DOSSIER D'ARCHITECTURE TECHNIQUE (DAT)

AMANN FLORIAN

GOOD FOOD 3.0

Table des matières

1. Contexte

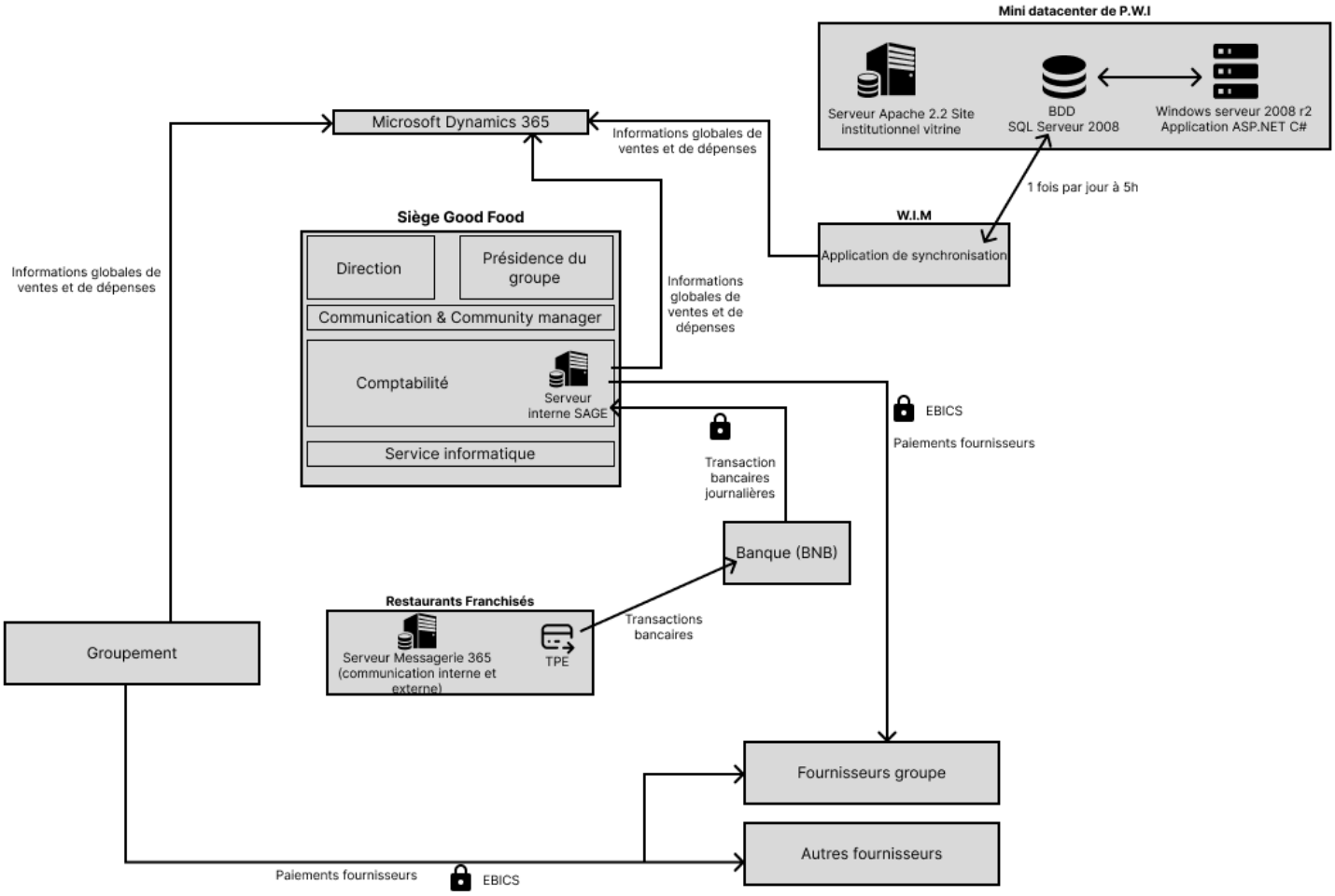
a. Introduction

Le groupe Good Food fondé en 1992 est implantée en France, en Belgique et au Luxembourg et compte plus de 150 restaurants. Le groupe compte un millier de collaborateurs, proposent 200 recettes sur l'ensemble des restaurants et a livré un million de repas sur l'exercice 2022. Le groupe est constitué de restaurants franchisés et d'autres groupements similaires.

Le groupe souhaite totalement refondre leur solution de commande en ligne. Ce document regroupe les différents points concernant l'architecture de la nouvelle application ainsi que différents conseils pour réussir la mise en place de cette dernière au sein de leur système d'information. Good Food tient également à ce que cette nouvelle solution soit compatible avec l'engagement de développement durable de l'entreprise.

b. Existant

I. Schéma de l'existant



II. Matrice des flux existants

Source	Destination	Données échangées	Fréquence	Protocole	Description
BDD SQL Server 2008	Application de synchro	Infos clients & ventes	Quotidienne		Synchronisation quotidienne des données de la BDD vers l'ERP
Application de synchro	ERP Dynamics 365	Infos clients & ventes	Quotidienne		Synchronisation quotidienne des données de la BDD vers l'ERP
Logiciel SAGE	ERP Dynamics 365	Infos globales de ventes et dépenses	Variable		Report des informations globales de ventes et de dépenses dans l'ERP
Logiciel SAGE	Fournisseurs groupe	Paiements bancaires	Variable	EBICS	Paiement des fournisseurs
Banque	Logiciel SAGE	Transactions bancaires	Quotidienne	EBICS	Transfert de toutes les transactions des restaurants franchisés vers le logiciel SAGE
Service communication et community management	ERP Dynamics 365	Réclamations et suivis clients	Variable	HTTPS	Report des réclamations client et leurs suivi dans l'ERP
Groupement	ERP Dynamics 365	Infos globales de ventes et dépenses	Variable		Report des informations globales de ventes et de dépenses dans l'ERP
Groupement	Fournisseurs groupe	Paiements bancaires	Variable	EBICS	Paiement des fournisseurs
Groupement	Autres fournisseurs	Paiements bancaires	Variable	EBICS	Paiement des fournisseurs

2. Besoins fonctionnels

a. Historique du projet

Initialement, l'entreprise disposait d'une première solution web de commande en ligne développée en Java qui datait 1998. Mais le changement de mode de consommation des clients a nécessité une refonte vers une solution d'e-commerce, ce qui a engendré une refonte de la solution en 2004 en ASP.NET avec le langage C#.

