Développement front

MMI 3 – TP#4 S5

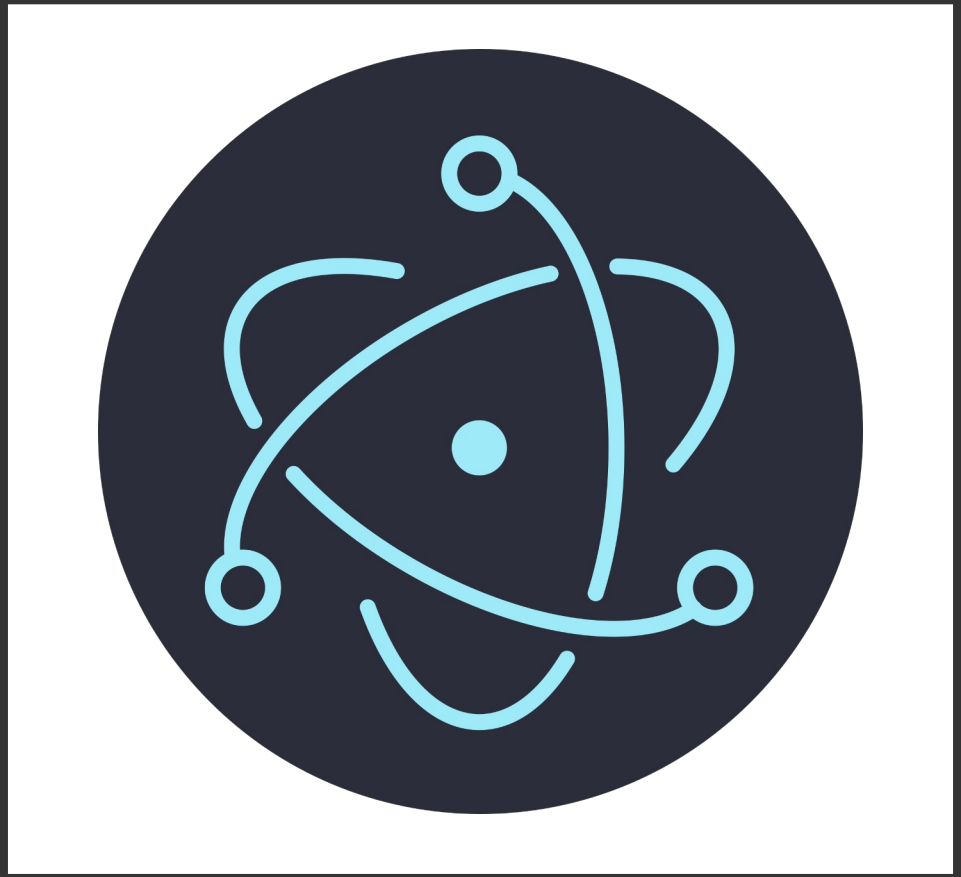
IUT CERGY-PONTOISE

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
  - S’installe comme une application normale
- 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/

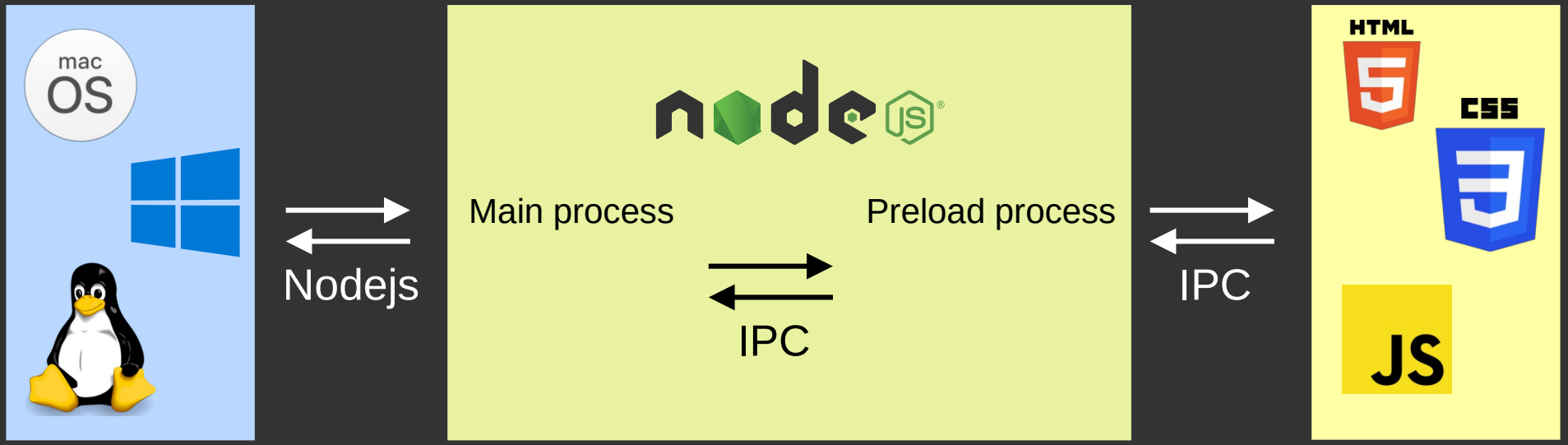
Pratiquons ! - Electron (Partie 1)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici : https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-4/ressources

Architecture (simplifiée)



Système d'exploitation

Main process + script de preload (fichiers "preload.js" et "main.js")

Renderer Process (Application)

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

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, par défaut

Source(s) :

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

IPC

- Inter-process communication
  - Fr : communication inter-processus
- Protocole de communication entre le main process et le renderer process
  - Système de canaux
- Communication évènementiel 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("my_event_main_process", () => {
  console.log("hello");
});
```

Ici notre main process executera un console.log() quand l'évènement "my_event_main_process" sera appelé.
(Cette fonction peut retourner une valeur)

IPC – Exemple – Preload Process – preload.js

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

contextBridge.exposeInMainWorld("my_event_preload_process", () => {
  ipcRenderer.invoke("my_event_main_process");
});
```

Dans le preload process, on expose une fonction (my_event_preload_process) au renderer process. Elle nous permettra d’appeler notre event dans le main process

IPC – Exemple – Renderer Process

```
// renderer process
window.addEventListener('DOMContentLoaded', () => {
    window.my_event_preload_process()
});
```

Les fonctions du preload process sont injectées dans l’objet window du renderer process

Source(s) :

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

Pratiquons ! - Electron (Partie 2)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici : https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-4/ressources

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
  - Solution : Passer par le main process
- Le nom des canaux est arbitraire
  - Préférez des noms explicites
  - Espace interdits

Source(s) :

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

IPC – Points à retenir

- La méthode ipcRenderer.invoke(“event”) peut retourner une valeur en provenance de ipcMain.handle(“event”)

Source(s) :

- https://www.electronjs.org/fr/docs/latest/tutorial/ipc
- https://www.electronjs.org/docs/latest/api/browser-window

IPC – Points à retenir

- La communication du main process vers le renderer process est différente
- Possibilité de grouper vos évènements dans un objet au lieu d’avoir des “exposeInMainWorld” un peu partout

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 3)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici : https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-4/ressources

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 4)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici : https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-4/ressources

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 5)

Pré-requis :

- Avoir la ressource ressources/electron

A télécharger ici : https://download-directory.github.io/?url=https://github.com/DanYellow/cours/tree/main/developpement-front-s5/travaux-pratiques/numero-4/ressources

API Native

- Possibilité d’appeler les API natives d’un OS :
  - Notifications, batterie, TouchBar (MacOS) …
- Appel uniquement dans le main process

API Native - Notifications

- dd

Questions ?