NodeJS : Express

- **Introduction à Express**
- MVC
- Installation et utilisation d’Express
- Routes simples
- Routes dynamiques
- Erreur 404
- Templates
- Middlewares

Introduction

- Express est un **framework** qui permet de créer des sites et des applications web sur NodeJS
- Au lieu de tout construire à partir de zéro, on **réutilise des modules existants** connus des développeurs, avec une **structure standard**
- La maintenance, les corrections de bugs et l'ajout de fonctionnalités sont **facilités**
- Express est un framework basé sur **MVC** (Model View Controller)

https://expressjs.com/

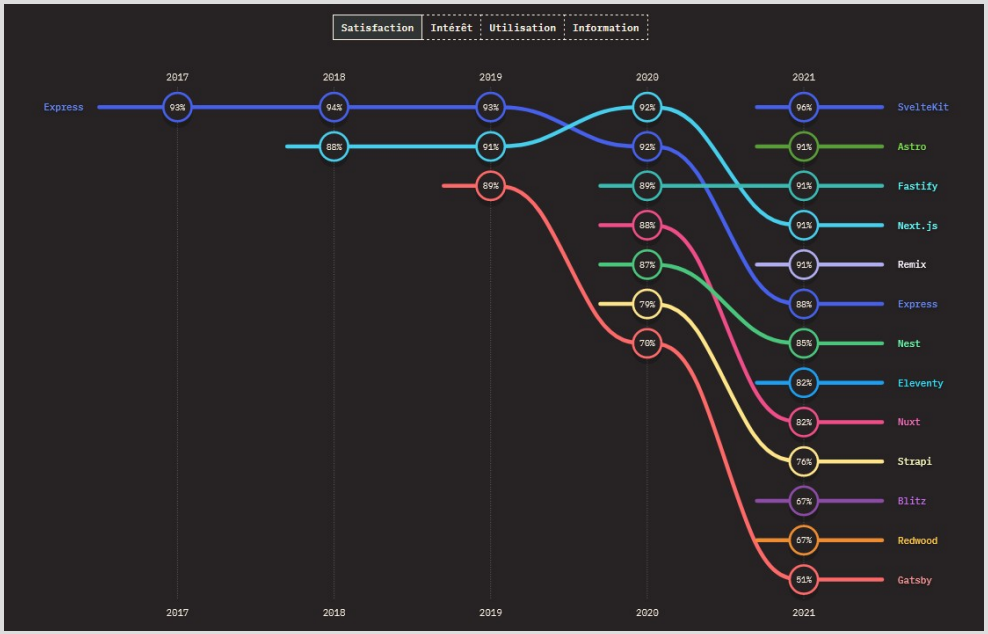
MVC (Model View Controller)

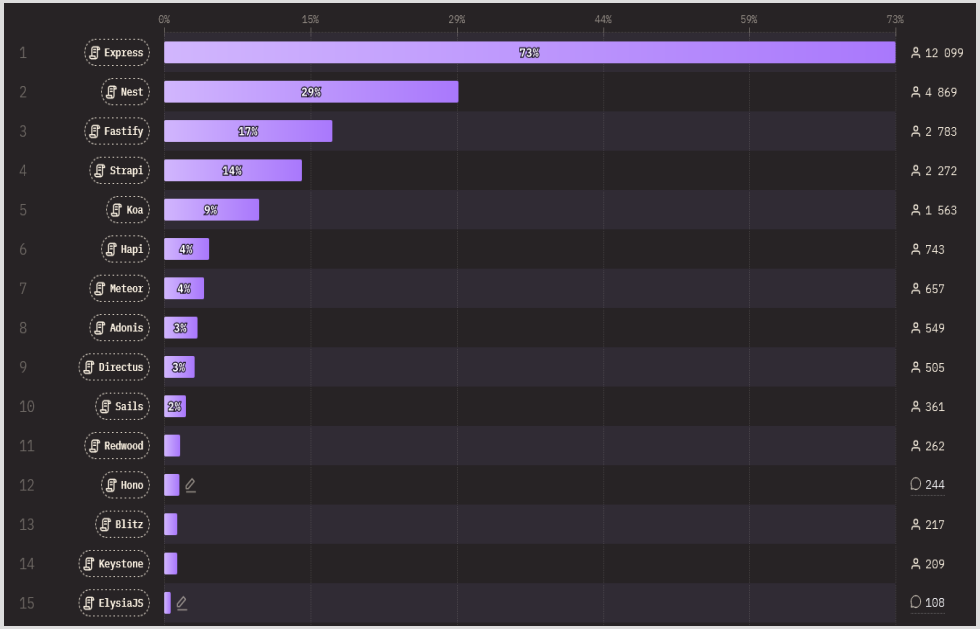
- **Model** : le **modèle** représente les données et les entités de l'application. Les données sont fournies par une base de données ou un service Web
- **View** : une **vue** est la couche présentation qui est responsable de l’affichage des informations pour l’utilisateur final
- **Controller** : le **contrôleur** fait le lien entre les modèles et les vues afin d’afficher la bonne page en fonction de la requête de l’utilisateur

