Webgencia

Spécifications techniques

Menu Maker by Qwenta

Version	Auteur	Date	Approbation
1.0	Arnaud PIETTE	24 août 2026	À valider par le chef de projet Qwenta

I. Choix technologiques

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

Besoin	Contraintes	Solution	Description de la solution	Justification — 2 arguments
Développer l'interface de l'application	Interface dynamique conforme à Figma et compatible avec les navigateurs demandés	**React avec Vite**	React permet de construire l'application à partir de composants réutilisables. Vite initialise et compile le projet front-end.	1. React convient à un éditeur dont l'affichage évolue en temps réel. 2. Vite offre un environnement de développement rapide et une compilation optimisée.
Gérer la navigation	Passer de la landing page au dashboard, à l'éditeur et aux menus sans recharger l'application	**React Router**	Gestion des routes publiques, privées et dynamiques comme /menus/:id/edit.	1. Solution adaptée aux applications React monopages. 2. Permet de protéger les pages connectées et de gérer les URL inconnues.
Organiser les styles	Respecter la charte graphique et réutiliser les couleurs, espacements et typographies	**Sass et variables CSS**	Sass organise les fichiers de styles par composants. Les variables CSS appliquent dynamiquement les couleurs et polices du menu.	1. Sass facilite la maintenance des styles. 2. Les variables CSS permettent d'actualiser immédiatement l'aperçu personnalisé.
Afficher les modales	Les formulaires de connexion, catégories et plats doivent s'ouvrir sans changer de page et rester accessibles	**Composant Modal React avec React Portal**	Composant commun utilisant role="dialog", aria-modal, gestion du focus et fermeture avec Échap.	1. Il peut être adapté précisément à la maquette. 2. Un composant commun garantit un comportement accessible identique dans toute l'application.
Connecter l'utilisateur sans mot de passe	Envoyer un lien unique, temporaire et utilisable une seule fois	**Authentification par lien magique**	L'API génère un jeton sécurisé, l'envoie par e-mail puis crée une session dans un cookie httpOnly.	1. L'utilisateur n'a aucun mot de passe à mémoriser. 2. Le jeton temporaire et le cookie sécurisé limitent les risques de vol de session.
Créer et enregistrer les menus	Gérer les menus, catégories et plats et vérifier que les données appartiennent au bon utilisateur	**API REST avec Node.js et Express**	L'interface échange avec des routes protégées permettant de créer, lire, modifier et supprimer les données.	1. Node.js permet d'utiliser JavaScript côté front-end et back-end. 2. Express simplifie la création d'une API REST structurée et maintenable.
Enregistrer les données	Données liées : utilisateurs, restaurants, menus, catégories et plats	**PostgreSQL avec Prisma**	Stockage relationnel des données. Prisma définit le schéma, les relations et les validations.	1. PostgreSQL garantit des relations cohérentes. 2. Prisma simplifie les requêtes et les migrations.
Ajouter les photos des plats	Images facultatives, formats contrôlés et taille maximale de 2 Mo	**Multer et Cloudinary**	Multer contrôle les fichiers reçus par l'API. Cloudinary stocke les images et retourne leur URL.	1. Les fichiers sont vérifiés avant leur stockage. 2. Le stockage externe évite de surcharger le serveur de l'application.

Besoin	Contraintes	Solution	Description de la solution	Justification — 2 arguments
Afficher l'aperçu en temps réel	Chaque changement doit apparaître sans rechargement	**État React avec Context et useReducer**	L'état commun contient les catégories, plats et préférences graphiques utilisés par les formulaires et l'aperçu.	1. Une source de données commune évite les incohérences. 2. React actualise automatiquement les composants concernés.
Télécharger le menu	Produire un fichier fidèle à l'aperçu et imprimable au format A4	**Puppeteer**	Le serveur utilise Puppeteer et Chromium headless pour convertir le HTML/CSS du menu en PDF A4.	1. Le rendu reste fidèle à l'aperçu. 2. Le PDF est homogène quel que soit le navigateur.
Générer les visuels Instagram	Produire des images carrées lisibles et permettre une solution de secours	**Sharp et Web Share API**	Sharp génère les images PNG en 1080 × 1080 px. Le navigateur propose le partage natif lorsqu'il est disponible.	1. Sharp permet de générer et d'optimiser efficacement les images. 2. Le téléchargement reste disponible si Instagram ne permet pas un envoi automatique.
Vérifier la qualité	Tester les composants, l'API et les principaux parcours utilisateur	**Vitest, Supertest et Playwright**	Vitest teste le front-end, Supertest les routes Express et Playwright les parcours complets.	1. Les différents niveaux de tests détectent davantage de régressions. 2. Les parcours critiques peuvent être vérifiés automatiquement avant le déploiement.

