

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

WEBRAILS

Projet – AdventureTrip

Rapport

Table des matières

1. Introduction.....	3
2. Cahier des charges.....	3
2.1. Type de site.....	3
2.2. Cas d'utilisation et définition des droits.....	4
2.2.1. Diagramme de cas d'utilisations.....	4
2.2.2. Définition des droits.....	5
2.3. Structure de la base de données.....	6
2.3.1. Schéma conceptuel.....	6
2.3.2. Schéma relationnel.....	7
2.3.3. Description de la DB.....	7
2.3.4. Contraintes d'intégrité.....	7
3. Structure du logiciel.....	8
4. Implémentation.....	9
4.1. Description de la partie Ajax.....	9
4.2. Description des librairies utilisées.....	10
4.2.1. Bootstrap.....	10
4.2.2. API de Google Maps.....	10
4.3. Éléments d'implémentation remarquables.....	11
4.3.1. Affichage de la liste des voyages (trip_controllers.rb).....	11
4.3.2. Calcul de l'itinéraire d'un voyage (trip.js).....	11
4.3.3. Calcul des coûts.....	13
4.3.4. Gestion des participants.....	13
5. Gestion de projet.....	14
5.1. Itération 1.....	14
5.2. Itération 2.....	15
5.3. Itération 3.....	15
5.4. Itération 4.....	16
6. État des lieux.....	16
6.1. Fonctionnalités partiellement implémentées.....	16
6.1.1. Calcul du coût en carburant et en péages.....	16
6.1.2. Demande d'adhésion à un trip publique.....	16
6.1.3. Quitter un trip.....	17
6.1.4. Amélioration de la GUI.....	17
6.2. Possibles extensions.....	17
7. Conclusion.....	17
8. Annexes.....	17

1. Introduction

Ce document contient le rapport de notre projet de groupe – AdventureTrip – effectué dans le cadre du cours à option « WEBRAILS » de 3ème année de l'HEIG-VD. Il contient notamment le cahier des charges, la structure du logiciel, son implémentation, ainsi que la section de gestion de projet.

AdventureTrip est en bref un site web réalisé avec Ruby on Rails, et qui se définit comme étant une application de gestion de voyages de type « roadtrips ». En plus d'offrir une gestion utilisateur des plus normales, elle permet de créer et de gérer ses propres voyages de manière ergonomique et en utilisant divers aspects sociaux (participants à un roadtrip, commentaires, etc.). Les voyages peuvent être publiques (accessible à n'importe qui) comme privés, et permettent notamment de planifier le trajet (via l'API de Google Maps), les coûts, ainsi qu'une liste d'objets à prendre.

2. Cahier des charges

2.1. Type de site

Notre site permet la planification de roadtrips, c'est-à-dire de voyages en camping-car, en voiture, à vélo ou par un autre moyen de transport. Il permet de centraliser les différents aspects liés à la gestion d'un tel voyage. Partager son voyage pourra ainsi inspirer d'autres utilisateurs à se lancer dans l'aventure.

La page d'accueil du site présente une description du site ainsi que les derniers roadtrips organisés sur le site.

Après avoir créé un compte et s'être connecté, un utilisateur peut créer un nouveau voyage et y ajouter plusieurs participants, également membres du site. Ensuite, il lui est possible de définir les différentes étapes. Grâce à une API de cartographie (Google Maps), l'utilisateur peut avoir un affichage sur une carte avec indication du nombre de kilomètres, du temps nécessaire en fonction du moyen de transports, ainsi que du coût selon un coût au kilomètre fourni.

Les participants, qui ont été ajouté par l'organisateur du voyage, peuvent faire des commentaires généraux sur l'organisation. S'ils ont été promu administrateurs, ils peuvent également modifier les différents aspects du voyage (détails globaux, parcours du voyage, participants et packing-list (liste des objets à prendre pour le voyage)).

Un simple visiteur du site, c'est-à-dire non inscrit, peut consulter les différents voyages planifiés sur le site, à condition qu'ils soient publics.

