DOSSIER TECHNIQUE – ASSISTANT TACTIQUE INTELLIGENT

Table des matières

1. Présentation générale du projet et de ses objectifs

L'Assistant Tactique Intelligent est une application web à vocation militaire servant d'outil d'aide à la décision en temps réel pour une mission unique. Le scénario opérationnel est le suivant : un convoi logistique composé de quatre véhicules et d'un véhicule de reconnaissance se dirige vers un point d'extraction où est positionné un A400M. Le véhicule de reconnaissance évolue en avant du convoi afin de détecter les menaces et obstacles le plus tôt possible, tandis que les quatre véhicules du convoi transportent personnel et matériel vers la zone d'embarquement.

Le projet vise à fournir une vision unifiée et en temps réel de la situation tactique aux différents acteurs :
• Chef de convoi et équipages des 4 véhicules
• Équipage du véhicule de reconnaissance
• Poste de commandement (PC) supervisant l'ensemble de la mission
• Éventuellement pilote/équipage de l'A400M pour le suivi de l'approche

Chaque véhicule dispose d'une instance de l'application (via un navigateur sur un terminal durci ou une tablette), et le poste de commandement dispose d'une vue centralisée. L'application est conçue comme un outil « jetable » : elle n'est utilisée que pour cette mission précise, sur une fenêtre temporelle courte. Les données sont liées à ce scénario unique et ne sont pas destinées à être stockées et réutilisées à long terme (hors besoins de débriefing).

Fonctionnellement, l'Assistant Tactique Intelligent doit permettre :
• La visualisation cartographique en temps quasi réel des positions des cinq véhicules (4 convoi + 1 reconnaissance) ainsi que des menaces détectées.
• Une messagerie opérationnelle où chaque véhicule peut envoyer des messages individuellement. Le poste de commandement reçoit tous les messages et toutes les alertes de manière centralisée.
• La remontée et la visualisation d'alertes (obstacles, menaces, zones dangereuses) issues à la fois de signalements humains et de la détection assistée par IA.
• Des fonctions de reconnaissance visuelle via la caméra (par exemple pour aider le véhicule de reconnaissance à identifier des menaces ou des anomalies).
• Un briefing automatisé en début de mission et un débriefing en fin de mission, générés à partir des données disponibles et des événements survenus.

L'objectif principal est de fournir aux utilisateurs un outil simple, fiable, sécurisé et focalisé sur une seule mission, afin d'améliorer la compréhension de la situation, la coordination et la rapidité de décision sur le terrain.

2. Architecture technique détaillée

L'architecture de l'application suit un modèle client–serveur multi-couches, avec une séparation nette entre :
• Le front-end web (React), exécuté dans le navigateur des terminaux embarqués (véhicules et

PC).
• Le back-end principal en Java, exposant une API REST qui centralise la logique métier et l'accès aux données.
• Un module d'intelligence artificielle en Python, dédié aux traitements avancés (vision, analyse, génération de synthèse).
• Une base de données relationnelle pour la persistance à court terme (durée de la mission) des positions, messages, alertes et logs.

2.1 Composants principaux de l'architecture

Clients front-end

Cinq clients principaux utilisent l'interface React :
• Véhicule 1, Véhicule 2, Véhicule 3, Véhicule 4 : véhicules du convoi.
• Véhicule RECO : véhicule de reconnaissance, déployé en avant du convoi.
• Poste de commandement : client particulier disposant d'une vue globale sur l'ensemble des flux.

Chaque client front-end est associé à une identité logique (par exemple un identifiant de véhicule ou de rôle) transmise au back-end lors des requêtes. Cet identifiant permet d'attribuer correctement l'origine de chaque message et de chaque alerte, et de filtrer éventuellement les données affichées selon le rôle (par exemple, le PC voit tout, les véhicules voient au minimum leurs propres messages et ceux qui leur sont destinés).

Back-end Java

Le serveur Java implémente l'API REST utilisée par tous les clients. Il gère :
• L'authentification ou l'identification simple des clients (id de véhicule, rôle).
• La logique métier associée aux messages, alertes, positions, briefing et débriefing.
• La persistance en base de données des objets métier (positions, messages, alertes, événements).
• La communication avec le module Python pour les traitements IA.

Le serveur Java est le point d'entrée unique pour tous les clients. Il centralise les flux entrants (positions, messages, alertes) pour les rendre cohérents et accessibles aux différents acteurs (en particulier le poste de commandement).

Module IA Python

Le module Python est conçu comme un micro-service spécialisé. Il prend en charge :
• La reconnaissance d'images (par exemple analyse d'images envoyées par le véhicule de reconnaissance).
• L'analyse de scénario (par exemple estimation de risque sur un itinéraire ou proposition d'itinéraire alternatif).
• La génération de contenus textuels assistés (résumés pour briefing/débriefing).

