## **I. Contexte et Objectifs du Projet**

**Objectif global :**  
Développer une application Shiny (en utilisant shiny.semantic) qui, à partir d’une base volumineuse de données AIS (3 102 887 observations et 20 variables), permet de :  
• Permettre à l’utilisateur de sélectionner un type de navire et un navire spécifique.  
• Calculer la distance maximale parcourue entre deux observations consécutives pour le navire choisi (en sélectionnant la plus récente en cas d’égalité).  
• Afficher ce résultat sur une carte interactive (avec leaflet) et présenter un résumé détaillant la distance parcourue en mètres.  
• Utiliser des techniques de calcul parallèle pour accélérer le traitement, permettant de gérer efficacement la grande taille de la base de données, et mettre en exergue ce concept dans l’interface (par exemple, afficher le nombre de cœurs disponibles, permettre à l’utilisateur de choisir le nombre de threads à utiliser, et afficher des statistiques de performance).  
• Assurer une qualité de code élevée à travers la modularisation du code, une bonne gestion des erreurs, l'implémentation de tests unitaires via testthat pour valider chaque fonction, et la documentation appropriée du code.  
• Déployer l’application sur shinyapps.io pour rendre l'application accessible à un public plus large, et publier le code source sur GitHub.

## **II. Préparation de l’Environnement et de la Base de Données**

### ****1. Installation et Configuration des Packages****

**But :**

* Identifier et installer les packages nécessaires (lecture de données, manipulation, Shiny, shiny.semantic, leaflet, parallélisation, tests, etc.).
* Configurer le répertoire de travail et le chemin d’accès à la base.

**Justification :**  
Une bonne configuration garantit que toutes les fonctionnalités seront disponibles et que l’environnement est homogène pour le développement et le déploiement.

### ****2. Importation de la Base de Données****

**But :**

* Charger la base (fichier CSV) dans R en utilisant le chemin absolu ou relatif.

**Justification :**  
Cela permet de disposer de toutes les observations et variables nécessaires pour explorer, nettoyer et traiter les données.

### ****3. Exploration et Nettoyage des Données****

**Étapes :**

* **Affichage et compréhension de la structure :**  
  Utiliser des fonctions comme head(), str(), colnames(), et summary() pour examiner les données.
* **Vérification des variables essentielles :**  
  S’assurer que les colonnes indispensables (LAT, LON, DATETIME, SPEED, etc.) sont présentes et correctement formatées (ex. convertir DATETIME en POSIXct).
* **Gestion des valeurs manquantes :**  
  Vérifier la présence de valeurs manquantes et décider si des imputations ou suppressions sont nécessaires.  
  Dans notre cas, les variables essentielles sont complètes.

**Justification :**  
Une exploration approfondie permet de garantir la qualité des données utilisées pour le calcul et d’éviter des erreurs lors de l’analyse.

## **III. Calcul de la Distance Entre Observations**

### ****4. Définir la Méthode de Calcul de Distance****

**But :**

* Choisir et implémenter la formule de Haversine pour calculer la distance en mètres entre deux points (latitude/longitude).

**Justification :**  
La formule de Haversine est la méthode standard pour obtenir une distance réaliste sur une sphère, ce qui est crucial pour obtenir des résultats précis dans le contexte maritime.

### ****5. Application du Calcul de Distance sur la Base****

**Étapes :**

* **Trier les observations par navire et par date/heure :**  
  Garantir que les calculs se font dans l’ordre chronologique.
* **Calculer la distance entre chaque paire d’observations successives pour chaque navire :**  
  Utiliser la fonction Haversine et la fonction lead() pour obtenir la position suivante.
* **Identifier la paire avec la distance maximale pour chaque navire :**  
  Si plusieurs paires ont la même distance, choisir la plus récente.

**Justification :**  
Cela permet d’extraire le trajet le plus significatif pour chaque navire, information centrale pour l’analyse et l’affichage.

## **IV. Mise en Place du Calcul Parallèle**

### ****6. Justification du Calcul Parallèle****

**But :**

* Expliquer que le traitement séquentiel des 3 millions d’observations est coûteux en temps, et que le parallélisme permet de diviser le travail en traitant chaque navire (ou groupe de navires) simultanément.

**Justification :**  
Le parallélisme permet une réduction significative du temps de calcul en exploitant la puissance des multi-cœurs. C’est une notion clé que le projet doit mettre en lumière.

### ****7. Implémentation du Calcul Parallèle****

**Étapes :**

* **Identifier les tâches indépendantes :**  
  Chaque navire peut être traité séparément puisque les calculs de distance pour un navire ne dépendent pas des autres.
* **Utiliser des packages de parallélisation :**  
  Par exemple, foreach et doParallel (ou le package parallel) pour répartir les calculs sur plusieurs cœurs.
* **Fusionner les résultats :**  
  Combiner les sorties de chaque thread pour obtenir la base complète avec les distances calculées.