2.2. Cas d'utilisation et définition des droits

2.2.1. Diagramme de cas d'utilisations

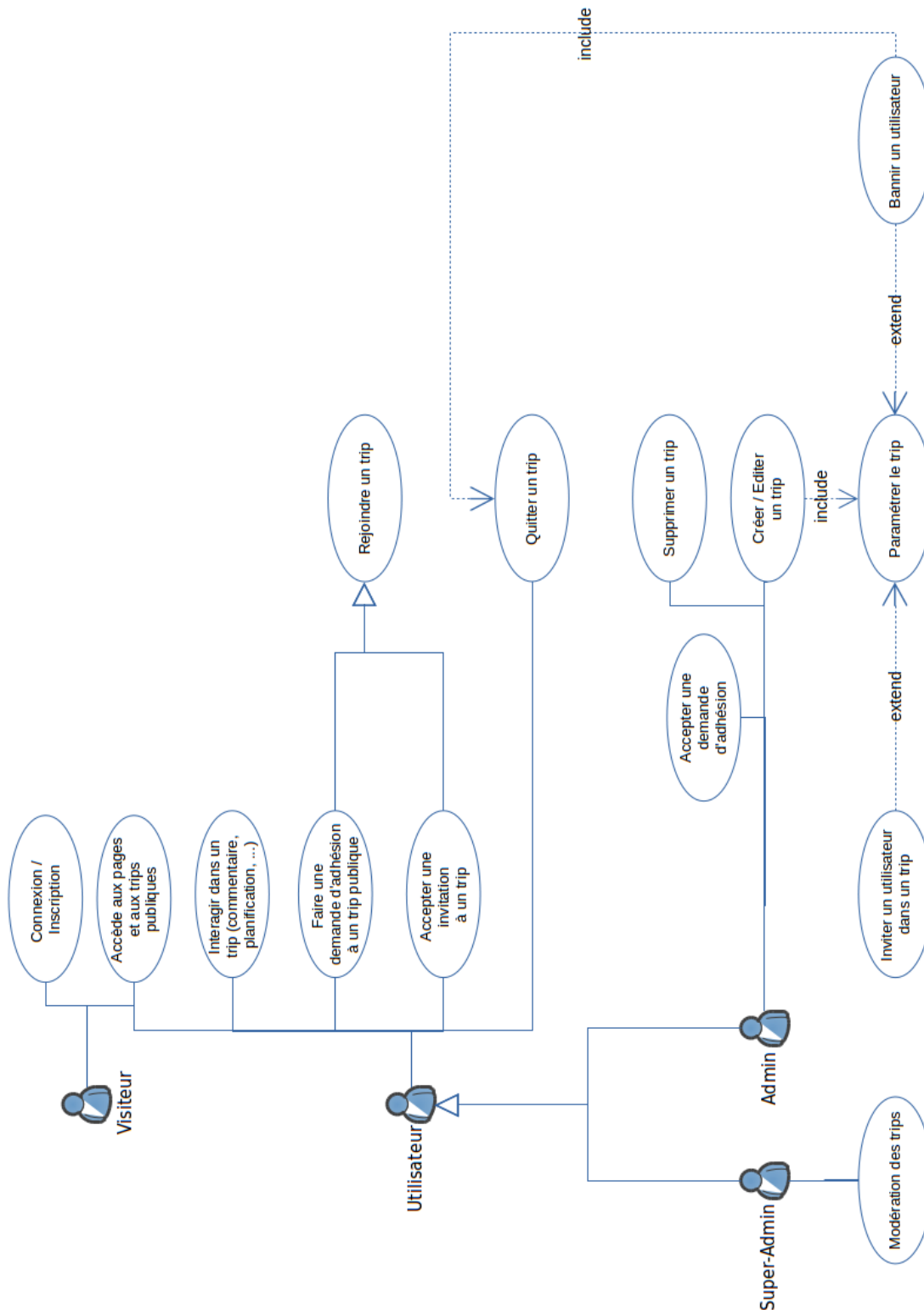


Illustration 1: diagramme des cas d'utilisation

2.2.2. Définition des droits

Le site possédera 4 types d'utilisateurs :

1. **Visiteur** : il s'agit ici du simple utilisateur non-connecté qui souhaite accéder au site ; il peut accéder aux différentes pages publiques (page d'accueil, page d'à-propos, etc.) ainsi qu'en lecture aux différents trips publiques. Il peut aussi s'inscrire et/ou se connecter, ce qui le fera grader au type d'utilisateur suivant.
2. **Utilisateur** : il s'agit de l'utilisateur connecté, qui – en plus des actions réalisables par un visiteur – peut rejoindre un trip (en faisant une demande d'adhésion si celui-ci est publique (finallement pas développé par manque de temps), ou en acceptant une invitation provenant d'un administrateur), interagir à l'intérieur (en postant un commentaire ou en apportant des modifications aux coûts et à la packing list), ainsi que quitter un trip.
3. **Administrateur** : ce rôle à la base est lié à un trip en particulier : celui qui aura été créé par cet utilisateur. En effet, lorsqu'un utilisateur crée un nouveau trip, il en sera automatiquement l'administrateur, puis pourra par la suite grader un ou plusieurs autres utilisateurs pour ce rôle. L'administrateur pourra – en plus des actions réalisables par un utilisateur – éditer entièrement le ou les trips auxquels il est assigné, ainsi qu'envoyer des invitations ou accepter des demandes d'adhésions s'il est publique (non-développé). Il peut de plus bannir un utilisateur qui serait un peu trop invasif à son goût, et – en cas de nécessité – peut supprimer le trip (ce qui aura pour effet de retirer tous ses membres).
4. **Super-Administrateur** : il s'agit ici du rôle suprême, qui permet à celui qui le détient d'accéder à tous les trips et de les modérer comme bon lui semble. Un super-administrateur ne peut être promu que via la base de données.

