Collectivité Région Auvergne Rhône Alpes

RAPPORT DE CONCEPTION ET DÉVELOPPEMENT

Plateforme "Trouve ton artisan !"

TONY MORIEUX
18/09/2025

. Table des Matières :

8. Conclusion et Perspectives
9. Annexes (Lien GitHub, Lien Site en ligne)

1.Contexte du projet :

Ce rapport détaille la conception et la réalisation de la plateforme web 'Trouve ton artisan !', commanditée par la Région Auvergne-Rhône-Alpes.

1.1.Présentation de l'entreprise :

La Région Auvergne-Rhône-Alpes est une collectivité territoriale majeure, couvrant 12 départements dans le quart sud-est de la France. Dotée d'antennes à Clermont-Ferrand et Lyon, c'est cette dernière qui a initié la demande de développement de cette plateforme.

1.2.Problématique :

Face à l'essor significatif de l'artisanat dans la région (représentant près d'un tiers des entreprises locales), la Région Auvergne-Rhône-Alpes a identifié le besoin de faciliter la mise en relation entre les particuliers et les artisans. L'objectif est de permettre aux citoyens de trouver aisément un artisan et de solliciter des informations, d'où la nécessité d'un site web performant et accessible à tous.

Les attentes du client :

- Un site web pleinement accessible à tous les publics (jeunes, personnes âgées, personnes en situation de handicap).
- Une expérience utilisateur (UX) simple et intuitive.
- Conformité à la norme (WCAG 2.1).
- Un design 'mobile-first' garantissant une adaptabilité et une performance optimales sur une variété de dispositifs et de tailles d'écran.
- Cohérence graphique avec leur environnement numérique.
- Mise en œuvre des bonnes pratiques de sécurité informatique.
- Potentiel d'évolution de la plateforme, notamment par l'alimentation ultérieure des pages légales par un cabinet spécialisé et l'intégration future d'une application d'administration.

1.3.Objectifs :

Les objectifs principaux de cette plateforme sont les suivants :

- Faciliter la recherche d'artisans : Offrir aux utilisateurs un moyen simple de trouver des artisans par catégorie, spécialité ou par nom.
- Promouvoir l'artisanat locale : Présenter les profils et services des artisans de la région.
- Accessibilité : Design simple et intuitif, utilisation pour tous public en se conformant à la norme (WCAG 2.1).
- Optimisation numérique : Optimisation mobile : Assurer une navigation fluide et performante sur les appareils mobiles, en accord avec l'approche 'mobile-first'.
- Robustesse de l'API : Robustesse de l'infrastructure technique : Développer une API sécurisée et une base de données structurée, capable de gérer des données dynamiques et de supporter des évolutions futures.

1.4.Contraintes imposées :

Pour la réalisation de cette plateforme, les technologies et contraintes suivantes ont été imposées :

- Pour la maquette : Figma
- Pour la partie front-end : ReactJs, Bootstrap, Sass
- API : Node.js, Express, Sequelize, MySQL (technologie imposée pour le devoir). Pour le déploiement en ligne, PostgreSQL a été utilisé car Render ne propose pas de service MySQL gratuit.
- Versionning : Git et GitHub.
- Développement : Visual studio Code.
- Normes : WCAG 2.1 pour la conformité , W3C pour l'HTML et CSS.
- Sécurité : Sécurité : Application des bonnes pratiques de sécurité, incluant la limitation des accès de l'API à l'application frontend et la validation rigoureuse des entrées.
- Hébergement : La plateforme devra être en ligne.

L'identité graphique spécifique (police 'Graphik', logo dédié, favicon, palette de couleurs) devra être intégrée pour assurer une cohérence visuelle avec l'environnement numérique existant de la région (auvergnerhonealpes.fr).

1.5.Livrable Attendus :

- Rapport de conception et de développement au format PDF.
- Capture d'écran des maquettes inséré dans le rapport ainsi que le lien vers les maquettes.
- Présentation de la base de données (incluant le Modèle Conceptuel de Données - MCD, le Modèle Logique de Données - MLD sous forme de diagrammes, et les scripts SQL de création et d'alimentation).
- Description détaillée des mesures de sécurité mise en œuvre et de la veille.
- Dépôt GitHub contenant le code source complet (Frontend, backend, scripts SQL).
- Le site fonctionnel et accessible en ligne.
- Un fichier README.md à la racine du projet avec les prérequis et instructions d'utilisation.

2.Conception des Maquettes (UI/UX)

La conception des maquettes a été réalisée avec **Figma**, l'outil imposé par le cahier des charges. Cette étape a été fondamentale pour visualiser l'interface utilisateur, définir l'expérience utilisateur (UX) et garantir la conformité avec les exigences graphiques et d'accessibilité de la plateforme "Trouve ton artisan !".

2.1. Outil Utilisé : Figma

Figma a été choisi pour sa puissance en matière de conception collaborative d'interfaces utilisateur. Il a permis de créer des maquettes haute fidélité, de les itérer rapidement et de simuler l'expérience utilisateur sur différents appareils.

2.2. Approche Design : Mobile-First et Responsive

Conformément aux attentes du client et aux pratiques modernes de développement web, la conception a adopté une approche **"Mobile-First"**. Cela signifie que le design a été initié et optimisé pour les petits écrans (mobiles) en premier lieu, avant d'être progressivement adapté et enrichi pour les tablettes et les ordinateurs de bureau. Cette méthode assure une performance et une ergonomie optimales quel que soit le dispositif utilisé par l'utilisateur. Le design est entièrement **responsive**, garantissant que l'interface s'ajuste dynamiquement à toutes les tailles d'écran, sans compromettre la lisibilité ou la fonctionnalité.

2.3. Intégration de l'Identité Graphique et Accessibilité (WCAG 2.1)

L'identité visuelle de la Région Auvergne-Rhône-Alpes a été scrupuleusement respectée et intégrée :

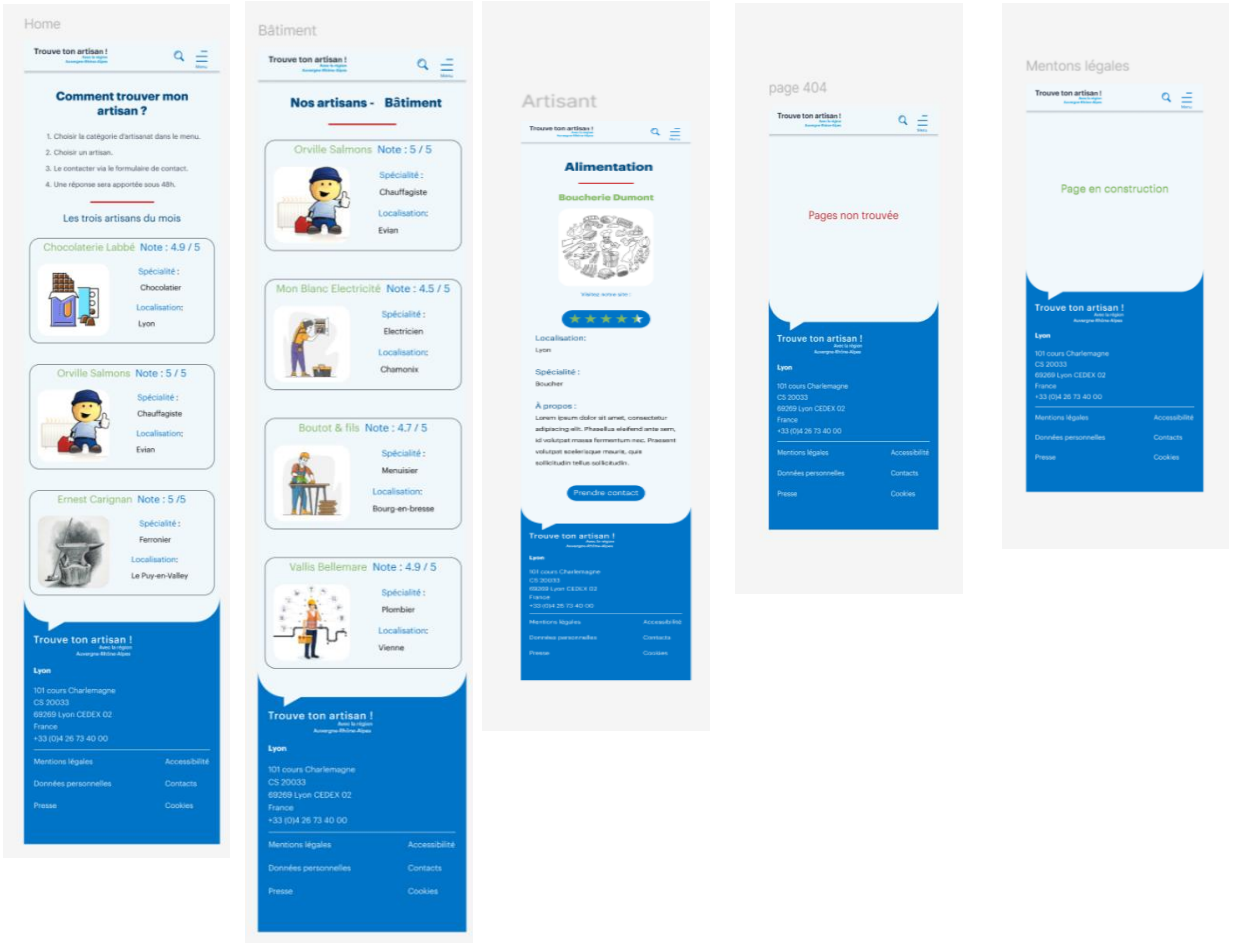
- **Typographie :** La police d'écriture "Graphik" a été utilisée pour l'ensemble des textes.
- **Logo et Favicon :** Le logo spécifique de la plateforme et le favicon fourni ont été implémentés dans l'en-tête et l'icône du site.
- **Palette de Couleurs :** La palette de couleurs définie par la région a été appliquée pour maintenir une cohérence visuelle avec son environnement numérique existant (auvergnerhonealpes.fr).

