MENU MAKER by Qwenta

Spécifications techniques

LOMBARDO Kevin

Version	Auteur	Date	Approbation
1.0	LOMBARDO Kevin	[date de réalisation du document]	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)
La page d'accueil doit être attrayante avec des animations fluides et un carrousel d'images pour la section "Personnalisez votre menu"	La page d'accueil doit capter l'attention de l'utilisateur dès son arrivée, avec des images dynamiques et une navigation fluide entre les différentes sections.	**Transitions CSS + React Carousel (react-responsive-carousel)**	Des **transitions CSS** seront utilisées pour animer les éléments de la page (comme les images) et créer un effet visuel engageant. Un carrousel d'images sera intégré via la bibliothèque React **react-responsive-carousel**, pour faire défiler les images dans la section "Personnalisez votre menu".	1) Les **transitions CSS** permettent de créer des animations légères et performantes qui rendent la page plus attrayante sans alourdir la performance. 2) L'utilisation de **react-responsive-carousel** permet une présentation dynamique et interactive des images, offrant une meilleure expérience utilisateur tout en conservant une navigation fluide.
L'utilisateur doit pouvoir naviguer de façon fluide sur l'application sans temps de chargement	L'application doit permettre une navigation rapide et fluide entre les différentes pages sans nécessiter un rechargement complet de la page.	**React Router**	**React Router** permet de gérer la navigation entre les différentes pages de l'application sans rechargement complet de la page. Les changements d'URL n'entraînent pas un nouveau chargement de la page, ce qui garantit une navigation instantanée.	1) **React Router** est la solution standard et optimisée pour la navigation via les routes dans les applications React. 2) Il offre une expérience utilisateur fluide en permettant des transitions rapides entre les pages sans rechargement complet, améliorant ainsi la performance et la réactivité de l'application.
Création de compte pour un internaute	Un internaute doit pouvoir créer un compte en renseignant son adresse mail.	**JWT (Node.js) / Amazon Cognito / Redux**	Lorsque l'internaute entre son adresse e-mail dans l'application, **Amazon Cognito** crée le compte utilisateur et envoie automatiquement un e-mail de vérification pour valider l'adresse e-mail. Ce processus est géré via **Cognito User Pools**, avec une configuration simple dans la console. Une fois l'e-mail vérifié, l'utilisateur peut accéder à l'application.	**Amazon Cognito** offre une solution robuste et sécurisée pour la gestion des utilisateurs, incluant la création de comptes, l'authentification, le chiffrement des mots de passe et l'authentification multi-facteurs, garantissant ainsi une sécurité renforcée. L'envoi automatique d'e-mails de confirmation simplifie le processus d'inscription, éliminant le besoin de services externes pour la gestion des e-mails.
Connexion d'un restaurateur	Le restaurateur doit pouvoir se connecter avec son adresse mail via une modale.	**React-modal / JWT / Redux / Amazon Cognito**	Pour la **connexion d'un restaurateur**, une **modale React** permet à l'utilisateur de saisir son adresse e-mail pour se connecter. **Amazon Cognito** gère l'authentification et l'envoi automatique d'un e-mail de vérification pour confirmer l'adresse de l'utilisateur. L'état du processus (chargement,	**Amazon Cognito** est utilisé ici pour gérer l'authentification du restaurateur de manière sécurisée, en automatisant l'envoi d'e-mails de vérification et en assurant la gestion des utilisateurs via des **pools d'utilisateurs**. Il permet de sécuriser le processus de connexion sans nécessiter de configuration complexe, tout en intégrant

