Table des matières

**Tapez le titre du chapitre (niveau 1)1**

Tapez le titre du chapitre (niveau 2)2

Tapez le titre du chapitre (niveau 3)3

**Tapez le titre du chapitre (niveau 1)4**

Tapez le titre du chapitre (niveau 2)5

Tapez le titre du chapitre (niveau 3)6

**Introduction** :

Le cahier d’analyse, axé sur l'Analyse du Système a établi les bases en définissant rigoureusement les exigences fonctionnelles du système, en identifiant les acteurs clés ainsi que les scénarios d'utilisation pertinents, tout en explorant en détail les contraintes opérationnelles et les besoins spécifiques du client. Pourtant, afin de transcender de la conception conceptuelle à une solution technique tangible, il est impératif de plonger plus profondément dans les détails de la conception.

C'est ici que le cahier de conception entre en jeu. Contrairement à l'approche orientée vers le Cahier des Charges (CDC) du cahier d’analyse, le cahier de conception se concentre sur la traduction des spécifications fonctionnelles en une architecture technique robuste, en exploitant les meilleures pratiques de conception logicielle et en choisissant judicieusement les technologies les plus appropriées.

Dans cette introduction, nous examinerons en détail l'architecture future du système, en mettant particulièrement l'accent sur les aspects serveur et client, ainsi que sur les défis techniques identifiés et les solutions proposées pour les surmonter. Nous aborderons également les maquettes de l'interface utilisateur (GUI) afin de concrétiser les spécifications fonctionnelles définies dans le cahier d’analyse. Enfin, nous détaillerons les différents diagrammes et éléments de conception qui composent ce document, démontrant ainsi notre approche méthodique et complète pour transformer les exigences en une solution technique viable.

Par conséquent, le cahier de conception représente une étape cruciale dans le processus de développement, guidant le cheminement de la conceptualisation à l'implémentation, et contribuant ainsi à la réalisation réussie du projet dans son ensemble.

# **Architecture Future** :

L'architecture à venir du système constitue les fondements techniques sur lesquels notre application sera développée et opérationnelle. Il est essentiel de faire des choix architecturaux adaptés, aussi bien côté serveur que côté client, afin de garantir la flexibilité, la robustesse et les performances globales du système.

Du côté serveur, notre application reposera sur un environnement PHP avec le framework Laravel. Le choix de Laravel comme framework backend offre une multitude d'avantages, notamment sa structure modulaire, sa facilité de prise en main, ainsi que ses nombreuses fonctionnalités intégrées qui simplifient le développement et améliorent la productivité des développeurs.

Le Document D2 offre une vue détaillée de la conception technique de l'application, mettant en lumière les choix architecturaux et les technologies sélectionnés pour répondre aux besoins spécifiques du projet. En nous appuyant sur le modèle MVC et une base de données relationnelle SQL, notre objectif est de fournir une architecture serveur robuste et sécurisée, soutenue par le Framework Laravel pour PHP.

## **Serveur** :

L'architecture serveur sera structurée selon le modèle MVC, qui divise l'application en trois composants majeurs : le Modèle, la Vue et le Contrôleur. Cette approche permet une séparation claire des préoccupations et facilite la maintenance et l'évolutivité de l'application.

* **Modèle(Model)** : Cette couche est responsable de la manipulation des données et des interactions avec la base de données. Nous utiliserons une base de données relationnelle SQL pour garantir l'intégrité des données et les transactions sécurisées. Les modèles dans Laravel fourniront une représentation des données ainsi que les règles de validation nécessaires.
* **Vue(View)** : La couche de vue est chargée de la présentation des données aux utilisateurs. Dans notre cas, les vues seront créées en utilisant les fonctionnalités de templating de Laravel, permettant ainsi une gestion efficace de l'interface utilisateur.
* **Contrôleur(Controller)** : Le contrôleur agit comme un intermédiaire entre le modèle et la vue. Il capture les activités de l'utilisateur et, en fonction de celles-ci, déclenche les changements nécessaires dans l'application. Le contrôleur traite les actions utilisateur (via des requêtes HTTP), demande au modèle d'effectuer les opérations nécessaires, puis transmet les données à la vue pour affichage.