Express

- **Express** est un des frameworks les plus utilisés pour construire des applications web (single-page, multi-page ou hybride)
- Il fournit les **briques de base** et les **outils** nécessaires pour créer et gérer un serveur web
- Il va servir essentiellement à :
  - la gestion des requêtes en définissant les **routes**
  - la gestion des affichages avec les **templates**
  - la gestion des **middlewares**

Pourquoi Express ?

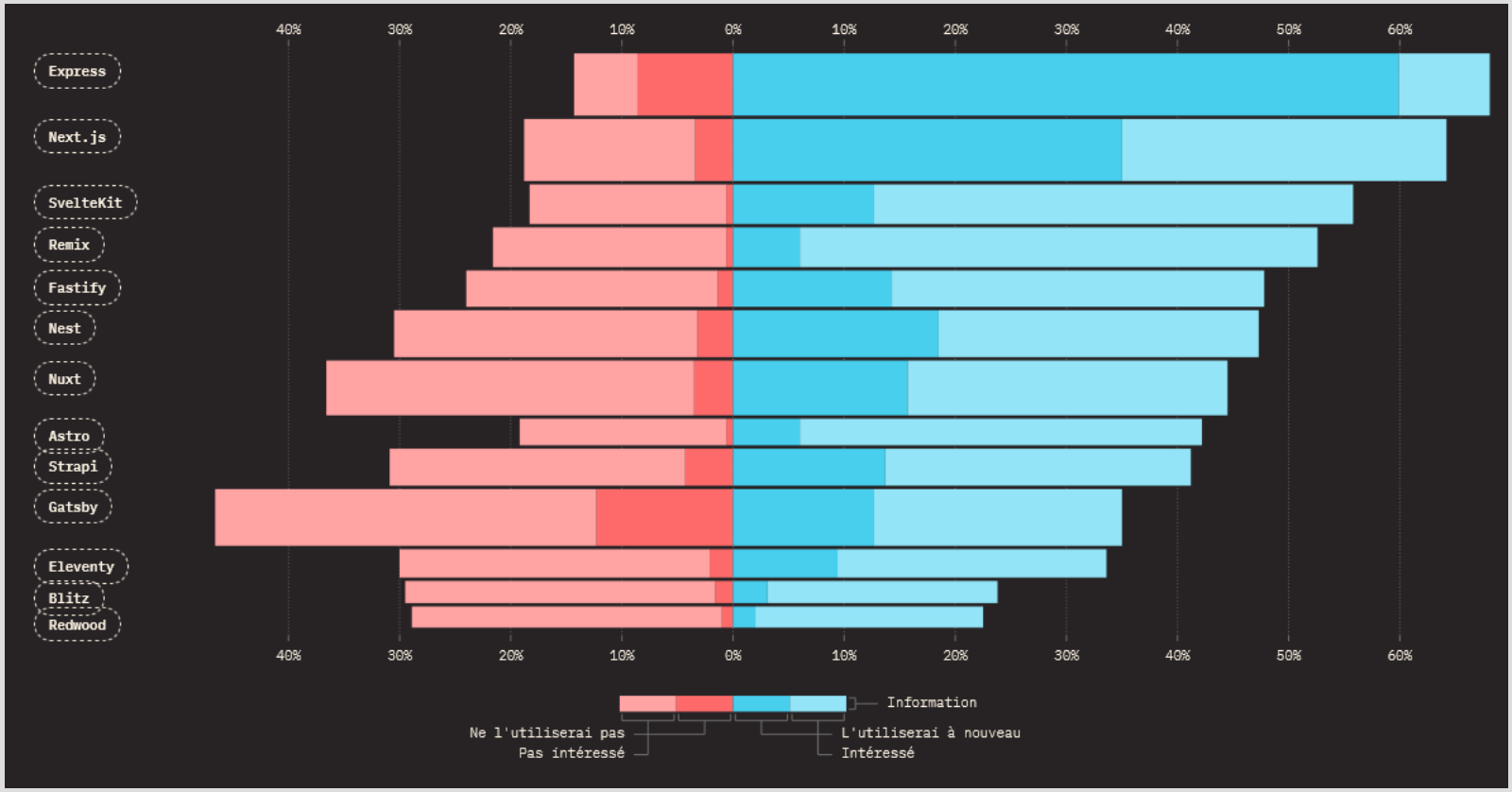




- Il existe plusieurs frameworks back-end réalisés avec JavaScript, et actuellement Express est le plus utilisé

The State Of JS - 2021 2022 2023

Retours d’expérience



- Les expériences positives sont encore très bonnes malgré l’ancienneté du framework

Routes

- Jusqu’à présent, on doit **tester l’URL** de la requête (et ses différentes composantes) pour savoir quelle page afficher... **Mais**, code affreux (succession de If/Else ou Switch)
- Il faut faire la **gestion des routes** c’est-à-dire des différentes URL auxquelles l’application répond
- On retrouve ce concept dans d’autres frameworks comme Symfony (PHP), Django (Python), Spring (Java),...
- **Express** facilite cette gestion

Installer et utiliser Express

- Créer un nouveau projet avec **npm init** , ensuite, pour installer Express, on utilise la commande

  npm install --save express (sauve dans les dépendances du fichier package.json)

- Créer le fichier principal : **index.js**
- Définir un objet **app**, qui gère l'application (requête, réponse, routing,…) et utiliser la fonction **express( )**
- Lancer **node index.js**

```
// ajout du module Express (framework web)
const express = require('express');

// création d'une application
const app = express();

// définition d'une route simple vers la page d'accueil (racine du site)
app.get('/', function(requete,reponse){
