iSMAGi
Institut Supérieur de Management
d'Administration et de Génie Informatique
المعهد العالي للتدبير و الإدارة و الهندسة المعلوماتية

Rapport de Projet de

Fin de Module :

Système distribué de gestion de réservation de billets de vols aériens

FlyR

Réalisé et fait par :

Chalabi Nada , Ivora Only, Khtou Younes et Boukhris Hamza

Année universitaire: 2025/2026

Résumé (Abstract)

Le projet FlyR est une application distribuée de réservation de vols basée sur une architecture client-serveur. Elle permet la recherche et la consultation de vols, la gestion des sièges disponibles, ainsi que la réservation, l'annulation et la modification de réservations. L'architecture combine plusieurs technologies adaptées à différents types d'opérations : **REST (JAX-RS)** pour les requêtes fréquentes (consultation/recherche), **SOAP (JAX-WS)** pour les opérations critiques et transactionnelles (réservation/modification/annulation/paiement simulé), **RMI** pour la communication interne entre services serveur (gestion centralisée des vols, vérification disponibilité, synchronisation), et **sockets TCP** pour les notifications en temps réel (confirmation, conflit, mise à jour des disponibilités). Les données sont persistées dans une base relationnelle **MySQL**, avec utilisation de transactions pour assurer la cohérence.

Table de matières

1. Introduction

Dans le domaine de la réservation aérienne, plusieurs utilisateurs consultent simultanément les mêmes vols et tentent parfois de réserver les mêmes sièges. Cela crée des défis typiques des systèmes distribués :

- **Accès concurrent** à des ressources partagées (sièges),
- nécessité d'une **cohérence forte** (pas de double réservation),
- besoin de **mise à jour rapide** des disponibilités,
- gestion propre des **erreurs et conflits**.

FlyR est conçu pour illustrer ces problématiques en séparant clairement les responsabilités entre différents mécanismes de communication distribuée (REST, SOAP, RMI, sockets), tout en conservant une base de données relationnelle garantissant la persistance et l'intégrité.

2. Objectifs et exigences

2.1 Fonctionnalités côté utilisateur

- Recherche de vols selon critères (départ, destination, date, aller simple / aller-retour, nombre de passagers).
- Consultation des informations d'un vol (numéro, horaire, avion associé).
- Consultation des sièges et de leur disponibilité.
- Réservation d'un siège (opération critique).
- Modification d'une réservation (changement de siège).
- Annulation d'une réservation.
- Consultation de l'historique personnel.

2.2 Fonctionnalités côté administrateur

- Gestion des vols (ajout, modification, suppression).
- Gestion des avions (capacité, modèle) et des sièges (classe, numérotation).
- Consultation globale des réservations.
- Génération de rapports (taux de remplissage, vols les plus demandés, etc.).

2.3 Exigences non fonctionnelles

- Cohérence des données via **transactions SQL**.
- Communication distribuée adaptée :
    - REST pour les lectures fréquentes,
    - SOAP pour opérations transactionnelles,
    - RMI pour communications internes,
    - Sockets pour notifications en temps réel.
- Extensibilité (architecture modulaire).

3. Architecture générale (vue distribuée)

3.1 Composants du système

FlyR est organisé autour de plusieurs composants :

(A) Client Java Swing (MVVM)

- Interface graphique moderne (Swing).
- Pattern MVVM (View ↔ ViewModel) avec data binding (Property<T>) et commandes (RelayCommand).
- Consommation REST pour la recherche/consultation.
- Consommation SOAP pour les opérations de réservation/annulation/modification.
- Connexion socket TCP pour recevoir les notifications temps réel.

(B) Serveur REST (JAX-RS)

- Expose des endpoints REST pour recherche, consultation et historique.
- Utilise une couche repository pour accéder à MySQL.
- Utilise un pool de connexions JDBC (composant maison) pour performance et stabilité.

(C) Serveur SOAP (JAX-WS)

- Expose un service de booking avec des méthodes transactionnelles.
- Gère les conflits et retourne des erreurs métier sous forme de **SOAP Faults**.

(D) Serveur RMI (interne)

- Communication “serveur↔serveur” uniquement.
- Services internes (gestion vols, disponibilité sièges, synchronisation).

(E) Serveur sockets TCP

- Envoi d’événements temps réel vers les clients connectés (confirmation, conflit, mise à jour disponibilité).

