TP3 - Concrétisation

Le TP se déroule en **au moins** 4 séances, **avec rendu final attendu** à rendre à votre intervenant.
L'objectif de ce TP est de développer les interfaces conçues en TD.
N'hésitez pas à vous appuyer sur différents guides techniques disponibles sur l'intranet ainsi que sur la documentation.
N'hésitez pas à solliciter votre intervenant !

Mise en situation

Vous avez fini la conception de l'interface de réservation de billets de train. Il est maintenant temps de la développer !
Pour rappel, cette interface doit permettre à un utilisateur :

- de se connecter
- de se déconnecter
- de rechercher un billet de train
- de mettre un billet de train dans son panier
- de visualiser son panier
- de retirer un billet de train de son panier
- de commander les billets de son panier

Un utilisateur n'est pas obligé d'être connecté à un compte pour réaliser les différentes actions.

Utilisez vos personas, scénarios et prototypes pour développer cette interface.
Quelque soit la conception réalisée en TD, il faut que votre interface finale soit **responsive**.

Pour l'échange de données, vous interagirez avec un serveur hébergé sur gigondas. Pour plus de détails, voir la partie "serveur".

Conseils de développement

- Votre interface contient plusieurs "écrans". Découpez donc votre projet en plusieurs pages (connexion, recherche, visualisation des résultats, panier etc…)

- N'oubliez pas de respecter les étapes de développement d'un document : d'abord le HTML de façon **sémantique**, puis le CSS, et enfin le JS pour le dynamisme

- Concentrez-vous **d'abord** sur tous les aspects principaux des pages (navigation, récupération / affichage / envoi de données etc…) et **après** sur l'aspect responsive.

- Vous allez sûrement avoir besoin de stocker des données entre vos pages (pour le panier par exemple). Vous pouvez utiliser l'API sessionStorage en JS. Vous trouverez plus de détails dans le guide technique.

- Il vous faudra probablement changer de page depuis le JS. Vous pouvez faire cela en utilisant window.location (https://developer.mozilla.org/fr/docs/Web/API/window/location)

- Vous pouvez commencer à développer votre interface sans la partie “utilisateur”. Vous pourrez alors rajouter dans un second temps la connexion au compte etc…

- Utilisez un maximum l’outil de développement du navigateur. Il comporte un onglet “réseau” qui vous permet de visualiser toutes les requêtes émises. De plus, vous pouvez tester très simplement vos envois de requêtes depuis la partie console.

- Ne prenez pas peur ! Le sujet de ce TP est un peu long, mais c’est parce qu’il est très détaillé ! Lisez-le une première fois rapidement sans vous attarder sur les détails de l’API. Déroulez les étapes une par une (HTML, puis CSS, puis JS, puis Responsive). Quand vous en serez au JS, pour l’interaction avec le serveur, lisez bien les détails des différentes routes, **testez** (j’insiste !) avec la console, essayez les exemples… Et à n’importe quel moment, si vous êtes perdus, n’hésitez pas à consulter votre intervenant (après avoir relu le sujet, consulté les guides techniques et la documentation, bien entendu !)

- Si vous avez des erreurs bizarres, si les exemples de fetch donnés à la fin ne fonctionnent pas, si le serveur renvoie des résultats incohérents, contactez directement votre intervenant ou le responsable (Nathanaël Spriet).

Interactions avec le serveur

Le serveur se trouve à l’adresse http://gigondas:1111/sprietna/ihm/tp3. Il contient une “pseudo base de données” qui est régulièrement remise à zéro.
Il permet de récupérer les informations des entités suivantes :

- utilisateurs
- villes
- gares
- trajets
- horaires
- billets

Voici les relations entre ces entités :

- Une gare a une référence vers une ville
- Un trajet correspond à une gare de départ, une gare d’arrivée, un temps et un type (TER ou TGV)
- Un horaire correspond à un trajet à une date et une heure de départ
- Un billet correspond à l’achat pour un horaire avec des coordonnées et **optionnellement** un utilisateur

Une entité “tire” toutes les informations dont elle fait référence. Par exemple, quand vous demandez un trajet, vous récupérerez automatiquement toutes les informations des gares de départ et d’arrivée, qui elles même contiendront les informations de leurs villes respectives.

Enfin, il n’y a pas à proprement parler de connexion de l’utilisateur. Les routes existent et permettent de **simuler** une connexion, mais l’état connecté ou non de l’utilisateur n’existe pas.

Interactions

La liste suivante présente le détail de l'API. À la fin se trouvent quelques exemples d'utilisation de fetch.

