

heig-vd

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



PDG – Projet de groupe – 2016

EasyGoing! – Rapport

Thibaud Duchoud
Karim Ghoslani
Michelle Vanessa Meguep Sakam
Raphaël Racine
Miguel Santamaria

A l'intention de Dr. René Rentsch

Table des matières

Introduction.....	4
Généralité.....	4
Objectifs du projet.....	4
Technologies utilisées et conventions de codage.....	5
Technologies.....	5
Conventions de codage.....	5
Général.....	5
Nommage.....	5
Autres.....	5
Notes importantes.....	6
Base de données.....	7
Conception.....	7
Schéma UML.....	7
Entités.....	7
Associations.....	8
Contraintes d'intégrité.....	9
Réalisation.....	10
Schéma relationnel.....	10
Tables.....	11
Utilisateurs.....	11
Projets.....	12
Tâches.....	14
Evénements.....	16
Vues.....	18
Fonctions et triggers.....	20
Fonctions.....	20
Triggers.....	21
Implémentation des différentes contraintes d'intégrité.....	21
Le framework Zend.....	22
Motivations.....	22
Fonctionnement.....	22
Introduction.....	22
Ajouter un contrôleur.....	23
Ajouter un modèle.....	25
Détails d'implémentation de l'application EasyGoing.....	27
Schéma UML.....	27
Rôles des contrôleurs.....	28
UserController.....	28
ProjectsController.....	29
ProjectController.....	29
AboutController.....	32
TutorialController.....	32
Architecture des fichiers du projet.....	33
Interface graphique.....	34
Inscription.....	35
Session.....	35

Cookies.....	36
Cookie d'inscription.....	36
Cookie de spécialisation d'un membre.....	36
Tutoriel.....	37
Réalisation.....	37
Partie serveur.....	37
Modèle « Tutorial ».....	37
Partie client.....	38
Javascript.....	38
Utilisation dans une page.....	39
Tooltip.....	39
Droits dans l'application.....	39
Fonctionnement du « Dashboard ».....	40
Généralités.....	40
Action sur les tâches.....	42
Edition (1).....	42
Suppression (2).....	43
Création d'une sous-tâche (4).....	43
Désassignation (3).....	43
Assignation.....	43
Afficher les détails d'une tâche.....	44
Déplacer une tâche.....	45
Action sur les sous-tâches.....	46
Rappel sur les sous-tâches.....	46
Edition (1).....	47
Suppression (2).....	47
Serveurs secondaires d'événements.....	47
Introduction.....	47
Diagramme de fonctionnement.....	48
Rôle des serveurs.....	49
Processus de connexion d'un client au serveur.....	52
Conclusions.....	53
Problèmes rencontrés.....	53
Problèmes organisationnels.....	53
Difficultés techniques.....	53
Points positifs.....	54
Fonctionnalités.....	54
Améliorations futures.....	55
Conclusion finale.....	56

Introduction

Généralité

PROJET

Nom du projet	EasyGoing
Début du projet	14.09.2015
Fin du projet	04.01.2016
Durée effective	90 heures

ACTEURS

Répondant	Dr. René Rentsch	rene.rentsch@heig-vd.ch
Développeurs	Miguel Santamaria (Chef de projet)	miguel.santamaria@heig-vd.ch
	Raphaël Racine (Suppléant)	raphael.racine@heig-vd.ch
	Thibaud Duchoud	thibaud.duchoud@heig-vd.ch
	Michelle Vanessa Meguep Sakam	michelle.meguepsakam@heig-vd.ch
	Karim Ghozlani	karim.ghozlani@heig-vd.ch

Objectifs du projet

Ce projet a pour objectif le développement d'un site web permettant la gestion de projets. Il permet aux membres d'une équipe de gérer un projet autour d'un « dashboard ». Il offre la possibilité de visualiser différentes tâches, qui travaille sur l'une de ces dites tâches et l'avancement de celle-ci.

Des événements ont été mis en place et ceux-ci permettent de suivre les différentes actions effectuées par un utilisateur (par exemple lors du déplacement d'une tâche ou de l'ajout d'un membre au projet). Ce système est conçu comme un fil d'actualité.

Notre projet dispose également d'un tutoriel d'utilisation qui facilite grandement la tâche des nouveaux venus et d'une grande ergonomie

Technologies utilisées et conventions de codage

Technologies

Voici la liste des technologies utilisées :

- **Apache v2.4** comme serveur http.
- **PHP v5.6** comme langage coté serveur.
 - **Zend Framework 2** comme framework php.
- **MySQL v5.6** comme moteur de base de données.
 - **PHPMysqlAdmin v4.4 et MySQL Workbench v6.3** pour gérer la base de données.
- **HTML – CSS** comme langages de structure et design des pages web.
- **Javascript/JQuery v2.1** comme langage/bibliothèque pour dynamiser les pages web.
- **Différentes bibliothèques Javascript** (comme Bootstrap) pour réaliser des actions spécifiques concernant la gestion des boards principalement (cliquer-déposer, animation, gestion d'une grille ...). En voici une liste non-exhaustive :
 - **ContextMenu** : Un menu contextuel en JQuery qui est utilisé pour faire différentes actions sur les tâches qui se trouvent dans le board.
 - **Bootbox** : C'est une bibliothèque JQuery qui remplace les fenêtres d'alerte classique de Javascript par des fenêtres plus jolies.
 - **Daterangepicker et Datetimepicker** : Des calendriers JQuery qui permettent de sélectionner respectivement la date et l'heure.

