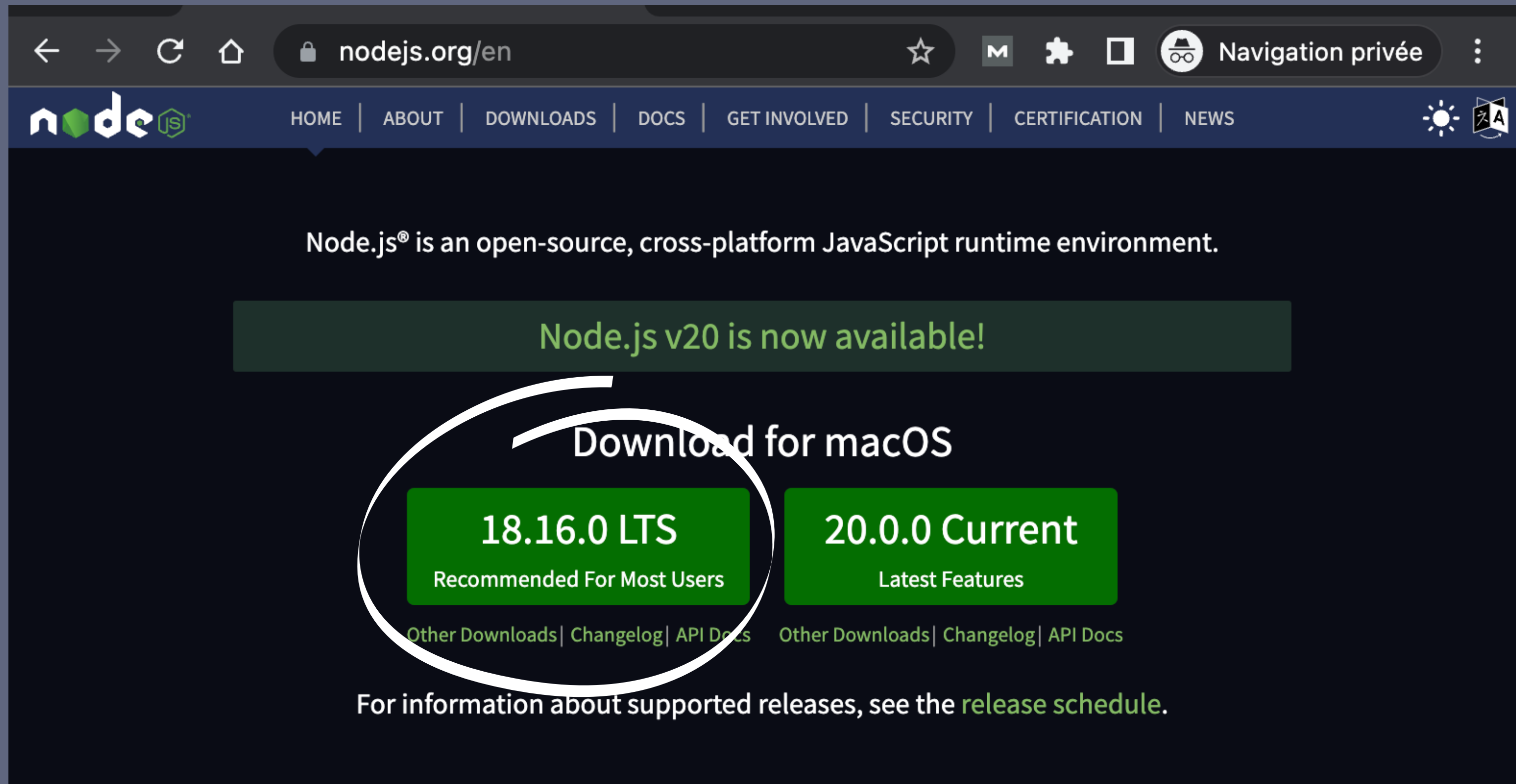




**PDF DU CONTENU DU COURS DE CETTE VIDÉO ET LE CODE
SOURCE SONT DISPONIBLES SUR MON GITHUB.**

**LES LIENS SE TROUVENT
DANS LA DESCRIPTION.**

RENDEZ-VOUS SUR LE SITE DE NODEJS [HTTPS://NODEJS.ORG](https://nodejs.org)
ET TÉLÉCHARGEZ LA DERNIÈRE VERSION LTS (LONG TERM SUPPORT).