(F) Base de données MySQL

- Persistance des entités principales : vols, avions, sièges, clients, réservations.
- Garantit l'intégrité relationnelle (clés étrangères) et l'atomicité via transactions.

3.2 Diagramme d'architecture

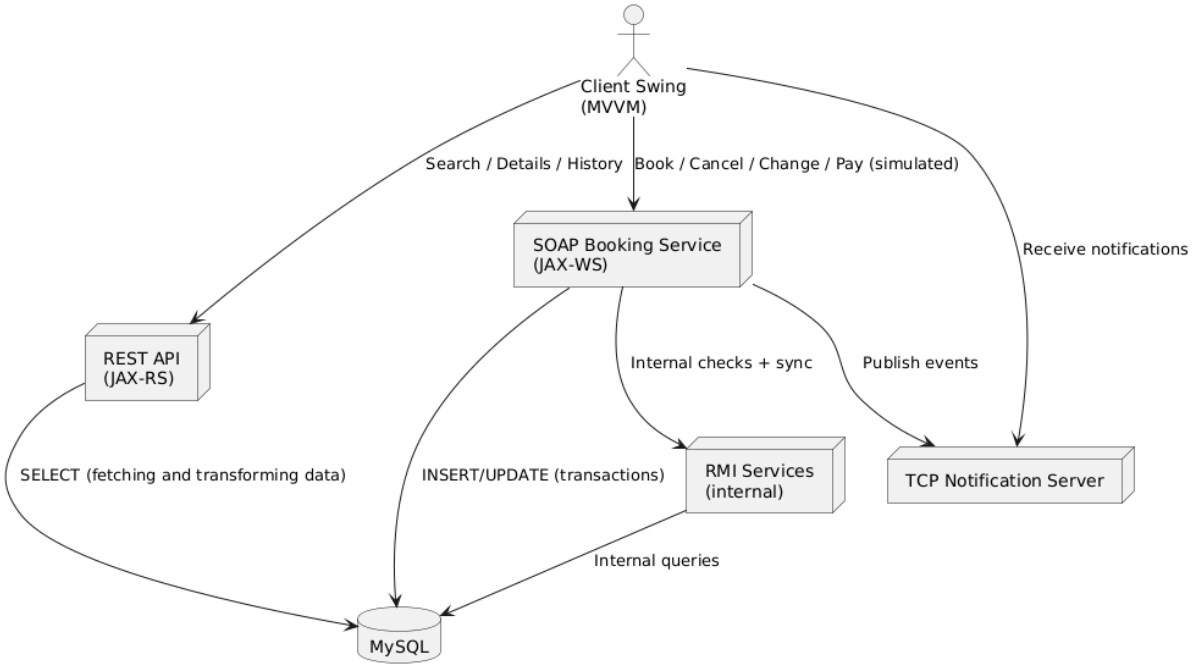

Figure 1 Diagramme d'architecture

Ce système repose sur un client lourd Java Swing (utilisant le pattern MVVM) qui communique avec un backend distribué via trois protocoles distincts. Pour la consultation de données (recherche, historique), le client interroge une API REST (JAX-RS) qui effectue des lectures (SELECT) et des transformations de données depuis une base MySQL.

Pour les opérations métier transactionnelles comme la réservation ou le paiement, le client sollicite un service SOAP (JAX-WS). Ce dernier assure la cohérence des données en effectuant des mises à jour (INSERT/UPDATE) dans la base de données, tout en s'appuyant sur des services RMI internes pour les vérifications de synchronisation.

Enfin, l'aspect temps réel est géré par un serveur de notifications TCP. Lorsqu'une action est validée par le service SOAP, un événement est publié vers ce serveur, qui pousse ensuite une notification directement vers le client Swing pour le tenir informé des changements d'état du système.

4. Modèle de données MySQL

4.1 Entités stockées

- **Aircraft (aircraft)** : code, modèle, capacité totale, capacité économique, capacité business.
- **Flight (flight)** : numéro, villes, date/heure, avion associé.
- **Client (client)** : informations personnelles (email unique, passeport unique).
- **Seat (seat)** : siège (numéro, classe), lié à un avion.
- **Reservation (reservation)** : association client + vol + siège + statut.