Conventions de codage

Général

Les conventions de codage que nous avons utilisées sont inspirées de celles utilisées dans le framework Zend.

Tout doit être écrit en anglais.

Nommage

Les noms de fonctions, de variables et de classes doivent être descriptifs : il faut éviter les abréviations.

Les noms des fichiers PHP correspondent à la classe qu'il contient.

Les variables et fonctions sont écrites en camelCase (première lettre en minuscule et chaque nouveau mot commence par une majuscule).

Pour les variables de classes (membres), si elles sont « protected » ou « private » ont les précède d'un « _ ».

Les méthodes respectent les mêmes conventions que les variables de classes.

Les constantes sont écrites en majuscule et chaque mot est séparé par un « _ ».

Les classes sont écrites en PascalCase (chaque mot commence par une majuscule).

Autres

En PHP, la première accolade suivant une instruction se place après un retour à la ligne. En Javascript, la première accolade se met directement après l'instruction.

L'indentation se compose de 3 espaces (donc pas de tabulation).

Notes importantes

La suite de ce document contient de nombreuses images tirées directement de notre application. Ayant avancé la documentation tout au long du semestre, il se peut que la GUI de certaines de ces captures ne soit plus parfaitement ressemblantes à la réalité, car de nombreux changements d'interface ont eu lieu depuis. Cependant, toutes les fonctionnalités sont à chaque fois présentes, et la réaction des page est la même que ce qui est actuellement fait ; seul l'interface est susceptible d'avoir changé.

Nous recommandons fortement au client de ne pas installer l'environnement Zend sur sa machine, car l'installation peut s'avérer complexe, et beaucoup de technologies entrent en jeu. Nous avons pu héberger momentanément le site sur le domaine <http://easygoing.my-chic-paradise.com/>, qui sera accessible jusqu'au rendu final des notes du projet.

Cependant, si l'installation sur une machine tierce est tout de même désirée, les instructions sont contenues dans le fichier README.txt fourni avec l'application.

Finalement, il est à noter que l'envoi d'e-mails ne fonctionnera pas si le projet est testé sur une machine locale, à moins qu'un serveur mail ait été configuré dessus, car l'application en a besoin d'un pour pouvoir les envoyer. En revanche, il est entièrement fonctionnel sur le domaine mis à disposition.

Base de données

Conception

Schéma UML

Voici dans un premier temps notre modèle de données conceptuel (sous forme d'un schéma UML) basé sur celui présenté dans cahier des charges, et agrémenté de légères modifications qui nous ont parues nécessaires au fil de l'avancement du projet :

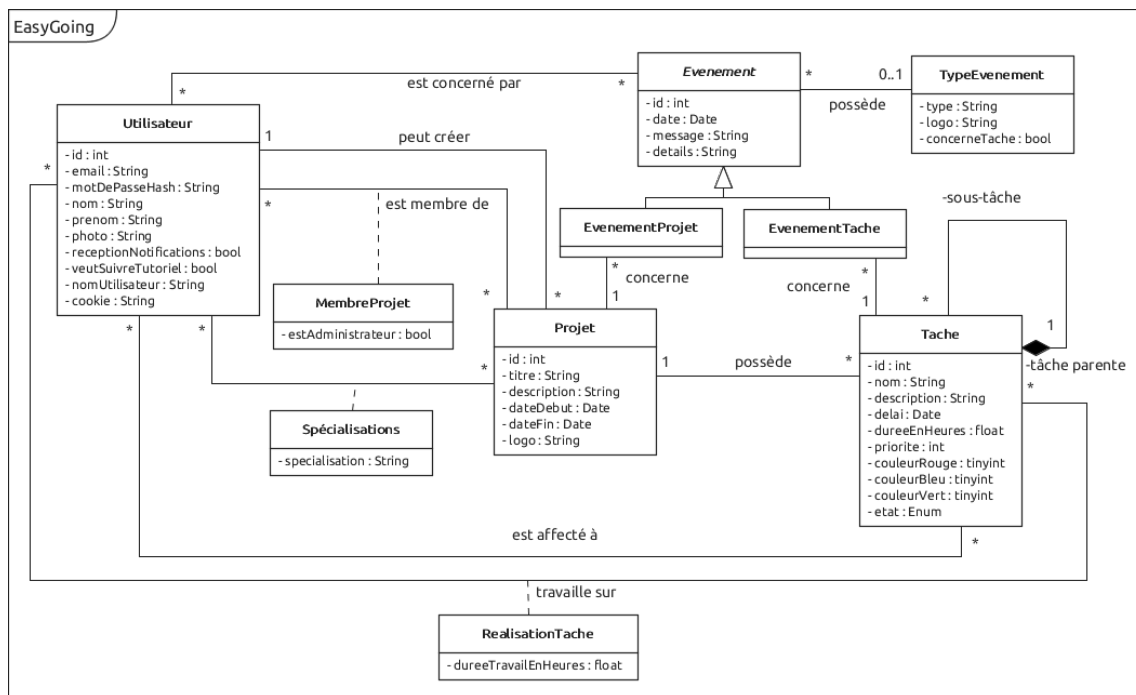


Illustration 1: modèle de domaine de la base de données.

