NOM	SAINSON
Prénom	Jocelyn
Date de naissance	27/02/2001

Copie à rendre

Bloc 1 – Conception et spécification d'une solution digitale

Documents à compléter et à rendre

Dossier 1 : Analyse du besoin

Toutes réponse est accompagnée d'un lien vers le ticket jira correspondant. Des éléments supplémentaires peuvent y être ajoutés (sources, annexes ou autre).
N'utilisez que ces identifiants pour y accéder (connexion via Google OAuth, ne pas utiliser la connexion via credentials) :
Login -> studicorrecteur@gmail.com
Password -> CorrecteurStudi2026!
N'hésitez pas à les consulter. Ici on traite le bloc 1

1. Énumérez toutes les fonctionnalités attendues par le client

L'objectif de la mission est de remplacer le standard téléphonique par un site web avec les fonctionnalités demandées par le client : (ticket jira)

1. Côté client :
    1. Accéder à une page simple et moderne
    2. Traduction des éléments pour l'internationalisation de l'événement
    3. Page de sélection d'offre :
        1. Offre simple (1 ticket)
        2. Offre duo (2 tickets)
        3. Offre famille (4 tickets)
    4. Gestion de formulaires d'inscription et de connexion :
        1. Couple login / mot de passe
        2. Génération d'une secret key liée au user
        3. Multi Factor Authentification (MFA) pour sécuriser
    5. Gestion du panier :
        1. Ajout et suppression d'offres
        2. Passage au paiement
    6. Génération du QR code ou code barre
    7. Génération du PDF
    8. Envoi par mail
    9. Génération de la clé pour chaque tickets
    10. Assemblage des 2 clés pour unicité
2. Côté JO :
    1. Vérification des clés
    2. Vérification de l'identité de la personne

2. Le projet vous paraît-il faisable compte tenu de l'aspect juridique ? Donnez votre avis sur les données collectées disponibles. (Ticket jira)

Le projet est faisable juridiquement en conformité avec la RGPD. Cependant il y'a quelques points à prendre en compte pour la faisabilité et la conformité. Au niveau des données qui seront récupérées il y'aura :

- Nom / Prénom
- Email / Mot de passe
- Les informations de paiement (non stockées)
- Les clés uniques générées (pour identifier le client et ses tickets)

Il faudra donc que je mette en place plusieurs points d'attention pour être en accord avec les lois européennes et françaises :

- Utiliser des cookies avec demande d'accord du user
- Informer le client de chaque donnée qui sera récupérée
- Expliquer pourquoi
- Détailler comment elles seront utilisées
- Demander l'accord explicite
- Faire une demande d'accord séparée pour la partie marketing, newsletter etc
- Prévenir de la durée de conservation des données
- S'assurer de la suppression à la fin de leur durée légale de conservation
- Donner le droit au user de lire / modifier / supprimer ses données
- Informer qu'il y'aura des clés générées sans qu'il puisse les voir et demander son accord
- Rédiger les Conditions Générales de Vente (CGV) en accord avec tout ce qui a été dit précédemment

Pour rester conforme, voici ce qu'il ne faut pas faire :

- Revendre les informations
- Les utiliser à d'autres fins que celles énoncées

3. Effectuez un état de l'art des outils existant permettant de vous aider à la mise en place de votre solution et expliquez en quoi ils sont pertinents. (Ticket jira)

Pour m'aider dans le développement de cette application j'ai pu recenser plusieurs outils / framework existants. Voyons ça en découpant en plusieurs catégories.
Dans un premier temps il y'a toute ce qui va être lié à l'authentification et la sécurité :

- Oauth 2.0 : pour déléguer la gestion de l'authentification à une entité sécurisée et ne gérer que l'autorisation
- JWT (json web token) : qui nous aidera à la création des tokens sécurisés lors de l'achat et de la création du compte
- Bcrypt : pour la sécurisation des mots de passes en base de données et offrir 2 options d'authentification aux users
- Spring security : qui est un tout en un pour intégrer la gestion sécurisée de l'api et autres points

