

Mathieu BIVERT, Sophie VALENTIN

---

Projet WASP  
FUGU : Find Your Guest Unisonously  
Site de covoiturage

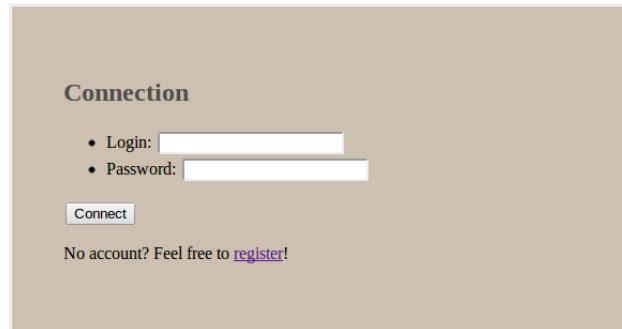
---



Professeur : Tamara REZK

# 1 Fonctionnalités de l'application web

L'utilisateur souhaitant faire du covoiturage doit tout d'abord s'authentifier. En effet, les utilisateurs possèdent leurs propres données. L'utilisateur s'authentifie via un formulaire de connexion, illustré dans la figure 1. Il doit renseigner son login et son mot de passe. S'il n'en possède pas, il peut s'enregistrer sur la page d'inscription, en cliquant sur le lien « register ».



**Connection**

- Login:
- Password:

No account? Feel free to [register!](#)

FIGURE 1 – Formulaire de connexion

## 1.1 Tableau de bord de gestion

Une fois authentifié, l'utilisateur accède à son tableau de bord, représenté en figure 2. Cet écran répertorie tous ses trajets, c'est-à-dire les trajets dont il est le :

**conducteur** , dans la partie haute de la page ;

**passager** , dans la partie basse de la page.

Manage

Propose

Search

Disconnect

Dashboard

Driver for

From	To	Duration	Distance	Description	Actions
43-57 Rue Berlioz Nice	220-250 Avenue du Golf Mougins	31.58	28	Je pars à 7h	<div>delete</div>
3 Rue Arthur Malaussena Levens	423 Route des Lucioles Valbonne	52.14	51	Trajet fait le dimanche soir	<div>delete</div>
Domaine du Loup Cagnes-sur-Mer	Route du Parc Valbonne	12.54	13	Je ne prends pas l'autoroute.	<div>delete</div>

Passenger for

Driver	From	To	Duration	Distance	Description	Actions
mathieu	43-57 Rue Berlioz Nice	220-250 Avenue du Golf Mougins	31.58	28		<div>delete</div>

FIGURE 2 – Tableau de bord

L'utilisateur a accès à tout moment à ce tableau de bord en cliquant sur « Manage » dans le menu de navigation. Enfin, il peut se déconnecter en cliquant sur « Disconnect ».



FIGURE 3 – Menu de navigation

Pour chaque trajet, les adresses de départ et d'arrivée sont affichées. Le temps (en minutes) et la distance (en kilomètres), qui ont été calculés, sont également affichés. L'utilisateur peut voir la description qui a été donnée au trajet. Généralement, ce sont des informations pratiques sur le rendez-vous.

L'utilisateur peut supprimer un trajet de son tableau de bord : pour cela, il clique sur le bouton « delete » du trajet qu'il souhaite supprimer. Son tableau de bord est alors mis à jour. S'il était conducteur pour ce trajet, alors le trajet n'apparaîtra plus dans le tableau de bord des autres passagers.

## 1.2 Proposition de trajet

En cliquant sur « Propose » sur le menu de navigation, l'utilisateur peut proposer un trajet comme dans la figure 4. Au chargement, une carte apparaît avec une adresse de départ et une adresse d'arrivée par défaut. Pour indiquer son trajet, l'utilisateur a deux solutions :

- il peut entrer les adresses de départ et destination dans les champs
- il peut déplacer les marqueurs sur la carte

Dans le premier cas, les marqueurs sur la carte sont immédiatement mis à jour. Dans le second cas, les champs d'adresses sont immédiatement mis à jour. Et quelque soit la méthode pour indiquer le trajet, le temps de trajet et la distance sont calculées. L'utilisateur a ensuite la possibilité de laisser une description en remplissant le champ de texte. Pour terminer, il enregistre son trajet en cliquant sur le bouton « Propose », en-dessous de la description.

