

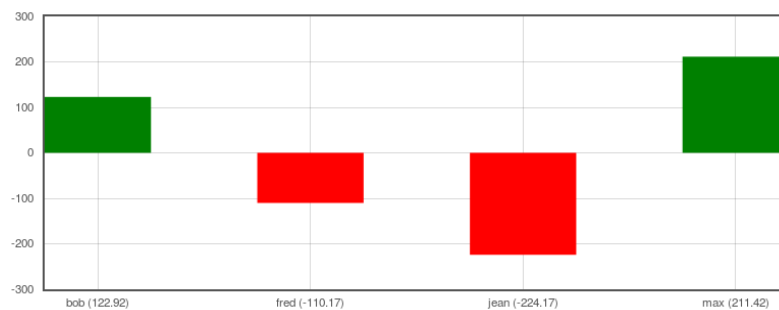
Petits comptes entre amis

Steven Liatti

Développement et services web - Prof. Stéphane Malandain

Hepia ITI 3^{ème} année

22 octobre 2017



Users

Username	Weight in event	Really payed	Part	Situation
bob	1	292.5	169.58	122.92
fred	2	237	347.17	-110.17
jean	2	125	349.17	-224.17
max	1	380	168.58	211.42

Equilibrium

fred should pay bob	110.17
jean should pay bob	12.75
jean should pay max	211.42

Table des matières

1	Introduction	3
1.1	Description	3
1.2	Technologies utilisées	3
2	Base de données	3
3	Back-end	4
3.1	Architecture MVC avec Silex	5
3.2	Composants Silex	6
3.3	Routage et configuration	6
3.4	Contrôleurs	6
3.4.1	IndexController.php	6
3.4.2	EventController.php	8
3.5	Modèles, accès aux données	10
3.6	Vues	10
4	Front-end	12
4.1	CSS avec Bootstrap	12
4.2	jQuery et Flot JS	12
5	Conclusion	12
5.1	État actuel du projet	12
5.2	Propositions d'améliorations	12

Table des figures

1	Schéma de la base de données relationnelle	4
---	--	---

Table des listings de code source

1	Arborescence du site	5
2	Configuration du login utilisateur - app/app.php	7
3	Inscription d'un utilisateur - src/Controller/IndexController.php	8
4	Calcul de l'équilibre - src/Controller/EventController.php	9
5	Insert d'un événement - src/DAO/EventDAO.php	10
6	Rendu d'une vue Twig - src/Controller/EventController.php	11
7	Vue d'une nouvelle dépense - views/new_spent.html.twig	11

1 Introduction

1.1 Description

Le but de ce mini-projet est de réaliser un site en php permettant à des amis de noter et partager les dépenses effectuées par et pour le groupe au cours de vacances communes. Lorsque lune des personnes fait des courses, par exemple, elle lenregistre. Chacun enregistre les dépenses qui concernent le groupe. Ainsi, à la fin du séjour (ou à tout moment) on peut savoir qui a payé quoi et surtout ce que chacun doit aux autres personnes du groupe damis.

1.2 Technologies utilisées

- Base de données :
 - [MySQL](#), avec
 - [MySQL Workbench](#) (pour la création du schéma)
- Back-end :
 - [Apache](#), serveur HTTP
 - [PHP](#), avec
 - [Silex](#), micro framework PHP basé entre autres sur [Symfony](#), déployé avec [Composer](#)
 - [Twig](#), moteur de templates pour PHP (utilisé de concert avec Silex)
- Front-end :
 - [Bootstrap](#) pour le design en CSS
 - [jQuery](#)
 - [Flot](#), un plugin pour dessiner des graphiques avec jQuery

2 Base de données

Les technologies imposées pour la base de données de ce travail pratique sont SQLite ou MySQL. J'ai choisi d'utiliser MySQL, car je suis familier avec. J'ai profité de cette occasion pour découvrir et utiliser Workbench, un programme permettant de modéliser les tables et relations d'une base de données de manière graphique. Une fois le model terminé, Workbench offre la possibilité de l'exporter en instructions SQL (création de tables et contraintes). Pour créer la base de données et les tables il suffit d'exécuter `db/schema.sql` dans MySQL. Il est également nécessaire d'adapter la configuration dans `app/config/prod.php`.

