**Introduction**

My name is Evgeny Zakharenka. My aim for today’s presentation is to give you information about Electron.

I am going to divide my report into 6 areas. Such as:

what is Electron.js

what are Electron’s key features

how Electron’s architecture and working process look like

Advantages & Disadvantages of using an Electron

Electron Alternatives

Electron App Examples

Let’s begin.

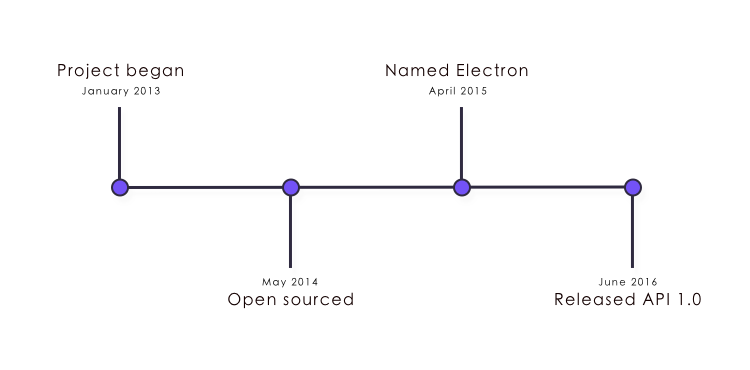
**What is Electron?**

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.

**A little bit of history of Electron.js**

The journey of Electron.js started in January 2013 with the search for a tool to build a cross-platform text editor on which the user can work with technologies like JavaScript, HTML, and CSS.

It was founded on 15th July 2013, intended to make the cross-platform development easier for the creation of “Atom”. It was initially known as Atom Shell. Have a look at the slide to get a more precise idea about the journey of Electron.js.



Let’s now talk about **Key features of Electron.js**

Assuming the above definition have solved your query on what is Electron.js, let’s move onto its features.

Any web application you have written can run on Electron.js. Similarly, any Node.JS application you write can utilize this technology.

Electron JS uses web technologies like simple HTML, CSS, and JavaScript. It does not require native skills unless you want to do something advanced. It can be designed for a single browser. Its file system belongs to Node.js APIs and works on Linux, Mac OS X, Windows.

It uses npm module which is used widely for JavaScript. It consists of a native menu for dialogs and notifications. The windows installers do not need any configuration.

It also has the facility of auto-updating and crash reporting on Windows and Mac with Squirrel. The crash reports are submitted to the remote server for further analysis. The content tracing activities like debugging and profiling are looked after by Chromium.

**Why should I care about Electron?**

The hard and tedious parts of creating a desktop application are the simplification of packaging, installation, updating, providing support for native menus, notifications, dialogs and in the end optimizing the app crash reports.

Electron JS pretty much takes care of all these essential steps so the user can focus on the core of his or her application.

When we are writing an application for a web browser, we are basically writing code that will be executed on someone else’s computer. We are not aware what browsers our target users will be using. It can be the latest version of Chrome or an outdated version of Internet Explorer.

Hence, we are not left with much of a choice but to be conservative in the technologies that we choose to implement and the kind of code that we need to draft.

When you build your applications with Electron, you’re packaging a particular version of Chromium and Node.JS, so you can rely on whatever features are available in those versions.

**A quick look at the architecture**

Now let’s shed some light on what Electron.js’s architecture is like. If you say Electron.js is a pizza and Node.JS is the base, then Chrome is the cheese and V8 JavaScript Engine is the topping.

*Libchromiumcontent*

Chrome is an open-source web browser built by Google, which provides a tabbed window manager or shell for the web. It has a minimalist user interface and uses V8 as the JavaScript engine and blink as the layout engine. Libchromiumcontent is Chromium’s rendering library, which is an open source foundation for Google’s browser Chrome.

*Node.JS*

Node.JS is an open source JavaScript runtime which uses V8 JavaScript engine. It enables you to run JavaScript outside the browser and also provides an interactive shell where you can execute raw JavaScript code.

The Node.JS project was initially released in 2009 as an open source, cross-platform runtime for developing server-side applications using JavaScript. Node.js comes bundled with a package manager called npm, which is the largest ecosystem of open source libraries.