    reponse.setHeader('Content-Type', 'text-plain');
    reponse.write("Ceci est la page d'accueil");
    reponse.end();
});

app.listen(8080);   // port 8080
console.log("Express démarré !");
```

Express : objet Application

- Voici les principales méthodes (voir doc : https://expressjs.com/en/4x/api.html#app)

Méthodes	Description
app.get( )	Appel d'une fonction de callback selon la route (HTTP GET)
app.post( )	Appel d'une fonction de callback selon la route (HTTP POST)
app.all( )	Appel d'une fonction de callback selon la route (pour tous les types de requête HTTP)
app.use( )	Appel d'un middleware selon la route
app.set( )	Définition des paramètres d'une application (moteur de template, nom des répertoires, port d'écoute,...)
app.listen()	Mise en écoute du serveur

Express : objet Response

- Voici les principales méthodes (voir doc : https://expressjs.com/en/4x/api.html#res)

Méthodes	Description
res.set( )	Définition de l'entête HTTP de la réponse (Content-type, charset,...)
res.send( )	Envoi de la réponse HTTP
res.status( )	Définition du status de la réponse HTTP (codes 404,200,...)
res.render( )	Envoi des données à afficher sur une vue (page HTML)
res.redirect( )	Redirige une requête sur un autre chemin
res.end()	Termine la construction de la réponse pour l'envoyer
res.setHeader( )	Ecriture de l'entête HTTP (objet Response du module http)
res.write( )	Ecriture de la réponse (objet Response du module http)

Routes simples

- On indique les **différentes routes** (URL) et on l'associe à une **fonction de callback** qui gère la requête.
- Selon la **méthode HTTP** (get, post, put,…) utilisée par la requête, on la traite avec des méthodes différentes : .get( ) ou .post( ) ou…
- Les URL sont simples et sans paramètre.
- Sinon il faut utiliser des **routes dynamiques**

Route simple (exemple)