Cette séparation des composants en trois catégories distinctes simplifie l'architecture des dossiers de l'application, offrant ainsi une meilleure organisation et une meilleure maintenabilité du code. La figure ci-dessous illustre l'architecture MVC de Laravel, décrivant clairement la répartition des responsabilités entre les différents composants.

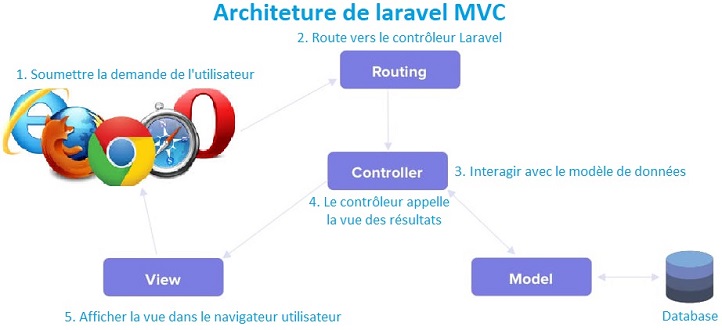


Figure l’architecture MVC de Laravel

1. **Le model :** contient les données et leur logique

Le diagramme de classes décrit les différentes entités du système ainsi que leurs attributs et associations. Les principales classes incluent :

* **Client** :

Cette classe représente les utilisateurs de l'application. Chaque client est identifié de manière unique par un ID\_client et possède des informations telles que son nom, adresse e-mail et son mot de passe.

* **Produit :**

Cette classe représente les différents produits de cuisine disponibles dans notre catalogue. Chaque produit est identifié par un ID\_produit et possède des attributs tels que son nom, sa description, son prix unitaire, etc.

* **Commande :**

Cette classe représente les commandes passées par les clients. Chaque commande est identifiée par un ID\_commande et contient des détails tels que la date de commande, le montant total, etc. Elle est associée à un client spécifique.

Chaque classe est associée à des attributs spécifiques qui capturent les caractéristiques pertinentes de chaque entité. Ces attributs sont cruciaux pour stocker et manipuler les données de manière efficace dans le système.

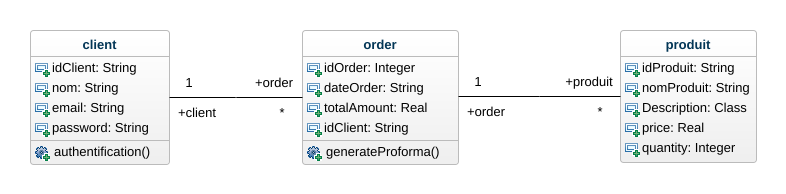


Figure : Diagramme de classes(Model)

Ensemble, ce diagramme de classes offre une représentation structurée et complète de l'architecture du système, ce qui facilitera la compréhension et la gestion du développement de l'application.

1. **La view :** contient la présentation graphique à renvoyer à l’utilisateur.
2. **Le controller :**  traite les actions utilisateur (via des requêtes), demande au modèle d’effectuer les changements, puis passe les données à la vue.

* Interaction Client-Système :

Cette section décrit le flux d'interaction entre le client et le système lors de la conception et de la commande de produits de cuisine personnalisés. Elle détaille les différentes étapes que le client traverse, de l'authentification à la confirmation de la commande, en passant par la consultation des éléments de cuisine, la modélisation de la cuisine et la gestion du panier. Chaque étape est expliquée en détail, y compris les actions effectuées par le client et les réponses du système.

* Authentification :

Le processus commence par l'authentification du client, où il fournit ses identifiants de connexion pour accéder à son compte utilisateur. Cela garantit la sécurité et la confidentialité des données personnelles du client.

* Consultation des Éléments de Cuisine :

Une fois authentifié, le client peut consulter les différents éléments de cuisine disponibles dans le catalogue du système. Il peut parcourir les options et sélectionner les articles qui l'intéressent pour sa conception personnalisée.

