|  |
| --- |
|  |
| EMACovoit’ |
| **Développement** |
| Projet GSI – IFIE 2016 |

|  |
| --- |
| Solène Pettier |
| Paul Girault |
| Arnaud Ballet |
| Grégoire Lecomte |



Table des matières

[PARTIE I : Organisation du code de l’application EMACovoit’ 4](#_Toc444182032)

[Controleurs de l’application 4](#_Toc444182033)

[Fichiers de fonctions 4](#_Toc444182034)

[PARTIE II : Choix de développement de l’application 5](#_Toc444182035)

[Choix de la base de données de l’application EMACovoit’ 5](#_Toc444182036)

[Choix de la sécurité de l’application 5](#_Toc444182037)

[Authentification 5](#_Toc444182038)

[Différents droits 5](#_Toc444182039)

[protection des formulaires 5](#_Toc444182040)

[Choix des services 5](#_Toc444182041)

[PARTIE III : Description des fonctions développées 6](#_Toc444182042)

[Publier une annonce de covoiturage 6](#_Toc444182043)

[Dans le cas d’un trajet en France 6](#_Toc444182044)

[Dans le cas d’un trajet dans Albi 6](#_Toc444182045)

[Rechercher une annonce de covoiturage 7](#_Toc444182046)

[Recherche de covoiturage en France 7](#_Toc444182047)

[Recherche de covoiturage dans Albi 7](#_Toc444182048)

[Authentification d’un utilisateur 7](#_Toc444182049)

[Statut Administrateur de l’application 7](#_Toc444182050)

[PARTIE IV : Les problèmes rencontrés lors du développement 8](#_Toc444182051)

[Langage et Framework 8](#_Toc444182052)

[Contraintes de validation de formulaires 8](#_Toc444182053)

[Organisation de la base de données 8](#_Toc444182054)

[Relation many-to-many 8](#_Toc444182055)

[Utilisation de services côté client et côté serveur 9](#_Toc444182056)

[La recherche d’annonces 9](#_Toc444182057)

[PARTIE V : Gestion du projet EMACovoit’ 10](#_Toc444182058)

[Outils mis en œuvre dans le cadre de la gestion du projet 10](#_Toc444182059)

[La répartition des tâches 10](#_Toc444182060)

[Une méthode de gestion agile 10](#_Toc444182061)

[PARTIE VI : Comparaison résultat souhaité et résultat final 11](#_Toc444182062)

[Le respect du planning d’estimation des charges 11](#_Toc444182063)

[Différentiel résultat souhaité et résultat final 11](#_Toc444182064)

[Points d’amélioration 13](#_Toc444182065)

[PARTIE VII : Essais fonctionnels et structurels 14](#_Toc444182066)

[PARTIE VII : GLOSSAIRE 16](#_Toc444182067)

|  |  |  |  |
| --- | --- | --- | --- |
| HISTORIQUE DES REVISIONS DU DOCUMENT – DEVELOPPEMENT | | | |
| Version | Date | Nom | Description |
| 1.1 | 21/02/2016 | Grégoire Lecomte | Collecte des informations concernant l’explication de la phase de développement de l’application. |
| 1.2 | 23/02/2016 | Grégoire Lecomte | Révision et intégration de toutes les informations concernant l’explication de la phase de développement de l’application. |
| 1.3 | 25/02/2016 | Grégoire Lecomte | Ultime révision du contenu du rapport concernant la phase de développement |

# PARTIE I : Organisation du code de l’application EMACovoit’

## Controleurs de l’application

La liste ci-dessous correspond à l’ensemble des fichiers contenus dans le code source et servant de contrôleurs à l’application EMACovoit’ :

* Le fichier ***myapp.pm*** est le contrôleur principal de l’application EMACovoit’.
* Le fichier ***admin.pm*** regroupe les différentes routes réservées aux utilisateurs ayant le statut d’administrateur.
* Le fichier ***annonce.pm*** regroupe les différentes routes traitant de l’objet « annonce de covoiturage » (Covoiturage en France ou dans Albi).
* Le fichier ***sauvegarde.pm*** est le contrôleur traitant des objets liés à la sauvegarde d’annonces.
* Le fichier ***user.pm*** est le contrôleur traitant des objets faisant référence aux utilisateurs.