Cependant, après plusieurs années de production, il s'est avéré que cette solution en ASP.NET posait différents problèmes aussi bien d'un point de vue technique que d'un point de vue fonctionnel. En plus de ces problèmes les habitudes des consommateurs ont évolué et la solution n'offrait plus l'expérience que les clients exigeaient, ce qui est critique pour l'enseigne Good Food car le marché sur lequel l'entreprise se place est de plus en plus concurrentiel. C'est pourquoi l'entreprise à décider de faire auditer sa solution par un prestataire spécialisé dans le développement web et mobile hybride du nom de Do It Web (DIW).

L'objectif de ce projet est de proposer une solution adaptée aux besoins de Good Food, qui leur permettra de répondre au besoin de leurs clients tout en s'inscrivant dans leur démarche de développement durable.

b. Attentes

L'objectif du projet est de réaliser une refonte complète de la solution déjà existante pour qu'elle soit mieux adapté aux habitudes de consommation des clients tout en restant concurrentiel. Ainsi, le groupe Good Food attend une solution web qui fournira les fonctionnalités suivantes :

- La gestion des clients (à savoir l'inscription, la gestion du compte et les réclamations)
- L'authentification
- La gestion des menus
- La commande de repas (via un système de panier en ligne)
- Le paiement
- L'organisation des livraisons (incluant la possibilité d'indiquer l'heure à laquelle il faut livrer)
- Le service des réservations
- Le service franchisé (pour créer des promotions, gérer les fournisseurs, etc.)

De plus, le groupe souhaite également mettre à disposition de leurs clients une application mobile pour qu'ils puissent effectuer leurs commandes et gérer leurs comptes. Le tout devant être conforme avec l'engagement de développement durable de l'entreprise.

c. Sponsor du projet

La direction de l'entreprise Good Food est à l'origine de ce projet de nouvelle application de gestion des commandes et en est également le sponsor.

d. Contraintes et problèmes

I. Problèmes

L'architecture actuelle de Good Food comporte plusieurs problèmes qui sont les suivants :

- Le logiciel est installé sur un serveur Windows Server 2008 R2. Cela pose un problème car le support Microsoft n'est plus assuré. Aussi, au vu de l'ancienneté de la technologie beaucoup de failles de sécurité rendent le SI vulnérable.

- Le code source de l'application n'a pas de documentation. Cela rend compliqué l'intégration de nouvelles fonctionnalités et la maintenance.

- L'expérience utilisateur est impactée par les performances qui plafonnent, ce qui fait baisser la satisfaction client. Cela s'explique par la présence de beaucoup de bugs dans le logiciel.

- Good Food ne dispose à ce jour d'aucune application mobile. Cela implique un désavantage conséquent par rapport aux concurrents et sans cette solution mobile, la société ne peut pas répondre aux exigences de sa clientèle qui a pris l'habitude d'utiliser des solutions mobiles.

- La communication actuelle peut être améliorée car l'entreprise reçoit beaucoup de demandes contradictoires par rapport à leur cahier des charges qui a été établi par le service informatique.

- Dans le service informatique de l'entreprise il n'y aucune compétence de développement, ce qui rend l'entreprise dépendante d'intervenants externes, donc un manque d'autonomie et cela peut avoir divers impacts négatifs sur le système d'information.

II. Contraintes

Lors de la réalisation de ce projet, diverses contraintes ont été formulées elles sont énoncées ci-dessous :

- La totalité des données doit être conservée et récupérée dans la nouvelle base.
- La migration des données vers la nouvelle solution devra se faire sans aucun impact sur la production.
- Le code devra être documenté.
- Adoption d'un nouveau système de caisse dont le déploiement est assuré par le fournisseur de caisse lui-même en simultané du développement de la nouvelle application.
- La nouvelle appli devra être accessible depuis d'autres solutions (compta, etc.)
- La nouvelle application devra être compatible avec l'engagement de développement durable de l'entreprise.

3. Besoins non-fonctionnels

a. Disponibilité

Une grande partie de l'activité commerciale de Good Food repose sur les ventes en ligne. Ainsi, le site doit rester disponible pour répondre aux besoins des clients. Une interruption pourrait avoir comme conséquence une perte de revenus, de client ainsi qu'une dégradation de l'image de l'entreprise et de la satisfaction client. Ainsi, le site doit rester opérationnel en permanence, en portant une importance particulière aux heures de repas.

b. Évolutivité

Pour rester compétitif et gagner des parts de marché l'application de Good Food devra évoluer pour s'adapter aux habitudes changeantes des consommateurs. Cela implique de faire des modifications sur l'applications et d'y ajouter des fonctionnalités.

c. Sécurité

La sécurité est un enjeu majeur pour la société Good Food, au vu de quantité et de la sensibilité des données qu'elle traite. Ainsi, respecter les normes comme les RGPD est crucial pour respecter la législation européenne et garder la confiance des consommateurs.

d. Scalabilité

Au vu du nombre de client actuel de Good Food, il est nécessaire pour la pérennité du logiciel et de l'entreprise d'anticiper une augmentation du nombre de clients. Pouvoir s'adapter à la demande est essentiel pour garantir une bonne satisfaction client et pour rester concurrentiel.

4. Décisions d'architecture

I. Architecture Générale

Critères	Evolutivité	Coût (Moins cher)	Maintenabilité	Flexibilité	Facilité de mise en place	Complexité (Facile d'accès)	Total pondéré
Pondération	5	5	4	3	2	2	
Micro-services	5	3	5	5	3	2	85
Orientée Evènement	3	4	3	3	5	4	74
Client-Serveur	4	4	5	4	5	4	90
Orienté Services	4	4	2	2	4	3	68
Modulaire	4	3	4	4	2	2	71
Couches	4	3	4	3	2	2	68

Micro-services : Les micro-services sont une approche architecturale où une application est décomposée en petits services indépendants, chacun responsable d'une fonction spécifique, favorisant la flexibilité et la scalabilité.

Orienté événements : L'architecture orientée événements se base sur la communication asynchrone entre composants logiciels, où les événements déclenchent des actions, facilitant ainsi la réactivité et la gestion distribuée des systèmes.

Client-serveur : L'architecture client-serveur divise une application en deux parties distinctes, le client qui demande des ressources ou des services, et le serveur qui fournit ces ressources ou services, permettant une répartition des tâches et une gestion centralisée.

Orientés services : Les architectures orientées services (SOA) découpent une application en services indépendants et interopérables, favorisant la réutilisation et l'intégration facile des composants logiciels.

Modulaire : L'architecture modulaire consiste à diviser une application en modules autonomes et interconnectés, simplifiant le développement, la maintenance et la mise à jour des logiciels.

Couches : L'architecture en couches organise une application en strates ou couches distinctes, chaque couche fournissant des fonctionnalités spécifiques et communiquant avec les couches adjacentes, facilitant la gestion de la complexité et la séparation des préoccupations.