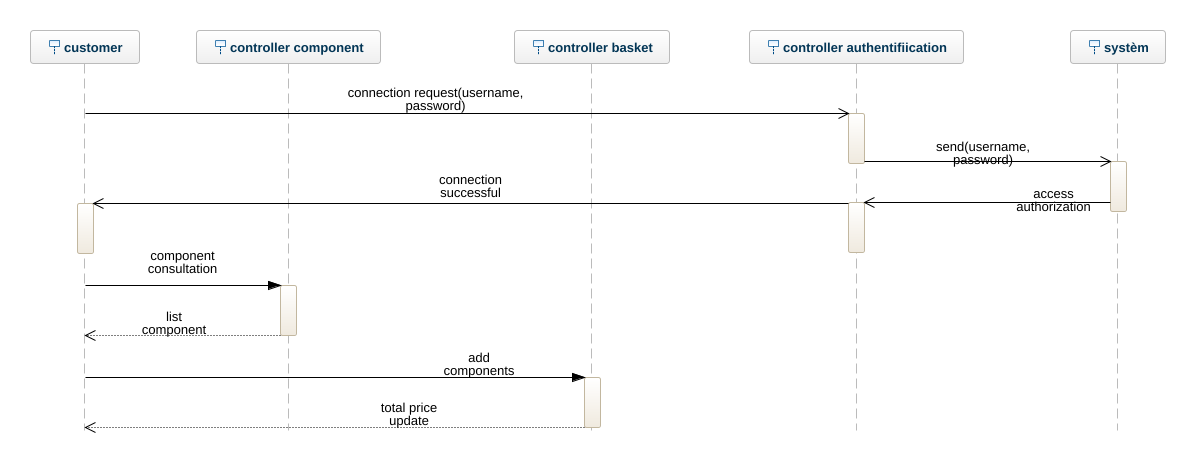
* Modélisation de la Cuisine :

Après avoir choisi les éléments de cuisine nécessaires, le client utilise les outils de modélisation pour placer les articles sélectionnés dans l'espace de travail virtuel. Cette étape lui permet de visualiser et de planifier la disposition de sa cuisine selon ses préférences.

* Gestion du Panier :

Les éléments sélectionnés sont automatiquement ajoutés au panier, où le client peut consulter le contenu et ajuster les quantités si nécessaires. Il peut également retirer des articles du panier s'il change d'avis ou souhaite apporter des modifications à sa commande.

Cette description détaillée du processus d'interaction client-système fournit une compréhension approfondie du flux de travail et des fonctionnalités offertes par l'application. Elle met en évidence les différentes étapes clés du processus de conception et de commande, offrant ainsi une référence claire pour les utilisateurs et les développeurs.



* + - **Le routing** : Bien qu’indépendant de l’architecture MVC, le routing fait partie intégrante de tous les frameworks PHP.

## **Fonctionnement de Laravel :**

### **Introduction à Laravel** :

Laravel est un framework d'application Web reconnu pour sa syntaxe expressive et élégante. En fournissant une structure robuste et un point de départ pour le développement, Laravel permet aux développeurs de se concentrer sur la création d'applications exceptionnelles tout en s'occupant des détails techniques.

### **Choix de Laravel :**

Dans l'océan des outils et des frameworks disponibles pour le développement Web, Laravel se démarque comme le choix idéal pour la création d'applications Web modernes et complètes. Son caractère progressif et évolutif, ainsi que sa communauté dynamique en font un choix privilégié des développeurs.

* Progressif et évolutif :

Laravel est qualifié de framework "progressif", car il convient aussi bien aux débutants qu'aux développeurs expérimentés. Avec une documentation exhaustive, des guides et des didacticiels vidéo, Laravel accompagne les novices dans leur apprentissage tout en offrant des fonctionnalités avancées telles que l'injection de dépendances, les tests unitaires, les files d'attente, et bien plus encore, aux développeurs expérimentés.

* Evolutif et adapté aux entreprises :

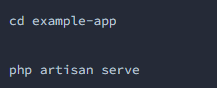
Laravel est conçu pour être hautement évolutif, capable de gérer efficacement des charges de travail importantes grâce à sa compatibilité avec des technologies telles que Redis et Laravel Vapor. Que ce soit pour des applications modestes ou des projets d'entreprise à grande échelle, Laravel offre les outils nécessaires pour répondre aux besoins de développement.