## Fichiers de fonctions

La liste des fichiers ci-dessous correspond à l’ensemble des fichiers de fonctions contenus dans le code source de l’application EMACovoit’ :

* Le fichier ***calcul.pm*** contient la fonction de calcul de distance entre différents points à partir des coordonnées GPS.
* Le fichier ***validation.pm*** contient les différentes fonctions de validation, utilisées dans ***annonce.pm.***
* Le fichier ***filter.pm*** contient toutes les fonctions utilisées par la route permettant la recherche d’annonces de covoiturage (située dans ***annonce.pm***).
* Le fichier ***firewall.pm*** contient les fonctions utilisées dans les différents contrôleurs pour effectuer les vérifications des droits de l’utilisateur de l’application.
* Le fichier ***messages.pm*** contient les fonctions permettant l’affichage de messages.

# PARTIE II : Choix de développement de l’application

## Choix de la base de données de l’application EMACovoit’

Nous avons décidé d’utiliser deux tables différentes en ce qui concerne les annonces de covoiturages, une table **CovoiturageFrance** et une table **CovoiturageAlbi**. Nous aurions pu utiliser une table **Covoiturage** générique qui regroupe les attributs communs aux deux tables, mais nous ne savions pas comment est géré l’héritage en perl et si le module DBIx ::Class permettait ce genre de relation, nous avons donc choisi cette solution. Un regroupement à l’aide d’une table générique simplifierait le code mais nécessite un certain temps pour retravailler les différentes routes que nous avons mises en place.

## Choix de la sécurité de l’application

### Authentification

L’application est prévue pour fonctionner à l’aide de **l’annuaire LDAP**.

### Différents droits

Nous avons pensé à utiliser le plugin **Dancer::Auth** pour gérer les accès routes. Cependant, comme nous n’avons uniquement des utilisateurs administrateurs et non administrateurs, nous avons réalisé un pare-feu « maison ».

### protection des formulaires

Nous avons tenté de mettre en place la protection **CSRF** effective pour les formulaires. Cependant, le peu de documentation que nous avons trouvé concernant la mise en œuvre de cette protection ne nous a pas été suffisante pour réussir.

La construction des formulaires avec le plugin **Dancer::Forms** pourrait permettre d’inclure la protection CSRF. Mais cela nécessiterait une refonte de l’application.

## Choix des services

Nous utilisons deux services de géolocalisation : **OpenstreetMap** et **GoogleMaps**.

Nous utilisons **OpenStreetMap** pour la recherche de lieux (villes/villages en France). A l’aide de ce service nous avons réalisé une auto complétion suggérant à l’utilisateur des lieux. L’avantage de l’utilisation de ce service, est qu’il nous permet de récupérer de nombreuses données sur le lieux : Ville, Région, Coordonnées GPS…

Nous avons utilisé **GoogleMaps** pour l’affichage d’un itinéraire sur une carte et le calcul des différentes étapes de cet itinéraire. La documentation **OpenStreetMap** étant trop pauvre à ce niveau nous n’avons pas réalisé cette fonctionnalité à l’aide de ce service.

# PARTIE III : Description des fonctions développées

## Publier une annonce de covoiturage

Deux types d’annonces peuvent être publiés : les covoiturages en France et les covoiturages dans Albi. Les covoiturages Albi-Toulouse sont une sous-catégorie des covoiturages « France » dont les villes de départs et d’arrivés sont déjà connues.

La publication d’annonce mènera donc vers une étape intermédiaire qui impliquera le choix du type de covoiturage.