4.2 Statuts métier (enums)

- ReservationStatus : CONFIRMED, CANCELLED
- SeatClass : ECONOMY, BUSINESS
- SeatStatus : FREE, RESERVED
- PaymentStatus : PENDING, SUCCESS, FAILED

4.3 Remarque sur la disponibilité

Le champ seat.seat_status est un statut global au niveau du siège de l'avion.
Dans FlyR, la disponibilité "par vol" est principalement déduite via la table reservation :

- Un siège est **réservé sur un vol** s'il existe une réservation CONFIRMED pour ce couple (flight_id, seat_id).

5. REST (JAX-RS)

5.1 Rôle de REST

REST est utilisé pour les opérations simples, très fréquentes et orientées lecture :

- recherche de vols,
- consultation des détails,
- consultation de l'historique.

5.2 Exemple : recherche de vols

```
@GET
@Path("/search")
public Response searchFlights(@QueryParam("departure") String departureCity,
                              @QueryParam("arrival") String arrivalCity,
                              @QueryParam("from") String fromDate,
                              @QueryParam("passengers") int passengerCount) {
  List<Flight> flights = repository.searchFlights(departureCity, arrivalCity, fromDate,
passengerCount);
  return Response.ok(flights).build();
}
```

5.3 Accès BD et pooling

La couche repository accède à MySQL via JDBC, avec pool de connexions :

- meilleure performance,
- limitation du coût de création de connexions,
- comportement plus stable lorsque plusieurs requêtes arrivent en même temps.

6. SOAP (JAX-WS)

6.1 Pourquoi SOAP ?

Les opérations critiques modifient l'état global (réservations) et nécessitent :

- transactions,
- gestion d'erreurs métier robuste,
- contrat clair entre client et serveur.

SOAP répond bien à ces besoins grâce au WSDL et aux faults.

6.2 Méthodes SOAP implémentées

- Reservation bookSeat(int clientId, int flightId, int seatId)
- Reservation cancelReservation(int reservationId)
- Reservation changeSeat(int reservationId, int newSeatId)
- PaymentTransaction simulatePayment(int reservationId, BigDecimal amount, String currency)

6.3 Sérialisation des entités (JAXB)

Le service SOAP retourne des **entités** (pas des DTO). Pour cela :

- ajout d'annotations JAXB (@XmlRootElement, @XmlAccessorType(XmlAccessType.FIELD))
- conversion des dates Java modernes via **XmlAdapters** :
    - LocalDateAdapter (LocalDate ↔ String ISO)
    - LocalDateTimeAdapter (LocalDateTime ↔ String ISO)

6.4 Gestion d'erreurs métier via SOAP Fault

Les erreurs sont renvoyées sous forme de fault custom :

- BookingException (annotée @WebFault)
- BookingFault contenant un code + message

Exemples de codes :

- SEAT_ALREADY_RESERVED

- RESERVATION_NOT_FOUND
- RESERVATION_CANCELLED
- INTERNAL_ERROR

6.5 Logique transactionnelle (explication simple)

Chaque opération critique suit le même principe :

1. ouvrir une connexion JDBC
2. démarrer une transaction (setAutoCommit(false))
3. exécuter les vérifications nécessaires
4. modifier la base (INSERT/UPDATE)
5. commit
6. retourner une entité reconstruite depuis la base

6.5.1 Réserver un siège : bookSeat

But : créer une réservation CONFIRMED si le siège est libre sur ce vol.

Pseudo-logique :

```
BEGIN
si réservation CONFIRMED existe déjà pour (flightId, seatId)
   ROLLBACK + fault "SEAT_ALREADY_RESERVED"
sinon
   INSERT reservation(CONFIRMED)
   COMMIT
   retourner Reservation complète
```

6.5.2 Annuler : cancelReservation

- verrouille la réservation (lecture FOR UPDATE)
- si pas trouvée → fault
- sinon : update statut à CANCELLED
- commit + retour

6.5.3 Modifier : changeSeat

- vérifie que la réservation n’est pas annulée
- vérifie que le nouveau siège n’est pas déjà confirmé sur le vol
- update seat_id
- commit + retour

6.5.4 Paiement simulé : simulatePayment

- vérifie que la réservation existe et n’est pas annulée
- retourne un PaymentTransaction simulé (référence de paiement générée)

7. RMI

7.1 Objectif (conforme au sujet)

RMI est utilisé en interne pour séparer les responsabilités entre composants serveur :

