Déroulement d’algorithme de guidage A* sur une carte

Analyse :

1. Contexte et définition du problème:

Notre projet vise à créer une application web pour la visualisation de cartes et la détermination des itinéraires les plus courts entre les points sélectionnés par l'utilisateur en utilisant l'algorithme A*.

2. Objectifs:

- Créer une interface utilisateur pour la visualisation des cartes en utilisant le framework Leaflet.
- Implémenter l'algorithme A* pour déterminer les itinéraires les plus courts entre les points sélectionnés par l'utilisateur.
- Intégrer une authentification pour permettre à chaque utilisateur de personnaliser l'utilisation de l'application en enregistrant ses emplacements et itinéraires favoris.

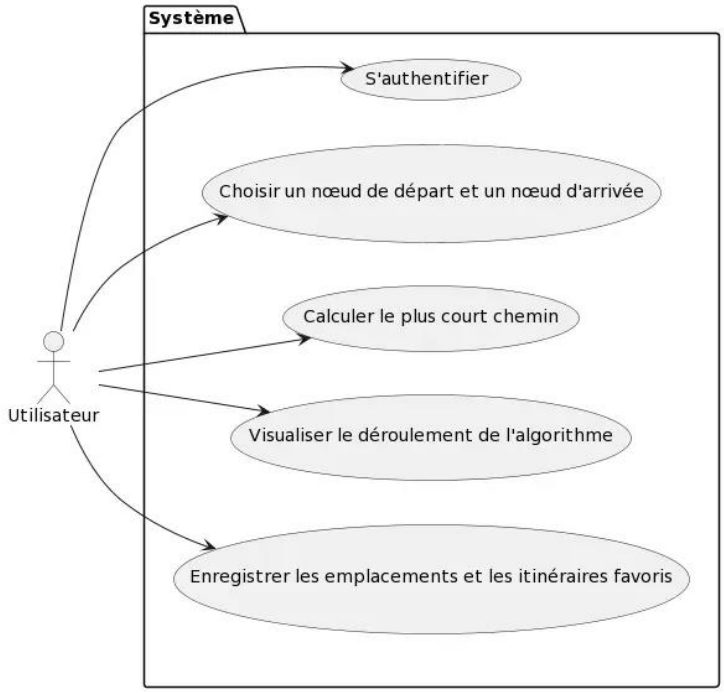
3. Périmètre:

- L'application sera développée en utilisant AngularJS pour la partie front-end et JAVA/SPRING pour la partie back-end.
- La base de données utilisée sera MySQL.
- L'algorithme A* sera utilisé pour déterminer les itinéraires les plus courts.
- La distance de Manhattan sera utilisée pour les calculs de distance entre les points sur la carte.

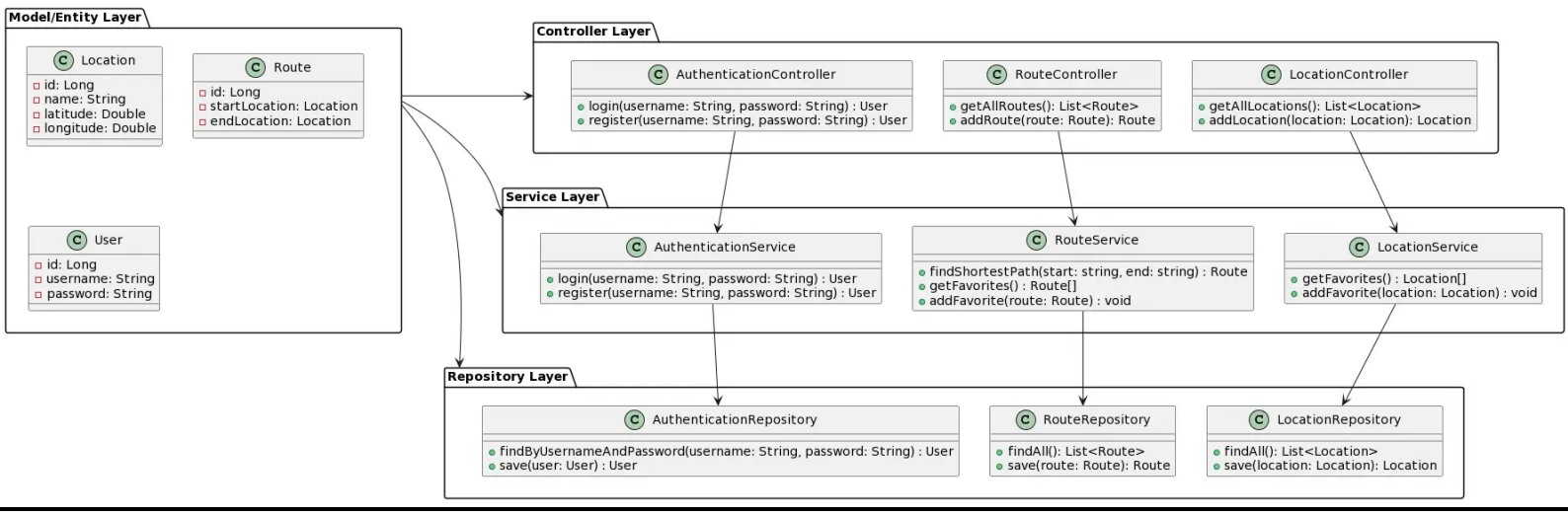
4. Description des besoins:

- Besoins fonctionnels:
    - Interface utilisateur pour la visualisation de cartes en utilisant Leaflet.
    - Possibilité pour l'utilisateur de sélectionner un point de départ et un point d'arrivée sur la carte.
    - Affichage du plus court itinéraire entre les points sélectionnés.
    - Authentification pour permettre à chaque utilisateur de personnaliser l'utilisation de l'application.
    - Enregistrement des emplacements et itinéraires favoris pour chaque utilisateur.
- Besoins non fonctionnels:
    - La vitesse de calcul de l'algorithme A* doit être raisonnable.
    - La sécurité des données utilisateur doit être garantie.
    - La scalabilité de l'application doit être prise en compte pour permettre son utilisation par un grand nombre d'utilisateurs.

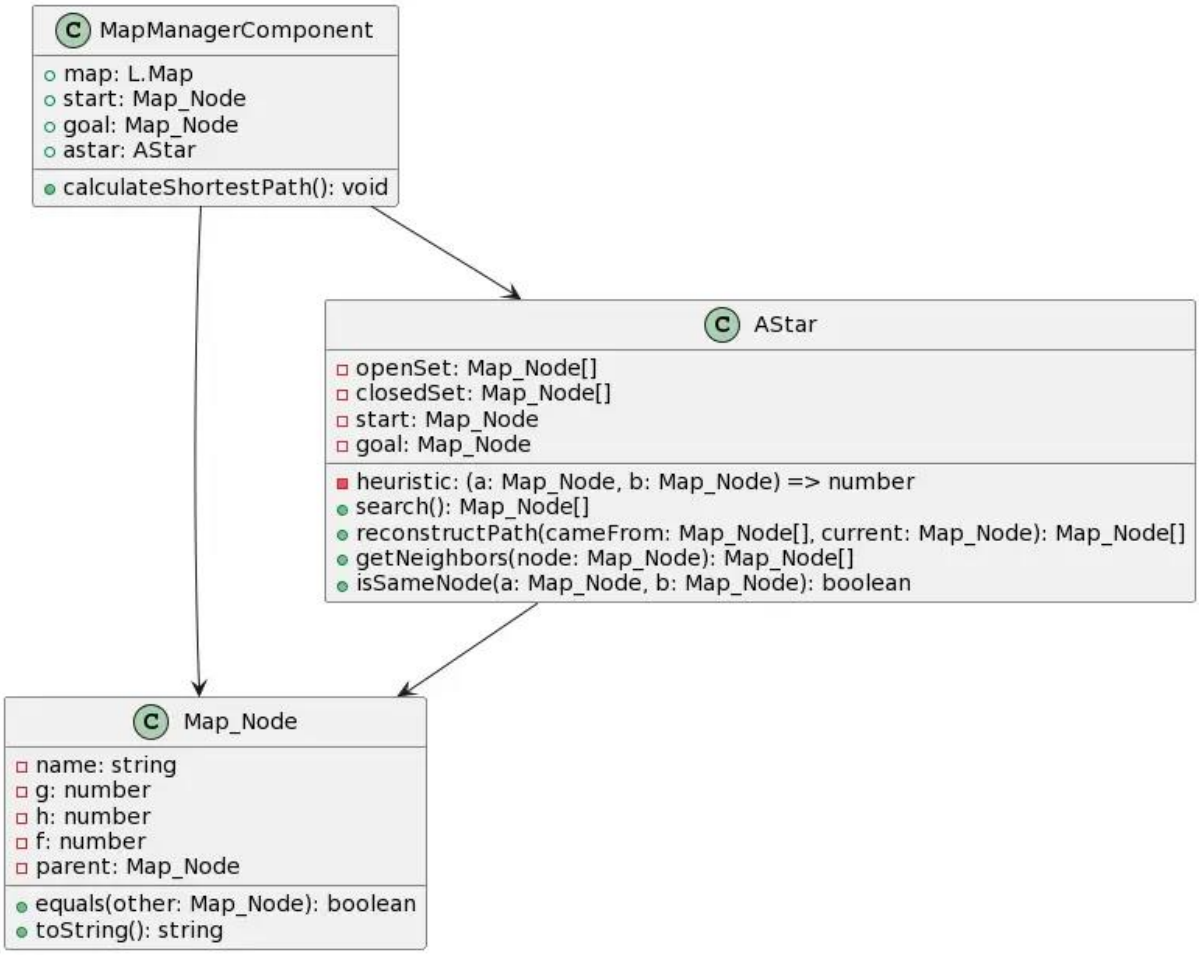
Diagramme de cas d’utilisation :