			succès, ou échec) est suivi en temps réel grâce à **Redux**, garantissant une expérience utilisateur fluide avec des transitions d'état claires. Une fois l'e-mail vérifié, un **token JWT** est généré pour permettre l'accès sécurisé aux ressources de l'application.	facilement l'authentification avec des technologies comme **React-modal** et **Redux** pour une expérience fluide. Grâce à **Cognito**, l'application bénéficie d'une gestion centralisée des utilisateurs, de la possibilité d'activer des fonctionnalités comme l'authentification multi-facteurs, et de la génération de **tokens JWT** pour un accès sécurisé aux ressources.
Le restaurateur connecté pourra renseigner une catégorie de plat.	1. Créer une nouvelle catégorie 2. La création de catégorie s'ouvre dans une modale spécifique, et doit pouvoir être validé. 3. L'utilisateur connecté doit pouvoir sélectionner une catégorie crée précédemment.	**React-Modal / API de la base de données / JWT**	Lors de l'ajout d'une nouvelle catégorie, une **modale React** s'ouvre pour permettre à l'utilisateur de renseigner le nom de la catégorie. Cette catégorie est ensuite envoyée via une requête API à la base de données pour être enregistrée. Lors de la création ou de la modification du menu, l'utilisateur pourra **sélectionner** une catégorie existante précédemment enregistrée.	1) **React-Modal** permet d'afficher de manière fluide et esthétique une fenêtre modale pour la saisie des catégories sans quitter la page. 2) L'utilisation d'une **API** pour interagir avec la base de données permet de **persister** les catégories crées auparavant de manière sécurisée et de les rendre disponibles pour réutilisation.
Le restaurateur connecté pourra ajouter un plat / des plats dans mon menu.	Le restaurateur peut : **1-** Sélectionner une catégorie et la modifier ; **2-** Entrer les informations de son plat ; **3-** Une modale s'ouvrira pour saisir les différents plats de la catégorie ; **4-** Chaque plat pourra avoir une photo associée, un nom, un prix et une description **5-** Et il est possible de créer autant de plats que l'on souhaite	**React-Modal / Redux / API base de données / JWT AWS S3**	Lors de la création d'un plat, une **modale React** s'ouvre pour permettre au restaurateur de saisir les informations de son plat (nom, prix, description, photo). Une fois le plat renseigné, les données sont envoyées via une **requête POST** à l'API qui les enregistre dans la base de données. Après cela, les données sont récupérées via une **requête GET** pour mettre à jour l'état via **Redux** et afficher les plats créés dans le menu du restaurateur. Le tout est sécurisé par l'utilisation de **JWT** pour authentifier les requêtes API.	1) **React-Modal** permet d'afficher facilement une interface modale pour la création des plats, offrant une expérience utilisateur fluide. 2) **Redux** permet de maintenir et de mettre à jour l'état de l'application en temps réel sans avoir à gérer l'état local dans chaque composant, ce qui simplifie la gestion des données tout en assurant que les modifications apportées (création de plats) sont répercutées partout dans l'application.

