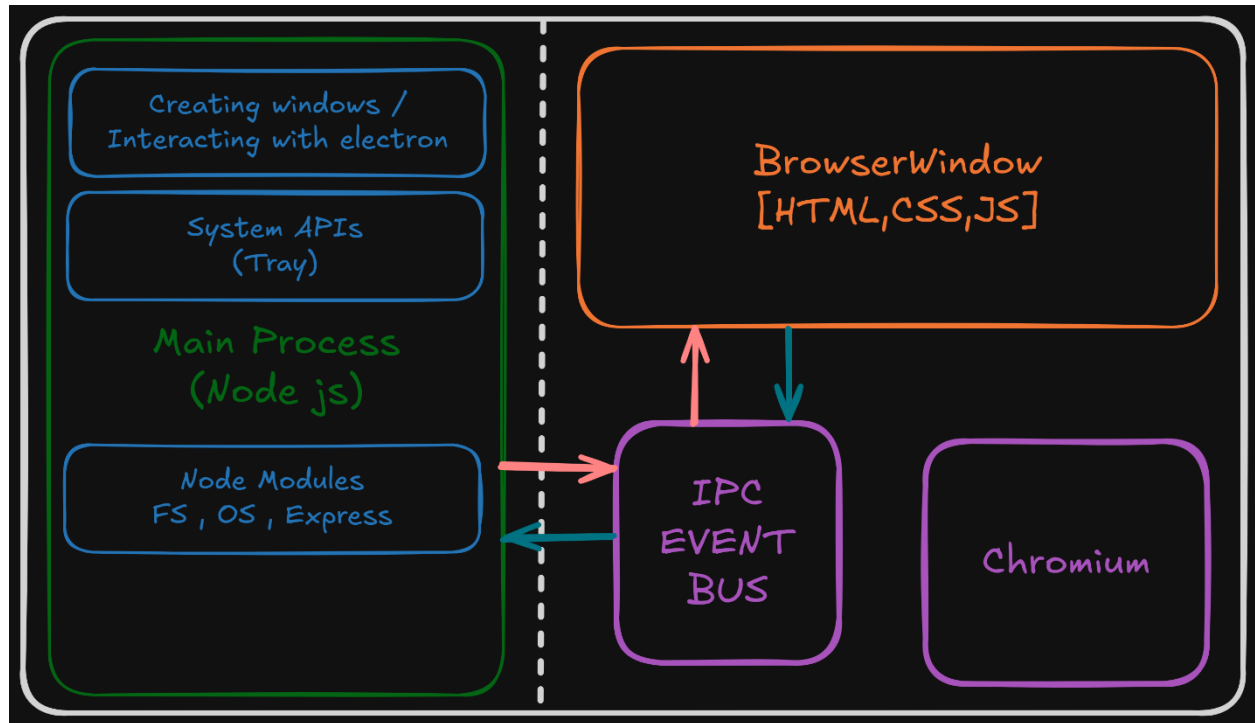


الکترون جی اس چیست؟ (Electron js)



- ❖ ما به کمک الکترون می توانیم به کمک محیط node js با زبان جاوا اسکریپت و HTML و CSS برنامه های تحت دسکتاپ چند سکویی توسعه بدهیم که بر روی لینوکس ، مک و ویندوز اجرا شوند.
- ❖ این فریمورک از node js برای برنامه نویسی Backend و از Chromium برای ساخت ظاهر برنامه و اجرای کدهای جاوا اسکریپت استفاده می کند .
- ❖ این فریمورک توسط کمپانی گیت هاب توسعه داده شده و برنامه های بزرگی از جمله Vs code و دیسکورد و ... با آن توسعه داده شده است.
- ❖ ما از این فریمورک نه تنها به صورت جاوا اسکریپت خام و یا تایپ اسکریپت می توانیم استفاده کنیم بلکه از آن در کنار ری اکت جی اس ، انگولار ، ویو جی اس و یا نکست و... هم می توانیم استفاده کنیم.
- ❖ برای نصب آن از دستور `npm i -D electron` می توانیم استفاده کنیم .

