Spécifications techniques

MENU MAKER by Qwenta

Version	Auteur	Date	Approbation
1.0	Jean-Louis	Juin 2025	Soufiane

Présentation du projet

Qwenta souhaite offrir aux restaurateurs un outil en ligne leur permettant de créer, personnaliser et publier facilement leurs menus.
Application web responsive permettant la création de menus interactifs.
Public cible : Restaurateurs souhaitant digitaliser leurs menus sans compétences techniques.

Objectifs principaux :

- ➔ Faciliter la création de menus personnalisés.
- ➔ Permettre la mise à jour en temps réel des menus.
- ➔ Offrir une interface intuitive et accessible.

SOMMAIRE

I. Choix technologiques..... 1 à 5

II. Liens avec le back-end..... 6

III. Préconisations concernant le domaine et l'hébergement..... 7

IV. Accessibilité..... 8

V. Recommandations en termes de sécurité..... 9 à 11

VI. Maintenance du site et futures mises à jour..... 12 & 13

I. Choix technologiques

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

Besoin	Contraintes	Solution	Description de la solution	Justification (2 arguments)
Navigation entre les pages	Assurer une navigation intuitive, fluide et sécurisée	React Router v6	Bibliothèque de routage pour React permettant de gérer la navigation entre les différentes pages de l'application.	1) Intégration transparente avec React pour une navigation côté client efficace. 2) Permet la gestion de routes imbriquées, de redirections et de paramètres d'URL, facilitant la structuration de l'application.
Création et gestion de menus	L'ajout d'une catégorie doit pouvoir se faire directement sur l'écran de création de menu depuis une modale.	React-modal	Composant React pour créer des modales accessibles et personnalisables, facilitant l'ajout ou la modification de catégories sans quitter l'écran principal.	1) Compatible avec React, ce qui assure une intégration fluide. 2) Respecte les normes d'accessibilité WAI-ARIA, améliorant l'expérience utilisateur pour tous.

Gestion du formulaire	Différents formulaires : ● de connexion ● d'ajout de plat avec média ● Création de menu	React Hook Form	Bibliothèque légère pour la gestion des formulaires dans React, offrant une validation intégrée, une gestion optimisée de l'état et une performance améliorée grâce à une réduction des re-rendus inutiles.	1) Réduction significative du code nécessaire pour gérer les formulaires, simplifiant le développement. 2) Performances accrues grâce à une gestion efficace de l'état et une minimisation des re-rendus, améliorant l'expérience utilisateur.
Authentification	Confirmation de l'email. Vérification email. Connexion sécurisée. Création de compte(à ajouter)	Firebase Authentication	Service d'authentification complet offrant des fonctionnalités telles que la gestion des utilisateurs, la vérification des emails.	1) Solution clé en main réduisant le temps de développement et les erreurs potentielles. 2) Sécurité renforcée avec des fonctionnalités telles que la vérification des emails et la gestion des sessions.
Personnalisation	Choix de thèmes, couleurs, polices avant la publication.	Styled Components	Bibliothèque permettant d'écrire du CSS en JavaScript, facilitant la création de composants stylisés et là gestion dynamique des thèmes.	1) Permet une personnalisation dynamique des styles en fonction des préférences de l'utilisateur. 2) Favorise la réutilisabilité et la maintenance du code grâce à une structuration claire des styles.

Aperçu en temps réel	Visualisation instantanée des modifications Synchronisation des changements.	React Context API	Mécanisme de gestion de l'état global dans React, permettant de partager des données entre composants sans avoir à passer explicitement des props à chaque niveau.	1) Facilite la synchronisation des modifications en temps réel entre les composants. 2) Réduit la complexité du code en évitant le prop drilling, améliorant la maintenabilité.
Publication/Exportation /Diffusion	Génération d'un lien partageable ou intégration sur un site existant. Permettre au restaurateur de partager facilement son menu créé	React to Print react-konva	Bibliothèque React permettant d'imprimer des composants spécifiques ou de les exporter en PDF, facilitant le partage ou l'intégration des menus créés. Permets de concevoir menu en SVG. Redirection vers le compte du restaurateur.	1) Offre une solution simple pour l'exportation des menus sans nécessiter de développement backend supplémentaire. 2) Permet aux restaurateurs de partager facilement leurs menus via différents canaux.
Responsive design	Adaptation aux différents appareils (mobile, tablette, desktop)	Tailwind CSS	Framework CSS utilitaire permettant de concevoir rapidement des interfaces responsive et personnalisables, avec une grande flexibilité.	1) Accélère le développement grâce à une large gamme de classes utilitaires prédéfinies. 2) Facilite la création de designs adaptatifs sans écrire de CSS personnalisé, améliorant la cohérence et la maintenabilité.