Ensuite il y'a la nécessité de générer un QR code pouvant être lu par le staff :

- ZXing : pour générer côté backend un QR code qui représentera les données et clés
- QRCode : lui, permet la gestion côté front

Pour la partie paiement il y'a plusieurs choix qui s'offrent à nous :

- Paypal : solution internationale et largement intégrée qui offre une option simple à tous
- Stripe : option sécurisée et simple qui prend en charge beaucoup de moyens de paiements

Pour la partie back-end, il y'a énormément de possibilités. Ici j'ai choisi d'utiliser Java avec le SDK 21 qui est une option robuste et qui à fait ses preuves dans toutes les applications d'entreprise. Il sera adapté car il possède de nombreuses intégrations et dépendences pour avoir un projet évolutif en cas d'ajout de features.
Pour le front-end, le choix s'est porté sur React pour sa forte communauté ainsi que ses server side rendering (SSG) qui prouvent sa puissance. Ici, il nous permettra d'avoir une interface moderne et réactive pour les clients. React email , qui servira à l'envoi de mail via Resend comme la confirmation d'identité, changement mot de passe, etc. Pour la partie administration côté staff, le choix s'est porté sur React Admin qui permet une interconnexion avec les datas côté client.
En terme de base de données et de caching pour la performance de l'application et pouvoir supporter le flux important :

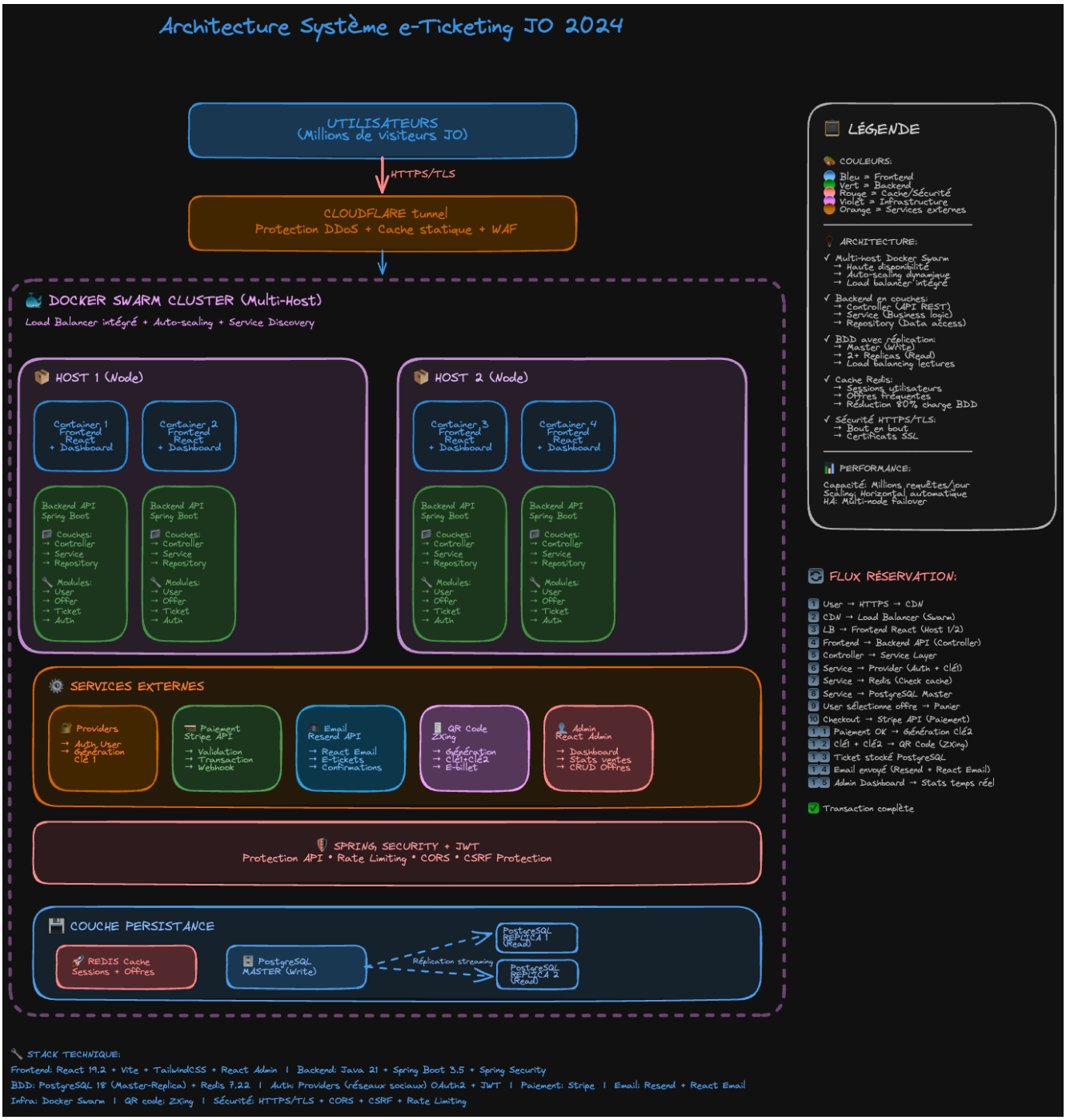
- PostgreSQL qui offre une solution robuste, capable de gérer efficacement les lectures écritures simultanées, tout en garantissant la conformité ACID et les types de données avancées.
- Redis qui permettra d'intégrer du caching de données pour améliorer encore les performances et maximiser les requêtes.

