Cahier des Charges

Présentation

Ce « *Cahier des Charges projet* » conclut l'apprentissage du langage ECMAScript 2018 et du TypeScript à travers un projet dont les objectifs pédagogiques sont :

- Mettre en pratique et développer ses connaissances algorithmiques ;
- Gérer la mémoire et se prémunir des fuites de mémoire ;
- Mettre en œuvre et utiliser le typage et la transpilation ;
- Mettre en œuvre et utiliser des structures objet ;
- Mettre en œuvre et utiliser des *design pattern* ;
- Utiliser et manipuler le BOM et le DOM ;
- Gérer les évènements ;
- Combiner les connaissances traitées en cours pour créer un programme original.

Il s'agit de créer un **outil de calcul du plus court chemin** entre 2 villes.

Description

Principe

Vous devez réaliser un programme qui sera un outil de calcul du plus court chemin entre 2 villes françaises.

Cet outil doit permettre de déterminer par quelles villes intermédiaires il faut passer pour minimiser la distance à parcourir entre la ville de départ et la ville d'arrivée. L'algorithme à appliquer pour résoudre ce problème est l'algorithme de Dijkstra.

L'outil doit se présenter sous la forme d'un document HTML sur lequel on retrouvera :

- Une liste de sélection pour choisir la ville de départ ;
- Une liste de sélection pour choisir la ville d'arrivée ;
- Un bouton pour lancer le calcul du plus court chemin ;
- Une carte de France sur laquelle sera affichée une représentation de chaque ville (l'échelle distance entre chaque ville sera de 1px pour 1km) ;
- Une zone d'affichage dans laquelle s'affichera la solution trouvée sous forme textuelle et sur la carte une représentation graphique de cette solution (sous la forme de lignes reliant les villes correspondant à la solution trouvée).

L'affichage de l'outil doit être entièrement généré par votre code. Le document HTML permettant le chargement et l'exécution de votre code doit être neutre.

Vous disposez de données indiquant les distances entre un échantillon de 27 villes françaises sous la forme de 27 fichiers au format JSON. Attention les données n'indiquent pas quelles villes sont voisines de quelles villes.

Fonctionnalités

1. **Prepare Dataset** (Préparation du jeu de données) : Charger et structurer les données fournies sous la forme d'une arborescence d'objets que vous pourrez exploiter par la suite.
2. **Generate Form** (Génération de Formulaire) : Créer une zone délimitée de l'affichage qui sera le formulaire contenant la liste de sélection d'une ville de départ, d'arrivée et la validation du choix. L'utilisateur doit pouvoir sélectionner une ville d'arrivée, de départ et valider son choix.
3. **Generate Map** (Génération de la Carte) : Créer une zone délimité de l'affichage qui sera une carte de France sur laquelle on retrouvera les villes françaises correctement positionnées. L'échelle de distance entre les villes sera de 1px pour 1km.
4. **Calculate Shortest Path** (Calculer le Plus Court Chemin) : Implémenter l'algorithme de Dijkstra pour calculer le plus court chemin entre la ville de départ et la ville d'arrivée sélectionnée.
5. **Show Shortest Path** (Montrer le Plus Court Chemin) : Créer une zone délimitée de l'affichage qui contiendra un affichage textuel du plus court chemin. Ajouter une représentation visuelle du plus court chemin sur la carte sous la forme de vecteurs reliant les villes intermédiaires entre la ville de départ et d'arrivée.

Obligations techniques

Gestion du Code Source

Vous devrez utiliser GIT pour gérer votre code source. Vous proposerez un dépôt distant sur un compte privé ou public chez un fournisseur de service tel que GitHUB ou BitBucket ou sur un système auto-hébergé tel que GitLab Community Edition.