Le restaurateur connecté pourra personnaliser le style de mon menu.	Le restaurateur doit pouvoir : **1**. Visualiser le menu actuellement créé **2**. Sélectionner une typographie **3**. Choisir une couleur de texte	**React / Redux** **React-color /** **Google-fonts-API**	Utilisation de **l'API Google Fonts** pour permettre à l'utilisateur de sélectionner une typographie personnalisée pour son menu. **React** et **Redux** géreront l'état des choix (typographie, couleur), et **React-color** permettra de choisir la couleur du texte. La typographie choisie sera ensuite appliquée dynamiquement à l'application via l'ajout d'une balise <link> dans le <head> du document.	1) **Google Fonts** est une API gratuite et largement utilisée qui permet d'intégrer facilement une large sélection de polices web, tout en garantissant un chargement rapide et performant grâce à son CDN. 2) L'intégration dynamique avec React via des liens <link> permet de personnaliser les polices en temps réel sans surcharger le bundle.
Le réstaurateur connecté pourra exporter son menu en PDF.	Le restaurateur doit pouvoir en un clic télécharger le fichier PDF correspondant à son menu	**React / Redux /** **React-pdf**	Utilisation de **React-pdf** pour générer le PDF directement à partir des composants React. Lors du clic sur un bouton, le menu est converti en un fichier PDF et proposé en téléchargement.	1) **React-pdf** permet de générer le PDF directement depuis des composants React, ce qui garantit une intégration fluide avec l'interface utilisateur et les données dynamiques de l'application. 2) L'exportation se fait côté client, ce qui évite de solliciter un serveur et améliore la réactivité de l'application. Le restaurateur peut donc télécharger immédiatement son menu sans latence.
Le restaurateur connecté pourra commander en un clic l'impression d'un menu.	L'encart "Imprimer un menu" doit être visible depuis ***le Dashboard.*** Il s'agit d'un lien qui s'ouvre dans un nouvel onglet. Le lien doit être fait vers le back-office de Qwenta	**React / Redux /** **React-Router /** **React-pdf**	Lorsque le restaurateur clique sur l'encart "Imprimer un menu" dans son Dashboard, **React-Router** redirige vers un nouvel onglet contenant l'interface du back-office de Qwenta. Ce dernier sera responsable de l'impression du menu.	1) **React** est utilisé pour la gestion de l'interface utilisateur du Dashboard, y compris l'affichage du bouton "Imprimer un menu" et la gestion de l'événement de clic. 2) **React-Router** permet de rediriger l'utilisateur vers le back-office de Qwenta dans un nouvel onglet, en garantissant que l'application actuelle reste intacte. 3) L'utilisation de **Redux** garantit que l'état du menu est centralisé et peut être récupéré par le back-office lors de l'impression, si nécessaire.

En tant que restaurateur, je dois pouvoir avoir accès à une vue regroupant les menus créés précédemment.	Au clic sur « Mes menus » une vue des menus précédemment crée s'affichent avec la date de création dans lequel il sera possible de modifier, de créer ou supprimer un menu.	**React / React-Modal/ Redux**	Lorsqu'un restaurateur clique sur "Mes menus", la vue des menus précédemment créés s'affiche via un composant **React**. L'état des menus est géré avec **Redux**. Des actions de création, modification et suppression sont effectuées via des appels API, et des fenêtres modales **React-Modal** permettent de modifier ou supprimer les menus.	1) **React** permet de rendre l'interface dynamique et d'afficher les menus créés. 2) **React-Modal** permet d'afficher des fenêtres modales pour la gestion des actions (création, modification, suppression). 3) **Redux** permet de centraliser l'état des menus et de gérer efficacement les interactions (création, modification, suppression).
En tant que restaurateur, je dois pouvoir exporter mon menu en un clic vers l'application Deliveroo.	L'encart **« Diffuser sur Deliveroo »** doit s'afficher dans la catégorie **« Exportez et diffusez »** Et au clic sur celle-ci l'utilisateur devra être redirigé sur l'application Deliveroo.	**API Deliveroo/ React + React Router / API OAuth2**	Lorsqu'un utilisateur clique sur le bouton diffusez sur deliveroo l'API Deliveroo nous enverra directement sur son application a partir de laquelle OAuth2 nous demandera de nous authentifier avec notre compte deliveroo afin d'y delivrer notre menu sous le format attendu par Deliveroo.	1) **API Deliveroo** est essentielle afin de permettre aux restaurateurs de transférer leurs menus pour qu'il soit accessible dans l'application Deliveroo et de les mettre à jour le cas échéant. 2) Elle utilise aussi un protocole d'autorisation largement utilisé pour gérer l'authentification et l'autorisation de manière sécurisé dans les applications web moderne qui se nomme **OAuth2**.
En tant que restaurateur je dois pouvoir partager mon menu sur Instagram facilement.	L'encart « Partager sur Instagram » doit s'afficher dans la catégorie « Exportez & diffusez », au clic sur l'encart, des images du menu au format carré sont générées et le restaurateur est redirigé vers son compte Instagram avec les photos carrées des menus.	**React / React-Router / API Instagram / Bibliothèque Fabric.js / API OAuth2**	Lorsqu'un utilisateur clique sur le bouton « partager sur Instagram », la bibliothèque externe Fabric.js permettra, avec des réglages préalables de modifier les images du menu de sortes à ce qu'elles soient au format carré. L'utilisateur est ensuite redirigé sur Instagram via l'API ou il pourra partager ses images a ce format recommandé par Instagram.	1)**L'API Instagram** permet de rediriger l'utilisateur directement sur Instagram, ce qui est essentiel pour partager ses Images & Menus. 2)**Fabric.js** : j'ai choisi fabric.js malgré que ça soit une bibliothèque externe et plus lourde que HTML5canvas par exemple car elle permet une redémentions automatique après réglage.