### Dans le cas d’un trajet en France

Un trajet en France suit un itinéraire à partir d’un lieu de départ jusqu’à un lieu de destination en passant ou non par un nombre indéfini d’étapes. A chaque objet **covoiturageFrance** est donc lié un **lieuFrance** de départ et d’arrivée. La table « **passe\_par** » fait la correspondance entre les covoiturages « **France** » et les étapes de ceux-ci.

Les champs de complétion de lieux sont munis d’une autocomplétion « **OpenStreetMap** » déclenchée dès lors que l’utilisateur a saisi un nombre minimum de caractères. Il n’est pas indispensable de s’aider de l’auto complétion. Le lieu entré par l’utilisateur fait l’objet d’une vérification côté serveur à l’aide du service **openstreetmap** pour vérifier la validité de celui-ci.

Si le lieu est valide, et qu’il n’existe pas déjà dans la base de données, il est enregistré. La latitude et la longitude du lieu sont également enregistrées pour permettre la recherche de lieux proches et l’affichage de l’itinéraire.

La publication de trajets Albi-Toulouse a pour particularité d’avoir les champs lieu de départ et lieu de destination pré-remplis et non modifiables. Un bouton permet d’inter-changer les deux villes selon le trajet désiré.

### Dans le cas d’un trajet dans Albi

Un trajet dans la ville d’Albi va d’un lieu de départ à un lieu de destination tout deux contenus dans une liste prédéfinie de lieux dans Albi : Leclerc Albi, Résidence Jacques Halfon etc.

Cette liste de lieux prédéfinie est maintenue à jour par les administrateurs de l’application qui auront la possibilité d’ajouter un lieu, ou de modifier (supprimer) un lieu existant. La liste n’est donc pas figée.

Quel que soit le type de trajet, les champs nombres de places, date et heure sont obligatoires. Les champs description et prix ne le sont pas.

La modification découle de la publication. Elle utilise la même vue à la différence que les champs sont pré-remplis. Une fois la modification effectuée, l’application renvoie vers la vue de l’annonce afin que l’utilisateur puisse visualiser ses modifications. Concernant la gestion des étapes lors de la modification, les entrées de la table PassePar correspondantes à l’annonce en question sont supprimées puis réinsérées par soucis de simplicité.

Une annonce est modifiable soit par l’auteur ou un utilisateur administrateur de l’application. Lors de la suppression d’annonce, un formulaire de validation permet d’effectuer la suppression.

## Rechercher une annonce de covoiturage

Comme pour la publication d’une annonce de covoiturage, il y a deux cas, la recherche d’annonces en France (avec la spécificité des covoiturages Albi-Toulouse) et la recherche d’annonces dans Albi.

Lorsqu’aucun paramètre n’est précisé, la liste de tous les covoiturages France, ou Albi est affiché dans une table muni de filtres JavaScript.

### Recherche de covoiturage en France

Il y a 3 manières de filtrer la recherche, par lieu de départ, par lieu de destination et par date.

Lorsqu’un lieu est précisé, l’application retourne la liste des covoiturages au départ (ou à destination) ou passant à proximité de ce lieu.

La recherche permet donc de retrouver le covoiturage Lyon –> Saint-Etienne –> Rodez -> Albi en cherchant un covoiturage Saint-Etienne -> Rodez.

Dans le cas de la recherche de covoiturage Albi-Toulouse. Le lieu de départ est par défaut Albi et la destination Toulouse. Un bouton permet d’inter changer les deux lieux.

### Recherche de covoiturage dans Albi

La recherche s’effectue par lieu de départ et lieu de destination. Le traitement est beaucoup plus simple que dans le cas de la recherche de covoiturage France.

## Authentification d’un utilisateur

La configuration du serveur implique que dès l’arrivée sur la page d’accueil de l’application, une authentification sera requise via le serveur **LDAP** de l’école. La méthode « **hook**» permet de vérifier à chaque requête de l’utilisateur, si celui-ci est bien enregistré en tant que variable de session. Si celui-ci n’est pas enregistré, l’application récupère le login de l’utilisateur fixé par LDAP, et cherche si l’utilisateur existe en base de données. Si celui-ci n’existe pas, l’application récupère via LDAP, le nom, le prénom et le type (ex :IFIEXXXX ou autre) via l’annuaire LDAP, et crée l’utilisateur en base de données.