### Propose a route

**Departure**

Coordinates : ( 43.703010001 , 7.259760001 )

Address : 43-57 Rue Berlioz Nice

**Arrival**

Coordinates : ( 43.61874 , 7.052480001 )

Address : 767-777 Route des Lucioles Valbonne

Total distance : 24.07 km

Duration : 26 min

Je pars à 8h

Description

Propose

FIGURE 4 – Proposition de trajet

Après enregistrement, l'utilisateur est redirigé sur le tableau de bord où apparaît alors le trajet nouvellement créé.

## 1.3 Recherche de trajet

En cliquant sur « Search » sur le menu de navigation, l'utilisateur peut chercher un trajet existant. Le principe est le même que pour la création de trajet à l'exception que l'utilisateur ne renseigne aucun champ de description. Après un clic sur le bouton « Search », une liste de trajets est affichée.

Research results

Results						
Driver	From	To	Duration	Distance	Description	Actions
toto	43-57 Rue Berlioz Nice	220-250 Avenue du Golf Mougins	31.58	28	popo	<a href="#">join</a>
toto	43-57 Rue Berlioz Nice	220-250 Avenue du Golf Mougins	31.58	28	oifaoifa	<a href="#">join</a>
toto	43-57 Rue Berlioz Nice	220-250 Avenue du Golf Mougins	31.58	28	iuiua	<a href="#">join</a>
toto	43-57 Rue Berlioz Nice	220-250 Avenue du Golf Mougins	31.58	28	popo	<a href="#">join</a>
foo	43-57 Rue Berlioz Nice	220-250 Avenue du Golf Mougins	31.58	28		<a href="#">join</a>

FIGURE 5 – Recherche de trajet

Pour chaque trajet, on peut s’inscrire en tant que passager. Il faut noter que les trajets sur lesquels on s’est déjà inscrit ne figurent pas dans la recherche. Pour s’inscrire, on clique sur le bouton « join » en face du trajet. On est alors redirigé vers le tableau de bord où apparaît maintenant le trajet que l’on a rejoint.

## 2 Serveur

### 2.1 Services

Les différentes pages fournissent les services suivants :

- `index.php` connexion ;
- `register.php` inscription ;
- `user.php` listes des trajets de l’utilisateur ;
- `proposer_trajet.php` enregistrer un nouveau trajet ;
- `chercher_trajet.php` chercher un trajet existant ;
- `lister_trajets.php` lister les trajets existants ;
- `disconnect.php` déconnexion ;

### 2.2 Sécurité

#### 2.2.1 Prévention de l’attaque XSRF

Une telle attaque envoie des requêtes à l’insu de l’utilisateur depuis son propre navigateur. Pour la contrer, on génère sur chaque page un jeton (ou token), qui est stocké côté serveur, avec la fonction PHP `openssl_random_pseudo_bytes()`, qui est un générateur pseudo-aléatoire d’octets.

Pour obtenir un service habituel, il faut préciser, selon les pages, en mode GET ou en mode POST le token obtenu. Le serveur vérifie que le paramètre (GET ou POST) est bien égal au token stocké côté serveur. Ainsi, si l’attaquant tente d’envoyer une requête sans connaître le token et la façon d’envoyer le paramètre, il n’obtiendra pas le service.

Par exemple, pour le service de proposition de trajet, la comparaison entre le token mis en mémoire sur le serveur et celui passé en paramètre se fait dans le fichier `proposer_trajet.php :54`. Sur cette ligne, on appelle en fait la fonction `compare_token_with()` définie dans le fichier `inc/xsrf.php`. Quant à la génération

du nouveau token, elle se fait dans *proposer\_trajet.php* :57. Sur cette ligne, on appelle la fonction *generate\_token()* définie dans le fichier *inc/xsrf.php*.