2.3. Structure de la base de données

2.3.1. Schéma conceptuel

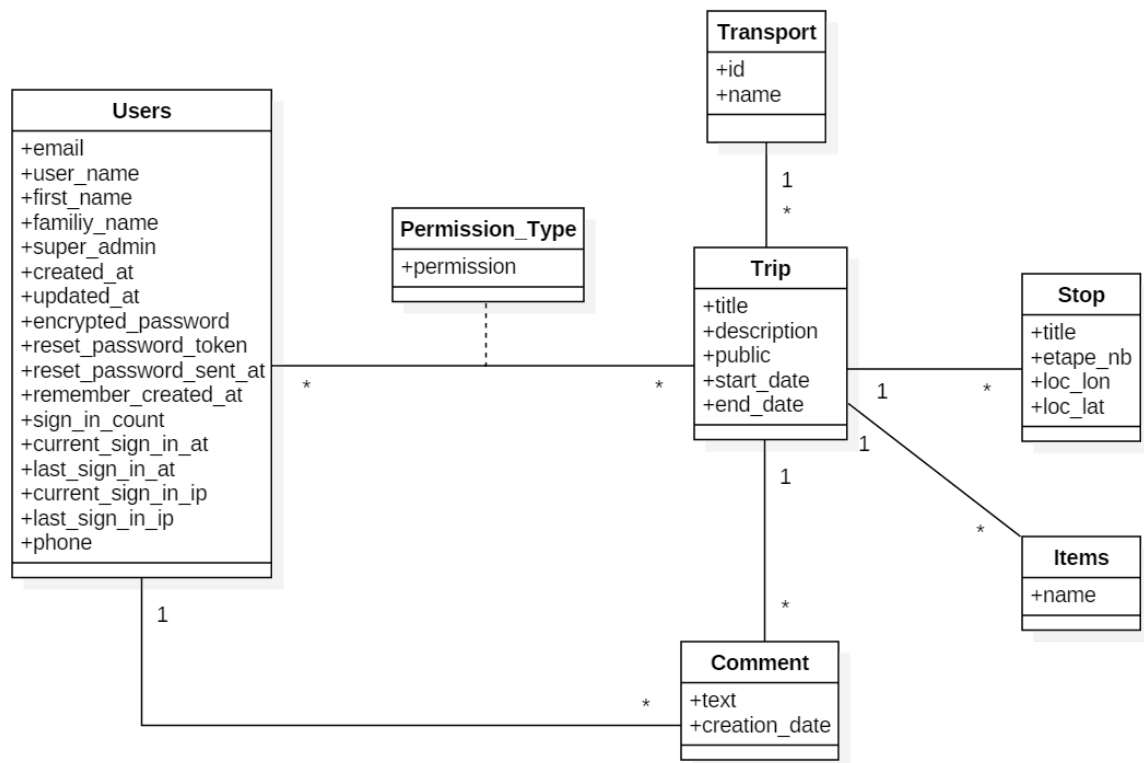


Illustration 2: schéma conceptuel de la DB

2.3.2. Schéma relationnel

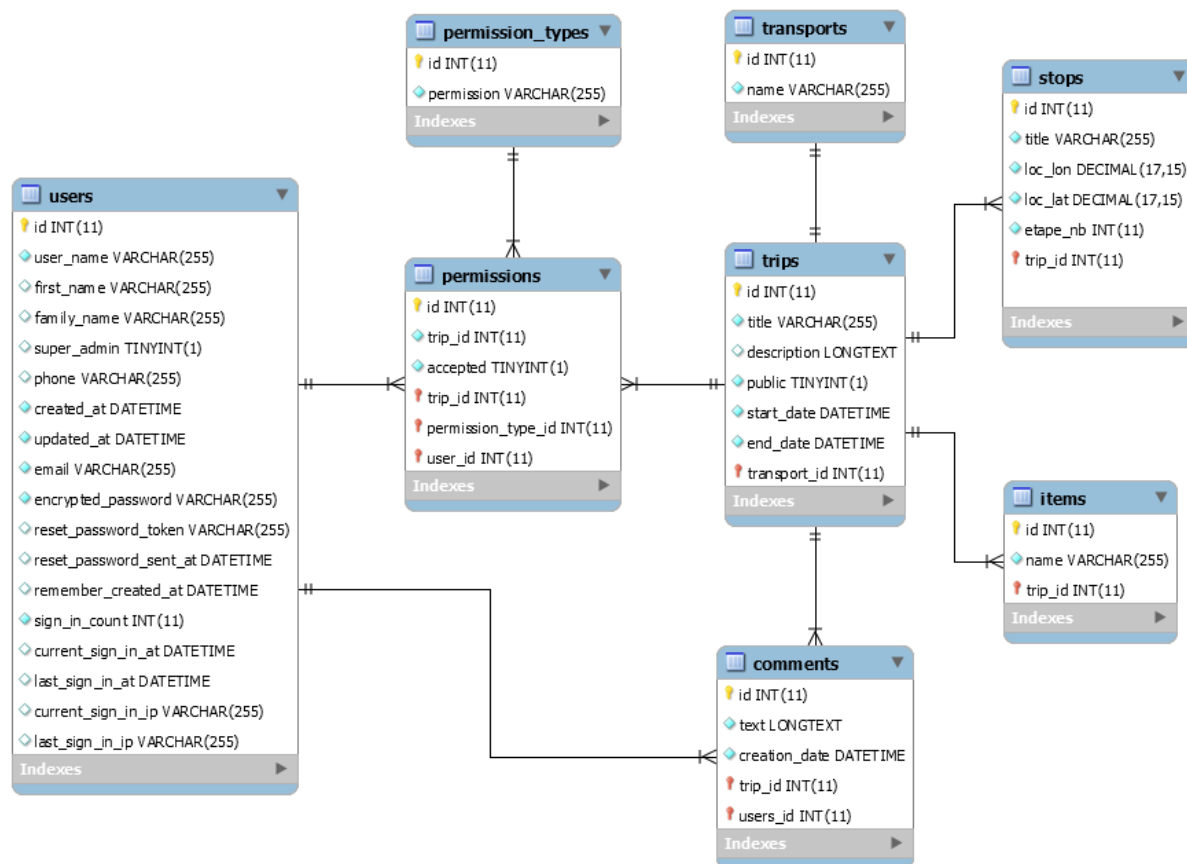


Illustration 3: schéma relationnel de la DB

2.3.3. Description de la DB

Notre site possède 6 entités conceptuelles qui se représentent en 8 tables SQL. La table trip est au centre du projet, elle est connectée à toutes les autres tables à travers différentes relations. A noter que la table « Users » est reliée à travers une table d'association « Permissions » qualifiant chaque appartenance d'un utilisateur à un voyage par un niveau de droit allant de simple participant à admin. Cette même table d'association est reliée à une table contenant les différents types de permissions.