Au vu de la taille du projet, il semble judicieux de mélanger plusieurs architectures pour répondre aux exigences et avoir une idée claire du fonctionnement de l'application. Ainsi, l'architecture Client-Serveur sera utilisée pour la structure, un frontend communiquera avec une API qui servira de Gateway pour accéder à différents micro-services. Concernant le frontend, il sera plus orienté sur les évènements.

II. Technologie utilisée pour le Frontend

Critères	Coût (Moins cher)	Evolutivité	Flexibilité	Prise en main	Total pondéré
Pondération	5	4	3	2	
Angular	4	5	5	3	61
Vue.js	4	3	3	2	45
React	3	4	4	5	53
Ember.js	3	2	2	4	37

Angular : C'est un Framework open source développée par Google, utilisé pour la création d'applications web dynamiques et robustes, grâce à son approche basée sur les composants et son système de liaison de données bidirectionnelle.

Vue.js : Vue.js est une bibliothèque JavaScript progressive pour la construction d'interfaces utilisateur interactives. Sa simplicité et sa flexibilité en font un choix populaire pour le développement frontal, avec une architecture axée sur les composants et une intégration aisée.

React : c'est un Framework créé par Facebook, est une bibliothèque JavaScript déclarative et efficace pour la construction d'interfaces utilisateur évolutives. Son approche basée sur les composants et la gestion efficace du DOM en font un choix prisé pour le développement d'applications web modernes.

Ember.js : Ember.js est un Framework JavaScript complet destiné au développement d'applications web ambitieuses. Offrant des conventions structurées, Ember simplifie le processus de création en suivant des modèles préétablis tout en garantissant une performance optimale.

Ainsi, on utilisera Angular pour la partie frontend car c'est le Framework qui répond parfaitement aux besoins de ce projet, et en plus c'est un Framework open source. Mais aussi parce qu'on peut utiliser combiner Ionic et Angular pour créer une application mobile, ce qui permettra de gagner du temps.

III. Technologie utilisée pour le Backend

Critères	Coût (Moins cher)	Evolutivité	Flexibilité	Prise en main	Total pondéré
Pondération	5	4	3	2	
Node.js	4	5	5	4	63
ASP.NET Core	4	3	3	5	51
Spring (boot)	4	3	4	5	54
Django	4	2	2	4	42

Node.js : Node.js est un environnement d'exécution côté serveur basé sur le moteur JavaScript V8 de Chrome. Il permet de développer des applications web hautes performances de manière évolutive, en utilisant un modèle asynchrone et basé sur des événements.

ASP.NET Core : C'est un Framework open source développée par Microsoft pour la construction d'applications web et services. Il offre une plateforme modulaire et multiplateforme, permettant aux développeurs de créer des applications robustes et évolutives.

Spring (Boot) : Spring est un Framework Java qui facilite le développement d'applications d'entreprise. Spring Boot, une extension de Spring, simplifie encore plus le processus en fournissant des conventions par défaut et des fonctionnalités prêtes à l'emploi pour accélérer le développement d'applications Java.

Django : C'est un Framework web Python hautement productif, axé sur la simplicité et la rapidité de développement. Il suit le principe "batteries incluses", fournissant des fonctionnalités préconstruites pour les tâches courantes, ce qui en fait un choix populaire pour la création d'applications web robustes.

Et pour le back, Node.js sera utilisé pour ses nombreux packages à sa disposition.

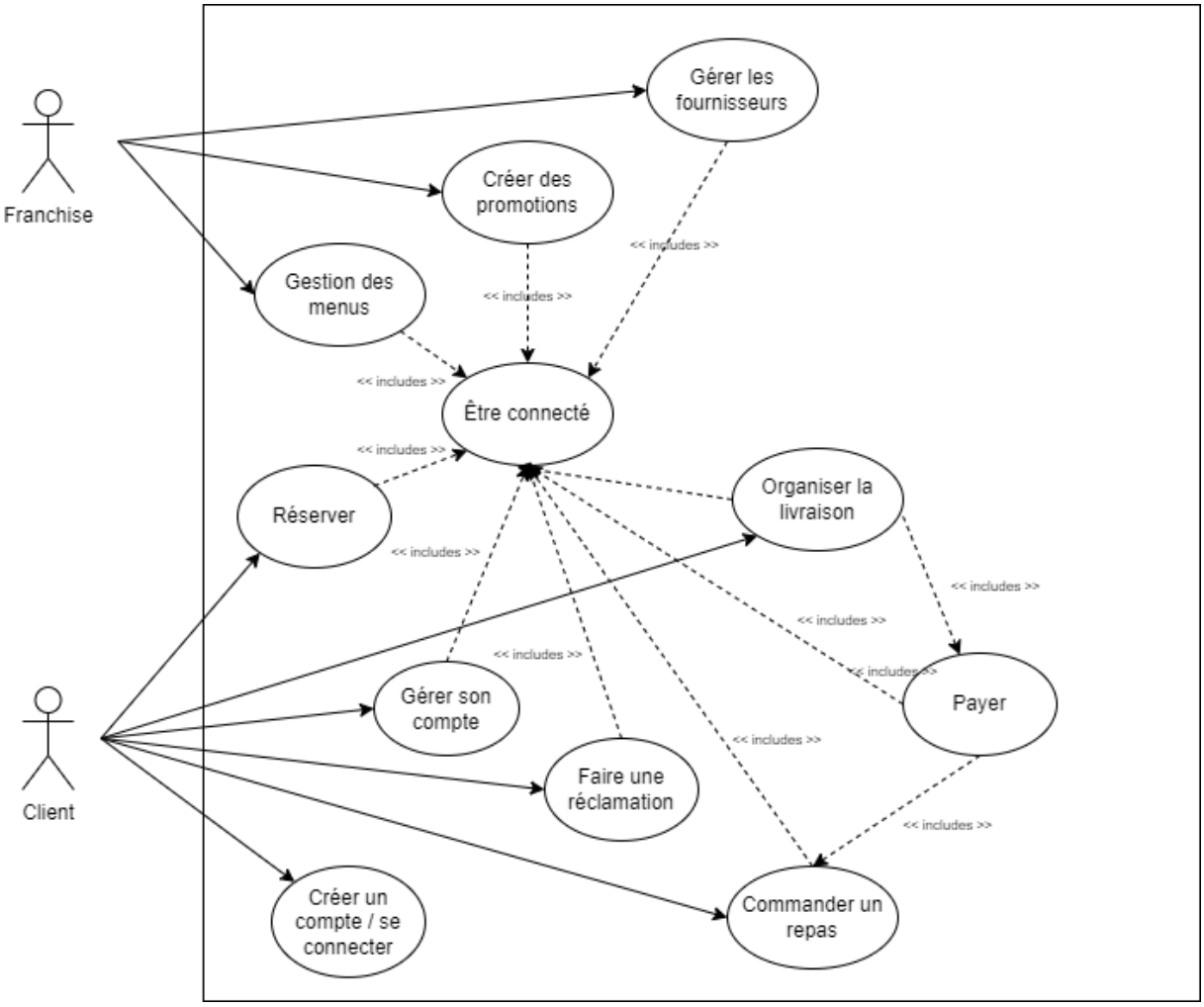
Concernant la base de données, une base SQL server sera toujours utilisée, mais dans une version plus récente, 2024 pour éviter tout soucis de compatibilité.