Un effort particulier a été porté sur l'**accessibilité**, en se conformant à la norme **WCAG 2.1** dès la phase de maquettage. Cela inclut des considérations sur les contrastes de couleurs, la taille des polices, la clarté des éléments de navigation, les états de focus des éléments interactifs et la structuration sémantique de l'information, afin d'assurer une utilisation aisée pour tous les publics, y compris les personnes en situation de handicap.

2.4. Maquettes Détaillées

Les maquettes complètes couvrent l'ensemble des pages et des états de l'application, pour les supports mobile, tablette et ordinateur. Elles détaillent la structure du header et du footer, la présentation des contenus dynamiques (artisans du mois, listes d'artisans, fiche détaillée), ainsi que les pages statiques (légales, 404).

Echantillonnage de pages pour la version mobile :



Lien vers le projet Figma complet :

https://www.figma.com/design/xOKsqnGnTt8A0tjQ3q9XyU/Devoir-Bilan-Morieux-Tony?node-id=0-1&t=mYo54lJSy2OAwu4I-1

3.Conception de la base de donnée :

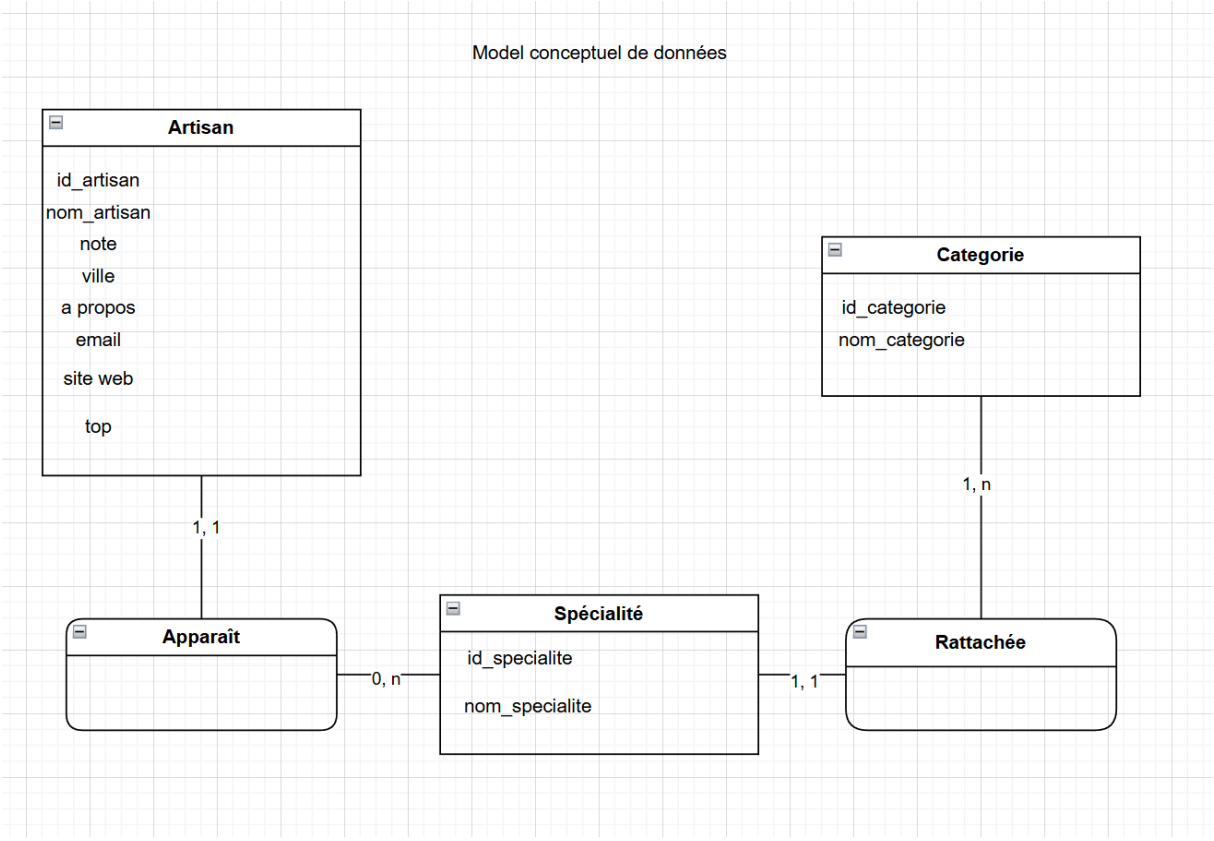
3.1. Modèle Conceptuel de Données (MCD)

Le Modèle Conceptuel de Données (MCD) est la première étape de la conception de la base de données. Il représente de manière abstraite et graphique les informations clés du système ("les entités"), leurs caractéristiques ("les attributs") et les liens logiques qui les unissent ("les relations"). Ce modèle est indépendant de toute contrainte technique et se concentre sur la compréhension des règles métier.

Les règles de gestion fondamentales qui ont guidé la conception de ce MCD sont les suivantes :

- Un artisan apparaît dans une seule spécialité.
- Une spécialité est rattachée à une et une seule catégorie.

Le diagramme ci-dessous illustre ce MCD pour la plateforme "Trouve ton artisan !" :



Description Détaillée des Éléments du MCD :

Le MCD se compose de trois entités principales, représentant les objets d'intérêt du système, et de deux relations qui les connectent :

1. **Entité : Artisan**

- Cette entité représente un professionnel de l'artisanat inscrit sur la plateforme.
- **Attributs :**
  - id_artisan : Identifiant unique de l'artisan (clé primaire conceptuelle).
  - nom_artisan : Nom de l'artisan ou de l'entreprise.
  - note : Note moyenne attribuée à l'artisan (sur 5).
  - ville : Localisation géographique de l'artisan.
  - a_propos : Description de l'artisan, de son savoir-faire et de ses services.
  - email : Adresse e-mail de contact de l'artisan.
  - site_web : Adresse du site web personnel de l'artisan (optionnel).
  - top : Indicateur booléen (vrai/faux) pour désigner les "artisans du mois".

