**Spécifications techniques**

[Menu Maker by Qwenta]

| **Version** | **Auteur** | **Date** | **Approbation** |
| --- | --- | --- | --- |
| 1.0 | mlhotellier | 05/06/2024 | Qwenta |

[I. Contexte et description du projet 2](#_6ysjogn08e4a)

[II. Architecture du site web 2](#_xytvfgbjem5l)

[III. Choix technologiques 3](#_1fob9te)

[IV. Liens avec le back-end 10](#_ads9wz5dcti1)

[V. Préconisations concernant le domaine et l’hébergement 11](#_qmg1ievpyvlf)

[VI. Accessibilité 12](#_q9q3ufd6bemo)

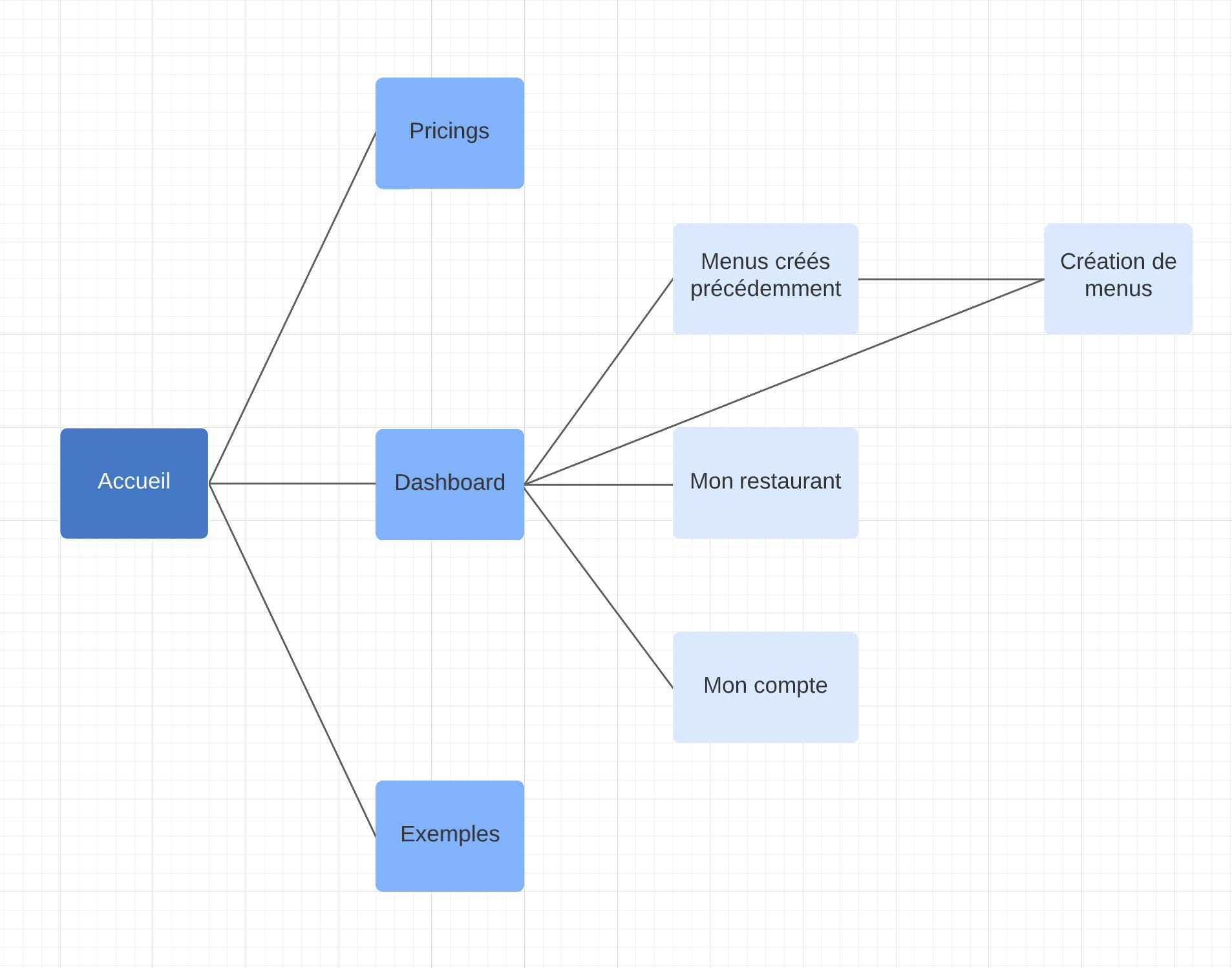
[VII. Recommandations en termes de sécurité 13](#_uiauozurxzcp)

