

# Développez des applications web desktop avec Electron de GitHub



《 DEVFEST NANTES 16 》

# Yvonnick Frin

- Consultant Zenika Nantes
- Développeur Front-End

 @YvonnickFrin

 @frinyvonnick



# Eric Briand

- Consultant Zenika Nantes
- Dev' touche à tout
- Co-organisateur du Docker Meetup Nantes

 @eric\_briand

 @ebriand



# Philippe Charrière

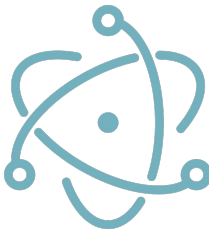
- Solution engineer GitHub
- Golo hardcore fan
- FP Guru

 @k33g\_org

 @k33g



# Sommaire



## 01 Qu'est-ce qu'Electron ?

Les motivations du projet, son historique, ses fonctionnalités

## 02 Ecosystème

Son utilisation, activité, et concurrents

## 03 Architecture

Qu'est-ce qui constitue une application Electron

## 04 Stratégie de build

Testing, packaging, update...

# Et après c'est à vous !



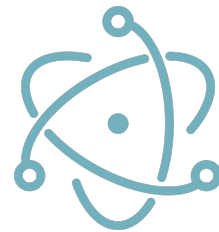
# Pourquoi faire une application desktop en 2016 ?



- Accéder aux fonctionnalités de l'OS hôte
- Confort d'utilisation (raccourcis clavier, menus...)
- Profiter des App Stores (OSX/Windows)
- Visibilité sur le poste



# Pourquoi utiliser les technologies web pour du desktop ?

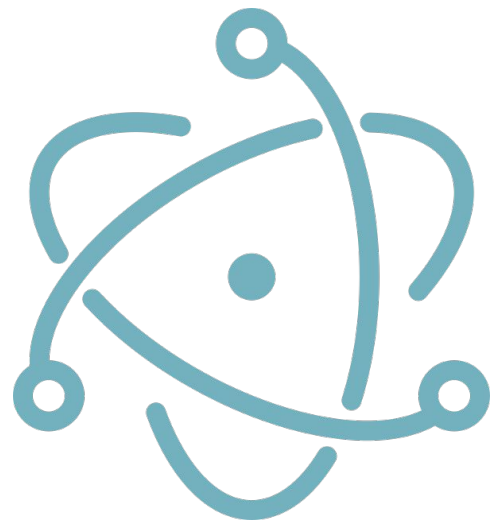


- Ecosystème riche
- Forte communauté
- Performances comparables au natif avec V8
- Simplicité de développement



# Qu'est-ce qu'Electron ?

- Framework pour créer des applications natives cross-platform avec des technologies web
- Créé par GitHub
- Open source depuis juillet 2014
- Développé initialement pour les besoins de l'éditeur Atom