Via son profil l’utilisateur aura la possibilité de modifier son Nom et son Prénom. Un utilisateur enregistré n’est pas administrateur par défaut.

## Statut Administrateur de l’application

Il existe deux types d’utilisateur : l’utilisateur standard et l’administrateur.

Les utilisateurs administrateurs auront la possibilité de modifier et supprimer n’importe quelle annonce. Ils pourront également gérer la liste des lieux dans Albi ainsi que la liste des utilisateurs. Ils auront la possibilité d’ajouter ou de supprimer le droit administrateur d’autres membres.

Un ajustement est possible afin de niveler les droits afin d’avoir un seul super-utilisateur, des modérateurs et des utilisateurs standards.

# PARTIE IV : Les problèmes rencontrés lors du développement

## Langage et Framework

La première des difficultés a été de comprendre la composition et le fonctionnement des objets manipulés. Pour cela, le fait de dumper les objets manipulés a été primordial. Nous avons ainsi pu appréhender peu à peu le fonctionnement du Framework **Dancer**. Nous sommes plus habitués à la manipulation d’objets **Java** ainsi le passage à la manipulation d’objets perl a été au début un peu perturbant.

## Contraintes de validation de formulaires

Il nous a été facile de comprendre comment réaliser des contraintes de validation simples, mais il nous a été un peu plus difficile de comprendre la création et le fonctionnement de contraintes propres pour les formulaires

## Organisation de la base de données

L’idée d’origine était de réaliser une seule table de gestion des annonces de covoiturage, qu’ils soient dans Albi ou en France. Chaque covoiturage était ainsi lié à un lieu possédant un nom, une latitude et une longitude.

Pour rechercher ces lieux, nous utilisions une auto complétion à l’aide du service **openstreetmap**. Le problème était le suivant : certains lieux classiques dans Albi telles que le Leclerc n’était accessibles qu’en tapant le libellé exact. D’autres lieux classiques pour un covoiturage dans Albi, comme la résidence Gambetta n’est pas connu sous ce nom par **openstreetmap**.

Nous avons donc décidé de faire deux tables l’une pour les covoiturages « France » fonctionnant de la manière décrite précédemment, l’autre pour les covoiturages dans Albi. Ces covoiturages ont un lieu de départ et un lieu d’arrivée se situant dans une liste définie de lieux dans Albi. Les différents administrateurs sont responsables de la gestion de cette liste.

Ainsi, au fur et à mesure du développement, le covoiturage Albi est resté un objet plutôt simple alors que le covoiturage France s’est étoffé, avec la publication de covoiturages avec étapes, etc.

Nous aurions pu regrouper les propriétés communes des deux types de covoiturages dans une table, et les propriétés propres à chaque covoiturage auraient été définies dans des sous-tables.

Mais nous avons trouvé peu de documentation concernant la manière dont **Dancer** gère ce cas d’héritage entre objets.

## Relation many-to-many

La publication de covoiturages avec étapes a été délicate à gérer. Réaliser le JavaScript permettant d’ajouter des champs dynamiques, auquel on lie l’auto complétion **openstreetmap** n’a pas été le plus difficile. Le plus difficile a été de récupérer les différentes étapes côté serveur, de les valider, et de les réafficher en cas de soumission de formulaire incomplète. Dès que nous modifions un covoiturage avec étapes, les entrées de la table **PassePar** liées à ce covoiturage étaient systématiquement supprimées puis réinsérés, par soucis de simplicité.

## Utilisation de services côté client et côté serveur

Nos appels au service **openstreetmap** s’est fait pendant une longue partie du développement uniquement côté client, mais une fois que nous avons appris à faire des requêtes côté serveur, cela nous a permis d’être certains que les lieux insérés en bases de données sont valides. S’il n’y avait pas de vérification côté serveur, l’utilisateur serait libre d’entrer un nom de lieu valide, mais avec des coordonnées gps erronées (en modifiant les champs cachés). Une fois la requêté côté serveur effective, nous nous sommes affranchis de ce fonctionnement, ce qui a simplifié la manière de publier une annonce, ainsi que la manière de rechercher une annonce.