IV. Hébergement

Critères	Coût (Moins cher)	Sécurité	Fiabilité	Vitesse	Total pondéré
Pondération	5	4	3	2	
Google Cloud	2	5	4	4	50
AWS	3	4	5	4	54
OVH Cloud	1	3	3	3	32
Azure Cloud	3	4	4	3	49

Pour l'hébergement le cloud correspond au mieux pour le projet. Ainsi, la solution choisie est AWS.

5. Représentation fonctionnelle

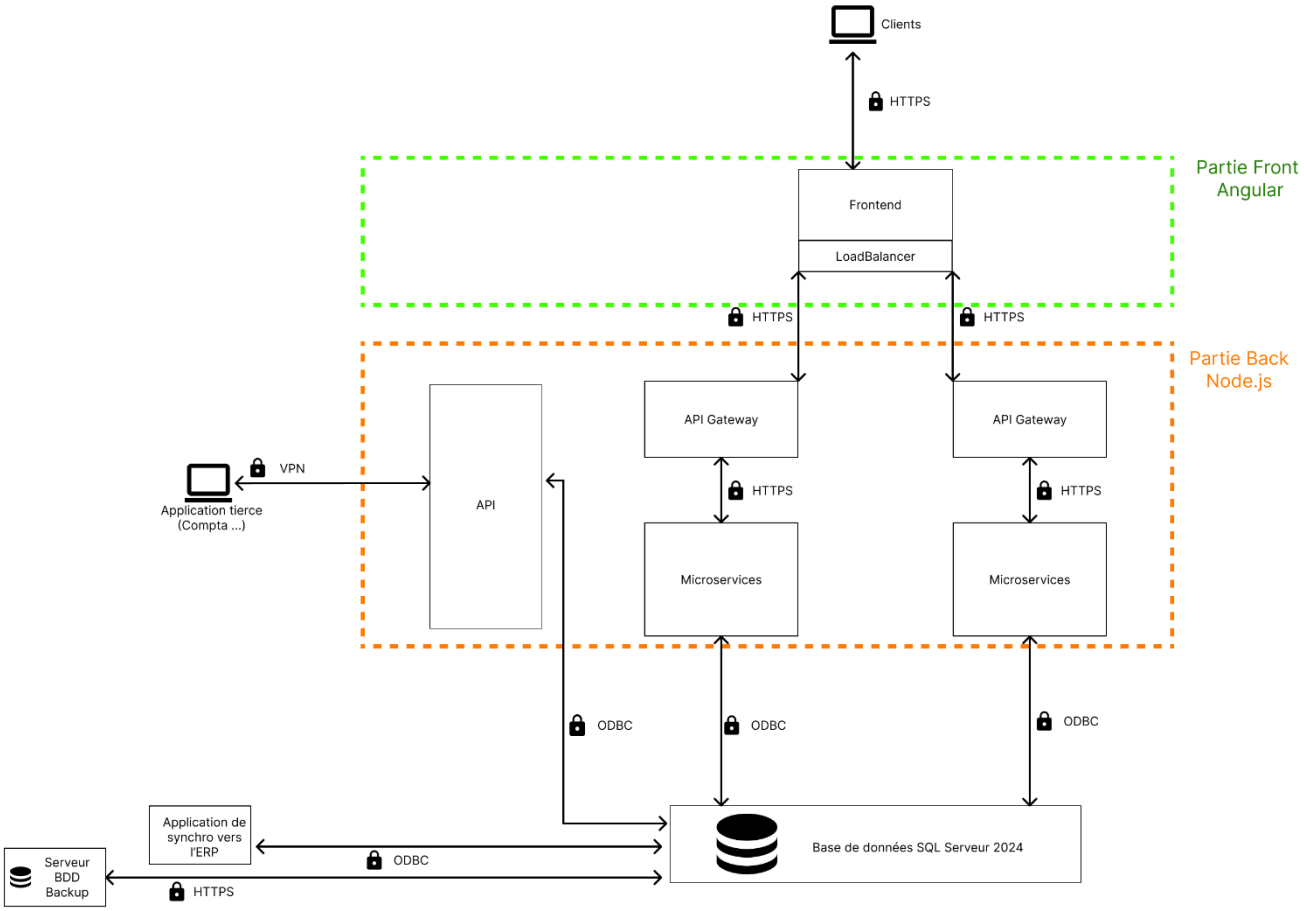


6. Environnements et dimensionnements

Nom	Type	OS	CPU	RAM	Stockage	Réseau
SRV_LB_FW	VM	Debian 12.4	16vCPU	128Go	500Go	AWS Cloud
SRV_FRONT_LB_1	VM	Debian 12.4	8vCPU	128Go	500Go	AWS Cloud
SRV_FRONT_LB_2	VM	Debian 12.4	8vCPU	128Go	500Go	AWS Cloud
SRV_API_G_MS_1	VM	Debian 12.4	4vCPU	64Go	1To	AWS Cloud
SRV_API_G_MS_2	VM	Debian 12.4	4vCPU	64Go	1To	AWS Cloud
SRV_API_G_MS_3	VM	Debian 12.4	4vCPU	64Go	1To	AWS Cloud
SRV_API_G_MS_4	VM	Debian 12.4	4vCPU	64Go	1To	AWS Cloud
SRV_API_FW	VM	Debian 12.4	4vCPU	64Go	1To	AWS Cloud
SRV_BDD	VM	Debian 12.4	8vCPU	128Go	4To	AWS Cloud
SRV_BDD_BACKUP	Physique	Debian 12.4	8vCPU	128Go	4To	Interne

A noter que ce dimensionnement est uniquement pour la production, il peut être dupliqué pour la préprod et pour le développement si besoin.

7. Représentation applicative



Interface Utilisateur (UI)

Frontend Développée avec Angular.
Fournit des points d'interaction pour les utilisateurs finaux.

Services d'Équilibrage de Charge (LoadBalancer)

Dirige les requêtes des utilisateurs vers les instances disponibles de l'UI et des services backend.

