Introduction

Ce document a pour but d'expliquer et de rendre compte du développement des différents sites internet et CRM "Safe Driving" ainsi que de servir de document de référence pour quelqu'un qui souhaiterait reprendre le projet. Le projet "Safe Driving" a été conçu suivant un schéma de conception M.V.C (Modèle Vue Contrôleur) en se basant sur le framework Symfony. Cette partie vient rendre compte du travail fourni sur la vue, à savoir l'interface web. Elle traite de la conception des pages, de leur enchaînement, de leur design ainsi que des interactions de cette dernière avec le contrôleur.

Applications

## Site communautaire

#### Design Layout

Un pattern commun à toutes les pages est défini. Une page est découpée en 2 blocs. Le premier bloc est commun à toutes les pages. Il s'agit du fichier suivant :

-layout.php : Ce bloc contient le header du site, l'appel au fichiers javascript et css, ainsi que l'espace de connexion/affichage du nom de l'utilisateur connecté, ainsi que l'affichage des categories. La sidebar et le footer font aussi partie de la page layout, de sorte que nous n'ayons plus que le contenu dynamique a afficher (affichage/creation/modification/suppression de posts/ categories/commentaires. (Voir schema 1)

Dans l'exemple du schema 1, nous voyons que le contenu dynamique affiche tous les articles de toutes les catégories, nous pouvons evidemment afficher un article en entier, ou encore afficher tous les articles d'une seule catégorie.