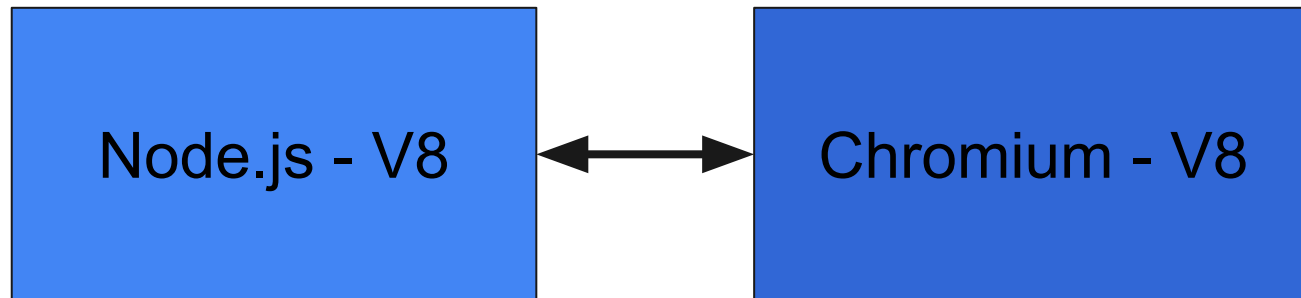




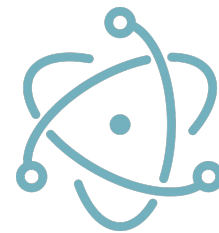
# Vision macro d'Electron



## Electron



# Principales fonctionnalités



Automatic  
updates



Native menus  
& notifications



App crash  
reporting



Debugging  
& profiling



Windows  
installers

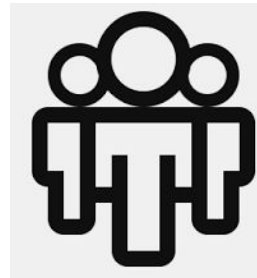
# Activité GitHub



35k+



4k

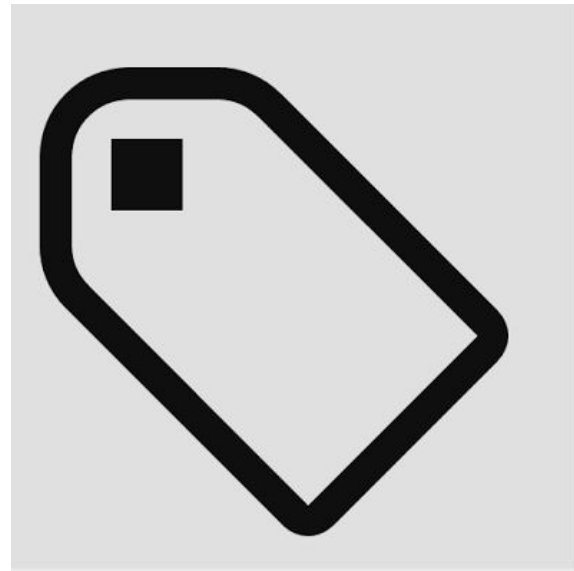


450

# Rythme des releases



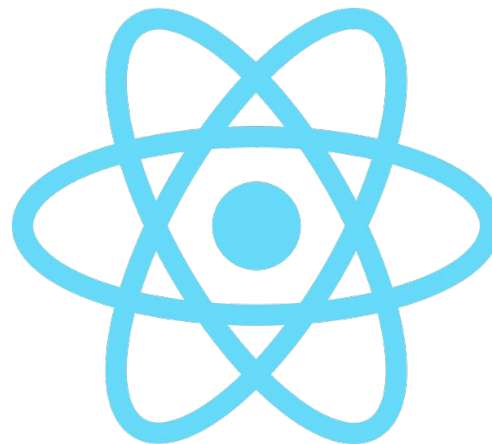
- Plus de 250 releases depuis avril 2014
- 10 mai 2016 : 1.0
- Une release toutes les semaines et une majeure tous les mois
- Actuellement en 1.4.6 (enfin quand on a regardé ce matin...)



