

Shinkasen.dev cahier des charges technique

▼ Introduction

📍 Contexte du projet

« Dans un contexte de mobilité croissante et d'événements majeurs, la sécurisation des documents sensibles devient primordiale. Shinkasen.dev développe une solution innovante de traçabilité en temps réel, permettant aux entreprises et aux particuliers de suivre et protéger efficacement leurs biens de valeur. »



timeline

title Suivi d'une valise perdue avec Shinkasen.dev

- 2025-06-01 : 📦 Perte de la valise
- 2025-06-01 : 💡 Activation du dispositif GNSS/4G
- 2025-06-01 : 🚤 Traçage via ESP32 + SIM7080G
- 2025-06-01 : 🛫 Transmission CBOR vers serveur TCP
- 2025-06-01 : 🧠 Traitement + mise en cache API
- 2025-06-01 : 🌎 Affichage sur Mapbox (React)
- 2025-06-01 : 📨 Notification utilisateur (React/UI)
- 2025-06-01 : 🔒 Sécurité retrouvée + historique disponible

🎯 Objectifs du projet

- Suivre en temps réel des valises avec une précision de 10 mètres.
- Moduler la fréquence d'envoi des données en fonction des besoins.
- Proposer dans le futur une plateforme B2C et B2B avec des tableaux de bord adaptés.
- **Scalabilité** : L'architecture du projet est conçue pour gérer une montée en charge progressive. Le système sera capable de traiter efficacement un nombre croissant de valises connectées, en adaptant ses ressources (serveurs, base de données, cache) selon la demande.

🌐 Portée du projet

Le projet couvre l'ensemble du territoire français, en ciblant :

- **Les particuliers** transportant objets sensibles
- **Les entreprises** équipant leurs employés en déplacement

Il pourra évoluer vers d'autres pays selon les contraintes techniques et réglementaires.

▼ Glossaire

Terme / Acronyme	Définition
ESP32	Microcontrôleur Wi-Fi/BT développé par Espressif, utilisé ici pour piloter le GNSS et le modem 4G.
SIM7080G	Module cellulaire 4G (CAT-M1 / NB-IoT / GPRS) permettant la communication via AT Commands.
GNSS	Global Navigation Satellite System : ensemble de systèmes de positionnement (GPS, Galileo, etc.).
AT Commandes	Jeu d'instructions textuelles envoyées au module SIM pour le configurer ou envoyer des données.
CBOR	Concise Binary Object Representation : format binaire compact pour représenter des objets JSON.
Server TCP (Node.js)	Serveur basé sur le module <code>net</code> de Node.js, capable de recevoir les connexions binaires entrantes.
API Express	Backend RESTful utilisant Express.js pour exposer des endpoints vers l'interface web React.
React	Framework JavaScript utilisé pour construire l'interface utilisateur (tableau de bord, cartes, etc.).
Mapbox	Service de cartographie interactif utilisé pour afficher la position des valises.
MongoDB	Base de données NoSQL utilisée pour stocker les données GPS et les informations utilisateurs.
Memcached	Système de cache en mémoire utilisé pour accélérer les requêtes côté API.
CAT-M1	Technologie cellulaire IoT (LTE-M), adaptée aux objets connectés, faible consommation et latence modérée.
NB-IoT	Alternative à CAT-M1, plus économique mais moins adaptée aux transmissions fréquentes.
B2B	Business to Business : vente ou service à destination des entreprises.
B2C	Business to Consumer : vente ou service à destination des particuliers.

IMEI	Identifiant unique d'un appareil mobile, utilisé ici pour identifier chaque carte SIM7080G.
Pinggy	Service de tunnel TCP permettant de relayer des connexions de l'ESP32 vers un localhost en développement.