- **Gestion centralisée des vols** : CRUD et consultation
- **Vérification disponibilité sièges** : service interne commun
- **Synchronisation des réservations** : propagation d'événements

7.2 Services RMI

7.2.1 FlightManagementRmi

- gestion des vols : création, modification, suppression, liste, détails

7.2.2 SeatAvailabilityRmi

- isSeatAvailable : vérifie via la table reservation
- listAvailableSeats : calcule la liste des sièges libres sur un vol

7.2.3 ReservationSyncRmi

- réception d'événements de réservation (créée/annulée/modifiée)
- utilisé pour synchronisation/monitoring (dans notre implémentation : logs serveur)

7.3 Choix RMI

- Simple et efficace en Java pur.
- Très adapté au "serveur↔serveur" demandé dans l'énoncé.
- Permet de garder REST/SOAP "orientés client" et RMI "orienté backend".

8. Sockets TCP : notifications en temps réel

8.1 Rôle

Les sockets permettent l'envoi temps réel vers les clients, par exemple :

- confirmation de réservation,
- conflit (siège déjà réservé),
- mise à jour des disponibilités.

8.2 Principe

- Serveur socket : accepte des connexions et garde une liste des clients connectés.
- Lors d'un événement (booking/cancel/change), le serveur envoie un message à tous (ou aux clients concernés).
- Client Swing : écoute en continu et met à jour l'affichage via le thread UI.

8.3 Exemple de messages (protocole simple)

- BOOKING_CONFIRMED reservationId=...
- BOOKING_CONFLICT flightId=... seatId=...
- SEATS_UPDATED flightId=...

9. Tableau de bord (Admin Dashboard)

9.1 Objectif du dashboard admin

Le tableau de bord administrateur sert à gérer les ressources du système et suivre l'activité globale. Il apporte :

- gestion centralisée des données (vols, avions, sièges),
- supervision des réservations,
- génération d'indicateurs utiles pour la prise de décision.

9.2 Fonctionnalités admin prévues/implémentées

(A) Gestion des vols

- Ajouter un vol : numéro, départ, destination, date/heure, avion
- Modifier un vol : changement d'horaire, d'avion, de route
- Supprimer un vol : si aucune réservation active ne bloque la suppression (intégrité FK)

(B) Gestion des avions

- Ajouter un avion : code, modèle, capacité totale, capacité eco/business
- Modifier un avion : mises à jour de capacité et informations

(C) Gestion des sièges

- Génération des sièges d'un avion (numérotation, classe)
- Modification d'un siège (classe, numéro)
- Consultation des sièges par avion

(D) Consultation globale des réservations

- Liste des réservations par vol
- Filtre par statut (CONFIRMED, CANCELLED)
- Consultation du détail d'une réservation (client, siège, vol)

(E) Rapports / statistiques (extension)

Exemples de rapports :

- Taux de remplissage d'un vol : confirmedSeats / totalSeats
- Vols les plus demandés : nombre de réservations confirmées par vol
- Évolution des réservations (par date)

9.3 Architecture côté admin

- Côté UI : une vue Swing "Admin" (menus + tableaux + formulaires)
- Côté backend :
    - REST pour les listes et consultations,
    - SOAP pour actions critiques (annulations/modifications si besoin),
    - RMI peut être utilisé en interne pour centraliser les opérations de gestion (FlightManagementRmi).

10. Sécurité, validation et gestion des erreurs

10.1 Validation des entrées

- Vérifier l'existence des identifiants (client, vol, siège).
- Empêcher modification/paiement si réservation annulée.
- Gérer les erreurs de base de données et les traduire en messages métiers.

10.2 Gestion d'erreurs par canal

- REST : codes HTTP + message JSON
- SOAP : faults typés (BookingException)
- RMI : RemoteException (erreur technique interne)
- sockets : protocole de messages + gestion d'erreurs côté client

Conclusion

FlyR illustre une architecture distribuée réaliste : REST pour les consultations fréquentes, SOAP pour les opérations critiques transactionnelles, RMI pour la communication interne entre services serveur et sockets TCP pour le temps réel. La persistance MySQL assure l'intégrité des données et les transactions garantissent une cohérence acceptable pour un prototype. Le projet respecte les exigences du mini-projet en mettant en avant la communication distribuée, la synchronisation et la gestion des conflits.