/login : la connexion

Méthode : **POST**
Paramètres de requêtes : Aucun
Corps (body) :

- mail : chaîne de caractère correspondant au mail d'un utilisateur
- password : chaîne de caractères. Doit correspondre au mail.

Cette route permet de **simuler** une connexion. Il faut spécifier une adresse mail existante et un mot de passe égal à cette adresse mail.

Retourne :

- **200** si l'utilisateur est connecté avec comme message une chaîne de caractère correspondant à l'ID de l'utilisateur
- **404** si l'utilisateur n'est pas trouvé
- **400** si un des paramètres est faux
- **403** si le mot de passe ne correspond pas au mail

/logout : la déconnexion

Méthode : **POST**
Paramètres de requêtes : Aucun
Corps (body) :

- id : nombre correspondant à l'ID de l'utilisateur

Cette route permet de **simuler** une déconnexion. Il faut spécifier l'ID de l'utilisateur.

Retourne :

- **200** si l'utilisateur est déconnecté, sans message
- **404** si l'utilisateur n'est pas trouvé
- **400** si un des paramètres est faux

/users

Méthode : **GET**
Paramètres de requêtes : Aucun

Cette route permet de récupérer tous les utilisateurs au format JSON

Retourne :

- **200** avec comme message le JSON décrivant les utilisateurs

/users/info/:userId

Il faut **remplacer** ":userId" par l'ID de l'utilisateur
Méthode : **GET**
Paramètres de requêtes : Aucun

Cette route permet de récupérer les informations de l'utilisateur au format JSON

Retourne :

- **200** avec comme message le JSON décrivant l'utilisateur
- **404** si l'utilisateur n'est pas trouvé
- **400** si l'identifiant n'est pas précisé

/users/history/:userId

Il faut **remplacer** ":userId" par l'ID de l'utilisateur
Méthode : **GET**
Paramètres de requêtes : Aucun

Cette route permet de récupérer l'historique d'achat de l'utilisateur sous forme de tableau JSON

Retourne :

- **200** avec comme message le JSON contenant les billets de l'utilisateur
- **404** si l'utilisateur n'est pas trouvé
- **400** si l'identifiant n'est pas précisé

/cities

Méthode : **GET**
Paramètres de requêtes : Aucun

Cette route permet de récupérer toutes les villes au format JSON

Retourne :

- **200** avec comme message le JSON décrivant les villes

/stations

Méthode : **GET**
Paramètres de requêtes : Aucun

Cette route permet de récupérer toutes les gares au format JSON

Retourne :

- **200** avec comme message le JSON décrivant les gares

/schedules : la recherche

Méthode : **GET**
Paramètres de requêtes :

- cityFrom : l'identifiant de la ville de départ
- stationFrom : l'identifiant de la gare de départ
- cityTo : l'identifiant de la ville d'arrivée
- stationTo : l'identifiant de la gare d'arrivée
- date : date du trajet
- timeFrom : horaire de départ
- timeTo : horaire d'arrivée

Cette route permet de rechercher les horaires de trajets selon plusieurs paramètres.
Il est **obligatoire** de spécifier une ville ou une gare de départ **et** une ville ou une gare d'arrivée.
Il est **obligatoire** de spécifier une date **au format** "YYYY-MM-DD" (par exemple "2021-03-28" pour le 28 mars 2021).
Si timeFrom est spécifié, seuls les horaires **supérieurs ou égaux** à ce paramètres seront retournés.
Si timeTo est spécifié, seuls les horaires **inférieurs ou égaux** à ce paramètres seront retournés.
Les horaires spécifiés doivent être au **format** "HH:mm" (par exemple "16:25" pour 16h et 25 minutes).

Si aucun horaire n'est trouvé, le tableau retournée sera vide.

Retourne :

- **200** avec comme message le JSON regroupant les horaires
- **400** si un des paramètres obligatoire n'est pas spécifié, ou un paramètre n'est pas du bon format
- **404** si une des gares ou une des villes n'est pas trouvée

/schedules/info/:scheduleId

Il faut **remplacer** ":scheduleId" par l'ID de l'utilisateur
Méthode : **GET**
Paramètres de requêtes : Aucun

Cette route permet de récupérer les informations d'un horaire au format JSON

Retourne :

- **200** avec comme message le JSON décrivant l'utilisateur
- **404** si l'utilisateur n'est pas trouvé
- **400** si l'identifiant n'est pas précisé

/book : la commande

Méthode : **POST**
Paramètres de requêtes : Aucun
Corps (body) :

- info : un objet contenant :
    - address : un objet décrivant l'adresse
    - surname : une chaîne de caractère contenant le nom de famille
    - firstname : une chaîne de caractère contenant le prénom
    - mail : une chaîne de caractère contenant l'adresse mail
    - user : (**optionnel**) identifiant de l'utilisateur
- schedules : Un tableau contenant les horaires

Cette route permet de commander un ou plusieurs billets. Il faut pour cela renseigner les informations du titulaire du billet (dans le champ "info"). Si un utilisateur est connecté, le champ info.user sera renseigné avec son identifiant. Il pourra ainsi le retrouver dans son historique.
Le ou les billets doivent être renseignés dans le tableau schedules avec leurs identifiants. Même si la commande ne comporte qu'un billet, le champ schedules **doit obligatoirement** être un tableau.