* Configuration et mise en route :

Avant de commencer à créer un projet Laravel, il faut s’assurer d'avoir PHP et Composer installés sur sa machine. Une fois installés, on peut créer un nouveau projet Laravel en utilisant la commande :



ou en installant le programme d'installation de Laravel via Composer. Une fois le projet créé, démarrez le serveur de développement local à l'aide de la commande :



* Configuration avancée :

Laravel offre une grande flexibilité en matière de configuration, permettant de personnaliser divers aspects de l’application, tels que la base de données, les services de messagerie, le fuseau horaire, etc. Les fichiers de configuration sont stockés dans le répertoire ***config***, et les valeurs peuvent être définies à l'aide du fichier ***.env***.

* Sécurité et chiffrement :

La sécurité est une priorité dans Laravel. Les fichiers d'environnement (***.env*)** contenant des informations sensibles ne doivent pas être stockés dans le contrôle de code source. Laravel offre des fonctionnalités de chiffrement pour protéger ces fichiers sensibles, permettant ainsi une gestion sécurisée des données d'environnement.

* Gestion de la configuration :

Laravel propose plusieurs outils pour gérer efficacement la configuration de l'application. Il est recommandé de mettre en cache les fichiers de configuration pour améliorer les performances de l'application. De plus, la commande ***config :publish*** permet de publier les fichiers de configuration non inclus par défaut, offrant ainsi une plus grande flexibilité aux développeurs.

Laravel offre une expérience de développement Web exceptionnelle, combinant puissance, flexibilité et sécurité. Avec une communauté active et des fonctionnalités avancées, Laravel est le choix idéal pour créer des applications Web modernes et évolutives.

* Structure du répertoire :

La structure par défaut des applications Laravel est destinée à fournir un excellent point de départ pour les grandes et petites applications. Laravel n'impose presque aucune restriction sur l'emplacement d'une classe donnée - tant que ***Composer*** peut charger automatiquement la classe.

* Le répertoire des applications :

Le répertoire des applications contient le code principal de l’application. Nous explorerons bientôt ce répertoire plus en détail ; cependant, presque toutes les classes de l’application se trouveront dans ce répertoire.

* Le répertoire d'amorçage :

Le répertoire contient le fichier qui amorce le framework. Ce répertoire héberge également un répertoire qui contient les fichiers générés par le framework pour l'optimisation des performances, tels que les fichiers de cache de route et de services.

* Le répertoire de configuration :

Le répertoire, comme son nom l'indique, contient tous les fichiers de configuration de l’application.

* Le répertoire de la base de données :

Le répertoire contient les migrations de bases de données, les fabriques de modèles et les graines. Il est possible si on le souhaite utiliser ce répertoire pour contenir une base de données SQLite.

* L'annuaire public :

Le répertoire contient le fichier, qui est le point d'entrée de toutes les requêtes entrant dans notre application et configure le chargement automatique. Ce répertoire héberge également nos actifs tels que les images, JavaScript et CSS.

* Le répertoire des ressources :

Le répertoire contient nos vues ainsi que nos ressources brutes non compilées telles que CSS ou JavaScript.

* L'annuaire des itinéraires :

Le ***root*** répertoire contient toutes les définitions de routes pour notre application. Par défaut, deux fichiers de route sont inclus avec Laravel **:  web.php et console.php**.

Le ***web.php*** fichier contient des routes que Laravel place dans le ***web*** groupe middleware, qui fournit l'état de session, la protection CSRF et le cryptage des cookies.

Le ***console.php*** fichier est l'endroit où nous pouvons définir toutes nos commandes de console basées sur la fermeture. Chaque fermeture est liée à une instance de commande permettant une approche simple pour interagir avec les méthodes IO de chaque commande. Même si ce fichier ne définit pas les routes HTTP, il définit les points d'entrée (routes) basés sur la console dans votre application.

Le ***api.php*** fichier contient des routes censées être sans état, de sorte que les demandes entrant dans l'application via ces routes sont destinées à être authentifiées via des jetons et n'auront pas accès à l'état de session.

Le ***channels.php*** fichier est l'endroit où nous pouvons enregistrer tous les canaux de diffusion d'événements pris en charge par notre application.

