zenika
ARCHITECTURE INFORMATIQUE

HTML 5
Avancé

Travaux pratiques

TP1 : XMLHttpRequest

Afin de mettre en œuvre les nouvelles fonctionnalités de l'objet `XMLHttpRequest`, nous allons envoyer un fichier sur le serveur (upload), puis le télécharger (download) entièrement en AJAX. Des barres de progression nous permettront de surveiller la progression des requêtes.

Le fichier envoyé puis téléchargé sera de type image. Le type de réponse AJAX "`blob`" nous permettra de récupérer le contenu de l'image et de l'afficher directement dans la page.

Le code de cet exercice est réparti entre la page `inscription.html` et le script `xhr.js`.

1. Préparation

Pour lancer le serveur basé sur `node.js`, ouvrez une console dans le répertoire "`server`" et lancez la commande suivante :
`node server_file.js`

2. Upload

La page `inscription.html` contient un sélecteur de fichier (`id="avatar"`).

- Complétez la balise de manière à appeler la fonction `sendData()` (fournie dans le script `xhr.js`) lorsqu'un fichier est sélectionné par l'utilisateur

Dans la fonction `sendData()`,

- Préparez un objet `FormData` pour encapsuler le fichier sélectionné
- Utilisez l'API `XmlHttpRequest` pour effectuer un appel de type POST à l'adresse /fileUpload
- Configurez les handlers permettant de réagir aux événements relatifs au processus d'upload ; des squelettes de handlers sont fournis dans le fichier `xhr.js`.

Dans la méthode `uploadProgress()`,

- mettez à jour la barre de progression d'upload (`id="progressUpload"`)
- affichez également le pourcentage sous forme textuelle dans la balise `<span id="progressUpload2">`

Lorsque le fichier a correctement été envoyé sur le serveur, celui-ci renvoie le nom du fichier sous forme de texte. Nous allons nous en servir pour créer un lien hypertexte permettant à l'utilisateur de télécharger l'image (le téléchargement proprement dit sera l'objet de la prochaine section du TP).

Dans la méthode `uploadComplete()` :

- Récupérez le nom du fichier renvoyé par le serveur.
  Astuce : la propriété "`target`" de l'événement passé en paramètre de la méthode représente l'objet XHR courant ; il est donc facile de l'interroger pour récupérer la réponse du serveur.
- Créez un nouveau lien hypertexte dans la balise `<div id="download">`. Pour l'instant, un clic sur ce lien ne fait qu'afficher le nom du fichier dans une popup ; le véritablement téléchargement sera effectué plus loin.

Vérifiez que le code d'upload fonctionne, en sélectionnant un fichier dans le formulaire d'inscription et en constatant la présence du fichier dans le répertoire "`server/uploads`" de votre workspace.

3. Download

Nous allons maintenant autoriser l'utilisateur à télécharger le fichier qu'il vient d'envoyer, grâce au lien que nous venons de mettre en place.

Dans la méthode `uploadComplete()` :

- Au lieu d'afficher le nom du fichier, appelez la méthode `download()` (fournie) en lui passant le nom du fichier à télécharger

Dans la méthode `download()` :

- Préparez un objet `XMLHttpRequest` pour effectuer une requête `GET` à l'adresse "`/<nom du fichier passé en paramètre>`".
  Attention, cette fois, la réponse sera de type "`blob`" car elle contiendra la représentation binaire de l'image téléchargée.
- Configurez les handlers pour gérer les cas d'erreur
- Monitorez la progression du téléchargement à l'aide de la méthode `downloadProgress()`, qui doit mettre à jour la barre de progression associée (`id="progressDownload"`) ainsi que le `<span id="progressDownload2">`

Lorsque le téléchargement est terminé,

- Créez une nouvelle balise `<img>` dans la zone de prévisualisation (`<div id="previsualisation">`)
- Fixez sa taille à `200px * 160px`
- Le contenu de l'image est donné par une URL virtuelle créée à partir de la réponse AJAX (Astuce: pensez à gérer la compatibilité inter-navigateurs !)

Bonus : vous pouvez supprimer les éventuelles anciennes images en utilisant le code ci-dessous :

```
var previsualisation = document.getElementById("previsualisation");
while (previsualition.hasChildNodes()) {
     previsualition.removeChild(previsualition.lastChild);
}
```

Pour finir, testez le fonctionnement de bout en bout :

- Sélectionnez une image
- Vérifiez sa présence sur le serveur
- Cliquez sur le lien généré pour cette image
- Vérifiez l'apparition d'une miniature de cette image dans la page

Vous pouvez également utiliser les outils de développement de votre navigateur pour tracer les requêtes et réponses AJAX, et vérifier leur contenu.