## La recherche d’annonces

Dès lors que nous avons mis en place la recherche d’annonces par coordonnées GPS, et non plus par lieux, la recherche d’annonces s’est compliquées. Jusque-là nous faisions des vérifications par rapport au lieu de départ, ou de destination du covoiturage en les comparant au(x) lieu(x) entré(s) par l’utilisateur et nous retrouvions la liste des covoiturages pertinent.

Une fois que nous étions capables de calculer côté serveur une distance entre deux coordonnées GPS, nous avons effectués le tri des annonces pertinentes à l’aide d’un algorithme de filtre. En fonction des données critères rentrés par l’utilisateur, nous cherchons la liste des lieux étant proches du ou des lieux entrés par l’utilisateur, puis nous cherchons la liste des covoiturages au départ (ou destination), ou passant par ce lieu.

La réalisation de ces filtres a été délicate mais permet une recherche vraiment précise pour l’utilisateur.

# PARTIE V : Gestion du projet EMACovoit’

## Outils mis en œuvre dans le cadre de la gestion du projet

* **Trello pour la planification**: Il s’agit d’un outil de gestion de projet en ligne. Il est basé sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, ce qui traduit leur avancement. Nous avons donc pu suivre à distance l’avancement de chaque membre du groupe projet. Cette méthode nous a permis de nous organiser plus facilement les uns par rapport aux autres et d’être chacun force de proposition quant à l’avancement du projet.
* **Git pour le partage** : Ce logiciel nous a permis de stocker l’ensemble des fichiers de l’application en conservant la chronologie de toutes les modifications qui avaient été effectuées. Ainsi, nous pouvions être constamment à jour en récupérant les dernières versions du code source de chacun, tout cela à distance.
* **Méthode Peer to Peer pour l’apprentissage**: Étant donné les différences de compétences en termes de programmation dans l’équipe GSIGN, nous avons privilégié la méthode pédagogique Peer to Peer. En effet, grâce à cette méthode, chacun apprenait en permanence des autres mais participait également à l’apprentissage comme à l’évaluation de tous.



* **Facebook pour l’organisation**: Afin de se tenir informé de chaque réunion ou de chaque changement conséquent, nous utilisions Facebook.

## La répartition des tâches

Comme dit précédemment, les différences de compétences en programmation de l’équipe ont été fortement prises en compte. Pour mener à bien ce projet, nous avons donc misé sur le partage de connaissances des meilleurs et l’acquisition de compétences pour les moins bons. La répartition des tâches était donc faite en fonction des facilités de chacun face au travail demandé.

L’utilisation de l’outil Trello nous permettait de connaître les tâches en cours de réalisation pour chaque membre, d’être informés sur les problèmes à régler ou d’évoquer les points à modifier. Ainsi, nous avancions au maximum chacun de notre côté en gardant une cohérence les uns par rapport aux autres.

Nous avons travaillé pendant les créneaux d’autonomie alloués au projet dans l’emploi du temps. Nous faisions en sorte de nous concerter régulièrement lors de jalons en groupe. Ces derniers nous permettaient de remettre à niveau les membres en difficulté, de spécifier certains points d’amélioration, de corriger les points d’ombre et redéfinir la répartition des tâches selon l’avancement du projet.

## Une méthode de gestion agile

La méthode de travail employée nous a permis d’avancer dans un cadre dynamique et efficace. Nous avons découpé le projet en petits blocs et nous avons procédé par cycles courts à valider avant de passer au suivant. Chacun avait une visibilité claire du projet grâce aux feedbacks réguliers et pouvait apporter des changements selon la pertinence de ses idées.