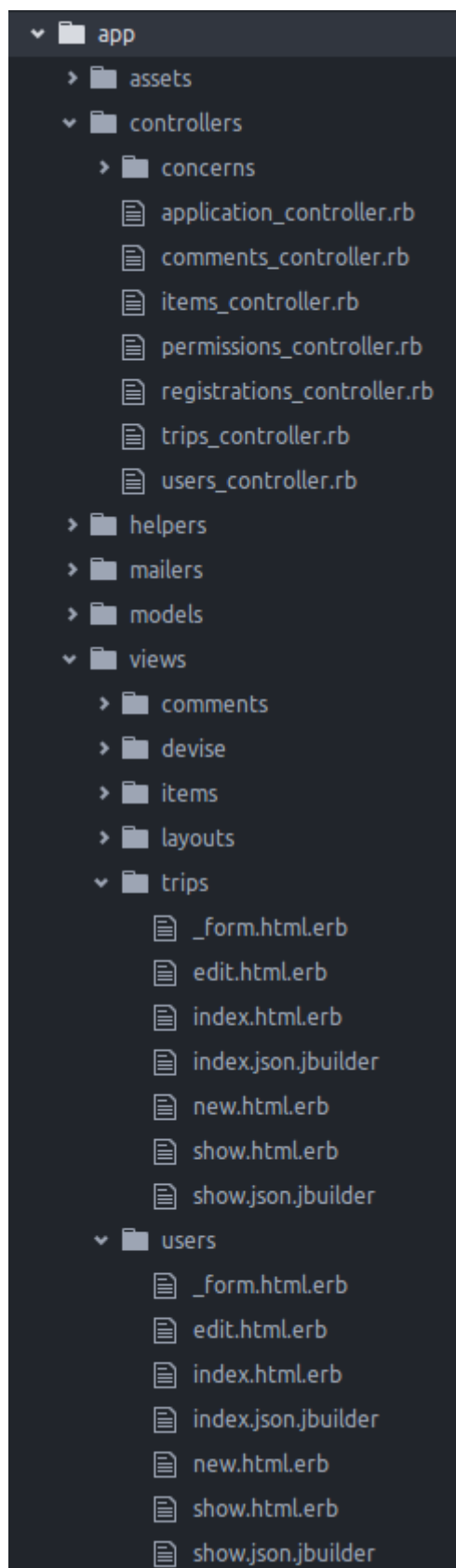
La table « Stops » contient les différentes étapes d'un voyage. Ces étapes ont un ordre et sont déterminées par des coordonnées. « Comments » contient les commentaires des participants d'un voyage. « Items » est constitués de la « packing-list » d'un voyage. « Transports » désigne les différents modes de transport disponibles lors de la création du voyage.

2.3.4. Contraintes d'intégrité

Les associations entre les tables se font via les champs « id » qui jouent le rôle de clé primaire et des « XXX_id » qui jouent le rôle de clés étrangères. Les liaisons se font concrètement au sein de l'application Ruby on Rails dans la définition des modèles.

Les champs « NOT NULL » sont représentés par des losanges pleins alors les champs facultatifs par des losanges vides. Les id sont bien sûrs uniques.

3. Structure du logiciel



Assets :

- Contient les fichiers JavaScript et CSS utilisés par le côté client de l'application.

Contrôleurs :

- **Application** : récupère les invitations de l'utilisateur à des voyages et les envoie au layout.
- **Comments** : contient les méthodes CRUD relatives aux commentaires des voyages.
- **Items** : contient les méthodes CRUD relatives à la « packing list » des voyages.
- **Permissions** : contient deux méthodes pour respectivement accepter et refuser des invitations.
- **Registrations** : contrôleur relatif à la gestion utilisateur, utilisé par la librairie.
- **Trips** : contient les méthodes CRUD relatives aux voyages, ainsi que diverses méthodes de validation des données.
- **Users** : contient les méthodes CRUD relatives aux utilisateurs.

Modèles :

- Rien de spécial à dire là-dessus, si ce n'est qu'il y en a un par table et que les liaisons (« belongs_to », etc.) ont été effectuées.

Vues :

- **Comments** et **Items** : vues générées par le Scaffolding mais non-utilisées.
- **Devise** : vues utilisées par la librairie de gestion utilisateur.
- **Layout** : contient le modèle de page des vues (en-tête et pied-de-page) ; affiche ou non l'image en page d'accueil et change le contenu de l'en-tête selon si l'utilisateur est connecté ou non.
- **Trips** : contient les vues relatives aux voyages.
- **Users** : contient les vues relatives aux utilisateurs.

Notons quelques fichiers importants :

- **app/assets/javascripts/trips.js** : contient toute le code JavaScript lié aux trips, à savoir la gestion de la carte avec l'API Google Map, les contrôles opérés sur les différents champs, la gestion de la tabulation, l'ajout de participants, etc.
- **app/models/comment.rb** : contient une méthode de génération de timestamp pour la date de publication d'un commentaire.
- **config/initializers/assets.rb** : contient une ligne permettant de pré-compiler les polices d'écriture se situant dans « app/assets/fonts ».

A noter que tout le code est commenté, ce qui devrait le rendre aisément compréhensible.

