Webgencia

Spécifications techniques

Menu Maker by Qwenta

Version	Auteur	Date	Approbation
1.0	Maxime Tison	11/01/2025	John Qwenta

I. Choix technologiques

- Etat des lieux des besoins fonctionnels et de leurs solutions techniques :

Besoin	Contraintes	Solution	Description de la solution	Justifications (2 arguments)
Créer un menu avec ajout de plats et informations	L'utilisateur doit pouvoir ajouter des plats avec nom, description, prix et catégorie	React.js Création d'une interface avec formulaire dynamique ouvrant deux modales (une pour la catégorie, l'autre pour le reste des infos du/des plats)	Un formulaire en utilisant react.js pour le nom, la catégorie, la description et le prix du plat, avec possibilité d'ajouter plusieurs plats	1. Facilite l'ajout des plats sans avoir besoin de coder. 2. Permet une mise à jour rapide et intuitive
Personnaliser le menu (logo, police, couleur)	Doit être simple à utiliser, sans nécessiter de connaissances en design	Interface avec prévisualisation en direct	Un éditeur visuel permettant aux restaurateurs de télécharger un logo, choisir une police et des couleurs, ainsi que la mise en page	1. Permet un branding personnalisé. 2. Expérience utilisateur intuitive et interactive.
Exporter en PDF	Génération dynamique du menu en PDF	Utilisation d'une bibliothèque jsPDF	Un bouton d'export en PDF qui génère une version téléchargeable du menu	1. Compatible avec l'impression. 2. Facile à stocker et partager.
Partager sur Deliveroo et Instagram	Besoin d'une API compatible	Intégration des API Deliveroo et Instagram	Boutons permettant de publier directement le menu sur ces plateformes via API	1. Améliore la visibilité du restaurant. 2. Automatisation du partage réduit les efforts manuels.

Imprimer le menu	La commande d'impression doit se faire en un clic.	Intégration avec le service d'impression Qwenta	Qwenta offre déjà une solution d'impression	1. Lien direct vers Qwenta pour l'impression. 2. Qwenta est la solution principale de l'entreprise (à savoir l'impression)
Compatibilité navigateurs	Support uniquement de Chrome, Safari et Firefox (dernières versions)	Utilisation des standards modernes du web	Développement en respectant les spécifications HTML5/CSS3 et tests sur les 3 navigateurs	1. Garantie de bon affichage sur les navigateurs ciblés. 2. Réduction du temps de

II. Liens avec le back-end

1. Langage pour le serveur

- *Le back-end sera développé en Node.js, qui est un choix logique pour plusieurs raisons :*

  - **Compatibilité avec React** : Permet une architecture full Javascript (React en front, Node.js en back).

  - **Performance** : Node.js est performant pour les applications avec interactions en temps réel.

  - **Ecosystème riche** : Permet d'utiliser Express.js pour simplifier la gestion des routes API.

  - **Scalabilité** : Adapté pour une application qui doit gérer des menus personnalisés et des partages sur des plateformes externes

📌 Recommandation pour les développeurs :

- Utiliser Node.js avec Express.js pour la gestion des requêtes http et interactions API.

2. API nécessaires

- *L'application doit intégrer deux API principales :*

  - *I) API Instagram*

    - *Lien officiel :* API Instagram
    - *Utilité : Permet aux restaurateurs de partager automatiquement leur menu sur Instagram*
    - *Méthode :*
      - *Authentification via OAuth 2.0.*
      - *Utilisation de l'API Instagram Graph API pour publier du contenu*
      - *Nécessite une application validée par Facebook pour accéder aux permissions requises*

  - *II) API Deliveroo*

    - *Lien officiel :* API Deliveroo
    - *Utilité : permet aux restaurateurs de lier leurs menus directement à leur compte Deliveroo*

- *Méthode :*
  - *API REST permettant l'intégration de menus et de commandes*
  - *Authentification sécurisée (clé API et OAuth 2.0.)*
  - *Permet d'envoyer des menus et de récupérer des données sur les commandes.*

📌 Recommandation pour les développeurs :

➢ *Intégrer ces API via des routes spécifiques dans Express.js, en mettant en place un système d'authentification sécurisé.*

3. Base de données choisie

La base de données retenue est MongoDB, avec l'option d'hébergement MongoDB Atlas.

