Webgencia

Spécifications techniques

[Nom du projet + nom du client]

Version	Auteur	Date	Approbation
1.0	Abderahmane	13/06/2025	Soufiane

I. 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 connectée	L'utilisateur doit pouvoir faire défiler et atteindre Bannière, « Personnalisez votre menu », « Étapes » sans temps de chargement visible.	React.js + reacter-router + Tailwind CSS	Composants réactifs rendus côté navigateur.	1) Interface très rapide. 2) Outil largement documenté et utilisé.
Modales accessibles en contexte	Modales accessibles : s'ouvrent en surimpression, prennent automatiquement le focus, se ferment via Échap et restent entièrement navigables au clavier.	react-modal	Librairie React légère qui affiche des fenêtres pop-up accessibles	1) Installation simple. 2) Compatible avec React
Page de login	Après saisie de l'e-mail, un lien de connexion sécurisé est envoyé ; aucun mot de passe n'est stocké ni demandé.	Firebase Auth – Google authentification (Magic Link)	J'utilise Google en tant que prestataire externe qui gére la base utilisateurs ainsi que l'envoi de mail et le processesus de connexion	1) Pas de gestion de PW et accès sécurisé aux données des menus et utilisateurs grâce à Firebase Auth. 2) Service Google fiable.
Export PDF en un clic	L'utilisateur génère un PDF fidèle à l'écran	jsPDF + html2canvas	html2canvas capture la section du DOM et jsPDF l'incorpore dans un	1) Fonctionne entièrement dans le navigateur. 2) PDF prêt à l'impression.

	actuel sans appel serveur.		document, récupere la police et permet le téléchargement immédiat.	
Personnalisation de l'apparence du menu (police et couleurs)	Respecter la charte graphique Figma desktop	React + Tailwind CSS	Classes Tailwind appliquées dynamiquement selon les choix du restaurateur.	1) Permet de modifier le design sans écrire beaucoup de CSS classique. 2) Garantit un rendu cohérent et léger, facile à maintenir.
Base de données souple pour menus	Les menus acceptent un nombre variable de catégories, plats, options, langues sans migration.	MongoDB Atlas + Mongoose	Stockage sous forme de documents structurés : nom, catégories, plats, style, date, ID utilisateur.	1) On peut stocker chaque menu complet, quel que soit le format. 2) MongoDB permet d'ajouter des champs sans migration. Sauvegardes et montée en charge gérées par Atlas.
Fournir au front les données nécessaires au dashboard (menus, articles, impression, diffusion)	Les données doivent être récupérées via une API sécurisée.	Node.js + Express	Le serveur Express fournit plusieurs routes REST que le front appelle pour afficher les infos dans le dashboard. Ces routes sont protégées par des jetons JWT. Chaque requête passe par un middleware qui vérifie l'identité de l'utilisateur connecté (via Firebase Auth).	1) On reste en JavaScript côté client et serveur, ce qui simplifie le développement. 2) Express est léger, rapide à mettre en place et parfait pour créer une API sur mesure.
Catégorie de plat	Interface modale, enregistrée dans une base de données, accessible depuis la création de menu.	React + Tailwind CSS + Node JS + Express + MongoDB	Modale React permet d'ajouter une catégorie, sauvegardée en base via une API Express.	1) Stack JS cohérente. 2) MongoDB permet une gestion flexible des catégories.

Création de plats	Chaque plat doit avoir un nom, une photo, un prix et une description ; interface modale	React + Multer + Sharp + MongoDB	Modale React pour saisir les infos, image envoyée au serveur, traitée puis enregistrée avec le plat.	1) Multer et Sharp s'intègrent facilement dans une API Express pour gérer l'upload et le traitement d'images côté serveur. 2) Permet une gestion complète du contenu du plat.
Générer automatiquement des images carrées du menu à partager sur Instagram	Les images doivent être au bon format (1080×1080), légères, optimisées et prêtes à l'export ou au téléchargement.	Multer + Sharp	Multer est un outil qui permet de recevoir facilement des fichiers envoyés depuis le front. Sharp est une librairie qui transforme les images (recadrage, compression, format). Ensemble, ils permettent de recevoir une image du menu, de la redimensionner en carré 1080×1080, puis de l'optimiser pour l'usage sur Instagram.	1) Multer et Sharp s'intègrent facilement dans une API Express pour gérer l'upload et le traitement d'images côté serveur. 2) Sharp permet d'obtenir un rendu propre et léger, parfait pour les réseaux sociaux.
Permettre au restaurateur de diffuser son menu sur Deliveroo	Le bouton doit rediriger vers Deliveroo, puis permettre plus tard d'envoyer le menu automatiquement	Lien externe vers Deliveroo + API Deliveroo (OAuth2)	Le bouton « Diffuser sur Deliveroo » ouvre la plateforme dans un nouvel onglet. Une intégration via leur API officielle (avec authentification OAuth2) est prévue ensuite pour envoyer directement le menu depuis l'application.	1) On démarre avec une simple redirection pour lancer rapidement la fonctionnalité. 2) L'API OAuth2 permettra ensuite une connexion sécurisée entre notre site et Deliveroo.