4. Implémentation

4.1. Description de la partie Ajax

Nous avons utilisé Ajax pour l'ajout de commentaires et d'objets dans la packing-list :

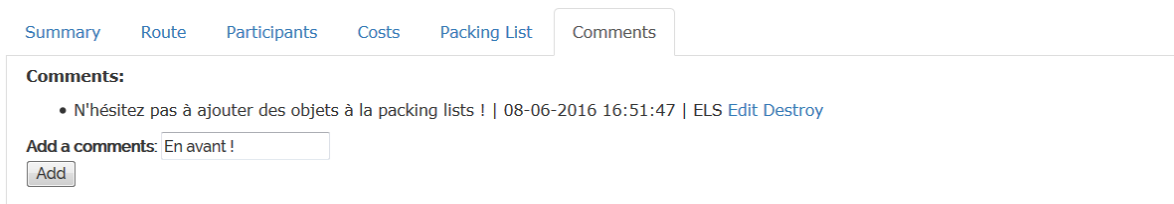


Illustration 4: ajout d'un commentaire

Lors de l'ajout d'un nouveau commentaire, ce dernier apparaît automatiquement dans la liste et est stocké dans la DB, ceci sans avoir besoin de rafraîchir la page dans sa totalité. La gestion des objets de la packing-list suit la même logique.

Afin que le formulaire d'ajout de commentaire ne charge pas une nouvelle page lors de l'envoi, nous indiquons la propriété « `remote => true` ».

```
<%= form_for(@comment, :remote => true) do |f| %>
```

Illustration 5: propriété "remote => true"

Ensuite, dans le contrôleur gérant la création de commentaires, on y ajoute la ligne `format.js` :

```
if @comment.save
  format.html { redirect_to @comment, notice: 'Comment was successfully created.' }
  format.json { render :show, status: :created, location: @comment }
  format.js { render 'create' }
```

Illustration 6: création de commentaires dans le contrôleur

Cela va exécuter le fichier `create.js.erb` déterminant l'action à effectuer en plus de l'ajout dans la DB :

```
$('#comment_list').append("<li> <%= @comment.text %> | <%=  
@comment.creation_date.strftime('%d-%m-%Y %H:%M:%S')%> |" + $  
('#user_name').html() + "<%= j(link_to 'Edit', edit_comment_path(@comment)) %> <  
%= j(link_to 'Destroy', @comment, :confirm => 'Are you sure?', :method =>  
:delete, :remote => true, :class => 'comment') %></li>");
```

Ce fichier permet l'ajout du nouveau commentaire avec l'affichage de l'heure, de l'utilisateur ainsi que des liens pour éditer et supprimer ce dernier.

4.2. Description des librairies utilisées

4.2.1. Bootstrap

Bootstrap a été utilisé à des fins de design pur, et a été installé selon la démarche qui avait été présentée dans le laboratoire. Rien de spécial à présenter ici, donc.

4.2.2. API de Google Maps

Nous utilisons l'API de Google Maps pour tout ce qui est relatif à la cartographie dans le projet.

Pour ce faire, nous réalisons un appel asynchrone à l'API dans la partie « `<head>` » du layout, nous permettant d'initialiser la variable « `google.maps` » qui sera utilisée pour le rendu des cartes.

```
<script async defer  
  src="https://maps.googleapis.com/maps/api/js?key=AIzaSyC-hp0MQg5eXA-K8Xn5_gGDC5oDmmFf0xM">  
</script>
```

Illustration 7: initialisation de l'API de Google Maps.

A noter que selon la documentation officielle de l'API, il aurait fallu donner une fonction callback dans la source, mais cela n'était pas réalisable à cause de Turbolink. En effet, ce dernier produit un bug qui pousse l'application à refaire un appel à chaque chargement de page, ce qui n'est pas apprécié par l'API qui nous indique que la carte a déjà été initialisée. Ce bug étant connu sur les forums, nous avons suivi ce qui était indiqué pour le résoudre en déplaçant les chargements Turbolink à la fin du layout et en enlevant le callback. L'initialisation de la carte se fait manuellement la première fois que l'utilisateur accède à l'onglet « Route » d'un voyage (que ce soit dans la page de détails ou les pages d'ajout/édition).

Une fois cette démarche réalisée, il est possible d'utiliser la variable « `google.maps` » afin de créer le rendu de nos cartes. Tout ce code est situé dans le fichier « `app/javascripts/trips.js` » et est entièrement commenté pour une meilleure compréhension.

4.3. Éléments d'implémentation remarquables

4.3.1. Affichage de la liste des voyages (trip_controllers.rb)

Lorsque l'utilisateur accède à la page listant tous les voyages auxquels il a le droit d'accéder (donc les voyages publics et les privés auxquels il est participant), l'action du contrôleur s'occupe de récupérer tous ces trips dans un premier temps, puis récupère tous les droits que possèdent l'utilisateur sur ses trips (donc administrateur ou simple participant). La vue s'occupe ensuite d'afficher ou non les liens d'administration en regardant pour chaque trip les droits que possèdent l'utilisateur. Ceci a été implémenté en deux temps, car il était très difficile de le faire en une seule requête avec l'ORM de Rails...

A noter que le super-administrateur outrepassa ces règles, car il possède un accès administrateur à absolument tous les voyages.

4.3.2. Calcul de l'itinéraire d'un voyage (trip.js)

Le calcul de l'itinéraire d'un voyage se déroule en plusieurs temps : l'utilisateur choisit tout d'abord le moyen de transport du voyage, entre ensuite les différents arrêts (stops) en cliquant de manière chronologique sur les endroits désirés sur la carte, indique ensuite si le voyage est une boucle ou non (donc si le point de départ correspond au point d'arrivée), puis demande finalement au système de calculer l'itinéraire selon les points. A noter que cette dernière étape est facultative, car seuls les points sont sauvegardés dans la base de données et l'itinéraire est recalculé à chaque fois que l'utilisateur accède au trip.