Remarque : Les entités principales possèdent chacune un id qui sera auto-généré dans la base de données.

Entités

Voici l'explication des entités principales apparaissant sur le modèle. Les données marquées en italique sont des exemples d'utilisation.

Entité	Explication
Projet	Cette entité représente un projet. Chaque projet est caractérisé par : <ol style="list-style-type: none"> 1. Un titre <i>Projet PDG 2015</i> 2. Une description <i>Projet de semestre à la HEIG-VD</i> 3. Une date de début <i>01.10.2015</i> 4. Une date de fin <i>04.01.2016</i>
Tâche	Cette entité représente une tâche d'un projet. Elle est caractérisée par : <ul style="list-style-type: none"> - Un nom <i>Faire la vaisselle</i> - Un délai <i>05.11.2015</i> - Une description <i>Mettre toutes les assiettes dans le lave-vaisselle</i>

	<ul style="list-style-type: none"> - Une durée en heures □ <i>La durée estimée de la tâche en heures</i> - Une priorité □ <i>Haute</i>
Utilisateur	<p>Cette entité représente un utilisateur. Chaque utilisateur est caractérisé par :</p> <ul style="list-style-type: none"> (1) Un email □ raphael.racine@heig-vd.ch (2) Un nom d'utilisateur □ <i>raphy</i> (3) Un mot de passé (sous forme de hash) □ <i>ebfh451v65sd1561dfsd1sdav5s6a</i> (4) Un nom □ <i>Raphaël</i> (5) Un prénom □ <i>Racine</i> (6) Un booléen « réception des notifications » (pour savoir si l'utilisateur désire recevoir des notifications ou pas) □ <i>True</i> (7) Un booléen « veut suivre tutoriel » (pour savoir si l'utilisateur désire avoir le tutoriel qui apparaît) □ <i>False</i> (8) Un cookie qui lui permettra de se reconnecter □ <i>14561f8avsd1fsfsda4fsa</i>
Événement	<p>Cette entité représente un événement (d'une tâche ou d'un projet). Un événement est caractérisé par :</p> <ul style="list-style-type: none"> • Une date □ <i>28.01.2016</i> • Un message □ <i>raphaelracine a terminé la tâche X</i> • Les détails de l'événement

Associations

Voici l'explication des différentes associations apparaissant sur le modèle.

Relation	Cardinalité	Explication
Projet et Utilisateur (MembreProjet)	N - N	<p>Cette relation représente le fait qu'un utilisateur soit membre d'un projet.</p> <p>Comme il s'agit d'une relation plusieurs à plusieurs, il y aura une entité intermédiaire nommé MembreProjet et cette dernière indiquera si le membre en question est administrateur du projet ou non (booléen « est administrateur »).</p>
Projet et Utilisateur (Spécialisations)	N - N	<p>Cette relation représente le fait qu'un membre puisse avoir plusieurs spécialisations dans un projet. <i>Par exemple le membre X est spécialisé dans « Java » et dans « Base de données » pour ce projet.</i></p> <p>Il est à noter que ce n'est pas la même association que celle citée précédemment (MembreProjet) car sinon si un membre aurait N spécialisations pour le même projet, il faudrait dire N fois qu'il y a le même membre dans le même projet (1 fois par spécialisation) ce qui serait redondant.</p>
Projet et	N – 1	Cette relation représente le fait qu'un utilisateur a

Utilisateur		créé un projet (elle permettra de savoir qui a créé le projet...)
Projet et Tâche	1 - N	Cette relation représente simplement le fait qu'une tâche fasse partie d'un projet. Un projet pouvant avoir plusieurs tâches.
Utilisateur et Tâche (affectation)	N - N	Cette relation indique le fait qu'un utilisateur soit affecté à une tâche. Une tâche peut être affectée à plusieurs utilisateurs. Elle permet de savoir qui doit réaliser une partie de chaque tâche.
Utilisateur et Tâche (réalisation)	N – N	Cette relation indique le fait qu'un utilisateur à travailler un certain temps (durée en heures) sur une tâche. <i>Par exemple le membre X a travaillé 3 heures sur la tâche Y.</i>
Tâche et Tâche	1 – N	Cette relation représente le fait qu'une tâche peut avoir plusieurs sous-tâches. <i>Par exemple faire la vaisselle c'est :</i> <ul style="list-style-type: none"> • Nettoyer les assiettes • Nettoyer les verres
Utilisateur et Événement	N – N	Cette relation représente le fait qu'un utilisateur soit concerné par un événement. Un événement peut concerner plusieurs utilisateurs.
Événement et TypeÉvénement	N – 1	Cette relation indique qu'un événement possède un type.
Événement et ÉvénementProjet	Héritage	L'entité EvenementProjet concerne un projet et elle hérite de l'entité Événement. Elle a comme caractéristique supplémentaire le fait qu'elle concerne un projet.
Événement et ÉvénementTâche	Héritage	L'entité EvenementTache concerne une tâche et elle hérite de l'entité Événement. Elle a comme caractéristique supplémentaire le fait qu'elle concerne une tâche.
Projet et ÉvénementProjet	1 – N	Cette relation représente simplement le fait qu'un événement de projet est lié à un projet.
Tâche et ÉvénementTâche	1 – N	Cette relation représente simplement le fait qu'un événement de tâche est lié à une tâche.