2. **Entité : Spécialité**
   - Cette entité représente un domaine d'expertise spécifique dans l'artisanat (ex: "Chocolatier", "Menuisier").
   - **Attributs :**
     - id_specialite : Identifiant unique de la spécialité (clé primaire conceptuelle).
     - nom_specialite : Nom de la spécialité.
3. **Entité : Catégorie**
   - Cette entité regroupe plusieurs spécialités sous un thème plus large (ex: "Alimentation", "Bâtiment").
   - **Attributs :**
     - id_categorie : Identifiant unique de la catégorie (clé primaire conceptuelle).
     - nom_categorie : Nom de la catégorie.

Relations et leurs Cardinalités :

Les relations décrivent comment les entités interagissent, et les cardinalités ((min, max)) précisent le nombre de participations de chaque entité à cette relation.

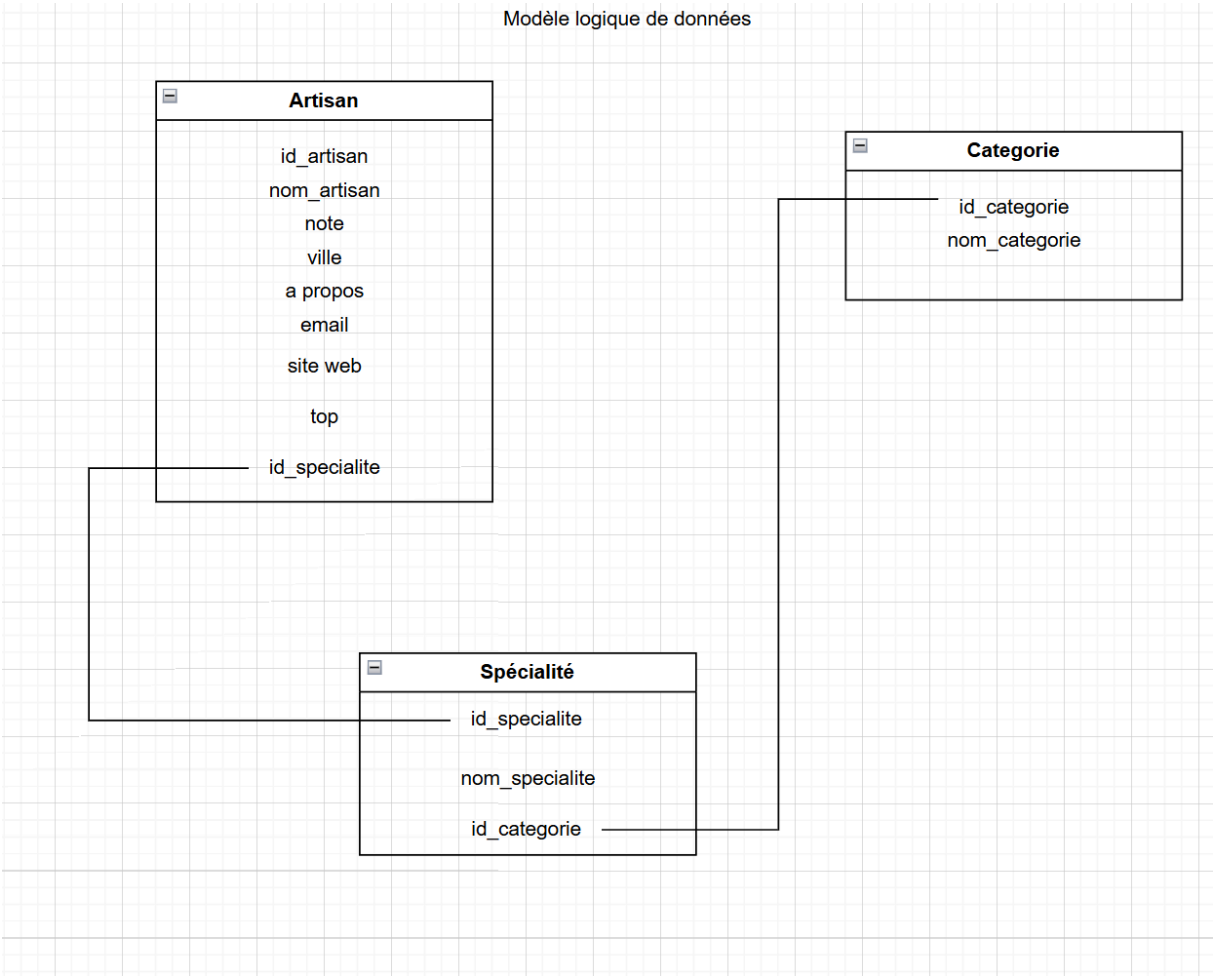
1. **Relation : Apparaît (entre Artisan et Spécialité)**
    - **Description :** Cette relation modélise le fait qu'un artisan est associé à une certaine spécialité.
    - **Cardinalités :**
        - De Artisan vers Apparaît : **(1,1)**
            - *Signification :* Un artisan doit obligatoirement exercer une et une seule spécialité sur la plateforme. Il ne peut pas exister sans spécialité, ni en avoir plusieurs.
        - De Spécialité vers Apparaît : **(0,N)**
            - *Signification :* Une spécialité peut être exercée par aucun, un, ou plusieurs artisans. Une nouvelle spécialité peut exister sans qu'aucun artisan ne lui soit encore rattaché.
2. **Relation : Rattachée (entre Spécialité et Catégorie)**
    - **Description :** Cette relation indique qu'une spécialité appartient à une catégorie plus générale.
    - **Cardinalités :**
        - De Spécialité vers Rattachée : **(1,1)**
            - *Signification :* Une spécialité doit obligatoirement être rattachée à une et une seule catégorie. Elle ne peut exister en dehors d'une catégorie.
        - De Catégorie vers Rattachée : **(1,N)**
            - *Signification :* Une catégorie doit regrouper au minimum une spécialité, et peut en regrouper plusieurs. Une catégorie ne peut pas exister si elle ne contient aucune spécialité.

3.2. Modèle Logique de Données (MLD)

Le Modèle Logique de Données (MLD) est la traduction du MCD en une structure de tables relationnelles. Il a été utilisé pour construire la base MySQL en environnement local (imposé par le devoir). En production, la base de données est gérée par **PostgreSQL Render**, et la création des tables est assurée automatiquement par Sequelize, sans scripts SQL manuels.

Les types SQL spécifiques à MySQL (AUTO_INCREMENT, TINYINT, InnoDB, FOREIGN_KEY_CHECKS) ne sont utilisés **que dans l'environnement local**. En production, PostgreSQL utilise ses propres types (SERIAL, BOOLEAN, INTEGER).

Le diagramme ci-dessous représente le MLD de la plateforme "Trouve ton artisan !" :



Description Détaillée des Éléments du MLD :

Le MLD se compose des tables suivantes, dérivées des entités du MCD, avec des attributs transformés en colonnes :

1. **Table : Artisan**
    - Cette table stocke toutes les informations relatives à un artisan.

- **Colonnes :**
  - id_artisan : Clé primaire (PK), de type entier (INT) et auto-incrémentée, garantissant l'identification unique de chaque artisan.
  - nom_artisan : Nom de l'artisan ou de l'entreprise (VARCHAR(50)).
  - note : Note moyenne attribuée, stockée en décimal (DECIMAL(5,1)) pour une précision à une décimale.
  - ville : Ville de localisation de l'artisan (VARCHAR(50)).
  - a_propos : Champ de texte libre pour une description détaillée (TEXT).
  - email : Adresse e-mail de contact (VARCHAR(50)).
  - site_web : URL du site web de l'artisan (VARCHAR(50), peut être NULL si l'artisan n'a pas de site).
  - top : Indicateur (TINYINT(1)) pour désigner les "artisans du mois" (0 pour non, 1 pour oui).
  - id_specialite : Clé étrangère (FK) vers la table Spécialité (id_specialite), établissant le lien entre un artisan et sa spécialité.

2. **Table : Spécialité**
   - Cette table liste les différentes spécialités artisanales.
   - **Colonnes :**
     - id_specialite : Clé primaire (PK), INT et auto-incrémentée, identifiant chaque spécialité de manière unique.
     - nom_specialite : Nom de la spécialité (VARCHAR(50)).
     - id_categorie : Clé étrangère (FK) vers la table Catégorie (id_categorie), indiquant à quelle catégorie chaque spécialité est rattachée.
3. **Table : Catégorie**
   - Cette table regroupe les grandes catégories d'artisanat.
   - **Colonnes :**
     - id_categorie : Clé primaire (PK), INT et auto-incrémentée, identifiant chaque catégorie de manière unique.
     - nom_categorie : Nom de la catégorie (VARCHAR(50)).