Dans le cadre de l'hébergement, le choix s'est porté sur Docker Swarm car il permettra d'implémenter du load balancing entre plusieurs services pour ne jamais saturer et augmenter en cas de forte demande et inversement. Grâce à ça, il sera aussi possible de faire du multi-host ce qui couvre les cas de down d'un server.

Dossier 2 : Conception de la nouvelle architecture

1. Concevez une architecture logicielle répondant aux besoins. N'oubliez pas que les Jeux olympiques reçoivent des millions d'appels par jours (ticket jira)

Voici l'architecture imaginée pour ce projet et son acceptation de la charge. Vous retrouverez dans le ticket jira le lien vers excalidraw pour une lecture plus simplifiée ainsi que le fichier .excalidraw si vous disposez d'un accès au logiciel.



2. Justifiez tous les choix technologiques de votre architecture logicielle. (Ticket jira)

Pour répondre aux besoins du client, j'ai pris le parti de choisir ces différents outils qui nous permettront de rendre l'architecture la plus évolutive possible selon le besoin et de gérer dès le départ un grand nombre de requêtes simultanées sans crash :

1. Infrastructure & Orchestration

Docker :

- Utilité générale : Conteneurisation des applications pour garantir la portabilité et l'isolation
- Dans ce projet : Empaqueter chaque service (backend, base de données, Redis, etc.) dans des conteneurs standardisés, facilitant le déploiement et la scalabilité

Docker Swarm :

- Utilité générale : Orchestration de conteneurs, load balancing automatique, haute disponibilité
- Dans ce projet :
  - Gérer les millions d'appels par jour via le load balancing intégré
  - Répartir automatiquement la charge entre plusieurs instances du backend
  - Auto-scaling en cas de pics de trafic (ouverture des ventes)
  - Haute disponibilité : si un nœud tombe, les conteneurs sont redémarrés ailleurs

Github Container Registry :

- Utilité générale : Registry Docker hébergé par GitHub permettant le stockage, le versioning et la distribution des images de conteneurs.
- Dans ce projet :
  - Les images buildées une fois sont distribuées rapidement vers tous les nœuds Docker Swarm via docker pull, évitant le build coûteux en ressources sur les serveurs de production.
  - Images privées par défaut, scan automatique des vulnérabilités, authentification via tokens, audit log des accès. Critique pour un événement sensible comme les JO.