The screenshot shows the Node.js website in a web browser. The browser's address bar displays `nodejs.org/en`. The website's navigation bar includes links for HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, CERTIFICATION, and NEWS. A dark green banner in the center of the page reads "Node.js v20 is now available!". Below this, the "Download for macOS" section features two green buttons: "18.16.0 LTS" (labeled "Recommended For Most Users") and "20.0.0 Current" (labeled "Latest Features"). The "18.16.0 LTS" button is circled in white. Below the buttons are links for "Other Downloads", "Changelog", and "API Docs" for both versions. At the bottom, a text line states: "For information about supported releases, see the [release schedule](#)."

nodejs.org/en

nodejs

HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS

Node.js® is an open-source, cross-platform JavaScript runtime environment.

Node.js v20 is now available!

Download for macOS

18.16.0 LTS
Recommended For Most Users

20.0.0 Current
Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#) [Other Downloads](#) | [Changelog](#) | [API Docs](#)

For information about supported releases, see the [release schedule](#).

Une fois **Node.js** installé, ouvrez un terminal pour lancer la commande "node -v" afin de vérifier que Node.js est bien installé.



```
> $ node -v  
v18.10.0
```

Créer un premier script

OUVREZ UN ÉDITEUR DE TEXTE ET CRÉEZ UN NOUVEAU FICHIER AVEC L'EXTENSION ".JS", PAR EXEMPLE "APP.JS".



```
console.log("Bonjour le monde !");
```

Exécuter script Node.js

ENREGISTREZ VOTRE FICHIER ET OUVREZ UNE INVITE DE COMMANDES OU UN TERMINAL. NAVIGUEZ JUSQU'AU DOSSIER CONTENANT VOTRE FICHIER "APP.JS" ET EXÉCUTEZ LA COMMANDE SUIVANTE



```
> $ node app.js  
Bonjour le monde !
```

Modules et gestion des dépendances



Modules et gestion des dépendances

LES TERMES "MODULES", "PACKAGES" ET "BIBLIOTHÈQUES" SONT SOUVENT UTILISÉS DE MANIÈRE INTERCHANGEABLE EN PROGRAMMATION, MAIS ILS ONT DES SIGNIFICATIONS LÉGÈREMENT DIFFÉRENTES.

Un module

UN MODULE EST UNE UNITÉ DE CODE RÉUTILISABLE QUI PEUT ÊTRE UTILISÉE DANS DIFFÉRENTES PARTIES D'UNE APPLICATION.

LES MODULES SONT GÉNÉRALEMENT STOCKÉS DANS DES FICHIERS DISTINCTS, AVEC UN FICHIER PAR MODULE.

POUR UTILISER UN MODULE DANS UN AUTRE MODULE, VOUS POUVEZ L'IMPORTER EN UTILISANT LE MOT-CLÉ REQUIRE().



```
// Importer le module "fs" (file system)  
const fs = require("fs");
```


Un package

UN PACKAGE, EN REVANCHE, EST UN ENSEMBLE DE FICHIERS, SOUVENT DES MODULES, QUI SONT DISTRIBUÉS ENSEMBLE EN TANT QU'UNITÉ LOGIQUE.

LES PACKAGES SONT GÉNÉRALEMENT STOCKÉS DANS DES RÉPERTOIRES ET SONT SOUVENT PUBLIÉS SUR UN REGISTRE DE PACKAGES TEL QUE [NPMJS.COM](https://www.npmjs.com).

UN PACKAGE PEUT CONTENIR PLUSIEURS MODULES, AINSI QUE D'AUTRES FICHIERS TELS QUE DES FICHIERS DE CONFIGURATION ET DES FICHIERS DE DOCUMENTATION.



la gestion des dépendances

LA GESTION DES DÉPENDANCES SE RÉFÈRE À LA MANIÈRE DONT LES PACKAGES D'UNE APPLICATION SONT ORGANISÉS ET GÉRÉS.