[VIII. Maintenance du site et futures mises à jour 14](#_leyjdvug9ddz)

## Contexte et description du projet

***Qwenta*** souhaite faire développer un “Menu Maker” : un site permettant aux restaurateurs d’afficher et de mettre en page leurs menus facilement, en quelques clics.

## Architecture du site web



## Choix technologiques

* État des lieux des besoins fonctionnels et de leurs solutions techniques :

| **Besoin** | **Contraintes** | **Solution** | **Description de la solution** | **Justification (2 arguments)** |
| --- | --- | --- | --- | --- |
| Landing non-connecté | L'internaute doit comprendre l'utilité de l'application. Elle doit également être accessible**\***. | react-helmet | **react-helmet :** Cette librairie React permet de gérer facilement le balisage HTML pour améliorer l'accessibilité. | **react-helmet** : Compatible avec React.  Permet une meilleure gestion des métadonnées et des attributs d'accessibilité (ARIA). |
| Page login | La fenêtre de login doit s'ouvrir sous forme de modale.  L'utilisateur doit pouvoir entrer son adresse mail.  Un mail doit être envoyé pour vérifier l'authentification ou la création de compte.  Un lien "Besoin d'aide" doit permettre d'envoyer un e-mail aux équipes de support. | react-modal  react-router-dom  nodemailer  localStorage | **react-modal :** Permet de créer une fenêtre de login sous forme de modal.  **react-router-dom** : Gestion des routes, redirection & besoin d’aide.  **nodemailer :** Envoi d'emails pour l'authentification et la création de compte.  **localStorage :** Utilisé pour stocker et ajouter les informations d'authentification de l'utilisateur. | **react-modal :** Compatible avec React.  Simple et largement utilisé pour créer des modales.  **react-router-dom :** Gère efficacement la navigation.  Avec useEffect il permet de réagir aux changements.  **nodemailer :** Solution robuste pour l'envoi d'emails. Facile à configurer et utiliser avec Node.js.  **localStorage :** Simple à utiliser pour stocker les informations de session.  Ne nécessite pas de configuration serveur supplémentaire. |
| Catégorie de plat | L'ajout d'une catégorie doit pouvoir se faire directement sur l'écran de création de menu depuis une modale.  L'utilisateur doit pouvoir créer, valider, et sélectionner une catégorie créée précédemment. | react-modal  redux  MongoDB | **react-modal :** Crée une fenêtre modale pour l'ajout de catégories.  **redux :** Gère l'état global pour stocker et accéder aux différentes catégories.  **MongoDB :** Base de données NoSQL pour stocker et gérer les catégories de plats de manière efficace. | **react-modal :** Compatible avec React.  Simple et largement utilisé pour créer des modales.  **redux :** Gestion centralisée de l'état.  Facilite l'accès et la mise à jour des catégories.  **MongoDB :** Solution de base de données flexible et évolutive. Facile à intégrer et à utiliser avec React. |
| Création de plat | L'ajout d'un plat doit se faire depuis une modale spécifique.  L'utilisateur doit pouvoir sélectionner et modifier une catégorie, et entrer les informations d'un plat.  Chaque plat doit avoir une photo, un nom, un prix, et une description.  Il doit être possible de scroller dans la modale pour voir toutes les informations affichées. | react-modal  redux  MongoDB | **react-modal :** Crée une fenêtre modale pour l'ajout de catégories.  **redux :** Gère l'état global pour stocker et accéder aux catégories et plats.  **MongoDB :** Base de données NoSQL pour stocker et gérer les plats de manière efficace. | **react-modal :** Compatible avec React.  Simple et largement utilisé pour créer des modales.  **redux :** Gestion centralisée de l'état.  Facilite l'accès et la mise à jour des catégories et plats.  **MongoDB :** Solution de base de données flexible et évolutive. Facile à intégrer et à utiliser avec React. |
| Style de menu | Le restaurateur doit pouvoir visualiser le menu en temps réel pendant la personnalisation.  L'utilisateur doit pouvoir sélectionner une typographie et choisir une couleur de texte. | styled-components | **styled-components :** Permet de gérer le style des composants de manière dynamique. | Compatible avec React.  Permet une gestion dynamique et contextuelle du style dans la fenêtre de création du menu. |
| Exportation PDF | Le restaurateur doit pouvoir en un clic télécharger le fichier PDF correspondant à son menu. | jsPDF | **jsPDF** : Génère des fichiers PDF à partir de contenu web. | **jsPDF** : Solution populaire et largement utilisée pour générer des PDF à partir de contenu web.  Facile à intégrer avec React. |