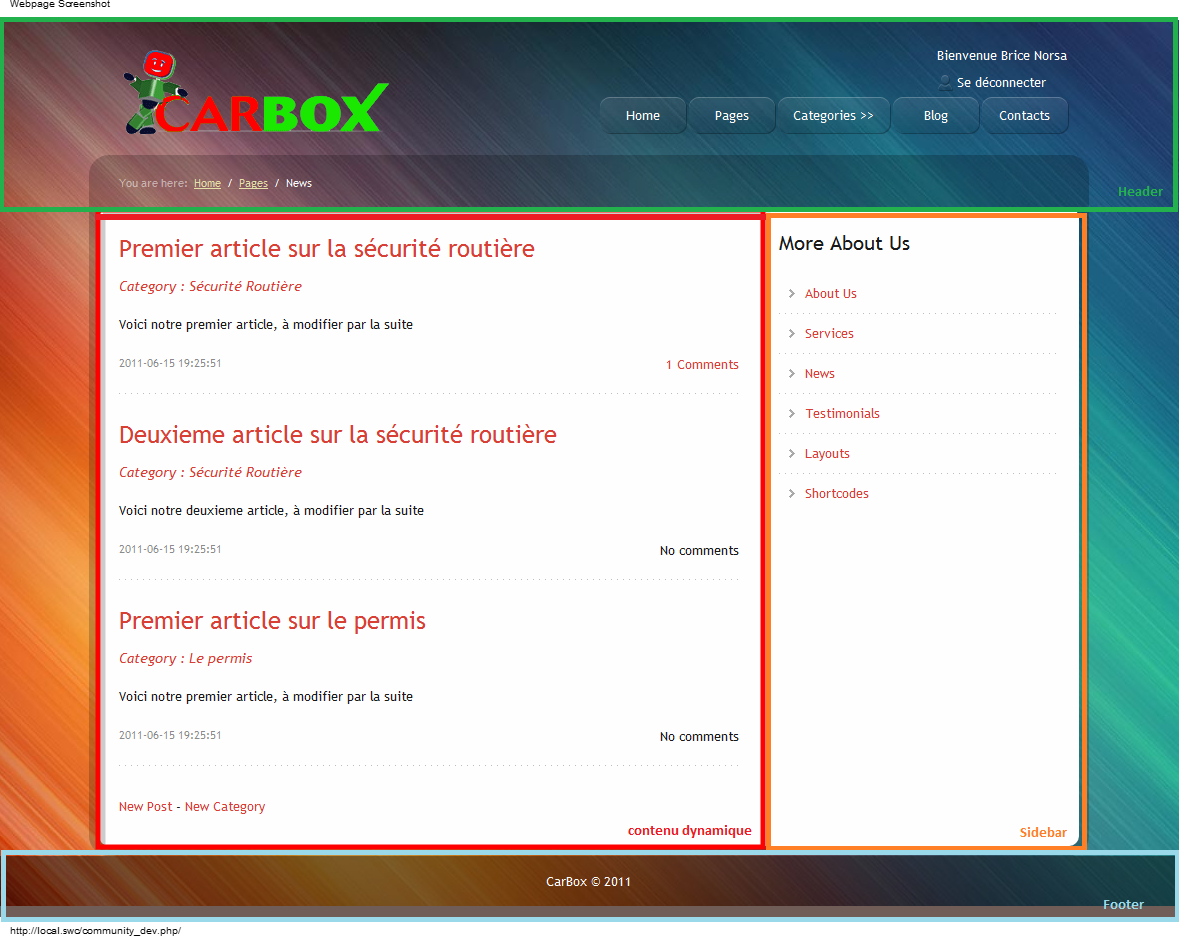


Schéma 1

### Style

Un élément important lors de la conception d'une interface web est la mise en place du CSS.

C'est en effet ce dernier qui agence les différents composants de la page et rend la page attractive ou pas. Il convient donc d'y attacher une attention particulière, puisque c'est l'une des seules choses que verra notre utilisateur lors de l'utilisation de l'application.

La mise en place du CSS s'est déroule en plusieurs étapes :

- La mise en place d'un découpage grossier pour les menus principal, les en-têtes et pieds

de pages ainsi que le contenu, dans le but d'avoir une application uniforme. Comme les

différents blocs concernés ont été réalisé rapidement, la mise en place du CSS s'est faite

rapidement.

- La mise en place du design sur chaque élément de contenu, c'est à dire les différentes pages

n'ayant pas de contenu commun avec les autres. Ceci s'est fait vers la fin du projet, étant

donné qu'il fallait attendre que ces pages soient opérationnelles pour être certains que le

style appliqué ne soit pas à modifier par la suite, pour toutes les pages autres que layout.

Choix technologique

## Partie interface

Dans la partie interface, nous avons utilisé quatre langages :

- XHTML : pour l'affichage des données statiques (ne nécessitant pas d'accès à la base de

données)

- CSS : pour la mise en page des pages web

- PHP : pour faire n'importe quelle action avant d'écrire dans

la base de données, comme un ajout de post ou de categorie par exemple. Ce langage a également été utilisé pour afficher les données retournées

par le contrôleur ;

- JavaScript : utilisé avant tout pour le confort de l'utilisateur. Il permet de réaliser des

tests en amont des vérifications faites par PHP, mais aussi et surtout parce que le JavaScript rend n'importe quel site beaucoup plus "user friendly". (facilité d'action client, ne pas recharger les pages, faire un menu dynamique, etc.)

Nous nous sommes basé sur le Framework Symfony 1.4, qui permet de très bien séparer les 3 couches (Modèle, Contrôleur, Vue). Le développement se fait sous forme d'application (une application par site). Chaque application se découpe en modules (Un module par entité logique).

Chaque module a ses contrôleurs et vue (templates).

Par exemple, pour l'application Community (qui correspond au blog), pour le module "post" (qui correspond logiquement aux articles), nous aurons le template "List" qui listera les articles (d'une même catégorie ou pas dépendant de ce que la couche modèle lui renvoit), et le template "Show" qui affichera un seul article en pleine page, avec les commentaires qui lui sont associés.

## Pourquoi Symfony ?

## Tout d'abord, utiliser Symfomy rend votre code mieux organisé. Symfony utilise une architecture bien particulière, et vous vous devez de vous y conformer ! Chaque type de composant doit être placé dans un répertoire particulier, ce qui fait qu'au final votre projet est bien ordonné. Voici un aperçu de l'organisation de Symfony.

## les applications sont placées dans un répertoire *« apps/ »*, et sont composés de contrôleurs et de composants de présentation.

* le ou les fichiers css sont dans rangés dans le répertoire ***«****web/css****»***
* les fichiers javascript vont dans ***«****web/js****»***
* les images, etc.

