Webgencia

Spécifications techniques

[Menu Maker + Qwenta]

Version	Auteur	Date	Approbation
1.0	Fanny, Webgencia Soufiane, Webgencia	01/12/2025	John, 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)
Compréhension du service via une landing non connectée	➢ Doit être claire, attractive, responsive, chargement rapide. ➢ Structure modulable permettant l'ajout futur d'animations sur la photo de la bannière et ajuster le design des sections.	Frontend : React (composants statiques) Sass (SCSS)	➢ Création d'une landing page composée de sections statiques : bannière, personnalisation du menu, étapes de création du menu. ➢ Mise en page en composants React et stylisée avec Sass (structure SCSS, variables, mixins, responsive).	1) React : UX moderne et simplifiée, standard dans de nombreuses applications professionnelles. 2) Sass permet un design organisé, extensible maintenable (variables, mixins, partials), compatible react, sans dépendances supplémentaires à installer.
Création d'un compte avec adresse mail.	➢ Lien envoyé automatiquement (magic link) ➢ Token à usage unique + expiration courte ➢ Validation obligatoire avant	Frontend : React React-modal Backend : API Node/Express Nodemailer Token(JWT) MongoDB	➢ L'utilisateur clic sur « se connecter », ➢ Ouverture d'une modale React. ➢ L'utilisateur y saisit son email pour créer son compte. (POST /auth/signup) ➢ Le backend génère un token temporaire (JWT) (usage unique, expiration courte).	1) Magic Link : expérience simple, sans mot de passe, sécurisée (token court + usage unique). 2) MongoDB : structure flexible idéale pour gérer plusieurs emails et futures extensions (paiement,

	création du compte		➢ Nodemailer envoie un Magic Link contenant ce token, à l'email saisi. ➢ L'utilisateur clique dessus → backend valide le compte (vérifie le token) → compte créé et utilisateur authentifié. ➢ MongoDB stocke le document utilisateur (emails, email principal, etc.).et éventuellement le token si usage unique.	préférences…).
Connexion sécurisée et accès aux menus	➢ Lien envoyé automatiquement par mail (magic link) à chaque connexion. ➢ Sécurité : vérification de l'email, lien expiration courte + token à usage unique	Frontend : React React-modal Backend : API Node/Express Middleware d'authentification (vérifie le JWT) Nodemailer JWT MongoDB	➢ Dans la modale, l'utilisateur saisit son email pour se connecter (POST /auth/login ➢ Le backend vérifie si l'email existe : ➢ Si Oui → génère un token temporaire (JWT). ➢ Nodemailer envoie un Magic Link contenant ce token, à l'email. ➢ L'utilisateur clique sur le lien ➢ Backend vérifie le token : →Token valide → utilisateur authentifié et connecté. ➢ MongoDB ne crée pas de nouvel utilisateur. ➢ Le token peut être supprimé après validation pour empêcher les réutilisations.	1) Conforme aux pratiques OWASP (token expiration courte, pas de mot de passe stocké côté serveur = pas de risque de fuite, système qui vérifie l'expiration et usage unique du token) 2) Architecture robuste : JWT + middleware → contrôle d'accès fiable pour chaque restaurateur.