- On a défini 3 URL différentes

```
// ajout du module externe Express.JS (framework Web)
const express = require('express');

// création d'une application web avec le framework Express
const app = express();

// définition d'une route vers la page d'accueil (racine du site)
app.get('/', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Bienvenue, vous êtes sur la page d'accueil !");
    reponse.end();
});

// Etape 2a : définition d'une nouvelle route vers la page "A propos"
app.get('/about', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Ceci est la page 'A propos' !");
    reponse.end();
});

// Etape 2b : définition d'une nouvelle route vers la page de contacts
app.get('/contacts', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Ceci est la page de 'Contacts' !");
    reponse.end();
});

// démarrage du serveur et écoute sur le port 8080
app.listen(8080);
console.log("Express est démarré...");
```

Erreur 404

- Si on veut **gérer les erreurs** dans les entêtes HTTP (exemple : 404 = page non trouvée), on ajoute **app.use( )** à la fin **juste avant app.listen( )**

  (voir principes du middleware)

```
// ajout du module externe Express.JS (framework Web)
const express = require('express');

// création d'une application web avec le framework Express
const app = express();

// définition d'une route vers la page d'accueil (racine du site)
app.get('/', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Bienvenue, vous êtes sur la page d'accueil !");
    reponse.end();
});

// Etape 2a : définition d'une nouvelle route vers la page "A propos"
app.get('/about', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Ceci est la page 'A propos' !");
    reponse.end();
});

// Etape 2b : définition d'une nouvelle route vers la page de contacts
app.get('/contacts', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Ceci est la page de 'Contacts' !");
    reponse.end();
});

// Etape 3 : gestion des pages non définies (erreur 404)
app.use(function(requete, reponse, next){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.status(404).send("Page introuvable !");
});

// démarrage du serveur et écoute sur le port 8080
app.listen(8080);
console.log("Express est démarré...");
```

Chaîner les méthodes

- Les méthodes de l'objet app renvoient aussi des objets app, on peut donc **raccourcir le code** :

app.get( ).get( ).get( ).use( );

```
const express = require('express');

const app = express();

// les 3 méthodes .get() et la 4ème . use() sont chaînées
app.get('/', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Bienvenue, vous êtes toujours sur la page d'accueil !");
    reponse.end();
})
.get('/about', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Ceci est la page 'A propos' !");
    reponse.end();
})
.get('/contacts', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Ceci est la page de 'Contacts' !");
    reponse.end();
})
.use(function(requete, reponse, next){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.status(404).send("Page encore introuvable !");
});

app.listen(8080);
console.log("Express est démarré...");
```

Route dynamique

- S'il y a des **parties variables** dans la requête, on peut les exploiter avec des paramètres
- On ajoute dans l'URL la notation **:<mavariable>** et cela devient un paramètre accessible depuis **req.params.<mavariable>**
- Attention à l'injection de code, vu que l'utilisateur peut écrire ce qu'il veut dans l'URL. Toujours **vérifier** dans la fonction de callback **si la valeur transmise est correcte**

Route dynamique (exemple)

```
const express = require('express');
const app = express();

app.get('/', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Bienvenue, vous êtes sur la page d'accueil !");
    reponse.end();
})
.get('/about', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Ceci est la page 'A propos' !");
    reponse.end();
})
.get('/contacts', function(requete, reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.write("Ceci est la page de 'Contacts' !");
    reponse.end();
})
.get('/page/numero=:pagenum', function(requete,reponse){
    reponse.setHeader('Content-Type', 'text/plain');
    const monParametre = requete.params.pagenum;
    // validation des paramètres (transmis dans la variable  pagenum )
    // exemple : on suppose qu'on accepte uniquement des valeurs entières de 1 à 5
    if (Number.isInteger(Number(monParametre)) && ((monParametre>0)&&(monParametre<=5))){
        reponse.write("Ceci est la page n°" + monParametre);
        // OU BIEN reponse.write(`Ceci est la page n°${monParametre}`);
        reponse.end();
    } else {
        reponse.status(404).send("La valeur de numero n'est pas autorisée !");
    }
})
.use(function(requete, reponse, next){
    reponse.setHeader('Content-Type', 'text/plain');
    reponse.status(404).send("Page introuvable !");
});

// démarrage du serveur et écoute sur le port 8080
app.listen(8080);
console.log("Express est démarré...");
```