TD 7 – Définition des services

TD 7 – Définition et Implémentation des Web Services

Nous allons dans ce TD spécifier les services implémentés dans le cadre de notre site Web. Pour rappel, la liste des fonctionnalités à implémenter est donnée dans l'énoncé du TD1.
Ces services seront programmés en suivant les spécifications REST, et en retournant des résultats JSON. Le travail effectué en TD servira de base pour la programmation de l'API.

7.1 Fondations

7.1.1 Rappel : qu'est-ce qu'une API REST ?

Insister sur le fait que REST est différent de HTTP.
HTTP est un protocole qui spécifie comment un client et un serveur peuvent communiquer.
REST est une spécification, qui s'appuie sur HTTP, et qui permet de dire comment on peut manipuler des entités (ajout avec le méthode PUT de HTTP, modification avec PATCH, suppression avec DELETE, et recherche avec GET). Exemple : on accède à l'utilisateur 1 via monsite.fr/user/1 (GET), *etc.*
Rappel du cours :

- Repose sur une architecture client-serveur
- l'URI est important : connaître l'URI doit suffire pour nommer et identifier une ressource.
- HTTP fournit toutes les opérations nécessaires (GET, POST, PUT et DELETE, essentiellement).
- Chaque opération est auto-suffisante : il n'y a pas d'état.
- Système de couches : chaque composant voit uniquement les composants de la couche avec laquelle il interagit directement
- Utilisation des standards hypermedia : HTML ou XML ou **JSON**

7.1.2 Comment organiser les services selon le principe d'API REST ?

Dresser la liste des services à implémenter, en donnant juste méthode, exemple d'URL et fonction.

Pour un utilisateur :

- afficher l'utilisateur, GET url :par exemple monsite.fr/user/1
- créer l'utilisateur, PUT url : monsite.fr/user
- supprimer l'utilisateur, DELETE l'url : par exemple monsite.fr/user/1

Pour les utilisateurs :

- liste des utilisateurs, GET url : monsite.fr/users

Pour l'authentification :

- créer la connexion, PUT monsite.fr/auth
- supprimer la connexion, DELETE monsite.fr/auth

Pour un message :

- afficher le message, GET url : par exemple monsite.fr/message/1
- créer le message, POST url : monsite.fr/message
- supprimer le message, DELETE url : par exemple monsite.fr/message/1

Pour les messages :

- liste des messages, GET url : monsite.fr/messages

Pour une demande pour devenir membre :

- afficher une demande, GET url : par exemple monsite.fr/demand/12
- supprimer une demande, DELETE url : par exemple monsite.fr/demand/12
- accepter une demande : POST url : par exemple monsite.fr/demand/12

Pour les demandes :

- liste des demandes en cours, GET url : monsite.fr/demands

Cette liste n'est pas exhaustive, votre réalisation pouvant en effet implémenter des fonctionnalités supplémentaires qu'il vous faudra aussi décrire.

7.1.3 Comment tester un service ?

Avec un navigateur, on peut tester uniquement les requêtes avec la méthode GET.
Pour les autres, on ne peut pas tester sur le navigateur (sauf une fois tout le code front réalisé et en utilisant par exemple la bibliothèque Axios). On doit utiliser Postman (GUI) ou curl (CLI).

7.2 Spécification du projet : documentation et architecture

Ce travail de spécification doit être un **travail précis** et il sera important de conserver **une documentation des services implémentés** pour la suite du projet.
Le projet sera organisé de la manière suivante :

- un répertoire `server` qui contient le code lié au serveur
- un répertoire `client` qui contient le code lié au client
- un répertoire `common` qui contient éventuellement le code partagé par le client et le serveur

Le côté serveur sera organisé de la manière suivante (arborescence des répertoires) :

- `package.json` généré lors du "npm init"
- `src/app.js` Contient le code qui définit le serveur
- `src/index.js` Contient le code qui permet de lancer le serveur (listen)
- `src/api.js` Contient le code qui définit l'API
- `src/entities/`, un répertoire qui contiendra les différentes entités à manipuler, par exemple `src/entities/users.js` pour la gestion des utilisateurs. Vous pourrez par exemple définir une classe `Users` qui permet de gérer les utilisateurs avec des méthodes permettant de créer et interroger la base de données (mais vous êtes libres de choisir l'organisation que vous préférez).

Le côté client sera organisé de la manière suivante (arborescence des répertoires) :

- `package.json` et `index.html` générés lors du "npm create vite@latest..."
- `src/main.jsx` généré lors du "npm create vite@latest..."
- `src/components` pour les composants (avec éventuellement des sous-répertoires)
- `src/css` pour les feuilles de style associées à chaque composant
- `public` pour les autres éléments statiques (images, feuille de style globale, éventuellement autres fichiers JavaScript, *etc.*)

Pour chaque service, il faudra spécifier :

Nom du web service	
URL du web service	
Description du service	
Paramètres en entrée	
Format de sortie	
Exemple de sortie	
Erreurs possibles	
Avancement du Service	
Classes/Fichiers JavaScript	
Informations additionnelles	

Décrivez les services dont vous avez dressé la liste. Par exemple, pour la création d'un nouvel utilisateur :

Nom du web service	CreateUser
URL du web service	/user/ avec PUT
Description du service	Permet la création d'un nouvel utilisateur
Paramètres en entrée	login ;password ;password2 ;nom ;prenom ;...
Format de sortie	ok, ou erreur
Exemple de sortie	`{status : XXX, message : "xxxxx", details: "yyyyy" }`, par exemple `{status : 201, message : "Utilisateur créé" }`
Erreurs possibles	requête mal formulée ; paramètres manquants ; password et password2 différents ; serveur base de données inaccessible ; utilisateur déjà existant
Avancement du Service	À faire
Classes/Fichiers JavaScript	Users.exists, Users.create
Informations additionnelles	xxxx

Pas de correction type puisqu'il s'agit là d'un travail à faire par binôme en fonction de leur projet.