Par conséquent, si vous reveniez sur votre projet après 6 mois d'interruption, vous retrouveriez facilement vos marques. De même, un autre développeur familier du Framework serait en mesure de travailler sur votre code très rapidement.  
  
Une autre bonne raison d'utiliser Symfony est la virtualisation de la base de données. Symfony exploite la technologie OpenSource **Doctrine** pour vous permettre de construire les classes de vos bases, avec leurs jointures, en deux requêtes. Vous pouvez ainsi accéder aux éléments de la base de données sans écrire de requête SQL. Cela vous permet de vous assurer que votre projet n'est pas dépendant d'un type de base de données particulier.

Un autre avantage de Symfony est la génération des fichiers CRUD (Create, read, update, delete).

Enfin, dernier atout de ce Framework : la simplification des URL. Avec Symfony, les URLs ressemblent à ***«****projet/ module/create****»***, plutôt qu'à ***«****index.php?pg=module&action=create****»***. L'url rewriting est géré en natif, et paramétrable via un fichier de configuration. Il est même possible d'utiliser un suffixe comme .html pour un meilleur référencement. De plus, vous n'avez pas à vous demander comment linker tel ou tel module. Une fonction le fait pour vous.

Il rend le PHP beaucoup plus confortable, l'ajax plus abordable, l'optimisation du référencement (url rewriting) plus simple, l'internationalisation (i18n) triviale, etc. C'est un vrai régal pour les programmeurs.

Cas d'utilisation

## Site communautaire

### Création de catégorie/post

La création de catégories ou de posts se fait très simplement (comme vous pouvez le voir sur le schema 1 en bas de page), la seule condition requise est d'être logué et autorisé. Une fois connecté, si vous possédez les droits, vous pouvez créer une categorie ou un post depuis presque n'importe quelle page, par exemple sur la page "lister les posts par categorie", vous pourrez modifier la catégorie choisie, ou alors ajouter/modifier/supprimer un post très facilement en un clic. Alors que sur la page "lister les catégories", vous pouvez ajouter/modifier/supprimer une catégorie en un clic.

### Ajout de commentaire

Pour ajouter un commentaire, il suffit d'être sur la page "afficher un article". Si vous êtes connecté, vous pouvez poster un commentaire, votre nom et la date apparaitra aussi dans votre commentaire. Si vous n'êtes pas connecté, vous ne pouvez tout simplement pas ajouter de commentaire.

### Suppression de catégorie/post/commentaire

Comme dit plus haut, la supression se fait en un clic. Si vous êtes connecté et autorisé, les boutons "edit" et "delete" apparaissent automatiquement a coté de chaque catégorie, post ou commentaire. Il vous suffit de cliquer dessus.

Description de l'interfaçage

## Introduction

Une fois le contenu des pages écrit, il a fallut s'inquiéter de l'interfaçage.

En l'occurrence, seul le contrôleur nous a intéressé puisque c'est lui qui fait le lien entre

le métier (qui gère la base de données et les classes) et la vue.

On peut distinguer 3 principales parties :

- Le traitement des formulaires

- La récupération des informations de connexion

- La récupération des catégories/posts/commentaires

## Le traitement des formulaires

Nous nous devons de regarder si toutes les variables du formulaire dont on a besoin

sont renseignées. Cela se fait par une vérification d'existence suivie d'un test de vacuité, car une

variable de formulaire peut être définie mais laissée a vide. Cela peut être le cas par exemple

pour les champs de type texte.