Le fait de régénérer le token à chaque requête est gênant si on souhaite ouvrir deux onglets par exemple. En effet, l'une des deux pages ouvertes modifie le token enregistré côté serveur et pour l'autre page, la vérification échouera à cause de la désynchronisation. Un autre moyen de prévenir XSRF peut être de générer un token par session pour éviter ce genre de désagréments mais cette façon de faire est moins sûre.

### 2.2.2 Prévention des attaques contre l'intégrité de la session

On souhaite éviter que l'utilisateur puisse modifier les données sensibles fournies par le serveur : celles-ci peuvent être stockées par exemple dans les cookies ou dans les champs hidden. Pour ce faire, on stocke côté serveur une empreinte HMAC de ces données. Cette empreinte est calculée grâce à deux entrées :

- la donnée que l'on souhaite passer à l'utilisateur ;
- la clé privée connue seulement du serveur.

Lors de l'envoi de données altérées, le serveur détectera que l'empreinte de la donnée envoyée par le client n'est pas la même que celle stockée côté serveur. Ainsi on prévient les attaques visant à altérer les données de session.

Dans notre cas, l'utilisation des cookies est restreinte à l'ID de session. Quant aux champs hidden, on les utilise uniquement pour des informations non sensibles.

- Tout d'abord, ils sont utilisés pour des actions prédéfinies sur le serveur. Ainsi, si on fournit des actions inexistantes, le code ne fera aucun traitement.
- Ensuite, ils sont utilisés pour maintenir les informations sur la distance et sur le temps de trajet. Ces informations sont calculées dynamiquement sur les données du trajet de l'utilisateur par le Javascript (API Google Maps). Par conséquent, le stockage d'un HMAC côté serveur nous semblait compromis dans le sens où le client peut modifier ces informations dès qu'elles sont calculées. De plus, ces informations ont un niveau de sécurité bas et n'entravent pas le bon fonctionnement de l'application basée sur les points de départ et d'arrivée. Dans l'idéal, il faudrait recalculer le temps et la distance du trajet côté serveur.

### 2.2.3 Prévention des attaques de fixation

Pour une telle attaque, l'attaquant fixe le Session ID dans l'URL pour un autre utilisateur. Il envoie ensuite cette URL à l'utilisateur. Ce dernier s'authentifie et l'attaquant peut alors utiliser la session de l'utilisateur car il connaît le SID. Pour prévenir cette attaque, il faut modifier le fichier *php.ini* du serveur :

- en activant l'option *session.use\_only\_cookies* afin que le SID soit transmis exclusivement par cookie ;
- en s'assurant que l'option *session.use\_trans\_sid* est désactivée. En la désactivant, on s'assure que les SID ne sont pas transmis par la méthode GET.

### 2.2.4 Prévention des attaques Man in the Middle

Pour réduire la probabilité d'une attaque de type MiM, on utilise un flux chiffré tel que HTTPS. Cela nous assure les propriétés de confidentialité et d'intégrité des paquets transmis.

Pour utiliser le protocole HTTPS, il faut activer SSL/TLS sur le serveur. Si celui-ci est de type Apache HTTP, on édite le fichier *httpd.conf*.

On précise d'abord l'encryption utilisée. On ne veut pas utiliser d'encryption faible.

```
SSLProtocol all
SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5
```

On définit un certificat et la clé RSA privée du serveur

```
SSLCertificateFile "/etc/ssl.crt/server.crt"
SSLCertificateKeyFile "/etc/ssl.key/server.key"
```

Le certificat que l'on a défini doit être vérifié par une autorité de certification externe, censée être définie dans le fichier de configuration du serveur. Ici, on se contente d'un certificat auto-délivré. Pour l'utilisateur, il y aura donc une incertitude quant à la provenance du certificat. Celui-ci pourrait très bien être fourni par un pirate. La vérification du certificat par une autorité de certification externe est donc primordiale dans la mise en place de HTTPS.

