

# Développement front

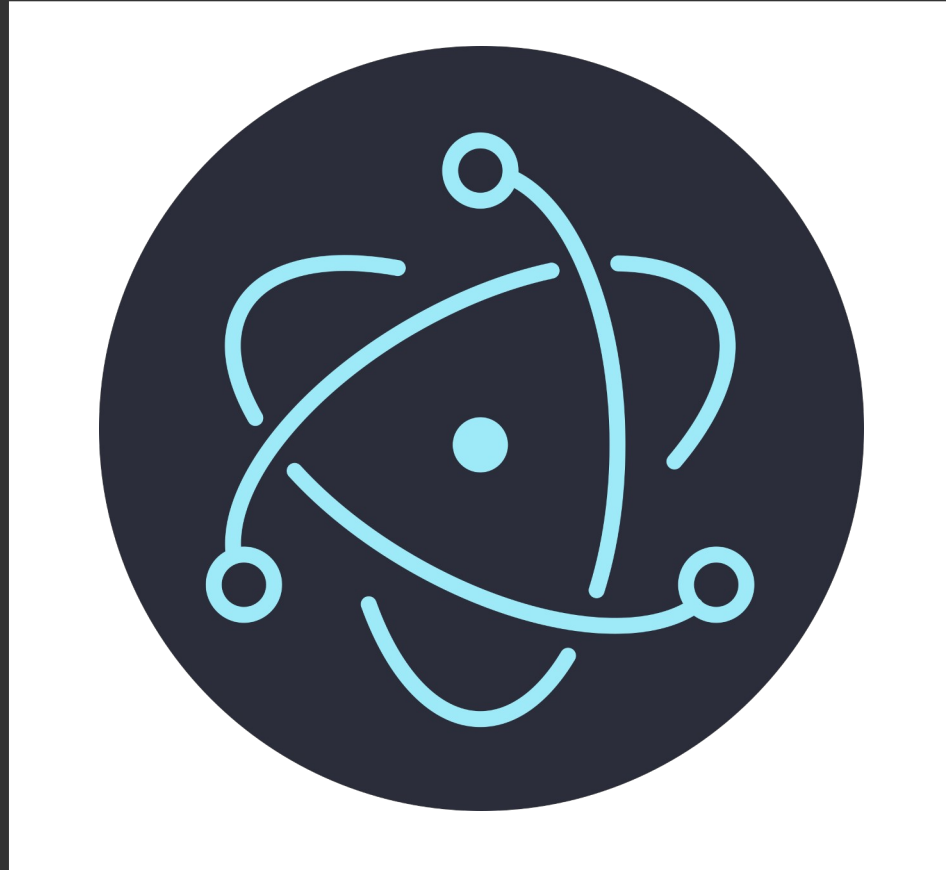
MMI 3 – TP#5 S5

Danielo **JEAN-LOUIS**

# Quels sont les points communs entre ?

- Discord
- Visual Studio Code
- Slack
- Postman

# Electron



Source(s) :

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

# Electron

- Framework front-end permettant de développer des applications natives pour MacOS, Windows et Linux
  - Permet de produire des executables
- Outil Open Source et gratuit
- Très populaire

## Source(s) :

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

# Electron

- Utilise les langages HTML, CSS et Javascript
  - Ticket d'entrée faible (surtout en S5)
- Un code = Plusieurs exécutables
- Utilise les API natives des systèmes d'exploitation
  - L'application ressemble à une app native

Source(s) :

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

# Electron

- Site web embarqué
  - Possibilité d'utiliser la console du navigateur
  - Apps très lourdes avec perfs moyennes
- Basé sur le moteur Chromium

Source(s) :

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

# Electron

- Documentation en français... plus ou moins
- Pensé pour fonctionner hors-ligne
- Éco-système vaste : Multiples outils dispos