- Pourquoi MongoDB ?
  - Flexible : Parfait pour stocker des menus structurés avec des éléments dynamiques (plats, descriptions, prix, branding...)
  - NoSQL : Permet de gérer facilement des documents JSON sans rigidité de schéma
  - Scalable : MongoDB Atlas offre une infrastructure cloud optimisée

📌 Recommandations pour les développeurs :

➢ Stockage des menus sous forme de documents JSON, avec une structure adaptée.
➢ Hébergement sur MongoDB Atlas pour éviter la gestion d'un serveur de base de données

III. Préconisations concernant le domaine et l'hébergement

1. Nom de domaine

- menu-maker.qwenta.com (sous domaine de Qwenta, comme mentionné dans les spécifications fonctionnelles)

2. Hébergement

- Vercel : Hébergement cloud (scalable et performant)

Le projet est principalement en front-end avec export PDF en client-side, Vercel conviendra parfaitement.

3. Adresses e-mail

- support@qwenta.com → Pour le service client et les restaurateurs ayant des questions
- contact@qwenta.com → Adresse générique pour les demandes commerciales et informations
- noreply@qwenta.com → Pour les notifications automatiques envoyées aux utilisateurs

📌 Recommandation pour les développeurs :

Infomaniak Mail a été retenu pour assurer une gestion fiable et professionnelle

Documentation officielle : Infomaniak Mail

IV. Accessibilité

I. Compatibilité navigateur

- *Le site doit être compatible avec les dernières versions de Chrome, Safari et Firefox uniquement*

📌 Recommandation pour les développeurs :

- Utilisation des standards modernes du web : HTML5, CSS3, JavaScript ES6+.
- Ajout d'attributs ARIA pour l'accessibilité.
- Test sur différents navigateurs avec des outils comme BrowserStack ou les dev tools
- Pas d'optimisation pour les anciens navigateurs (ex : Internet Explorer, Edge Legacy).

II. Types d'appareils

- *Le site est prévu uniquement en version desktop, aucune version mobile n'est à prévoir.*

📌 Recommandation pour les développeurs :

- Utilisation d'une grille CSS (Flexbox/Grid) pour une mise en page adaptée aux écrans larges.

V. Recommandations en termes de sécurité

I. Sécurisation de la base de données (MongoDB, Mongoose, Schema)

- *La base de données MongoDB doit être sécurisée pour éviter toute fuite ou corruption des données.*

🔐 Validation des données avec Mongoose :

- Définir un schema strict pour éviter les entrées invalides.

🔐 Désactiver l'accès public à la base de données (🛑 MongoDB expose ses ports par défaut) :

- Utiliser MongoDB Atlas avec IP Whitelist pour limiter les accès
- Ne jamais stocker les identifiants en dur dans le code, utiliser des variables d'environnement (.env)

🔐 Chiffrement des données sensibles :

- Stocker les mots de passe avec Bcrypt avant enregistrement
- Ne pas stocker d'informations sensibles en clair dans la base

📌 Justifications :

- Réduit les risques d'injection et de corruption des données
- Empêche les accès non autorisés en limitant les connexions externes

II. Sécurisation des routes et authentification (Auth, Middleware, JWT)

- *Les utilisateurs doivent être identifiés avant d'accéder aux fonctionnalités sensibles (création, édition, suppression de menu)*

📌 Solution choisie

I. JWT (Json Web Token) pour l'authentification:

- Utiliser JWT pour générer un token sécurisé
- Stocker le token côté client dans un cookie sécurisé (httpOnly + Secure + SameSite=Strict)

II. Middleware pour protéger les routes API :

- Appliquer le Middleware sur les routes sensibles

III. Sécurisation des API

- *Les API utilisés (Instagram, Deliveroo) doivent être sécurisées pour éviter les fuites de données et abus.*

📌 Solution choisie

I. Limiter les accès avec OAuth 2.0 :

- Toujours utiliser les tokens OAuth fournis par les API pour s'authentifier
- Ne jamais stocker les clés API en dur, utiliser des variables d'environnement (.env)