# Alternatives



**NW.JS**

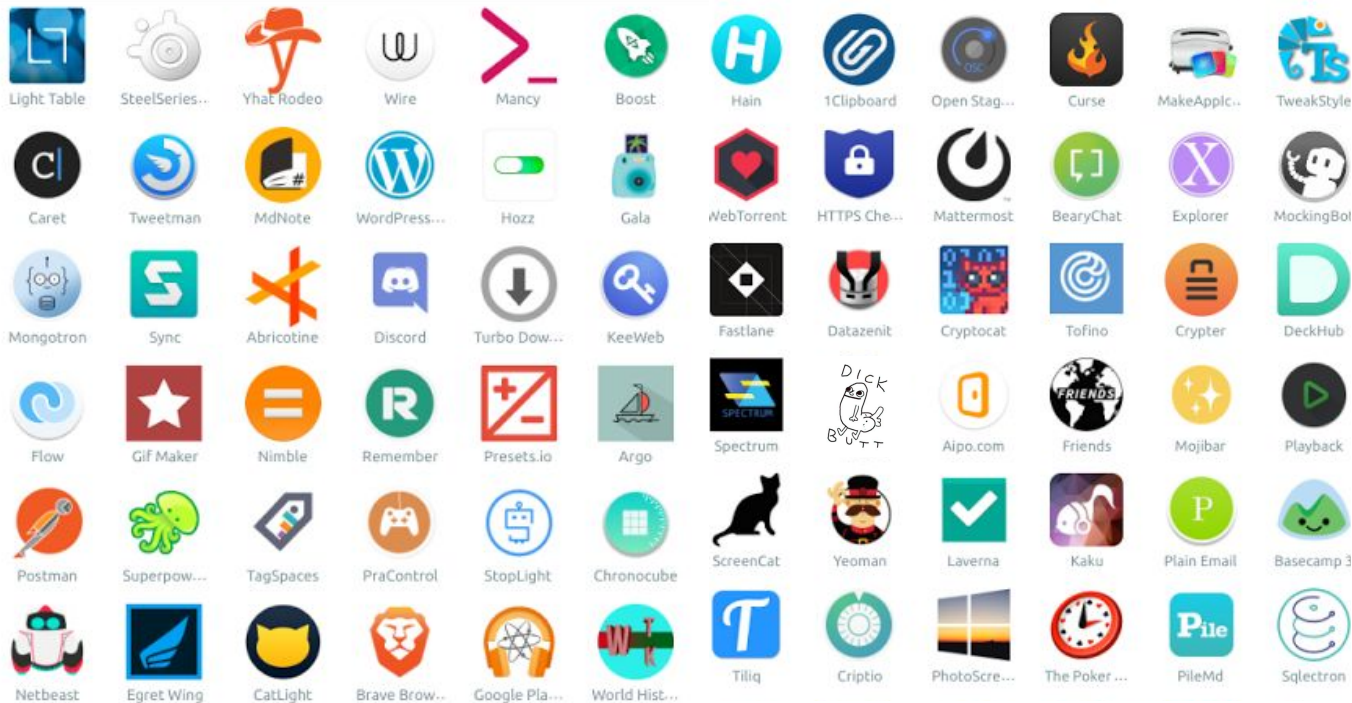


**React Native**

# Quelles applications l'utilisent ?



# Et bien d'autres !



# Plugins



Visuals



Packagers



Tools



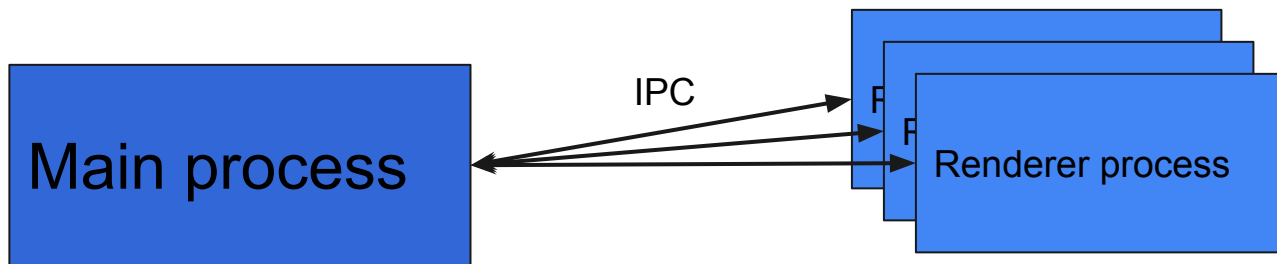
Updaters

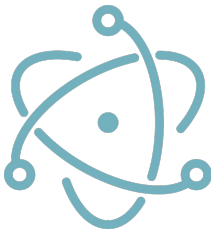


# Architecture logicielle d'une application Electron



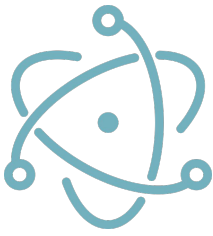
## Electron





# Main Process

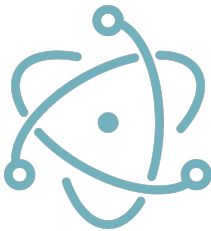
- Gère les instances des fenêtres
- Accède aux fonctionnalités du système
- Réalise les opérations coûteuses