Ci-dessous quelques captures après mise en place de SSL/TLS.

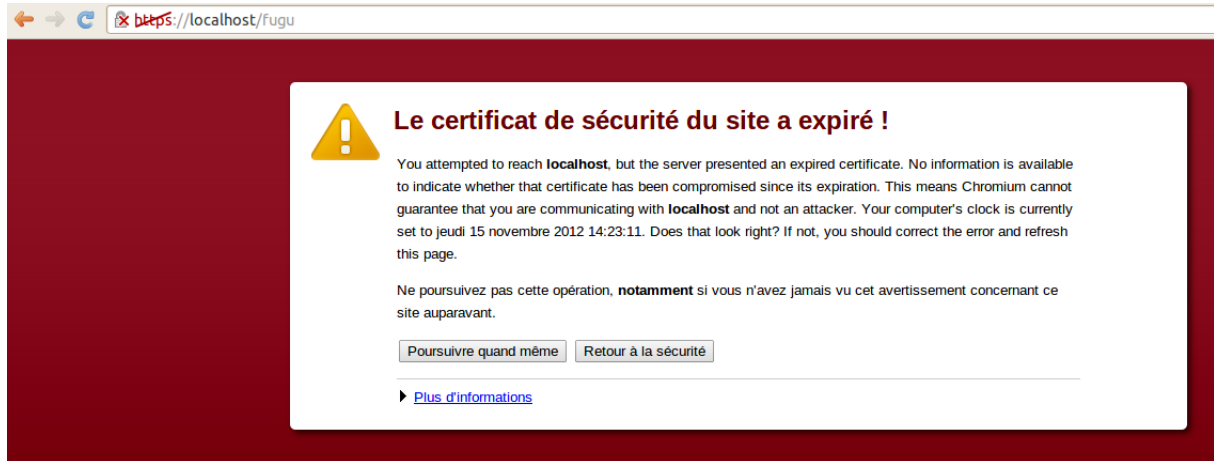


FIGURE 6 – Demande d'ajout d'exception pour un certificat auto-délivré



FIGURE 7 – Détails sur la connexion

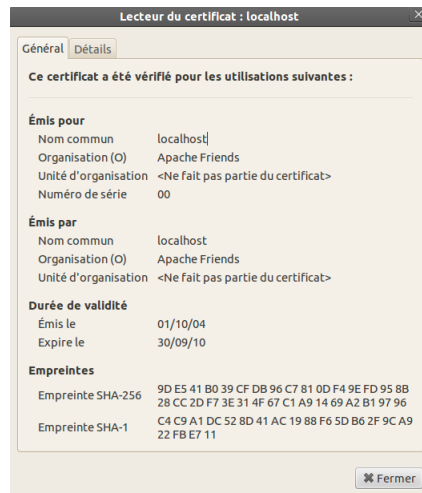


FIGURE 8 – Détails certificat

### 2.2.5 Prévention des « Replay Attacks »

Les replay attacks permettent à un attaquant d'envoyer plusieurs fois un même paquet (eg. celui de connexion).

Comme la protection des failles CSRF/XSRF intègre la génération d'un token à chaque nouvelle action de l'utilisateur, la retransmission d'un même paquet est impossible.

### 2.2.6 Prévention des attaques XSS

Cette attaque utilise le navigateur du client. Du code d'un domaine qui n'est pas de confiance est interprété dans le navigateur du client. Il peut s'agir d'une redirection forcée pour faire du phishing ou d'un vol de cookie avec du Javascript.

Pour protéger le vol de cookie, on active l'option `session.cookie_httponly` dans le fichier `php.ini`. Cela permet de rendre les cookies HttpOnly, c'est-à-dire qu'ils ne seront pas accessibles par des langages de script tels que Javascript. Il faut également vérifier les données utilisateur comme dans le cas des attaques d'injections de code en retraitant le code HTML produit. Pour ce faire, on appelle la fonction `htmlspecialchars`.