Permettre au restaurateur de personnaliser le style de son menu (polices, couleurs)	L'utilisateur doit pouvoir choisir une typographie et une couleur de texte en temps réel	Système de thèmes personnalisables (Tailwind CSS)	Depuis l'interface, le restaurateur peut sélectionner une police et une couleur à appliquer à son menu. Ces choix sont appliqués instantanément via des classes Tailwind générées dynamiquement.	1) Tailwind CSS permet de gérer rapidement les variations de styles sans surcharge. 2) Le rendu en direct améliore l'expérience utilisateur et réduit les erreurs.
Permettre au restaurateur d'imprimer son menu depuis Menu Maker	Un bouton « Imprimer un menu » doit ouvrir la page d'impression dans un nouvel onglet	Lien externe vers le back-office de Qwenta	En cliquant sur le bouton, l'utilisateur est redirigé vers une page d'impression déjà gérée par le back-office Qwenta. L'URL contient les informations du menu pour que l'impression soit prête sans autre action.	1) Cela permet de réutiliser l'outil déjà en place chez Qwenta. 2) Menu Maker n'a pas à gérer l'impression, ce qui simplifie le développement.
Voir les menus précédents	Permettre de consulter, modifier, supprimer, ou créer un nouveau menu depuis une vue dédiée	React + Express + MongoDB	Une vue affiche les menus stockés avec leurs dates de création, actions de modification ou suppression incluses	1) L'utilisateur retrouve ses anciens menus facilement. 2) Stockage souple avec MongoDB.

Informations légales	Les informations doivent être accessibles à tout moment, lisibles et à jour.	Lien en footer réalisé en React + React Router qui ouvre une page ou une modale (composant dédié).	Ajout d'un lien "Mentions légales" en bas de page (footer, composant React) ; au clic, ouverture d'une page dédiée (route gérée par React Router) ou d'une modale React affichant le contenu statique ou markdown.	1) Respect des obligations légales. 2) Rassure l'utilisateur et instaure la confiance.
Tarifs	Tarifs mis à jour, présentation claire et attractive, visible sans scroll excessif.	Bloc "Tarifs" développé en React + Tailwind CSS	Une section bien visible sur la page d'accueil (composant React, stylé avec Tailwind) présente les différentes offres et leurs tarifs, sous forme de tableau ou de cartes dynamiques (données statiques ou issues d'un JSON).	1) Transparence sur l'offre commerciale. 2) Aide l'utilisateur à choisir rapidement.
Déconnexion	Bouton accessible partout en étant connecté, suppression de la session sécurisée.	Bouton "Déconnexion" (React) qui appelle la méthode signOut de Firebase Auth.	Ajout d'un bouton visible sur toutes les pages privées (header/menu, composant React) ; au clic, appel à firebase.auth().signOut() puis redirection via React Router vers la landing.	1) Sécurité des données utilisateur. 2) Meilleure maîtrise de la session par l'utilisateur

Infos utilisateur	Accès sécurisé, édition simple, respect RGPD.	Section “Profil” (composant React) connectée à Firebase Auth.	Ajout d’une page ou modale “Mon profil” (React) permettant de voir/modifier nom, email, et éventuellement supprimer le compte via les méthodes Firebase.	1) Renforce la confiance. 2) Obligation légale (RGPD).
Dashboard	Accès uniquement après connexion, interface claire, données actualisées.	Vue Dashboard dédiée (React) qui consomme les données depuis l’API Node.js/Express sécurisée avec JWT.	Une page “Dashboard” centralise : menus créés, accès à la création/modification, liens vers profil, export PDF, etc.	1) Facilite la gestion de l’espace utilisateur. 2) Offre une expérience centralisée et professionnelle.
Branding restaurateur	Interface simple, respect du design global, support du format d’image.	Module de personnalisation intégré au dashboard (React, Multer/Sharp côté Node.js/Express pour l’upload).	Le restaurateur peut uploader son logo (upload géré côté serveur avec Multer et Sharp), choisir couleurs/images (stockées en base via l’API Express), et prévisualiser le rendu en temps réel sur ses pages (React + Tailwind).	1) Valorise l’image du restaurateur. 2) Renforce la cohérence visuelle et l’expérience client.

II. Liens avec le back-end

- Quel langage pour le serveur ?