* Le répertoire de stockage :

Le répertoire contient nos journaux, modèles Blade compilés, sessions basées sur des fichiers, caches de fichiers et autres fichiers générés par le framework. Ce répertoire est divisé en répertoires. Le répertoire peut être utilisé pour stocker tous les fichiers générés par notre application. Le répertoire est utilisé pour stocker les fichiers et les caches générés par le framework. Enfin, le répertoire contient les fichiers journaux de notre application.

Le répertoire peut être utilisé pour stocker des fichiers générés par l'utilisateur, tels que des avatars de profil, qui doivent être accessibles au public.

* Le répertoire des tests :

Le ***tests*** répertoire contient nos tests automatisés. Des exemples de tests unitaires et de tests de fonctionnalités Pest ou PHPUnit sont fournis prêts à l'emploi. Chaque classe de test doit être suffixée par le mot ***test***. Vous pouvez exécuter vos tests à l'aide des ***commandes php artisan test***. Ou, si vous souhaitez une représentation plus détaillée et plus belle de vos résultats de tests, vous pouvez exécuter vos tests à l'aide de la commande.

* L'annuaire des fournisseurs :

Le ***vendor*** répertoire contient nos dépendances Composer.

* Le répertoire des applications :

Notre application est hébergée dans le ***app*** répertoire. Par défaut, ce répertoire est placé sous un espace de noms ***App*** et est chargé automatiquement par Composer à l'aide de la norme de chargement automatique PSR-4.

Par défaut, le répertoire contient les répertoires ***Http***, ***Models*** et ***Providers***. Cependant, au fil du temps, divers autres répertoires seront générés dans le répertoire de l'application lorsque nous utiliserons les commandes pour générer des classes. Par exemple, le ***app/Console*** répertoire n'existera pas tant que nous n'exécuterons pas la commande pour générer une classe de commandes.

Les répertoires ***Console*** et ***Http*** sont expliqués plus en détail dans leurs sections respectives ci-dessous, mais considérons les répertoires ***Console*** et ***Http*** comme fournissant une API au cœur de notre application. Le protocole HTTP et la CLI sont tous deux des mécanismes permettant d'interagir avec notre application, mais ne contiennent pas réellement de logique d'application. En d’autres termes, il s’agit de deux manières d’émettre des commandes à notre application. Le répertoire ***Console*** contient toutes nos commandes Artisan, tandis que le répertoire ***Http*** contient nos contrôleurs, middleware et requêtes.

* L'annuaire de radiodiffusion :

Le ***Broadcasting*** répertoire contiendra toutes les classes de canaux de diffusion pour notre application. Ces classes seront générées à l'aide de la ***make :channel*** commande. Ce répertoire n'existe pas par défaut, mais sera créé lors de la création de notre première chaîne.

* Le répertoire de la console :

Le ***Console*** répertoire contient toutes les commandes Artisan personnalisées pour notre application. Ces commandes peuvent être générées à l'aide de la ***make:command*** commande.

* L'annuaire des événements :

Ce répertoire n'existe pas par défaut, mais sera créé par les commandes ***event:generate*** et ***make:event*** . Le répertoire héberge les classes d'événements. Les événements peuvent être utilisés pour alerter d'autres parties de notre application qu'une action donnée s'est produite, offrant ainsi une grande flexibilité et un découplage.

* Le répertoire des exceptions :

Le ***Exceptions*** répertoire contiendra toutes les exceptions personnalisées de notre application. Ces exceptions pourrons être générées à l'aide de la ***make:exception*** commande.

* L'annuaire http :

Le ***Http*** répertoire contient nos contrôleurs, middleware et demandes de formulaire. Presque toute la logique permettant de gérer les requêtes entrant dans notre application sera placée dans ce répertoire.

* L'annuaire des emplois :

Ce répertoire n'existe pas par défaut, mais sera créé si nous exécutons la ***make:job*** commande. Le répertoire héberge les tâches pouvant être mises en file d'attente pour l’application. Les tâches peuvent être mises en file d'attente par l’application ou exécutées de manière synchrone au cours du cycle de vie actuel de la demande. Les tâches qui s'exécuterons de manière synchrone pendant la requête en cours sont parfois appelées « commandes » car elles constituent une implémentation du modèle de commande.