Ce module est exposé via une API (par exemple REST via Flask/FastAPI). Il est appelé par le serveur Java, qui lui transmet les données brutes à analyser (image, séries d'événements, etc.) et récupère un résultat structuré qu'il renvoie ensuite au front-end ou persist en base.

Base de données

La base de données relationnelle stocke les éléments suivants :
• Véhicules et rôles (convoi 1 à 4, reconnaissance, poste de commandement).
• Positions dans le temps (traces des véhicules, utile pour la carte et le débriefing).
• Messages envoyés par les véhicules et le PC.

• Alertes (type, origine, niveau de criticité, position associée).
• Événements de mission utiles au débriefing (par exemple franchissement de zones clé, contacts, incidents).

La durée de vie de ces données est liée à la mission. Une fois le débriefing réalisé, les données peuvent être archivées ou supprimées selon les contraintes définies (prototype jetable ou mission réelle).

2.2 Interactions entre composants

Les clients React communiquent exclusivement avec l'API REST du back-end Java via HTTP(S). Le back-end Java :
• Lit et écrit en base de données pour conserver les informations de mission.
• Appelle le module Python pour les traitements IA lorsque nécessaire.
• Diffuse les événements pertinents aux clients (par exemple via WebSocket ou par polling côté front).

Cette architecture permet :
• Une séparation claire entre interface, logique métier et IA.
• Une évolution indépendante des composants (on peut faire évoluer la partie IA sans toucher au front).
• Un déploiement relativement simple en environnement contrôlé (par exemple réseau interne de la mission).

3. Composants Front-End (React)

L'interface utilisateur est réalisée en React sous forme de Single Page Application.
L'application est structurée autour de plusieurs pages et composants correspondant aux grandes fonctionnalités : carte, messagerie, alertes, caméra/reconnaissance, briefing et débriefing.

3.1 Structure générale

La structure typique du projet côté front peut être organisée comme suit :
• src/
  • App.jsx (ou App.js) : point d'entrée de l'application, configuration du routage.
  • pages/
    • CartePage.jsx
    • MessageriePage.jsx
    • AlertesPage.jsx
    • CameraPage.jsx
    • BriefingPage.jsx
    • DebriefingPage.jsx
  • components/
    • Carte
    • ListeMessages
    • FormulaireMessage
    • ListeAlertes
    • FormulaireAlerte
    • Composants communs (boutons, navigation, etc.)
  • services/
    • api.js (fonctions pour appeler le back-end Java)
    • éventuellement des hooks personnalisés pour les appels périodiques ou temps réel.

App.jsx configure les routes (par exemple avec React Router) :
• /carte
• /messagerie

• /alertes
• /camera
• /briefing
• /debriefing

Chaque véhicule et le poste de commandement utilisent la même application, mais avec une configuration ou un profil différent (par exemple un paramètre d'URL, un réglage lors de la connexion ou un fichier de configuration) indiquant leur identité (véhicule 1, 2, 3, 4, RECO, PC).

3.2 Composants fonctionnels clés

Carte

Le composant Carte affiche en temps quasi réel :
• La position des 4 véhicules du convoi.
• La position du véhicule de reconnaissance.
• Les menaces, alertes et zones à risque.
• Le point d'extraction (A400M).

Il interroge régulièrement le back-end (par exemple GET /api/positions) pour récupérer les positions actuelles, et met à jour les marqueurs. Dans un contexte plus avancé, des notifications push peuvent être utilisées pour accélérer les mises à jour.

Messagerie

La messagerie est un élément central du projet. Chaque véhicule dispose d'une capacité d'envoi de messages individuels. Les principes sont :
• Un message est toujours associé à une source (id de véhicule ou PC).
• Les véhicules envoient leurs messages via POST /api/messages.
• Le poste de commandement reçoit l'intégralité des messages et des alertes, ce qui lui permet d'avoir une vision globale.

Côté front, la MessageriePage se compose de :
• Une liste de messages filtrée selon le profil de l'utilisateur (par exemple un véhicule voit au minimum ses propres messages échangés avec le PC, le PC voit tous les messages).
• Un formulaire permettant de saisir et d'envoyer un nouveau message.

Le rafraîchissement de la liste peut être réalisé par polling régulier sur GET /api/messages ou via WebSocket si l'API le permet.

Alertes

La page d'Alertes présente la liste des alertes en cours et passées pour la mission :
• Alertes créées par les véhicules (signalements humains).
• Alertes générées à partir des analyses de l'IA (par exemple détection automatique sur image ou analyse de risque d'itinéraire).