Il existe deux types de cartes dans l'application, à savoir les cartes modifiables (dans les pages d'ajout/édition de voyages) et les cartes non-modifiables (dans la page « Show »), identifiées par la variable booléenne « editableMap » :

Voici donc les détails techniques de ces opérations :

1. Lorsque l'utilisateur accède pour la première fois à l'onglet « Route » d'un trip, la fonction JavaScript « initMap() » est appelée et initialise la carte, comme son nom l'indique. Si la carte est modifiable, nous initialisons dessus un listener qui nous permet de détecter le clic de l'utilisateur pour y ajouter un point.
2. La fonction « initExistingPoints() » est appelée et s'occupe d'afficher sur la carte les points déjà existant, si l'utilisateur est en train d'éditer un voyage possédant déjà un trajet. Cette fonction appelle automatiquement la fonction « calcRoute() » permettant de calculer le trajet du voyage (voir plus bas). Les points sont récupérés depuis des « input » de type « hidden », contenant les coordonnées récupérées depuis le contrôleur.

3. L'utilisateur peut ensuite ajouter des points sur la carte en cliquant dessus, ou en retirer à l'aide du bouton prévu à cet effet. Attention au fait que l'API limite le nombre de points par trajet à 10 (donc le départ, l'arrivée et 8 points intermédiaires ; attention au fait que si le départ est le même que l'arrivée, deux points seront tout de même comptés). Deux tableaux sont utilisés du côté du code pour stocker les points de l'utilisateur : « `coordinatesArray` » qui contient les coordonnées de chacun des points et qui sera passé à l'API pour calculer le trajet, et « `markersArray` » qui contient les objets de type « `google.maps.Marker` » (donc les points sur la map) et qui permet donc de pouvoir les supprimer. De plus, pour chaque point ajouté, nous ajoutons aussi un champ « `input` » de type « `hidden` » contenant les coordonnées du point dans la vue. Ceci est nécessaire pour pouvoir envoyer les informations au contrôleur lors du POST du formulaire.
- A chaque fois que l'utilisateur ajoute ou retire un point, la fonction « `updateInstructions()` » est appelée et permet de modifier le texte d'instruction affiché dans le panel de gauche.

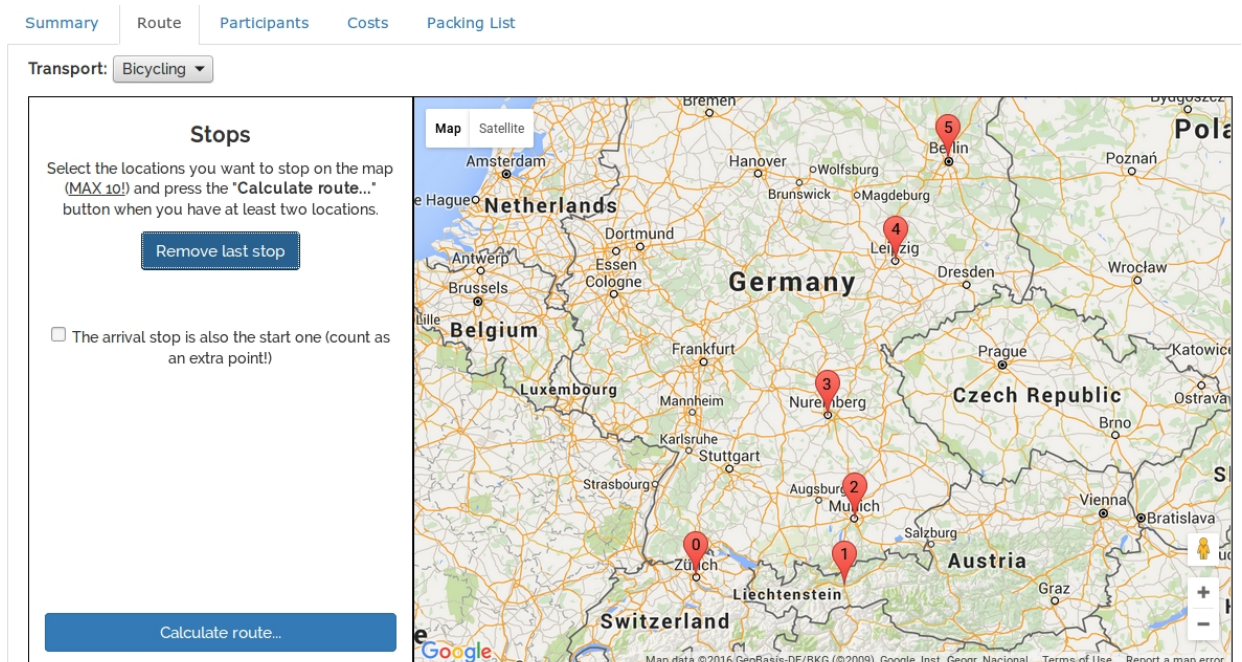


Illustration 8: ajout des points sur la carte par l'utilisateur

4. Lorsque l'utilisateur a terminé, il peut appuyer sur « Calculate route... » pour appeler la fonction `calcRoute()` et ainsi faire une requête à l'API pour calculer la direction. Cette fonction s'occupe de récupérer le moyen de transport sélectionné, puis de passer les coordonnées des différents points à l'API. Une fois le trajet calculé, nous calculons manuellement la distance ainsi que la durée du trip, qui nous seront aussi utiles pour calculer les coûts.