LA GESTION DES DÉPENDANCES CONSISTE À S'ASSURER QUE TOUS LES PACKAGES DONT UN PACKAGE DÉPEND SONT PRÉSENTS ET À JOUR, ET À LES INSTALLER SI NÉCESSAIRE.

NODE.JS UTILISE UN GESTIONNAIRE DE PAQUETS APPELÉ NPM (NODE PACKAGE MANAGER) POUR GÉRER LES PACKAGES ET LES DÉPENDANCES D'UNE APPLICATION.



Installer un module avec npm

POUR INSTALLER LE MODULE EXPRESS DANS VOTRE APPLICATION, VOUS POUVEZ EXÉCUTER LA COMMANDE SUIVANTE



```
npm install express
```

Initialiser un projet **Node.js**

POUR COMMENCER, IL FAUT OUVRIR UN TERMINAL ET SE DÉPLACER DANS LE RÉPERTOIRE OÙ VOUS SOUHAITEZ CRÉER VOTRE PROJET NODE.JS.

UNE FOIS DANS LE RÉPERTOIRE, VOUS POUVEZ INITIALISER VOTRE PROJET NODE.JS EN EXÉCUTANT LA COMMANDE SUIVANTE :

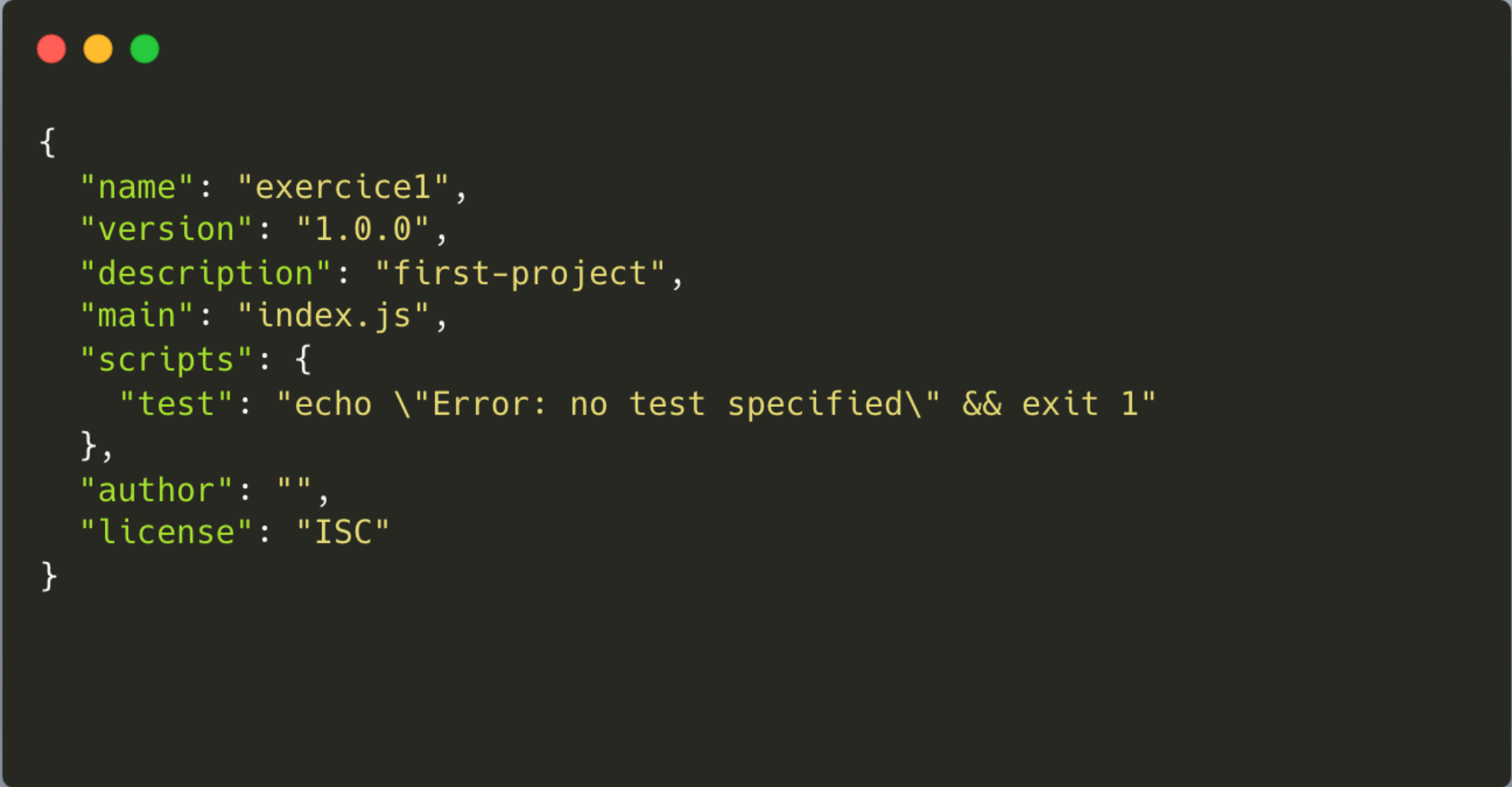


```
npm init
```

Fichier package.json

DANS CE FICHIER PACKAGE.JSON, NOUS AVONS DÉFINI LE NOM, LA VERSION ET LA DESCRIPTION DU PROJET DANS LES PROPRIÉTÉS NAME, VERSION ET DESCRIPTION.

LA PROPRIÉTÉ MAIN INDIQUE LE FICHIER JAVASCRIPT PRINCIPAL DE NOTRE PROJET.

A code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays the JSON content of a package.json file.

```
{  
  "name": "exercice1",  
  "version": "1.0.0",  
  "description": "first-project",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "author": "",  
  "license": "ISC"  
}
```