| Commander l’impression d’un menu | L'encart "Imprimer un menu" doit être visible depuis la page d'accueil.  Le lien doit s'ouvrir dans un nouvel onglet. Le lien doit être fait vers le back-office de Qwenta. | react-router-dom  window.open() | **react-router-dom** : Gère la navigation et l'affichage de l'encart "Imprimer un menu".  **window.open()** : Ouvre le lien dans un nouvel onglet. | **react-router-dom :** Gestion efficace de la navigation et de l'affichage des composants.  Facilite l'intégration et la mise à jour des liens.  **window.open() :** Permet d'ouvrir des liens dans de nouveaux onglets de manière simple et efficace.  Facile à utiliser et à intégrer avec les composants React. |
| Menus précédents | Au clic sur "Mes menus", le restaurateur doit avoir accès aux menus créés précédemment.  La date de création doit s'afficher pour chaque menu précédent.  Il doit être possible de modifier et supprimer un menu précédent. Sur la même vue, le restaurateur doit pouvoir créer un nouveau menu. | react-router-dom  redux  MongoDB | **react-router-dom :** Gère la navigation vers la vue des menus précédents et la création d'un nouveau menu.  **redux :** Gère l'état global pour stocker les menus créés précédemment et faciliter leur modification et suppression.  **MongoDB :** Base de données NoSQL pour stocker et gérer les menus précédents de manière efficace. | **react-router-dom** : Gestion efficace de la navigation entre les vues.  Facilite l'intégration et la gestion des routes.  **redux** : Gestion centralisée de l'état des menus.  Facilite l'accès et la modification des menus précédents.  **MongoDB :** Solution de base de données flexible et évolutive. Facile à intégrer et à utiliser avec React. |
| Informations légales | Les éléments texte contenus dans la modale sont statiques et ne sont pas amenés à bouger pour le moment.  La mention "Tous droits réservés" doit figurer sur la page d'accueil et sur toutes les autres pages. | react-modal | **react-modal :** Crée une fenêtre modale pour consulter les mentions légales. | **react-modal :** Compatible avec React.  Simple et largement utilisé pour créer des modales. |
| Tarifs | La page des tarifs est créée côté Qwenta. L'URL suivra la logique *https://URL\_DE\_QWENTA/tarifs/menumaker*. | react-router-dom  window.open() | **react-router-dom** : Utilisé pour gérer la navigation vers la page des tarifs.  **window.open()** : Utilisé pour ouvrir un nouvel onglet avec l'URL des tarifs. | **react-router-dom** : Gestion efficace de la navigation entre les vues.  Facilite l'intégration et la mise à jour des routes.  **window.open()** : Permet d'ouvrir des liens dans de nouveaux onglets de manière simple et efficace.  Facile à utiliser et à intégrer avec les composants React. |
| Exportation Deliveroo | L'encart "Diffuser sur Deliveroo" doit s'afficher dans la catégorie "Exportez et diffusez". Au clic sur l'encart, le menu doit être exporté directement vers Deliveroo. | react-router-dom  window.open()  Deliveroo API  axios | **react-router-dom :** Utilisé pour gérer la navigation et l'affichage de l'encart. **window.open() :** Utilisé pour ouvrir Deliveroo dans un nouvel onglet. **Deliveroo API :** Utilisée pour exporter les menus directement vers Deliveroo. **axios :** Utilisé pour faire des requêtes HTTP vers l'API de Deliveroo. | **react-router-dom** : Gestion efficace de la navigation et des composants React.  Facilite l'intégration et la mise à jour des routes.  **window.open()** : Permet d'ouvrir des liens externes dans de nouveaux onglets de manière simple et efficace.  Facile à utiliser et à intégrer avec les composants React.  **Deliveroo API** : Intégration directe avec la plateforme Deliveroo.  Permet de gérer les menus de manière automatisée et efficace.  **axios** : Bibliothèque performante pour les requêtes HTTP.  Facilité d'utilisation avec React. |