### 2.2.7 Prévention des attaques d'injection de code

La prévention des attaques d'injection de code passe par :

- une identification précise des entrées qui ne sont pas de confiance pour ensuite s'assurer que ces données ne sont pas considérées avec un niveau de sécurité plus haut dans le code (taint analysis)
- un assainissement des entrées

## Injections SQL

Il faut vérifier les données de l'utilisateur et échapper les caractères spéciaux. Dans notre cas, nous utilisons des requêtes préparées en PHP. Tout d'abord, on écrit une requête à trous sans les paramètres. Ensuite, la requête est analysée et compilée. Enfin, les paramètres sont donnés et le driver SQLite pour PHP gère automatiquement l'échappement des caractères spéciaux. Concernant la vérification, elle est faite en utilisant des expressions régulières. Par exemple, on vérifie un format raisonnable pour les adresses emails dans `inc/utls.php :38`.

Par exemple pour tester si un utilisateur existe, on prépare la requête dans `inc/user.php :19`, puis on l'exécute `inc/user.php :25`.

## Injection Javascript

On veut éviter qu'un utilisateur insère un script Javascript dans un champ. Pour pallier ce problème, on transforme les caractères spéciaux du texte entré afin qu'un éventuel script Javascript deviennent du texte non interprété. Pour cela, on utilise la fonction `sanitize` d'*inc/utls.php*. Cette fonction appelle la fonction PHP `htmlspecialchars()`, *inc/utls.php :124*.

### 2.2.8 Prévention des attaques RFI/LFI

#### RFI

On ne fait pas de `include` en PHP en utilisant les données utilisateur. L'utilisateur ne peut pas nous forcer à inclure un fichier important du serveur. Cela nous protège des attaques LFI.

#### LFI

On n'inclut pas d'URL externe dans le code PHP. Par sécurité dans le fichier `php.ini`, on désactive l'option `allow_url_include`. Cela nous protège des attaques RFI.

### 2.2.9 Prévention des attaques par distingueur

On utilise des systèmes de hashage « robustes », par exemple, on préfère SHA à MD5. Le générateur de nombre aléatoire utilisés est `/dev/random` sous UNIX : en effet, bien que plus long que son homologue `/dev/urandom`, il fournit des nombres « plus » aléatoires.

### 2.2.10 D'autres éléments de prévention

Pour limiter les vols de cookies, lors d'une déconnexion, les cookies sont effacés de la machine cliente. Cela permet, par exemple, de limiter des vols causés par une intrusion sur le disque dur de l'utilisateur.

Au niveau du déploiement, le serveur HTTP peut-être placé dans un environnement virtualisé (VM), ou pseudo-virtualisé (chroot, BSD jail) pour éviter de compromettre le système hôte en cas de défaillance au niveau du serveur comme du code du site web.

Une protection basique contre les attaques de type **DDoS** est faisable en utilisant des modules comme `mod_evasive` ou `mod_slotlimit` dans le cadre d'un déploiement sous apache2.

Les mots de passes sont hashés (`hash('sha512', $password)`) avant d'être stockés dans la base de données. Cela évite une divulgation des mots de passe si le contenu de la bdd venait à être compromis, ce qui est une bonne idée lorsque un utilisateur utilise le même mot de passe pour plusieurs comptes sur Internet.

## 3 Client

### 3.1 Google Maps

Pour dialoguer avec le mashup intégré dans la page, on utilise l'API Google Maps.

#### 3.1.1 Carte, coordonnées et adresses

##### Service *Geocode*

Tout d'abord, on utilise le type `Map` pour obtenir le mashup et le lier à un élément de notre page. Et on utilise le service `Geocode` de Google qui permet de traduire des adresses en coordonnées et inversement. Par exemple, dans *js/carte.js :8*, il y a création d'un objet *Geocode* et appel de la fonction `geocode()`. On définit une fonction de callback pour la réponse du service où on peut alors accéder au résultat de la traduction.



## Type *LatLng*

Les points (coordonnées) sont représentés par le type *LatLng*. On l'utilise notamment pour définir le point de centre de la carte au chargement et ensuite, pour définir les coordonnées des points de départ et d'arrivée.