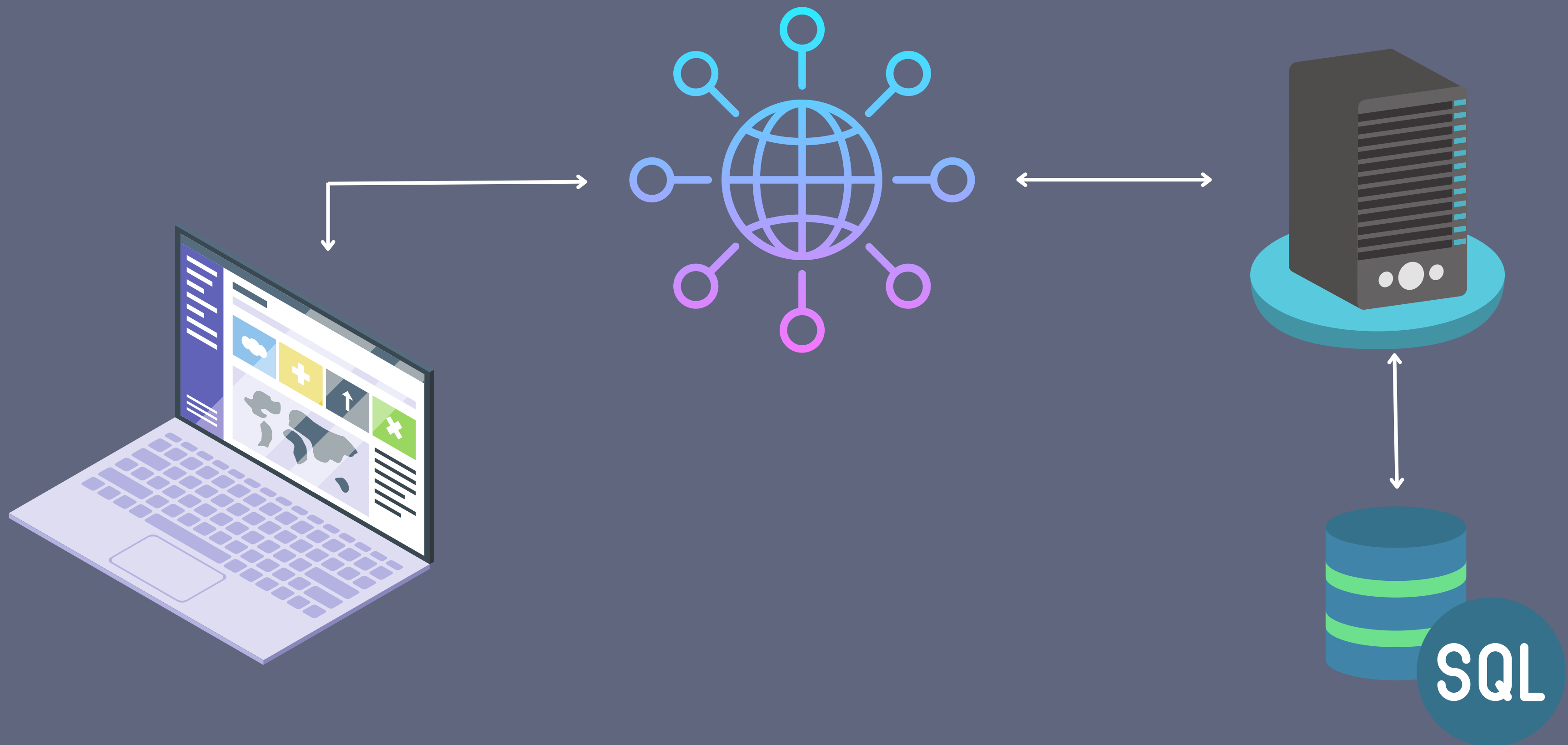
Exécuter script Node.js

ENREGISTREZ VOTRE FICHIER ET OUVREZ UNE INVITE DE COMMANDES OU UN TERMINAL. NAVIGUEZ JUSQU'AU