| Partage sur Instagram | L'encart "Partager sur Instagram" doit s'afficher dans la catégorie "Exportez et diffusez".  Au clic, des images carrées du menu sont générées et l'utilisateur est redirigé vers son compte Instagram avec les photos prêtes à être partagées. | react-router-dom  window.open()  html-to-image  Instagram API | **react-router-dom :** Utilisé pour gérer la navigation et l'affichage de l'encart.  **window.open() :** Utilisé pour ouvrir Instagram dans un nouvel onglet.  **html-to-image :** Utilisé pour générer des images du menu au format carré.  **Instagram API :** Utilisée pour rediriger les utilisateurs vers Instagram avec les images générées. | **react-router-dom** : Gestion efficace de la navigation entre les vues.  Facilite l'intégration et la mise à jour des routes.  **window.open()** : Permet d'ouvrir des liens dans de nouveaux onglets de manière simple et efficace.  Facile à utiliser et à intégrer avec les composants React.  **html-to-image** : Convertit facilement des éléments HTML en images.  Pratique pour générer des images de menu prêtes à être partagées.  **Instagram API** : Permet l'intégration directe avec Instagram.  Simplifie le processus de partage pour l'utilisateur. |
| Déconnexion | Le restaurateur doit pouvoir se déconnecter depuis n'importe quelle page connectée. | react-router-dom  localStorage | **react-router-dom** : Utilisé pour rediriger l'utilisateur vers la page de connexion après déconnexion.  **localStorage :** Utilisé pour stocker et supprimer les informations d'authentification de l'utilisateur. | **react-router-dom :** Permet une redirection facile après déconnexion.  Cohérent avec notre choix de React pour le routage.  **localStorage :** Simple à utiliser pour stocker les informations de session.  Ne nécessite pas de configuration serveur supplémentaire. |
| Infos utilisateur | Le restaurateur doit pouvoir lier plusieurs adresses e-mail à son compte.  Il doit pouvoir modifier son adresse e-mail de base. | react-modal  axios  MongoDB | **react-modal :** Utilisé pour créer une interface modale permettant de modifier les informations utilisateur.  **Axios:** Utilisé pour effectuer des requêtes HTTP vers l'API pour récupérer et modifier les informations utilisateur.  **MongoDB :** Utilisé pour stocker et gérer les informations utilisateur. | **react-modal :** Permet de créer des interfaces utilisateur interactives et accessibles.  Facile à intégrer avec React pour une gestion dynamique des formulaires.  **axios** : Simplifie les requêtes HTTP avec une syntaxe claire et concise.  Gère facilement les promesses pour des opérations asynchrones.  **MongoDB :** Solution NoSQL flexible permettant de gérer des documents utilisateur avec des champs dynamiques.  Facile à intégrer et à utiliser avec React. |
| Dashboard | Doit regrouper plusieurs sections : création de menu, diffusion, impression, articles de blog.  Les articles de blog doivent afficher le titre, la photo de couverture et un lien. | react-router-dom  MongoDB  axios  RSS Parser | **react-router-dom** : Bibliothèque pour gérer les routes dans l'application React.  **MongoDB** : Stocke les informations des menus créés, diffusés, et imprimés.  **axios** : Utilisé pour les requêtes HTTP vers l'API pour récupérer les menus.  **RSS Parser :** Bibliothèque pour récupérer et parser les flux RSS du blog de Qwenta. | **react-router-dom :** Facilite la navigation entre les différentes pages et sections de l'application.  Permet de créer des routes dynamiques et bien structurées.  **MongoDB :** Gestion efficace des données des menus. Facile à intégrer et à utiliser avec React.  **axios** : Facilite la communication avec le backend pour récupérer les données en temps réel.  Supporte les opérations asynchrones et les promesses pour une manipulation fluide des données.  **RSS Parser :** Permet de récupérer et afficher les derniers articles de blog en temps réel. Facile à intégrer avec axios pour les requêtes HTTP. |
| Branding restaurateur | Doit permettre l'ajout, la modification et la suppression d'un logo et des couleurs de base du restaurant. | react-modal MongoDB  axios | **react-modal :** Utilisé pour afficher les formulaires d'ajout, modification et suppression de manière dynamique.  **MongoDB :** Stocke les informations de branding du restaurant (logo et couleurs).  **axios :** Utilisé pour les requêtes HTTP vers l'API pour gérer les données de branding. | **react-modal :** Permet une interface utilisateur interactive et accessible pour la gestion des informations de branding.  S'intègre parfaitement avec React.  **MongoDB :** Gestion efficace des informations de branding utilisateur et évolutivité.  Permet la mise à jour et la récupération en temps réel des informations de branding.  **axios** : Facilite la communication avec le backend.  Supporte les opérations asynchrones et les promesses pour une manipulation fluide des données. |

