**RAPPEL : CMS, Frameworks, Librairies & Bases JS/React**

**1. Concepts de Développement Web : Une Analogie de Construction**

Imaginez que vous construisez un bâtiment. Ces outils représentent différentes manières de mener votre projet.

| Concept | Nature | Analogie Simple | Différence Fondamentale |
| --- | --- | --- | --- |
| **CMS** (Drupal, WordPress) | Application web **complète** et prête à l'emploi. | **La Maison Prête à Habiter**. Vous l'assemblez et commencez à y vivre. | Vous travaillez **à l'intérieur** de sa structure rigide. |
| **Framework** (Symfony, ExpressJS) | **Structure** logicielle organisée fournissant des règles et des fondations. | **Le Cadre de Construction (Fondations et Poutres)**. Il vous guide dans la construction. | C'est le Framework qui **vous appelle** (Inversion de Contrôle). |
| **Librairie** (ReactJS, jQuery) | **Ensemble de fonctions** que vous utilisez pour des tâches spécifiques. | **La Boîte à Outils Spécialisée** (perceuse, niveau laser). | **Vous appelez** la librairie uniquement quand vous en avez besoin. |
| **Plugin/Module** | Petit paquet de code qui **ajoute une fonctionnalité** à un système existant (CMS/Framework). | **L'Appareil Spécifique** (un système d'alarme, un chauffe-eau). | Ne fonctionne **que s'il est branché** dans son système hôte. |

Exporter vers Sheets

**2. Bases Fondamentales de Drupal (CMS)**

Drupal est un CMS robuste, reconnu pour sa **flexibilité**, sa **sécurité** et sa capacité à gérer des sites complexes.

* **Tout est un Nœud (*Node*)** : Le Nœud est l'unité de contenu de base (une page, un article, un produit). Les Nœuds sont organisés par **Types de Contenu** (ex: Article, Page Basique).
* **Champs et Entités (*Fields and Entities*)** : Le contenu est géré via des **Entités** (Nœuds, Utilisateurs) qui sont composées de **Champs** hautement configurables (texte, image, référence).
* **Modules et Crochets (*Hooks*)** :
  + Les **Modules** sont les extensions.
  + Les **Crochets** sont le mécanisme d'extensibilité clé : ce sont des fonctions prédéfinies que vos modules personnalisés peuvent "accrocher" pour injecter leur propre logique à des moments précis de l'exécution du système.
* **Vues (*Views*)** : Outil puissant permettant de créer, **sans coder**, des listes, des blocs et des pages affichant le contenu selon des critères complexes (filtres, tris).

**3. Bases Fondamentales de JavaScript (JS - ES6+)**

Essentielles pour ReactJS et ExpressJS.

| Concept | Syntaxe | Rôle Clé |
| --- | --- | --- |
| **Variables** | const (valeur constante, privilégiée), let (valeur réaffectable). | Gestion de la portée **Bloc** ({}). |
| **Fonctions Fléchées** | const maFonction = () => {} | **Ne possède pas son propre this**. Indispensable pour les **Composants Fonctionnels** en React. |
| **Destructuration** | const { nom, age } = utilisateur; | Extraction rapide des **props** en React ou des données req.body en Express. |
| **Opérateur de Propagation** | const nouvelObj = { ...ancienObj, nouvelleProp: 1 }; | **Copie** non mutante des tableaux et objets. Crucial pour gérer l'**État (State)** en React. |
| **Modules** | export et import | Organisation du code en fichiers réutilisables, fondamental dans les projets Node.js et React. |

Exporter vers Sheets

**4. Distinction ReactJS vs. React Native et Frameworks/Librairies**

| Caractéristique | ReactJS (ou React) | React Native |
| --- | --- | --- |
| **Nature** | **Librairie** JavaScript. | **Framework** (basé sur React). |
| **Cible** | Applications **Web** (Navigateurs). | Applications **Mobiles Natifs** (iOS et Android). |
| **Rendu** | Dans le **DOM** du navigateur. | Via un *bridge* pour les **composants natifs** de l'OS. |
| **Langage Mobile** | N/A | Utilise **JavaScript** pour interagir avec les API **natives** de l'appareil. |

Exporter vers Sheets

| Catégorie | Outil | Rôle du Contrôle |
| --- | --- | --- |
| **Librairie** | **ReactJS** | **Le développeur** appelle la librairie. |
| **Framework** | **ExpressJS**, **React Native** | **Le Framework** appelle le code du développeur. |

## 3. Bases Fondamentales de JavaScript et ECMAScript (ES6+) pour React/Express

Le développement moderne (React côté client, Express/Node.js côté serveur) s'appuie massivement sur les fonctionnalités introduites dans **ECMAScript 2015 (ES6)** et les versions ultérieures.