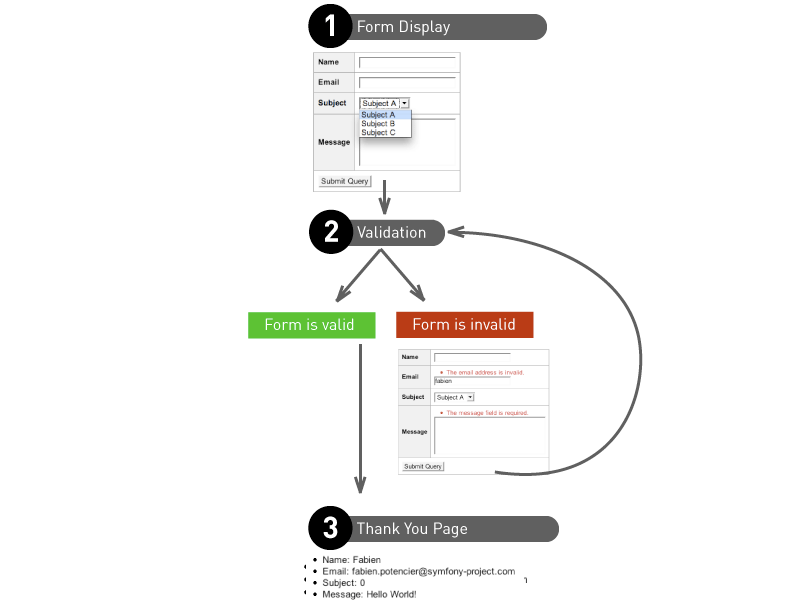
Enfin, il existe une fonction php pour se protéger des attaques par injection de code HTML. Cette fonction regarde si il y a des balises html dans les champs du

formulaire et si c'est le cas, les rend ininterprétable. On s'assure donc d'appeler cette fonction

sur chaque variable post du formulaire que l'on souhaite utiliser.

On peut noter aussi que nous avons mis en place une solution pour se protéger des attaques

par injection de code SQL.



## La récupération des informations de connexion

Un formulaire particulier est celui de la page de connexion. A sa réception nous faisons appel

au plugin sfGuardUser précédemment installé sur Symfony, celui ci nous retourne, en cas de réussite les informations nous permettant d'authentifier l'utilisateur de manière unique. En utilisant le plugin, nous assurons une homogénéité au niveau des différentes application de notre solution, car un utilisateur à besoin de créer son compte une seule fois, et peut l'utiliser sur le CRM, le site communautaire ou encore le forum.

## La récupération des catégories/posts/commentaires

Avec Symfony, la récupération des différentes entités se fait par le contrôleur, qui lui même fait appel a la couche modèle (incluant toutes les vérifications et test permettant de savoir si les objets demandés existent, etc.)

Le résultat est ensuite envoyé à la vue et est formaté pour l'affichage.

Notice d'installation

Les différentes applications ne nécessitent que peu d'installation. Un simple copier-coller des fichiers du projet suffit. Il faudra veiller a respecter l'arborescence. Cependant, l'application contenant du code php et des script JavaScript, elle ne pourra fonctionner qu'en présence de php5 sur le serveur et de JavaScript sur les clients. Ainsi si le client désactive JavaScript, l'application sera inutilisable, mais ne met pas en danger l'intégrité des données de l'application car un double contrôle est toujours effectué en php.

Notice d'utilisation du code

Le code interface développé s'insère dans le cadre du développement du projet "Safe Driving", selon

le concept M.V.C (Modèle Vue Contrôleur). Le code présenté est la vue de cette application :

c'est elle qui affiche les données et récupère les saisies de l'utilisateur.

Elle s'articule autour de trois axes : le moteur, les pages et le design. Chacun de ces éléments

est plus ou moins indépendant des autres et peut donc être modifié sans trop affecter les autres

parties.

Les pages en elles-mêmes permettent l'affichage des informations récupérées par le moteur

lors de l'interaction avec le contrôleur. Leur rôle est de mettre en page les données sous format

xHTML. L'ajout d'une page doit donc respecter cette contrainte.

Le rendu correspond a un ensemble de fichier CSS. Ceux-ci sont répartis par thèmes

(dimension, couleur, typographie) et traite donc un point particulier. Le changement de design

revient à modifier les différents fichiers proposés, ou alors de les remplacer dans l'arborescence.

Les pages étant découpées selon une certaine hiérarchie (menu, en-tête, ...), il est assez simple de

modifier le style dès l'instant que l'on réutilise les mêmes identifiants. De plus, certains éléments

de design sont communs a plusieurs pages, la mise en place du design d'une nouvelle page peut

se faire en réutilisant les identifiants existants.