Application de Gestion (Compta...)

Se connecte via VPN en envoyant des requêtes HTTPS.
Intègre les fonctionnalités de l'application pour des applications externes.

Passerelle API (API Gateway)

Point d'entrée unifié pour les microservices backend.
Gère la sécurité, le routage des requêtes, et la médiation entre les clients et les microservices.

Microservices Back-End

Ensemble de services autonomes développés avec Node.js.
Exécutés dans des conteneurs pour une meilleure isolation et déploiement.

Base de Données Principale (SQL Server 2024)

Stocke les données transactionnelles et opérationnelles.
Interagit avec les microservices via ODBC.

Base de Données de Sauvegarde (Backup)

Maintient une copie des données pour la récupération en cas de défaillance.

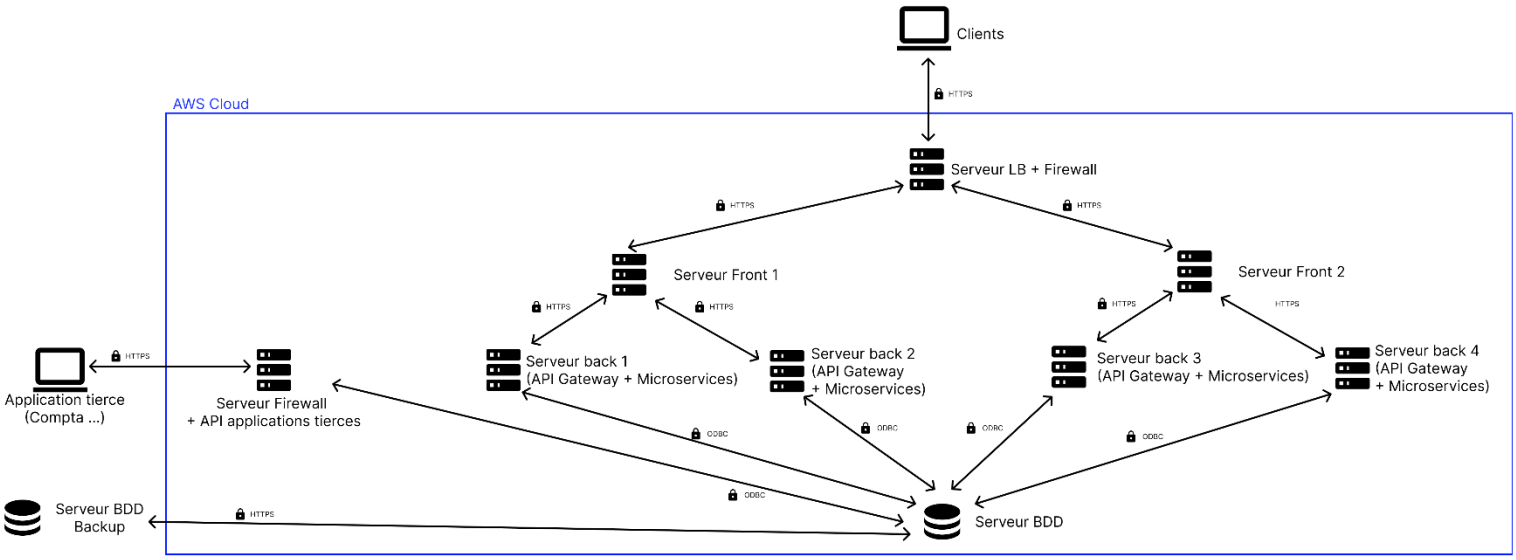
Considérations de Sécurité

Utilisation de HTTPS et VPN pour sécuriser toutes les communications.
Les données sensibles sont gérées et stockées en conformité avec les meilleures pratiques de sécurité.

8. Matrice de flux

Source	Destination	Données échangées	Fréquence	Protocole	Bidirectionnel
Client	Serveur Firewall + Lb	Requêtes/Réponses	A la demande Client	HTTPS	Oui
Serveur Firewall + Lb	Serveur Front 1	Requêtes/Réponses	A la demande Client	HTTPS	Oui
Serveur Firewall + Lb	Serveur Front 2	Requêtes/Réponses	A la demande Client	HTTPS	Oui
Serveur Front 1	Serveur Back 1	Requêtes/Réponses	A la demande Client	HTTPS	Oui
Serveur Front 1	Serveur Back 2	Requêtes/Réponses	A la demande Client	HTTPS	Oui
Serveur Front 2	Serveur Back 3	Requêtes/Réponses	A la demande Client	HTTPS	Oui
Serveur Front 2	Serveur Back 4	Requêtes/Réponses	A la demande Client	HTTPS	Oui
Serveur Back 1	Serveur BDD	Requêtes/Réponses	A la demande Client	ODBC	Oui
Serveur Back 2	Serveur BDD	Requêtes/Réponses	A la demande Client	ODBC	Oui
Serveur Back 3	Serveur BDD	Requêtes/Réponses	A la demande Client	ODBC	Oui
Serveur Back 4	Serveur BDD	Requêtes/Réponses	A la demande Client	ODBC	Oui
Serveur Firewall + API tierce	Serveur BDD	Requêtes/Réponses	A la demande Client	ODBC	Oui
Serveur Firewall + API tierce	Application tierce	Requêtes/Réponses	A la demande Client	HTTPS	Oui
Serveur BDD	Serveur BDD Backup	Requêtes/Réponses	Quotidienne	HTTPS	Oui

9. Représentation infrastructure



L'architecture représentée dans le schéma détaille un déploiement complexe dans l'environnement cloud d'Amazon Web Services (AWS). Cette architecture est conçue pour assurer à la fois la haute disponibilité, la scalabilité, la sécurité et la séparation des préoccupations entre les différents composants du système.

Composants de l'Architecture

Clients

Description : Utilisateurs finaux interagissant avec le système via une interface sécurisée.
Communication : Connexion sécurisée par HTTPS.

Serveur LB + Firewall

Description : Composant qui répartit la charge des requêtes utilisateurs entre les serveurs frontaux. Inclut un pare-feu pour une sécurité accrue.
Rôle : LoadBalancer (LB) : Répartition équilibrée du trafic pour optimiser les performances et la disponibilité.
Firewall : Filtrage du trafic pour bloquer les menaces potentielles.

Serveurs Frontaux

Composants : Serveur Front 1 et Serveur Front 2.
Description : Serveurs qui agissent comme point d'entrée initial pour les requêtes clients.
Rôle : Traitement des requêtes clients et redirection vers les serveurs back-end si nécessaire.

Serveur Firewall + API applications tierces

Description : Serveur intermédiaire dédié pour les communications avec les services externes.
Rôle : Sécurisation des interactions avec les applications tierces et exposition des API nécessaires.