Relations et Intégrité Référentielle :

Les liens entre les tables sont garantis par l'implémentation des clés étrangères :

- **Relation Artisan - Spécialité :** La colonne id_specialite dans la table Artisan est une clé étrangère qui référence la clé primaire id_specialite de la table Spécialité.
    - *Signification :* Cette relation assure que chaque artisan est rattaché à une spécialité existante. Si une spécialité est supprimée, les artisans qui lui sont liés peuvent être gérés via une règle de suppression en cascade (définie dans le script SQL de création).
- **Relation Spécialité - Catégorie :** La colonne id_categorie dans la table Spécialité est une clé étrangère qui référence la clé primaire id_categorie de la table Catégorie.
    - *Signification :* Cette relation garantit que chaque spécialité appartient à une catégorie existante.

Ce MLD est la base technique sur laquelle les scripts SQL de création de la base de données et les modèles ORM (Sequelize) du backend ont été développés, assurant une correspondance exacte entre la conception logique et l'implémentation physique

3.3. Scripts SQL de création et d'alimentation

Cette sous-section présente les scripts SQL utilisés pour la mise en place de la base de données trouvetonartisanapi. Ces scripts sont fondamentaux pour recréer l'environnement de la base de données et la peupler avec les données initiales du jeu d'essais, assurant ainsi la conformité de l'implémentation physique avec le Modèle Logique de Données (MLD).

Les scripts create.sql et seed.sql ont été fournis pour répondre aux exigences pédagogiques du devoir, et permettent de créer et d’alimenter la base MySQL en environnement local.

En production, ces scripts ne sont pas utilisés : la base PostgreSQL Render est générée automatiquement par Sequelize lors du déploiement. Les mécanismes MySQL (InnoDB, AUTO_INCREMENT, FOREIGN_KEY_CHECKS) ne s’appliquent donc pas à l’environnement de production.

4. Développement de l'API (Backend)

Cette section décrit la conception et la mise en œuvre de l'API RESTful qui sert de colonne vertébrale à la plateforme "Trouve ton artisan !". Elle est responsable de la gestion des données des artisans, spécialités et catégories, en interagissant avec la base de données (MySQL en local, PostgreSQL en production) et en fournissant ces informations au frontend.

4.1. Architecture et Technologies

L'API backend a été développée en utilisant l'écosystème Node.js, conformément aux exigences du projet. Les technologies spécifiques employées sont :

- **Node.js :** Environnement d'exécution JavaScript côté serveur.
- **Express.js :** Framework web minimaliste et flexible pour Node.js, utilisé pour la construction de l'API RESTful.
- **Sequelize :** Object-Relational Mapper (ORM) puissant et moderne, utilisé pour interagir avec la base de données MySQL. Sequelize permet de manipuler les données en utilisant des objets JavaScript, offrant une couche d'abstraction par rapport au SQL brut et renforçant la sécurité (protection contre les injections SQL).
- **MySQL :**
    - En **local**, conformément au cahier des charges, l'API se connecte à une base **MySQL**.
    - En **production**, l'API est déployée sur **Render Web Service** et utilise une base **PostgreSQL Render**, Render ne proposant pas de service MySQL gratuit.
- **dotenv :** Bibliothèque pour charger les variables d'environnement depuis un fichier .env.
- **cors :** Middleware Express pour gérer les requêtes Cross-Origin Resource Sharing (CORS), essentiel pour la communication sécurisée entre le frontend et le backend.

Cette architecture permet une séparation claire des préoccupations entre le serveur de données (API) et l'interface utilisateur (frontend), favorisant la scalabilité, la maintenabilité et la flexibilité du développement.

4.2. Structure du Projet (MVC Adapté)

Le projet backend est organisé selon une architecture s'inspirant du modèle **MVC (Model-View-Controller)**, adapté au contexte d'une API RESTful où la "Vue" est déléguée au frontend. Cette structure favorise la modularité, la lisibilité et la maintenance du code.

La hiérarchie des dossiers et des fichiers est la suivante :

codeCode

/backend

├— config/ # Configuration de la base de données

| └— db.js # Fichier de connexion Sequelize

├— controllers/ # Logique métier et gestion des requêtes HTTP

| └— artisan.controller.js

| └— categorie.controller.js

| └— specialite.controller.js

| └— contact.controller.js

| └— specialite.controller.js

├— models/ # Définition des modèles de données et de leurs associations (Sequelize)

| └— artisan.model.js

| └— categorie.model.js

| └— index.js # Centralisation des modèles et des relations

| └— specialite.model.js

├— routes/ # Définition des endpoints API et de leur mappage aux contrôleurs

| └— api.js # Routes principales de l'API

├— .env # Variables d'environnement (non versionné)

├— seed.sql # Scripts SQL d'alimentation de la base de donnée

├— create.sql # Scripts SQL de création de la base de donnée

└—server.js # Point d'entrée de l'application Express

- **config/ :** Contient le fichier db.js qui initialise l'instance de Sequelize et établit la connexion à la base de données.

- **models/ :** Définit les schémas de données pour chaque entité (Artisan, Catégorie, Spécialité) via Sequelize, en mappant les objets JavaScript aux tables de la base de données, qu'il s'agisse de MySQL en local ou de PostgreSQL en production. Le fichier index.js centralise ces modèles et configure leurs associations (relations).
- **controllers/ :** Contient la logique métier de l'application. Chaque contrôleur (artisan.controller.js, categorie.controller.js, etc.) gère les requêtes HTTP spécifiques (récupération, filtrage, traitement des données) en interagissant avec les modèles.
- **routes/ :** Définit les points d'entrée de l'API (URLs) et les mappe aux fonctions des contrôleurs correspondants. api.js agrège toutes ces routes.
- **server.js :** Est le point d'entrée de l'application. Il configure le serveur Express, charge les middlewares globaux (CORS, JSON parser), monte les routes définies dans routes/api.js et gère le démarrage de l'API après une connexion réussie à la base de données.

Cette séparation des préoccupations rend le code plus organisé, plus facile à développer, à tester et à maintenir.

4.3. Définition des Modèles Sequelize et Associations

Les modèles Sequelize définissent les schémas de données et les relations. Ils permettent de mapper les objets JavaScript aux tables de la base de données, qu'il s'agisse de **MySQL en local** ou de **PostgreSQL en production**.

Le fichier models/index.js est crucial car il établit les **associations** entre ces modèles, reflétant les relations définies dans le Modèle Conceptuel de Données (MCD) et le Modèle Logique de Données (MLD).

- **Categorie.hasMany(Specialite) et Specialite.belongsTo(Categorie) :** Modélise qu'une catégorie peut avoir plusieurs spécialités, et qu'une spécialité appartient à une seule catégorie. (id_categorie est la clé étrangère dans Specialite).
- **Specialite.hasMany(Artisan) et Artisan.belongsTo(Specialite) :** Modélise qu'une spécialité peut avoir plusieurs artisans, et qu'un artisan appartient à une seule spécialité. (id_specialite est la clé étrangère dans Artisan).

Ces associations permettent à Sequelize d'effectuer des requêtes complexes avec include, récupérant ainsi les données liées (par exemple, un artisan avec sa spécialité et sa catégorie) en une seule opération, simplifiant le code des contrôleurs et optimisant les performances de la base de données.

4.4. Implémentation des Contrôleurs et des Routes

4.4.1. Contrôleurs (controllers/)

Chaque contrôleur contient les fonctions de rappel (handlers) qui traitent les requêtes HTTP. Ces fonctions utilisent les modèles Sequelize pour interagir avec la base de données et préparent les données à envoyer au client.