2. Réseau & Sécurité

Cloudflare Tunnel :

- Utilité générale : Tunnel sécurisé, protection DDoS, CDN, certificats TLS automatiques
- Dans ce projet :
  - Sécurité TLS/HTTPS : Chiffrement automatique des communications (exigence client sur la sécurité)
  - Protection DDoS : Protection contre les attaques lors des pics de vente
  - WAF (Web Application Firewall) : Filtrage des requêtes malveillantes
  - CDN : Mise en cache des ressources statiques (images, CSS, JS) pour accélérer le chargement

3. Base de Données & Stockage

PostgreSQL :

- Utilité générale : Base de données relationnelle robuste, conformité ACID, performante
- Dans ce projet :
  - Relations complexes : liaison entre utilisateurs, clés de sécurité, tickets, achats
  - Transactions ACID : Garantir la cohérence lors des paiements (pas de double facturation)
  - Gestion des clés de sécurité : Stockage sécurisé des deux clés (création compte + achat)
  - Conformité RGPD : Gestion des données personnelles (nom, prénom, email)

pgAdmin :

- Utilité générale : Interface graphique pour administrer PostgreSQL
- Dans ce projet :
  - Gestion des backups
  - Debug en développement

pgBackRest :

- Utilité générale : Solution de sauvegarde et restauration pour PostgreSQL
- Dans ce projet :
  - Sauvegardes automatisées : Prévenir la perte de données critiques (billets, paiements)
  - Restauration rapide : En cas de corruption ou d'incident
  - Exigence métier : Les JO sont un événement unique, aucune perte de donnée acceptable

4. Cache & File d'attente

Redis

- Utilité générale : Base de données en mémoire, cache haute performance, pub/sub messaging
- Dans ce projet (triple usage) :
  - Cache :
    - Mise en cache des offres (solo, duo, familiale) pour réduire la charge sur PostgreSQL
    - Réponse aux millions d'appels/jour : réduction de 70-80% des requêtes BDD
  - Pub/Sub (Queue) :
    - File d'attente asynchrone pour l'envoi des emails (via Resend)
    - Génération asynchrone des QR codes
    - Décharge le backend principal, améliore le temps de réponse utilisateur
  - Rate limiting : Limiter les tentatives de connexion (sécurité contre le brute force)

5. Observabilité & Monitoring

Sentry

- Utilité générale : Monitoring des erreurs applicatives et des performances
- Dans ce projet :
  - Capture des exceptions en temps réel (erreurs de paiement, génération de tickets)
  - Traces de performance : Identifier les endpoints lents
  - Alertes : Notification immédiate en cas de problème critique
  - Contexte détaillé : Stack traces, environnement, utilisateur concerné

Prometheus

- Utilité générale : Collecte et stockage de métriques système et applicatives en time-series
- Dans ce projet :
  - Monitoring des ressources système : CPU, RAM, disque, réseau
  - Métriques applicatives :
    - Nombre de requêtes/seconde
    - Temps de réponse moyen
    - Taux d'erreur HTTP
    - Nombre de tickets générés
    - État de la queue Redis
  - Métriques BDD : Connexions actives, requêtes lentes, locks
  - Exigence projet : Anticiper les problèmes lors des pics de charge (millions d'appels)

Grafana

- Utilité générale : Plateforme de visualisation de données et dashboards
- Dans ce projet :
  - Dashboards en temps réel :
    - Tableau de bord "Santé système" : CPU, RAM, latence
    - Dashboard "Métier" : Ventes par offre, revenus, utilisateurs actifs
    - Dashboard "Sécurité" : Tentatives de connexion échouées, alertes WAF
  - Interface unifiée pour visualiser Prometheus + Loki
  - Alerting visuel : Seuils configurables (ex: alerte si temps réponse > 2s)