Nous avons donc abordé cette dernière phase de projet avec une approche dite **Agile**. En effet, nous avons préféré privilégier :

* Des interactions efficaces lors des jalons,
* Des logiciels opérationnels tels que Trello et Git,
* Une collaboration avec les clients I2D, rencontrés dans le but de présenter un premier jet,
* Une adaptation au changement plutôt que le suivi d’un plan précis.

Nous nous laissions une certaine liberté quant aux tâches à réaliser et nous étions suffisamment autonomes pour faire de nouvelles propositions ou apporter des modifications judicieuses, tout cela dans un souci de satisfaction client.

# PARTIE VI : Comparaison résultat souhaité et résultat final

## Le respect du planning d’estimation des charges

Une estimation des charges en phase 2 nous avait permis de déceler les tâches qui nous prendraient le plus de temps selon leur difficulté. Si de trop grandes difficultés venaient à être rencontrées, nous avions choisi d’opter pour des solutions plus simples, autrement dit, de ne pas intégrer le développement avec OpenStreetMap et de se restreindre à l’utilisation de JavaScript pour les fonctions compliquées.

Cependant, lors du développement des fonctions, nous avons réalisé que nous avions assez de temps pour nous pencher sur la réalisation de l’application dans le cas de difficulté maximum établi lors des phases de planification. Le planning initial de développement des fonctions de l’application a donc bien été respecté.

La fonction « Rechercher un covoiturage » a été la plus longue et difficile à coder, suivie de la fonction « Publier un covoiturage ». Seule la fonction « Login », comprenant l’inscription d’un nouvel utilisateur sur LDAP, a pris plus de temps que prévu.

En parallèle, les tâches de suivi d’avancement et d’intégration avaient été surévaluées. En effet, grâce aux outils utilisés, ces étapes étaient presque instantanées. Nous avons donc gagné un temps précieux en pilotage et en partage de travail.

Pour la rédaction du rapport, de la documentation ou d’une présentation, le temps évalué était aussi un peu élevé. C’est donc sans doute sur ce type de tâches que nous avons su gagner du temps afin de nous concentrer sur le développement.

## Différentiel résultat souhaité et résultat final

**Récapitulatif des fonctions principales**:

|  |  |  |
| --- | --- | --- |
| **Code** | **Fonction :** | **Implémentée ?** |
| FP1-1 | Publier une proposition de covoiturage | OK |
| FP1-2 | Modifier une proposition de covoiturage | OK |
| FP1-3 | Consulter une proposition de covoiturage | OK |
| FP1-4 | Supprimer une proposition de covoiturage | OK |
| FP2-1 | Afficher les propositions de covoiturage | OK |
| FP2-2 | Rechercher une proposition de covoiturage parmi les annonces postées | OK |
| FP3-1 | Permettre l’accès à l’outil par l’utilisateur | OK |
| FP3-2 | Permettre l’accès à l’outil au gestionnaire | OK |
| FP3-3 | Permettre au gestionnaire d’accéder à la gestion de toutes les annonces | OK |
| FP3-4 | Permettre au gestionnaire de détecter certains abus dans la publication d’annonces | OK |
| FP3-5 | Permettre l’accès à l’historique pour le gestionnaire | OK |

Les tests (voir partie suivante) prouvent que toutes ces fonctions principales sont fonctionnelles. Nous nous sommes concentrés sur le développement de ces fonctions en premier car ce sont les fonctions principales à l’utilisation de l’outil.

Pour les publications et recherches d’annonces de covoiturage, nous pensions raisonner de la même façon, qu’il s’agisse d’un covoiturage pour la France (Albi-Toulouse ou autre ville) ou d’un covoiturage dans la ville d’Albi. Cela n’a pas été le cas, il a fallu différencier le type de covoiturage dans la programmation. Cependant, le résultat répond au cahier des charges.