II. Liens avec le back-end

Langage utilisé pour le serveur

Le serveur sera développé en **JavaScript avec Node.js et Express**.

Node.js permet d'utiliser le même langage côté client et côté serveur. Express fournit les outils nécessaires pour créer les routes, appliquer les contrôles d'authentification, recevoir les images et gérer les réponses de l'API.

Le code back-end sera organisé selon une architecture modulaire :

- routes ;
- contrôleurs ;
- modèles ;
- middlewares ;
- services ;
- gestion centralisée des erreurs.

Une **API REST** sera nécessaire pour assurer la communication entre l'interface React, le serveur et la base de données.

Les données seront principalement échangées au format JSON. Les photos seront envoyées avec le format multipart/form-data. Toutes les communications utiliseront le protocole HTTPS.

Les routes privées seront protégées par un middleware d'authentification. Le front-end transmettra automatiquement le cookie de session avec l'option credentials: "include".

Méthode et route	**Fonction**
POST /api/auth/magic-link	Envoyer un lien magique
GET /api/auth/verify	Vérifier le jeton de connexion
POST /api/auth/logout	Déconnecter l'utilisateur
GET /api/users/me	Récupérer les informations du compte
PATCH /api/users/me	Modifier les informations du compte
GET /api/menus	Récupérer les menus de l'utilisateur
POST /api/menus	Créer un menu
GET /api/menus/:id	Récupérer un menu
PATCH /api/menus/:id	Modifier un menu
DELETE /api/menus/:id	Supprimer un menu
POST /api/menus/:menuId/categories	Ajouter une catégorie
PATCH /api/categories/:id	Modifier une catégorie
DELETE /api/categories/:id	Supprimer une catégorie
POST /api/categories/:categoryId/dishes	Ajouter un plat
PATCH /api/dishes/:id	Modifier un plat

Méthode et route	Fonction
DELETE /api/dishes/:id	Supprimer un plat
GET /api/menus/:id/export/pdf	Télécharger le menu en PDF
GET /api/articles	Récupérer les derniers articles Qwenta

Gestion des réponses

L'API utilisera les codes HTTP adaptés :

- 200 : requête réussie ;
- 201 : ressource créée ;
- 400 : données incorrectes ;
- 401 : utilisateur non authentifié ;
- 403 : action non autorisée ;
- 404 : ressource inexistante ;
- 500 : erreur interne du serveur.

Les erreurs seront renvoyées dans un format JSON commun
afin que le front-end puisse afficher un message compréhensible.

Base de données choisie

La base de données choisie est PostgreSQL, une base relationnelle.
Elle sera utilisée avec Prisma pour définir le schéma, les relations, les validations et les migrations.

Les principaux modèles Prisma seront :

- User : compte et adresses e-mail ;
- Restaurant : nom, logo et couleurs ;
- Menu : nom, style et propriétaire ;
- Category : nom, position et menu associé ;
- Dish : nom, prix, description, photo et catégorie ;
- MagicLinkToken : jeton temporaire de connexion.

Relations entre les données

Un utilisateur peut être associé à un restaurant. Un restaurant peut posséder plusieurs menus. Chaque menu peut contenir plusieurs catégories et chaque catégorie peut contenir plusieurs plats.

Les relations seront gérées par clés primaires et étrangères. Les ressources posséderont également des propriétés createdAt et updatedAt.

PostgreSQL est adapté au projet pour deux raisons :

1. la structure flexible des documents correspond à la composition évolutive d'un menu ;
2. Prisma facilite les relations, les requêtes et les migrations du schéma.

III. Préconisations concernant le domaine et l'hébergement

Nom de domaine

L'application sera accessible depuis un sous-domaine appartenant à Qwenta :

- application : https://menu-maker.qwenta.com ;
- API : https://api.menu-maker.qwenta.com.

Ces noms restent à valider par Qwenta. Des enregistrements DNS seront ajoutés pour relier les sous-domaines aux services d'hébergement.