Loki + Promtail

- Utilité générale : Stack légère pour la collecte et l'indexation des logs
- Dans ce projet :
  - Logs centralisés : Agréger les logs de tous les conteneurs Docker
  - Logs métier :
    - Audit trail : "Utilisateur X a acheté le ticket Y à l'heure Z"
    - Historique des actions admin (création/modification d'offres)
    - Logs de sécurité : Connexions, déconnexions, échecs d'authentification
  - Recherche et filtrage : Retrouver rapidement les logs d'un utilisateur ou d'une transaction
  - Conformité RGPD : Traçabilité des accès aux données personnelles
  - Intégration Grafana : Visualisation dans la même interface que les métriques

6. CI/CD & Versionning

GitHub

- Utilité générale : Gestion de version, collaboration, code review
- Dans ce projet :
  - Versionning du code : Backend Java, templates React Email, configurations Docker
  - Branches : Stratégie Git Flow (dev, staging, production)
  - Code review : Pull requests avec validation avant merge
  - Backup du code : Historique complet

GitHub Actions

- Utilité générale : Automatisation CI/CD intégrée à GitHub
- Dans ce projet :
  - Pipeline CI :
    - Build automatique du code Java
    - Exécution des tests unitaires et d'intégration
    - Analyse de code avec SonarQube
    - Scan de vulnérabilités (OWASP Dependency Check)
  - Pipeline CD :
    - Build des images Docker
    - Push vers Github Container Registry
    - Déploiement automatique sur Docker Swarm

7. Envoi d'Emails

React Email

- Utilité générale : Framework pour créer des templates d'emails modernes avec React
- Dans ce projet :
  - Templates professionnels : Design responsive et compatible tous clients email (Gmail, Outlook, Apple Mail)
  - Composants réutilisables : Header, Footer, Button, QR Code section
  - Dark mode automatique : Adaptation au thème de l'utilisateur
  - Preview en développement : Visualisation en temps réel des modifications
  - Emails critiques :
    - Email de confirmation d'achat avec QR code
    - Email de bienvenue après création de compte
    - Email de réinitialisation de mot de passe

Resend

- Utilité générale : Service d'envoi d'emails transactionnels moderne
- Dans ce projet :
  - Délivrabilité optimale : Infrastructure optimisée, réputation IP gérée
  - Rate limits élevés : Gestion des pics d'envoi (milliers d'achats simultanés)
  - Analytics : Taux d'ouverture, taux de clic, bounces
  - API simple : Intégration facile avec le backend Java
  - Gestion des échecs : Retry automatique, webhooks pour les notifications
  - Conformité : Gestion automatique des unsubscribe, DKIM, SPF

8. Qualité & Sécurité du Code

SonarQube

- Utilité générale : Analyse statique du code, détection de bugs et vulnérabilités
- Dans ce projet :
  - Qualité du code : Détection de code smell, duplication, complexité
  - Sécurité : Identification de vulnérabilités (injection SQL, XSS, etc.)
  - Coverage : Vérification du taux de couverture des tests
  - Exigence projet : Sécurité critique mentionnée par le client

OWASP Dependency Check

- Utilité générale : Scanner de vulnérabilités dans les dépendances
- Dans ce projet :
  - Scan des libraries Java : Détection de CVE dans les dépendances Maven
  - Alertes automatiques : Notification si une dépendance est compromise
  - Sécurité proactive : Anticiper les failles avant la production