TP2 : Web messaging

Dans cet exercice, nous allons mettre en œuvre la communication inter-fenêtres à l'aide des API de Web Messaging, qui peuvent prendre deux formes :

- Cross-Document Messaging, pour envoyer des messages à une autre fenêtre
- Channel Messaging, qui établit un canal de communication bidirectionnel

Nous allons faire en sorte que, lorsque notre page d'inscription est ouverte depuis un site partenaire, l'utilisateur bénéficie d'une réduction.

1. Préparation

Pour lancer le serveur "partenaire" basé sur `node.js`, ouvrez une console dans le répertoire "`server`" et lancez la commande suivante :
`node partner_server.js`

Vérifiez que la page partenaire s'ouvre correctement à l'adresse `http://localhost:8081`.

2. Cross-document Messaging

Nous allons tout d'abord travailler sur la page partenaire, afin qu'elle ouvre la page d'inscription en popup

- Dans la page `partner/index.html`, faites en sorte que le bouton ouvre dans une popup la page "`inscription.html`" du serveur de réservation. (Astuce : utilisez la propriété "`target`" de la balise <a>)

Dans la page `inscription.html`, il faut maintenant détecter que nous venons d'une page partenaire, et lui demander un code de réduction.

Dans la page `inscription.html`, complétez la fonction associée à l'événement `window.onload` :

- Testez l'attribut `window.opener` pour déterminer si la page a été ouverte par une autre page
- Envoyez un message à la page ouvrante pour lui demander un code de réduction

Dans la page `partner/index.html`, écoutez les messages entrants et

- Vérifiez leur origine et leur validité
- Renvoyez un code de réduction à l'appelant

Pour finir, dans la page `inscription.html`, écoutez les messages entrants et

- Vérifiez leur origine et leur validité
- Appliquez une réduction de 20% au prix unitaire des tickets

Vérifiez le bon fonctionnement de l'ensemble :

- Ouvrez la page partenaire et lancez la page d'inscription
- Sur la page d'inscription, vérifiez que la réduction a bien été appliquée en demandant le calcul du prix total (note : désactivez pour le moment la pause de 10 secondes dans le script `calculPrix.js`)

3. Channel Messaging

La communication par Cross-Document Messaging n'est pas idéale ; il faut vérifier en permanence la provenance des messages. Lorsque les pages doivent échanger davantage qu'un simple message, il est préférable de recourir à l'API de communication par canaux bidirectionnels (Channel Messaging).

Dans cet exercice, nous allons utiliser l'API Channel Messaging pour demander et recevoir le code de réduction. L'API Cross-Document Messaging vue précédemment uniquement ne nous servira plus qu'à établir le canal de communication initial.

Rappel : la propriété "`data`" des messages représente leur contenu. Il est possible de s'en servir pour déterminer le type de message entrant, et prendre les actions appropriées.

Dans la page `inscription.html` :

- Si la page a été ouverte depuis un site partenaire, créez un `MessageChannel` et envoyez son 2° port (`port2`) à la page appelante à l'aide d'un message simple (Cross-Document Messaging).

Dans la page `partner/index.html` :

- A la réception d'un port de communication, envoyez un message sur le canal associé pour signaler le bon établissement de la liaison

Dans la page `inscription.html` :

- A la réception de ce message de confirmation, demander un code de réduction, toujours en passant par le canal

Dans la page `partner/index.html` :

- A la réception de la demande de code, envoyer un code de réduction

Enfin, dans la page `inscription.html` :

- A la réception du code, appliquer la réduction au tarif unitaire, et fermer le canal.

TP3 : Server Sent Events

Maintenant que nous savons faire communiquer des fenêtres ou onglets de navigateurs entre eux, intéressons-nous aux messages envoyés depuis un serveur vers un client web.

Dans cet exercice, nous allons utiliser le mécanisme des Server-Sent Events (SSE) pour afficher sur la page d'accueil une liste de tweets, en temps réel.
Pour des raisons pratiques, nous ne nous brancherons pas sur le véritable flux de messages de Twitter ; une page d'administration (`http://localhost:8080/admin`) nous permettra de générer de faux messages à la demande.

1. Côté serveur

Le script "`server/server_file.js`" contient déjà un squelette d'implémentation de la partie serveur.