Contraintes d'intégrité

Voici les contraintes d'intégrités principales liées à notre base de données :

- Membre projet : Un utilisateur ne peut pas être 2 fois membre d'un même projet, et il doit y avoir au moins un administrateur par projet
- Affectation de tâche : Un utilisateur ne peut pas être affecté à une tâche s'il n'est pas membre du projet dans lequel la tâche est inscrite
- Réalisation de tâche : Un utilisateur ne peut pas réaliser une tâche s'il n'y est pas affecté
- Sous tâche : Une tâche qui a déjà une tâche parente ne peut pas avoir de sous-tâche. Autrement dit, on s'arrête à un seul niveau de sous-tâche.

Réalisation

Schéma relationnel

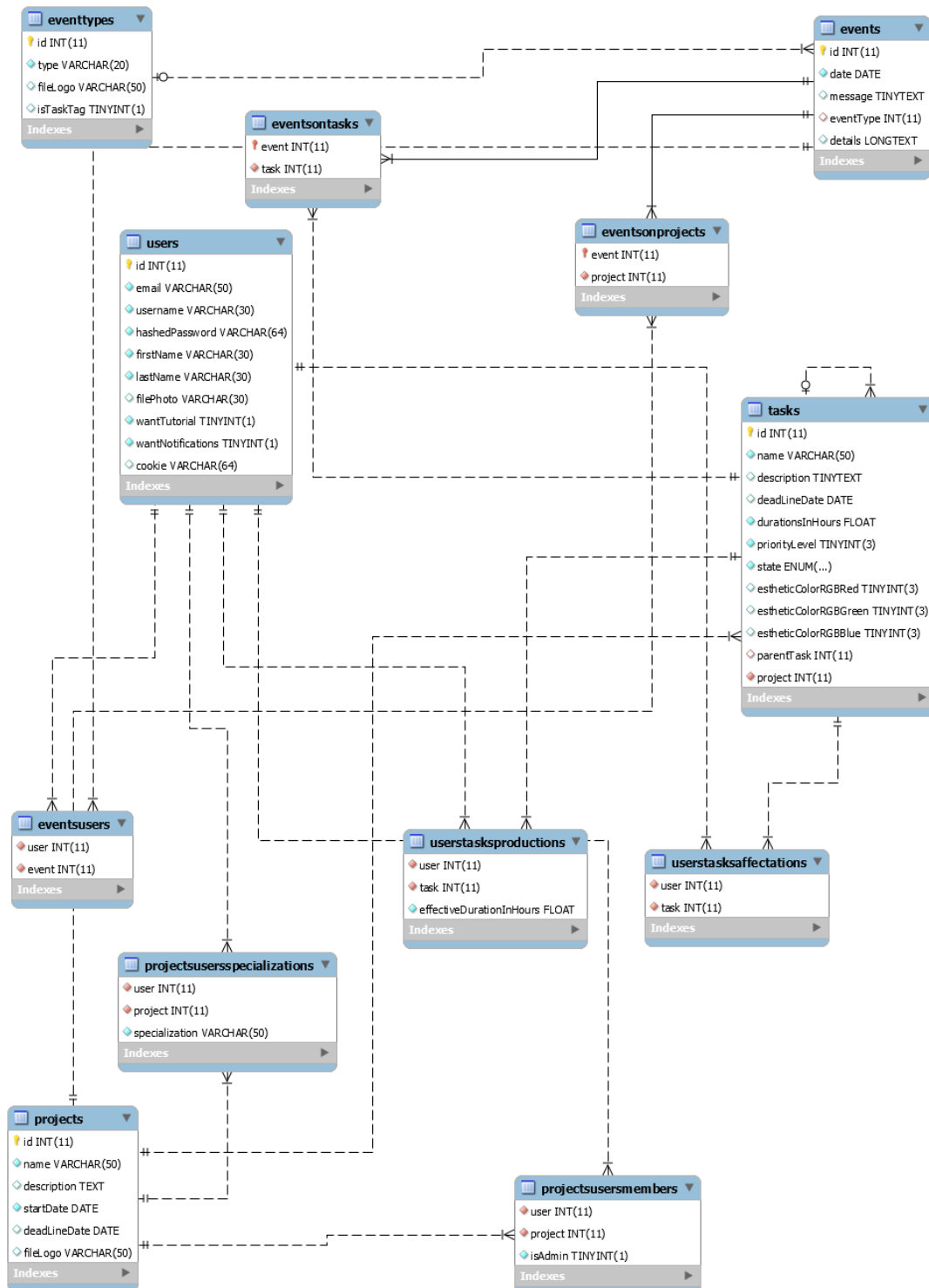


Illustration 2: schéma relationnel de la base de données

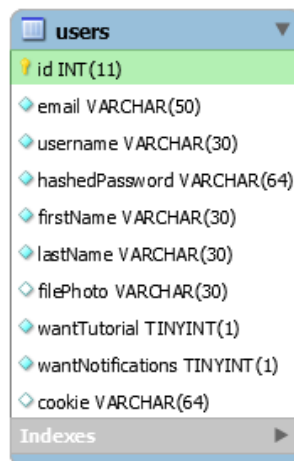
Tables

Voici une description des différentes tables de la base de données, regroupées par catégories.

Utilisateurs

Ici est expliquée la table concernant les utilisateurs :

- Users



Dessin 1: schéma relationnel - Utilisateurs

Table	Champs	Type	Explication
Users	id	INT NOT NULL PRIMARY KEY AUTO_INCREMENT	Clé primaire de la table
	email	VARCHAR(50) NOT NULL UNIQUE	Email de l'utilisateur
	username	VARCHAR(30) NOT NULL UNIQUE	Pseudonyme de l'utilisateur
	hashedPassword	VARCHAR(64) NOT NULL	Mot de passe sous sa forme hashée (algorithme SHA-256)
	firstName	VARCHAR(30) NOT NULL	Prénom de l'utilisateur
	lastName	VARCHAR(30) NOT NULL	Nom de famille de l'utilisateur
	filePhoto	VARCHAR(30)	Nom du fichier de la photo de l'utilisateur
	wantTutorial	BOOLEAN NOT NULL DEFAULT TRUE	Indique si l'utilisateur désire le tutoriel ou non

	wantNotifications	BOOLEAN NOT NULL DEFAULT TRUE	Indique si l'utilisateur désire avoir des notifications
	cookie	VARCHAR(64)	Cookie de l'utilisateur afin qu'il puisse se reconnecter

Projets

Ici sont expliquées les tables concernant les projets :