Architecture de back-end :



Classes front-end :



Choix d’outils techniques :

AngularJS est un framework JavaScript open-source développé et maintenu par Google. Il permet de créer des applications web dynamiques en utilisant le modèle-vue-contrôleur (MVC). Il est connu pour sa facilité d'utilisation, sa flexibilité et sa puissance pour créer des applications complexes. Il permet de structurer efficacement le code, de faciliter la maintenance et de faciliter le développement d'applications à grande échelle. Il a été choisi pour ce projet car il permet de créer une interface utilisateur interactive pour la visualisation des résultats de l'algorithme de plus court chemin sur la carte, en utilisant les fonctionnalités de Leaflet. Il a également un grand écosystème de développeurs et de bibliothèques disponibles pour l'intégration avec d'autres technologies.

Description technique générale :

Le système utilise la librairie Leaflet pour afficher une carte sur l'interface utilisateur, ainsi que pour obtenir les coordonnées de latitude et de longitude des points sélectionnés par l'utilisateur sur la carte.

La classe MapManagerComponent est responsable de l'initialisation de la carte Leaflet et de l'affichage de celle-ci sur l'interface utilisateur. Elle contient également un constructeur qui injecte une instance de la classe Map_Node pour stocker les informations sur les nœuds utilisés par l'algorithme A*.

La classe Map_Node représente un nœud de la carte, contenant des informations telles que la position (latitude et longitude), la distance de déplacement (g) et la distance totale estimée (f) entre ce nœud et les nœuds voisins. Elle est utilisée pour stocker les nœuds dans l'algorithme A*.

La classe AStar est utilisée pour implémenter l'algorithme A* qui calcule le plus court chemin entre les nœuds. Elle prend en entrée une fonction heuristique pour estimer la distance entre les nœuds, qui dans notre cas est la distance de Manhattan [1] entre les nœuds.

La méthode calculateShortestPath de MapManagerComponent utilise l'instance de AStar pour trouver le plus court chemin entre les nœuds de départ et d'arrivée sélectionnés par l'utilisateur sur la carte. Il utilise également des marqueurs de départ et d'arrivée pour visualiser ces points sur la carte.

Enfin, la méthode visualizePath de MapManagerComponent est utilisée pour afficher graphiquement le chemin calculé sur la carte en utilisant des lignes pour relier les nœuds du chemin.

Fonctionnement :

L'algorithme A* est un algorithme de recherche de chemin utilisé pour trouver le plus court chemin entre deux points dans un graphe. Il utilise une heuristique pour évaluer la distance estimée entre le nœud actuel et le nœud final, ainsi qu'une distance déjà parcourue pour évaluer la distance totale entre le nœud actuel et le nœud final.

Dans ce code, nous avons implémenté l'algorithme A* pour trouver le plus court chemin entre deux points sur une carte en utilisant la bibliothèque Leaflet. Nous avons défini une classe Map_Node pour représenter chaque nœud sur la carte, qui contient des informations telles que la position (latitude et longitude) et les nœuds voisins. Nous avons également défini une classe AStar pour gérer les

calculs de l'algorithme, qui prend en entrée une fonction d'heuristique pour évaluer la distance estimée entre les nœuds.

Lorsque l'utilisateur demande à trouver le plus court chemin entre deux points sur la carte, la classe MapManagerComponent est utilisée pour gérer la logique de l'application. Dans le constructeur de cette classe, nous créons une nouvelle instance de AStar en fournissant une fonction d'heuristique qui calcule la distance de Manhattan entre les nœuds.