Demande d'aide lors d'un souci d'inscription ou de connexion	➢ Solution simple et rapide à mettre en place, ➢ Pas de développement serveur supplémentaire ➢ Dans la modal d'inscription et de connexion	<u>Frontend :</u> Lien mailto: support@menu-maker.com, avec sujet pré-rempli.	Le lien ouvre directement l'application de messagerie de l'utilisateur avec l'adresse support et un sujet pré-rempli	1) Mise en œuvre rapide et légère (pas de back-end nécessaire) 2) Expérience utilisateur immédiate et intuitive
Création d'une catégorie de plat	➢ Uniquement restaurateur inscrit. ➢ Faire l'action dans une modale (non-rupture du flux). ➢ Validation simple et accessible. ➢ Catégorie immédiatement sélectionnable après sa création.	<u>Frontend</u> : React-modal <u>Backend</u> : JWT et vérification de rôle, Node/Express (API) MongoDB	➢ L'utilisateur clic sur l'onglet « catégorie ». ➢ Une modale s'ouvre pour saisir la catégorie. ➢ Validation frontend ➢ Envoie de la nouvelle catégorie saisie au backend (POST /categories) ➢ Le backend vérifie le token JWT de l'utilisateur, valide la donnée, puis ➢ Enregistre la catégorie dans MongoDB ➢ React actualise la liste pour qu'elle apparaisse immédiatement. ➢ A côté de la catégorie créée, le bouton « modifier » permet de modifier ou supprimer une catégorie (PUT /categories/:id, DELETE /categories/:id)	1) React-modal : modales performantes, accessibles ARIA avec un minimum de code. 2) MongoDB (noSQL) : → Flexible et extensible : si le projet évolue (ajout de champs, objets complexes). → l'échange React ↔ Node ↔ MongoDB "naturel" avec JSON (pas de lignes a transformer pour le frontend, comme en SQL).

Création d'un plat (catégorie, nom, prix, description, photo)	➢ Interface simple et fluide, réactive pour éviter les rechargements de page. ➢ Validations des champs ➢ Stockage illimité des plats ➢ Modifier une catégorie existante ➢ Modale scrollable.	Frontend : React React-modal Formik et Yup pour le formulaire Backend : Express MongoDB pour stocker les plats Moogose (schéma strictes et validation des données) JWT (auth)	➢ React gère l'interface et les modales (création de catégorie et ajout de plats). ➢ Le restaurateur crée une catégorie puis ajoute ses plats via des formulaires validés avec Formik/Yup. ➢ Les données sont envoyées à l'API Express, sécurisée par JWT, puis stockées dans MongoDB avec des contraintes Mongoose. ➢ Les modales sont scrollables grâce au styling Sass.	1) Formik/Yup améliore la qualité des formulaires (validation stricte). 2) API Express + MongoDB permet une logique métier claire, extensible, et scalable.
Upload d'images (logo, plats)	➢ Stockage fiable et rapide, ➢ Optimisation (qualité, taille), ➢ URLs accessibles depuis le frontend (publiques) ➢ Visibles immédiatement	Frontend : React Backend : Multer (upload) Sharp (otpimisation) Cloudinary (hébergement) JWT	➢ Le backend reçoit les images via Multer, ➢ Les optimisent via Sharp, ➢ Les envoie au service cloud choisi (Cloudinary) ➢ L'URL est renvoyée au frontend (liée au plat ou au logo) qui les affichent immédiatement. ➢ Token vérifié	1) Multer + sharp : Solution standard dans les projets Node.js modernes. 2) Cloudinary = stockage sécurisé des médias avec CDN intégré (accélère l'accès pour les utilisateurs)

Styliser le menu et choisir un template (1 colonne ou 2 colonnes) avec personnalisation du branding (couleurs, typographie, logo, mise en page).	➢ Prévisualisation en temps réel ➢ Ajout / modification / suppression du logo ➢ Choix des couleurs ➢ Sélection d'une typographie ➢ Sélection du Template (1 ou 2 colonnes) ➢ Sauvegarde persistante ➢ Routes sécurisées	Frontend : Redux Toolkit Backend : API Express MongoDB JWT	➢ Toute modification du style ou du Template du menu met à jour immédiatement l'aperçu du menu. ➢ Les préférences sont envoyées au backend via l'API et enregistrées dans MongoDB. ➢ L'état global Redux sert à générer le PDF, l'impression et les exports (Instagram, etc.) sans duplication. ➢ Les routes sont protégées par JWT pour garantir que seul le restaurateur authentifié peut modifier le branding.	1) Redux Toolkit : état centralisé, prévisible et réactif → toutes les fonctionnalités (aperçu, PDF, partage) utilisent le même état instantanément. 2) UX sécurisante : le restaurateur voit immédiatement ses modifications et peut partager/exporter son menu sans incohérence.
Export PDF	➢ Le PDF fidèle au design du restaurateur, ➢ Inclure tous les éléments du menu (plats, prix, images, logo)	Backend : Node + Puppeteer JWT	➢ Le restaurateur clic sur « Exportez en .pdf. ➢ La route est sécurisée par JWT. ➢ L'API génère le PDF depuis les données stockées (menus, images, logo). ➢ Grâce à Puppeteer, le rendu est fidèle. ➢ Un fichier téléchargeable est renvoyé au restaurateur.	1) Rendu fidèle : Puppeteer garantit un PDF identique au design grâce à un vrai moteur Chromium. 2) Contrôle et stabilité : Le backend Node maîtrise totalement la génération, assurant un PDF cohérent et fiable.