- ❖ برای اجرا برنامه لازم است در فایل package.json یک مقدار main برای الکترون مشخص کنیم مثل "index.js" : "main" .
- ❖ در فایل اصلی باید فقط منطق اصلی برنامه را پیاده سازی کنیم .
- ❖ Main Process مسئول تعامل با سیستم عامل و دسترسی به فایل ها است. در این بخش ماژول های Node.js به سیستم عامل کاربر دسترسی دارند و عملیات مربوط به File System Manipulation را انجام می دهند.
- ❖ وظیفه Renderer Process ایجاد رابط کاربری گرافیکی است. این پردازش، وظیفه کنترل مرورگر کرومیوم برنامه را بر عهده دارد.
- ❖ فایل index.html : رابط کاربری گرافیکی در قالب این فایل پیاده سازی می شود.
- ❖ با دستور . electron در ترمینال و روت پروژه می توانیم پروژه خود را اجرا کنیم.
- ❖ برنامه های الکترون با استفاده از جاوااسکریپت و بهره گیری از اصول و روش های مشابهی که در برنامه نویسی با Node.js وجود دارد، توسعه می یابند. تمامی API ها و ویژگی هایی که در الکترون یافت می شوند، از طریق ماژول electron قابل دسترسی می باشند و مانند هر ماژول دیگر مربوط به Node.js در برنامه بارگذاری می شود.
- ❖ برای ساخت یک برنامه باید ابتدا چرخه حیات آن برنامه را در نظر گرفت. چرخه حیات یک برنامه الکترون از طریق ویژگی electron.app مدیریت می شود. تمام عملیات و رخدادهای برنامه درون این قسمت ایجاد می شوند و از بین می روند.

❖ پنجره های الکترون به وسیله کلاس `electron.BrowserWindow` ایجاد میشوند.

فایل `main.js` را نشان می دهد. فایل `main.js` منتظر می ماند تا `application` آماده شود و یک پنجره را باز کند. به عبارت دقیقتر رویداد `whenReady` پس از وقوع، تابع `createWindow` را فراخوانی میکند:

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

function createWindow () {
  // یک پنجره جدید ساخته میشود
  let win = new BrowserWindow({
    width: 800,
    height: 600,
    webPreferences: {
      nodeIntegration: true
    }
  })

  // فایل مورد نظر را درون پنجره های که قبلاً باز شده است، بارگذاری مینماید
  win.loadFile('index.html')
}

app.whenReady().then(createWindow)
```

ماژول **powerMonitor** : به وسیله این ماژول می توانیم رویداد هایی مثل حالت خواب سیستم یا بیدار شدن سیستم را کنترل کنیم و دستورات لازم را در این رویداد ها اجرا کنیم.

```
// powerMonitor session
powerMonitor.on("suspend", () => {
  console.log("This is suspend event on powerMonitor");
});

powerMonitor.on("resume", () => {
  // if (mainWindow = null) {
  //   createWindow();
  // }
  console.log("This is resume event on powerMonitor");
});
// *****
```

ماژول **globalShortcut** : به وسیله این ماژول می توانیم برای برنامه خود کلید های شورت کات تعریف کنیم .

```
// globalShortcut session
globalShortcut.register("CommandOrControl + F", () => {
  console.log("User pressed : Control + F");
  globalShortcut.unregister("CommandOrControl + F");
});
// *****
```

ماژول desktopCapturer : به وسیله این ماژول به دستکتاپ خود دسترسی داریم و می توانیم عملیاتی مثل اسکرین شات گرفتن از صفحه را انجام دهیم .

```
function takeScreenShot() {  
  desktopCapturer.getSources({  
    types: ["screen"],  
    thumbnailSize: { width: 1366, height: 768 }  
  }).then(res => {  
    mainWindow.webContents.send('screenshot-channel', res[0].thumbnail.toDataURL());  
  })  
}
```

ماژول dialog : به وسیله این ماژول می توانیم MessageBox ، و دیالوگ های متفاوت را نمایش بدهیم .