*V8 JavaScript Engine*

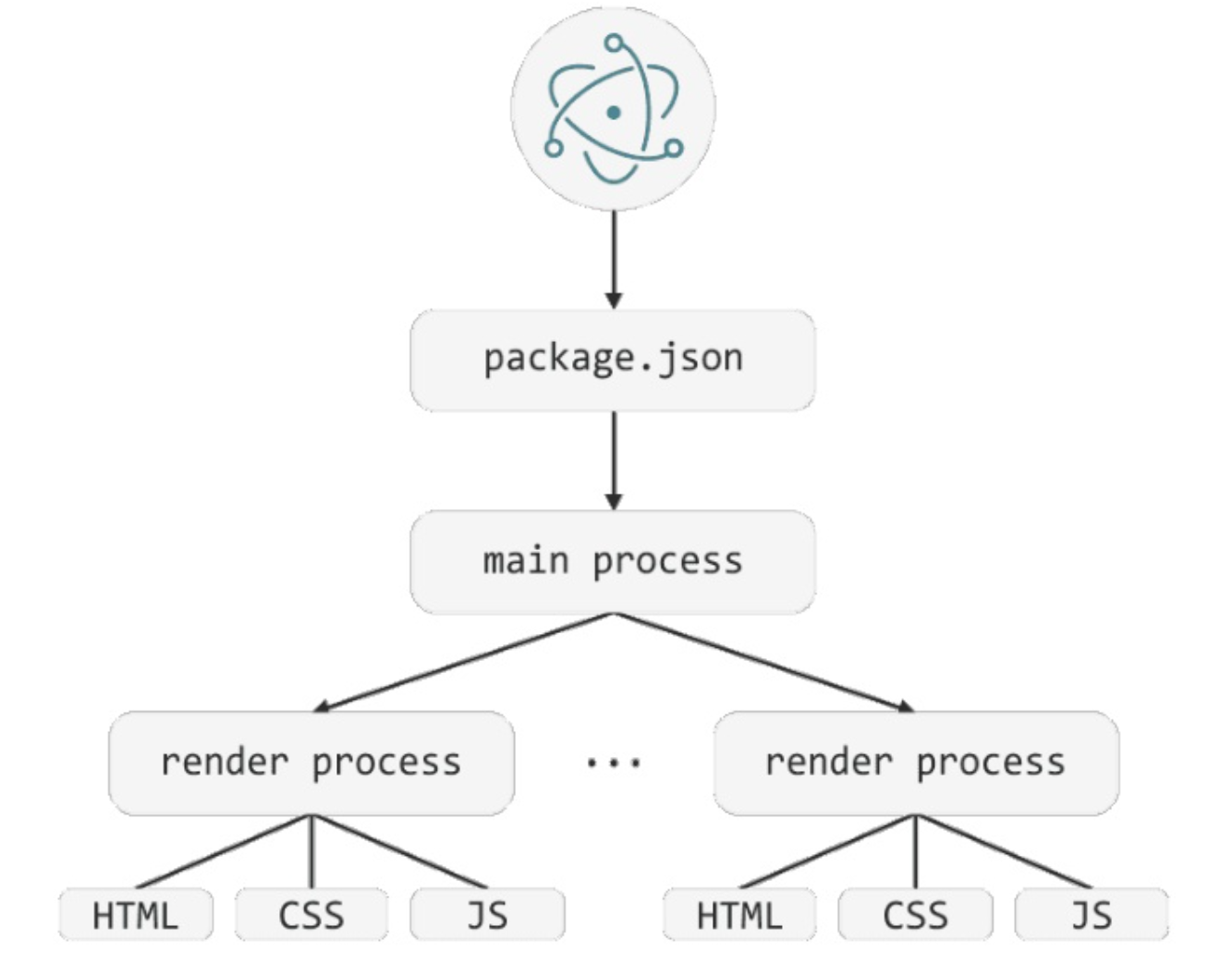
V8 JavaScript Engine is also an open-source JavaScript engine developed by Google, written in C++ and JavaScript.