II. Liens avec le back-end

- Stack : MERN (MongoDB, Express.js, React, Node.js)
- Langage pour le serveur : Node.js avec le framework Express.js
- API nécessaire : Oui, une API RESTful (CRUD) développée avec Express.js
- Base de données choisie : MongoDB Atlas (NoSQL)

Architecture et intégration

L'application est structurée selon une architecture MERN (MongoDB, Express.js, React, Node.js), permettant une séparation claire entre le frontend et le backend tout en facilitant leur communication.

- Express.js : Utilisé pour créer des routes API qui gèrent les opérations CRUD (Create, Read, Update, Delete) sur les menus, les catégories et les plats. Ces routes sont définies sous le préfixe /api.
- React Router : Gère la navigation côté client, permettant une expérience utilisateur fluide sans rechargement de page.
- Communication frontend-backend : Le frontend React interagit avec l'API Express via des requêtes HTTP (par exemple, avec fetch ou axios) pour récupérer ou envoyer des données.
- Déploiement : En production, Express sert les fichiers statiques générés par React (build) et redirige toutes les routes non-API vers index.html pour permettre la gestion des routes côté client par React Router.

Justification :

1. Stockage de structure JSON no SQL (plutôt que tables strictes) permet d'ajouter facilement des champs/attributs sans migration complexe.
2. Scalabilité horizontale native de MongoDB, adaptée pour héberger de potentielles volumétries et de pics d'usage.

III. Préconisations concernant le domaine et l'hébergement

- Nom du domaine : Exemple proposé : menu-maker.qwenta.com
- Hébergement front-end : Vercel ou Netlify pour le déploiement de l'application React en statique, avec CDN intégré ScaleupAllyHostAdvice.

- Hébergement back-end + BDD :

    - Heroku (dynos gratuits/premiums) pour Express + MongoDB via add-on Mongo Atlas.

    - Alternative : AWS Elastic Beanstalk pour Express + Amazon DocumentDB (compatible MongoDB).

- Adresses e-mail :

    - Utiliser un service d'e-mail transactionnel (SendGrid, Mailgun) pour l'envoi de confirmations d'inscription, réinitialisation de mot de passe.

    - Configurer un domaine d'envoi dédié (noreply@menu-maker-qwenta.com) avec DKIM/SPF pour garantir la délivrabilité.

IV. Accessibilité

- Compatibilité navigateur : Support des versions récentes de Chrome, Firefox, Safari, Edge : test via BrowserStack ou services similaires. W3CW3C.

- Normes WCAG 2.1 niveau AA avec WAVE et W3C :

    - Texte avec contraste ≥ 4,5 :1.

    - Navigation au clavier (tous les éléments interactifs focusables) W3C.

    - Utilisation d'attributs ARIA pour les composants dynamiques (modales, notifications, menus déroulants). reactcommunity.orgW3C.

- Types d'appareils : Desktop, tablette, mobile (iOS et Android) : tests avec Lighthouse et audits manuels pour vérifier la conformité. PublicInputEquidox.

V. Recommandations en termes de sécurité

HTTPS obligatoire :

- Tous les échanges chiffrés TLS 1.3+ via certificat Let's Encrypt géré automatiquement par hébergeur. Oligo Security.
- Le renouvellement automatique du certificat est vérifié (logs, alertes).
- Redirection HTTP → HTTPS forcée via serveur (NGINX, Apache) ou proxy.

OWASP Top 10 :

- Injection
  - Pour MongoDB, utilise toujours des requêtes paramétrées (ex: Mongoose ou MongoDB Drivers avec filtres).
  - Valide et filtre les données en entrée (sanitize).
  - Évite le passage direct de chaînes dynamiques dans les requêtes.
- Contrôle d'accès (RBAC)
  - Implémente une gestion fine des rôles et permissions (ex: Admin, User, Guest).