### 3.1.2 Calcul des itinéraires

#### Service *DirectionsService*

On utilise le service *DirectionsService*. Effectivement, pour calculer les données de l'itinéraire (temps et distance), on a besoin que l'API de Google détermine les différents segments de l'itinéraire. Ainsi, on appelle la fonction *route()* de *DirectionsService* où on spécifie : - les coordonnées de départ ; - les coordonnées d'arrivée ; - le moyen de locomotion (ici voiture). On définit là aussi une fonction de callback pour la réponse. Un exemple d'appel est disponible dans *js/carte.js :91* On obtient dans le résultat de la réponse de type *DirectionsResult* différents tableaux. Par exemple, on obtient un premier tableau avec des points (coordonnées) appartenant à l'itinéraire calculé. On obtient également un tableau avec des segments de l'itinéraire. Le second tableau nous sert à calculer la distance totale et le temps total car chaque segment contient une information sur son temps et sa distance. Voir la fonction *calculerDistanceEtTemps* dans *js/carte.js :29*).

#### Synchronisation sur la carte

Quant à l'objet *DirectionsRenderer*, il nous permet de synchroniser sur l'objet *Map* l'itinéraire calculé de type *DirectionsResult*. C'est cet objet qui affiche l'itinéraire en bleu sur la carte. Ainsi, à chaque réponse de la méthode *route()*, on met à jour l'information d'itinéraire *DirectionsResult* dans l'objet *DirectionsRenderer* pour que l'affichage de la carte soit modifié et bien entendu, on calcule les informations de distance et de durée. Cela est fait à chaque fois que l'utilisateur modifie les champs d'adresse dans le formulaire.

#### Utilisation d'un listener

Mais il faut également que les actions de l'utilisateur sur la carte soient prises en compte. En effet, un déplacement des marqueurs ou un déplacement de l'itinéraire en bleu doit provoquer une mise à jour des marqueurs, des adresses dans les champs du formulaire, et des données de l'itinéraire comme la distance et le temps. Pour cela, un événement est créé : l'objet *DirectionsRenderer* écoute les modifications sur la carte.

## 3.2 Javascript

### 3.2.1 Utilisation de la Prototype Chain

Javascript est un langage orienté objet à prototype. Ce dernier point le différencie des langages objets dits à classes (Smalltalk, Java, etc.).

En js, un objet possède une référence à un autre objet appelé son prototype. Celui-ci permet de mettre en place une relation d'héritage.

Supposons la relation de prototype suivante : on définit *a* comme une variable contenant un champ entier *v* de valeur 1, puis *b* comme un objet dont le prototype est l'objet *a* :

```
js> var a = {v: 1};  
js> var b = Object.create(a);
```

Lorsque l'on essaye d'accéder à une propriété de *b*, la chaîne de prototype est remontée jusqu'à trouver ladite propriété :

```
js> b.v  
1
```

```
js> b.baz
js>
```

Dans le premier cas, aucun champs *v* n'existe dans *b*; on regarde donc dans son prototype, où l'on trouve bien ce champ. Dans le second, ni *b* ni *a* ne contiennent de champ *baz*. On remonte donc au prototype de *a*, qui est inexistant (plus exactement, il vaut *null*). Aucune valeur n'est donc retournée.

L'exemple suivant permet de s'assurer que la relation de prototype fonctionne bien sur des objets (et pas sur des interfaces par exemple) :

```
js> a.v = 5
5
js> b.v
5
/*
 * ici, on ne remonte pas la chaîne de prototype : un nouveau champ
 * v est créé dans b.
 */
js> b.v = 42
42
js> a.v
5
/*
 * certaines implémentations de javascript permettent d'accéder au
 * prototype d'un objet via la notation non standard __proto__:
 */
js> b.__proto__.v = 42
42
js> a.v
42
```

### 3.2.2 Utilisation de la Scope Chain

La règle de la scope chain est la suivante : Si une propriété n'est pas trouvée dans le contexte de l'objet courant, alors une recherche est lancée dans l'objet parent, et ainsi de suite jusqu'à éventuellement atteindre l'objet *global*.