Serveurs Back-End (Microservices)

Composants : Serveur back 1, Serveur back 2, Serveur back 3, Serveur back 4.
Description : Serveurs dédiés à la logique métier, organisés autour de microservices.
Rôle :
API Gateway : Fournit un point d'accès unifié pour les microservices.
Microservices : Services indépendants qui exécutent des fonctions spécifiques de l'application.

Serveurs de Base de Données

Composants : Serveur BDD et Serveur BDD Backup.
Description :
Serveur BDD : Contient la base de données principale du système.
Serveur BDD Backup : Sert de système de sauvegarde pour la base de données principale.
Rôle : Stockage des données et garantie de leur intégrité et disponibilité.

Applications tierces

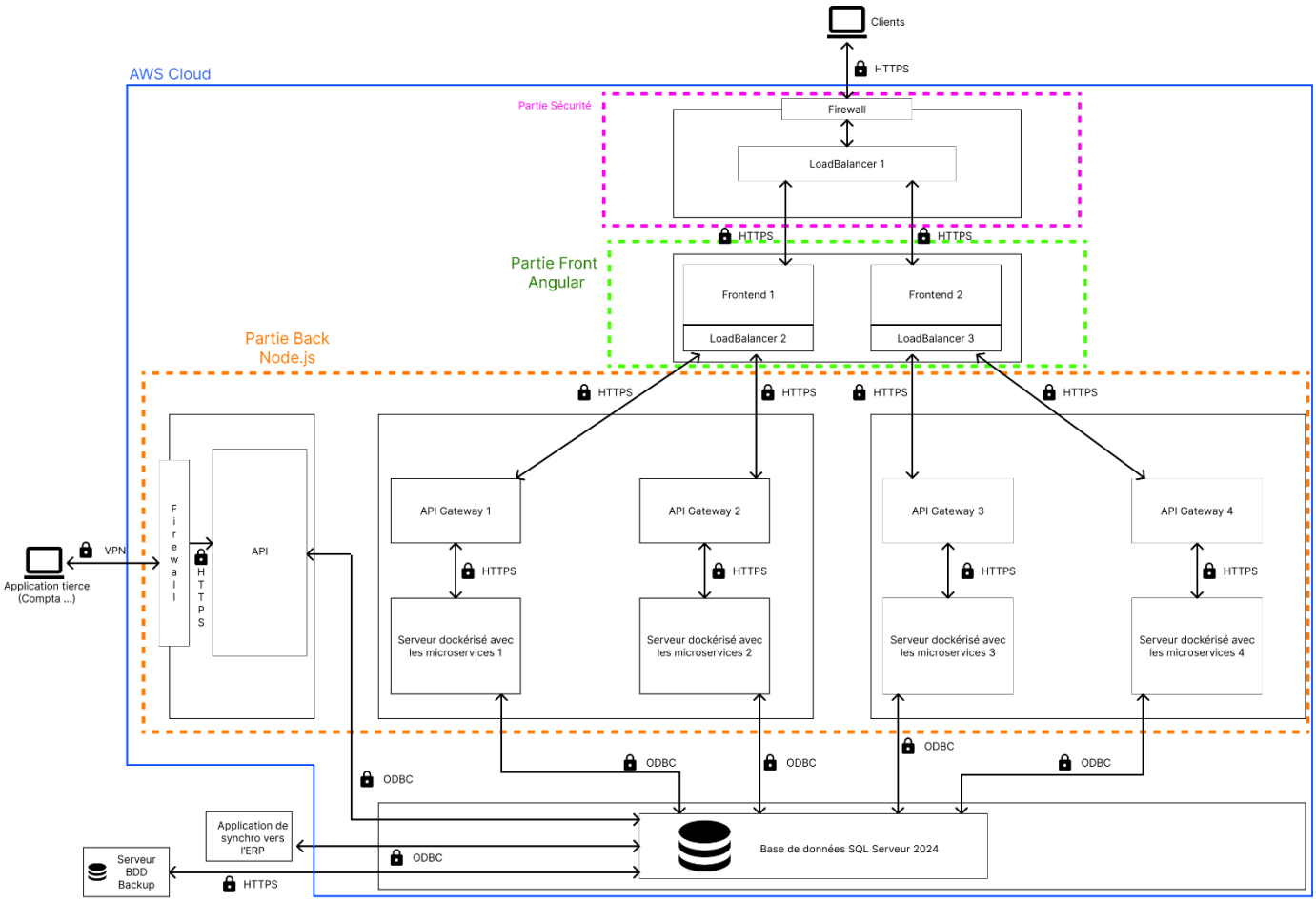
Exemple : Application tierce (Compta...).
Description : Applications métier externes qui nécessitent une interaction avec le système principal.
Communication : Sécurisée par un serveur de pare-feu dédié.

Protocoles de Communication

HTTPS : Utilisé pour sécuriser toutes les communications entre les clients, les serveurs frontaux, et le serveur d'applications tierces.
ODBC : Protocole standard utilisé pour la connexion et l'interaction avec la base de données par les microservices.

Considérations de Sécurité
- Utilisation systématique de HTTPS pour prévenir les écoutes indiscrètes et les attaques de type "man-in-the-middle".
- Pare-feu intégrés dans les points critiques de l'architecture pour protéger les ressources internes et externes.
- Stratégie de sauvegarde des données pour une résilience maximale face aux pannes éventuelles.

10. Représentation opérationnelle



La présente documentation décrit une architecture système déployée sur le cloud AWS, conçue pour fournir une application web sécurisée, scalable et résiliente. L'architecture est divisée en trois parties principales : sécurité, front-end et back-end, avec l'utilisation de conteneurs pour les microservices et de multiples équilibrages de charge.

Description des Composants

Partie Sécurité

Firewall : Fournit une protection contre les attaques et filtre le trafic entrant.
LoadBalancer 1 : Distribue les requêtes entrantes vers les serveurs frontaux pour une gestion optimale de la charge.

Partie Front : Angular

Frontend 1 et Frontend 2 : Serveurs qui hébergent l'interface utilisateur de l'application, développée avec le framework Angular.
LoadBalancer 2 et LoadBalancer 3 : Équilibreurs de charge supplémentaires qui garantissent la distribution équitable du trafic vers les serveurs frontaux et offrent une haute disponibilité.

Partie Back : Node.js

API Gateway 1, 2, 3, 4 : Points d'entrée pour les microservices back-end qui permettent un accès

sécurisé et une gestion simplifiée des API.
Serveurs dockerisés avec les microservices 1, 2, 3, 4 : Serveurs exécutant des conteneurs Docker, chacun hébergeant des microservices construits avec Node.js, permettant une architecture modulable et facile à maintenir.