L'mutilateur connecté pourra se déconnecter.	Le restaurateur doit pouvoir se déconnecter depuis n'importe quelle page connectée	**Amazon Cognito**	Le restaurateur pourra se déconnecter depuis n'importe quelle page connectée grâce à **Amazon Cognito**, qui gère la suppression des tokens d'accès et la fin de la session utilisateur. Cela garantit une déconnexion sécurisée et immédiate.	**Amazon Cognito** gère la déconnexion de manière sécurisée en supprimant les **tokens d'accès** et les **tokens de rafraîchissement** associé à l'utilisateur, assurant ainsi que l'accès aux ressources protégées est révoqué. Cette fonctionnalité intégrée permet une gestion facile de la session utilisateur, sans nécessiter de configuration supplémentaire, tout en garantissant une expérience fluide et sécurisée lors de la déconnexion, quel que soit l'endroit où l'utilisateur se trouve dans l'application.

II. Liens avec le back-end

Quel langage pour le serveur ?

Nous avons choisi d'utiliser **Node.js** comme langage pour le serveur pour plusieurs raisons :

Il nous permet d'utiliser JavaScript à la fois pour le frontend et le backend, ce qui simplifie la gestion du projet en éliminant la nécessité de jongler entre

différents langages. Cela facilite également la communication et le partage des ressources et des connaissances entre les équipes frontend et backend.

Grâce à son architecture **asynchrone non-bloquante**, il permet une gestion optimale des requêtes concurrentes, ce qui est essentiel pour une application évolutive comme la nôtre. Cette caractéristique est particulièrement utile pour des opérations comme la création de menus ou la gestion des interactions avec des API tierces (Instagram, Deliveroo)
Il dispose de **NPM (Node Package Manager)** qui permet la mise a disposition d'un écosystème riche en bibliothèques et modules qui faciliteront la gestion des tâches courantes (authentification des utilisateurs, gestion des menus, intégrations des APIs etc.) Il en découlera un gain de temps considérable pour nos équipes.

Node.js bénéficie d'une communauté de développeurs dynamique à laquelle nos équipes pourront faire appel en cas de besoin. Quant au Framework **Express.js**, il est à la fois simple, flexible et rapide, permettant de mettre en place l'architecture d'une application web fonctionnelle en un temps record. Cela représente, encore une fois, un **gain de temps** considérable pour nos équipes.

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

Oui, nous avons besoin de plusieurs API pour cette application :

- **API Instagram** :
  Cette API est essentielle pour permettre aux restaurateurs de partager facilement leur menu et les images associées sur leur compte Instagram.

- **API Deliveroo** :
  Cette API est utilisée pour intégrer des fonctionnalités de partage de menus directement vers la plateforme de livraison Deliveroo ainsi que pour la redirection vers celui. Cette API est utile si vous voulez offrir la possibilité aux restaurateurs de diffuser automatiquement leurs menus sur Deliveroo, afin qu'ils soient visibles pour les utilisateurs de cette plateforme.