Impression du menu	➢ L'encart "Imprimer un menu" doit être visible depuis la page le Dashboard. ➢ L'impression doit passer par les outils officiels Qwenta.	Frontend : Lien externe vers le back-office Qwenta. (Accès protégé par authentification)	➢ L'utilisateur clique sur "Imprimer un menu" depuis le Dashboard. ➢ Il est automatiquement redirigé vers le back-office Qwenta, (accès sécurisé) où il peut imprimer son menu via les outils officiels de la plateforme. ➢ Le design imprimé respecte automatiquement les standards Qwenta.	1) Conformité totale avec les flux définis par Qwenta (impression centralisée). 2) Maintenance simplifiée : aucune gestion d'impression côté MenuMaker
Accès aux menus précédents et modifications	➢ Affichage clair des menus avec date. ➢ Possibilité de modifier ou supprimer, ➢ Création d'un nouveau menu depuis la même vue, ➢ Interface réactive et responsive (mises à jour instantanées)	Frontend : React (hooks) Backend : API Node/Express + MongoDB JWT	➢ Au clic sur "Mes menus", React affiche la liste des menus récupérés via l'API Express depuis MongoDB. ➢ Les Hooks React (useState, useEffect) permettent de gérer l'état local du menu et rafraîchir automatiquement l'affichage après chaque action utilisateur. ➢ Chaque menu peut être modifié, supprimé ou dupliqué pour créer un nouveau menu. ➢ La date est remise à jour. ➢ Requêtes sécurisées (JWT)	1) API REST + MongoDB permet une gestion centralisée et scalable des menus. 2) Hooks React permettent une interface fluide et réactive sans rechargement de page.

Affichage des « Mentions légales » et « Tous droits réservés » en modales.	➢ Accessibles depuis toutes les pages (via composant commun (footer). ➢ Contenu statique	Frontend : React + react-modal	➢ Les liens “Mentions légales” et « Tous droits réservés » ouvrent chacun une modale statique gérée par react-modal, accessibles sur pages connectées et non-connectées.	1) react-modal = accessibilité ARIA + intégration simple, gestion fiable des modales. 2) React : réutilisation facile du composant footer sur l’ensemble du site.
Accès aux tarifs de MenuMaker	➢ Disponible depuis la navigation principale ➢ Lien externe fourni par Qwenta ➢ Ouverture dans un nouvel onglet	Frontend : React (simple lien externe)	➢ L’onglet “Tarifs” dans la navbar ouvre en <a href target="_blank"> la page Qwenta (/tarifs/menumaker) dès qu’elle sera disponible. ➢ Solution extensible : possibilité d’intégrer ultérieurement les tarifs directement dans MenuMaker.	1) Aucun backend requis : simple redirection externe. 2) Comportement standard UX → ouverture en nouvel onglet pour ne pas quitter l’app.
Diffusion du menu sur Deliveroo	➢ Deliveroo n’a pas d’API publique pour les développeurs externes ➢ La publication automatique n’est pas possible ➢ L’import doit être manuel	Frontend : Lien externe vers Deliveroo Backend : Génération d’une image du menu (backend Puppeteer)	➢ Dans la catégorie « Exporter et diffuser », l’utilisateur clique sur « Diffuser sur Deliveroo » ➢ Il est redirigé vers le site Deliveroo ➢ Le backend génère une image du menu pour l’aider à saisir les informations dans Deliveroo (route sécurisée) ➢ L’import reste manuel.	1) C’est la seule méthode conforme à la réalité technique de Deliveroo 2) L’image exportée sert de support universel pour tous les restaurateurs.