- **categorie.controller.js :** Gère la récupération de toutes les catégories (getAllCategories) et la recherche d'une catégorie par son nom (getCategoryByName).
- **specialite.controller.js :** Gère la récupération de toutes les spécialités (getAllSpecialites) et des spécialités par catégorie (getSpecialitesByCategory).
- **artisan.controller.js :** Gère la récupération de tous les artisans avec des options de filtrage par terme de recherche (search), par ID de spécialité (specialiteId) ou par ID de catégorie (categoryId) via getAllArtisans. Il inclut également des fonctions pour récupérer un artisan spécifique par son ID (getArtisanById) et les trois artisans mis en avant (getTopArtisans). L'opérateur Op.like de Sequelize est utilisé pour la recherche partielle et insensible à la casse sur les noms d'artisans.
- **contact.controller.js :** Gère la logique de traitement du formulaire de contact (submitContactForm), y compris la validation des entrées et la simulation de l'envoi d'un e-mail à l'artisan concerné.

Chaque fonction de contrôleur est encapsulée dans un bloc try...catch pour assurer une gestion robuste des erreurs, renvoyant des codes de statut HTTP appropriés (200 OK, 400 Bad Request, 404 Not Found, 500 Internal Server Error) et des messages génériques au client.

4.4.2. Routes (routes/api.js)

Le fichier routes/api.js centralise toutes les définitions des endpoints de l'API à l'aide d'express.Router(). Chaque route est mappée à une fonction spécifique d'un contrôleur.

- **Catégories :**
    - GET /api/categories
    - GET /api/categories/name?name={nom_categorie}
- **Artisans :**
    - GET /api/artisans?search={query}&specialiteId={id}&categoryId={id}
    - GET /api/artisans/top

    - GET /api/artisans/:id_artisan
- **Spécialités :**
    - GET /api/specialites
    - GET /api/categories/:id_categorie/specialites
- **Contact :**
    - POST /api/contact

Cette organisation des routes facilite la compréhension de l'API et permet une maintenance aisée.

4.5. Sécurité de l'API

Les mesures de sécurité suivantes ont été intégrées au niveau de l'API backend :

- **Protection Contre les Attaques Cross-Origin (CORS) :** Le middleware cors() d'Express est configuré pour n'autoriser que les requêtes provenant du domaine de l'application frontend, prévenant ainsi les requêtes malveillantes inter-sites.
- **Validation et Nettoyage des Entrées Utilisateur :** Bien que des vérifications de base soient effectuées dans les contrôleurs (contact.controller.js), l'utilisation de Sequelize offre une protection native contre les injections SQL.
- **Gestion des Erreurs Robuste :** Les messages d'erreur renvoyés au client sont génériques, évitant de révéler des informations sensibles sur l'infrastructure en cas de problème.
- **Variables d'Environnement pour les Configurations Sensibles :** Les configurations critiques (port de l'API, identifiants de base de données) sont stockées dans le fichier .env (non versionné sur GitHub) pour éviter toute fuite d'informations sensibles.
- **Utilisation de HTTPS (en Production) :** Il est prévu que l'API soit déployée derrière un serveur proxy (comme Nginx) qui gérera les connexions HTTPS en production pour chiffrer toutes les communications et protéger la confidentialité des données échangées.

5. Développement Frontend (Application Web)

Cette section décrit la conception et la mise en œuvre de l'application web côté client, "Trouve ton artisan !", développée pour offrir une interface utilisateur interactive, accessible et performante. L'application est conçue pour consommer les données fournies par l'API backend et les présenter de manière dynamique aux utilisateurs.

5.1. Architecture et Technologies

L'application frontend est une application monopage (SPA - Single Page Application) développée avec ReactJS, conformément aux exigences du projet. Les technologies et outils utilisés sont :

- **ReactJS :** Bibliothèque JavaScript pour la construction d'interfaces utilisateur réactives et modulaires. React permet de créer des composants réutilisables, améliorant la maintenabilité et la scalabilité du code.
- **React Router DOM :** Bibliothèque pour gérer la navigation au sein de l'application React, permettant des URLs propres et une gestion des vues basée sur le chemin.
- **Bootstrap :** Framework CSS populaire fournissant un système de grille flexible et des composants UI prédéfinis. Il a été utilisé comme base pour l'agencement responsive et les éléments d'interface.
- **Sass :** Préprocesseur CSS qui étend les fonctionnalités du CSS standard (variables, mixins, imbrication). Il a été utilisé pour organiser et personnaliser les styles de l'application, en intégrant la charte graphique.
- **Fetch API :** API native des navigateurs utilisée pour effectuer des requêtes HTTP (GET, POST) vers l'API backend et récupérer/envoyer des données.
- **Création d'application :** Le projet a été initialisé avec create-react-app

Cette architecture offre une expérience utilisateur fluide (chargement rapide des pages, pas de rechargement complet) et une séparation claire entre la logique métier côté serveur et la présentation côté client.

5.2. Structure des Composants et Pages

L'application React est organisée en une structure modulaire, favorisant la réutilisabilité des composants et une gestion claire des différentes vues.

- **Dossiers Principaux :**
    - /src : Contient le code source de l'application.

- /src/components : Regroupe les éléments d'interface réutilisables à travers l'application.
- /src/views : Contient les composants représentant des vues complètes ou des pages spécifiques.
- /src/styles : Organise les fichiers Sass pour le styling global et spécifique aux composants.
- /src/assets : Stocke les ressources statiques comme les images (logos, icônes) et les polices.
- /src/hooks : Contient le composant *dédié à la génération dynamique de balises titre et méta description pour le SEO.*

- **Composants (/src/components) :**
  - **Navigation.jsx :** L'en-tête de l'application, incluant le logo, le menu de navigation dynamique (catégories récupérées de l'API) et la barre de recherche.
  - **Footer.jsx:** Le pied de page, affichant les coordonnées et les liens vers les pages légales.
  - **MenuCard.jsx :** Composant réutilisable pour afficher un aperçu d'un artisan dans la section "Artisans du mois".
  - **Cards.jsx :** Composant réutilisable pour afficher un aperçu d'un artisan dans les listes de catégories ou de recherche.
  - **ArtisanCard.jsx :** Composant dédié à l'affichage détaillé des informations d'un artisan sur sa fiche, incluant son image, sa note, sa description, et le bouton de contact.
  - **ImageCard.jsx :** Composant responsable de l'affichage des images dans les Cards.
    - **Gestion des Images :** Les images utilisées sont des ressources statiques stockées localement dans le dossier `src/assets/images` du frontend (ex: `Atelier.jpg`, `Boucher.jpg`, `Logo.png`). Elles sont importées directement dans les composants React concernés (ex: `HomePage.jsx` pour l'image d'atelier, `ArtisanCard.jsx` pour l'image spécifique de l'artisan).

        - **Perspective d'Évolution :** L'intégration future d'une gestion dynamique des images (par exemple, via l'ajout d'une colonne `image_url` dans la table `tab_artisan` de la base de données et une liaison avec l'API) est une évolution envisagée pour permettre aux artisans de personnaliser leurs profils avec leurs propres visuels.

    - **Formulaire.jsx :** Le formulaire de contact détaillé, gérant la saisie des informations de l'utilisateur et l'envoi à l'API.
    - **StarRating.jsx :** Composant d'affichage des notes sous forme d'étoiles (avec support des dixièmes) de la note numérique.
    - **ValidationCard.jsx :** Carte affichant un message de confirmation après l'envoi réussi d'un formulaire.

- **Pages (/src/views) :**
    - **HomePage.jsx :** La page d'accueil de l'application, présentant les étapes de fonctionnement et les "Artisans du mois".
    - **CategoryPage.jsx :** Page dynamique affichant la liste des artisans pour une catégorie donnée, en récupérant les données de l'API.
    - **ArtisanPage.jsx :** Page dédiée à l'affichage détaillé d'un artisan spécifique, en récupérant ses informations complètes de l'API et en utilisant ArtisanCard.
    - **SearchResultsPage.jsx :** Page dédiée à l'affichage des résultats de la barre de recherche.
    - **LegalPage.jsx :** Page dynamique pour toutes les mentions légales (Mentions légales, Données personnelles, etc.), affichant un message "Page en construction".
    - **ErrorPage.jsx :** La page 404, affichée lorsque l'URL demandée n'existe pas.