Ensuite, dans la fonction calculateShortestPath, nous utilisons l'instance d'AStar pour trouver le plus court chemin entre le point de départ et le point d'arrivée en utilisant l'algorithme A*. Nous utilisons également des marqueurs de Leaflet pour afficher visuellement le début et la fin du chemin sur la carte.

Pour visualiser le déroulement de l'algorithme, nous utilisons également des marqueurs de Leaflet pour afficher visuellement les nœuds explorés par l'algorithme.

Une fois que l'algorithme A* a été initialisé, il commence à traiter les nœuds. Il commence par ajouter le nœud de départ à la liste des nœuds à traiter. Ensuite, il boucle à travers la liste des nœuds à traiter jusqu'à ce qu'elle soit vide ou jusqu'à ce qu'il trouve le nœud d'arrivée.

A chaque itération, l'algorithme prend le nœud ayant le plus faible coût F (coût total estimé) et le traite en le marquant comme visité. Il calcule alors le coût G (coût réel) pour chaque nœud voisin non visité en utilisant la distance entre le nœud actuel et le nœud voisin. Il calcule également le coût H (coût estimé) en utilisant la fonction heuristique fournie lors de l'initialisation de l'algorithme. Il utilise ensuite ces coûts pour calculer le coût total F pour chaque nœud voisin.

Si le nœud voisin a déjà été visité, l'algorithme vérifie si le nouveau chemin est plus court que le chemin précédent. Si c'est le cas, il met à jour le chemin pour utiliser le nouveau chemin plus court. Sinon, il ignore ce nœud. Si le nœud voisin n'a pas encore été visité, il est ajouté à la liste des nœuds à traiter et son chemin est enregistré.

Lorsque le nœud d'arrivée est atteint, l'algorithme retourne le chemin en remontant les nœuds depuis le nœud d'arrivée jusqu'au nœud de départ en utilisant les références de parent stockées pour chaque nœud.

En ce qui concerne la visualisation du déroulement de l'algorithme, cela peut être fait en utilisant des marqueurs pour marquer les nœuds visités et en tracant

une ligne entre les nœuds pour montrer le chemin final. Il peut également être utile d'afficher les coûts G, H et F pour chaque nœud pour une meilleure compréhension de l'algorithme.

Choix d'algorithme :

L'algorithme A* est un algorithme de plus court chemin qui a été développé par Peter Hart, Nils Nilsson et Bertram Raphael en 1968. Il est basé sur l'algorithme Dijkstra mais utilise une heuristique pour estimer la distance restante entre le nœud actuel et le nœud objectif. Cette heuristique permet à l'algorithme de "guider" sa recherche vers le nœud objectif, ce qui le rend plus efficace que Dijkstra dans de nombreux cas.

L'algorithme A* est souvent utilisé dans les jeux vidéo pour trouver le chemin le plus court entre deux points pour les personnages non joueurs (PNJ) et dans la robotique pour la navigation autonome. Il est également couramment utilisé pour la planification de trajet dans les systèmes de transport en commun, la planification de tournées de véhicules et dans les systèmes de cartographie pour la génération de cartes routières.

Il est souvent préféré à Dijkstra car il est plus efficace dans les cas où il existe des obstacles et des contraintes de temps. Il est également utile dans les cas où il existe un nœud objectif spécifique, car il peut trouver un chemin plus court vers ce nœud en utilisant une heuristique pour orienter sa recherche.

[1] : La distance de Manhattan, également appelée distance de L1 ou distance rectiligne, est une mesure de distance entre deux points dans un espace à coordonnées cartésiennes. Elle est définie comme la somme des différences absolues des coordonnées de ces deux points. Pour deux points (x1, y1) et (x2, y2) dans un espace à deux dimensions, la distance de Manhattan est calculée comme : |x1 - x2| + |y1 - y2|. Dans un espace à plus de deux dimensions, la

formule est similaire, avec la somme des différences absolues de chaque coordonnée.

Références bibliographiques :

- " A* with Bounded Costs " par Brian Logan, Natasha Alechina, disponible à cette adresse: https://www.aaai.org/Papers/AAAI/1998/AAAI98-063.pdf
- "An Analysis of the A* Algorithm" par Sven Koenig, Maxim Likhachev disponible à cette adresse : https://www.aaai.org/Papers/Workshops/2006/WS-06-11/WS06-11-010.pdf
- https://fr.wikipedia.org/wiki/Algorithme_A*
- https://en.wikipedia.org/wiki/A*_search_algorithm
- https://fr.wikipedia.org/wiki/Distance_de_Manhattan