- Projects
- ProjectsUsersSpecializations
- ProjectsUsersMembers

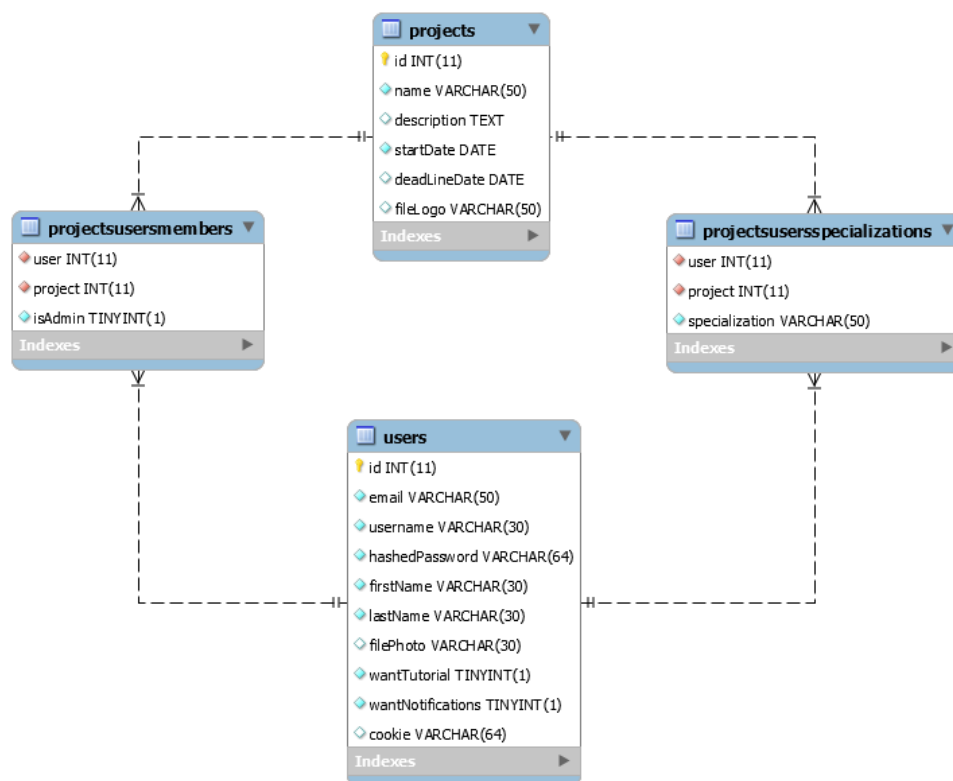


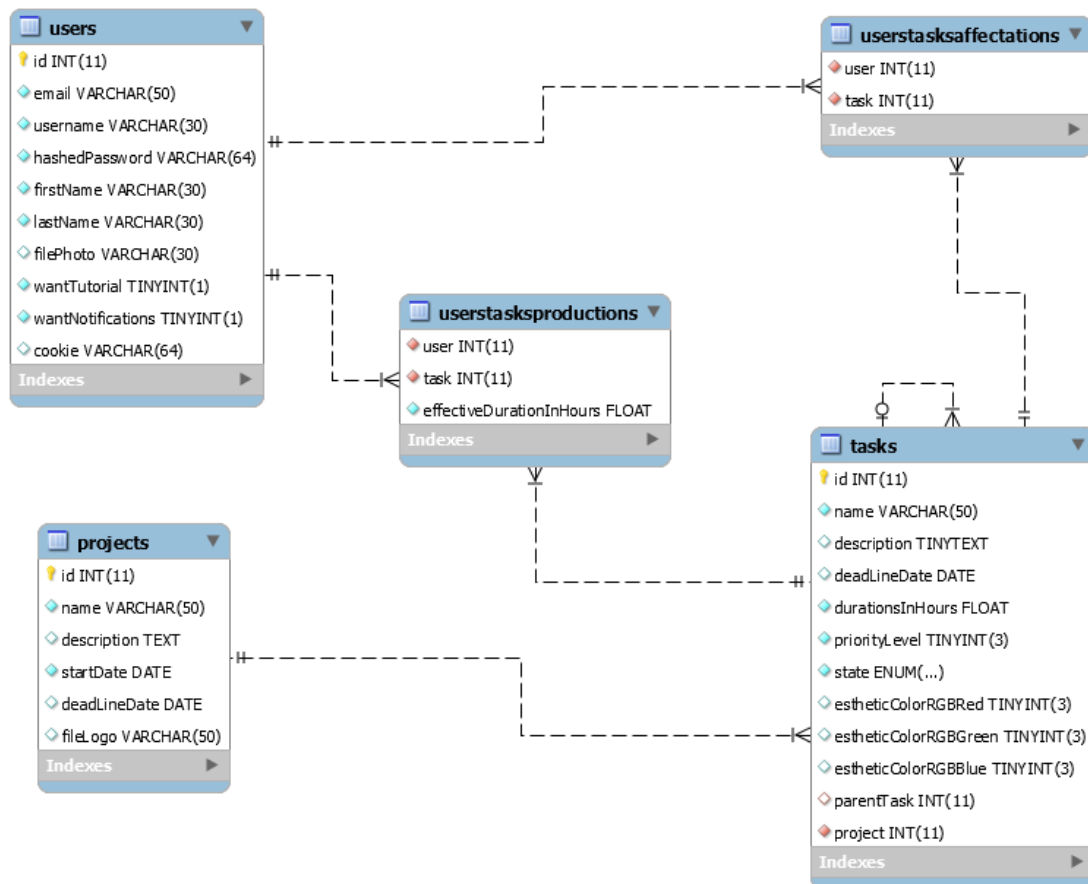
Illustration 3: schéma relationnel - Projets

Table	Champs	Type	Explication
Projects	id	INT NOT NULL PRIMARY KEY AUTO_INCREMENT	Clé primaire de la table
	name	VARCHAR(50) NOT NULL	Nom du projet
	description	TEXT	Description du projet
	fileLogo	DATE NOT NULL	Fichier contenant le logo du projet
	creator	INT NOT NULL FOREIGN KEY	Clé étrangère vers le créateur du projet (table Users)
	lastName	VARCHAR(50)	Nom de famille de l'utilisateur
	filePhoto	VARCHAR(30)	Nom du fichier de la photo de l'utilisateur
ProjectsUsersMembers	user	INT NOT NULL FOREIGN KEY	Clé étrangère vers l'utilisateur concerné (table Users)
	project	INT NOT NULL FOREIGN KEY	Clé étrangère vers le projet pour lequel l'utilisateur est membre (table Projects)
	isAdmin	BOOLEAN NOT NULL	Booléen qui indique si l'utilisateur est administrateur du projet ou non
	Remarque : Il est à noter que le couple (user, project) est marqué comme UNIQUE car il ne peut pas y avoir le même membre plusieurs fois dans le même projet)		
ProjectsUsersSpecializations	user	INT NOT NULL FOREIGN KEY	Clé étrangère vers l'utilisateur concerné (table Users)
	project	INT NOT NULL FOREIGN KEY	Clé étrangère vers le projet dans lequel l'utilisateur a une spécialisation (table Project)
	specialization	VARCHAR(50) NOT NULL	Spécialisation que cet utilisateur a dans ce projet
	Remarque : Il est à noter que le triplet (user, project, specialization) est marqué comme UNIQUE pour éviter les doublons		

Tâches

Ici sont expliquées les tables concernant les tâches :