**Récapitulatif des fonctions supplémentaires**:

|  |  |  |
| --- | --- | --- |
| **Code** | **Fonction :** | **Implémentée ?** |
| FsP1-1 | Supprimer les propositions de covoiturage obsolètes | OK |
| FsP1-2 | Enregistrer toutes les propositions de covoiturage dans un historique | NON |
| FsP1-3 | Générer un « reporting » mensuel des annonces de covoiturage | NON |
| FsP1-4 | Détecter tout abus concernant la publication de propositions de covoiturage | OK |
| FsP1-5 | Mettre à jour automatiquement les propositions de covoiturage | OK |
| FsP2 | Sauvegarder/Copier une proposition de covoiturage de manière individuelle | OK |
| FsP3 | Gérer les propositions de covoiturage sauvegardées par l’utilisateur | OK |
| FsP4 | Réserver une ou plusieurs places dans la proposition de covoiturage | NON |
| FsP5 | Voir le nombre de places restantes dans la proposition de covoiturage | NON |
| FsP6 | Établir un compteur kilométrique par compte | NON |

Les fonctions supplémentaires ci-dessus n’ont pas toutes été implémentées :

Pour les fonctions administrateur : nous avons ajouté des fonctionnalités telles que la création d’une requête de suppression des annonces passées par exemple. Nous avons fait au mieux pour que la gestion soit la plus simple et intuitive possible pour le gestionnaire de l’application.

Nous avons développé la possibilité de sauvegarde des covoiturages par les utilisateurs. Cette fonction n’avait pas été mentionnée par les clients mais nous avons trouvé bon de l’intégrer à l’application pour plus de facilité dans l’utilisation.

Cependant, nous n’avons pas donné aux utilisateurs la possibilité de connaître le nombre de places disponibles en temps réel en fonction des réservations. Comme l’application sert à mettre en contact les utilisateurs pour partager un covoiturage, nous avons laissé de côté cet aspect dynamique de réservation en ligne.

Nos clients nous ont suggéré d’ajouter une fonctionnalité permettant de connaître le compteur kilométrique des utilisateurs et la quantité de CO2 émise par trajet. Nous n’avons pas intégré ces fonctionnalités car nous avons dû prioriser d’autre fonctionnalités.

**Récapitulatif des fonctions contraintes**:

|  |  |  |
| --- | --- | --- |
| **Code** | **Fonction :** | **Implémentée ?** |
| FC1 | Pouvoir être utilisé par 800 personnes simultanément | OK |
| FC2 | Utiliser un système d’authentification déjà implémenté dans l’école | OK |
| FC3 | Permettre de trouver un covoiturage en moins de 2 minutes | OK |
| FC4 | Permettre de publier une annonce en moins de 5 minutes | OK |
| FC5 | Documenter l’outil en français | OK |
| FC6 | Être accessible 7j/7, 24h/24 quel que soit le lieu | OK |
| FC7 | Utiliser le langage Perl en cas de développement d’application web | OK |

Les fonctions contraintes ont toutes été respectées. Il est juste difficile de tester la connexion simultanée de 800 utilisateurs mais nous pensons que cela est réalisable.

## Points d’amélioration

Globalement, le projet a été bien mené et nous sommes aujourd’hui satisfaits du résultat. Nous aurions cependant être plus efficaces et plus coordonnés durant cette phase de développement, notamment lors de l’organisation des jalons.

# PARTIE VII : Essais fonctionnels et structurels

Avant de donner un accès public à l’utilisation du nouvel outil, des tests seront effectués afin de savoir si la solution a été mise en place.

Les fonctions principales et secondaires devront être vérifiées :

Nous avons utilisé le plugin **Selenium** disponible sur le navigateur Firefox afin de réaliser une série de test pour garantir le bon fonctionnement de notre application. **Selenium** permet d’enregistrer une suite d’action effectué par l’utilisateur sur l’application. Nous avons regroupé les tests selon les fonctions principales de l’application (publier une annonce, rechercher une annonce, gestion utilisateur etc..).

