**Analyse de l'Application**

Votre application est un système de gestion complet pour un garage automobile, construit en Vue.js (probablement Vue 3 avec l'API de Composition). Elle gère l'ensemble du flux de travail, de la prise en charge du client à la facturation et à l'analyse des bénéfices.

**Fonctionnalités Clés :**

1. **Gestion des Entités de Base (CRUD) :**
   * **Clients** : Gestion des informations des clients.
   * **Véhicules** : Gestion des véhicules, liés à des clients. La page de gestion des véhicules ([VoituressssssPage.vue](code-assist-path:c:\Users\AlfMay\Downloads\gestion\gestion\src\pages\VoituressssssPage.vue)) est assez avancée, permettant d'ajouter dynamiquement des marques et des modèles.
   * **Fournisseseurs, Experts, Assurances** : Pages CRUD simples pour gérer ces contacts professionnels.
   * **Articles** : Gestion d'un catalogue de pièces détachées avec prix et fournisseur par défaut.
2. **Flux de Travail Central - Les "Affaires" :**
   * L'**Affaire** est l'entité centrale qui représente un dossier de réparation pour un véhicule.
   * Elle est liée à un client, un véhicule, une assurance et un expert.
   * Le **Tableau de Bord** ([HomePage.vue](code-assist-path:c:\Users\AlfMay\Downloads\gestion\gestion\src\pages\HomePage.vue)) offre une vue d'ensemble des affaires en cours, avec un système de filtres et une barre de progression.
3. **Gestion des Devis (**[**DevisPage.vue**](code-assist-path:c:\Users\AlfMay\Downloads\gestion\gestion\src\pages\DevisPage.vue)**) :**
   * Interface complexe pour créer des devis pour une affaire.
   * Permet d'ajouter des **articles** (pièces) et de spécifier le type d'intervention (Remplacement, Occasion, etc.).
   * Calcule automatiquement les totaux (HT, TVA, TTC) en fonction des articles et des coûts de main-d'œuvre (Tôlerie, Peinture, etc.).
   * Génère un PDF du devis pour l'impression.
4. **Gestion des Bons de Commande (**[**BonsCommandePage.vue**](code-assist-path:c:\Users\AlfMay\Downloads\gestion\gestion\src\pages\BonsCommandePage.vue)**) :**
   * Fonctionnalité clé : un **bon de commande est créé automatiquement lorsqu'un devis passe au statut "accepté"**.
   * S'il y a plusieurs fournisseurs pour les pièces d'un même devis, l'application génère un bon de commande distinct pour chaque fournisseur.
   * L'interface permet un suivi détaillé du statut de chaque bon de commande et même de chaque article commandé (En attente, Reçu, etc.).
5. **Analyse et Rapports :**
   * **Comparaison (**[**Comparison.vue**](code-assist-path:c:\Users\AlfMay\Downloads\gestion\gestion\src\views\Comparison.vue)**)** : Compare les coûts prévus dans le devis aux dépenses réelles enregistrées pour une affaire.
   * **Bénéfices (**[**Profits.vue**](code-assist-path:c:\Users\AlfMay\Downloads\gestion\gestion\src\views\Profits.vue)**)** : Calcule la rentabilité de chaque affaire en comparant les revenus (du devis) et les dépenses.
   * **Galerie Photos (**[**PhotoGallery.vue**](code-assist-path:c:\Users\AlfMay\Downloads\gestion\gestion\src\views\PhotoGallery.vue)**)** : Centralise toutes les photos (véhicules, affaires avant/après réparation) avec des outils de filtrage.
6. **Paramètres et Données (**[**Settings.vue**](code-assist-path:c:\Users\AlfMay\Downloads\gestion\gestion\src\views\Settings.vue)**) :**
   * Fonctions d'**import/export** de toutes les données en JSON, agissant comme un système de sauvegarde/restauration.
   * Statistiques globales sur le nombre d'enregistrements.
   * La persistance des données se fait côté client, probablement via localStorage ou, de manière plus robuste, via **IndexedDB** (ce qui est plus adapté pour ce volume de données relationnelles).