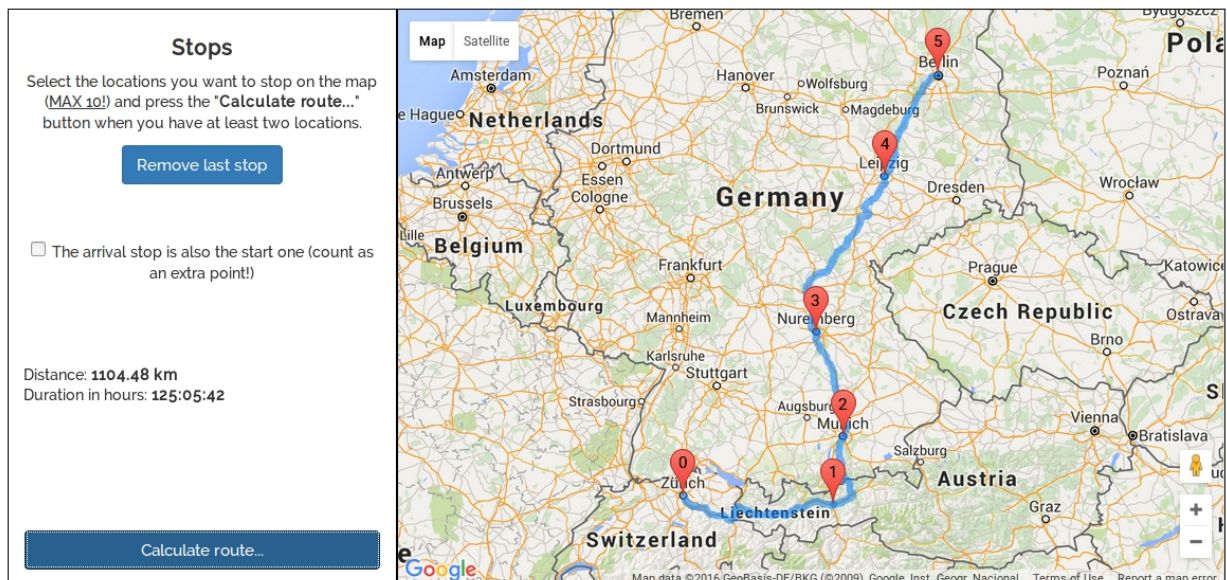


Illustration 9: calcul du trajet

En cas de modification d'un trip, les points (stops) du voyage sont tous effacés de la table puis les nouveaux sont réinsérés. Ceci a été fait pour simplifier le processus de modification.

4.3.3. Calcul des coûts

Le calcul des coûts sont réalisés à la volée selon le prix au kilomètre indiqué par l'utilisateur, à l'aide de l'appel à la fonction « `calculateCosts()` » et de la variable « `totalDistance` » calculée au préalable dans la fonction « `calcRoute()` ». Si la route n'a pas encore été calculée, un message est affiché, indiquant de le faire.

4.3.4. Gestion des participants

Le gestionnaire des participants d'un voyage est composé de deux listes : la première affichant les utilisateur n'appartenant pas au voyage, et la deuxième affichant les utilisateur y appartenant. Il est possible de les déplacer à l'aide des deux boutons « `>` » et « `<` », appelant respectivement les fonctions « `addParticipants()` » et « `removeParticipants(currentUserId)` », le paramètre servant à identifier l'utilisateur connecté pour l'empêcher de se retirer lui-même du projet. Seule la liste des participants est ensuite envoyée au contrôleur. Comme pour les points, la liste des participants est à

chaque fois entièrement effacée de la table avant d'être réinsérée, pour moins s'embêter avec les modifications.

Il est possible de changer les droits d'un participant appuyant sur le bouton « Toggle Admin » (s'il est en gras il s'agit d'un administrateur, sinon il s'agit d'un participant normal) qui appellera la fonction « toggleAdmin(currentUserId) », le paramètre ayant le même but qu'avant (un utilisateur ne peut pas s'enlever lui-même les droits d'administration sur un trip). Afin de simplifier la gestion des droits, chacun des éléments des listes contient des attributs « isadmin » et « userid », indiquant respectivement si l'utilisateur est un administrateur ou non et son ID ; ils peuvent aussi posséder un attribut « iscurrent » qui indique s'il s'agit de l'utilisateur connecté, par exemple :

```
<option userid="5" value="1-5"
ondblclick="changeParticipantStatus(5)" id="participant-5"
iscurrent="1" class="adminUser" admin="1">edri (Miguel
Santamaria)</option>
```

Ces attributs seront utilisés dans la fonction « toggleAdmin » pour déterminer l'action à réaliser.

La valeur (attribut « value ») du champ sera utilisée lors du POST par le contrôleur, afin de correctement ajouter l'utilisateur. Elle est composée de deux nombres formaté de la manière « X-Y », X indiquant si l'utilisateur est administrateur (1) ou non (0), et Y indiquant l'ID de l'utilisateur.

5. Gestion de projet

Nous possédons un total de 9 semaines à disposition (sans compter la semaine de présentation), à partir de la semaine du 11.04.2016.

Toutes les itérations ont été respectées, même si cela a été particulièrement difficile à cause des travaux que nous avons en parallèle dans les autres cours. Nous en tirons donc un bilan positif et sommes satisfaits.