On utilise par exemple la variable *coordonneesDepart*, qui est en fait une propriété de l'objet *global*, dans la fonction *init* (*js/carte.js* :66). La définition de cette fonction est un objet. Ainsi, quand on utilise la variable *coordonneesDepart* dans cette fonction, on cherche s'il existe une propriété définie dans l'objet courant (la fonction). Comme ce n'est pas le cas, on remonte la scope chain d'un cran et on trouve la propriété *coordonneesDepart* dans l'objet *global*.

### 3.2.3 Utilisation du mot-clé this

Le mot-clé *this* est une propriété du contexte d'exécution. Le mot-clé *this* est donc une référence vers l'objet *global* sauf dans certains cas. Par exemple, si *this* est utilisé dans une fonction, alors la valeur de *this* dépend du contexte d'exécution lors de l'appel de la fonction.

Nous utilisons le mot-clé *this* par exemple dans *proposer\_trajet.php* :98. La valeur de *this* est ici *global*.

### 3.2.4 Utilisation de la récursivité

Nous utilisons des structures de contrôle de type boucle for, par exemple dans *proposer\_trajet.php* :37.

### 3.2.5 Manipulation du DOM

L'accès aux éléments de la page HTML se base sur l'id des balises et se fait dans le Javascript avec *document.getElementById()*. Des manipulations du DOM sont effectuées depuis le Javascript. En effet, on modifie par exemple la valeur de champs hidden dans *proposer\_trajet.php* :45. Ou encore, on remplace le contenu d'une balise dans *proposer\_trajet.php* :44.

### 3.3 Sécurité : Prévention des failles Javascript

Concernant la sécurité, l'objet *global* ne doit pas être délivré à l'attaquant puisque l'intégralité de la page y est accessible. De plus, on peut y trouver les cookies.

On a donc vu dans la deuxième partie de ce rapport, qu'il fallait prévenir les attaques de type XSS et injection de code.

Ici, du code extérieur est inclus. On utilise le mashup avec la balise *script*. Or, en utilisant *script*, la mémoire de la page intégrant le mashup et la mémoire du mashup ne sont pas scindées : l'objet *global* est partagé. Le code du script inclus peut donc accéder à l'objet *global*.

Une alternative pour intégrer un tel mashup serait d'utiliser la balise *iframe* pour avoir des mémoires séparées. Cependant, il faudrait mettre en place un mécanisme de communication entre les deux parties avec *postMessage*.

Cependant, le code intégré ici provient de Google, qui est considéré comme un domaine de confiance.

## A php.ini

On donne ici un extrait d'un *php.ini* pouvant être utilisé par notre application.

```
; don't give unnecessary informations about PHP
expose_php = Off

; restrict execution and input processing time
max_execution_time = 30
max_input_time = 60

; limit the memory used by a script
memory_limit = 128M

; report every error and log them
error_reporting = E_ALL
log_errors = On
log_errors_max_len = 1024
report_memleaks = On
; no need for pretty error messages
html_errors = Off

; don't give unnecessary debug informations to users
display_errors = Off
display_startup_errors = Off

; default mimetype and charset
default_mimetype = "text/html"
default_charset = "UTF-8"

[Session]
; use files to store/retrieve session data
session.save_handler = files
session.save_path = "/var/lib/php"

; do use cookie, secure cookie
session.use_cookies = 1
session.cookie_secure = 1
; store session ID in cookie
session.use_only_cookies = 1
session.name = PHPSESSID

; cookie unavailable from js
session.cookie_httponly = 1

; use /dev/random over /dev/urandom as an entropy generator
; slower but more resistant to statistical attacks
session.entropy_file = /dev/random

; avoid sending SID within an URL
session.use_trans_sid = 0

; 1 : SHA-1 (160b), 0 : MD5 (128b)
session.hash_function = 1
```