Ces solutions techniques sont adaptées aux **besoins spécifiques du projet** et sont justifiées par leur pertinence et leur efficacité dans la réalisation des fonctionnalités.

*\* Accessible: navigable au clavier et lisible par un lecteur d'écran.*

## IV. Liens avec le back-end

* **Langage pour le serveur : NodeJS (JavaScript)**

Node.js est un environnement d'exécution JavaScript côté serveur qui permet de construire des applications réseau évolutives. Il utilise un modèle d'E/S non-bloquant, ce qui le rend efficace et performant pour les applications nécessitant des requêtes simultanées élevées, comme dans notre cas avec MenuMaker.

* **API nécessaire : API RESTful avec Express.js**

Pour gérer les différentes interactions entre le front-end et le back-end, nous allons utiliser des API RESTful. Voici les détails des principales API nécessaires :

* **Deliveroo API** : Pour permettre aux restaurateurs de diffuser leurs menus directement sur Deliveroo. L'intégration sera réalisée via des requêtes HTTP sécurisées, utilisant les tokens d'authentification fournis par Deliveroo. L'API de Deliveroo permet de créer, modifier et supprimer des menus via des appels HTTP standard (GET, POST, PUT, DELETE).
* **Instagram API** : Pour faciliter le partage des menus sur Instagram. Cette API sera utilisée pour rediriger les utilisateurs vers Instagram avec les images des menus prêtes à être partagées. L'authentification OAuth 2.0 sera utilisée pour accéder aux comptes des utilisateurs de manière sécurisée.

Pour implémenter ces API, nous utiliserons **Express.js**, un framework web minimaliste et flexible pour Node.js, qui fournit un ensemble robuste de fonctionnalités pour les applications web et mobiles.

* **Base de données choisie : MongoDB**

Pour stocker et gérer les données de MenuMaker, nous avons choisi MongoDB, une base de données NoSQL orientée document. MongoDB offre plusieurs avantages pour notre projet :

* **Flexibilité et Scalabilité** : MongoDB permet de stocker des documents JSON, offrant une grande flexibilité pour les données de menus, qui peuvent varier en structure. De plus, MongoDB est facilement scalable, ce qui est essentiel pour une application qui peut croître rapidement en termes de nombre d'utilisateurs et de volume de données.
* **Intégration facile avec Node.js** : MongoDB s'intègre bien avec Node.js grâce à des bibliothèques comme Mongoose, qui simplifie les opérations de base de données et fournit une structure pour les modèles de données.

## V. Préconisations concernant le domaine et l’hébergement

* **Nom du domaine**

Le choix du nom de domaine est crucial pour assurer une bonne visibilité et une accessibilité optimale de l'application MenuMaker. Voici les options recommandées :

1. **menu-maker.qwenta.com** (préconisé)
2. **menumaker.qwenta.com**

Ces sous-domaines sont facilement mémorisables et directement associés à la marque Qwenta, ce qui aide à renforcer l'image de l'application tout en garantissant une navigation intuitive pour les utilisateurs.