Mon schéma est constitué des tables suivantes :

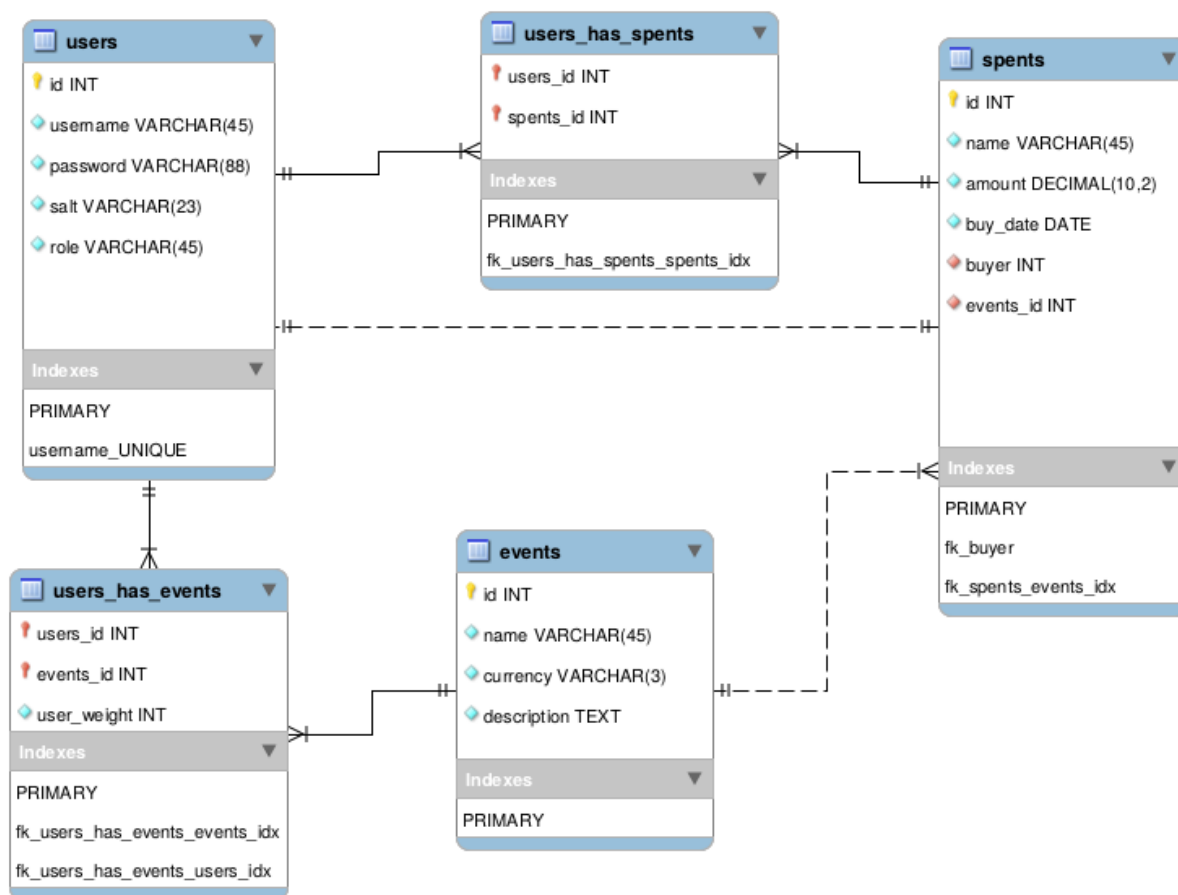


FIGURE 1 – Schéma de la base de données relationnelle

Il y a 3 tables principales : les utilisateurs, les événements et les dépenses. 2 autres tables secondaires font la liaison entre les utilisateurs et les événements et les utilisateurs et les dépenses respectivement (liaison Many-To-Many). La table des utilisateurs possède un champ salt et un autre **role**, ils sont nécessaires au fonctionnement de Silex (explications plus loin). Le poids de chaque utilisateur au sein d'un événement est indiqué dans la table croisée users_has_events. Chaque dépense référence l'acheteur (dans users) et l'événement lié (dans events). Ce schéma représente l'interface minimum pour les données de ce travail, mais il a l'avantage d'être simple à comprendre et à maintenir.

3 Back-end

Je profite également de ce TP pour appréhender Silex, un micro framework PHP dérivé de Symfony (que j'ai eu l'occasion de tester), beaucoup plus léger que son grand frère mais tout de même robuste et modulaire. Il bénéficie d'un grand nombre de modules à ajouter, en vrac : système de templates, connexion à la base de données, routes, etc.