DOSSIER CONTENANT VOTRE FICHIER "APP.JS" ET EXÉCUTEZ LA COMMANDE SUIVANTE

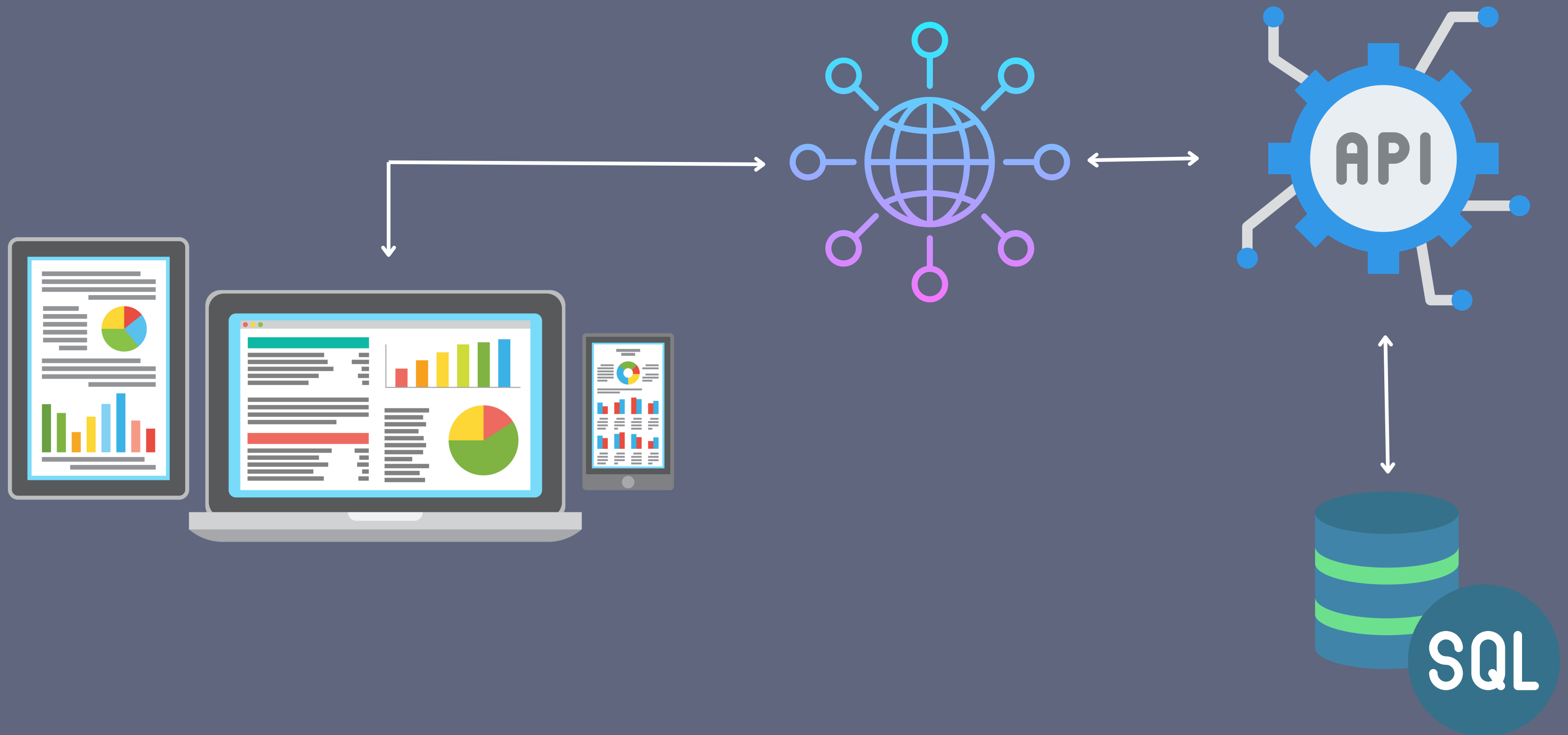


```
> $ node app.js  
Bonjour le monde !
```