Cet API devra être RESTful selon moi car :

- **Deliveroo** et **Instagram** sont des **APIs REST**, donc en utiliser une pour notre application nous ferait bénéficier de l'interopérabilité, de la simplicité et de la flexibilité pour intégrer ces services et bien d'autres (car largement adopté).

- Permet l'ajout de nouvelles fonctionnalités très simplement (système de paiement en ligne dans le futur)

Quelle base de données avez-vous choisie et pourquoi ?

Nous avons choisi la base de données **NoSQL** de Amazon « **DynamoDB** », voici pourquoi :

NoSQL car :

- **Le projet a une forte scalabilité**, or les bases de données NoSQL sont conçues pour se scaler horizontalement, donc nous pouvons facilement ajouter des serveurs supplémentaires pour gérer une charge accrue (utilisateurs, menus etc).

- **La flexibilité du modèle de données**, en effet nous n'avons pas de schéma rigide prédéfinie pour nos données, ce qui est intéressant dans le sens ou les menus peuvent avoir des variations et des ajouts constants (plats, entrées, desserts, photos etc.).

DynamoDB car :

- Faible latence, parfait pour nos nombreuses requêtes simultanées (création de menu, connexion utilisateur etc.) ;
- Tarif basé sur ce que nous utilisons en termes de lecture, d'écriture et de stockage ;
- Permet une gestion de la sécurité accrue (en couple avec **AMAZON Cognito** que nous utiliserons aussi) ;

III. Préconisations concernant le domaine et l'hébergement

- Comme nom de domaine nous réfléchissons encore actuellement mais nos avis semblent porter sur :

  www.QwentaMenuMaker.fr
  et :
  www.MenuMakerbyQwenta.fr

  Afin de rendre le nom plus adaptable, dans l'éventualité d'une expansion vers de nouveaux marchés à court ou moyen terme.

- Pour ce qui est de l'hébergement, notre application a une **forte scalabilité** car nous sommes déjà **leader** et avons d'ores et déjà **une clientèle fidèle et fournis**.
  Il nous faut donc un Cloud **fiable**, **rapide** et capable de pouvoir **gérer une croissance rapide** et **des besoins en traitement de données massives**.
  Nous avons donc pensé au Cloud d'**« Amazon Web Service ».**

- Nous avons déjà un lien « Besoin d'aide ? » sur la modale de connexion de l'application qui permettra d'envoyer un mail à notre support dont voici le mail :

« support@QwentaMenuMaker.fr »

Il y aura aussi un mail pour que les restaurateurs puissent nous contacter :
« contact@QwentaMenuMaker.fr »

Et le mail info pour les informations générales ou le marketing :
« info@QwentaMenuMaker.fr »

IV. Accessibilité

- Le site devra-t-être développé en **version desktop**. Pas de version mobile à développer ni & prévoir.

- L'application doit être compatible avec les dernières versions des navigateurs :
  - **Google Chrome** ;
  - **Safari** ;
  - **Firefox**.

- Les supports technologiques sur lesquelles l'application sera utilisable sont :
  - **PC Portable** ;
  - **PC de bureau** ;
  - **Tablette** (*en mode paysage*) ;

- L'application devra être accessible au minimum :

- **Navigable depuis un clavier** ;
- **Lisible par un lecteur d'écran**.

V. Recommandations en termes de sécurité

- Pour gérer l'authentification des utilisateurs nous utiliserons les fonctionnalités principales d'**AMAZON Cognito** dont la
  - **Gestion sécurisée des mots de passes** (Mots de passe **Forts** & politiques de sécurité strictes pour ceux-ci) ;
  - **La création des comptes utilisateurs** (vérification d'e-mail valide et validation par envoi d'e-mail limité dans le temps, limitation de tentatives de connections, Captchas etc.).
  - **La gestion des sessions d'utilisateur** (gestion de durée de vie du **Token** d'accès, protection contre vol de **Token** etc)