Cette modularité garantit un code organisé, facile à naviguer, à tester et à faire évoluer.

5.3. Routage et Navigation

La navigation au sein de l'application est gérée par **React Router DOM**, configuré dans le fichier src/App.js. Ce système permet de mapper les URL de l'application à des composants de page spécifiques, offrant une expérience utilisateur fluide et sans rechargement complet de la page.

- **Routes Principales :**

- / : HomePage
- /batiment, /services, /fabrication, /alimentation : Des routes spécifiques pour chaque catégorie, renvoyant au composant CategoryPage avec le nom de la catégorie en prop.
- /artisans : Route pour les résultats de recherche, renvoyant au composant SearchResultsPage (avec le terme de recherche récupéré via useSearchParams).
- /artisan/:artisanId : Route dynamique pour afficher la fiche détaillée d'un artisan (ArtisanPage), où :artisanId est un paramètre d'URL.
- /mentions-legales, /donnees-personnelles, /accessibilite, /cookies, /contacts, /presse : Routes pour les pages légales, renvoyant au composant LegalPage avec le type de page en prop.
- * : ErrorPage (page 404), qui gère toutes les routes non définies et est toujours la dernière route déclarée.

Pour assurer une navigation cohérente et pleinement fonctionnelle côté client, l'utilisation des composants **<Link to> et <NavLink to>** de React Router DOM est essentielle. Ces composants interceptent les clics et effectuent la transition entre les routes sans recharger la page. L'utilisation de balises <a> HTML standards avec href provoquerait un rechargement complet de la page et des erreurs 404 sur GitHub Pages, car le serveur ne reconnaîtra pas les routes internes de l'application SPA. L'utilisation de useNavigate pour les redirections programmatiques (notamment après une recherche) complète ce système de navigation fluide.

5.4. Intégration des Appels API (Fetch)

L'application frontend communique avec l'API backend pour récupérer et envoyer des données dynamiquement. La **Fetch API** native des navigateurs est utilisée pour ces interactions HTTP.

- **Points d'Intégration Clés :**
  - **Navigation.js :** Utilise useEffect pour fetch la liste des catégories (GET /api/categories) au chargement et les affiche dans le menu. La soumission de la barre de recherche déclenche une redirection vers /artisans?search=....
  - **HomePage.js :** Utilise useEffect pour fetch les trois artisans du mois (GET /api/artisans/top) et les passe aux MenuCards.

- **CategoryPage.js :** Utilise useEffect pour d'abord fetch l'ID de la catégorie par son nom (GET /api/categories/name), puis fetch la liste des artisans de cette catégorie (GET /api/artisans?categoryId={id_categorie}). Les résultats sont passés aux Cards.
- **SearchResultsPage.js :** Utilise useEffect et useSearchParams pour récupérer le terme de recherche de l'URL, puis fetch les artisans correspondants (GET /api/artisans?search={query}).
- **ArtisanPage.js :** Utilise useEffect et useParams pour récupérer l'artisanId de l'URL, puis fetch les détails complets de l'artisan (GET /api/artisans/:artisanId). Les données sont ensuite passées à l'ArtisanCard.
- **Formulaire.js :** La fonction handleSubmit utilise fetch avec la méthode POST pour envoyer les données du formulaire de contact (nom, email, objet, message, artisanId) à l'API (POST /api/contact).

- **Gestion des États :** Pour chaque appel API, les états loading et error sont gérés via useState pour fournir un retour utilisateur (message de chargement, d'erreur) et useEffect pour déclencher les appels lors du montage ou de la mise à jour des dépendances.

L'URL de l'API est configurée via une variable d'environnement (REACT_APP_API_URL dans le fichier .env du frontend) pour une gestion flexible entre les environnements de développement et de production.

5.5. Accessibilité et Expérience Utilisateur (UX)

L'accessibilité et l'expérience utilisateur ont été des axes majeurs de développement, en adéquation avec les attentes du client et la norme WCAG 2.1.

- **Design Mobile-First :** L'approche de conception a permis d'optimiser l'interface pour les appareils mobiles, garantissant une navigation intuitive et fluide sur petits écrans.
- **Conformité WCAG 2.1 :**
  - **Attributs sémantiques :** Utilisation appropriée des balises HTML (<header>, <nav>, <main>, <footer>, <h1>-<h6>, <button>, <a>, <img>, <label>, <form>, <input>, <textarea>) pour structurer le contenu de manière logique.
  - **Texte Alternatif (Alt Text) :** Tous les éléments <img> incluent des attributs alt descriptifs et non redondants, essentiels pour les lecteurs d'écran.

  - **Labels explicites :** Les champs de formulaire sont correctement associés à des <label> via l'attribut htmlFor.
  - **Cibles Tactiles :** Les éléments interactifs (boutons, liens) ont des tailles de cibles tactiles suffisantes (largeurs de 52px-54px, hauteurs ajustées à un minimum de 44px) pour faciliter l'interaction sur mobile.
  - **Contraste des Couleurs et Taille de Police :** Les choix de couleurs et de tailles de texte (minimum 14px pour le texte de corps sur mobile) respectent les exigences de contraste et de lisibilité.
  - **Navigation au clavier :** La structure HTML et l'utilisation de NavLink/Link facilitent la navigation au clavier.
- **Feedback Utilisateur :** Les messages de chargement, d'erreur et de succès (ValidationCard, messages du formulaire) fournissent un retour immédiat à l'utilisateur.

5.6. Optimisation du Référencement (SEO)

Bien qu'une application React SPA nécessite des techniques SEO spécifiques (comme le Server-Side Rendering - SSR ou le Pre-rendering pour une optimisation avancée non couverte par ce projet), des mesures de base ont été intégrées pour améliorer la visibilité du site :

- **Structure HTML Sémantique :** L'utilisation de balises HTML5 appropriées (<main>, <h1>, etc.) aide les moteurs de recherche à comprendre la structure et la hiérarchie du contenu.
- **Titres et Méta-descriptions Dynamiques :**
- Les titres des pages (<title>) et les méta-descriptions (<meta name="description">) sont générés dynamiquement par page (par exemple, " Fiche de l'artisan Orville Salmons de la catégorie Bâtiment | Trouve ton artisan !", " Découvrez Orville Salmons, Chauffagiste à Evian.") pour fournir des informations pertinentes aux moteurs de recherche. (Cette implémentation est créer via un Hook).
- **URLs Claires :** Les URLs sont propres et descriptives (ex: /artisan/1, /alimentation), facilitant l'indexation.
-

6. Tests et Validation

Cette section détaille les différentes phases de test et de validation effectuées pour s'assurer que la plateforme "Trouve ton artisan !" fonctionne conformément aux spécifications, est accessible à tous les utilisateurs et offre des performances adéquates.

6.1. Tests Fonctionnels

Les tests fonctionnels ont eu pour objectif de vérifier que chaque fonctionnalité de l'application (frontend et API backend) se comporte comme attendu, en répondant aux besoins exprimés dans le cahier des charges. Ces tests ont été menés manuellement tout au long du cycle de développement.

- **Navigation :**
    - Vérification de la navigation entre toutes les pages via les liens du menu, le logo, et la saisie directe d'URLs.
    - Confirmation de la redirection vers la page 404 (ErrorPage) pour les URLs non valides.
    - Initialement, des erreurs 404 se produisaient sur les liens internes. Cela a été corrigé en remplaçant les balises <a> standard par les composants **<Link to> et <NavLink to>** de React Router DOM, assurant une navigation client-side sans rechargement de page.
- **Affichage Dynamique des Catégories :**
    - Vérification que le menu de navigation affiche correctement les catégories récupérées de l'API.
- **Page d'Accueil :**
    - Validation de la présence et de la mise en forme de la section "Comment trouver mon artisan ?".
    - Vérification que la section "Les trois artisans du mois" affiche dynamiquement les artisans marqués "Top" par l'API, avec leurs informations (nom, note, spécialité, localisation, image).
- **Listes d'Artisans (Pages Catégories et Recherche) :**
    - Pour chaque page de catégorie (ex: "Bâtiment", "Alimentation"), vérification que la liste des artisans correspondants est affichée.
    - Validation du fonctionnement de la barre de recherche :
        - Saisie d'un terme de recherche (ex: "Boucherie").