```
dialog.showOpenDialog(mainWindow, {  
  title: "select New Item",  
  buttonLabel: "Selaetc Item",  
  defaultPath: app.getPath("desktop"),  
  properties: ["createDirectory", "promptToCreate"],  
})  
  .then((res) => {  
    console.log(res.filePaths);  
  });  
  
dialog.showMessageDialog(mainWindow, {  
  title: "Message Box Title",  
  message: "This Is message of Message Box",  
  detail: "This Is Details of Message Box",  
  buttons: ["Yes", "No", "Cancel"],  
})  
  .then((res) => {  
    console.log(res);  
  });
```

ماژول session : به وسیله این ماژول به session دسترسی داریم و از جمله کارهایی مثل ست کردن کوکی ، حذف کوکی و یا بدست آوردن کوکی ها رانجام بدهیم .

```
// cookie session
session.defaultSession.cookies
.set({
  url: "http://localhost",
  name: "FullName",
  value: "ramin Majidi",
})
.then((res) => {
  console.log("set cookie response :", res);
  session.defaultSession.cookies.get({}).then((data) => {
    console.log("data :", data);
  });
});
// *****
```

ماژول Menu : به وسیله این ماژول می توانیم منو های برنامه را تعریف کنیم.

```
const mainMenu = Menu.buildFromTemplate([
  {
    label: "Home",
    submenu: [
      { label: "item1", enabled: false },
      {
        label: "item2",
        click: () => {
          console.log("Item 2 from home is click");
        },
      },
      {
        label: "item3",
        accelerator: "shift + f",
        click: () => {
          console.log("Item 3 from home is click");
        },
      },
      {
        label: "item4",
        role: "togglefullscreen",
      },
    ],
  },
  {
    label: "about",
    submenu: [{ label: "item1" }, { label: "item2" }, { label: "item3" }],
  },
  {
    label: "Edit",
    submenu: [{ label: "item1" }, { label: "item2" }, { label: "item3" }],
  },
]);

export default mainMenu;
```

ماژول webFrame : به وسیله این ماژول ما فریم صفحه دسترسی داریم و کارهایی از قبیل زوم کردن در صفحه را به وسیله متد `setZoomFactor` می توانیم انجام دهیم که عددی پیش فرض آن 1 است .

مفهوم Inter-Process Communication به اختصار IPC به معنای لغوی ارتباط پردازشی داخلی .

✓ اگر در الکترون بخواهیم یک ارتباط بین `main process` و `renderer process` ایجاد کنیم باید از مفهوم IPC استفاده کنیم.

ماژول ipcRenderer بحث رندر کردن را ساپورت می کند ، ما می توانیم اطلاعاتی رو از فایل Main ارسال کنیم و به وسیله این ماژول در فایل `preload` آن را اطلاعات را دریافت کنیم .

ماژول ipcMain : از این ماژول هم در فایل `main` میتوانیم برای تبادل اطلاعات با فایل `preload` استفاده کنیم .

فایل preload : از خصوصیات این فایل این است که همه `api` مربوط به `node js` در فرآیند پیش بارگذاری قابل دسترس است و همینطور تمام افزونه های کرومیوم هم قابل استفاده است .

نحوه کار `ipc` ها چطوره : ما برای `ipc` یک کانال با نام یونیک ایجاد می کنیم و حالا به وسیله `ipcMain` و `ipcRenderer` به این کانال گوش می کنیم و هر دیتای را از طریق این کانال ارسال یا دریافت می کنیم و عملیات مورد نظر را اجرا می کنیم .

```
function takeScreenShot() {
  desktopCapturer.getSources({
    types: ["screen"],
    thumbnailSize: { width: 1366, height: 768 }
  }).then(res => {
    mainWindow.webContents.send('screenshot-channel', res[0].thumbnail.toDataURL());
  })
}
```

به عنوان مثال ما برای گرفتن اسکرین شات از صفحه باید از ماژول desktopScreenShot در فایل main.ts استفاده کنیم و چون در این فایل به dom دسترسی نداریم برای نمایش این عکس در نتیجه آن را به حالت base64 از طریق کانالی به نام screenshot-channel ارسال کردیم.