3.1 Architecture MVC avec Silex

Grâce à Silex, mon architecture respecte le design pattern [MVC](#). On peut configurer Silex pour que le code source, défini dans le répertoire `src`, soit ajouté au mécanisme de chargement automatique (autoloading) géré par Composer. Pour que cela fonctionne, il faut que le code source respecte le standard [PSR-4](#). Voici l'arborescence du site :

```
|-- app
|   |-- app.php
|   |-- config
|   |   |-- dev.php
5 |   |   |-- prod.php
|   |-- routes.php
|-- composer.json
|-- composer.lock
|-- src
10 |   |-- Controller
|   |   |-- EventController.php
|   |   |-- IndexController.php
|   |-- DAO
|   |   |-- DAO.php
15 |   |   |-- EventDAO.php
|   |   |-- SpentDAO.php
|   |   |-- UserDAO.php
|   |-- Entity
|       |-- Event.php
20 |       |-- Spent.php
|       |-- User.php
|-- var
|   |-- logs
|-- vendor
25 |-- views
|   |-- error.html.twig
|   |-- event.html.twig
|   |-- index.html.twig
|   |-- layout.html.twig
30 |   |-- new_event.html.twig
|   |-- new_spent.html.twig
|-- web
|   |-- css
|       |-- style.css
35 |   |-- images
|       |-- 404-ghost.png
|-- index.php
|-- js
    |-- jquery.flot.min.js
```

Listing 1 – Arborescence du site

- app : fichiers de config et routes
- composer.json : fichier de dépendances
- src : fichiers source PHP ("POPO", DAO, Contrôleurs)
- var : logs
- vendor : fichiers sources des composants Silex/Symfony
- views : fichiers de vues en Twig
- web : fichiers CSS, JS, images, etc. publics livrés au client

3.2 Composants Silex

Silex fournit plusieurs composants vraiment pratiques et très efficaces. Les composants suivants m'ont été particulièrement utiles :

- Un composant de sécurité qui gère le login des utilisateurs
- Un composant de construction de formulaires
- Un composant pour se connecter à la base de données
- Un composant de rendu de templates (Twig)
- Un composant permettant de facilement appliquer des contraintes pour valider les données

3.3 Routage et configuration

Dans le dossier app se trouvent les fichiers de configuration (pour MySQL notamment) et les déclarations/configurations des modules et composants récupérés grâce à Composer (dans app/app.php). Le point d'entrée du site est web/index.php. C'est le contrôleur frontal, c'est par ici que toutes les requêtes passent (grâce au fichier de configuration d'Apache .htaccess). Il instancie l'objet principal \$app et fait suivre aux fichiers de routes. Silex permet de définir des routes, c'est-à-dire des points d'entrée dans l'application. À chaque route est associée une action (requête GET et/ou POST) définie dans un contrôleur. Pour ce site, j'ai défini 6 routes : page d'accueil (avec formulaires de connexion et d'inscription), page d'un événement, nouvel événement, nouvel événement, suppression d'événement et suppression de dépense.

3.4 Contrôleurs

Les contrôleurs sont le socle de ce site, c'est ici que sont récupérées les requêtes et les données (grâce aux DAO) et construits les formulaires et les vues. Une fonction xxxAction() d'un contrôleur a pratiquement toujours la forme suivante : test si la page est accessible par le client qui la demande, récupération des données (voir la sous-section 3.5), si besoin construction/vérification des formulaires et enfin rendu de la vue ou redirection vers une page donnée.

3.4.1 IndexController.php

Ce contrôleur gère les actions de la page principale. Si un utilisateur est connecté, il lui affiche les événements auxquels il participe. Sinon, deux formulaires, de login et d'inscription, sont affichés. Grâce au composant de sécurité intégré, j'ai facilement pu mettre en place le login utilisateur. Il m'a suffi de configurer quelques réglages dans app/app.php : la route du

formulaire de login et celle de la vérification du login (automatiquement faite), si les clients anonymes peuvent accéder à la page principale, la provenance des données utilisateurs, etc. (voir listing 2) :

```
// For login of users
$app->register(new Silex\Provider\SecurityServiceProvider(), array(
    'security.firewalls' => array(
        'secured' => array(
            'pattern' => '^/',
            'anonymous' => true,
            'logout' => true,
            'form' => array('login_path' => '/', 'check_path' =>
'/login_check'),
            'users' => function () use ($app) {
                return new Pcea\DAO\UserDAO($app['db']);
            },
        ),
    ),
));
```