5.1. Itération 1

Objectif général

Modélisation métier et spécification des besoins du projet

Objectifs détaillés

- **Gestion** (spécification) :
 - o Mise au point des spécifications
 - o Analyse et définition des cas d'utilisations
 - o Création du plan d'itération
 - o Réalisation du modèle de la base de données
 - o Réalisation du rapport intermédiaire
 - o Réalisation de tests de faisabilité
 - o Découverte des solutions disponibles (bibliothèques, APIs, etc.)

Durée

1 semaine

Dates de début et de fin

Du 11.04.2016 au 17.04.2016

Partage du travail entre les membres du groupe

Les analyses et discussions se feront en commun pour cette itération.

5.2. Itération 2

Objectif général

Conception des fonctionnalités de base

Objectifs détaillés

- Développement des bases de l'application
 - Connexion
 - Inscription
 - Création et gestion basique des trips (création, suppression, modification, gestion des utilisateurs)
- Réalisation d'une GUI basique (sans design particulier)
- Tests unitaires et fonctionnels

Durée

3 semaines

Dates de début et de fin

Du 18.04.2016 au 08.05.2015

Partage du travail entre les membres du groupe

- **Bastien** : réalisation de la connexion et de l'inscription
- **Miguel** : réalisation de la création de trips

Les deux membres réaliseront en commun la GUI, ainsi que la gestion des trips. Chacun testera ses fonctionnalités, ainsi que celles de l'autre.

5.3. Itération 3

Objectif principal

Conception des fonctionnalités avancées

Objectifs détaillés

- Développement des fonctionnalités avancées de l'application
 - Page d'accueil
 - Gestion de trips
 - Fonctionnalités sociales (invitations, commentaires, etc.)
- Finalisation de la GUI
- Tests unitaires et fonctionnels

Durée

4 semaines

Dates de début et de fin

Du 09.05.2016 au 05.06.2016

Partage du travail entre les membres du groupe

- **Bastien** : fonctionnalités sociales
- **Miguel** : page d'accueil + GUI

La gestion des trips sera réalisée en commun ; chacun testera ses fonctionnalités, ainsi que celles de l'autre.

5.4. Itération 4**Objectifs principaux**

Tests et validations

Objectifs détaillés

- Réalisation des tests finaux
- Validation des différentes fonctionnalités
- Mise à jour de la documentation
- Rendu du projet

Durée

1 semaine

Dates de début et de fin

Du 06.06.2016 au 12.06.2016

Partage du travail entre les membres du groupe

Chacun réalisera les tests et la documentation.

6. État des lieux**6.1. Fonctionnalités partiellement implémentées**

La plupart des fonctionnalités ont été implémentées. Voici les fonctionnalités partiellement implémentées :

6.1.1. Calcul du coût en carburant et en péages

Nous avons initialement prévu d'utiliser la REST API fournie par Michelin afin de calculer automatiquement les coûts en carburants et éventuels péages sur le trajet. En raison des contraintes pour l'obtention d'un compte et des délais d'attentes, nous avons préféré utiliser l'API fournie par Google Maps qui est plus limitée mais plus facile d'utilisation. Elle permet le calcul d'itinéraire pour des trajets en vélo et à pieds, ce que ne permet pas Michelin.

Nous permettons tout de même à l'utilisateur de fournir un coût par kilomètres pour ainsi calculer dynamiquement un coût pour le trajet sélectionnés. Ceci dit, rien n'est actuellement stocké dans la base de données, concernant les coûts ; il s'agit simplement d'un utilitaire de calcul.

6.1.2. Demande d'adhésion à un trip publique

Il n'est pas actuellement possible pour un utilisateur de demander à joindre un trip public. Il reçoit bien une invitation lorsqu'il est ajouté à un trip mais cela ne va pas dans l'autre sens.

6.1.3. Quitter un trip

La possibilité de quitter un trip n'a pas été implémentée par manque de temps. La seule manière de le faire pour l'instant est de supprimer totalement un trip.

6.1.4. Amélioration de la GUI

L'interface générale du site web est actuellement très simpliste, et nous aurions souhaité aller plus loin à ce niveau là. Malheureusement, par manque de temps, nous avons dû couper court à ces souhaits. Il serait donc appréciable d'améliorer cette GUI.

6.2. Possibles extensions

En plus de terminer de développer les fonctionnalités partiellement implémentées, nous pourrions ajouter d'autres fonctionnalités au site. La première serait d'adapter notre site en un réseau social, facilitant les échanges et les interactions entre les utilisateurs (votes, hashtags, liste d'amis, partages, ...)

Une autre fonctionnalité pourrait être un système de suggestion d'étapes. Par exemple, l'utilisateur fournit un lieu de départ et d'arrivée ainsi qu'une durée en jour et le système lui suggérerait des étapes. Ces étapes pourront par exemple être situées à proximité de lieux d'intérêt. On pourrait imaginer d'autres critères, tels qu'un nombre maximum de kilomètres par jour ou tout simplement une météo clémente.

7. Conclusion

Ce projet nous a permis de mieux comprendre le fonctionnement de Ruby on Rails. Bien que pendant les laboratoires, nous avons déjà pu intégrer la majorité de la matière, c'est en appliquant dans un cas pratique sans contrainte que nous avons pu intégrer réellement le concept.

Nous estimons avoir fait un bon travail bien que pas parfait. Principalement par manque de temps, certaines fonctionnalités sont manquantes mais le site est complètement fonctionnel en l'état.

8. Annexes

- Mode d'emploi du programme
- Indications pour l'installation