Partager le menu sur Instagram	➢ Nécessite un compte Instagram Business ; ➢ Publication possible uniquement via l'API Graph ; ➢ Format carré obligatoire	Backend : Génération d'images carrées (via Puppeteer) Publication via Instagram Graph API	➢ Dans la catégorie « Exporter et diffuser », le restaurateur clic sur « Partager sur Instagram », ➢ Le backend génère automatiquement des visuels carrés du menu (via Puppeteer) (->route sécurisée) ➢ Ces visuels peuvent ensuite être publiés via l'Instagram Graph API : -> soit automatiquement si les permissions et tokens Business sont valides, -> soit via une validation dans l'application Instagram.	1) Instagram Graph API = seule API officielle permettant une publication. 2) La génération d'images carrées respecte les contraintes techniques de la plateforme.
Déconnexion du restaurateur	➢ Accessible depuis toute page connectée, ➢ Suppression sécurisée de la session	Frontend : React API Express JWT	➢ Sur toute page connectée, un bouton "Déconnexion" appelle l'endpoint /auth/logout. ➢ Le backend invalide le token, et le frontend supprime le JWT stocké (dans le localStorage). ➢ Le backend peut répondre 200 sans action particulière (JWT stateless). ➢ L'utilisateur est redirigé vers la page d'accueil non connectée.	1) Bonne pratique de sécurité (invalidation côté serveur + purge du token côté client). 2) Implémentation standard dans les apps React + Node utilisant JWT.
Modification des informations utilisateur	➢ Gestion de plusieurs e-mails par un même utilisateur.	Frontend : React	➢ En cliquant sur l'onglet « mon compte », l'utilisateur peut ajouter/supprimer des e-mails secondaires ou définir un nouvel	1) MongoDB : documents flexibles → les emails peuvent être stockés dans un tableau et

	➢ Changement d'e-mail principal, ➢ Sécurité renforcée ➢ Architecture modulable permettant l'ajout futur de la gestion des moyens de paiement.	Backend : API Express MongoDB JWT	e-mail principal. ➢ Chaque action déclenche une requête sécurisée vers l'API Express (PUT /user/emails). ➢ Le backend vérifie l'identité via JWT, met à jour le document utilisateur dans MongoDB et renvoie l'état mis à jour.	modifiés facilement sans schéma rigide. →Idéal pour des structures évolutives (emails, préférences, paiements...). 2) Sécurité maîtrisée via JWT (seul le restaurateur connecté peut modifier ses informations).
Dashboard restaurateur	➢ Doit regrouper création, diffusion, impression de menus et derniers articles de blog ; ➢ UX claire et responsive.	Frontend : React Backend : API Express MongoDB JWT	➢ L'utilisateur connecté et authentifié (JWT) à accès à un tableau de bord avec : encarts "Créer", "Diffuser", "Imprimer" un menu et section "Pour aller plus loin" avec les 3 derniers articles de blog (titre, photo, lien) récupérés via API. ➢ Solution extensible future : possibilité d'ajouter un blog interne à MenuMaker.	1) Tableau de bord centralise les actions clés → UX améliorée. 2) Architecture modulable : ajout futur de fonctionnalités (blog interne) possible sans refonte.
Autosave des menus	➢ Pas de perturbation utilisateur, ➢ Limiter les requêtes au serveur.	Frontend : Debounce React Backend : endpoint « drafts »	➢ Sauvegarde automatique après une période d'inactivité, stockée dans une collection "drafts". ➢ L'utilisateur retrouve ses brouillons dans la page "Mes menus", avec possibilité de les rééditer.	1) Le debounce : réduction du nombre de requêtes et protection des performances du serveur. 2) UI sécurisante pour l'utilisateur : pas de perte de travail en cours en cas de fermeture d'onglet ou de coupure réseau.