```
ipcRenderer.on('screenshot-channel', (e, result) => {
  const imgScreen = <HTMLImageElement>document.getElementById('screenShotImage');
  imgScreen.setAttribute('src', result);
})
```

و بعد در فایل preload.ts به وسیله ipcRenderer به کانال screenshot-channel گوش کردیم تا در فرآیند پیش بارگذاری دیتا ارسالی رو دریافت کنیم و در نتیجه آن را در یک تگ img در سند html به نمایش در آوریم .

متدهای IPC :

متد on : برای گوش کردن به رویداد کانال است و با هربار تکرار رویداد این دستورات مجدد اجرا می شوند.

متد once : این متد هم برای گوش کردن به رویداد های کانال استفاده می شود با این تفاوت که فقط یک بار دستورات را اجرا می کند.

متد send : این متد برای ارسال دیتا در کانال استفاده می شود.


```
const btnSendData = <HTMLButtonElement>document.getElementById('btnSendData');
btnSendData.addEventListener('click', () => {
  ipcRenderer.send('test-channel-1', 'Hello World !');
});

// با این متد با هریار رویداد دستورات اجرا میشوند
ipcRenderer.on('test-channel1-res', function (e, args) {
  console.log('test-channel1-res : ', args);
});

// با این متد دستورات فقط یک بار اجرا میشود
ipcRenderer.once('test-channel1-res', function (e, args) {
  console.log('test-channel1-res : ', args);
});
```

مفهوم shared API :

ما داخل node js یکسری ماژول داریم که با استفاده از آنها می توانیم یکسری کارهای سیستمی و مربوط به سیستم عامل یا کارهای فایل و ... را انجام بدیم . ما در الکترون هم این ماژول ها را داریم که می توانیم از آن ها در main process یا renderer process هم استفاده کنیم ، به همین دلیل به این ماژول ها shared API گفته می شود .

ماژول process : به وسیله این ماژول می توانیم اطلاعاتی از سیستم ، کرش کردن برنامه ، هنگ کردن برنامه ، سیستم عامل ، میزان CPU مورد استفاده و ... را انجام داده یا بدست بیاوریم . [لینک مربوطه در داکيومنت](#) .

ماژول screen : به وسیله این ماژول می توانیم مشخصات مانیتور ، محل قرارگیری موس در صفحه ، اضافه شدن یک مانیتور جدید به سیستم ، حذف مانیتور جانبی از سیستم و ... را بدست آوریم . [لینک مربوطه در داکيومنت](#) .

متد `getCursorScreenPoint` : به وسیله این متد از ماژول `screen` می توانیم محل قرار گیری موس در صفحه اپلیکیشن رو بدست آوریم که یک آبجکت جاوا اسکریپتی با دو پروپرتی `x` , `y` را به ما می دهد.

ماژول `Tray` : به وسیله این ماژول می توانیم یک آیکن در قسمت نوار پایینی صفحه کامپیوتر اضافه کنیم و برای آن منو یا منطق خاص پیاده سازی کنیم.

```
Tray

import { app, Menu, Tray } from "electron";
import { join } from "path";
import { mainWindow } from "../main";
const trayMenu = Menu.buildFromTemplate([
  { label: "item1" },
  { label: "item2" },
  { label: "item3" },
]);
function createAppTray() {
  const imagePath = join("assets", "icon.png");
  const appTry = new Tray(imagePath);
  appTry.setToolTip("My Application");
  appTry.setContextMenu(trayMenu);
  appTry.on("click", (e) => {
    if (e.shiftKey) {
      app.quit();
    } else {
      mainWindow.isVisible() ? mainWindow.hide() : mainWindow.show();
    }
  });
}
export default createAppTray;
```

نکته : برای پاک کردن کش برنامه میتوانیم از دستور زیر استفاده کنیم .

```
session.defaultSession.clearCache().then(()=>{  
  console.log("Cache cleared!");  
});
```

ماژول clipboard : به وسیله این ماژول می توانیم متن ، عکس و... که در کلیپ
بورد هستن را مدیریت کنیم .

ماژول shell : به وسیله این ماژول می توانیم عملکرد های مرتبط با یکپارچه سازی
دسکتاپ را انجام دهیم . به وسیله این ماژول می توانیم مدیریت فایل ها و لینک هارا
به وسیله نرم افزار پیش فرض سیستم انجام دهیم.