* L'annuaire des auditeurs :

Ce répertoire n'existe pas par défaut, mais sera créé si nous exécutons la commande ***event:generate*** ou ***make:listener*** Artisan. Le ***Listenersrépertoire*** répertoire contiendra les classes qui gèrent nos événements. Les écouteurs d'événements reçoivent une instance d'événement et exécutent une logique en réponse à l'événement déclenché. Par exemple, un événement peut être géré par un écouteur.

* L'annuaire des modèles :

Le ***Models*** répertoire contient toutes vos classes de modèles. L'ORM Eloquent inclus avec Laravel fournit une implémentation ActiveRecord belle et simple pour travailler avec notre base de données. Chaque table de base de données possède un « Modèle » correspondant qui est utilisé pour interagir avec cette table. Les modèles nous permettent d'interroger des données dans nos tables, ainsi que d'insérer de nouveaux enregistrements dans la table.

* Le répertoire des règles :

Ce répertoire n'existe pas par défaut, mais sera créé si nous exécutons la commande ***make:rule***. Le ***Rules*** répertoire contient les objets de règle de validation personnalisée pour notre application. Les règles sont utilisées pour encapsuler une logique de validation complexe dans un objet simple.

## **Client :**

Pour la partie visuelle et la modélisation, notre choix se porte sur l'utilisation de Three.js une bibliothèque JavaScript spécialisée dans la création d'animations et de graphismes 3D dans les navigateurs web. Three.js offre une panoplie de fonctionnalités permettant de concevoir des environnements 3D interactifs de manière efficace et intuitive.

L'échange de données entre le serveur et la technologie Three.js se fera via le format JSON (JavaScript Object Notation), reconnu pour sa légèreté et sa facilité d'utilisation. JSON facilitera la communication entre le backend et le frontend, simplifiant ainsi le transfert et la manipulation des données nécessaires à la visualisation 3D.

### **Fonctionnement de Three.js :**

Three.js exploite la technologie WebGL pour exploiter la puissance du processeur graphique (GPU) de l'appareil de l'utilisateur, ce qui permet de générer des environnements 3D interactifs en temps réel. Son fonctionnement se décompose en plusieurs étapes :

* **Initialisation** : Une scène est créée pour contenir tous les objets 3D. Une caméra est configurée pour définir le point de vue de l'utilisateur dans la scène.
* **Création d'objets** : Des objets géométriques tels que des cubes, des sphères ou des plans, ainsi que des matériaux pour leur apparence visuelle, sont créés. Des modèles 3D peuvent également être chargés à partir de fichiers externes.
* **Positionnement et animation** : Les objets sont positionnés dans la scène et des animations peuvent être ajoutées pour créer des interactions dynamiques avec l'utilisateur.
* **Rendu** : Three.js prend en charge le rendu de la scène en utilisant WebGL pour générer les images finales à afficher dans le navigateur.

### **Structure JSON :**

JSON est un format de données léger utilisé pour l'échange de données entre un serveur et un client. Il est basé sur des paires clé-valeur et offre une grande flexibilité. Voici un exemple de sa structure de base :

* **Format de données** : JSON est un format de données texte composé de paires clé-valeur. Il est inspiré de la syntaxe des objets JavaScript, ce qui le rend facile à comprendre pour les développeurs familiarisés avec JavaScript.
* **Structuration des données** : Les données JSON sont organisées en objets, tableaux et valeurs scalaires. Un objet JSON est délimité par des accolades **{}** et contient des paires clé-valeur séparées par des virgules. Un tableau JSON est délimité par des crochets **[]** et contient une liste ordonnée de valeurs. Les valeurs scalaires peuvent être des chaînes de caractères, des nombres, des booléens, des null, etc.
* **Flexibilité** : JSON est un format très flexible qui peut représenter une grande variété de structures de données. Il peut être utilisé pour représenter des données simples telles que des configurations ou des messages, ainsi que des structures de données complexes telles que des objets imbriqués et des tableaux multidimensionnels.
* **Interchangeabilité** : JSON est indépendant du langage et peut être utilisé dans de nombreux environnements de programmation. Il est largement pris en charge par les langages de programmation modernes, ce qui facilite l'échange de données entre différentes applications et plateformes.
* **Lisibilité** : En raison de sa syntaxe simple et de sa structure intuitive, JSON est facile à lire et à écrire pour les humains. Cela en fait un choix populaire pour le stockage de données et les échanges de données entre les systèmes.