* **Hébergement : AWS (Amazon Web Services)**

Pour héberger l'application MenuMaker, AWS (Amazon Web Services) est recommandé pour plusieurs raisons :

1. **Évolutivité et Flexibilité**
2. **Fiabilité et Performance**
3. **Sécurité**
4. **Coût-Efficacité**

* **Adresses e-mail**

Pour assurer une communication efficace avec les utilisateurs et gérer les aspects administratifs de l'application, deux adresses e-mail sont recommandées :

1. **support@qwenta.com** : Cette adresse sera utilisée pour le support client. Les utilisateurs pourront envoyer leurs questions, signaler des problèmes ou demander de l'aide via cette adresse, notamment sur la Page de connexion et le lien “Besoin d’aide ?“
2. **admin@qwenta.com** : Cette adresse sera utilisée pour les communications administratives et techniques. Elle peut servir à gérer les comptes administrateurs, les notifications internes et les communications importantes liées à la gestion de l'application.

## VI. Accessibilité

* **Compatibilité Navigateur**

Pour garantir que l'application MenuMaker offre une expérience utilisateur fluide, il est essentiel qu'elle soit compatible avec les navigateurs web les plus utilisés. La V1 de l'application doit donc être testée et optimisée pour les dernières versions des navigateurs suivants :

1. **Google Chrome**
2. **Mozilla Firefox**
3. **Safari**

Ces navigateurs couvrent la majorité des utilisateurs, assurant ainsi une large accessibilité et une expérience utilisateur cohérente.

* **Types d’appareils**

Pour la première version, l'application se concentrera uniquement sur les utilisateurs de bureau (**Desktop**). Cela permet de cibler les principaux utilisateurs potentiels, tels que les restaurateurs qui travaillent principalement sur des ordinateurs pour gérer leurs menus.

* **Accessibilité Web (WCAG)**
* **Contraste des couleurs** : Assurer un bon contraste pour les utilisateurs malvoyants.
* **Navigation au clavier** : Vérifier que toutes les fonctionnalités sont accessibles via le clavier.
* **Aria Labels** : Utiliser les attributs ARIA pour améliorer l'accessibilité des composants interactifs.

## VII. Recommandations en termes de sécurité

* **Accès aux comptes**

Pour sécuriser l'accès aux comptes utilisateur et gérer les sessions, nous utiliserons JSON Web Tokens (JWT). JWT offre une méthode sécurisée et standardisée pour la gestion des authentifications et des autorisations. Il est important de veiller à donner des durées de vie raisonnables pour les liens d'authentification et les tokens de connexion. Une durée trop courte, en revanche, risque de gonfler inutilement le volume de mails envoyés.

* **Plugins de sécurité**

Pour renforcer la sécurité de l'application, nous utiliserons divers plugins de sécurité. En particulier, Helmet sera utilisé avec Express.js pour sécuriser les en-têtes HTTP.

**Utilisation de Helmet :** Helmet est configuré pour activer une série de protections par défaut (Content Security Policy (CSP), X-Frame-Options, X-XSS-Protection, X-Content-Type-Options, Strict-Transport-Security (HSTS)).

* **Configuration de HTTPS** : Utilisation de certificats SSL/TLS pour toutes les communications pour assurer que les données en transit sont chiffrées. Forcer toutes les requêtes HTTP à rediriger vers HTTPS pour éviter les attaques de type man-in-the-middle (MITM).
* **Bonnes pratiques de sécurité supplémentaires**

1. **Sécurisation des API**
2. **Protection des données**
3. **Mise à jour et patching**
4. **Gestion des autorisations**

En suivant ces recommandations de sécurité, MenuMaker peut assurer une protection robuste contre les menaces courantes et offrir une expérience sécurisée à ses utilisateurs.

## VIII. Maintenance du site et futures mises à jour

* Maintenance :
  + Mise à jour régulière des dépendances et librairies utilisées.
  + Surveillance et correction des bugs.
  + Support technique et assistance en cas de problèmes.
  + Sauvegardes régulières des données.
* Evolution :
  + Ajouter de nouvelles fonctionnalités basées sur les retours des utilisateurs.
  + Ajouter des animations sur la photo de la bannière + sur les formes géométriques des sections. (cf. Landing-page non connectée)
  + Intégrer une page spécifique pour le tarif directement sur MenuMaker. (cf. Tarifs)
  + Créer possibilité de changer de moyen de paiement. (cf. Infos utilisateur)
  + Créer un blog interne à MenuMaker. (cf. Dashboard)

Ce document doit être lu, approuvé et signé par les différents gestionnaires de projets.