Le projet utilise Node.js pour le développement côté serveur, associé à Express pour la gestion des routes.
Ce choix permet de rester en JavaScript sur l'ensemble du projet (front-end et back-end), ce qui facilite le développement, la maintenance et la cohérence technique avec React.

Node.js est également compatible avec de nombreuses bibliothèques utiles pour le traitement d'images, la génération de PDF, ou encore les échanges avec des services externes comme Deliveroo ou Instagram.

- A-t-on besoin d'une API ? Si oui laquelle ?

Oui, le projet repose sur une API REST développée avec Express.js afin de :

- Gérer les menus, catégories et plats,
- Exposer les données au dashboard,
- Sécuriser les accès grâce à Firebase Auth (authentification via e-mail),
- Interagir avec des services externes (ex. : API OAuth2 de Deliveroo).

Cette API est protégée par des jetons JWT et structurée pour répondre aux besoins futurs d'export, de partage, et d'évolution.

- Base de données choisie :

La base choisie est MongoDB Atlas, une base NoSQL orientée documents.

Elle permet de stocker chaque menu sous forme d'objet JSON, incluant les catégories, les plats, le style personnalisé, et l'utilisateur.

MongoDB offre une grande flexibilité (ajout de nouveaux champs sans migration), ainsi qu'une gestion automatisée des sauvegardes, de la scalabilité, et des performances via l'hébergement Atlas.

III. Préconisations concernant le domaine et l'hébergement

- **Nom de domaine : menumaker.qwenta.fr**
- **Nom de l'hébergement : Netlify, qui offrent un déploiement rapide, sécurisé et adapté aux projets React.**
- **Adresses e-mail : noreply@qwenta.fr**

IV. Accessibilité

- **Compatibilité navigateur : Chrome, FireFox, Safari**
- **Types d'appareils : Desktop Only**

V. Recommandations en termes de sécurité

- **Authentification sécurisée**
  L'authentification se fait via **Firebase Auth** avec envoi de liens de connexion temporaires par e-mail.
  Aucun mot de passe n'est stocké côté application. Les routes de l'API sont protégées par des **tokens JWT**, vérifiés à chaque appel serveur.

- **Gestion des accès**
  Les utilisateurs sont identifiés via leur e-mail, et les données sont **filtrées par ID utilisateur** pour garantir qu'un restaurateur ne puisse accéder qu'à ses propres menus.

- **Protection des données sensibles**
  Les clés API (Firebase, etc.) et les variables critiques sont stockées dans des **fichiers .env** côté serveur, non versionnés, et gérés via un système d'environnement sécurisé.

- **Validation et sécurité des entrées utilisateur**
  Les données envoyées aux APIs sont systématiquement **validées côté serveur** (via des schémas ou middlewares) pour éviter les attaques de type **injection ou XSS.**

- **Mises à jour régulières et audit des dépendances**
  L'ensemble des dépendances front et back doivent être **suivies via npm**, avec des alertes de sécurité activées. Les packages obsolètes sont mis à jour dès que possible pour réduire les vulnérabilités.

- **Respect du RGPD**
  Les adresses e-mail et les contenus des menus ne sont pas partagés à des tiers.
  Aucune donnée sensible (mot de passe, numéro de carte, etc.) n'est collectée.
  L'utilisateur peut à tout moment demander la suppression de ses données.

VI. Maintenance du site et futures mises à jour

- **Mise à jour des dépendances**
  Les bibliothèques et frameworks utilisés (React, Node.js, Firebase, Tailwind…) feront l'objet d'une veille régulière afin d'appliquer les correctifs de sécurité et de maintenir la compatibilité avec les dernières versions.

- **Surveillance et hébergement**
  Le front est déployé sur Vercel ou Netlify, et le back sur Render ou Railway. Ces services intègrent un système de déploiement continu, de monitoring basique, et d'alertes en cas de dysfonctionnement.

- **Sauvegarde automatique des données**
  MongoDB Atlas gère automatiquement les sauvegardes quotidiennes, la réplication des données et la restauration en cas d'incident.

- **Évolutions fonctionnelles**
  Le code est structuré de façon modulaire (composants, routes, middleware), ce qui facilite l'ajout futur de nouvelles fonctionnalités comme l'export vers d'autres plateformes ou la gestion multilingue des menus.

- **Documentation technique**
  Une documentation claire sera tenue à jour pour décrire l'installation, la configuration, et l'utilisation du projet (via un README versionné avec le code source).

- **Assistance**
  Une adresse e-mail dédiée (ex : support@qwenta.fr) est mise en place pour les utilisateurs qui rencontreraient des problèmes. L'équipe technique peut intervenir rapidement pour appliquer des correctifs.