```
btnOpenElectron.addEventListener('click',function(){  
  shell.openExternal('https://www.electronjs.org');  
});  
btnOpenImage.addEventListener('click',function(){  
  shell.openPath('my-picture.jpg');  
});  
btnOpenFolder.addEventListener('click',function(){  
  shell.showItemInFolder('E:\\example\\electron\\learn-electron\\my-picture.jpg');  
});
```

Notification API : این api مربوط به جاوااسکریپت است و برای نمایش
نوتیفیکیشن به کاربر استفاده می شود . [داکیومنت مربوطه](#)

```

btnShowNotification?.addEventListener('click', function () {
  const notification = new Notification('My Notification', {
    body: 'This is My Body Notification',
  });
  notification.onclick = (e) => {
    console.log("notification clicked");
  };

  notification.onclose = (e) => {
    console.log('notification closed');
  }
});

```

متد `setProgressBar` : به وسیله این متد می توانیم در نوار پایینی دسکتاپ یک نوار وضعیت برای آیکون اپلیکیشن تنظیم کنیم.

```

mainWindow.on('ready-to-show', function () {
  mainWindow.show();
  let progressNumber: number = 0.01;
  const progressBarInterval = setInterval(function () {
    mainWindow.setProgressBar(progressNumber, {mode: "error"});
    if (progressNumber ≤ 1) {
      progressNumber += 0.01;
    } else {
      mainWindow.setProgressBar(-1);
      clearInterval(progressBarInterval);
    }
  }, 75);
});

```

متد **`app.disableHardwareAcceleration()`** : در Electron JS برای غیرفعال کردن شتابدهی سخت افزاری (Hardware Acceleration) استفاده می شود.

شتابدهی سخت افزاری چیست؟

شتابدهی سخت افزاری به استفاده از GPU واحد پردازش گرافیکی) به جای CPU برای انجام وظایف گرافیکی یا پردازش هایی است که می توانند با GPU سریع تر انجام شوند، مانند رندر کردن گرافیک ها، ویدیوها یا انیمیشن ها.

عملکرد `app.disableHardwareAcceleration()`

هنگامی که این متد فراخوانی می‌شود:

1. Electron از شتاب‌دهی سخت‌افزاری برای رندر کردن گرافیک‌ها استفاده نمی‌کند.
2. به جای آن، رندرینگ به کمک CPU انجام می‌شود.
3. این تغییر ممکن است باعث کاهش سرعت در برخی وظایف گرافیکی شود، اما ممکن است برای حل برخی مشکلات سازگاری یا بهبود پایداری برنامه مفید باشد.

زمان استفاده از این متد

- **مشکلات درایور گرافیک:** اگر کاربران مشکلاتی مربوط به درایور GPU داشته باشند (مانند کرش یا ناسازگاری با سیستم‌عامل خاص)، غیرفعال کردن شتاب‌دهی سخت‌افزاری می‌تواند این مشکلات را کاهش دهد.
- **مصرف انرژی:** در برخی موارد، ممکن است CPU بهینه‌تر از GPU برای برنامه‌های ساده عمل کند.
- **سیستم‌های مجازی یا قدیمی:** در برخی سیستم‌های قدیمی یا مجازی، GPU ممکن است به درستی پشتیبانی نشود، و استفاده از CPU می‌تواند عملکرد بهتری داشته باشد.

نحوه استفاده

این متد باید قبل از وقوع رویداد `ready` در ماژول `app` فراخوانی شود، زیرا پس از آن امکان تغییر حالت شتاب‌دهی سخت‌افزاری وجود ندارد.

```
const { app } = require('electron');

// Disable hardware acceleration
app.disableHardwareAcceleration();

app.on('ready', () => {
  // Create your browser window here
});
```

نکته مهم: غیرفعال کردن شتاب‌دهی سخت‌افزاری به صورت عمومی توصیه نمی‌شود مگر اینکه دلیل خاصی برای این کار داشته باشید، زیرا در بیشتر موارد، استفاده از GPU بهینه‌تر است.