### A. Les Fondamentaux du Langage

| Concept | Description | Application React/Express |
| --- | --- | --- |
| **let et const** | **Portée de Bloc (*Block Scoped*)**. const est la valeur par défaut (ne peut pas être réaffectée). let est utilisé si la variable doit changer. | Préférés partout pour éviter les erreurs de portée (var n'est plus utilisé). |
| **Fonctions Fléchées (*Arrow Functions*)** | Syntaxe concise : (param) => { ... }. Elles sont importantes car elles ne créent **pas leur propre contexte this**, mais héritent du contexte parent. | Cœur des **Composants Fonctionnels** React et des fonctions de rappel (*callbacks*) en Express. |
| **Destructuration** | Extraction facile des propriétés d'un objet ou des éléments d'un tableau. | **React :** Récupérer les props d'un composant (const { titre, contenu } = props;). **Express :** Récupérer des données du corps d'une requête (const { email, motDePasse } = req.body;). |
| **Opérateur de Propagation (*Spread Operator*)...** | Permet de copier ou de fusionner des tableaux et des objets sans modifier l'original (copie superficielle). | **Crucial en React** pour la **gestion de l'état (State)** (useState) : setListe([...ancienneListe, nouvelElement]). Cela garantit l'**Immutabilité**. |
| **Littéraux de Gabarit (*Template Literals*)** | Utilisation du *backtick* (`) pour inclure des variables et des expressions directement dans une chaîne de caractères : ${variable}. | Simplifie la création de chaînes de caractères dynamiques (messages d'erreur, chemins d'API, JSX). |

Exporter vers Sheets

### B. Programmation Asynchrone (Essentielle pour Express)

Express étant basé sur Node.js, qui est **mono-threadé** et fonctionne avec des événements, la gestion de l'asynchronisme est vitale.

| Concept | Description | Application Express |
| --- | --- | --- |
| **Promesses (*Promises*)** | Un objet représentant l'achèvement ou l'échec éventuel d'une opération asynchrone. Utilise les méthodes .then() et .catch(). | Express utilise des Promesses pour gérer les interactions avec les bases de données (MongoDB, PostgreSQL) ou les API externes. |
| **Async/Await** | Une surcouche syntaxique aux Promesses, rendant le code asynchrone beaucoup plus lisible (il ressemble à du code synchrone). | **Express :** Utilisé dans les **Contrôleurs** pour attendre la fin d'une opération de base de données avant de renvoyer la réponse au client. |

Exporter vers Sheets

### C. Le Système de Modules (Frontend & Backend)

Le système de modules permet d'organiser le code dans des fichiers séparés et réutilisables.

| Concept | Description | Côté Client (React) | Côté Serveur (Express/Node) |
| --- | --- | --- | --- |
| **Modules ES6** | Utilise les mots-clés import et export. C'est la norme moderne pour tout le JavaScript. | **Standard de facto** pour importer et exporter les composants et les fonctions utilitaires. | Souvent utilisé dans les projets Node.js récents (via Babel ou ES Modules). |
| **CommonJS** | Utilise require() et module.exports. C'est le système de modules historique de Node.js. | Moins fréquent, car le Frontend utilise nativement ES6. | Encore très présent, notamment dans les fichiers de configuration (Ex: module.exports = { ... };). |

Exporter vers Sheets

**Conclusion :** La maîtrise de l'ES6+ (fonctions fléchées, destructuration, spread) et des outils asynchrones (async/await) est le pont qui relie l'écriture de composants React propres et efficaces à la construction de routes Express robustes et performantes.