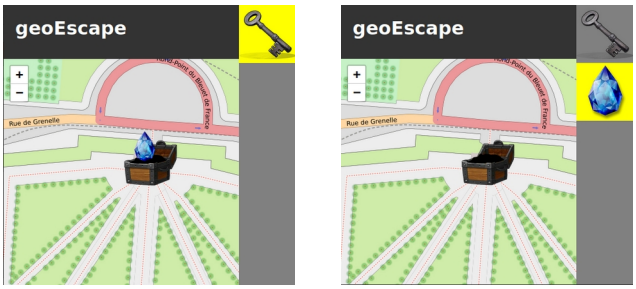
« Escape game » géographique !

L'objectif de ce TP est de créer un « escape game », notamment en résolvant des énigmes et en trouvant des objets. Le tout, sur une carte web.

Pour avoir une idée plus claire, vous trouverez ici (https://tinyurl.com/4a829mw) une vidéo d'un rendu basique (lien valide 1 mois).



Le design de l'application est totalement libre. À vous donc de faire les bons choix en matière d'ergonomie, de rendu graphique, d'interaction, etc. Pensez cette application pour le « grand public », elle ne s'adresse pas à des professionnels. Demandez-vous comment vous aimeriez qu'elle fonctionne. Trouvez-lui un nom.

Vous n'êtes pas contraints pour la technologie serveur (PHP, Python, Node.js, etc.), mais le système de base de données devra être PostgreSQL (avec l'extension PostGIS) et le serveur cartographique sera GeoServer.

Enfin, le plus important, cette application est développée dans le cadre de la validation du cours. **N'en faites pas plus que demandé**. Ne restez pas bloqués trop longtemps sur un problème, venez-nous voir en L308, ou envoyez-nous vos questions par mail (pour des soucis d'installation, de compréhension, de résolution de bugs, etc.).

Livrables attendus

- Tous les fichiers clients/serveurs
  - HTML/CSS/JavaScript
  - Images, Vidéos, Sons, etc.
  - PHP ou autre
- Le dossier complet `workspaces/<votre-workspace>` de votre GeoServer (qui contient vos layers et vos styles)
- Un export SQL de votre base de données
- Un README pour les consignes « d'installation » des différents composants (notamment les numéros de versions)
- les solutions claires des énigmes (position et ordre des objets à trouver)

Pour le rendu, sur clé USB en L308, ou mieux, sur un dépôt Git (GitHub, GitLab, etc.). Ou encore sur depuis votre espace OneDrive ou une solution en ligne.

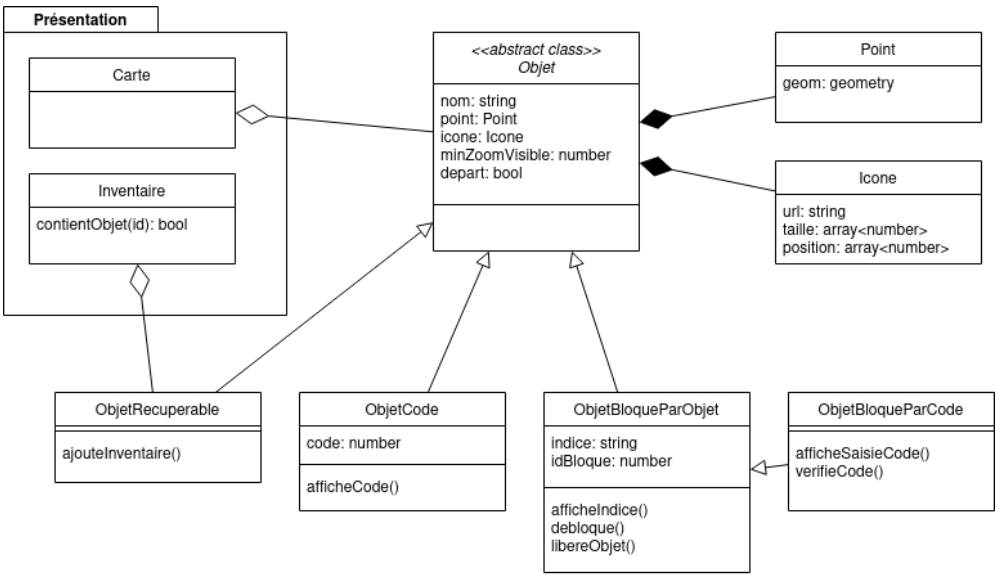
Note : attention au copiés-collés ! Bien que ce soit un travail en groupe, cela ne signifie pas que vos fichiers doivent être identiques entre les groupes, ni même avec les années précédentes. Il est même difficile que ça puisse être le cas. Et vous devez vous douter qu'il est très facile de faire des recherches de similitudes. À bon entendeur, bon travail ! ;)

Type d’objets

Tout d’abord, voici les différents types d’objets que nous souhaitons :

- Objet récupérable
  - Un objet que l’on peut récupérer et conserver dans notre inventaire
- Objet code
  - Un objet qui affiche un code à 4 chiffres
- Objet bloqué par un autre objet
  - Un objet qui nécessite d’avoir le bon objet dans son inventaire pour le débloquer
  - Quand débloqué, libère un autre objet
- Objet bloqué par un code
  - Un objet qui nécessite un code pour le débloquer
  - Quand débloqué, libère un autre objet

Un objet peut être chargé au démarrage du jeu ou non. Un objet doit également avoir un nom, une position géographique, un niveau de zoom minimum à partir duquel il est visible, une icône (image, taille, position de l’ancre). Un objet bloqué peut également avoir un indice pour aider le joueur. Voilà un diagramme de classe représentant le modèle conceptuel des données :



Création des énigmes

Commencez par créer le déroulement de votre jeu (enchaînements des objets, histoire, etc.). Il doit y avoir au minimum un objet de chaque type (donc 4, plus l’objet final), et au maximum 16 objets. Il peut être assez simple au départ, il sera ensuite facile de l’enrichir, sans pour autant modifier la structure de votre code. C’est même l’objectif principal.