▼ Description générale

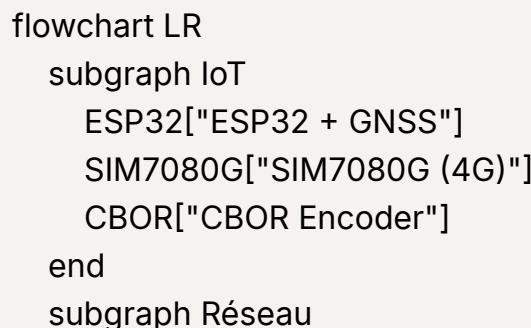
▼ Vue d'ensemble du système

Le système Shinkasen.dev repose sur une architecture orientée IoT, conçue pour suivre des valises en temps réel ou en différé. Il est composé de plusieurs briques :

- **ESP32 + SIM7080G** : capte la position GPS grâce au module GNSS, encode les données en CBOR, puis les transmet via 4G (CAT-M1) au serveur.
- **Serveur TCP (Node.js)** : reçoit les messages CBOR, les décode, et les stocke dans une base MongoDB.
- **MongoDB** : base de données centrale NoSQL, elle stocke les positions géographiques avec l'IMEI de l'appareil émetteur.
- **API Express.js** : expose des endpoints REST pour que le front-end puisse récupérer les informations.
- **Memcached** : permet de mettre en cache les réponses API fréquentes (ex : dernières positions).
- **React + Mapbox** : permet à l'utilisateur de visualiser les valises sur une carte interactive ainsi que des statistiques via des line charts.

 **Important** : Le serveur TCP et l'API Express sont **indépendants**. Ils n'interagissent pas directement : c'est la base de données qui sert de pont entre les deux.

▼ Architecture système



```

Pinggy["Pinggy (Tunnel TCP)"]
end
subgraph Backend
    TCP["Serveur TCP (Node.js)"]
    DB["MongoDB"]
    API["API Express"]
    Cache["Memcached"]
end
subgraph Frontend
    React["React UI"]
    Mapbox["Carte Mapbox"]
    Charts["Line Charts"]
end

ESP32 → CBOR → SIM7080G → Pinggy → TCP → DB
API → |Lecture/écriture| DB
API → Cache
React → |Requête API| API
React → Mapbox
React → Charts

```

▼ 🧑 Besoins utilisateurs

Partie commune :

- **Authentification (login / register)**
- **Accès à un tableau de bord personnalisé**
- Visualisation de :
 - la carte Mapbox avec valises en **temps réel** ou **mode historique**
 - des **statistiques** (nombre de points collectés par jour, par appareil, etc.)

Cas B2C (particulier) :

- Consultation de ses propres valises
- Changement de la fréquence de transmission des données (si activé dans l'offre)

Cas B2B (entreprise) :

- Visualisation groupée de tous les dispositifs
- Accès à des dashboards consolidés
- Possibilité d'offrir une interface d'administration interne (via une option dans l'offre)

▼ Contraintes générales

- **Plateforme** : dans un premier temps uniquement sur **PC**, via navigateur (Chrome, Firefox)
- **Extension mobile** envisagée à moyen terme
- **Partenariat futur avec Google Maps / Apple Maps** : possible évolution du front-end

▼ RGPD & CNIL

Shinkasen.dev est soumis aux régulations européennes en matière de protection des données.

- Les données de localisation sont considérées comme **sensibles**.
- Aucune donnée personnelle (nom, prénom, etc.) n'est collectée **sans consentement explicite**.
- Les données sont **stockées en France** et ne sont accessibles qu'aux services autorisés.

▼ Exigences fonctionnelles

▼ Fonctionnalités principales

Côté IoT (ESP32 + SIM7080G)

- Envoi de la position GNSS **encodée en CBOR** vers le serveur TCP via la puce 4G.
- Fréquence de transmission **modifiable à distance** (via commande ou API).
- Fonctionnement en **mode CAT-M1**, plus rapide que NB-IoT, adapté à la mobilité.
- Utilisation des **AT Commandes** pour configurer et communiquer avec le réseau.
- Implémentation de **machines d'états en C++** pour :

- Activer/désactiver GNSS
- Gérer la connectivité réseau
- Gérer les erreurs et relancer les séquences en cas d'échec