        - Redirection vers la page de résultats de recherche (/artisans?search=...).
        - Affichage des artisans dont le nom correspond au terme de recherche.
    - Confirmation que chaque "card" d'artisan est cliquable et redirige vers la fiche détaillée de l'artisan correspondant.

- **Fiche Détaillée d'un Artisan :**
    - Vérification de l'affichage de toutes les informations détaillées de l'artisan (nom, note, spécialité, localisation, description "À propos", e-mail, lien vers le site web, image).
    - Validation que la note est correctement représentée par le système d'étoiles avec remplissage par dixièmes et la note numérique affichée.
    - Confirmation de l'affichage du titre de la catégorie de l'artisan sur la page (ex: 'Catégorie : Alimentation').
- **Formulaire de Contact :**
    - Vérification que le bouton "Prendre contact" sur la fiche artisan affiche/masque correctement le formulaire.
    - Test de la soumission du formulaire avec des données valides :
        - Confirmation du message de succès (ValidationCard) dans le frontend.
        - Vérification de l'apparition du log détaillé du message dans la console du serveur backend.
    - Test de la soumission avec des champs vides pour vérifier les messages d'erreur du frontend.
- **Pages Légales :**
    - Vérification que chaque page légale affiche son titre correct et le message "Page en construction".
- **Cohérence Frontend/Backend :**
    - Des requêtes directes à l'API (via le navigateur ou Postman/Insomnia) ont été effectuées pour valider que les endpoints renvoyaient les données attendues. Cette validation a permis de confirmer la bonne communication avec le frontend et la conformité des structures de données.
    - Exemples de vérifications :

- Les requêtes `GET /api/categories` et `GET /api/artisans/top` ont confirmé que les données d'initialisation du menu et de la page d'accueil étaient correctement formatées.
- Les requêtes `GET /api/artisans/:id_artisan` ont validé que les fiches détaillées des artisans renvoyaient l'ensemble des informations attendues (y compris les spécialités et catégories associées) pour le composant `ArtisanCard`.
- La requête `POST /api/contact` a confirmé que l'API recevait et traitait correctement les données du formulaire, et qu'elle journalisait le message dans la console backend.

6.2. Tests d'Accessibilité

La conformité à la norme WCAG 2.1 a été une exigence clé du projet. Les tests d'accessibilité ont été menés à la fois par une revue de code et des vérifications manuelles.

- **Validation du Code HTML et CSS :**
  - **Mise en œuvre :** L'application a été soumise aux validateurs officiels du World Wide Web Consortium (W3C), spécifiquement le Validateur W3C pour HTML et le Validateur W3C pour CSS. Le code HTML résultant du rendu dynamique par React a été extrait du DOM du navigateur pour chaque page clé de l'application afin d'être analysé. De même, les fichiers CSS compilés ont été vérifiés, des visuels sont disponible dans le dossier assets(/src/assets/Vérification W3C/Css
  - **Problèmes rencontrés/corrigés :**
    - Des avertissements **ESLint** (tels que Invalid DOM property class nécessitant une correction en className) et Redundant alt attribute (exigeant une description plus concise des images) ont été systématiquement traités.
    - Des **erreurs de validation HTML** (telles que L'élément li n'est pas autorisé en tant qu'enfant de l'élément div) ont été corrigées en structurant correctement les listes avec des balises <ul> et <li>.
    - Des erreurs critiques de **"Carte d'identité en double"** (id="Card" ou id="MenuCard") ont été résolues en dynamisant les attributs id pour les rendre uniques (ex: id="card-123"), conformément à l'exigence HTML d'unicité des IDs.
    - Des erreurs de syntaxe CSS ont été rectifiées.

- **Contraste des Couleurs :**
    - Vérification visuelle et potentiellement avec des outils d'analyse de contraste pour s'assurer que les combinaisons de couleurs texte/fond respectent un ratio minimum de 4.5:1.
- **Taille des Polices :**
    - Confirmation que la taille minimale des polices (notamment 14px pour le texte de corps sur mobile) est respectée pour une bonne lisibilité.
- **Cibles Tactiles :**
    - Validation que les éléments interactifs (boutons, liens, icônes) ont une zone cliquable d'au moins 44px par 44px sur mobile (ex: ajustement de la hauteur du menu burger et des boutons de recherche à plus de 44px).
- **Navigation au Clavier :**
    - Test de la navigation complète du site uniquement au clavier (touches Tab, Shift+Tab, Entrée, Espace) pour s'assurer que tous les éléments interactifs sont atteignables et que l'ordre de tabulation est logique.
- **Texte Alternatif pour les Images :**
    - Vérification que toutes les images non décoratives possèdent un attribut alt descriptif.

6.3. Tests de Performance (simples)

Des tests de performance simples ont été réalisés pour évaluer la vitesse de chargement et la réactivité de l'application.

- **Temps de Chargement de la Page :**
    - Mesure du temps de chargement initial des pages (notamment la page d'accueil et les pages de catégories) dans un environnement de développement local (navigateur). En moyenne le LCP (Largest Contentful Paint) de la page d’accueil est de 0.36s
    - **Observations :** Le chargement est rapide grâce à l'architecture SPA (React) et à la rapidité de l'API locale.
- **Réactivité de l'Interface :**

- Interaction avec l'interface utilisateur (navigation, saisie dans la barre de recherche, ouverture du formulaire) pour évaluer la fluidité et l'absence de latence significative.
- **Observations :** L'application est réactive, les appels API sont traités rapidement et les mises à jour d'état de React sont fluides.

- **Optimisation des Requêtes API :**
  - Vérification dans l'onglet "Réseau" du navigateur que les requêtes API sont optimisées (par exemple, les requêtes include de Sequelize permettent de récupérer toutes les données liées en une seule fois, réduisant le nombre total de requêtes).

Ces tests confirment que la plateforme est fonctionnelle, accessible et offre une bonne expérience utilisateur, servant de base pour des tests plus approfondis en environnement de production.

7. Déploiement

Cette section détaille la stratégie de déploiement de la plateforme "Trouve ton artisan !". Conformément aux directives du devoir, le frontend de l'application sera déployé en utilisant GitHub Pages. Étant donné l'architecture fullstack du projet, le backend (API) devra être géré séparément, soit en étant lancé localement par l'évaluateur, soit en étant déployé sur un service distinct qui supporte les applications Node.js.

7.1. Plateformes d'Hébergement

7.1.1. Frontend (Application React) : Déploiement sur Render

- **Plateforme : Render Web Service**

  Render est une plateforme d’hébergement cloud permettant de déployer facilement des applications web complètes. Contrairement à GitHub Pages, Render prend en charge les applications React dynamiques, les API Node.js et les bases de données, ce qui en fait une solution adaptée au déploiement de la plateforme « Trouve ton artisan ! ».

Mise en œuvre technique :

- **Création d’un service Web Render**

Le dossier du frontend a été déployé en tant que *Web Service* sur Render. Render détecte automatiquement les projets React et configure l'environnement de build.

- **Configuration du build**

  Dans Render, la commande suivante est utilisée :

  Build Command : npm install && npm run build

  Start Command : serve -s build

  (Render propose automatiquement serve pour les applications React.)

- **Variables d'environnement**

  Une variable REACT_APP_API_URL a été ajoutée dans Render afin de pointer vers l'API backend également hébergée sur Render.

- **Routage React Router**

  Contrairement à GitHub Pages, Render gère correctement les routes côté client. Aucune configuration basename n'est nécessaire. Le routage fonctionne nativement avec BrowserRouter.

Processus de déploiement :

1. Dépôt du code sur GitHub
2. Connexion du dépôt à Render
3. Détection automatique du projet React
4. Build et déploiement automatique à chaque push sur la branche principale
5. Mise en ligne immédiate via une URL Render du type : https://app-trouve-ton-artisan.onrender.com/

Intérêt :

- Permet d'héberger une application React complète (SPA) sans limitations.
- Compatible avec un backend Node.js et une base PostgreSQL hébergés sur Render.
- Déploiement automatisé et mise à jour continue à chaque commit.
- Solution gratuite et adaptée à un environnement full-stack, contrairement à GitHub Pages qui ne supporte que les sites statiques.