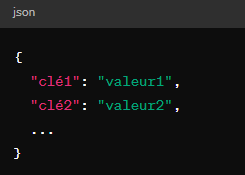


Figure  : Structure JSON

### **Échange de données avec Three.js :**

L'échange de données entre le serveur et la technologie Three.js se fera en utilisant le format JSON. Par exemple, le serveur peut envoyer des données sur les modèles 3D à afficher dans la scène, telles que les coordonnées des sommets, les couleurs des matériaux ou les informations sur les animations.

Une fois les données reçues côté client, elles seront analysées et utilisées pour créer les objets 3D correspondants dans la scène Three.js. Les informations provenant du serveur peuvent également être mises à jour dynamiquement pour refléter les changements dans la scène, tels que les mouvements d'animation ou les interactions de l'utilisateur.

En résumé, Three.js utilise JSON comme format de données pour l'échange d'informations avec le serveur, permettant ainsi une communication efficace et une mise à jour dynamique de la scène 3D dans le navigateur web de l'utilisateur.

# **Verrous Techniques et Solutions :**

Dans le cadre du développement de notre application, plusieurs défis techniques doivent être surmontés pour garantir son bon fonctionnement et ses performances optimales. Voici les principaux verrous techniques identifiés, ainsi que les solutions envisagées pour les résoudre.

### **Sauvegarde du Modèle :**

### **Défi :**

Assurer la sauvegarde et la restauration efficaces des données du modèle, notamment les conceptions de cuisine, les informations sur les produits et les préférences des utilisateurs.

### **Solution :**

Mettre en place une stratégie de sauvegarde régulière des données, en utilisant des outils et des technologies de sauvegarde fiables. Utiliser des bases de données robustes et des mécanismes de gestion des transactions pour garantir l'intégrité des données et la récupération en cas de panne.

### **Exécution des Composants Côté Client :**

### **Défi :**

Optimiser les performances des composants exécutés côté client, tels que la visualisation 3D et les interactions utilisateur, pour garantir une expérience fluide et réactive.

### **Solution :**

Utiliser des techniques de programmation efficaces et des optimisations de code pour réduire la charge de travail côté client. Mettre en cache les ressources statiques et minimiser les requêtes réseau pour accélérer le chargement et l'exécution des composants. Adopter des pratiques de développement responsif pour s'assurer que l'application fonctionne de manière optimale sur une variété de dispositifs et de navigateurs.

### **Modélisation/Visualisation 3D :**

### **Défi :**

Réaliser une modélisation et une visualisation 3D fluides et réalistes des cuisines personnalisées, en intégrant des fonctionnalités avancées telles que la rotation, le zoom et la manipulation des objets.

### **Solution** :

Utiliser des bibliothèques et des frameworks spécialisés tels que Three.js pour créer des environnements 3D interactifs dans le navigateur. Optimiser les modèles 3D pour réduire la charge de rendu et maximiser les performances. Exploiter les capacités du GPU de l'appareil de l'utilisateur pour accélérer le rendu des scènes 3D.

### **Gestion des Éléments Non-Superposables :**

### **Défi :**

Les éléments de cuisine ont des formes et des dimensions spécifiques qui ne permettent pas leur superposition, ce qui complique leur disposition pour éviter les collisions.

### **Solution :**

Mettre en place des contraintes de positionnement pour chaque élément, utiliser des algorithmes de détection de collisions et fournir des guides visuels pour aider les utilisateurs à placer les éléments correctement.

### **Calcul Automatique des Éléments Sélectionnés :**

### **Défi :**

Calculer automatiquement les propriétés combinées des éléments sélectionnés, telles que les dimensions totales et les coûts estimés.

### **Solution :**

Implémenter des fonctions de calcul côté client basées sur les données des produits et des configurations stockées localement ou récupérées du serveur, avec des mécanismes de mise à jour dynamique pour refléter instantanément les changements dans la conception.

En surmontant ces défis, nous serons en mesure de développer une application robuste, performante et conviviale pour nos utilisateurs, offrant une expérience immersive et personnalisée dans la conception et l'aménagement de cuisines.