**What is Electron.js’s working process like?**

After getting to know what Electron JS is and carefully analyzing its architecture, let’s understand how it works.

Thing to remeber: Electron applications have two mandatory processes, the main process, and the rendering process.

Each process has a different role to enact. Bootstrapping the application is performed by the main process. It can withstand other application lifecycle events like starting up, quitting, preparing to quit and other lightweight tasks like going to the background and coming to the foreground.



On the other hand, the rendering process is spawned by the main process. The render processes will display the UI of the application. Each process takes advantage of Chromium’s multiprocess architecture and runs on its own thread.

**Executing your first Electron.js application – “Hello World” installation**

Create a new folder in your file system.

To create a basic application you need three types of files: package.json, main.js, and index.html

To create the package.json file with `yarn init` command:

{

"name": "electron",

"version": "2.0.0",

"main": "index.js",

"license": "MIT"

}

This JSON file points to the app’s main file and lists its details and dependencies.

Now, in the package.json file, change the main to main.js file and add the following script:

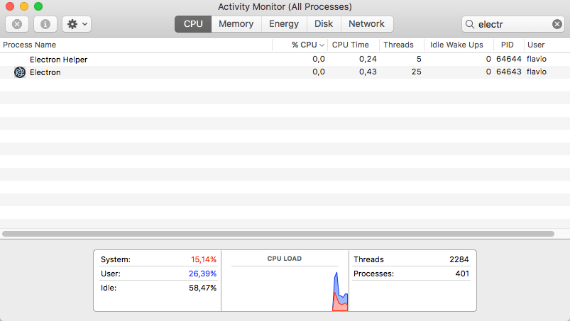
"scripts": {

"start": "electron ."

}

Install electron: yarn add --dev electron

Now it can be started by: yarn start

Because you have not added any code yet, you will get this:

Now it’s time to add code into the files and create an application of Hello World.

Add the following script to the main.js file:

'use strict'

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

const path = require('path')

const url = require('url')

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

// Create the browser window.

const win = new BrowserWindow({ width: 1000, height: 800 })

// and load the index.html of the app.

win.loadURL(

url.format({

pathname: path.join(\_\_dirname, 'index.html'),

protocol: 'file:',

slashes: true

})

)

})

This file starts the app and creates a browser window to render HTML as well as the app’s main process. It is responsible for the integration and interaction with the operating system GUI.

You can use any JavaScript library like Angular, React, JQuery etc. for the renderer process.

Create the index.html file:

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Hello World!</title>

</head>

<body>

<h1>Hello World!</h1>

We are using node <script>document.write(process.versions.node)</script>,

Chrome <script>document.write(process.versions.chrome)</script>,

and Electron <script>document.write(process.versions.electron)</script>.

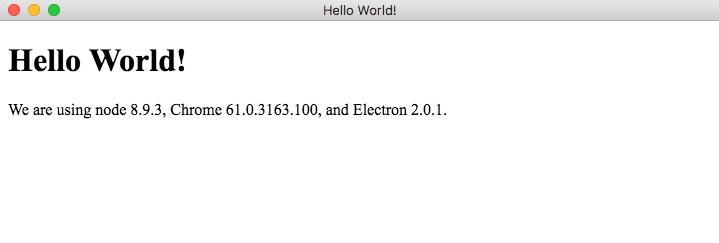
</body>

</html>

Index.html is a web page to render. This is the app’s renderer process.

Once you have created your initial main.js, index.html, and package.json files, you can try your app by running ‘yarn start’ from your application’s directory.

When you run your files, the app window pops up and shows Hello World.



It is a simple application, and the application exits when you close the window. Creating this application will give you a more practical idea of what Electron JS is.

Let’s now turn to Electron Advantages & Disadvantages

**Electron Advantages**

*Building desktop apps with web technologies*,

*low barrier to entry*

*ease with finding developers*

Electron allows to build desktop apps with JavaScript. To be more specific, developers use web technologies like HTML, CSS, and JavaScript.

*Cross Platform* With Electron, developers build cross platform apps. It means that one app works on Mac, Windows, and Linux. It eliminates the need of having skills in technologies that are used to develop for these platforms natively.