Toutes les communications seront protégées par HTTPS. Vercel prend en charge les domaines personnalisés et génère automatiquement leur certificat SSL

Hébergement recommandé

Élément	Service recommandé	Utilisation
Front-end React	**Vercel**	Hébergement et déploiement de l'interface
API Node.js/Express	**Render**	Exécution du serveur et génération des PDF
Base de données	**PostgreSQL**	Stockage relationnel des utilisateurs, restaurants et menus via Prisma
Images	**Cloudinary**	Stockage et optimisation des logos et photos
E-mails transactionnels	**Brevo**	Envoi des liens magiques et e-mails de vérification
Code source	**GitHub**	Versionnement et déclenchement des déploiements

Vercel sera utilisé pour le front-end en raison de son intégration avec GitHub,
de ses déploiements automatiques et de sa gestion des domaines personnalisés.

Render sera utilisé pour l'API Node.js/Express. Il prend en charge les domaines personnalisés, les certificats TLS et les déploiements sans interruption. Une offre adaptée à la production devra être utilisée, car l'offre gratuite peut mettre le serveur en veille après une période d'inactivité.

PostgreSQL utilisera un utilisateur dédié, une connexion chiffrée et des accès réseau limités.

Brevo sera utilisé pour envoyer les liens magiques sous forme d'e-mails transactionnels déclenchés par l'action de l'utilisateur.

Environnements

Deux environnements seront configurés :

- un environnement de préproduction pour effectuer les validations ;
- un environnement de production accessible aux utilisateurs.

Chaque environnement disposera de ses propres variables, adresses et accès à la base de données.
Les secrets et clés API ne seront jamais enregistrés dans le dépôt GitHub.

Adresse	Fonction
no-reply@qwenta.com	Envoi automatique des liens magiques
support@qwenta.com	Lien « Besoin d'aide » et assistance utilisateur
tech@qwenta.com	Réception des alertes techniques et de sécurité

Le domaine d'envoi devra être authentifié avec les enregistrements DNS SPF, DKIM et DMARC. L'adresse no-reply ne devra pas être utilisée pour recevoir les demandes d'assistance.

IV. Accessibilité

Référentiel retenu

L'objectif est de respecter les principales recommandations des **WCAG 2.2 de niveau AA**, notamment pour la navigation au clavier, les contrastes, les formulaires, les modales et la restitution par les lecteurs d'écran.

Les textes standards devront atteindre un contraste minimal de 4,5:1, contre 3:1 pour les textes de grande taille. Le contenu devra également rester utilisable lorsque le texte est agrandi jusqu'à 200 %.

Mesures d'accessibilité prévues

- utilisation des balises sémantiques header, nav, main, section et footer ;
- organisation logique des titres de h1 à h3 ;
- navigation complète au clavier ;
- indicateur de focus visible avec :focus-visible ;
- intitulés explicites pour les liens et les boutons ;
- association de chaque champ à un élément label ;

- erreurs de formulaire associées avec aria-describedby ;
- messages dynamiques annoncés avec aria-live ;
- textes alternatifs pour les images utiles ;
- attribut alt="" pour les images uniquement décoratives ;
- contrôle des contrastes avant validation d'une couleur personnalisée ;
- absence d'informations transmises uniquement par la couleur.
- maintien de la lisibilité lorsque le texte est agrandi.

Accessibilité des modales

Les modales de connexion, de catégories, de plats et de confirmation utiliseront
role="dialog", aria-modal="true" et un titre relié avec aria-labelledby.

À l'ouverture, le focus sera placé dans la modale. Les touches Tab et Maj + Tab resteront dans la modale.
La touche Échap permettra de la fermer et le focus reviendra ensuite sur le bouton ayant déclenché son ouverture.
Ces comportements correspondent au modèle de dialogue recommandé par le W3C.

Compatibilité avec les navigateurs

Navigateur	Version prise en charge
Google Chrome	Dernière version stable
Safari	Dernière version stable
Mozilla Firefox	Dernière version stable

Des tests manuels seront réalisés sur les trois navigateurs avant la mise en production.
Les fonctionnalités essentielles ne devront pas dépendre d'une API disponible dans un seul navigateur.

Pour la Web Share API, une solution de téléchargement manuel sera proposée si le partage natif n'est pas pris en charge.

Types d’appareils

Appareil	Prise en charge
Ordinateur de bureau	Oui
Ordinateur portable	Oui
Tablette	Pas de version spécifique pour la V1
Smartphone	Pas de version spécifique pour la V1

La première version est conçue pour ordinateur conformément aux spécifications fonctionnelles.
L’interface devra néanmoins supporter le zoom du navigateur sans perte des fonctionnalités principales.