**Justification :**  
Cette approche garantit que le calcul est effectué de manière efficace et met en avant l’avantage du parallélisme, qui est un point central de l’exposé.

### ****8. Intégrer des Options Interactives pour le Parallélisme****

**Étapes :**

* **Afficher le nombre de cœurs disponibles :**  
  Utiliser parallel::detectCores() pour informer l’utilisateur.
* **Permettre le choix du nombre de threads à utiliser :**  
  Ajouter un contrôle (sliderInput ou numericInput) pour que l’utilisateur puisse définir le nombre de threads, ce qui configure dynamiquement le cluster (via makeCluster()).
* **(Optionnel) Comparer le temps d’exécution séquentiel vs parallèle :**  
  Mesurer et afficher le temps de calcul pour montrer l’impact du parallélisme (par exemple, “Calcul réalisé en X secondes avec Y threads”).
* **Visualisation de la charge de travail et journalisation :**  
  Mettre en place un indicateur (graphique, barre de progression ou message en temps réel) pour montrer l’avancement des tâches parallèles et éventuellement des messages du type "Thread X a terminé le traitement du navire Y".

**Justification :**  
Ces options pédagogiques permettent de rendre le concept de parallélisme tangible et interactif pour l’utilisateur. Elles offrent également des éléments de comparaison et de diagnostic pour évaluer l’efficacité de la parallélisation.

## **V. Développement de l’Application Shiny**

### ****9. Conception de l’Interface Utilisateur (UI) avec shiny.semantic****

**Composants de l’interface :**

* **Onglet Accueil (Home) :**
  + Introduction générale et explication du projet, y compris une brève description du calcul parallèle.
* **Onglet Paramétrage et Sélection :**
  + Menus déroulants pour choisir le type de navire et le navire spécifique (utilisation de modules pour une meilleure organisation).
  + Section d’information sur le nombre de cœurs disponibles.
  + Contrôle interactif permettant de choisir le nombre de threads pour le calcul parallèle.
  + Bouton pour lancer l’analyse.
* **Onglet Résultats et Visualisation :**
  + Carte interactive (via leaflet) affichant les deux points du trajet maximal (départ et arrivée).
  + Zone textuelle résumant la distance parcourue et d’autres statistiques associées (heure de départ, heure d’arrivée).
* **Onglet Performance & Parallélisme :**
  + Affichage des métriques de performance : temps de calcul, nombre de threads utilisés, speedup factor, etc.
  + Graphique ou barre de progression illustrant l’avancement du calcul parallèle.
  + Explication pédagogique du fonctionnement du parallélisme dans le contexte du projet.
* **Onglet Documentation / À Propos :**
  + Détails sur l’architecture du projet, les choix technologiques, le fonctionnement du calcul parallèle.
  + Liens vers GitHub, shinyapps.io et autres ressources complémentaires.

**Justification :**  
Une interface claire et divisée en onglets permet de segmenter les fonctionnalités et d’offrir à l’utilisateur une navigation intuitive tout en mettant en avant les différents aspects techniques et pédagogiques du projet.

### ****10. Logique Serveur de l’Application****

**Fonctionnalités clés :**

* **Filtrage des données en fonction des sélections de l’utilisateur :**  
  Le serveur doit récupérer le type de navire et le navire sélectionné pour extraire la sous-base correspondante.
* **Application du calcul parallèle :**  
  Lorsque l’utilisateur lance l’analyse, le serveur configure le cluster en fonction du nombre de threads choisi, exécute le calcul parallèle pour déterminer le trajet maximal et renvoie les résultats.
* **Mise à jour dynamique :**  
  Toute modification (choix du navire ou du nombre de threads) déclenche une mise à jour de la carte et du résumé.
* **Affichage des indicateurs de performance et des messages de progression :**  
  Le serveur doit fournir en temps réel ou après traitement les informations de temps de calcul, la répartition des tâches, etc.

**Justification :**  
La logique serveur coordonne le traitement intensif (calcul de distance, parallélisation) et assure la réactivité de l’interface, ce qui est crucial pour démontrer les avantages du calcul parallèle dans un contexte interactif.

## **VI. Déploiement et Partage**

### ****11. Déploiement sur shinyapps.io****

**Objectif :**

* Préparer et mettre en ligne l’application pour qu’elle soit accessible aux utilisateurs et au professeur.

**Justification :**  
Le déploiement sur une plateforme en ligne démontre la finalisation du projet et permet une démonstration en conditions réelles.

### ****12. Publication du Code sur GitHub****

**Objectif :**

* Organiser et documenter le code source, et le partager via GitHub.

**Justification :**  
La publication sur GitHub assure la transparence, permet le suivi des versions et montre la qualité du code via une documentation complète et des tests unitaires.