- Tasks
- UsersTasksAffectations
- UsersTasksProductions



Dessin 2: schéma relationnel - Tâches

Table	Champs	Type	Explication
Tasks	Id	INT NOT NULL PRIMARY KEY AUTO_INCREMENT	Clé primaire de la table
	name	VARCHAR(50) NOT NULL	Nom de la tâche
	description	TINYTEXT	Description de la tâche
	deadLineDate	DATE	Date d'échéance de la tâche
	durationsInHours	FLOAT NOT NULL	Durée de la tâche en heures

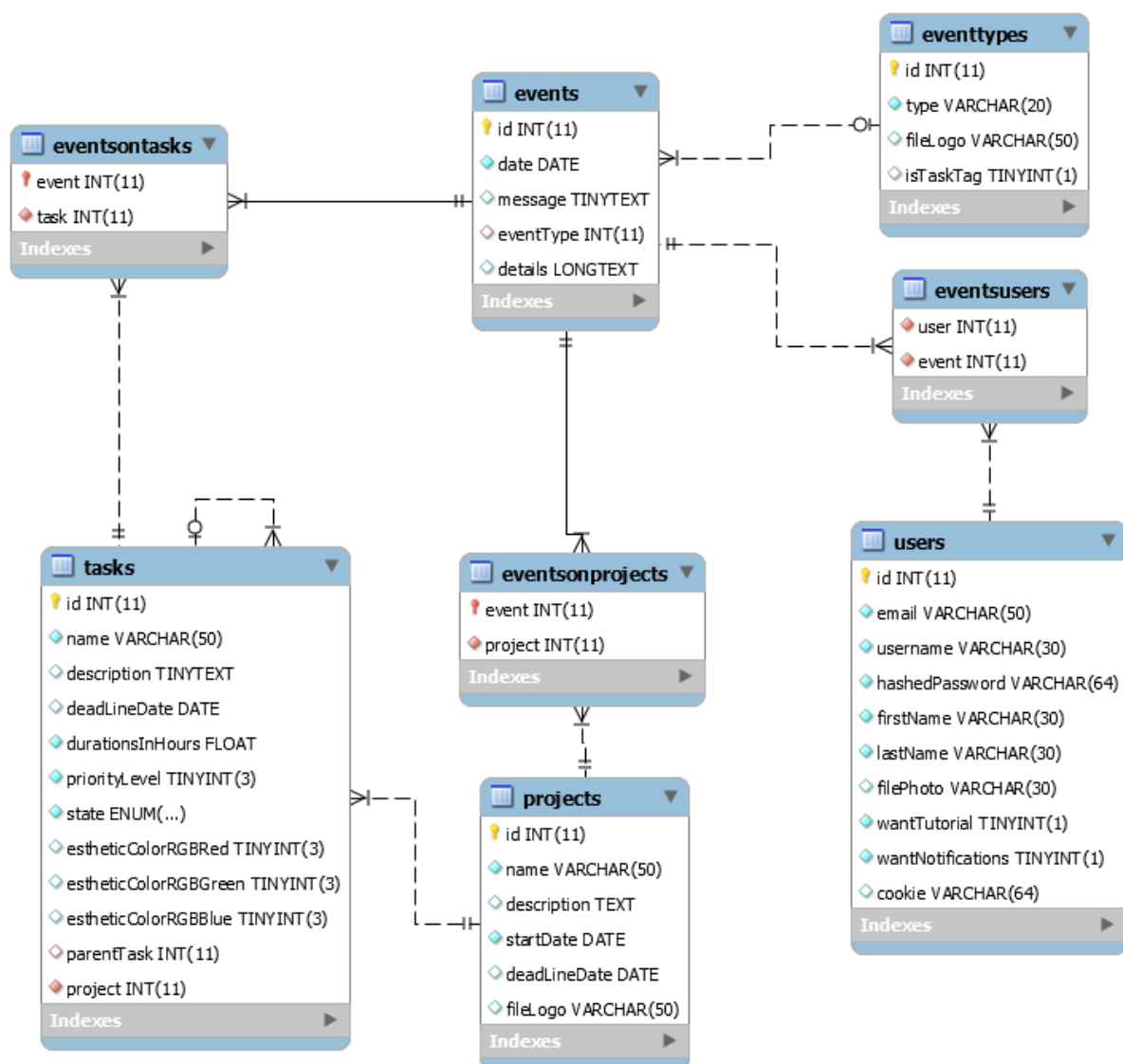
	priorityLevel	TINYINT UNSIGNED NOT NULL DEFAULT 0	Niveau de priorité de la tâche
	state	ENUM('TODO', 'DOING', 'DONE') NOT NULL DEFAULT 'TODO'	Etat de la tâche (A faire, En cours, ou Terminée)
ProjectsUsersMembers	estheticColorRGBRed	TINYINT UNSIGNED DEFAULT 255	Couleur RGB (rouge) de la tâche
	estheticColorRGBGreen	TINYINT UNSIGNED DEFAULT 255	Couleur RGB (vert) de la tâche
	estheticColorRGBBlue	TINYINT UNSIGNED DEFAULT 255	Couleur RGB (bleu) de la tâche
	parentTask	INT FOREIGN KEY	Clé étrangère vers la tâche parente (table Tasks)
	project	INT NOT NULL FOREIGN KEY	Clé étrangère vers le projet auquel la tâche est associée (table Project)
UsersTasksAffectations	user	INT NOT NULL FOREIGN KEY	Clé étrangère vers l'utilisateur concerné (table Users)
	task	INT NOT NULL FOREIGN KEY	Clé étrangère vers la tâche auquel l'utilisateur est affecté (table Tasks)
	Remarque : Il est à noter que le couple (user, task) est marqué comme UNIQUE pour éviter les doublons		
UsersTasksProductions	user	INT NOT NULL FOREIGN KEY	Clé étrangère vers l'utilisateur concerné (table Users)
	task	INT NOT NULL FOREIGN KEY	Clé étrangère vers la tâche auquel l'utilisateur est affecté (table Tasks)
	effectiveDurationInHours	FLOAT NOT NULL	Temps que l'utilisateur a déjà passé sur cette tâche