Vérification

L’accessibilité sera contrôlée avec :

- Lighthouse ;
- axe-core ;
- la navigation manuelle au clavier ;
- VoiceOver sur macOS ;
- un contrôle manuel des contrastes ;
- des tests de zoom et de redimensionnement du texte.

Toute anomalie bloquant la navigation, la compréhension d’un formulaire
ou l’utilisation d’une modale devra être corrigée avant la mise en production.

V. Recommandations en termes de sécurité

Authentification par lien magique

Menu Maker ne stockera aucun mot de passe utilisateur.
La connexion utilisera un lien magique envoyé par e-mail.

Le jeton du lien magique devra :

- être généré avec une fonction cryptographique sécurisée ;
- être suffisamment long et imprévisible ;
- être enregistré sous forme hachée dans la base ;
- expirer après 15 minutes ;
- être utilisable une seule fois ;
- être supprimé après utilisation ou expiration.

La réponse du formulaire restera volontairement générique :
« Si cette adresse est valide, un lien de connexion a été envoyé ». Cela évitera d'indiquer publiquement si un compte existe.

Une limitation du nombre de demandes sera appliquée par adresse e-mail et par adresse IP.
OWASP recommande notamment de limiter les tentatives d'authentification afin de réduire les attaques automatisées.

Gestion des sessions

Après la validation du lien magique, le serveur créera une session sécurisée.
Son identifiant ou son jeton sera placé dans un cookie configuré avec :

- httpOnly pour empêcher sa lecture par JavaScript ;
- secure pour autoriser son envoi uniquement en HTTPS ;
- sameSite pour réduire les requêtes provenant d'autres sites ;
- une durée de validité limitée ;
- un domaine et un chemin aussi restrictifs que possible.

La session sera invalidée lors de la déconnexion et après sa date d'expiration.
Les attributs HttpOnly, Secure et SameSite sont recommandés par OWASP pour protéger les cookies de session.

Risque	**Mesure prévue**
Accès sans authentification	Middleware appliqué à toutes les routes privées
Accès au menu d'un autre utilisateur	Vérification du propriétaire pour chaque ressource
Requêtes provenant d'un autre site	Configuration CORS limitée au domaine Qwenta
Attaque CSRF	Cookie SameSite et jeton CSRF pour les actions sensibles

Risque	Mesure prévue
Trop grand nombre de requêtes	Limitation de débit avec express-rate-limit
Injection de données	Validation et nettoyage de toutes les entrées
Erreurs techniques exposées	Message générique côté client et détails uniquement dans les journaux
Attaque par en-têtes HTTP	Utilisation du middleware Helmet
Interception des données	API disponible uniquement en HTTPS

L'autorisation sera contrôlée directement sur chaque route de l'API. Une personne connaissant l'identifiant d'un menu ne pourra pas le consulter ou le modifier si elle n'en est pas propriétaire. OWASP recommande d'effectuer un contrôle d'accès sur chaque point d'entrée non public d'une API REST.

Validation des données

Toutes les données seront contrôlées côté serveur, même lorsqu'une validation existe déjà dans le navigateur.

Les règles incluront :

- vérification du format des adresses e-mail ;
- contrôle de la longueur des textes ;
- contrôle du format et de la valeur des prix ;
- validation des identifiants et des relations ;
- liste précise des propriétés pouvant être modifiées ;
- suppression ou encodage des contenus HTML non autorisés ;
- refus des requêtes dépassant la taille prévue.

Sécurité des images

Les fichiers seront limités aux formats réellement nécessaires, par exemple JPEG, PNG et WebP. Leur taille maximale sera fixée à 2 Mo.

L'extension, le type MIME et la signature du fichier seront contrôlés. Le fichier recevra un nom aléatoire et sera stocké sur Cloudinary, en dehors du serveur de l'application.

OWASP recommande d'utiliser une liste de formats autorisés, de contrôler le type et la signature du fichier et de remplacer le nom fourni par l'utilisateur.

Protection des secrets et des services

Les clés API, identifiants de base de données et secrets de session seront enregistrés dans les variables d'environnement de Vercel et Render. Ils ne devront jamais être présents dans le code source ou dans le dépôt GitHub.

Les accès à GitHub, Vercel, Render, PostgreSQL, Cloudinary et Brevo respecteront les règles suivantes :

- un compte individuel par membre de l'équipe ;
- activation de l'authentification à deux facteurs ;
- droits limités au besoin de chaque personne ;
- interdiction des comptes partagés ;
- stockage des accès dans un gestionnaire de mots de passe ;
- suppression immédiate des accès devenus inutiles ;
- renouvellement des clés exposées ou compromises.