II. Limiter le nombre de requêtes API (Rate Limiting) :

- Empêcher les abus en utilisant express-rate-limit

📌 Justification :

- OAuth garantit que seuls les utilisateurs autorisés peuvent accéder aux API externes
- Le rate-limiting protège l'API contre les attaques par déni de service (DDoS)

IV. Protection contre les attaques Web (XS, CSRF, SQL Injection)

- Le site doit être protégé contre les attaques courantes sur le web

📌 Solution choisie

I. Protéger contre le Cross-Site Scripting (XSS) :

- Désactiver le rendu HTML dans les entrées utilisateur

II. Protéger contre les attaques CSRF :

- Utiliser un token CSRF (avec Csurf dans Express)
- Vérifier l'origine des requêtes (Origin et Referer)

III. Prévenir les injections NoSQL :

- Pour MongoDB : interdire $where, eval() et filtrer les entrées

📌 Justification :

- Helmet et sanitizeHtml empêche l'injection de scripts malveillant (XSS)
- Les protections CSRF et contre les injections empêchent le vol de sessions et la modification illégitime des données

V. Stockage sécurisé des mots de passe et données sensibles

I. Les mots de passe et données sensibles doivent être stockés de manières sécurisées

📌 Solution choisie

- Stockage des mots de passe avec Bcrypt
- Ne jamais stocker les tokens en localStorage (risque XSS)
- Utilisation de dotenv pour sécuriser les clés API

📌 Justification du choix

- Empêche les fuites de mots de passe en cas de piratage
- Réduit le risque d'exposition accidentelle des clés sensibles

VI. Maintenance du site et futures mises à jour

I. Types de maintenance

➢ La maintenance se divise en trois catégories :

Type de maintenance	Objectif	Actions prévues
Maintenance corrective	Corriger les bugs et dysfonctionnements	- Surveillance des erreurs - Détection et correction des bugs (affichage, API, performances)
Maintenance évolutive	Ajouter de nouvelles fonctionnalités	- Ajout d'options de personnalisation des menus -Nouvelles intégrations API (ex : Uber Eats, Google business)
Maintenance préventive	Sécuriser et optimiser le site	- Mises à jour de sécurité - Optimisation des performances - Tests de compatibilité navigateurs

📌 Recommandation pour le client :

- Un contrat de maintenance mensuel ou annuel couvrant ces trois aspects est essentiel pour assurer la continuité du service

II. Modalités du contrat de maintenance

- Le contrat de maintenance devra définir les engagements et responsabilités des parties prenantes

1) Durée et engagement

- Contrat renouvelable sur 6 mois ou 1 an
- Un niveau de service (SLA) définissant les délais d'intervention
- Un audit technique régulier (tout les 3 ou 6 mois)

2) Support technique

- Un système de tickets pour signaler les incidents (via Notion ou un CRM)
- Délais de réponse selon la gravité du problème ;
  - Critique (ex : site indisponible) → intervention en moins de 4h
  - Moyen (ex : bug impactant une fonctionnalité) → intervention en moins de 24h
  - Mineur (ex : amélioration UX) → prise en compte dans le cycle de mise à jour

3) Sécurisation et mises à jour

- Mise à jour des dépendances (React, Node.js, MongoDB ...)
- Surveillance des failles de sécurité avec des outils comme Snyk
- Backup automatique des données (quotidien sur serveur sécurisé)

📌 Justification pour le client :

- Garanti un site toujours fonctionnel et sécurisé
- Offre une réactivité adaptée aux besoins des restaurateurs

III. Conceptualisation d'un plan des mises à jour futures

- L'évolution du site pourra être planifié en plusieurs phases

Phase	**Fonctionnalités prévues**
Phase 1 (T2 2025)	- Amélioration de l'interface utilisateur - Optimisation du temps de chargement
Phase 2 (T3 2025)	- Ajout d'un module de statistique pour les menus les plus consultés
Phase 3 (T4 2025)	- Intégration avec Uber Eats et Google My Business
Phase 4 (T1 2026)	- Version mobile du site (si besoin identifié)

📌 Recommandation pour le CDP et le client :

- Mettre en place une roadmap produit avec suivi des feedbacks clients pour prioriser les évolutions