# Renderer Process

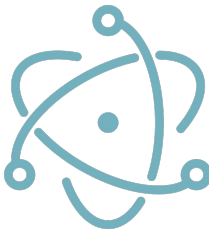
- Gère l'affichage des pages
- Gère les interactions utilisateurs
- Un renderer process par fenêtre

# IPC



- Gère la communication entre MainProcess et les instances de RendererProcess
- Basé sur l'IPC de chromium
- Pour communiquer entre les instances de RendererProcess, il faut passer par le MainProcess

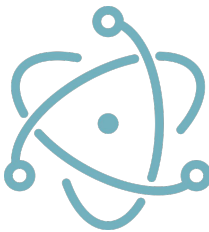
# Spectron



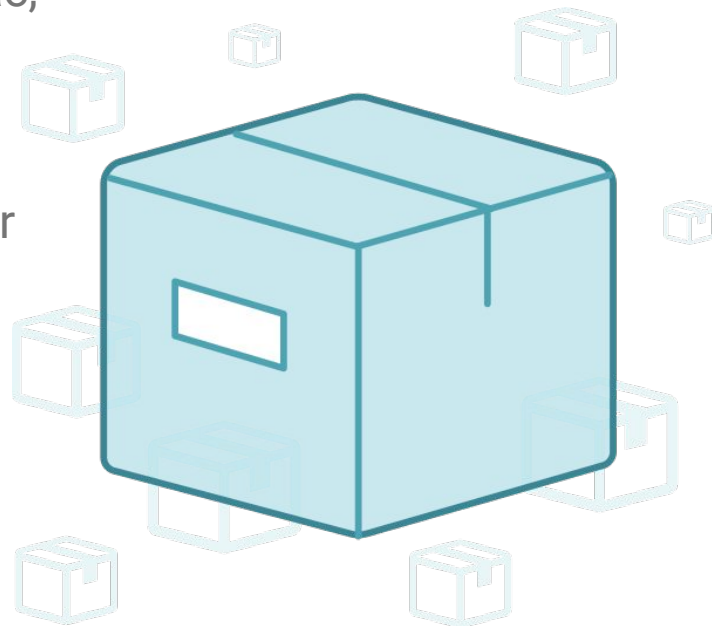
- Framework de test pour electron
- Basé sur le ChromeDriver
- Permet de gérer facilement le cycle de vie de l'appli
- Agnostique du framework de test utilisé

```
it('opens a window displaying the memes', function () {  
  return app.client.getWindowCount().should.eventually.equal(1)  
    .browserWindow.isMinimized().should.eventually.be.false  
    .browserWindow.isDevToolsOpened().should.eventually.be.false  
    .browserWindow.isVisible().should.eventually.be.true  
    .browserWindow.isFocused().should.eventually.be.true  
    .browserWindow.getBounds().should.eventually.have.property('width').and.be.above(0)  
    .browserWindow.getBounds().should.eventually.have.property('height').and.be.above(0)  
    .getTitle().should.eventually.equal('Electron meme generator')  
})
```

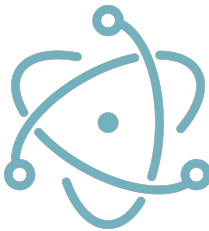
# Packaging



- Cross-platform en 32 ou 64 bits : windows, mac, linux
- Package out of the box avec electron-packager
- Création d'installateur windows
- Possibilité de packager pour les stores mac & windows