Listing 2 – Configuration du login utilisateur - app/app.php

Si l'utilisateur s'inscrit, son formulaire est construit grâce au composant Silex, et lorsqu'il est reçu valide et conforme en retour, son mot de passe est hashé et le nouvel utilisateur est inscrit en base de données (voir listing 3) :

```
else {
    $saltLength = 23;
    $user = new User();

    $userForm = $app['form.factory']->createBuilder(FormType::class,
$user)
        ->add('username', TextType::class)
        ->add('password', RepeatedType::class, array(
            'type' => PasswordType::class,
            'invalid_message' => 'The password fields must match.',
            'options' => array('required' => true),
            'first_options' => array('label' => 'Password'),
            'second_options' => array('label' => 'Repeat password'),
        ))
        ->getForm();

    $userForm->handleRequest($request);
    if ($userForm->isSubmitted() && $userForm->isValid()) {
        // generate a random salt value
        $salt = substr(md5(time()), 0, $saltLength);
        $user->setSalt($salt);
        // compute the encoded password
```

```

        $password =
$app['security.encoder.bcrypt']->encodePassword($user->getPassword(),
$salt);
        $user->setPassword($password);

45
        try {
            $app['dao.user']->create($user);
        } catch (UniqueConstraintViolationException $e) {
            $app['session']->getFlashBag()->add('error',
'Username already taken.');
```

```

50
            return $app['twig']->render('index.html.twig', array(
                'userForm' => $userForm->createView()
            ));
        }

55
        $app['session']->getFlashBag()->add('success',
'User successfully created.');
```

```

    }

```

Listing 3 – Inscription d'un utilisateur - src/Controller/IndexController.php

3.4.2 EventController.php

Ce contrôleur gère les actions des pages des événements et des dépenses. Les actions de créer et supprimer dépenses et événements sont semblables, le formulaire est créé, envoyé au client, au retour il est vérifié et si tout est en ordre on enregistre la nouvelle entrée en base de données. L'action d'afficher l'événement avec ses dépenses et l'équilibre est un peu plus longue. Je commence par effectuer un tableau des parts que chaque membre puis je calcule la situation pour chacun (positive ou négative) et finalement, en parcourant plusieurs fois le tableau des situations je une manière de créer l'équilibre (savoir qui doit combien à qui) (voir listing 4) :

```

        // At this time, we have in situations the
        // positive or negative amount per user.
        // The next while compute debts array
        // to show who must pay who.

60
        $gaps = $situations;
        $isBalanced = false;
        $posCursor = -1;
        $negCursor = -1;

65

        while (!$isBalanced) {
            // find positive and negative values
            foreach ($gaps as $key => $value) {
                if ($posCursor < 0 && $value > 0) {
70
                    $posCursor = $key;
                }
                if ($negCursor < 0 && $value < 0) {

```



```

        $negCursor = $key;
    }
}

    if ($posCursor >= 0 && $negCursor >= 0) {
        $balance = $gaps[$posCursor] + $gaps[$negCursor];
        if ($balance < 0) {
            $debts[] = array(
                "from" =>
$sevent->getUsers()[$negCursor]->getUsername(),
                "howMuch" => $gaps[$posCursor],
                "to" => $sevent->getUsers()[$posCursor]->getUsername()
            );
            $gaps[$negCursor] = $balance;
            $gaps[$posCursor] = 0;
            $posCursor = -1;
        }
        elseif ($balance > 0) {
            $debts[] = array(
                "from" =>
$sevent->getUsers()[$negCursor]->getUsername(),
                "howMuch" => abs($gaps[$negCursor]),
                "to" => $sevent->getUsers()[$posCursor]->getUsername()
            );
            $gaps[$posCursor] = $balance;
            $gaps[$negCursor] = 0;
            $negCursor = -1;
        }
        else {
            $debts[] = array(
                "from" =>
$sevent->getUsers()[$negCursor]->getUsername(),
                "howMuch" => $gaps[$posCursor],
                "to" => $sevent->getUsers()[$posCursor]->getUsername()
            );
            $isBalanced = true;
        }
    }
    else {
        $isBalanced = true;
    }
}

