app module.*

Các ứng dụng Electron có thể tự cập nhật bằng cách trỏ module autoUpdater đến một feed từ update server. update.electronjs.org là một update server miễn phí do Electron cung cấp cho các ứng dụng mã nguồn mở được publish trên GitHub releases. Việc cấu hình ứng dụng Electron của bạn để sử dụng dịch vụ này rất dễ dàng, chỉ cần cài đặt và import module update-electron-app.

*If your application is not eligible for update.electronjs.org, you should instead deploy your own update server and configure the autoUpdater module yourself.*

Nếu ứng dụng của bạn không đủ điều kiện sử dụng update.electronjs.org, bạn nên tự triển khai update server của riêng mình và tự cấu hình module autoUpdater.

***🌟 YOU'RE DONE!***

**🌟 HOÀN THÀNH!**

*From here, you have officially completed our tutorial to Electron. Feel free to explore the rest of our docs and happy developing! If you have questions, please stop by our community* [***Discord server***](https://discord.gg/electronjs)*.*

Từ đây, bạn đã chính thức hoàn thành hướng dẫn về Electron của chúng tôi. Hãy thoải mái khám phá các tài liệu còn lại và happy developing! Nếu có bất kỳ câu hỏi nào, đừng ngần ngại ghé thăm [**Discord server**](https://discord.gg/electronjs) của cộng đồng chúng tôi nhé.