Base de Données et Sauvegarde

Base de données SQL Server 2024 : Système de gestion de base de données relationnelle pour le stockage et la récupération des données de l'application.
Serveur BDD Backup : Serveur de sauvegarde pour la base de données, assurant la persistance des données en cas de défaillance du système principal.

Applications Tierces et Connectivité

Application tierce (Compta...) : Application externe qui peut se connecter au système via une connexion VPN pour un accès sécurisé.

Architecture de Communication

HTTPS : Toutes les communications entre les clients, les serveurs frontaux, les API Gateways et les serveurs de base de données sont sécurisées via HTTPS.
VPN : Une connexion VPN est utilisée pour les intégrations avec les applications tierces, renforçant la sécurité de l'échange de données.
ODBC : Utilisé pour la connexion entre les microservices et la base de données SQL Server, assurant l'intégrité des échanges de données.

Considérations de Sécurité

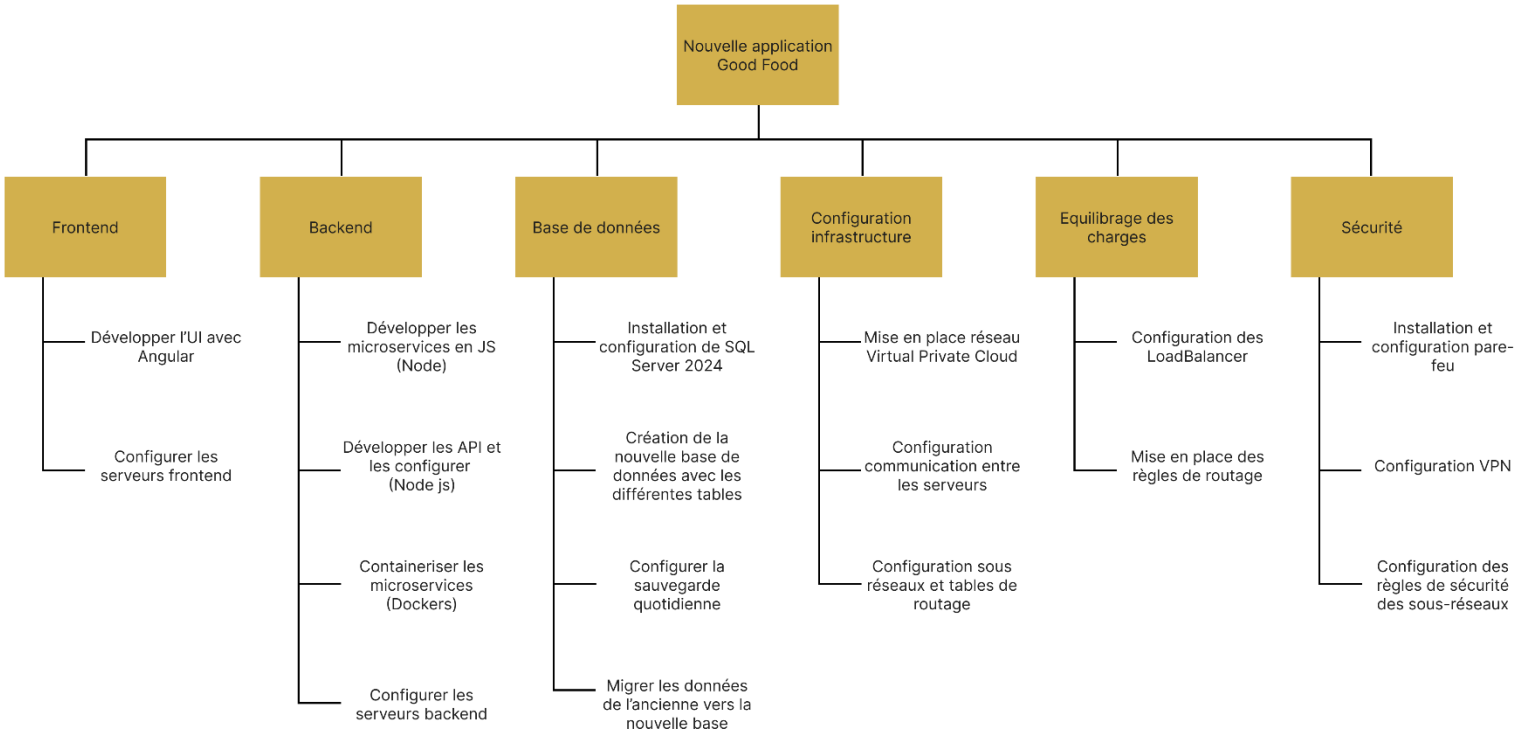
L'utilisation extensive de pare-feu et de protocoles sécurisés comme HTTPS et VPN.
Les équilibrages de charge sont utilisés non seulement pour la distribution du trafic mais aussi pour contribuer à la stratégie de sécurité globale en réduisant les points de défaillance uniques.

11. Indicateurs de performance (KPI)

Indicateur	Objectif	Métrique
Disponibilité	95%	Temps d'arrêt de l'application par mois
Temps de réponse	Inférieur à 3 secondes	Temps de réponse moyen
Sécurité	Inférieur à 2	Nombre de failles de sécurité découvertes
Maintenabilité	Inférieur à 48h	Temps moyen de résolution des problèmes
Satisfaction	85%	Note de satisfaction client
Accessibilité	95%	Conformité aux normes d'accessibilité
Évolutivité	5 par an	Nombre de nouvelles fonctionnalités ajoutées

12. Stratégie de développement des applicatifs de l'entreprise

a. Découpage des tâches



b. Propositions d'amélioration du SI

Bien que le système d'information actuel de Good Food soit performant, il peut être améliorer et ce en plusieurs points.

Tout d'abord, la mise en place d'une démarche DevOps. Pour intégrer cette dernière dans le système d'information, il est essentiel de favoriser la collaboration entre les équipes de développement et d'opérations. Cela implique l'automatisation des processus de déploiement, de tests et de gestion des infrastructures, ainsi que l'adoption d'outils de monitoring et de suivi des performances pour garantir un déploiement continu et une livraison rapide et fiable des services.

Ensuite, la mise en place d'un système de supervision du parc applicatif. Cela permettra de surveiller en temps réel la santé et les performances des différents services. Cela implique l'utilisation d'outils de monitoring et de logging comme Prometheus pour collecter et analyser les données, ainsi que la mise en place de mécanismes d'alerte pour détecter et résoudre rapidement les incidents et les anomalies, assurant ainsi la disponibilité et la fiabilité des applications.