II. Liens avec le back-end

● Langage serveur

Le serveur sera développé en **Node.js**, associé au framework **Express.js.**
Node.js est adapté car il permet d'utiliser **le même langage (JavaScript)** côté front et côté back. Cela facilite la cohérence du projet, la communication entre les équipes et la maintenance du code. Il est particulièrement adapté aux applications web interactives nécessitant des mises à jour fréquentes, comme un éditeur de menus en temps réel.
Express apporte une structure simple et performante pour créer des routes et gérer les requêtes HTTP.

● API nécessaire

Une **API REST** est nécessaire.
Cette API permettra au front-end (React) de communiquer avec la base de données pour gérer le menu du restaurant (**CRUD**).
Les principales routes prévues seront :

- **GET** : récupérer tous les éléments du menu ou un menu complet.
- **POST** : ajouter un nouvel élément ou créer un nouveau menu.
- **PUT / PATCH** : modifier un élément existant ou mettre à jour un menu.
- **DELETE** : supprimer un élément ou supprimer un menu complet.

Cette API rend possible l'édition en temps réel du menu par l'utilisateur et assure une architecture claire et scalable.

On utilisera également **l'API Instagram Graph** pour la publication automatique ou semi-automatique des visuels.
Deliveroo ne disposant pas d'API publique, la diffusion se fait via un lien externe et un export manuel.

• Base de données choisie

La base utilisée sera **MongoDB**, une base **NoSQL**.
Ce choix est pertinent car les données du menu (plats, prix, catégories, images) sont des données **souples**, dont la structure peut évoluer au fil du projet. Les données liées aux comptes utilisateurs, telles que les plusieurs adresses e-mail associées à un même compte, peuvent également être stockées facilement dans des documents flexibles.
Ce modèle rend l'architecture naturellement extensible, notamment pour ajouter ultérieurement la gestion des moyens de paiement et autres données.
MongoDB permet de stocker ces éléments sous forme de documents flexibles (JSON), ce qui s'intègre naturellement avec JavaScript et avec l'API REST.
Ce format facilite également les mises à jour fréquentes des menus (ajouts, suppressions, modifications).

Pour simplifier l'interaction avec la base, on utilise Mongoose, un ODM JavaScript fonctionnant dans Node.js, dont les schémas garantissent la validation et la cohérence des données.

III. Préconisations concernant le domaine et l'hébergement

• Nom du domaine

Le nom du domaine doit être simple, mémorisable et représentatif de l'application.
Exemple : www.menumaker.com ou www.monmenupro.fr.
Il est recommandé de choisir un domaine en **.com** ou **.fr**, selon la cible des utilisateurs.

• Nom de l'hébergement

Nous avons choisi le type d'hébergement **cloud commercial** (**Vercel / Render + MongoDB Atlas**), qui utilise

plusieurs serveurs virtuels travaillant ensemble pour fournir des ressources flexibles et évolutives.

Il **simplifie** le déploiement, permet une **montée en charge automatique** (lorsqu'il y a beaucoup d'utilisateurs), ne nécessite pas de configuration système complexe, et offre un **quota gratuit** pour les petits projets, avec paiement uniquement si on dépasse ce quota.
Il est particulièrement adapté à notre projet moderne (**Node.js + API + MongoDB**).