Côté Serveur TCP

- Réception des paquets CBOR
- Décodage + vérification du format
- Insertion dans la base MongoDB

Côté API Express

- Authentification (login, register)
- Endpoints sécurisés pour :
 - Récupération des positions
 - Statistiques par jour / valise (IMEI)
 - Gestion de la fréquence d'envoi
- Mise en cache Memcached pour les appels fréquents

Côté Frontend React

- Interface responsive (PC)
- Dashboard avec :
 - Carte Mapbox avec filtres : live / historique
 - Graphiques (line charts) des points collectés
 - Récupération des données toutes les minutes

▼ Cas d'utilisation

Utilisateur particulier (B2C)

Étape	Description
1.	L'utilisateur se connecte avec login/mot de passe
2.	Il accède à son dashboard personnalisé
3.	Il visualise ses valises en temps réel (ou historique)

4.	Il consulte les données collectées (nombre de points, fréquence, etc.)
5.	Il ajuste la fréquence d'envoi s'il a souscrit à l'option

👤 Utilisateur entreprise (B2B / Admin)

- Même parcours que le B2C avec des **droits étendus** :
 - Visualisation **multi-dispositifs** par employé
 - Regroupement des données par utilisateur ou département
 - **Modification des paramètres de plusieurs cartes** à distance (via commandes CBOR ou autres)

▼ 🎬 Scénario d'utilisation bout en bout

```

sequenceDiagram
    participant ESP32
    participant GNSS
    participant SIM7080G
    participant TCP_Server
    participant MongoDB
    participant Express_API
    participant React_UI

    ESP32->>GNSS: Activation
    GNSS->>ESP32: Coordonnées GPS
    ESP32->>GNSS: Extinction
    ESP32->>SIM7080G: AT Commandes (mode CAT-M1)
    ESP32->>TCP_Server: Envoi CBOR via Pinggy
    TCP_Server->>MongoDB: Stockage des données

    React_UI->>Express_API: Requête login
    Express_API->>MongoDB: Vérification user
    MongoDB->>Express_API: Résultat auth
    Express_API->>React_UI: Token + accès

    React_UI->>Express_API: Requêtes valises + stats
    Express_API->>Memcached: Cache (si disponible)
  
```

Express_API → MongoDB: Données GPS
 MongoDB → Express_API: Coordonnées + stats
 Express_API → React_UI: JSON réponse
 React_UI → React_UI: Affichage sur carte et graphique

▼ Exigences non fonctionnelles

▼ ⚡ Performance

Critère	Spécification
Fréquence d'envoi	Toutes les 3 minutes par défaut, modifiable à distance (1 min / 10 min, etc.)
Scalabilité	Prévu pour 2 dispositifs au départ, avec montée à 100 , voire 1000+ pour les offres B2B

▼ 🔒 Sécurité

Composant	Niveau de sécurité actuel	Évolutions prévues
ESP32 → TCP	Aucune sécurité pour l'instant (pas de chiffrement)	Possible ajout de TLS/DTLS ou CBOR Web Token (CWT)
API Express	Authentification gérée via Memcached pour stocker les sessions utilisateurs de manière temporaire, remplaçant ainsi la solution JWT initialement envisagée.	<ul style="list-style-type: none"> Mettre en place un système de refresh token pour prolonger automatiquement les sessions valides Implémenter un mécanisme de déconnexion forcée en cas de détection d'activité suspecte Ajouter une authentification à deux facteurs (2FA) pour renforcer la sécurité
Données	Accès limité aux données selon les droits	RGPD appliqué, données pseudonymisées
Base MongoDB	Hébergement en France, accès sécurisé	Règles strictes sur les rôles utilisateurs

▼ 🔄 Fiabilité

- L'ESP32 intègre des **mécanismes de relance automatique** si la transmission échoue (timeout, état "réessay" dans la machine d'état).
- Les données sont **temporairement mises en tampon** en mémoire (dans la RAM de l'ESP32), avant d'être renvoyées.