**Schéma de Base de Données et Relations**

Pour construire cette application de manière robuste, une base de données relationnelle (côté client comme IndexedDB/Dexie.js, ou côté serveur comme SQLite/PostgreSQL) est idéale. Voici un schéma possible :

1. **clients**
   * id (PK)
   * nom, prenom, adresse, telephone
2. **voitures**
   * id (PK)
   * immatriculation (UNIQUE), marque, modele, num\_chassis, date\_mec
   * client\_id (FK -> clients.id)
3. **fournisseurs**
   * id (PK)
   * nom, adresse, telephone
4. **experts** & **assurances** (schémas similaires à fournisseurs)
5. **articles** (Catalogue de pièces)
   * id (PK)
   * Description, PrixHT
   * Fournisseur (FK -> fournisseurs.id)
6. **affaires** (Le coeur du système)
   * id (PK)
   * numero\_affaire (UNIQUE), date\_ouverture, statut
   * client\_id (FK -> clients.id)
   * voiture\_id (FK -> voitures.id)
   * assurance\_id (FK -> assurances.id)
   * expert\_id (FK -> experts.id)
7. **devis**
   * id (PK)
   * numero\_devis (UNIQUE), date\_devis, statut
   * affaire\_id (FK -> affaires.id)
   * montant\_ht, montant\_tva, montant\_ttc
   * MO\_tolerie, MO\_peinture, MO\_mecanique, MO\_electrique (coûts de main d'oeuvre)
8. **devis\_articles** (Table de liaison pour les articles d'un devis)
   * id (PK)
   * devis\_id (FK -> devis.id)
   * article\_id (FK -> articles.id, peut être nul pour un article personnalisé)
   * description, quantite, prix\_ht (prix au moment du devis)
   * fournisseur\_id (FK -> fournisseurs.id)
   * intervention (ex: 'REMPLACEMENT')
9. **bons\_commande**
   * id (PK)
   * numero\_bon (UNIQUE), date\_commande, statut
   * devis\_id (FK -> devis.id)
   * fournisseur\_id (FK -> fournisseurs.id)
   * montant\_total
10. **bon\_commande\_articles** (Articles contenus dans un bon de commande)
    * id (PK)
    * bon\_commande\_id (FK -> bons\_commande.id)
    * devis\_article\_id (FK -> devis\_articles.id)
    * description, quantite, prix\_unitaire
    * statut\_id (1: En attente, 2: Commandé, 3: Reçu, 4: Terminé)
    * date\_reception, notes
11. **depenses** (Pour le calcul des bénéfices)
    * id (PK)
    * affaire\_id (FK -> affaires.id)
    * type, description, montant, date
    * fournisseur\_id (FK -> fournisseurs.id)

**Prompts Exacts pour la Génération de l'Application**

Voici une série de prompts que vous pouvez donner à une IA comme Gemini pour recréer cette application, étape par étape.

**Prompt 1 : Initialisation du Projet et Entités de Base**

Je veux créer une application de gestion de garage avec Vue 3 (Composition API) et JavaScript. Utilise Dexie.js (un wrapper IndexedDB) pour la base de données côté client.

\*\*Tâche 1 : Définis le schéma de base de données suivant avec Dexie.js :\*\*

- `clients`: `++id, nom, prenom, adresse, telephone`

- `voitures`: `++id, immatriculation, marque, modele, num\_chassis, date\_mec, client\_id`

- `fournisseurs`: `++id, nom, adresse, telephone`

- `experts`: `++id, nom, prenom`

- `assurances`: `++id, nom, adresse, telephone`

- `articles`: `++id, Description, PrixHT, Fournisseur`

\*\*Tâche 2 : Crée les pages Vue suivantes pour la gestion CRUD (Créer, Lire, Mettre à jour, Supprimer) de ces entités :\*\*

1. `ClientsPage.vue`

2. `FournisseursPage.vue`

3. `ExpertsPage.vue`

4. `AssurancesPage.vue`

5. `ArticlesPage.vue`

Chaque page doit contenir :

- Un bouton pour ouvrir un formulaire modal (ou dans la page) pour ajouter/modifier une entrée.

- Un tableau pour lister toutes les entrées.

- Des boutons sur chaque ligne du tableau pour "Modifier" et "Supprimer".

- Utilise un style sobre et professionnel.

**Prompt 2 : Gestion des "Affaires" et du Tableau de Bord**

\*\*Tâche 1 : Ajoute les tables suivantes au schéma Dexie.js :\*\*

- `affaires`: `++id, &numero\_affaire, date\_ouverture, statut, client\_id, voiture\_id, assurance\_id, expert\_id`

- `photos`: `++id, url, name, date, entity\_type, entity\_id, photo\_type`

\*\*Tâche 2 : Crée la page `HomePage.vue` qui servira de tableau de bord.\*\*

- Elle doit afficher une grille de cartes, où chaque carte représente une "affaire".