Comparé aux autres solutions : le mutualisé manque de flexibilité (ex : o2switch), le dédié ou VPS demandent beaucoup de maintenance et coûtent plus cher (ex : siteGround), et le cloud non commercial nécessite de construire et gérer soi-même l'infrastructure (ex : OpenStack).

• Adresses e-mail

Pour la communication officielle et la gestion des comptes administrateurs/restaurateurs, il est recommandé d'utiliser :

- ✓ Une **adresse générale** : contact@menumaker.com
- ✓ Une **adresse support** : support@menumaker.com
- ✓ Une **adresse administrative** : admin@menumaker.com

Ces adresses peuvent être configurées via le fournisseur de domaine ou des services professionnels type **Gmail Workspace** ou **Microsoft 365** pour garantir la fiabilité et la sécurité des mails.

IV. Accessibilité

- **Compatibilité navigateur :** L'application sera compatible avec les principaux navigateurs modernes (Chrome, Firefox, Edge, Safari) afin de toucher le maximum d'utilisateurs.

- **Types d'appareils :** L'interface sera responsive, utilisable sur ordinateurs, tablettes et smartphones, pour permettre aux restaurateurs de gérer leurs menus en mobilité.

- **Accessibilité pour tous :** le design et l'interface respecteront les bonnes pratiques WCAG (contraste suffisant, navigation clavier, labels pour lecteurs d'écran) afin de faciliter l'accès aux personnes malvoyantes ou ayant d'autres handicaps.

V. Recommandations en termes de sécurité

• Accès aux comptes

- ✓ **Authentification sécurisée** : MenuMaker utilise une authentification 100% passwordless : à chaque tentative de connexion, un **Magic Link** (contenant un token unique à durée de vie courte) est envoyé à l'adresse e-mail saisie. Le même flux sert à la création de compte et à la connexion. Cela évite la gestion classique des mots de passe et réduit les risques liés à leur vol.
- ✓ **JWT (JSON Web Token)** : les sessions sont gérées via des tokens signés, avec durée de vie limitée et possibilité d'invalidation côté serveur.
- ✓ Respect des bonnes pratiques **OWASP** pour se protéger contre XSS, injections et CSRF.

• Packages / librairies externes

- ✓ **Mises à jour** régulières des packages npm utilisés (React, Express, Mongoose, Puppeteer…) pour corriger les vulnérabilités connues.
- ✓ Limiter l'usage des packages uniquement à ce qui est nécessaire et vérifier leur **fiabilité et maintenance**.
- ✓ Effectuer un **audit régulier** des dépendances avec des outils comme npm audit ou Snyk.

• Protection des données et RGPD

- ✓ Collecte minimale des données personnelles : e-mail pour Magic Link et informations liées aux menus.

- ✓ Stockage sécurisé des données dans MongoDB (accès restreint, HTTPS).
- ✓ Possibilité pour l'utilisateur de **demander la suppression de ses données**, conformément à la réglementation européenne **RGPD**.

- **Sécurité complémentaire**

- ✓ API : rate limiting, validation stricte des requêtes, CORS configuré.
- ✓ Serveur : Helmet sur Express, logs de sécurité, séparation des environnements.
- ✓ Secrets : clés stockées dans des variables d'environnement, rotation régulière.

VI. Maintenance du site et futures mises à jour

• **Maintenance générale** : vérification régulière du fonctionnement, correction des bugs, mises à jour de sécurité et des dépendances, et surveillance de l'infrastructure.

• **Mises à jour majeures :** les évolutions importantes (ex. Node.js, MongoDB) sont traitées comme des projets spécifiques.

• **Évolutions fonctionnelles :** ajout de nouvelles fonctionnalités, améliorations UI/UX et adaptations selon les besoins des utilisateurs.

• **Objectif :** garantir la disponibilité, la sécurité, la performance et la fiabilité du site sur le long terme.

• **Supervision :** sauvegardes régulières, contrôle de l'intégrité des données et conformité aux bonnes pratiques et à la réglementation (RGPD).