serveur web



API



Créer un serveur web simple

AVEC NODE.JS, IL EST FACILE DE CRÉER UN SERVEUR WEB.

VOICI UN EXEMPLE SIMPLE DE SERVEUR WEB QUI RÉPOND "HELLO, WORLD!" À CHAQUE REQUÊTE.



```
// Importer le module "http"
const http = require("http");

// Créer un serveur web
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello, World!");
});

// Écouter les requêtes sur le port 3000

server.listen(3000, () => {
  console.log("Serveur démarré sur le port 3000");
});
```

Explication

NOUS IMPORTONS LE MODULE "HTTP" EN UTILISANT LA COMMANDE REQUIRE.

CELA NOUS PERMET D'ACCÉDER À TOUTES LES FONCTIONNALITÉS FOURNIES PAR CE MODULE.

```
// Importer le module "http"  
const http = require("http");
```

CreateServer

NOUS CRÉONS UN SERVEUR WEB EN UTILISANT LA MÉTHODE CREATESERVER FOURNIE PAR LE MODULE "HTTP".

CETTE MÉTHODE PREND UNE FONCTION CALLBACK QUI SERA EXÉCUTÉE CHAQUE FOIS QU'UNE REQUÊTE EST REÇUE SUR LE SERVEUR.

```
// Créer un serveur web
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello, World!");
});
```

Les objets req et res

NOUS UTILISONS LES OBJETS REQ ET RES POUR GÉRER LA REQUÊTE ET LA RÉPONSE RESPECTIVEMENT.

LA MÉTHODE WRITEHEAD EST UTILISÉE POUR DÉFINIR LES EN-TÊTES DE LA RÉPONSE.

LA MÉTHODE END EST UTILISÉE POUR ENVOYER LA RÉPONSE AVEC LE CORPS DU MESSAGE "HELLO, WORLD!".

```
// Créer un serveur web
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello, World!");
});
```

Les objets req et res

- **REQ** REPRÉSENTE L'OBJET DE DEMANDE HTTP ENTRANT QUI CONTIENT TOUTES LES INFORMATIONS RELATIVES À LA REQUÊTE ENVOYÉE PAR LE CLIENT.
- **RES** REPRÉSENTE L'OBJET DE RÉPONSE HTTP SORTANT QUI CONTIENT TOUTES LES INFORMATIONS RELATIVES À LA RÉPONSE RENVOYÉE PAR LE SERVEUR.

```
// Créer un serveur web
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello, World!");
});
```

Méthode listen

ENFIN, NOUS ÉCOUTONS LES REQUÊTES SUR LE PORT 3000 EN UTILISANT LA MÉTHODE LISTEN FOURNIE PAR L'OBJET SERVER.