```

Listing 4 – Calcul de l'équilibre - src/Controller/EventController.php

3.5 Modèles, accès aux données

L'accès aux données avec Silex/Symfony se fait généralement avec l'ORM [Doctrine](#) (l'utilisation de base est semblable à PDO PHP). Tout ce qui touche cet accès se trouve dans les classes `src/DAO/xxxDAO.php`, avec pour chaque fichier les requêtes traitant avec la table du même nom. La classe `src/DAO/UserDAO.php` a une contrainte supplémentaire, elle implémente l'interface `UserProviderInterface` nécessaire au fonctionnement du module de sécurité/login des utilisateurs. Pour les besoins de la construction des formulaires, des vues et pour un code plus modulaire, chaque table est représentée par un "POPO", un simple objet PHP avec attributs et accesseurs. Voici par exemple l'insertion d'un nouvel événement en base de données (voir listing 5) :

```
public function create(Event $event, $weight) {  
    $eventData = array(  
        'name' => $event->getName(),  
        'description' => $event->getDescription(),  
        'currency' => $event->getCurrency()  
    );  
  
    $this->getDb()->insert('events', $eventData);  
    $id = $this->getDb()->lastInsertId();  
    $event->setId($id);  
  
    foreach ($event->getUsers() as $userId) {  
        $usersEventsData = array(  
            'users_id' => $userId,  
            'events_id' => $event->getId(),  
            'user_weight' => $weight[$userId]  
        );  
  
        $this->getDb()->insert('users_has_events', $usersEventsData);  
    }  
  
    return $event;  
}
```

Listing 5 – Insert d'un événement - `src/DAO/EventDAO.php`

3.6 Vues

La création d'une vue avec Twig se fait vraiment simplement. Twig est un moteur de templates PHP, il a sa propre syntaxe épurée destinée à des structures de contrôles simples pensées pour l'affichage. Il permet de faire de l'inclusion intelligente de templates : le fichier de vue parent est `views/layout.html.twig`, il définit plusieurs blocs que les templates enfants pourront "étendre", un peu à la manière de l'héritage en POO, ce qui facilite la réutilisation de code. De concert avec le composant de construction de formulaires, il offre une manière simple et efficace pour générer ses formulaire. Côté contrôleur, on envoie les variables et tableaux au moment du rendu de la vue (voir listing 6). Côté vue Twig, on y accède simplement grâce à

la syntaxe `variable.attribut` (il faut au préalable que la variable soit représentée par un objet ou tableau PHP et possède des getters) (voir listing 7).

```
230     return $app['twig']->render('new_spent.html.twig', array(
        'title' => 'New spent',
        'spentForm' => $spentForm->createView(),
        'eventId' => $eventId
    ));
```

Listing 6 – Rendu d'une vue Twig - `src/Controller/EventController.php`

```
{% extends 'layout.html.twig' %}
{% form_theme spentForm 'bootstrap_3_horizontal_layout.html.twig' %}
{% block title %}{{ title }}{% endblock %}

5 {% block content %}
<h2 class="text-center">{{ block('title') }}</h2>

<p><a href="{{ path('event', { 'eventId': eventId }) }}">Return to
event</a></p>

10 <div class="well">
    {{ form_start(spentForm) }}
        {{ form_row(spentForm.name) }}
        {{ form_widget(spentForm.name) }}

15     {{ form_row(spentForm.amount) }}
        {{ form_widget(spentForm.amount) }}

        {{ form_row(spentForm.buyDate) }}
        {{ form_widget(spentForm.buyDate) }}

20     {{ form_row(spentForm.buyer) }}
        {{ form_widget(spentForm.buyer) }}

        {{ form_row(spentForm.users) }}
25     {{ form_widget(spentForm.users) }}

    <div class="form-group">
        <div class="col-sm-offset-5 col-sm-3">
            <input type="submit" class="btn btn-primary" value="Save" />
30        </div>
    </div>
    {{ form_end(spentForm) }}
</div>
{% endblock %}
```

Listing 7 – Vue d'une nouvelle dépense - `views/new_spent.html.twig`

4 Front-end

4.1 CSS avec Bootstrap

4.2 jQuery et Flot JS

5 Conclusion

5.1 État actuel du projet

Toutes les fonctionnalités demandées sont implémentées, pas de bugs connus.

5.2 Propositions d'améliorations

- Tests unitaires
- Dans une situation réelle de production, on ne laisserait pas un utilisateur choisir parmi tous les autres utilisateurs au moment de créer un nouvel événement. Il faudrait plutôt ajouter des adresses email, d'utilisateurs déjà inscrits (ou non) sur le site, ainsi chaque invité peut rejoindre l'événement.