▼ 🔧 Maintenabilité

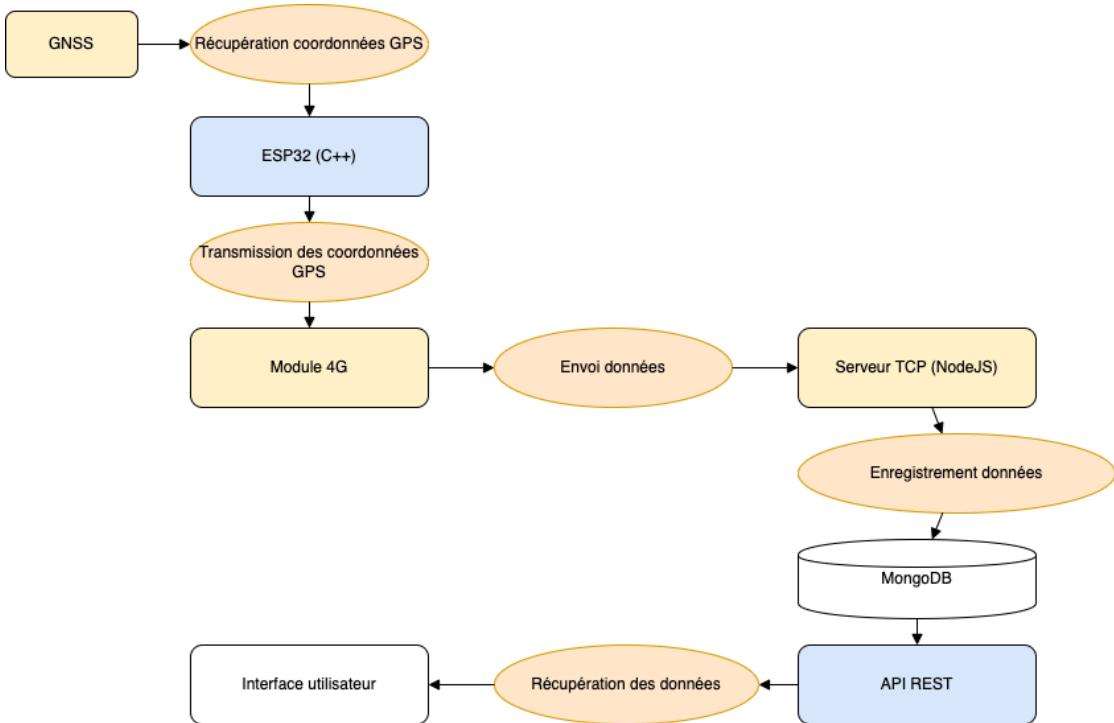
Élément	Bonnes pratiques prévues
Code ESP32	Modulaire, commenté, structuré en modules réutilisables (GNSS , 4G , pipeline)
Serveur TCP	décodage CBOR séparé dans un module
API Express	Middleware, gestion des erreurs, documentation Swagger
Frontend React	Utilisation de composants réutilisables
Tests	Tests unitaires sur chaque module de l'application embarquée, tests fonctionnels React, tests de charge à prévoir

▼ 🔄 Compatibilité

Système	Détail
Navigateurs	Compatibilité Chrome / Firefox assurée (desktop uniquement pour l'instant)
Modules IoT	Compatible avec tout module SIM7080G disposant d'un firmware à jour
Plateformes Cloud	Prévu pour être dockerisé , donc compatible avec Azure / OVH / AWS
Évolutivité	Possibilité d'intégrer d'autres types de capteurs (température, humidité) et formats (JSON, Protobuf)

▼ Architecture technique

▼ 📦 Description de l'architecture globale



Le système est découpé en **5 couches principales** :

- 1. Matériel embarqué (ESP32 + SIM7080G + GNSS)**
 - Acquisition et envoi des données
- 2. Infrastructure de communication (Réseau 4G / Pinggy)**
 - Acheminement vers le serveur local pendant la phase de développement
- 3. Serveur TCP (Node.js)**
 - Réception et décodage CBOR, stockage MongoDB
- 4. API REST Express.js**
 - Serveur web exposant les données pour le front-end
- 5. Interface utilisateur (React)**
 - Visualisation via Mapbox et charts, avec login et tableaux de bord