NOUS UTILISONS ÉGALEMENT UNE FONCTION DE CALLBACK POUR AFFICHER UN MESSAGE LORSQUE LE SERVEUR DÉMARRE.


```
// Écouter les requêtes sur le port 3000  
  
server.listen(3000, () => {  
    console.log("Serveur démarré sur le port 3000");  
});
```

Les fonctions fléchées

LES FONCTIONS FLÉCHÉES PERMETTENT D'ÉCRIRE DES FONCTIONS PLUS COURTES ET PLUS LISIBLES. DE PLUS, ELLES ONT UNE PARTICULARITÉ : ELLES HÉRITENT DU CONTEXTE "THIS" DE LEUR ENGLOBANT.



```
() => {  
  // Votre code ici  
};
```



```
let addition = (a, b) => {  
  return a + b;  
};
```

Les callbacks

LES CALLBACKS SONT DES FONCTIONS QUI SONT PASSÉES EN TANT QU'ARGUMENT À UNE AUTRE FONCTION ET QUI SONT EXÉCUTÉES ULTÉRIEUREMENT, GÉNÉRALEMENT À LA FIN D'UNE OPÉRATION ASYNCHRONE OU LORSQU'UN CERTAIN ÉVÉNEMENT SE PRODUIT.

SIMPLE D'UTILISATION D'UN CALLBACK POUR AFFICHER UN MESSAGE APRÈS UN CERTAIN DÉLAI:



```
setTimeout(() => {  
    console.log("Message affiché après 2 secondes");  
}, 2000);
```




```
function greet(name, callback) {  
    let greeting = `Bonjour, ${name} !`;  
    callback(greeting);  
}  
  
greet("Alice", (message) => {  
    console.log(message); // Affiche "Bonjour, Alice !"  
});
```

synchrones

LES OPÉRATIONS SYNCHRONES BLOQUENT L'EXÉCUTION DU PROGRAMME
JUSQU'À CE QU'ELLES SOIENT TERMINÉES

asynchrones

LES OPÉRATIONS ASYNCHRONES PERMETTENT À L'EXÉCUTION DU
PROGRAMME DE CONTINUER PENDANT QU'ELLES SONT EN COURS.

synchrone

LES OPÉRATIONS SYNCHRONES BLOQUENT L'EXÉCUTION DU PROGRAMME
JUSQU'À CE QU'ELLES SOIENT TERMINÉES



```
const fs = require("fs");


console.log("Début de la lecture synchrone");

try
{
    let data = fs.readFileSync("example.txt", "utf8");
    console.log("Contenu du fichier (synchrone) :", data);
} catch (err) {
    console.error("Erreur lors de la lecture du fichier (synchrone) :", err);
}

console.log("Fin de la lecture synchrone");
```