|  |
| --- |
| **FP : Publier une proposition de covoiturage** |
| * publicationCovoiturageAlbiDateInvalide   *vérification des dates invalides.*   * publicationCovoiturageAlbiDestinationEgalDepart   *vérification annonce non conforme si destination est identique au départ*   * publicationCovoiturageAlbiValeursDeChampsManquante   *vérification annonce non conforme si certains champs/ valeurs manquent*   * publicationCovoiturageAlbiToulouseDateInvalide * publicationCovoiturageAlbiToulouseDestinationEgalDepart * publicationCovoiturageAlbiToulouseValeursDeChampsManquante * publicationCovoiturageFranceAvcEtapes * publicationCovoiturageFranceSansEtapes * publicationCovoiturageFranceDateInvalide * publicationCovoiturageFranceDestinationEgalDepart * publicationCovoiturageFranceValeursDeChampsManquantes |
| **FP : Modifier une proposition de covoiturage** |
| * modificationCovoiturageAlbiModificationOk   *vérification de la fonction de modification d’une annonce*   * modificationCovoiturageAlbiToulouseModificationOk * modificationCovoiturageFranceModificationOk * modificationCovoiturageFranceValeursDeChampsManquantes   *vérification annonce modifiée non conforme si certains / valeurs manquent* |
| **FP : Supprimer une proposition de covoiturage** |
| * suppressionAnnoncePubliée   *suppression annonce effective* |
| **FP : Rechercher une proposition de covoiturage parmi les annonces postées** |
| * rechercherAlbi0result   *vérification de la fonction rechercher avec des mauvais critères*   * rechercherAlbiValide   *vérification de la fonction rechercher avec les bons critères*   * rechercherAlbiToulouse0result * rechercherAlbiToulouseValide * rechercherFrance0result * rechercherFranceValide |
| **FP : Permettre l’accès à l’outil par l’utilisateur** |
| * modificationProfilUtilisateurNonOK   *vérification de la modification du profil utilisateur effective*   * modificationProfilUtilisateurOK |
| **FP : Permettre l’accès à l’outil au gestionnaire** |
| * modificationProfilUtilisateurAjoutSuppressionDroit   *vérification de la fonction ajout et suppression de droit*   * LieuxAlbiAjout   *Vérification de l’ajout / modification et suppression de nouveau lieu dans Albi*   * LieuxAlbiModif * LieuxAlbiSuppression |
| **FsP : Sauvegarder une proposition de covoiturage** |
| * rechercherSauvegarderAnnonceOK |
| **FsP : Gérer les propositions de covoiturages sauvegardées par l’utilisateur** |
| * suppressionAnnonceSauvegardée |

# PARTIE VII : GLOSSAIRE

Le glossaire suivant a pour but de répertorier le vocabulaire lié au projet et à son développement

|  |  |
| --- | --- |
| **ANNONCE** | Evénement créé par un conducteur décidant de partager son trajet. |
| **CLIENT** | L’association I2D, Ingénieur pour un Développement Durable |
| **CONDUCTEUR** | Individu qui propose le covoiturage (ici qui possède le véhicule). |
| **COVOITURAGE** | Le covoiturage est l'utilisation conjointe et organisée (à la différence de l'auto-stop) d'un véhicule, par un conducteur non professionnel et un ou plusieurs tiers passagers, dans le but d’effectuer un trajet commun. |
| **PASSAGER** | Personne qui participe au covoiturage en tant que voyageur ponctuel. |
| **PROPOSITION** | Proposition de covoiturage (cf « Annonce »). |
| **UTILISATEUR** | Personne qui va utiliser le service de covoiturage, aussi bien passager que conducteur. |
| **TRAJET** | Parcours du covoiturage d’un endroit à un autre. |
| **API Google Maps** | Api de Google permettant de géolocaliser des adresses sur une carte à l'aide de sa latitude et de sa longitude. |
| **OpenstreetMap** | OpenStreetMap (OSM) est un projet qui a pour but de constituer une base de données géographique libre du monde en utilisant le système GPS et d'autres données libres. |
| **Dancer** | Dancer est un application web open source dont le framework est écrit en Perl |