▼ Technologies et outils utilisés

Côté	Technologies / Protocoles
Microcontrôleur	ESP32 (C++ avec PlatformIO)
Modem cellulaire	SIM7080G, protocole AT commandes, réseau CAT-M1

Encodage	CBOR (Concise Binary Object Representation)
Tunnel de dev	Pinggy.io (TCP tunneling vers localhost)
Serveur TCP	Node.js (<code>net</code> , <code>cbor</code> , <code>dotenv</code> , <code>mongodb</code>)
Base de données	MongoDB
Cache	Memcached
API Web	Express.js, JWT Auth, REST
Frontend	React.js, Tailwind, Mapbox, ApexCharts
Containerisation	Docker (serveur TCP, API, Mongo, Memcached)
Déploiement cible (à venir)	Cloud public (OVH, Azure, AWS – à définir)

▼ 📦 Conteneurisation Docker (prévue)

Dockeriser les services suivants :

- `tcp-server` : port 4000
- `api-server` : port 3000
- `mongo` : port 27017
- `memcached` : port 11211

Cela permettra :

- Une **installation simplifiée**
- Des **tests en local homogènes**
- Une base pour un futur **CI/CD** avec GitHub Actions ou Jenkins

▼ Tableau des ports et services exposés

Service	Port	Description
TCP Server	4000	Réception données CBOR
API Express	3000	Endpoints REST sécurisés
MongoDB	27017	Stockage central
Memcached	11211	Cache pour les données fréquentes
React	5173	Interface utilisateur

▼ Spécifications détaillées

▼ Spécifications des composants

Données CBOR

Le message envoyé par l'ESP32 est un **objet JSON encodé en CBOR**, structuré ainsi :

```
{  
    "imei": "867123456789012",  
    "lat": 48.8566,  
    "lon": 2.3522,  
    "timestamp": "2025-06-06T10:00:00Z"  
}
```

 D'autres champs pourraient être ajoutés selon les capteurs connectés (ex : température, humidité, choc...).

Format de stockage MongoDB

Chaque document dans MongoDB suit le schéma suivant :

```
{  
    "_id": ObjectId,  
    "imei": "867123456789012",  
    "position": {  
        "latitude": 48.8566,  
        "longitude": 2.3522  
    },  
    "receivedAt": ISODate("2025-06-06T10:00:00Z")  
}
```

Actuellement, la notion de valise n'est pas matérialisée, seul l'IMEI sert d'identifiant.

⚠ Il sera pertinent d'introduire une collection  liée à un utilisateur.

▼ Plan de test et validation

Stratégie de test globale

Le projet Shinkasen.dev nécessite des tests à plusieurs niveaux pour garantir fiabilité, sécurité et robustesse :

Niveau	Type de test	Objectif principal
ESP32 (embarqué)	Tests manuels et unitaires	Vérifier la machine d'état, GNSS, envoi CBOR, etc.
Serveur TCP	Tests d'intégration	Vérifier le décodage, les erreurs de format, les insertions MongoDB
API Express	Tests unitaires + fonctionnels	Vérifier les endpoints (auth, GET valises, etc.)
Frontend React	Tests fonctionnels	Vérifier l'affichage des données, les graphiques, le login
Système complet	Tests bout-en-bout (E2E)	Vérifier qu'un envoi GNSS apparaît bien côté carte UI

Scénarios de tests fonctionnels (React)

Scénario	Résultat attendu
Authentification utilisateur	Accès au dashboard avec son propre token
Visualisation des données	Carte Mapbox + données valides + graphique dynamique
Fréquence d'envoi modifiée	L'ESP32 adapte son comportement dans les 3 min suivantes

Critères d'acceptation

Fonctionnalité	Critère de succès
Envoi de données CBOR	Donnée stockée dans MongoDB avec les bons champs
Interface utilisateur complète	Carte, graphes et sidebar disponibles

Résilience de l'ESP32	En cas d'échec, une nouvelle tentative est effectuée
API cache	Appels répétés renvoyés plus vite via Memcached

▼ Plan de déploiement

⌚ Objectif

Déployer le système complet **de manière progressive et modulaire**, avec une stratégie adaptée aux besoins du POC puis à la montée en charge.