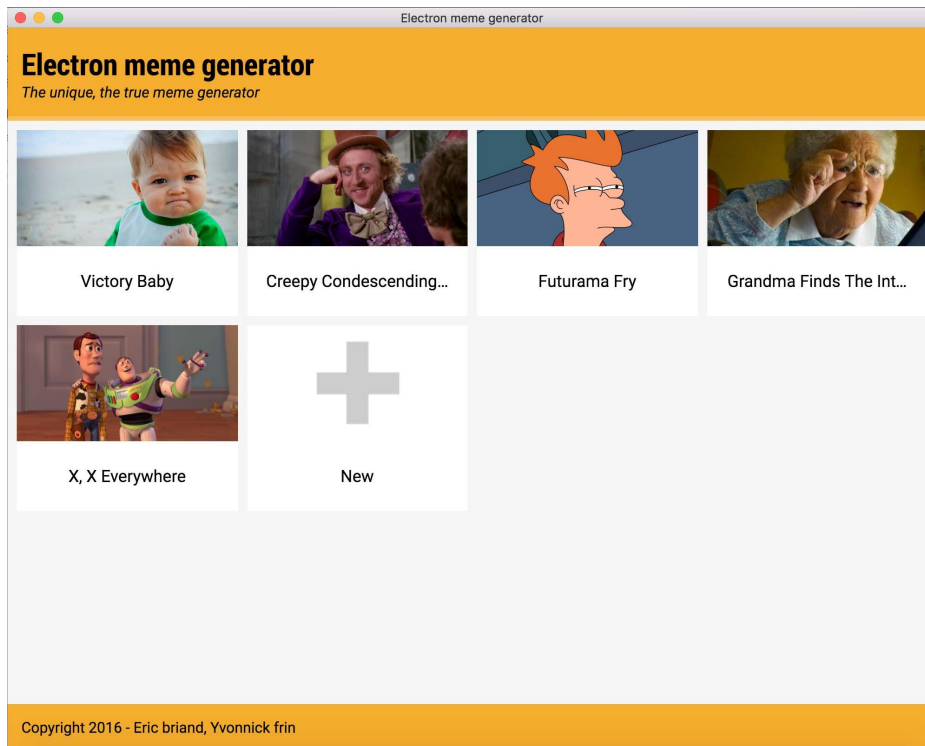
# Mise à jour



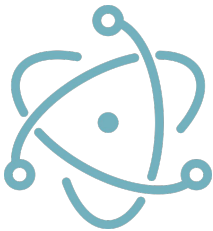
- Appel d'un service REST distant pour avoir les infos sur les dernières versions
- Peut être basée sur les GitHub releases
- electron-release-server
- Auto update via Squirrel



# Workshop : <https://git.io/vXZZR>



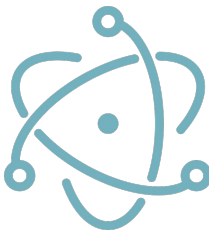




# Workshop : <https://git.io/vXZZR>

## Init workspace

- Dézipper l'archive **electron-workshop-<votre os>.zip**
- Ouvrir un invite de ligne de commande dans le répertoire **electron-workshop** nouvellement créé
- Lancer la commande suivante pour setter vos variables d'environnements:  
Windows : **setupws.cmd**  
Linux / OSX : **. setupws.sh**
- Tout le code du workshop se trouve dans le répertoire workshop

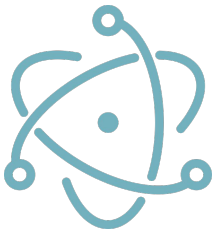


# Workshop : <https://git.io/vXZZR>

## Etape 1 : Hello world!

Nous allons commencer par démarrer notre application Electron en affichant une première page statique.

