Dossier/Fichier	Description
/app	Contient le code principal de l'application (logique métier, contrôleurs, services, modèles).
/config	Contient les fichiers de configuration du projet (base de données, cache, services, API, etc.).
/public	Dossier où sont stockés les fichiers accessibles publiquement (images, JavaScript, CSS, index.html).
/resources	Contient les vues (templates HTML), les fichiers de localisation (traductions) et les autres ressources.
/routes	Contient les fichiers définissant les routes (URLs) de l'application et leur logique (contrôleurs).
/storage	Contient les fichiers générés par l'application (logs, fichiers téléchargés, cache, etc.).
/tests	Contient les tests unitaires et d'intégration de l'application.
/database	Contient les migrations, les factories et les seeders (pour peupler la base de données).
/vendor	Contient les dépendances externes du projet (généré par le gestionnaire de paquets, comme Composer ou npm).
/node_modules	Dossier créé par npm pour les projets JavaScript, contenant les modules installés.
/docs	Dossier pour la documentation du projet (facultatif).
/build	Dossier pour les fichiers générés pour le déploiement (par exemple, les fichiers minifiés, compilés).

Description de chaque section :

- **/app** : Contient la logique métier de l'application, comme les contrôleurs, les modèles, et les services. C'est ici que se trouvent les fichiers qui gèrent l'interaction avec l'utilisateur et le traitement des données.
- **/config** : Contient les fichiers de configuration pour les différents services de l'application (base de données, cache, session, etc.).
- **/database** : Contient les migrations de base de données, qui définissent la structure des tables. Il peut aussi y avoir des factories pour générer des données factices et des seeders pour remplir la base de données.
- **/public** : Ce répertoire contient tous les fichiers accessibles au public, comme les images, les fichiers CSS et JavaScript. Le fichier principal est généralement `index.php` ou `index.html`, qui est le point d'entrée de l'application.
- **/resources** : Ce dossier contient les vues (templates HTML), les traductions (fichiers de langue), et d'autres ressources telles que les fichiers CSS ou JavaScript non compilés.
- **/routes** : Les routes définissent les URL de l'application et les actions associées. Par exemple, quel contrôleur ou fonction doit être appelé lorsqu'un utilisateur accède à une page donnée.
- **/storage** : Ce répertoire contient les fichiers générés par l'application, comme les logs, les fichiers temporaires, et les fichiers téléchargés par les utilisateurs.
- **/tests** : Contient les tests unitaires et d'intégration de l'application, ce qui permet de valider le bon fonctionnement de l'application pendant son développement.
- **/vendor** : Contient toutes les dépendances installées via un gestionnaire de paquets comme Composer (pour PHP) ou npm (pour JavaScript).
- **/build** : Fichiers générés lors de la compilation ou de la minification du code, comme dans les projets utilisant des frameworks front-end comme Vue.js ou React.js.

.

Critère	SQL	NoSQL
Modèle	Relationnel (tables, colonnes)	Non relationnel (clé-valeur, document, colonne, graphe)
Schéma	Fixe (schéma défini)	Flexible (schéma libre)
Scalabilité	Scalabilité verticale (ajout de ressources au serveur)	Scalabilité horizontale (ajout de serveurs)
Transactions	ACID (garantie d'intégrité)	Pas toujours ACID, parfois BASE (Basically Available, Soft state, Eventually consistent)
Performance	Moins performant sur de grandes quantités de données	Haute performance sur les données non structurées ou volumineuses
Cas d'utilisation	Applications transactionnelles, gestion de données structurées	Applications nécessitant de la flexibilité, Big Data, réseaux sociaux, IoT

Critère	Tests unitaires (Unit Testing)	Tests d'intégration (Integration Testing)
Objectif	Tester une petite unité de code (fonction, méthode) de manière isolée	Tester l'interaction entre plusieurs composants ou systèmes
But	Vérifier qu'une fonction ou méthode fonctionne correctement	Vérifier que les différentes parties de l'application fonctionnent bien ensemble
Composants testés	Une seule fonction ou méthode	Plusieurs composants ou modules (ex : base de données, API)
Dépendances	Utilisation de "mock" ou "stub" pour simuler les dépendances	Tests réels impliquant plusieurs composants (ex : appel à une API)
Vitesse d'exécution	Rapides	Plus lents, car ils impliquent plusieurs composants
Exemple	Tester une fonction qui additionne deux nombres	Tester l'enregistrement d'un utilisateur dans une base de données via une API
Exemple de code (JavaScript)	`js function add(a, b) { return a + b; } test('additionne 2 et 3 pour donner 5', () => { expect(add(2, 3)).toBe(5); });`	`js const request = require('supertest'); const app = require('./app'); test('POST /user crée un utilisateur', async () => { const response = await request(app).post('/user').send({ name: 'Alice', email: 'alice@example.com' }); expect(response.status).toBe(201); exp response.body).toHaveProperty('id'); });`

Protocole	Fonction	Usage principal	Caractéristiques clés
HTTP/HTTPS	Transfert de pages web et de ressources	Navigation web	Requêtes (GET, POST) et réponses ; HTTPS est sécurisé (SSL/TLS).
FTP	Transfert de fichiers	Gestion des fichiers sur un serveur	Simple mais non sécurisé par défaut.
SMTP	Envoi d'emails	Communication entre serveurs de messagerie	Utilisé avec IMAP/POP3 pour un service complet.
IMAP	Lecture des emails sur le serveur	Accès aux emails depuis plusieurs appareils	Synchronisation en temps réel.
POP3	Téléchargement des emails	Récupération locale des emails	Supprime souvent les emails du serveur.
DNS	Résolution de noms de domaine	Conversion des noms de domaine en adresses IP	Essentiel pour la navigation web.
TCP/IP	Fiabilité et routage des communications	Transmission des données sur le réseau	TCP garantit l'intégrité, IP s'occupe du routage.
WebSocket	Communication bidirectionnelle en temps réel	Chat, notifications, jeux en ligne	Réduit la latence pour des interactions fluides.