    - Vérifie systématiquement les droits en backend, ne te fie pas qu'au frontend.

- Sécurité des sessions (JWT) :
    - Stocke JWT dans cookie httpOnly, secure (HTTPS), avec SameSite=Strict ou Lax.
    - Prévois un mécanisme de renouvellement (refresh token).
    - Valide la signature et la validité à chaque requête.
- CSRF :
    - csurf est une bonne solution pour Express.js.
    - Pour les APIs REST, préfèrer un double-submit cookie ou tokens CSRF.
    - Protéger les endpoints modifiant les données.
- Gestion des environnements de développement :
    - Utilisation de variables d'environnement (.env) pour les clés API, chaînes de connexion DB. Reintech.
    - Ne jamais stocker les secrets dans le repo : config GitHub Secrets pour CI/CD.
    - Mise à jour des dépendances : Scanner régulier avec Snyk ou Dependabot pour corriger les vulnérabilités.

Validation par mail :

Pour assurer la vérification de l'identité des utilisateurs et renforcer la sécurité :

- À l'inscription, un compte est créé avec un statut non validé (isVerified = false).
- Un token unique, sécurisé et à durée limitée (ex : 24h) est généré côté serveur et stocké avec l'utilisateur.
- Un email contenant un lien de validation HTTPS incluant ce token est envoyé via un service SMTP sécurisé.
- Lorsque l'utilisateur clique sur le lien, le serveur vérifie la validité du token, active le compte (isVerified = true) et invalide le token.
- Des protections anti-abus sont mises en place : limitation des envois d'emails, tokens à usage unique.
- La validation par mail complète les autres mesures de sécurité en garantissant que seuls les utilisateurs légitimes peuvent activer leur compte.

Pare-feu applicatif (WAF)

- Cloudflare ou AWS WAF pour filtrage du trafic malveillant.
- Configure des règles spécifiques adaptées à ton application (ex: bloquer certains user-agents, IP blacklist).
- Surveille les logs et alertes WAF régulièrement.

VI. Maintenance du site et futures mises à jour

Contrat de maintenance :

- Corrections de bugs, mises à jour de sécurité, évolutions mineures définies dans un scope mensuel.
- SLA (Service Level Agreement) : temps de réponse initial inférieur à 24 h pour incidents critiques. Daily.dev.
- Structure tarifaire : forfait mensuel fixe + facturation horaire pour évolutions majeures hors scope. ailawyer.pro.

Mises à jour du framework :

- Veille et tests avant chaque mise à jour majeure de React, Express, MongoDB. Netguruweqtechnologies.com.

Sauvegardes régulières :

- Backups automatiques quotidiens de la BDD (MongoDB Atlas) avec rétention de 30 jours. OneNine.
- Sauvegarde des fichiers médias (S3 ou équivalent) avec versionnage activé. newtarget.com.

Monitoring & alerting :

Intégration Datadog ou Prometheus pour surveiller la performance serveur, latence des requêtes, erreurs 5xx.

fullstack.comarXiv.

Outils qui collectent en continu des données sur ton serveur et application :

- Performance CPU et mémoire
- Temps de réponse des requêtes (latence)
- Nombre et type d'erreurs serveur (notamment les erreurs 5xx qui indiquent un problème serveur)
- Alertes SMS/Slack en cas de seuils critiques dépassés (CPU, mémoire, latence). weqtechnologies.com.

Évolutivité :

- Utilisation d'une architecture conteneurisée Docker pour homogénéiser les environnements dev/production. arXiv.

  Docker permet d'encapsuler une application et ses dépendances dans un conteneur, garantissant qu'elle fonctionne de la même façon en développement, en test, et en production. Cela évite les problèmes liés aux différences d'environnement (versions de logiciels, configurations, etc.).

- Possibilité d'auto-scaler dynamiquement via Kubernetes (EKS) ou Heroku dynos horizontaux. arXiv. Cela garantit une bonne performance même en cas de pic d'utilisation.

  L'auto-scaling consiste à ajuster automatiquement le nombre de ressources (serveurs, conteneurs) selon la charge :

  - Kubernetes (avec EKS chez AWS) permet de gérer le scaling automatiquement en lançant plus ou moins de conteneurs selon le trafic.
  - Heroku propose des dynos (unités d'exécution) qu'on peut ajouter ou retirer à la volée pour répondre à la demande.