Avant de continuer, il est important de comprendre les points de design suivants :

- L'objet `Subscribers` contient la liste des clients connectés au flux. Chaque client est représenté non pas par un objet, mais par une fonction qui permet de lui transmettre les tweets sur le canal SSE.
- La fonction qui répond à l'URL "`/admin`" gère le formulaire de génération des tweets sur la page d'administration. Elle extrait les paramètres de la requête, puis appelle la méthode "`Subscribers.notify()`" qui dispatche le tweet à tous les clients enregistrés.
- La fonction (à compléter) qui répond à l'URL "`/tweets`" est chargée d'établir le canal SSE avec le client, et de l'enregistrer auprès de l'objet `Subscribers`.

Dans la méthode gérant l'URL "`/tweets`" :

- Etablissez le canal SSE avec le client en lui renvoyant une réponse HTTP possédant les entêtes adéquats
- Définissez une fonction locale "`onTweet(user,tweet)`" qui représente le moyen d'envoyer un tweet à ce client particulier, sous la forme d'une structure JSON de la forme : `({"user":<user>, "tweet": <tweet>})`.
- Enregistrez cette fonction auprès de l'objet `Subscribers`
- Mettez en place un handler pour gérer la déconnexion du client et supprimer la fonction dans l'objet `Subscriber`.

Relancez le serveur pour prendre en compte ces modifications.

2. Côté client

Dans la page "`/tweets`" :

- Au chargement de la page (`window.onload`), initialisez un canal SSE en provenance de l'URL "`/tweets`"
- A la réception d'un message, appelez la fonction `addTweet(message)` (fournie) pour en afficher le contenu sur la page.

Pour tester, ouvrez la page de génération des tweets dans une nouvelle fenêtre ou onglet, à l'adresse `http://localhost:8080/admin`
A la soumission du formulaire, vérifiez que le tweet a bien été reçu par toutes les fenêtres affichant la page d'accueil de l'application.

TP4 : WebSocket

Les Server-Sent Events (SSE) ne permettent qu'une communication du serveur vers les clients ; lorsqu'une communication bidirectionnelle est nécessaire, les WebSockets (WS) s'imposent !

Dans cet exercice, nous allons développer un compteur de places disponibles pour chaque conférence.
Résultat attendu : lorsqu'un utilisateur sélectionne une conférence par drag'n'drop, un message est envoyé au serveur, qui décrémente le nombre de places restantes puis avertit en temps réel tous les clients connectés. Inversement, le fait de désélectionner une conférence libère immédiatement une place.

Les messages échangés seront au format suivant :

- Client → Serveur : "`<id conf>:<opération>`"
  Avec opération :
  "`-`" : décrémenter le nombre de places disponibles (réservation)
  "`+`" : incrémenter le nombre de places disponibles (annulation)
- Serveur → Client : "`<id conf>:<nb places>`"

Rappel : les ID des conférences vont de 1 à 12

1. Préparation

Pour lancer le serveur compatible WebSockets basé sur `node.js`, ouvrez une console dans le répertoire "`server`" et lancez la commande suivante :
`node ws_server.js`

Notez que nous aurons toujours besoin du serveur "`server_file.js`" pour faire fonctionner la partie cliente de l'application.

2. Côté serveur

Le nombre de places disponibles par conférence est maintenant géré côté serveur, il nous faut donc une structure (ex: tableau) pour stocker l'information.

Lors de la première connexion d'un client, nous devons lui envoyer le nombre de places disponibles pour chaque conférence, afin qu'il initialise l'affichage de ses compteurs de places.

Egalement, lorsqu'un client signale la réservation ou l'annulation d'une place pour une conférence précise, le nombre de places de cette conférence sera mis à jour côté serveur puis envoyé unitairement à tous les clients.

Dans le script "`server/ws_server.js`" :

- Déclarez une structure pour stocker le nombre de places disponibles par conférence.
  Rappel : les IDs des conférences vont de 1 à 12 (cf. `scripts/loadData.js`).
- Lors de la connexion d'un client, lui envoyer le nombre de places disponibles par conférence (un message par conférence).
- Complétez le handler réagissant aux messages WS en provenance des clients, afin d'ajuster le nombre de places disponibles, et de transmettre la nouvelle valeur à tous les clients.

3. Côté client

Côté client, il nous faut tout d'abord un emplacement pour afficher le nombre de places disponibles pour chaque conférence.

Dans la page "`agenda-edit.html`" :

- Ajoutez un `<span>` à chaque conférence pour afficher le nombre de places

Ensuite, il faut établir une connexion WS avec le serveur.

Dans la page "`agenda-edit.html`" :

- Appelez la fonction `loadWebSocket()` au chargement de la page

Dans le script "`scripts/ws.js`" :

- Etudiez le script, notamment la fonction `displayMessage()` qui traite les messages en provenance du serveur

Dans le script "`scripts/dragDrop.js`" :

- Lors du "drop" d'une conférence, envoyez un message au serveur pour signaler la réservation ou la libération d'une place.