- Ouvrir le fichier **src/main.js**
- Importer les dépendances **app** et **BrowserWindow** depuis **electron**
- Importer la librairie **path**
- Sur l'événement **ready** de app, instancier une nouvelle **BrowserWindow** et assigner à **mainWindow**
- Charger le fichier **windows/hello.html** dans votre fenêtre nouvellement créée
- Démarrer votre application en exécutant **electron** .

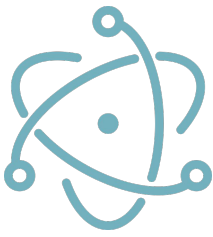


# Workshop : <https://git.io/vXZZR>

## Etape 2 : Customizer la fenêtre

Maintenant que notre application Electron affiche une première fenêtre, nous vous proposons de changer l'affichage à l'aide de différentes options. Nous allons aussi exploiter la capacité de live-reloading du module **electron-connect**.

- Démarrer l'application en lançant **npm run dev**, l'application va démarrer en mode dev avec du live-reloading
- Changer la taille de la fenêtre dans le fichier **src/main.js**
- Enlever les bordures de la fenêtre
- Ouvrir par défaut les devTools via **mainWindow.webContents.openDevTools()**



# Workshop : <https://git.io/vXZZR>

## Etape 3 : Affichage de la liste des memes

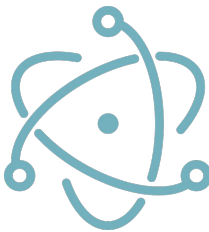
Nous allons maintenant afficher la galerie de memes dans notre fenêtre.

Dans le fichier **src/main.js**

- Changer le fichier HTML chargé dans la mainWindow par le fichier **windows/index.html**

Dans le fichier **src/windows/index.html**

- Avec la fonction `require`, importer le fichier **src/renderer-process/grid.js** de manière relative au fichier **index.html** dans la balise **<script>**

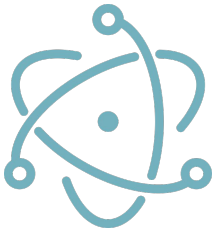


# Workshop : <https://git.io/vXZZR>

## Etape 4 : Inter Process Communication (1/2)

Notre application affiche maintenant une liste statique d'images. La prochaine étape va consister à récupérer la liste des memes à afficher depuis un espace de stockage local (**electron-json-storage**). Nous allons utiliser l'IPC (Inter Process Communication) pour échanger des informations entre le main-process et le renderer-process.

- Dans le fichier **src/renderer-process/grid.js**
- Envoyer un message **get-memes** via le module **ipcRenderer**
- Déplacer le rendu de la galerie dans le callback appelé lors de la réception d'un message **meme-sended**
- Utiliser la liste des images passée en paramètre de ce callback

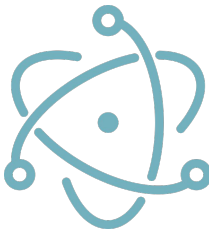


# Workshop : <https://git.io/vXZZR>

## Etape 4 : Inter Process Communication (2/2)

Dans le fichier **src/main-process/grid.js**

- Mettre en place un handler pour le message **get-memes** avec le module **ipcMain**
- Dans le callback du handler, appeler la fonction **getMemes** qui prend un callback comme paramètre
- Dans le callback de **getMemes**, émettre en retour un message **meme-sended** avec la liste des images fournie en paramètre



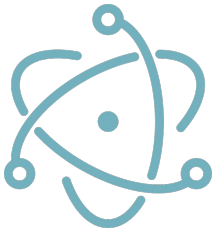
# Workshop : <https://git.io/vXZZR>

## Etape 5 : File dialog (1/2)

Maintenant que nous avons une liste de memes par défaut, nous allons donner la possibilité à l'utilisateur d'ajouter l'image de son choix via une file dialog.