- Chaque carte doit montrer : le numéro d'affaire, le nom du client, le véhicule, l'expert, l'assurance, la date d'ouverture et un badge coloré pour le statut (`ouvert`, `en-cours`, `prepare`, `envoye`, `accepte`, `rejete`).

- Ajoute une barre de progression sur chaque carte qui reflète l'avancement en fonction du statut.

- Ajoute des filtres en haut de la page pour trier les affaires par statut.

\*\*Tâche 3 : Crée la page `GestionVoiturePage.vue` pour le CRUD des véhicules.\*\*

- Le formulaire doit avoir des listes déroulantes pour la marque et le modèle.

- Le champ "Client" doit être une liste déroulante peuplée par la table `clients`.

**Prompt 3 : Création des Devis Complexes**

Maintenant, créons le module de devis.

\*\*Tâche 1 : Ajoute ces tables au schéma Dexie.js :\*\*

- `devis`: `++id, &numero\_devis, date\_devis, statut, affaire\_id, montant\_ht, montant\_tva, montant\_ttc, MO\_tolerie, MO\_peinture, MO\_mecanique, MO\_electrique`

- `devis\_articles`: `++id, devis\_id, article\_id, description, quantite, prix\_ht, fournisseur\_id, intervention`

\*\*Tâche 2 : Crée la page `DevisPage.vue`.\*\*

- La page doit lister les devis existants dans un tableau.

- Un bouton "Ajouter un devis" doit afficher un formulaire complexe.

- \*\*Le formulaire de devis doit permettre de :\*\*

1. Sélectionner une "affaire" existante.

2. Ajouter dynamiquement des lignes d'articles. Pour chaque ligne :

- Champ de texte pour la description de l'article (avec autocomplétion depuis la table `articles`).

- Liste déroulante pour le type d'intervention ('REMPLACEMENT', 'OCCASION', etc.).

- Champ pour la quantité et le prix HT.

- Liste déroulante pour sélectionner un fournisseur.

- Un bouton pour supprimer la ligne.

3. Avoir des champs pour les coûts de main-d'œuvre (Tôlerie, Peinture, etc.).

4. \*\*Calculer et afficher en temps réel les totaux (Total HT, TVA, Total TTC)\*\* en se basant sur la somme des articles et de la main-d'œuvre. La TVA est de 20%, sauf pour l'intervention 'OCCASION' où elle est de 0%.

5. Un bouton pour enregistrer le devis.

6. Un bouton pour générer un PDF du devis.

**Prompt 4 : Automatisation et Gestion des Bons de Commande**

Cette étape est cruciale : la conversion de devis en bons de commande.

\*\*Tâche 1 : Ajoute ces tables au schéma Dexie.js :\*\*

- `bons\_commande`: `++id, &numero\_bon, date\_commande, statut, devis\_id, fournisseur\_id, montant\_total`

- `bon\_commande\_articles`: `++id, bon\_commande\_id, devis\_article\_id, description, quantite, prix\_unitaire, statut\_id, date\_reception, notes`

\*\*Tâche 2 : Implémente la logique de conversion.\*\*

- Quand le statut d'un `devis` est mis à jour à \*\*"devis accepter"\*\*, déclenche une fonction qui :

1. Lit tous les `devis\_articles` de ce devis.

2. Regroupe les articles par `fournisseur\_id`.

3. Pour chaque fournisseur, crée une nouvelle entrée dans la table `bons\_commande`.

4. Pour chaque article du groupe, crée une entrée correspondante dans `bon\_commande\_articles` liée au bon de commande nouvellement créé.

\*\*Tâche 3 : Crée la page `BonsCommandePage.vue`.\*\*

- Affiche une liste de cartes, chaque carte représentant un `bon\_commande`.

- Chaque carte doit afficher les détails du bon (numéro, date, fournisseur, affaire/client associé) et son statut global (`brouillon`, `envoye`, `termine`).

- À l'intérieur de chaque carte, liste les `bon\_commande\_articles` associés.

- Pour chaque article, permets de \*\*modifier son statut de suivi\*\* (En attente, Commandé, Reçu, Terminé) via une liste déroulante.

- Ajoute une barre de progression sur la carte du bon de commande qui se met à jour en fonction du nombre d'articles "Terminés".