The desktop apps you build are cross-platform and will work on Windows, Linux, and macOS without requiring multiple codebases

You write code in regular old HTML, CSS, and JavaScript and *use whatever tools and web frameworks/libraries you are already familiar with*

*Debugging the renderer* Electron includes default Chrome Dev Tools Extension.

*Your web code can access native system APIs and pretty much anything else on the computer*

*Auto Update* Installation and auto-updating are all provided to us for free

*Your code and related content can be local to your app or it can be served remotely from a server similar to a hybrid app*

Why is that important?

Electron allows businesses to speed up time-to-market (cross-platform development) and use talents they already have or can hire fairly quickly (using web technologies).

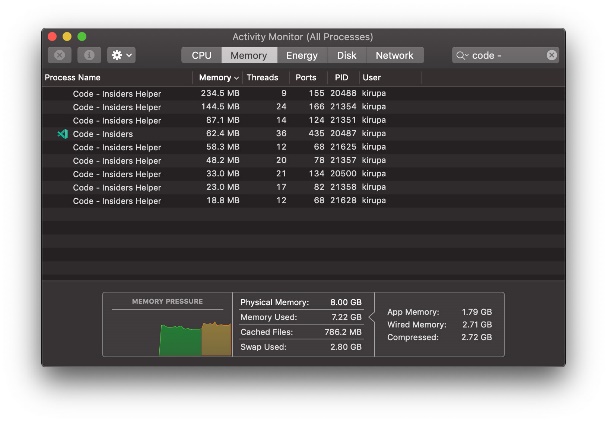
Web developers, even if they’re not specialized in Electron, will most likely catch it up quickly, especially if they have experience with Node.js.

App development with Electron eliminates time and talent waste.

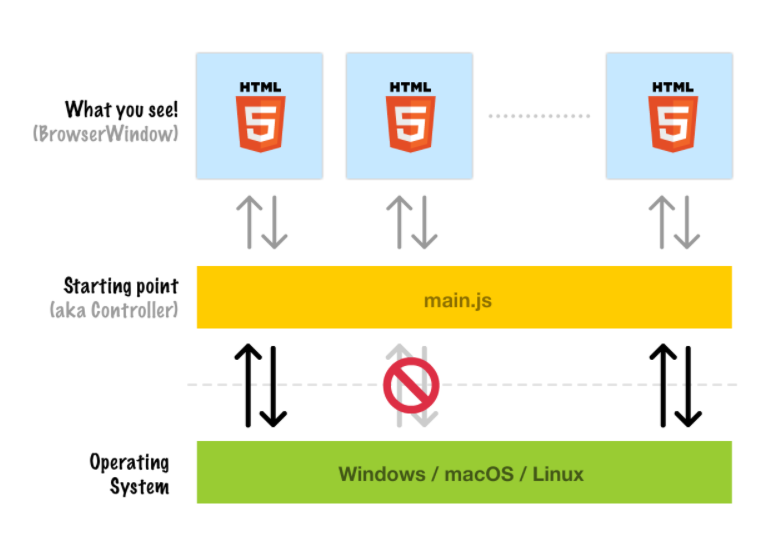
Let’s move on to **Electron Disadvantages**

*Slow start / slow running (Compared to native executables)*

*High Memory and Storage Usage (*If your computer is strapped for memory, an Electron app is not your friend. A full-fledged browser component is also heavy on the file size front. NodeJS isn't a lightweight either. Bundling both Chromium and NodeJS into each app adds a large fixed cost (**~130MB**) to how much storage space you need.*)*

**

*Security Challenges* (Electron allows your web code to access native APIs by removing the security guardrails that exist in the browser to prevent this access from working. This is a good thing to make your Electron apps more powerful and capable of doing the sorts of things only native apps could do. The downside is that poorly written or malicious code has the potential of doing a lot of harm. Electron does allow you to specify on a per BrowserWindow basis whether the web code loaded inside it needs system-level API access or not:



This level of control ensures that well informed developers can minimize any potential security risks. Poorly informed developers can maximize any potential security risks just as easily by allowing all BrowserWindow objects access to everything. As with many things on the desktop, the burden falls onto the end-user to keep themselves safe.