Dans le fichier **src/renderer-process/grid.js**

- Ajouter un event listener **click** sur l'élément avec l'id **new-meme**
- Dans cet event listener, émettre un événement **open-file-dialog** avec l'IPC



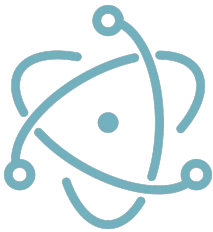
# Workshop : <https://git.io/vXZZR>

## Etape 5 : File dialog (2/2)

Dans le fichier **src/main-process/grid.js**

- Dans celui-ci, importer le module **dialog** depuis **electron**
- Déclarer l'event handler **open-file-dialog**
- En réponse à cet event, afficher une **dialog** qui va lister seulement les fichiers images (extensions jpg, gif, png)
- Implémenter un callback qui va appeler la fonction **newEditWindow** avec le fichier choisi par l'utilisateur
- Gérer l'événement **closed** de la fenêtre nouvellement créée en renvoyant la liste de memes à jour



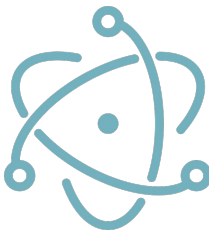


# Workshop : <https://git.io/vXZZR>

## Etape 6 : Menu contextuel

A cette étape, nous allons ajouter un menu contextuel afin de supprimer et de sauvegarder chacune des images de la galerie de memes. Nous allons utiliser les classes de menu présentes dans Electron.

- Ouvrir le fichier **src/render-process/grid.js**
- Importer le module **remote** depuis le module **electron** pour pouvoir accéder à l'API du main process
- Importer les classes **Menu** et **MenuItem** depuis **remote**
- Ajouter un event listener **contextmenu** sur chacun des éléments de la galerie
- Créer un menu contextuel dans le callback de l'event listener avec comme items :
  - **Save as** qui enverra un message **save-from-grid** sur l'IPC
  - **Delete** qui enverra un message **deleted-selected-meme** sur l'IPC

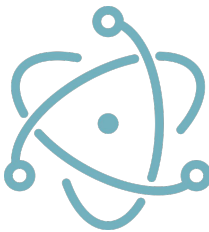


# Workshop : <https://git.io/vXZZR>

## Etape 7 : Notifications

Maintenant que nous avons la possibilité d'ajouter et d'enlever des memes, nous allons émettre des notifications pour que l'utilisateur obtienne une confirmation de ses actions. Pour ce faire nous allons utiliser la classe **Notification** de l'API HTML5.

- Ouvrir le fichier **src/renderer-process/grid.js**
- Ajouter une notification en utilisant la classe **Notification** après la suppression d'un meme
- Ajouter une notification après l'enregistrement d'un meme

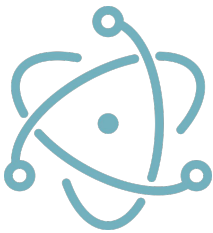


# Workshop : <https://git.io/vXZZR>

## Etape 8 : Packaging

Nous allons terminer l'atelier en packageant notre application. Pour cela, nous allons utiliser electron-packager qui est maintenu par la communauté.

- Ouvrir le fichier **package.json**
- Implémenter le npm script **package** qui va appeler electron-packager
- Ajouter les options pour :
  - cibler votre plateforme et son architecture
  - choisir **dist** comme répertoire de sortie
  - pouvoir repackager l'application même si le packaging a déjà été créé



# Workshop : <https://git.io/vXZZR>

## Etape bonus : Testing

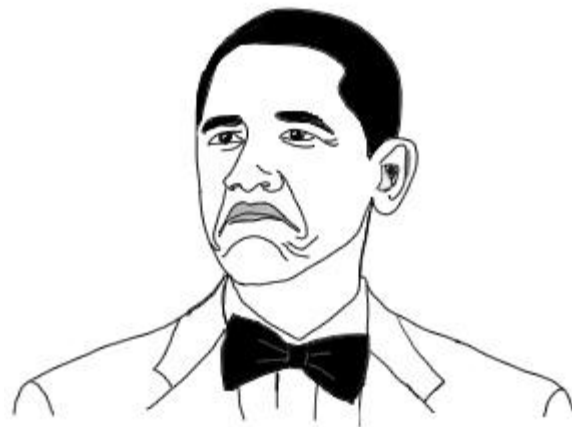
Pour ceux qui veulent aller plus loin, vous pouvez ajouter des tests.