▼ 🛠 Stratégie de déploiement (étapes)

Étape	Objectif	Environnement
Local / Dev	Développement + test manuel de chaque brique	Docker local + tunnel Pinggy
Pré-production (POC)	Démo pour encadrants ou partenaires B2B	cloud léger (AWS/Azure)
Production B2C/B2B	Accès public sécurisé avec hébergement cloud	Docker full stack + domaine custom

▼ 📦 Conteneurisation (Docker)

Tu prévois de dockeriser :

Conteneur	Rôle	Port exposé	Dockerfile ?
tcp-server	Réception des données CBOR	4000	<input checked="" type="checkbox"/>
api-server	Express.js avec JWT & Memcached	3000	<input checked="" type="checkbox"/>
mongo-iot	Base MongoDB	27017	image officielle
memcached	Cache en mémoire	11211	image officielle
react-app	Interface utilisateur (option : nginx)	5173	<input checked="" type="checkbox"/>

▼ ☁ Déploiement cloud envisagé

Service	Fournisseur(s) recommandés
TCP / API	Azure Container App
MongoDB	MongoDB Atlas (hébergement FR ou EU)
React UI	AWS
Nom de domaine	OVH, Gandi, Cloudflare

HTTPS	Let's Encrypt via certbot/NGINX
--------------	---------------------------------

▼ Formation et documentation

-  Création d'un **README.md** pour chaque composant
-  Documentation Swagger pour l'API
-  Possibilité de tourner une **vidéo de démonstration**

▼ Gestion de projet

Gestion des risques

Risque identifié	Plan de mitigation
 Perte de connectivité 4G	Retry automatique + logs + timeout sur l'ESP32
 Données perdues ou mal envoyées	Validation CBOR + schéma + backups MongoDB
 Charge serveur élevée (B2B)	Cache via Memcached
 Problème de sécurité ou faille API	Authentification JWT, validation des entrées, tests de sécurité
 Retard dans le dev React ou déploiement	Répartir le focus / aider ponctuellement entre coéquipiers
 Puce SIM ou GNSS défectueuse	Avoir un second module pour basculer rapidement

Suivi et méthodes

-  Méthode agile : avancement par blocs fonctionnels avec itérations rapides
-  Validation par tests unitaires
-  JIRA

The screenshot shows a Jira Kanban board for the project "Shinkansen.dev". The board is organized into four columns: "À FAIRE 16", "EN COURS 5", "CODE REVIEW", and "FAIT 11".

- À FAIRE 16:** Contains 16 items, including:
 - "ccs-7 Développement bas niveau C/C++ (10 tickets)"
 - "Tests unitaires" (CCS-21)
 - "Ajouter capteur (Bonus)" (CCS-33)
 - "Tests fonctionnels" (CCS-36)
 - "Documentation" (CCS-48)
- EN COURS 5:** Contains 5 items, including:
 - "Erreur de récupération de données GNSS à partir d'un certain nombre de tentatives" (CCS-47)
- CODE REVIEW:** Contains 1 item: "Établir une communication TCP entre l'application C++ et le serveur tcp" (CCS-12).
- FAIT 11:** Contains 11 items, including:
 - "Implémenter la récupération des données GPS depuis l'ESP32" (CCS-11)
 - "Structure de la machine" (CCS-22) - has a "Quickstart" button.

The sidebar on the left provides navigation links for the project, including "Tableau", "Calendrier", "Liste", "Formulaires", "Objectifs", "Tous les tickets", and "Ajouter une vue". It also indicates that the project is in "Développement BÉTA".