En s'inspirant de la méthodologie Git Flow, vous devez créer 5 types de branches :

- Une branche **master** pour les versions définitives. Chaque version devra être taguée avec un numéro de version. C'est sur cette branche que vous fusionnerez vos **releases** ou vos **hotfix**;
- Une branche **release** pour les versions de recette. C'est sur cette branche que vous fusionnerez vos versions de développement stables issues de **develop**.
- Une branche **develop** pour les versions en cours de développement. C'est sur cette branche que vous fusionnerez vos **features** ;
- Plusieurs branches **feature** pour chaque fonctionnalité détaillée plus haut. Par exemple :
    - Une branche **feature/prepare-dataset** ;
    - Une branche **feature/generate-form** ;
    - Une branche **feature/generate-map** ;
    - Une branche **feature/calculate-shortest-path** ;
    - Une branche **feature/show-shortest-path** ;
- Une branche **hotfix** sur laquelle vous corrigerez les bugs des versions de la branche **master**.

Vous veillerez à effectuer des commit régulièrement et à produire des messages de commit suffisamment descriptifs de votre travail.

Ergonomie et graphisme

La représentation graphique de la carte et des villes est sans importance. Cependant :

- Le formulaire devra se situer en haut à gauche du document et à gauche de la carte ;

- La zone d'affichage des résultats, sous forme textuelle, se situera sous le formulaire (toujours à gauche de la carte).
- La carte devra s'afficher dans une balise <canvas>. Cette dernière sera construite par couches (3 couches). La première couche sera une carte de France neutre (sans reliefs et sans informations particulières). La deuxième couche sera celle sur laquelle seront représentées les villes. La troisième couche sera celle sur laquelle seront représentés les chemins (sous la forme de droites).
- Les villes peuvent être représentés par des carrés ou des ronds. Les villes doivent être visibles et leur nom doit être indiqué en dessous/dessus.
- Les résultats, sous forme graphique, seront des vecteurs reliant les villes. A côté de chaque vecteurs, on retrouvera la distance correspondante indiquée sous forme textuelle.

Outils de développement

L'ensemble du site sera entièrement développé en HTML5, CSS3 et **TypeScript**. Le document Web sera valide W3C.

Vous pouvez utiliser l'IDE de votre choix.

Vous ne devez utiliser **aucun** framework ou librairies.

Compatibilité

L'outil sera entièrement fonctionnel sur tous les navigateurs compatibles HTML5, c'est-à-dire : Edge dernière version, Firefox dernière version, Chrome dernière version, Safari dernière version et Opera dernière version.

La compatibilité avec les appareils mobiles n'est pas obligatoire.

Recommandations

Vous veillerez à bien organiser votre code pour améliorer sa présentation (indentation, sauts de ligne, choix de noms explicites). Pour ce faire, vous soignerez particulièrement les commentaires.

Vous n'hésiterez pas à scinder votre code en plusieurs fichiers.

Préalablement à la réalisation du site, vous élaborerez l'organisation de votre code.

En TypeScript, typez au maximum toutes les variables que vous créez. Pensez à privilégier l'usage d'objets plutôt que de nombreuses variables. Vous limiterez l'usage des variables globales en usant préférentiellement de variables locales.

Aides

Pour la partie visuelle :

La carte doit être représentée sous la forme d'un élément HTML de type <canvas>.

Pour l'organisation du code :

Vous devez créer au moins 7 « classes » : **Ville, Route, Carte, Coordonnées, Formulaire, Dijkstra et Outil**.

Vous devez créer une « classe » pour une **ville** :

- Une ville contient :

    - Son nom ;
    - Une référence à la **carte** qui la contient ;
    - Une référence à ses **coordonnées** sur le canvas ;
    - Une référence au tableau des **route**s vers les **ville**s voisines ;
    - … et éventuellement d'autres propriétés.
- Les méthodes d'instance (méthodes du prototype) d'une ville sont :
    - Un méthode qui prend en entrée un contexte et qui dessine la ville dans ce contexte (en utilisant les coordonnées de la ville).
    - … et éventuellement d'autres méthodes.