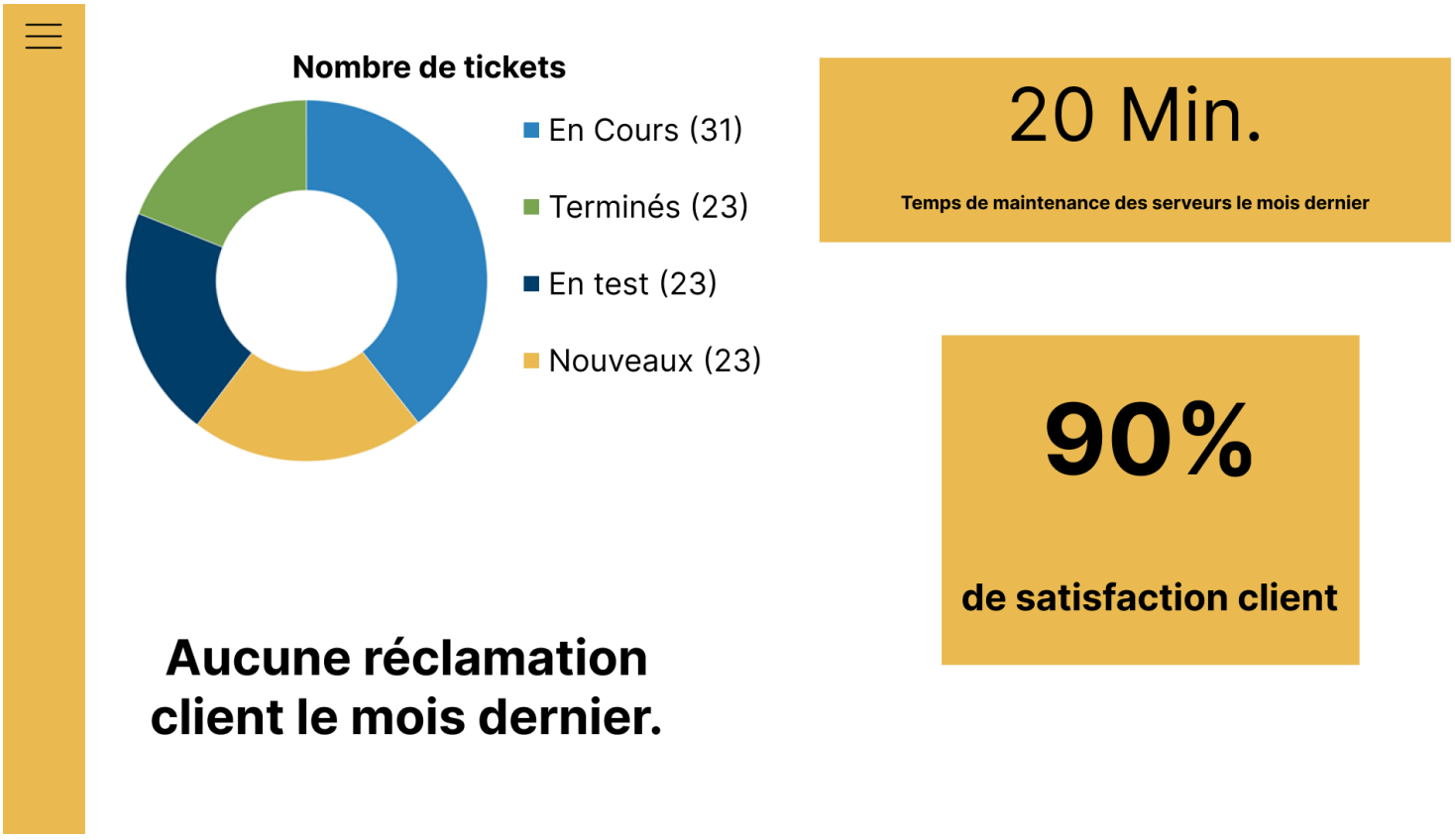
Aussi, mettre en place une solution de qualité de code telle que SonarQube. Pour garantir la qualité du code dans un système d'information, il est nécessaire de mettre en place des pratiques de développement telles que la revue de code, les tests unitaires et l'automatisation des tests

d'intégration continue. De plus, l'utilisation d'outils d'analyse statique et dynamique du code peut aider à identifier et à corriger les erreurs et les vulnérabilités, contribuant ainsi à améliorer la robustesse et la sécurité des applications.

Mais également en appliquant un référentiel comme ITIL. C'est utile pour répondre aux besoins spécifiques de l'architecture distribuée. Cela implique d'identifier les processus et les pratiques clés qui soutiennent la gestion des services, le développement logiciel et l'architecture d'entreprise, tout en intégrant des approches agiles et DevOps pour favoriser l'innovation et la flexibilité.

Et enfin, la mise en place d'un outil de gestion des incidents comme JIRA Service Management. Cela permet de centraliser et de suivre les incidents et les problèmes rencontrés dans les différents services. Cela implique la création d'un processus clair de gestion des incidents, incluant la notification, l'escalade et la résolution des problèmes, ainsi que la documentation et l'apprentissage des incidents pour prévenir les récurrences. L'outil doit également permettre une analyse approfondie des incidents pour identifier les tendances et les opportunités d'amélioration continue.

c. Indicateurs visuels



Les différents indicateurs présents sont :

- Le nombre de tickets avec leur statut, pour garder une vision d'ensemble de l'avancement des différents développements.
- Le temps de maintenance des serveurs le mois dernier, pour pouvoir mesurer la disponibilité du service mensuellement.

- Le pourcentage de satisfaction client, qui est une des métriques les plus importantes pour une entreprise compétitive comme Good Food
- Le nombre de réclamation réalisé par les clients le mois dernier, pour garder une trace de l'évolution de l'application et des habitudes de consommation des clients.

13. Bibliographie

Innowise, Article de blog, Comparatif de différentes architectures possibles :
https://innowise.com/fr/blog/best-software-architecture-patterns/

Mobiskill, Article de blog, Comparatif de différents framework front-end :
https://mobiskill.fr/blog/conseils-emploi-tech/les-meilleurs-frameworks-front-end-a-utiliser-en-2021/

Appmaster.io, Article de blog, Comparatif de différents framework back-end :
https://appmaster.io/fr/blog/10-meilleurs-frameworks-web-backend

AquaRay, Article, Comment dimensionner une plateforme informatque :
https://www.aquaray.com/blog/articles/comment-dimensionner-une-plateforme-informatique

PMorisseau, Article de blog, les facteurs clés de la mise en place d'une démarche DevOps :
https://pmorisseau.ineaweb.net/articles/devops/vrille/02.pourquoi.devops/

Motilde, Article, Supervision du parc informatique avantages et étapes :
https://motilde.com/la-supervision-du-parc-informatique-un-levier-pour-la-securite-et-la-continuite-des-operations/