**Electron Alternatives﹣5 Best JavaScript Frameworks for Desktop Apps**

**#1 Electron**

**#2 NW.js**

**#3 AppJS**

**#4 Meteor**

**#5 Proton Native**

Electron vs nw.js:

Entry of Application: js vs html

Build System: libchromiumcontent vs Chromium

Node Integration: node\_bindings vs patching Chromium

Context: Multi-context vs Single-context

To sum up, JavaScript frameworks for desktop apps can be divided into three categories:

Frameworks that produce web browser hosted desktop apps, based on Node.js and Chromium (Electron, NW.js, AppJS).

Frameworks that need to be used with Cordova-like tools (Meteor).

Frameworks that use genuinely native components to build a desktop app (Proton Native).

It’s up to you which you choose, and it primarily depends on the type of project that you develop.

The significant advantage of electron solution is that there’s no need for a JavaScript web developer to learn new technology or language to build a desktop app.

If you’re a JavaScript developer, you’re going to need to learn a few relatively simple things on how Electron works and it’s API. You will most probably be able to set up your first Electron desktop application in just a few days.

**Electron App Example: Famous Desktop Apps Built With Electron JS**

***#1 WebTorrent desktop app***

Let’s face it – we all love torrents, whether they are downloaded on our mobile or desktop screens.

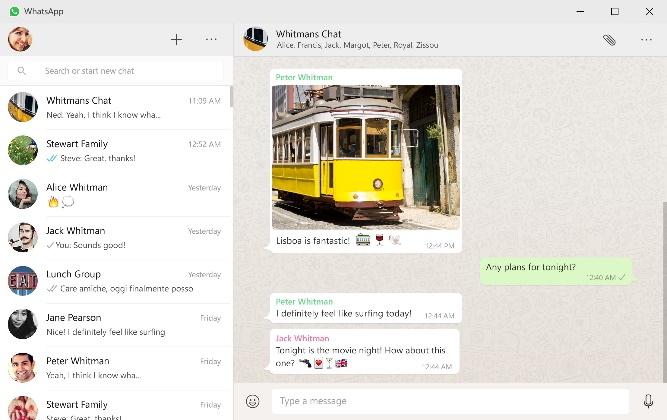
*Why does WebTorrent for desktop use Electron?*

Electron steps into action with the WebTorrent desktop version, making it as lightweight, ad-free and open source as possible. Also, it aids the streaming and acts as a hybrid client that connects the app to all the popular BitTorrent and WebTorrent networks.

**#2 WhatsApp Electron app**

Another one in the line of popular desktop apps using ElectronJS is WhatsApp – the most downloaded messenger app.

*Why does WhatsApp desktop app use Electron?*



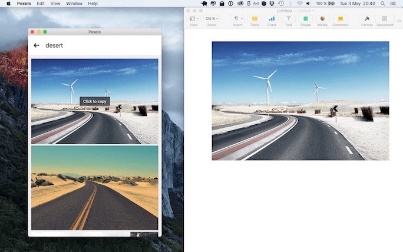
**#3 Pexels desktop app**

For a lot of writers, designers, and publicists, Pexels is a life-saver. As a platform that packs thousands of royalty-free stock images, it definitely needed improvement. And that is how the Pexels desktop app was born, obviously built with Electron JS.

*Why does Pexels desktop app use Electron?*

This app makes it easier than ever to copy a photo into your clipboard – with only one click. Then, you can paste the photo into any app and use it as a stock image for your article, infographic or social media post.

No downloads means no problems, and discovering new photos is easy thanks to the Pexels desktop app.

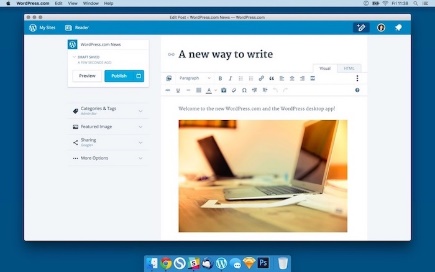
****

**#4 WordPress desktop app**

We all know about WordPress and launching it via our browsers. However, WordPress for desktop is something entirely new – a desktop app using the Electron framework that provides a seamless cross-platform experience, allowing users to focus on their content and design without any browser tabs acting as distractions.

