TP — SYSTÈME DE RÉSERVATION DE VOYAGES

Le présent travail vous amènera à **concevoir et prototyper un système de réservation de voyages** couvrant deux modes : aérien (avion) et ferroviaire (train). L'application comprend deux volets : un **volet administratif** (gestion de l'offre) et un **volet client** (recherche, réservation, paiement simulé), dont les besoins sont décrits ci-dessous. Le projet combine **rédaction d'un SEL**, **modélisation UML** (diagramme de classes + patrons) et **implémentation MVC légère**.

Général

L'objectif est de livrer un prototype fonctionnel minimal qui illustre un enchaînement simple : 1) publier des trajets (admin) → 2) rechercher (client) → 3) reserver un siège (client) → 4) confirmer/payer (client, paiement simulé).

Éléments hors portée (focalisés sur les besoins de base):

- Aller simple uniquement; une réservation porte sur un trajet (pas de panier multi-trajets, pas d'allers-retours).
- Aucune gestion d'authentification forte (pas de paiements réels, pas d'authentification/autorisation avancée).
- Un stockage en mémoire (pas de base de données exigée / persistance durable).

Volet administratif

L'administrateur gère les **terminaux**, les **opérateurs** et **l'offre de trajets**.

- Un **terminal** (aéroport ou gare) est identifié par un **code de trois lettres unique** et est associé à une **ville** quelque part dans le monde.
- Un **opérateur** (compagnie aérienne ou ligne de train) détient un **identifiant alphanumérique ≤ 6 caractères**, unique dans le système.
- L'**offre** se compose de *8trajets programmés** avec **date/heure de départ** (et d'arrivée si pertinent), rattachés à un **opérateur** et reliant des **terminaux distincts** (origine ≠ destination).

✈

Variante aviation : chaque **vol** porte un **ID** commençant par **deux lettres** (réservées à la compagnie) suivies de **chiffres** ; la partie alphabétique est **unique à la compagnie**, la partie numérique est **unique par vol** au sein de cette compagnie.

🚆

Variante ferroviaire : un **trajet de train** appartient à une **ligne** et parcourt une **suite fixe de gares** dans un ordre déterminé (les arrêts intermédiaires font partie de la définition du trajet).

Chaque véhicule comporte des **sections** (classes) et des **sièges**. Les **sièges d'une même section partagent exactement le même prix**. La **tarification** de base dépend de l'opérateur, avec les coefficients : $\alpha_F=1\quad\alpha_A=0{,}75\quad\alpha_P=0{,}60\quad\alpha_E=0{,}50$, **sans taxe**.

✈️ **Variante *aviation*** : sections possibles : **F**, **A**, **P**, **E** (*zéro ou plusieurs, sans doublon*).

La cabine est maillée en **≤ 100 rangées** et **≤ 10 colonnes** (**A–J**).

Gabarits autorisés par section :

- **S** : 3 colonnes, couloir entre **1–2**
- **C** : 4 colonnes, couloir entre **2–3**
- **M** : 6 colonnes, couloir entre **3–4**
- **L** : 10 colonnes, couloirs entre **3–4** et **7–8**

🚆 **Variante *ferroviaire*** : chaque **train** contient au **minimum** les sections **première (P)** et **économie (E)** ; la **disposition des sièges** y est **étroit (S)**.

Enfin, le système doit permettre la **consultation** de l'offre : **lister tous les trajets au départ/à l'arrivée d'un terminal** (aéroport/gare) avec leurs détails, ainsi que **tous les trajets d'un opérateur** donné (compagnie/ligne), toujours avec les informations complètes.

Volet client

La recherche par trajet
Le client peut **rechercher des trajets** à partir d'un **terminal d'origine** (aéroport ou gare) vers une **destination**, pour une **date donnée** et **une section** (classe) précisée, en ne retournant que les trajets avec des **sièges libres**.

Le **système affiche une liste de résultats** dans laquelle pour chaque trajet, on présente :

- la date et l'heure de départ,
- la durée,
- l'heure d'arrivée,
- l'opérateur (compagnie aérienne ou ligne ferroviaire),
- l'identifiant du trajet (numéro de vol ou identifiant de trajet train),
- le prix pour la section demandée
- et le nombre de sièges encore disponibles dans cette section.

✈️ **Variante *aviation*** : la recherche porte sur des vols entre deux aéroports, en tenant compte des classes F/A/P/E.

🚆 **Variante *ferroviaire*** : la recherche cible des trajets de train entre deux gares, en tenant compte des sections P/E et de la disponibilité sur le tout le segment demandé.

La réservation de sièges

Le client peut **réserver un siège disponible sur un trajet** (contrainte : la réservation n'est possible que si un siège est réellement libre). Il peut indiquer une **préférence de placement** (couloir ou fenêtre). À la création, le système fournit un **numéro de réservation**. Un **siège réservé** devient **indisponible** pour les autres passagers pendant **24 heures** ; passé ce délai, **en absence de paiement**, il **redevient disponible**.

Le paiement

Avec son **numéro de réservation**, le client peut **payer son siège réservé**. Il fournit ses **informations personnelles** (nom, courriel, numéro de passeport) et effectue un **paiement par carte de crédit** (paiement simulé). Le système émet une **confirmation** : la réservation devient un **siège assigné** au passager et le statut passe à **CONFIRMÉ**.

Votre mendat

Le deroulement de ce travail se fera de manière **ittérative**, avec une **démo partielle évaluée** à chaque itération jusqu'à la remise finale.