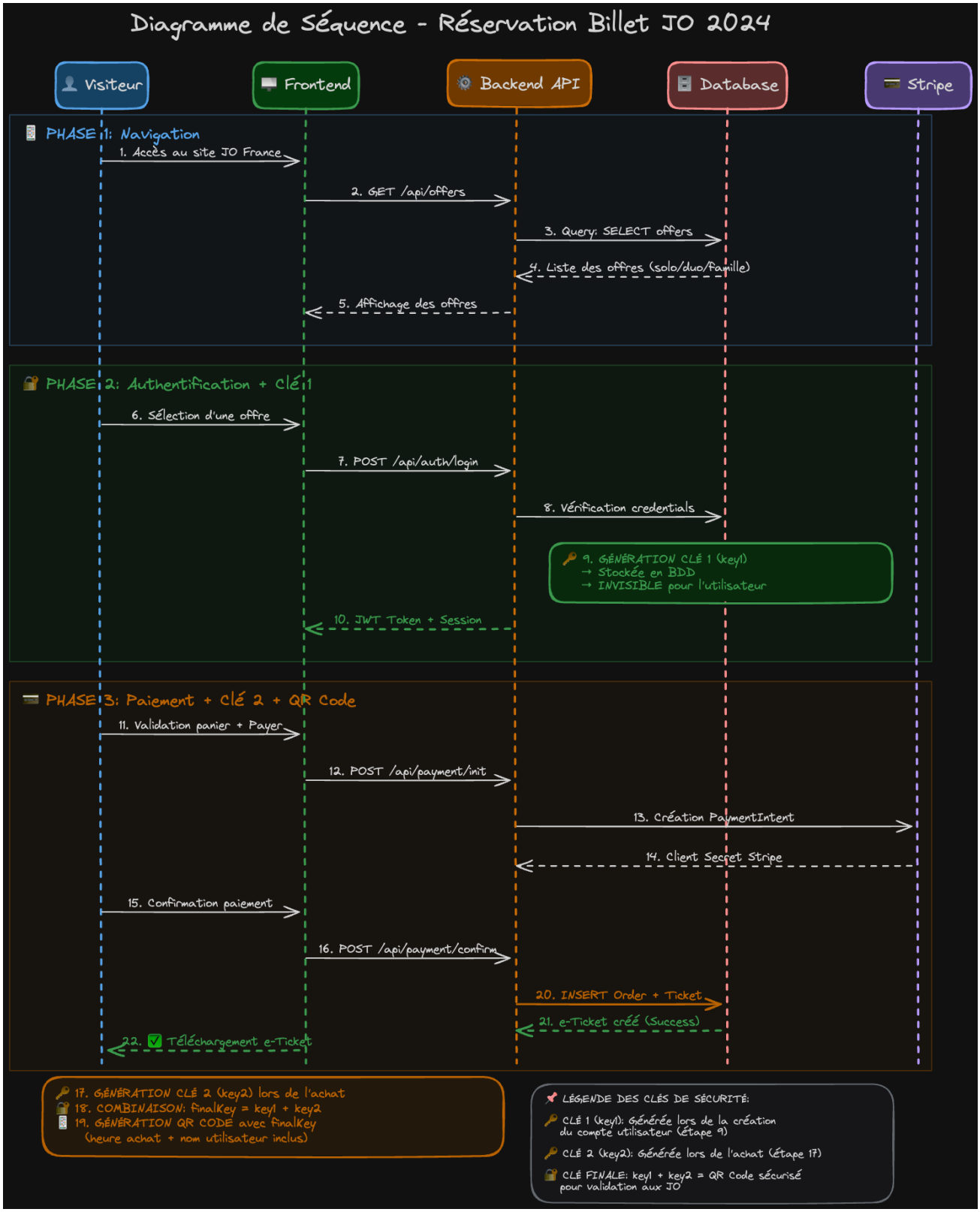
9. Gestion des Secrets

Docker Secrets

- Utilité générale : Stockage sécurisé de données sensibles dans Docker Swarm
- Dans ce projet :
  - Secrets gérés :
    - Clés API (Resend, Cloudflare)
    - Credentials PostgreSQL
    - JWT secret key
    - Clés de chiffrement pour les tickets
  - Sécurité : Secrets chiffrés au repos et en transit
  - Pas de secrets dans le code : Conformité aux bonnes pratiques
  - Rotation facilitée : Mise à jour sans rebuild des images

3. Effectuez un diagramme de séquence sur la fonctionnalité de réservation de billet. (Ticket Jira)

Le fichier .excalidraw ainsi que le .png seront liés au ticket jira pour plus de lisibilité.