asynchrones

LES OPÉRATIONS ASYNCHRONES PERMETTENT À L'EXÉCUTION DU PROGRAMME DE CONTINUER PENDANT QU'ELLES SONT EN COURS.



```
const fs = require("fs");

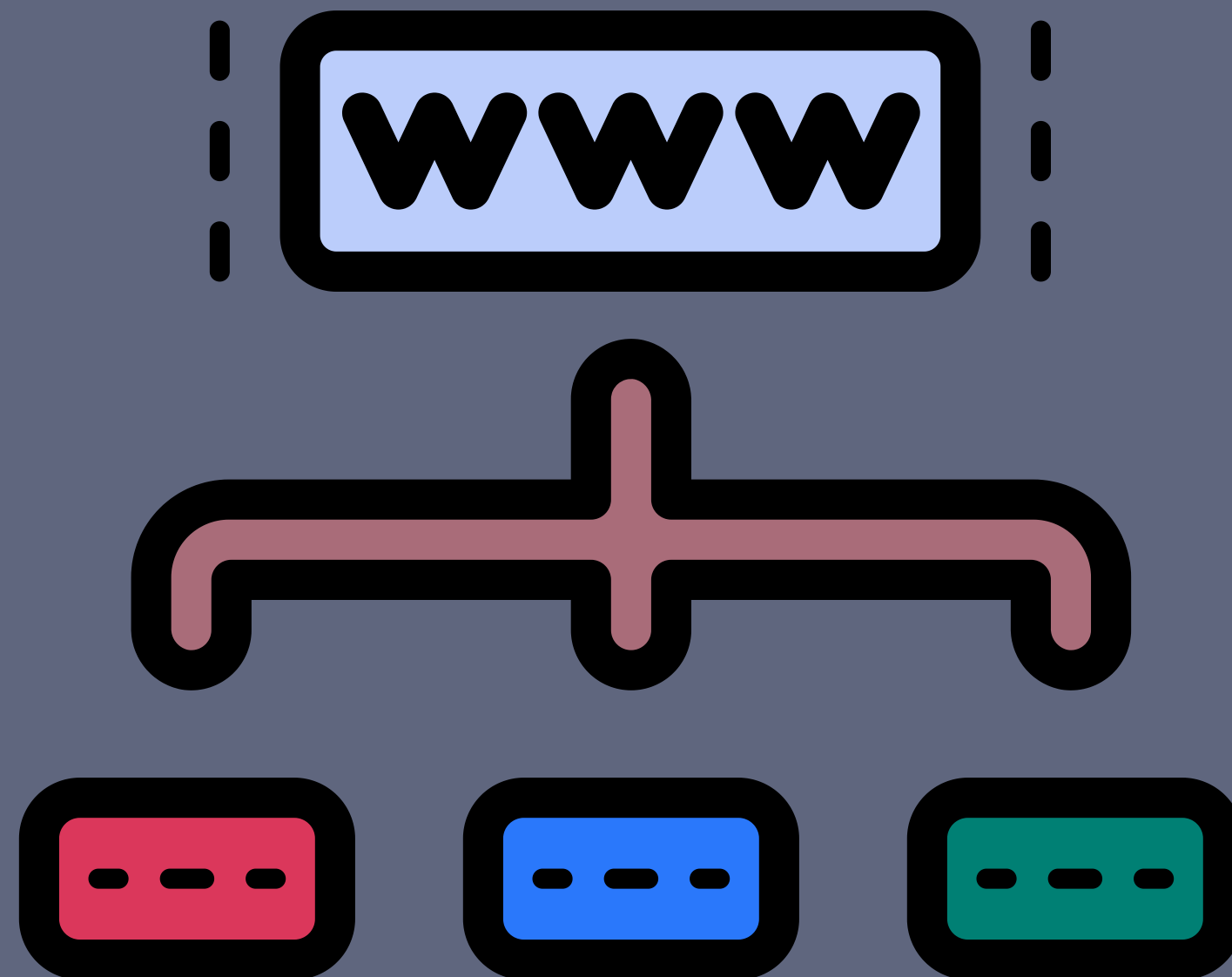
console.log("Début de la lecture asynchrone");

fs.readFile("example.txt", "utf8", (err, data) => {
  if (err) {
    console.error("Erreur lors de la lecture du fichier (asynchrone) :",
err);
    return;
  }
  console.log("Contenu du fichier (asynchrone) :", data);
});

console.log("Fin de la lecture asynchrone");
```

Gestion des requêtes et des routes

POUR CRÉER DES APPLICATIONS WEB PLUS ÉLABORÉES, NOUS DEVONS ÊTRE EN MESURE DE GÉRER DIFFÉRENTES ROUTES ET DE RÉPONDRE AUX REQUÊTES HTTP.





```
// Importer le module "http"
const http = require("http");

// Créer un serveur web
const server = http.createServer((req, res) => {
  res.writeHead(200, { "Content-Type": "text/plain" });
  res.end("Hello, World!");
});

// Écouter les requêtes sur le port 3000

server.listen(3000, () => {
  console.log("Serveur démarré sur le port 3000");
});
```



```
// Importer le module "http" et "url"
const http = require("http");

const url = require("url");

// Créer un serveur web
const server = http.createServer((req, res) => {
  let pathname = url.parse(req.url).pathname;

  if (pathname === "/") {
    res.writeHead(200, { "Content-Type": "text/plain" });
    res.end("Page d'accueil");
  } else if (pathname === "/a-propos") {
    res.writeHead(200, { "Content-Type": "text/plain" });
    res.end("Page à propos");
  } else {
    res.writeHead(404, { "Content-Type": "text/plain" });
    res.end("Page introuvable");
  }
});

// Écouter les requêtes sur le port 3000
server.listen(3000, () => {
  console.log("Serveur démarré sur le port 3000");
});
```