- Pour les plugins et les extensions :
  - Nous ferons attention à toujours utiliser des sources fiables (**Npm** pour node.js) ou **Github** et qui sont souvent mis à jour ;
  - Nous vérifierons la réputation de ceux-ci auprès de la communauté de développeur ;
  - Nous limiterons les accès API, et les plugins externes au strict minimum ;
  - Nous mettrons à jour régulièrement les plugins et les dépendances utilisés dans l'application pour inclure les derniers correctifs de sécurité.
  - Pour les communications, nous nous assurerons que toutes les connexions à notre application, y compris les APIs et les services externes, sont sécurisées en utilisant HTTPS afin que les données échangées soit cryptées.

- Quelques recommandations générales :
  - Nous pourrions aussi envisager plus tard une protection contre les attaques DDOS (**AWS Shield**) ;
  - Nous pourrions mettre en place des sauvegardes régulières de la base de données et des fichiers importants pour garantir la récupération rapide des données en cas d'incident (**AMAZON DynamoDB** en propose);

VI. Maintenance du site et futures mises à jour

Dans le cadre du contrat de maintenance nous nous engageons à :

- Instaurer une veille technologique afin de rester informé des dernières innovations et tendances à venir en matière de développement en :

    - Lisant attentivement les rapports des mises à jour des technologies que nous utilisons ;
    - Participant à des forums actifs de développeurs passionnés ;
    - Lisant des blogs/magazines spécialisés.

- Instaurer une planification des mises à jour (sécurité, corrections de bugs, améliorations fonctionnelles, maj dépendances, optimisations) :

    - On mettra à jour régulièrement les **dépendances** et nous utiliserons des outils comme **NPM audit** ou **SNYK** pour vérifier les vulnérabilités de sécurité de celle-ci ;
    - On instaurera une automatisation du processus de mise à jour avec la méthode **CI/CD** (test automatique au commit / déploiement en cas de réussite de test)

- Instaurer une surveillance continue :

    - Suivi de l'impact des mises à jour effectué avec **AWS Cloud Watch & Google Lighthouse** afin de réagir au plus vite en cas d'anomalie ou de baisse de performance sur notre application.
    - Collecter les retours des utilisateurs de notre application afin de trouver d'éventuelles problèmes non détectés en interne.
    - Effectuer des sauvegardes régulières de notre base de données, comme dit plus haut, nous utilisons **DynamoDB** pour notre application, ce service propose des options de sauvegarde et de reprise de données.

VII. Conclusion

En résumé, nous avons fait des choix technologiques stratégiques pour assurer la scalabilité, la sécurité, et la performance de l'application **Menu Maker**. En optant pour **Node.js** et son environnement, nous avons choisi une solution moderne, performante et adaptée aux besoins d'une application web évolutive. L'utilisation d'une **base de données NoSQL** avec **Amazon DynamoDB** nous permet d'assurer une gestion optimale des données et une grande souplesse pour évoluer en fonction de la demande.

L'intégration des **APIs Instagram** et **Deliveroo** à travers une API **RESTful** nous garantit une interopérabilité parfaite, facilitant l'ajout de nouvelles fonctionnalités à l'avenir.

Le choix de l'**hébergement Amazon Web Services (AWS)** et des outils associés, ainsi que les solutions mises en place pour l'**accessibilité** et la **sécurité** (notamment avec **Amazon Cognito**), permettent de garantir une application robuste, sécurisée, et prête à évoluer face à la croissance du projet.

De plus, une **veille technologique** constante, ainsi qu'un suivi rigoureux des mises à jour et de la performance, seront effectués afin de maintenir l'application à la pointe de la technologie et répondre aux besoins de nos utilisateurs.
Enfin, avec un focus sur l'optimisation continue et la gestion de la scalabilité, le projet Menu Maker est conçu pour répondre aux besoins présents tout en anticipant son expansion et ses évolutions futures.