4. Comment allez-vous sécuriser votre application sur tous les points mentionnés par le client ? (Ticket Jira)

Pour la sécurité de l'application globale ainsi que sur les points précis mentionnés par le client, les éléments à mettre en place seront les suivants :

1. Comptes users

Authentification renforcée :

- Hachage des mots de passe : Utilisation de BCrypt avec un salt aléatoire (work factor 12) pour stocker les mots de passe
- Politique de mots de passe forte :
  - Minimum 12 caractères
  - Combinaison obligatoire : majuscules, minuscules, chiffres, caractères spéciaux
- MFA (Multi-Factor Authentication) :
  - TOTP (Time-based One-Time Password) via Google Authenticator ou autre
  - Envoi de code par email via Resend comme fallback
- Rate Limiting :
  - Via Redis : maximum 5 tentatives de connexion par IP/compte sur 15 minutes
  - Blocage temporaire de 30 minutes après 5 échecs

Sessions

- JWT :
  - Tokens signés avec algorithme HS256
  - Durée de vie courte : 15 minutes pour l'access token
  - Refresh token : 7 jours, stocké en base et révocable
  - Claims minimaux : userId, rôles, expiration, création
- Rotation des tokens : Refresh automatique avant expiration
- Invalidation : Blacklist Redis pour déconnexion immédiate

2. Système de Double Clé

Clé 1 - Génération à la création du compte :

- Algorithme : UUID + hash SHA-256 du timestamp + userId
- Stockage : Chiffrée en base avec AES-256-GCM
- Visibilité : Uniquement accessible par les services backend et l'organisation JO

Clé 2 - Génération à l'achat :

- Algorithme : UUID + hash SHA-256 du timestamp achat + orderId + ticketId
- Stockage : Chiffrée en base avec AES-256-GCM, liée à la transaction

Clé Finale (Combinaison) :

- Cette clé finale est encodée dans le QR Code
- Format QR Code : clé finale + metadata chiffrées en base 64

3. Sécurisation du E-Ticket

QR Code sécurisé :

- Signature numérique : HMAC-SHA256 avec clé secrète côté serveur
- One-time scan : Le QR code est invalidé après premier scan réussi

Génération PDF :

- Bibliothèque : iText 7 avec signature numérique PDF
- Protection :
  - PDF signé numériquement avec certificat
  - Interdiction de modification (permissions PDF)

4. Vérification à l'Entrée

Scan et Validation :

- Application Staff :
  3. Scan du QR Code
  4. Déchiffrement et extraction des données
  5. Vérification de la signature HMAC
  6. Appel API backend : POST /api/verify-data

Vérification d'identité :

- Affichage sur tablette staff : Nom + Prénom du titulaire
- Comparaison visuelle avec pièce d'identité

5. Sécurité Applicative Globale

Backend (Spring Boot) :

- Spring Security :
  - CORS configuré strictement (whitelist des domaines autorisés)
  - CSRF protection via tokens
  - Helmet headers : X-Frame-Options, X-Content-Type-Options, CSP
- Input Validation :
  - Validation sur tous les DTO
  - Nettoyage des inputs pour injections XSS/SQL
- API Rate Limiting :
  - Bucket4j + Redis : 100 requêtes/minute/IP
  - Endpoints sensibles (login, payment) : 10/minute

Frontend (React) :

- HTTPS Only : Redirection automatique via Cloudflare
- Secure Cookies :
  - Flags : HttpOnly, Secure, SameSite=Strict
  - Pas de stockage de tokens en localStorage (XSS risk)

Infrastructure :

- Cloudflare WAF :
  - Règles OWASP Core Rule Set
  - Protection DDoS Layer 3/4/7
  - Rate limiting géographique
- Docker Secrets : Toutes les clés sensibles stockées en Docker Secrets
- TLS 1.3 : Chiffrement de bout en bout
- Isolation réseau :
  - Réseau Docker overlay pour isolation des services
  - Base de données non exposée publiquement