## Source(s) :

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



# Pratiquons ! - Electron (Partie 1)

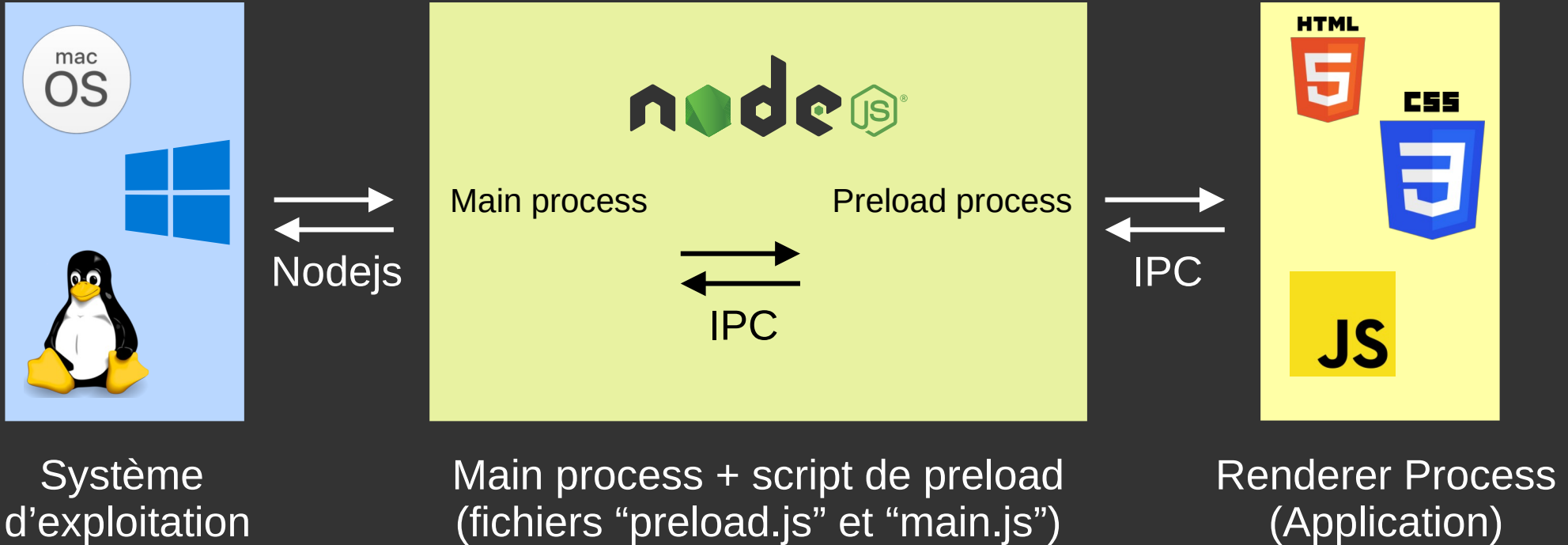
Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-web-et-dispositif-interactif-s6%2Ftravaux-pratiques%2Fnumero-1%2Fressources%2Felectron>

# Architecture (simplifiée)



Source(s) :

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

# Preload process

- Accès au Renderer Process
- Accès limité aux API de Node (par défaut)
- Lien entre main process et renderer process
- Impossibilité d'importer des modules personnels (par défaut)

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/tutorial/tutorial-preload>

# Renderer process

- Représente UNE vue/onglet dans votre application
- Electron est multi-processus
  - Basé sur le processus de Chromium
- Est chargé dans un fichier HTML

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/tutorial/process-model>

# Renderer process

- Appelé juste après le preload process
- Ne communique qu'avec le preload process
- Impossibilité de communication entre plusieurs Renderer process

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/tutorial/process-model>

# Pratiquons ! - Electron (Partie 2, 1 à 2)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-web-et-dispositif-interactif-s6%2Ftravaux-pratiques%2Fnumero-1%2Fressources%2Felectron>

# Renderer process - Sécurité

- Utilise le Content Security Policy ou CSP
- Bloque par défaut les ressources externes non autorisées : iframe, css, js...
- Peut être géré en front et back-end

Source(s) :

- <https://content-security-policy.com/>

# Renderer process - Sécurité

- Le CSP n'est pas obligatoire
  - Mais conseillé pour renfoncer la sécurité
- Modification possible du CSP
  - Voir fichier csp-exemples.html

Source(s) :

- <https://content-security-policy.com/>



# Pratiquons ! - Electron (Partie 2, 3)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopement-web-et-dispositif-interactif-s6%2Ftravaux-pratiques%2Fnumero-1%2Fressources%2Felectron>

# IPC

- Inter-process communication
  - fr : communication inter-processus
- Protocole de communication entre le main process et le renderer process
  - Système de canaux
- Communication événementiel bi-directionnelle

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/glossary#ipc>

# IPC – Exemple – Main Process – main.js

```

// main process
const { ipcMain } = require("electron");
// [...]
ipcMain.handle("evt_main", (evt, arg) => {
  console.log(arg);
  return arg * 2;
})

```

Notre main process exécutera un `console.log()` quand l'évènement "evt\_main" sera appelé depuis le renderer. De plus, il revoit la valeur reçue.

# IPC – Exemple – Preload Process – preload.js

```

// preload process
const { contextBridge, ipcRenderer } = require("electron");

contextBridge.exposeInMainWorld("evt_preload", () => {
  const val = ipcRenderer.invoke("evt_main", 42)
  console.log("The value returned is : " + val);
})
```

Dans le preload process, on expose une fonction (evt\_preload) au renderer process. Elle permet d'appeler l'évènement "evt\_main" dans le main process

# IPC – Exemple – Renderer Process



```
// renderer process
window.addEventListener("DOMContentLoaded", () => {
  window.evt_preload()
})
```

Les fonctions du preload process sont injectées dans l'objet window du renderer process pour être appelées

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/glossary#ipc>

# Pratiquons ! - Electron (Partie 3)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-web-et-dispositif-interactif-s6%2Ftravaux-pratiques%2Fnumero-1%2Fressources%2Felectron>