Retourne :

- **200** si la commande s'est correctement déroulée. Le message contient un tableau avec les informations des billets créés.
- **404** si l'utilisateur n'est pas trouvé
- **400** si un des paramètres est faux

/ticket/:ticketId

Il faut **remplacer** ":ticketId" par l'ID du ticket
Méthode : **DELETE**
Paramètres de requêtes :

- mail : le mail de l'utilisateur titulaire du billet

Cette route permet d'annuler un billet. Il faut préciser l'identifiant du ticket dans la route et mettre en paramètre le mail de l'utilisateur titulaire du billet.

Retourne :

- **200** si le ticket est annulé, sans message de retour
- **404** si le billet n'est pas trouvé
- **400** si un des paramètres est faux

Petit point sur le format JSON

Le format JSON utilisé majoritairement dans les échanges avec le serveur correspond exactement aux objets JavaScript (JSON signifiant JavaScript Object Notation).
Il permet donc très simplement de transformer des objets créés dans le client pour les envoyer dans une requête au serveur, et inversement.

Exemple de requêtes fetch

Voici quelques exemples de l'utilisation de fetch avec l'API. Pour plus de détails, vous pouvez aller lire le guide technique.
Dans ces exemples, nous nous contentons d'afficher le résultat.

login :

```
fetch('http://gigondas:1111/sprietna/ihm/tp3/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            mail: "sam.etegal@test.com",
            password: "sam.etegal@test.com"
        })
})
.then((response) => {
    if (response.ok) {
        return response.text();
    } else {
        throw response;
    }
})
.then((userId) => {
    console.log(userId);
})
.catch((error) => {
    error.text().then((errorMessage) => {
        console.log('Request Failed : ' + errorMessage);
    });
});
```

Résultat dans la console :
1

Recherche :

Nous allons faire une recherche pour un trajet entre Valence et Lyon le 12 mars 2021 avec une arrivée au maximum à 17h00.

```
fetch('http://gigondas:1111/sprietna/ihm/tp3/schedules?cityFrom=1&cityTo=2&date=2021-03-12&timeTo=17:00').then(response => {
    if (response.ok) {
        return response.json()
    } else {
        throw response
    }
}).then(schedules => {
    console.log(schedules)
}).catch(error => {
    error.text().then(errorMessage => {
        console.log('Request failed : ' + errorMessage);
    });
});
```

La console affichera le tableau contenant les 9 entrées correspondantes.

Commande :

```
fetch('http://gigondas:1111/sprietna/ihm/tp3/book', {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({
        info:{
            address:{
                number: 10,
                street: "rue de la Fontaine",
                city: "Guipy",
                postcode: 58420
            },
            surname: "Zarella",
            firstname: "Maud",
            mail:"maud.zarella@test.com"
        },
        schedules:[633,634,635]
    })
}).then(response => {
    if (response.ok) {
        return response.json()
    } else {
        throw response
    }
}).then(tickets => {
    console.log(tickets)
}).catch(error => {
    error.text().then(errorMessage => {
        console.log('Request failed : ' + errorMessage);
    });
});
```

La console affichera le tableau avec les trois nouveaux billets créés.

Annulation :

```
fetch('http://gigondas:1111/sprietna/ihm/tp3/ticket/13?mail=maud.zarella@test.com', {
    method: 'DELETE',
}).then(response => {
    if (response.ok) {
        console.log('Removed')
    } else {
        throw response
    }
}).catch(error => {

    error.text().then(errorMessage => {
        console.log('Request failed : ' + errorMessage);
    });
});
```

Bonus

Si **tout** est terminé (toutes les fonctionnalités implémentées et affichage responsive), vous pouvez gérer l'historique de l'utilisateur (avec l'appel à l'API /users/history/:userId).

Vous pouvez également réfléchir aux fonctionnalités manquantes (trajets favoris, affichage d'un QR Code pour le billet etc…). Si ces fonctionnalités nécessitent des modifications du serveur, faites-en part au responsable (Nathanaël Spriet) qui se fera une joie de les implémenter (sous réserve de la pertinence et de la complexité de celles-ci).