Le poste de commandement voit l'ensemble des alertes, les véhicules au minimum celles qui concernent leur zone ou leur progression. Le poste de commandement peut être l'acteur principal pour valider ou re-catégoriser certaines alertes.

Caméra et reconnaissance visuelle

Le composant CameraPage permet aux véhicules (surtout le RECO) de :
• Activer la caméra du terminal.

• Capturer une image ou envoyer périodiquement des images au back-end.
• Obtenir en retour une analyse IA (présence d'un véhicule suspect, d'un groupe, d'un obstacle).

Le front-end capture une image, l'envoie via POST /api/analyseImage, puis affiche le résultat (texte d'analyse, éventuellement surcouches graphiques sur l'image).

Briefing et débriefing

BriefingPage :
• Affiche un résumé généré en début de mission (contexte, objectif, itinéraire prévu, situation ennemie supposée).
• Ce résumé est fourni par le back-end, éventuellement généré ou enrichi par l'IA Python.

DebriefingPage :
• Affiche une synthèse de la mission en fin d'opération.
• Le contenu est produit à partir des données collectées (positions, messages, alertes, incidents) et éventuellement résumé par le module IA.

3.3 Gestion de l'état et des appels API

Les hooks React (useState, useEffect, éventuellement useContext) sont utilisés pour :
• Stocker l'état local des pages (listes, formulaires, états de chargement).
• Effectuer les appels API au montage des composants ou à intervalles réguliers.
• Partager certaines informations globales (profil utilisateur, identité du véhicule, rôle PC ou véhicule) via un contexte.

Des hooks personnalisés peuvent être mis en place, par exemple :
• usePolling(endpoint, interval) pour interroger régulièrement une API.
• useMessages() pour centraliser la gestion des messages.

4. Composants Back-End (Java et Python)

4.1 Back-end Java – API REST et logique métier

Le back-end Java expose une API REST structurée en plusieurs domaines fonctionnels : positions, messages, alertes, briefing/débriefing, et éventuellement gestion de profil/rôles.

Organisation typique :
• Contrôleurs (Controllers) : définissent les endpoints (/api/positions, /api/messages, /api/alertes, /api/briefing, /api/debriefing, /api/analyseImageProxy, etc.).
• Services : implémentent la logique métier pour chaque cas d'usage.
• Repositories : gèrent la persistance (accès à la base de données).

Exemples de responsabilités :
• PositionsController :
  • GET /api/positions : renvoie les positions actuelles des 5 véhicules.
  • POST /api/positions : utilisé par les clients pour mettre à jour la position de leur véhicule.
• MessagesController :
  • GET /api/messages : renvoie les messages pertinents pour le client (tous pour le PC, filtrés pour un véhicule).
  • POST /api/messages : réception d'un nouveau message d'un véhicule ou du PC.
• AlertesController :
  • GET /api/alertes : renvoie les alertes en cours et passées.
  • POST /api/alertes : enregistre une nouvelle alerte (créée par un véhicule ou le PC).

• BriefingController :
  • GET /api/briefing : renvoie le briefing de début de mission.
  • GET /api/debriefing : renvoie le débriefing de fin de mission.

Le poste de commandement étant le point de vue global, la logique de filtrage côté back-end est importante :
• Pour un client véhicule, les endpoints comme GET /api/messages ou GET /api/alertes renvoient un sous-ensemble adapté (messages où le véhicule est émetteur ou destinataire, alertes pertinentes).
• Pour le PC, ces mêmes endpoints renvoient l'intégralité des données relatives à la mission.

Le back-end Java gère aussi :
• L'agrégation des données pour le débriefing (récupération des événements clés).
• Le relais des images et des données vers le service Python lorsque nécessaire (par exemple proxy pour l'analyse d'image).

4.2 Module IA Python – vision, analyse et génération de synthèse

Le module Python agit comme un service annexe spécialisé. Il n'est pas directement appelé par le front, mais uniquement par le back-end Java. Ses principales capacités sont :
• Analyse d'images :
  • Réception d'une image depuis le serveur Java.
  • Application d'un modèle de vision (ou de règles) pour détecter des objets ou menaces.
  • Renvoi d'un résultat structuré (types d'objets, niveau de confiance, éventuelles zones d'intérêt sur l'image).
• Analyse de scénario et aide à la décision :
  • Éventuelle estimation de risque sur un segment d'itinéraire à partir de données (alertes, historique).
  • Proposition de recommandations de contournement ou d'itinéraire alternatif.
• Génération de textes synthétiques :
  • Création d'un texte de briefing basé sur les données d'entrée (zone, mission, forces en présence).
  • Génération d'un débriefing à partir des événements enregistrés pendant la mission.

La communication se fait généralement via JSON. Par exemple :
• Java envoie à Python un payload contenant l'historique de la mission pour générer le débrief.
• Python renvoie un texte synthétique et éventuellement quelques indicateurs chiffrés.