Vous devez créer une « classe » pour une **route** :

- Une route contient :
    - Sa distance ;
    - Une référence à la **ville** d'arrivée ;
    - Une référence à la **ville** de départ ;
    - … et éventuellement d'autres propriétés.
- Les méthodes d'instance (méthodes du prototype) d'une route sont :
    - Un méthode qui prend en entrée un contexte et qui dessine la route dans ce contexte (en utilisant les coordonnées de la ville de d'arrivée et de la ville de départ).
    - … et éventuellement d'autres méthodes.

Vous devez créer une « classe » pour une **coordonnée** :

- Une coordonnées contient :
    - Sa position en x – left – sur le canvas ;
    - Sa position en y – left – sur le canvas ;
    - … et éventuellement d'autres propriétés.
- Les méthodes d'instance (méthodes du prototype) d'une coordonnée sont :
    - … éventuellement d'autres méthodes.

Vous devez créer une « classe » pour la **carte** :

- Une carte contient :
    - Sa largeur ;
    - Sa hauteur ;
    - L'URL de l'image de fond ;
    - Une référence à l'élément HTML canvas correspondant ;
    - Une référence au tableau des **ville**s qu'elle contient ;
    - … et éventuellement d'autres propriétés.
- Les méthodes d'instance (méthodes du prototype) d'une carte sont :
    - Un méthode qui initialise la carte en :
        - Initialisant le tableau des villes qu'elle contient ;
        - Créant la balise canvas correspondant à la carte ;
        - Dessinant dans la balise canvas la carte, les villes et éventuellement les routes ;
    - … et éventuellement d'autres méthodes.

Vous devez créer une « classe » pour le **formulaire** :

- Un formulaire contient :

    - Une référence à la carte à laquelle il s'applique ;
    - Une référence à l'élément HTML form correspondant ;
    - Une référence à l'élément HTML select correspondant à la première liste ;
    - Une référence à l'élément HTML select correspondant à la deuxième liste ;
    - … et éventuellement d'autres propriétés.
- Les méthodes d'instance (méthodes du prototype) d'un formulaire sont :
    - Une méthode qui initialise le formulaire en :
        - Créant et Initialisant une **carte** ;
        - Créant les éléments HTML correspondant au formulaire ;
        - Déclarant les gestionnaires d'événements relatifs aux choix de l'utilisateur ;
    - Une méthode qui :
        - Prend en entrée une **ville** de départ et une **ville** d'arrivée ;
        - Initialise et exécute un (algorithme de) **Dijskstra**.
    - … et éventuellement d'autres méthodes.

Vous devez créer une « classe » pour un (algorithme de) **Dijskstra** :

- Le **Dijkstra** contient :
    - Une référence à la **ville de départ** ;
    - Une référence à la **ville d'arrivée** ;
    - Une référence à un tableau des **ville**s intermédiaires ;
    - … et éventuellement d'autres propriétés.
- Les méthodes d'instance (méthodes du prototype) d'un Dijkstra sont :
    - Une méthode qui initialise l'algorithme de Dijkstra ;
    - … et éventuellement d'autres méthodes.

Vous devez créer une « classe » pour l'**outil** :

- L'**outil** contient :
    - Une **carte** ;
    - Un **formulaire** ;
    - … et éventuellement d'autres propriétés.
- Les méthodes d'instance (méthodes du prototype) d'un **outil** sont :
    - Une méthode qui initialise l'outil en :
        - Chargeant et Initialisant l'ensemble des données relatives au **ville**s ;
        - Créant et Initialisant un **formulaire** ;
    - … et éventuellement d'autres méthodes.

Pour les calculs :

L'algorithme de Dijkstra est très documenté sur Internet. C'est un algorithme classique de la programmation pour lequel vous trouverez facilement de nombreuses ressources (y compris en ECMAScript). Pour faciliter la compréhension :

- Le **Graphe** selon Dijkstra est pour nous : la carte.
- Les **Arcs** selon Dijkstra sont pour nous : les routes.
- Les **Nœuds** selon Dijkstra sont pour nous : les villes.
- Les **Poids** selon Dijkstra sont pour nous : les distances.