7.1.2. Backend (API Node.js) : Stratégie de Déploiement

- **Limitation de GitHub Pages**

  GitHub Pages ne prend pas en charge les applications côté serveur (Node.js). L'API backend ne peut donc pas y être déployée.

- **Stratégie Adoptée pour le Devoir**

  Conformément aux exigences initiales du projet, l'API backend a été développée en Node.js/Express avec Sequelize et MySQL en environnement local. Pour l'évaluation, l'API peut être exécutée localement par l'évaluateur via :

  - **node server.js**

  L'application React peut alors communiquer avec l'API via :

  - http://localhost:3001

Déploiement réel (production)

Pour le déploiement en ligne, GitHub Pages n'étant pas compatible avec Node.js, l'API a été hébergée sur **Render Web Service**, une plateforme PaaS prenant en charge les applications serveur.

- L'API est déployée sur Render.
- La base de données est migrée vers **PostgreSQL Render**, Render ne proposant pas de service MySQL gratuit.
- Les variables d'environnement sont configurées directement dans Render.

Intérêt :

- Cette approche garantit un environnement serveur adapté, sécurisé, scalable et cohérent avec les bonnes pratiques de déploiement d'une API REST moderne.

7.1.3. Base de Données (MySQL → PostgreSQL Render)

Base de données en développement (MySQL imposé)

Pour répondre aux contraintes du devoir, la base de données a été conçue en **MySQL**, conformément au cahier des charges. Les scripts create.sql et seed.sql ont été fournis pour permettre la création et l'alimentation de la base en environnement local.

Base de données en production (PostgreSQL Render)

Pour le déploiement en ligne, la base a été migrée vers **PostgreSQL**, Render ne proposant pas de service MySQL gratuit. La structure logique (MCD/MLD) reste identique, mais la création des tables est désormais gérée automatiquement par Sequelize.

Accès

- En local : l'API se connecte à MySQL.
- En production : l'API se connecte à PostgreSQL via l'URL Render (DATABASE_URL).

7.2. Configuration des Environnements

La gestion des configurations entre développement local et production est essentielle pour garantir la flexibilité et la sécurité.

Variables d'environnement en développement (local)

- **Frontend (`.env` à la racine du frontend) :**

  Le frontend est configuré pour appeler l'API via l'URL Render : (`REACT_APP_API_URL=https://votre-backend.onrender.com`)

- **Backend (Render Web Service) :**

  Render utilise une base PostgreSQL. Les variables sont configurées dans le tableau de bord Render :

  DATABASE_URL=postgres://...

  DB_DIALECT=postgres

  PORT=10000 (port interne Render)

Intérêt :

Cette séparation garantit que :

- aucune information sensible n'est codée en dur,
- chaque environnement utilise ses propres paramètres,
- le passage du développement local (MySQL) à la production (PostgreSQL Render) est transparent.

8. Conclusion et Perspectives

8.1. Conclusion

La réalisation de la plateforme "Trouve ton artisan !" a été un projet complet et enrichissant, répondant aux besoins exprimés par la Région Auvergne-Rhône-Alpes de faciliter la mise en relation entre les citoyens et les artisans locaux. Ce projet a permis de concevoir et de

développer une solution web fullstack moderne, respectant les contraintes technologiques et les normes d'accessibilité.

En adoptant une approche "Mobile-First" dès la conception des maquettes avec Figma, l'application offre une expérience utilisateur intuitive et un design responsive, parfaitement adapté à une variété de dispositifs. L'intégration rigoureuse de la charte graphique de la région assure une cohérence visuelle avec son environnement numérique.

L'architecture backend, bâtie sur Node.js, Express et Sequelize avec une base de données MySQL, garantit une gestion robuste et sécurisée des données des artisans, catégories et spécialités. La structuration en MVC et l'implémentation de mesures de sécurité (CORS, validation basique, gestion des erreurs, utilisation d'ORM) contribuent à la fiabilité et à la protection de l'API.

Côté frontend, l'application ReactJS interagit dynamiquement avec l'API via Fetch API, affichant les informations en temps réel sur les pages d'accueil, de catégories, de recherche et les fiches détaillées des artisans. Les tests fonctionnels ont validé la conformité des fonctionnalités, tandis que les vérifications d'accessibilité ont permis d'atteindre un haut niveau de respect de la norme WCAG 2.1, rendant la plateforme utilisable par tous.

Ce projet démontre la capacité à développer une application web complexe de bout en bout, en intégrant les bonnes pratiques de développement, de sécurité et d'accessibilité.

8.2. Perspectives d'Évolution

La plateforme "Trouve ton artisan !" est conçue avec une architecture modulaire qui ouvre la voie à de nombreuses évolutions futures. Plusieurs axes d'amélioration et de nouvelles fonctionnalités peuvent être envisagés :

- **Administration des Données (CRUD) :**
    - Développement d'une application d'administration dédiée (mentionnée dans le cahier des charges) pour permettre à la Région ou aux artisans de gérer de manière autonome les données (création, lecture, mise à jour, suppression - CRUD) des artisans, spécialités et catégories.
    - Cela impliquerait l'ajout d'endpoints API sécurisés pour les opérations POST, PUT, DELETE.
- **Système d'Authentification / Inscription :**
    - Mise en place d'un module d'inscription pour les artisans, leur permettant de créer et de gérer leur propre profil.

    - Implémentation d'un système d'authentification (ex: JWT) pour sécuriser l'accès aux fonctionnalités d'administration ou de gestion de profil artisan.

- **Amélioration du Formulaire de Contact :**
    - Intégration d'un véritable service d'envoi d'e-mails (ex: Nodemailer avec un service SMTP) au lieu de la simulation actuelle, pour que les messages parviennent réellement aux artisans.
    - Mise en place d'une table de base de données pour archiver les messages envoyés via le formulaire.
- **Fonctionnalités Utilisateur Avancées :**
    - Ajout d'un système de notation et de commentaires pour les utilisateurs après avoir contacté un artisan.
    - Développement de fonctionnalités de géolocalisation pour trouver des artisans à proximité.
    - Mise en place de filtres de recherche plus avancés (par département, par disponibilité).
- **Optimisation de l'Accessibilité et du SEO :**
    - Pour le SEO, l'implémentation de techniques comme le Server-Side Rendering (SSR) ou le Pre-rendering pour ReactJS pourrait être envisagée pour une meilleure indexation par les moteurs de recherche.
    - Audit d'accessibilité plus approfondi et mise en œuvre des recommandations pour atteindre des niveaux de conformité WCAG plus élevés.
- **Contenu Riche :**
    - Ajout de galeries d'images ou de portfolios pour les artisans sur leur fiche détaillée.
    - Intégration de calendriers de disponibilité pour les artisans.

Ces perspectives démontrent le potentiel d'évolution de la plateforme, garantissant qu'elle pourra continuer à répondre aux besoins changeants de la Région Auvergne-Rhône-Alpes et de ses citoyens.

9. Annexes

Cette section regroupe les liens essentiels permettant d'accéder au code source complet du projet et à l'application fonctionnelle déployée en ligne.

9.1. Dépôt GitHub

L'intégralité du code source de la plateforme "Trouve ton artisan !", incluant les parties frontend (ReactJS), backend (Node.js/Express) et les scripts SQL de la base de données (MySQL), est hébergée sur GitHub. Le dépôt est organisé de manière à faciliter la compréhension de l'architecture et la réplication de l'environnement de développement.

Un fichier README.md détaillé est disponible à la racine du dépôt, fournissant les prérequis du projet ainsi que les instructions complètes pour l'installation, la configuration et le lancement des services frontend et backend.

Lien vers le Dépôt GitHub :
https://github.com/Toon-mo/app-trouve-ton-artisan.git

9.2. Site Web Accessible en Ligne

La plateforme "Trouve ton artisan !" est accessible via une URL publique. Cette version en ligne permet de visualiser et d'interagir avec l'application dans un environnement réel.

Lien vers le Site Web en Ligne :
https://app-trouve-ton-artisan.onrender.com