5. Schéma logique de l'architecture

On distingue les blocs logiques suivants :
• Bloc clients front-end :
  • Véhicules (4 + 1 RECO)
  • Poste de commandement
• Bloc back-end Java :
  • API REST (contrôleurs)
  • Services métier
  • Accès base de données
  • Connecteur vers le service IA Python
• Bloc IA Python :
  • Service d'analyse d'images
  • Service de génération de synthèses
• Bloc base de données :
  • Tables pour véhicules, positions, messages, alertes, événements

Le flux général est :

• Les clients front-end envoient des requêtes au back-end Java.
• Le back-end lit/écrit en base et délègue le cas échéant au service IA Python.
• Le back-end renvoie des réponses JSON aux clients et peut pousser certaines mises à jour (temps réel) selon l'implémentation.

6. Flux de données et interactions front/back

6.1 Lancement de l'application et briefing

1. L'utilisateur ouvre l'application sur son terminal (véhicule ou PC).
2. Le front-end charge le profil (véhicule 1, 2, 3, 4, RECO ou PC).
3. BriefingPage appelle GET /api/briefing.
4. Le back-end Java récupère les données de contexte et éventuellement appelle le service Python pour générer le texte de briefing.
5. Le front-end affiche le briefing à l'utilisateur.

6.2 Suivi du convoi et du véhicule de reconnaissance sur la carte

1. Chaque véhicule envoie périodiquement sa position via POST /api/positions (avec son identifiant).
2. Le back-end Java met à jour en base les positions.
3. Les différents clients (véhicules et PC) appellent régulièrement GET /api/positions.
4. Le front-end met à jour la carte en fonction des données reçues, montrant les quatre véhicules du convoi, le véhicule de reconnaissance et la position de l'A400M.

6.3 Messagerie entre véhicules et poste de commandement

1. Un véhicule saisit un message dans la MessageriePage et envoie POST /api/messages.
2. Le back-end Java enregistre le message (source = véhicule X) en base.
3. Le PC, via GET /api/messages ou via un flux temps réel, reçoit l'intégralité des messages, y compris celui-ci.
4. Le PC peut répondre, par exemple via POST /api/messages (source = PC, destinataire = véhicule X ou groupe de véhicules).
5. Les véhicules concernés récupèrent les réponses via GET /api/messages ou flux temps réel.

Ainsi, chaque véhicule envoie ses messages individuellement, et le PC dispose d'une vue centralisée lui permettant de suivre la communication globale et de coordonner la mission.

6.4 Création et diffusion d'alertes

1. Un véhicule (par exemple le RECO) observe une menace ou un obstacle et crée une alerte via AlertesPage (POST /api/alertes).
2. Le back-end Java enregistre l'alerte et peut éventuellement appeler le module IA Python pour la catégoriser ou l'enrichir.
3. Le PC reçoit l'alerte dans sa liste globale (GET /api/alertes). Il peut la valider, la requalifier ou en créer d'autres.
4. Les véhicules récupèrent les alertes pertinentes via GET /api/alertes, ce qui permet d'afficher sur la carte les zones dangereuses ou points à éviter.

6.5 Reconnaissance visuelle du véhicule de reconnaissance

1. Le véhicule de reconnaissance active la caméra via CameraPage.
2. Le front-end capture une image et l'envoie à POST /api/analyseImage.
3. Le back-end Java relaie l'image au service IA Python.
4. Python renvoie une analyse (exemple : véhicule potentiellement hostile, niveau de confiance, zone d'intérêt).

5. Le back-end renvoie ce résultat au front-end, qui l’affiche au véhicule de reconnaissance et peut éventuellement générer une alerte automatique.

6.6 Débriefing de fin de mission

1. À l’arrivée du convoi à l’A400M, la mission est considérée comme terminée.
2. Le PC déclenche la génération du débriefing via GET /api/debriefing.
3. Le back-end Java rassemble les données de mission (positions, messages, alertes, événements) et les transmet au module IA Python.
4. Python produit un texte de synthèse et des indicateurs clés.
5. Le back-end renvoie ces informations au front-end, qui les affiche sur DebriefingPage.
6. Éventuellement, le rapport peut être exporté pour archivage, sachant que l’application est conçue pour cette mission unique et peut être supprimée ou réinitialisée après le débriefing.

Ce dossier décrit ainsi l’architecture logique et le fonctionnement détaillé de l’Assistant Tactique Intelligent pour le scénario d’un convoi de quatre véhicules et d’un véhicule de reconnaissance se dirigeant vers un A400M, avec une messagerie individualisée par véhicule et une vue globale centralisée au poste de commandement, dans un contexte d’application web éphémère dédiée à une mission unique.