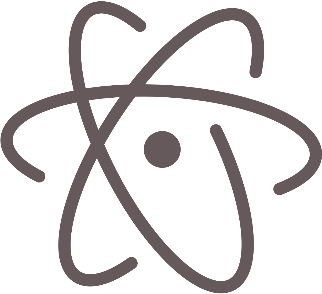
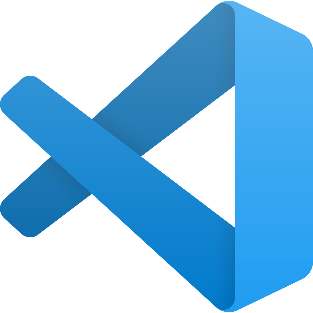
*Why does WordPress desktop app use Electron?*

Locally hosted and with minimal load times, WordPress for desktop is a desktop app using Electron as a framework, and JavaScript using React as the main language in this emerging desktop technology.

****

**Atom and Visual Studio Code: a free and open-source text and source code editor that makes collaborating on code easy.**

**Discord: a VoIP application and digital distribution platform designed for video gaming communities.**



**Conclusion**

Hope I helped you understand what Electron JS is along with its features, implementation and its fundamental tools.

Overall, Electron.js takes care of most of the dynamic applications which use our browser as a platform to deliver their embedded features and help us connect with a wider audience. After all, building and delivering the right applications that can help the users get their job done is the principal purpose of any open-source framework.

**Thank you for your attention!**

**https://github.com/zhenuua**

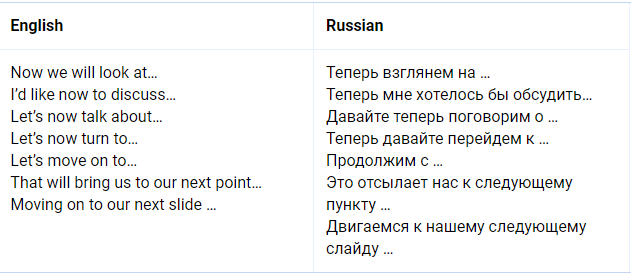
ДОПМАТЕРИАЛЫ

1. Начало (starting).
2. Главная часть (introduction and focusing attention).
   * первое (firstly);
   * второе (secondly);
   * третье (thirdly);
   * ...
   * наконец (lastly)
3. Заключение (conclusion).

Фразы

**Good morning** ladies and gentlemen….. My name is Yauheni Zakharenka and I am studying to be a junior frontend developer. Our **purpose** this morning is to hear a presentation, and to **discuss** it with all of you.

Главная часть



Now we will look at…  
I’d like now to discuss…  
Let’s now talk about…  
Let’s now turn to…  
Let’s move on to…  
That will bring us to our next point…  
Moving on to our next slide …

1. Введение What is electron?
2. история електрон

1)в кратце по поводу установки

2)Принцип работы, что можно делать.

3)преимущества и недостатки **Advantages & Disadvantages**

**Преимущества: кроссплатформенность**

**Недостатки:** проблемы с количеством используемой ими памяти, процессора и размером занимаемого ими места на диске.

4) сравнение с другими аналогами. Так же сравнение с нативными подходами для создания приложений

4)И примеры приложений,

5)и почему выбрали электрон в данных приложениях

6)заключение Finally let me just sum up today’s main topics…

Литеарутра:

<https://www.kirupa.com/apps/what_is_electron.htm>

<https://medium.com/shipmnts/why-not-to-build-an-electron-app-92b2f5a99d33>

<https://brainhub.eu/library/electron-desktop-app/>

<https://brainhub.eu/library/electron-app-examples/>

<https://brainhub.eu/library/electron-alternatives-javascript-frameworks-for-desktop-apps/>

<https://www.electronjs.org/>

https://brainhub.eu/library/what-is-electron-js/

http://tangiblejs.com/posts/nw-js-and-electron-compared-2016-edition