Pour finir, ouvrez plusieurs onglets ou navigateurs, et vérifiez que la réservation ou la libération d'une place à une conférence impacte immédiatement le nombre de places disponibles pour tous les clients.

TP5 : WebRTC

La dernière arrivée des API de communication de HTML5 est WebRTC. Cette API permet a 2 clients (peers) de communiquer directement sans passser transiter en permanence par un serveur. Le serveur sert au départ pour mettre en relation les les peer et éventuellement lors des échecs de connexion.

Note importante : WebRTC est encore très mal supporté et toutes les composantes de l'API ne sont pas également implémentées dans les navigateurs. A noter :
- addStream doit être appelé AVANT de définir les localDescription
- activer « sctp datachannel » dans chrome:flags
- les data channel fonctionne mal (voire pas) entre navigateurs différents

Dans cet exercice, nous allons voir comment établir une communication audio et vidéo avec un autre peer puis nous verrons comment envoyer des messages entre les 2 peers.

L'API étant encore préfixée sur tous les navigateurs l'implémentant, nous utiliserons un polyfill pour simplifier cette partie. Polyfill-webrtc.js
L'API requiert l'existence d'un SignalChannel pour communiquer entre les peers pour l'initialisation de la communication. Ce système de communication est déjà implémenté et se trouve dans signal-channel.js.

Avant de commencer, regardez l'implémentation du SignalChannel et du fichier serveur signaling-server.js pour bien comprendre le fonctionnement.

Suivre les TODO numérotés pour effectuer les éléments dans le bon ordre.

1. Préparation

Pour lancer le serveur servant pour le signal channel, ouvrez une console dans le répertoire "`server`" et lancez la commande suivante :
```
node signaling-server.js
```

2. Communication audio et vidéo

L'ouverture d'une communcation vidéo se fait en plusieurs étapes :
- Demander l'utilisation de la webcam et/ou du micro
- Ajouter le stream vidéo à la connexion
- Echanger nos informations de connexion avec le peer distant :
  - Envoyer une Offer au peer distant et attendre sa réponse (l'offer doit être assignée comme localDescription)

    - Recevoir l'Answer du peer distant et attendre son stream vidéo (l'offer doit être assignée comme remoteDescription et l'answer comme localDescription)
    - Dans les 2 cas, écouter les ice candidate générés par le navigateur et les envoyer à l'autre peer.

3. DataChannel

L'ouverture d'un DataChannel entre 2 peers fonctionne de la même manière :

- Créer un DataChannel sur la connexion
- (Ré)échanger nos informations de connexion avec le peer distant

TP6 : Web Workers

Pour finir, nous allons mettre en place les Web Workers, qui permettent de lancer des traitements asynchrones en Javascript.

Dans le cadre de l'exercice, nous considérerons que le prix total d'une conférence est très coûteux à calculer – nous matérialiserons cela sous la forme d'un délai artificiel de 10 secondes dans la fonction de calcul.

Note: si vous l'aviez désactivée dans un précédent exercice, rétablissez dès maintenant la pause dans la fonction `calcul()` du script `scripts/calculPrix.js`.

Avant de commencer, ouvrez la page d'inscription, puis demandez le calcul du prix. Constatez un délai de 10 secondes pendant lesquelles tout le navigateur est bloqué.

1. Mise en place du Worker

Premièrement, modifions le comportement du bouton de lancement du calcul.
Dans la page `inscription.html` :

- modifiez le bouton de façon à appeler la fonction `calculPrixWorker()` au lieu de la fonction `calcul()`.

Dans la fonction `calculPrixWorker()` :

- Initialisez un Worker de façon à lui faire exécuter le script `scripts/computeWorker.js`
- Appelez la fonction `desactive()` pour figer les éléments d'interface participant au calcul du prix (nombre de places, nombre de jours)
- Envoyez une demande de calcul au Worker en lui fournissant tous les éléments nécessaires

- Mettez en place un handler pour gérer la réponse du Worker : mettre à jour la zone d'affichage du prix, et réactiver les zones de saisie précédemment figées

Dans le script "`scripts/computeWorker.js`" :

- Copiez la fonction `pausecomp()` depuis le script `calculPrix.js`
- Mettez en place un handler pour gérer les messages en provenance de la page d'inscription : récupération des éléments de tarification, calcul du prix total, et renvoi du résultat. N'oubliez pas d'appeler également `pausecom()` pour simuler un traitement particulièrement coûteux.

Rafraîchissez la page d'inscription, demandez le calcul du prix, et constatez que l'interface reste cette fois réactive pendant le calcul.