# IPC – Point sécurité

- N'exposez jamais ipcRenderer en entier



```
contextBridge.exposeInMainWorld("ipcRenderer", ipcRenderer);
```

## Grosse faille de sécurité

N'importe qui peut exécuter du code arbitraire. **Ne faites jamais ça.**

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/tutorial/ipc>

# IPC – Points à retenir

- Impossibilité de faire des API calls depuis le renderer ou preload process, par défaut
  - Solution : Passer par le main process
- Le nom des canaux est arbitraire
  - Préférez des noms explicites
  - Espaces interdits

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/tutorial/ipc>




# IPC – Points à retenir

- La communication du main process vers le renderer process est différente
- Possibilité de grouper vos événements dans un objet au lieu d'avoir plusieurs "exposeInMainWorld"

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/tutorial/ipc>

# Multiples exposeInMainWorld



```
// preload process  
contextBridge.exposeInMainWorld("function_1", function_1);  
contextBridge.exposeInMainWorld("function_2", function_2);  
contextBridge.exposeInMainWorld("function_3", function_3);
```

Chaque fonction exposée de façon distincte

# Multiples exposeInMainWorld

```

// preload process
const listFunctions = {
  function_1: () => {},
  function_2: () => {},
  function_3: () => {},
}

contextBridge.exposeInMainWorld("listFunctions", listFunctions);

```

Les fonctions sont groupées sous le même objet

# IPC – Main process vers Renderer process

- Nécessite de préciser le renderer process visé
  - Rappel : Electron fonctionne avec plusieurs renderer process

Source(s) :

- <https://www.electronjs.org/fr/docs/latest/tutorial/ipc>

# IPC – Main process vers Renderer process

```
● ● ●  
  
// main process  
/* [...] */  
const createWindow = () => {  
  const mainWindow = new BrowserWindow({/*...*/});  
  
  mainWindow.webContents.send('my_event', "hello world")  
  
  mainWindow.loadFile("index.html");  
  mainWindow.webContents.openDevTools()  
};
```

La propriété “webContents” contient un renderer process

# IPC – Main process vers Renderer process

```

// preload process
/* [...] */
const { contextBridge, ipcRenderer } = require('electron')

contextBridge.exposeInMainWorld('electronAPI', {
  handleMainProcess: (callback) => ipcRenderer.on('my_event', callback)
})

```

Le preload process proxyfie l'écouteur d'évènement pour le renderer process.  
Callback représente les données envoyées depuis le main renderer

# IPC – Main process vers Renderer process

```

// renderer process
/* [...] */
const { contextBridge, ipcRenderer } = require('electron')

contextBridge.exposeInMainWorld('electronAPI', {
  handleMainProcess: (callback) => ipcRenderer.on('my_event', callback)
})

window.electronAPI.handleMainProcess((event, value) => {
  console.log(value);
  // event.sender représente le main process
  event.sender.send('renderer', "Bonjour")
})
```

Dans le renderer process, on définit une fonction de retour (callback) pour récupérer la valeur en provenance du main process

# Pratiquons ! - Electron (Partie 4)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-web-et-dispositif-interactif-s6%2Ftravaux-pratiques%2Fnumero-1%2Fressources%2Felectron>



# Appels HTTP

- Impossible dans le renderer process ni le preload process
- Utilisation d'IPC pour faire “remonter” les données vers le renderer process

# Appels HTTP

```
● ● ●  
  
// main process  
ipcMain.handle("APICall", async () => {  
  const res = await fetch("url");  
  const body = await res.json();  
  
  return body;  
});
```

L'appel de la méthode fetch est identique à ce qu'on ferait dans le navigateur.  
On retourne la réponse qui sera capturée par le preload puis le renderer.

# Pratiquons ! - Electron (Partie 5)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-web-et-dispositif-interactif-s6%2Ftravaux-pratiques%2Fnumero-1%2Fressources%2Felectron>

# Design

- Utilisation du CSS, le même CSS utilisé pour faire un site web classique
  - Balise link pour faire le lien entre les CSS et le HTML
  - ...
- Rappel : Electron se base sur le moteur de Chrome
  - Impossibilité d'utiliser des spécificités de Firefox

# Pratiquons ! - Electron (Partie 6)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-web-et-dispositif-interactif-s6%2Ftravaux-pratiques%2Fnumero-1%2Fressources%2Felectron>

# API Native

- Possibilité d'appeler les API natives d'un OS :
  - Notifications, batterie, TouchBar...
  - Attention : certains API sont exclusives
- Appel uniquement dans le main process
  - Possibilité de callback dans le renderer via IPC

# Pratiquons ! - Electron (Partie 7)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici :

<https://download-directory.github.io/?url=https%3A%2F%2Fgithub.com%2FDanYellow%2Fcours%2Ftree%2Fmain%2Fdevelopment-web-et-dispositif-interactif-s6%2Ftravaux-pratiques%2Fnumero-1%2Fressources%2Felectron>

**Questions ?**