Base de données, dépendances et journaux

PostgreSQL utilisera un compte propre à l'application avec des permissions limitées.
L'accès réseau sera restreint et des sauvegardes régulières seront activées. Les migrations seront gérées avec Prisma.

Les dépendances seront contrôlées avec npm audit et Dependabot. Les mises à jour de sécurité critiques seront appliquées après validation des tests.

Les journaux conserveront les erreurs, actions administratives et tentatives anormales, mais ils ne devront jamais enregistrer les liens magiques, cookies de session, clés API ou données personnelles inutiles.

VI. Maintenance du site et futures mises à jour

Types de maintenance

Le contrat devra couvrir trois catégories de maintenance :

- **maintenance corrective** : correction des erreurs et dysfonctionnements ;
- **maintenance préventive** : mises à jour de sécurité, surveillance et sauvegardes ;
- **maintenance évolutive** : ajout ou modification de fonctionnalités.

Les demandes seront enregistrées dans un outil de suivi avec leur priorité, leur responsable et leur statut.

Niveaux de priorité

Niveau	Exemple	Prise en charge recommandée
Critique	Application inaccessible, connexion impossible ou perte de données	Sous 4 heures ouvrées
Important	Création ou exportation d'un menu défaillante	Sous 1 jour ouvré
Normal	Erreur visuelle ou fonction secondaire	Sous 3 jours ouvrés
Évolution	Nouvelle fonctionnalité ou amélioration	Planification dans un prochain sprint

Maintenance préventive

Les opérations suivantes seront réalisées régulièrement :

- contrôle de la disponibilité du front-end et de l'API ;
- surveillance des journaux d'erreurs ;
- contrôle de l'espace de stockage ;
- vérification des sauvegardes PostgreSQL ;
- test périodique de restauration des données ;
- mise à jour des dépendances JavaScript ;
- vérification des alertes de sécurité GitHub ;
- renouvellement des clés et accès compromis ;
- contrôle des certificats HTTPS et des noms de domaine ;
- tests des liens magiques et des services externes ;
- vérification de la génération des PDF et des images.

Fréquence recommandée

Fréquence	Opérations
Continue	Surveillance de la disponibilité et des erreurs critiques
Hebdomadaire	Vérification des journaux et alertes de sécurité
Mensuelle	Mise à jour des dépendances et contrôle des sauvegardes
Trimestrielle	Audit des accès, performances et services externes
Annuelle	Révision de l'architecture et du plan de maintenance

Procédure de mise à jour

Le code sera versionné sur GitHub. Les évolutions seront développées sur une branche distincte, puis vérifiées par les tests automatisés.

Après validation, la modification sera déployée dans l'environnement de préproduction.
Une recette fonctionnelle sera réalisée avant la mise en production.

En cas d'anomalie après le déploiement, l'équipe devra pouvoir revenir à la version stable précédente.
Les modifications importantes seront accompagnées d'une note de version.

Sauvegardes et surveillance

La base PostgreSQL fera l'objet de sauvegardes régulières. Une politique de conservation
d'au moins 30 jours est recommandée pour la production.

Les erreurs du front-end et du serveur pourront être centralisées dans un service de suivi tel que Sentry.
Une alerte sera transmise à l'adresse tech@qwenta.com en cas d'erreur critique ou d'indisponibilité.

Évolutions futures envisagées

Les fonctionnalités suivantes ne font pas partie de la première version, mais pourront être ajoutées ultérieurement :

- adaptation complète aux smartphones et tablettes ;
- modification des informations de paiement ;
- blog hébergé directement dans Menu Maker ;
- animations avancées de la landing page ;
- intégration plus complète avec Deliveroo ;
- publication directe sur les réseaux sociaux ;
- nouveaux formats d'exportation ;
- modèles graphiques supplémentaires ;
- duplication et partage des menus ;
- historique des versions d'un menu ;
- statistiques d'utilisation et d'exportation ;
- gestion de plusieurs restaurants par utilisateur.

Documentation à maintenir

La documentation devra contenir :

- les instructions d'installation ;
- les variables d'environnement nécessaires ;
- la structure de l'application ;
- les routes de l'API ;
- les procédures de déploiement ;
- la procédure de restauration ;
- la liste des services externes ;
- les coordonnées des responsables techniques.