Base de données

- Comme mentionné plus haut, votre base de données doit obligatoirement utiliser le système PostgreSQL, avec l’extension PostGIS.
- En partant du modèle proposé, implémentez une structure de base de données pour vos objets (cours UML)
- Remplissez cette base avec vos données

- Les données de position (un point géographique) doivent être stockées sous le type `geometry`, avec le bon SRS.

API

- Créez votre service web. Par exemple :
  - `GET api/objets` renvoie tous les objets qui permettent de commencer le jeu
  - `GET api/objets?id=N` ou `GET api/objets/N` renvoie les informations de l'objet dont l'identifiant est N
- Votre service web/API doit renvoyer les résultats en JSON
  - Vérifier l'encodage des nombres
  - Faites un choix pour la géométrie
- Si vous avez plusieurs tables, faites les bonnes jointures et faites attention aux noms de champs identiques

Rappel: Vous pouvez utiliser n'importe quel langage coté serveur. Un framework doit cependant être utilisé. Pour PHP, choisissez FlightPHP ou Symfony. Pour Python, nous conseillons Flask ou Django. Pour Node.js, c'est Express ou Fastify. Dans tous les cas, si vous n'utilisez pas PHP et FlightPHP, venez d'abord en discuter.

Interface web

La partie cliente doit contenir une carte (Leaflet ou OpenLayers), avec le provider de tuiles que vous souhaitez (OpenStreetMap, Mapbox, etc.) et un inventaire qui permet de stocker les objets récupérés. La librairie Vue.js doit également être utilisée, pour simplifier la mise à jour de la partie HTML du site.

Note : Pour simplifier les interactions entre l'application Vue.js et la carte, il est préférable de créer la carte à l'intérieur de l'application Vue.js. Pour cela, les hooks[1] `created` ou `mounted` peuvent être utilisés.

Pour le commencement, appelez votre API en AJAX pour charger uniquement les objets utiles au départ du jeu (retour JSON). Pour chaque objet, créez un marqueur sur la carte (à la bonne position, avec son icône, et seulement visible en fonction du niveau de zoom)

Pour chaque objet ajouté à la carte, il faut ensuite gérer l'évènement `click` et exécuter les bonnes actions en fonction de son type :

- L'objet est récupérable :
  - On le déplace dans l'inventaire
  - L'objet n'est plus visible sur la carte
- L'objet est un code :
  - On affiche le code
- L'objet est bloqué par un autre objet :
  - On a déjà l'objet qui débloque (l'objet est bien sélectionné dans mon inventaire)
    - L'objet n'est plus bloqué, et libère un nouvel objet
      - On appelle l'API pour récupérer les données de ce nouvel objet (en passant son identifiant)
      - On crée le nouvel objet (marqueur) comme précédemment

1 https://vuejs.org/guide/essentials/lifecycle.html

        - On n’a pas encore l’objet qui débloque
            - On appelle l’API en AJAX pour connaître l’objet bloquant et l’indice associé
- L’objet est bloqué par un code :
    - On appelle l’API en AJAX pour connaître l’objet code et afficher l’indice
    - On crée un formulaire pour saisir le code
    - On vérifie les données envoyées lors de la validation
    - On valide ou non
    - L’objet libère un nouvel objet (idem précédemment)

Serveur cartographique (GeoServer)

Note : Pour l’installation de GeoServer, suivez les tutos officiels (pensez bien à installer JAVA 11 ou 17 avant). Il faut également autoriser l’accès à votre serveur (Jetty) depuis un client avec les CORS (https://tinyurl.com/2yp5kn65)

- Nous voulons créer un flux WMS, contenant une carte de chaleur[2] (ou Heat Map) de nos différents objets. Cela permettra de «tricher» et ainsi découvrir plus facilement où sont cachés les objets.
- Pour cela, dans l’interface de GeoServer :
    - créez d’abord un nouveau Workspace
    - puis un Store, lié à votre base PostGIS
    - puis un layer (la géométrie devrait être utilisée automatiquement)
- Créez ensuite un nouveau style SLD pour générer la carte de chaleur (depuis le menu Styles)
    - Récupérez ce style : https://tinyurl.com/2c7kadea
    - ou adaptez un peu (en fonction de la distance entre vos objets)
- Utilisez la carte de chaleur dans votre interface carto (couche WMS)
    - une case à cocher doit permettre de «tricher», et donc de rendre la couche visible ou non
    - des «bugs» de rendu lié au tuilage peuvent apparaître dans certains cas, ce n’est pas grave

Scores

- À la fin du jeu, sauvegardez le score du joueur dans la base de données. Vous aurez besoin :
    - de son nom/pseudo
    - d’une méthode de calcul d’un score
    - d’une table spécifique
    - de nouvelles routes
- Ajoutez sur votre page d’accueil le « Hall of fame » (le top 10 des meilleurs scores par ex) pour tous les joueurs, généré dès le chargement de la page d’accueil

Bon travail !

2 https://fr.wikipedia.org/wiki/Carte_thermique