L'**avancement** sera suivi sur **Trello** : vous devez y créer un **tableau Kanban** avec votre coéquipier, inviter l'enseignante (lecture/évaluation) et tenir le *board* **à jour** (colonnes, cartes, étiquettes, échéances, commentaires) tout au long du projet. Le **tableau sera noté**, au même titre que les **livrables techniques**, et le **travail d'équipe** (organisation, répartition des tâches, communication, respect des délais).

Le travail est divisé en **trois itérations**, chacune avec son livrable :

- I. spécification des exigences logicielle.
- II. diagramme de classes UML & patrons.
- III. mini MVC en mémoire et tests.

I. SEL : spécification des exigences logicielle
À partir de l'**enocné des besoins**, vous devez amorcer la **rédaction d'un SEL**. Ce dernier sera **révisé et enrichi** au fil des itérations jusqu'à sa version finale pour la remise.

- Vous devrez suivre le **gabarit SEL** disponible sur moodle pour structurer ce livrable.
- La **totalité des sections** du gabarit devront être **complètes** pour la remise finale, sauf indication contraire donnée en classe.

II. Conception : diagramme de classes UML & patrons
Vous devrez produire un **diagramme de classes** centré sur le **domaine** (trajets, sections, réservation, opérateurs/terminaux, services).

- **Intégrez les patrons vus en classe** et rendez-les **explicites** dans le diagramme (stéréotypes/*tagging*), en veillant à ce qu'ils **servent le besoin** (réduction du couplage, variabilité, réutilisation, testabilité).
- **Justifiez** brièvement leur **usage** (en 3 phrases : problème visé, choix de conception, impacts).

Les patrons peuvent être **employés dans des contextes différents** (p. ex. tarification, allocation de sièges, sélection d'un service, configuration…) : si vous en utilisez un **à plusieurs endroits**, expliquez ce qui **distingue ces contextes**.

Veillez à illustrer la **séparation logique** entre couche **présentation** (contrôleurs, minces), couche **service** (cas d'usage/applicatif) et couche **domaine** (entités, valeurs, règles), sans surcharger le diagramme de détails techniques superflus.

Enfin, soigner la **cohérence sémantique** avec le SEL : le vocabulaire, les règles métier (p. ex. origine ≠ destination, prix par section, réservation 24 h) et les cas d'utilisation doivent **se refléter** dans le modèle.

> **Option (bonus)** : ajoutez un **patron GoF supplémentaire**, pertinent pour votre conception, que vous avez appris à utiliser et qui aura été **validé par l'enseignante** (par ex. un Observer pour notifier la disponibilité).

III. Implémentation : mini MVC (léger, en mémoire)
Implémentez un **MVC minimal** pour démontrer **deux contrôleurs** et leurs **cas d'usage associés**, avec une logique métier dans les services et un stockage en mémoire (pas de BD).

Voici des **exemples concrets de contrôleurs** avec ce qu'ils couvrent en grandes lignes :

- `SearchController` — parcours **"consulter l'offre"**
  Intègre la recherche de trajets (par mode (avion/rail), origine/destination, date, section) et affiche le détail d'un trajet (prix par section, disponibilité restante, etc.).

- `BookingController` — parcours **"réserver → payer → confirmer"**.
  Paiement simulé et confirmation (statut passe CONFIRMÉ). Crée une réservation de siège (vérif dispo, préférence couloir/fenêtre, création reservationId, TTL 24 h), puis effectue le paiement simulé et (passe le statut à CONFIRMÉ). Il peut égaleent permettre la consultation/annulation d'une réservation.

- `AdminTripsController` — parcours **"alimenter l'offre"**.
  Gère la création, la mise à jour et la suppression de trajets (opérateur, horaires, sections, prix par section), avec validations. Il permet également le *seed* rapide de quelques trajets à des fins de de démo pour la correction.

- `ReservationStatusController` — parcours **"suivre mon dossier"**.
  Expose le statut d'une réservation (RESERVÉ, CONFIRMÉ, expiré) et le temps restant avant expiration.

Le code devra être accompagné de **tests unitaires** visant des **méthodes du modèle** (p. ex. logique de tarification, disponibilité, allocation de sièges).

> Le nombre de tests et les méthodes ciblées seront discutés en classe et pourront varier d'une équipe à l'autre.

Grille de correction

Volet	Pondération	Détails d'évaluation (exemples)
Itérations (démos partielles)	**30 %**	3 itérations (10 % chacune) : respect des jalons, qualité des démos, capacité à intégrer la rétroaction, progression tangible.
SEL (Remise finale)	**15 %**	Gestion des versions, clarté/portée, glossaire, cas d'utilisation, exigences fonctionnelles / non fonctionnelles, règles métier, hypothèses/contraintes, cohérence interne.
UML (diagramme de classes)	**20 %**	lisibilité et alignement avec le SEL, modèle centré domaine, patrons intégrés et justifiés, invariants.
Implémentation (mini-MVC en mémoire)	**20 %**	2 contrôleurs min., services minces/explicites, flux *rechercher* → *réserver* → *payer*, données d'amorçage, propreté du code/README.
Unit tests	**10 %**	Niveau de tests adéquat, ciblant le modèle (ex. tarification, dispo, allocation), et pertinence des cas, clarté/indépendance.
Trello & travail d'équipe	**5 %**	Kanban à jour (colonnes, cartes, étiquettes, échéances, commentaires), invitation de l'enseignante, répartition/communication, respect des délais.