- Ouvrir le fichier **src/tests/index.js**
- Rajouter les tests suivant :
  - l'application n'ouvre qu'une seule fenêtre au lancement
  - le titre de la fenêtre est bien **Electron meme generator**
  - la taille de la fenêtre est bien celle que vous avez précisée au lancement
  - l'application affiche au moins un meme (élément HTML avec la classe CSS **meme**)

# On a aimé



- Documentation très bien faite
- Courbe d'apprentissage
- Ecosystème Node.js + web à dispo
- Un seul navigateur à supporter
- TRES actif



**NOT BAD**

# <https://github.com/electron/electron-api-demos>



ELECTRON API DEMOS

WINDOWS

Create and manage **windows**

Handling window **crashes and hangs**

MENUS

Customize **menus**

Register keyboard **shortcuts**

NATIVE USER INTERFACE

Open **external links** or system **file manager**

Use system **dialogs**

Put your app in the **tray**

COMMUNICATION

Communicate between the **two processes**

SYSTEM

Get app or user **system information**

Copy and paste from the **clipboard**

Launch app from **protocol handler**

MEDIA

**Print** to PDF

Take a **screenshot**

About

<> with ❤️ by GitHub

Create and Manage Windows

The **BrowserWindow** module in Electron allows you to create a new browser window or manage an existing one.

Each browser window is a separate process, known as the renderer process. This process, like the main process that controls the life cycle of the app, has full access to the Node.js APIs.

Open the [full API documentation](#) in your browser.

Create a new window

SUPPORTS: WIN, OS X, LINUX | PROCESS: MAIN

Manage window state

SUPPORTS: WIN, OS X, LINUX | PROCESS: MAIN

Create a frameless window

SUPPORTS: WIN, OS X, LINUX | PROCESS: MAIN

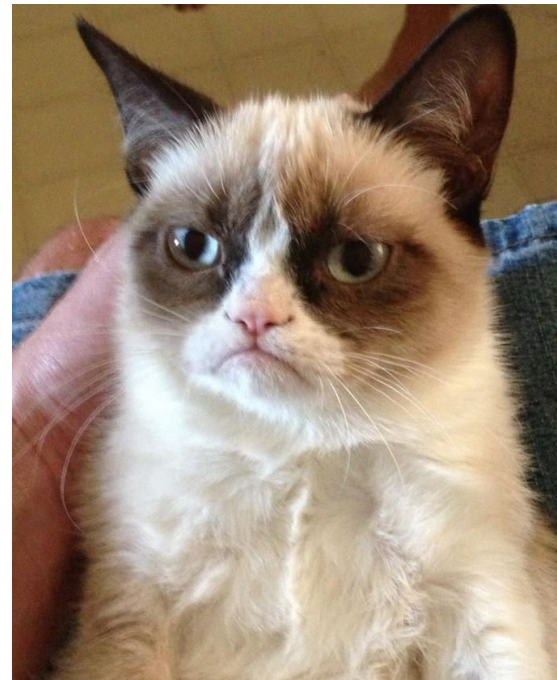
《 DEVFEST NANTES 16 》

38

# On a moins aimé



- N'abstrait pas totalement les spécificités OS
- Privilégie les fonctionnalités communes à tous les OS
- Taille du “livrable”
- Pas de support mobile





# Merci !

Yvonnick Frin (@YvonnickFrin) Eric Briand (@eric\_briand)  
Philippe Charrière (@k33g\_org)