Remarque : Il est à noter qu'il peut y avoir plusieurs tuples dans cette table concernant le même utilisateur pour la même tâche. A ce moment-là, la somme de la durée effective donnera le temps total que l'utilisateur a travaillé sur cette tâche.

Événements

Ici sont expliquées les tables concernant les événements :

- Events
- EventTypes
- EventsOnProjects
- EventsOnTasks
- EventsUsers



Dessin 3: schéma relationnel - Événements

Table	Champs	Type	Explication
Events	id	INT NOT NULL PRIMARY KEY AUTO_INCREMENT	Clé primaire de la table
	date	DATE NOT NULL	Date de l'événement
	message	TINYTEXT	Message de l'événement
	eventType	INT FOREIGN KEY	Clé étrangère vers le type de l'événement (table EventTypes)
	details	LONGTEXT	Détails de l'événement
EventTypes	id	INT NOT NULL PRIMARY KEY AUTO_INCREMENT	Clé primaire de la table
	type	VARCHAR(20) NOT NULL UNIQUE	Nom du type d'événement
	fileLogo	VARCHAR(50)	Nom du fichier qui contient le logo de ce type d'événement
	isTaskTag	BOOLEAN DEFAULT 0	Indique si c'est un type d'événement Tâche ou pas
EventsOnProjects	event	INT NOT NULL PRIMARY KEY FOREIGN KEY	Clé primaire et aussi étrangère de la table (table Events) pour la relation d'héritage
	project	INT NOT NULL FOREIGN KEY	Clé étrangère vers le projet concerné par cet événement (table Projects)
EventsOnProjects	event	INT NOT NULL PRIMARY KEY FOREIGN KEY	Clé primaire et aussi étrangère de la table (table Events) pour la relation d'héritage
	project	INT NOT NULL FOREIGN KEY	Clé étrangère vers la tâche concernée par cet événement (table Tasks)
EventsUsers	event	INT NOT NULL FOREIGN KEY	Clé étrangère vers l'événement concerné (table Events)
	user	INT NOT NULL FOREIGN KEY	Clé étrangère vers l'utilisateur concerné par cet événement

Vues

Voici les différentes vues de notre base de données utilisées dans l'application.

- **view_projects_min**

- Cette vue permet d'avoir une vue minimale sur les projets

Exemple :

	id	name	fileLogo	userId	isAdmin	creator
▶	1	Travail de Bachelor	default.png	2	1	2
	1	Travail de Bachelor	default.png	3	0	2
	1	Travail de Bachelor	default.png	4	0	2
	1	Travail de Bachelor	default.png	5	1	2
	2	TWEB Liehti Moustache Project	default.png	4	1	4
	2	TWEB Liehti Moustache Project	default.png	5	0	4
	2	TWEB Liehti Moustache Project	default.png	6	0	4

Illustration 4: exemple de vue - view_projects_min

- **view_projects_members_specializations**

- Cette vue permet d'avoir pour chaque projet chaque membre avec leur spécialisations dans le projet (elle réalise un LEFT JOIN du côté de la table des utilisateurs donc les utilisateurs qui n'ont pas de spécialisation dans un projet s'afficheront aussi, car on aimerait aussi afficher les utilisateur qui n'ont pas de spécialisation dans un projet)

Exemple :

project	username	specialization	isAdmin
1	edri	Programmation C++	1
2	edri	Base de données	0
2	edri	Styles CSS	0
1	karimghozlani	Java 8	0
1	manamiz	NULL	0
2	manamiz	Node JS	1
1	raphaelracine	Base de données	1
1	raphaelracine	Programmation répartie	1
2	vanessameguep	Internet Explorer	0

Illustration 5: exemple de vue - view_projects_members_specializations

On voit ici que manamiz n'a pas de spécialisation dans le projet qui possède l'id 1, d'où la valeur NULL (dû au LEFT JOIN) mais le résultat est celui désiré.

- **view_users_projects**

- Cette vue permet d'afficher tous les utilisateurs d'un projet (avec toutes les colonnes de la table users et les id des projets auxquels ils appartiennent, ainsi qu'un booléen qui dit si l'utilisateur appartient au projet ou pas.

Exemple :

	username	hashedPassword	firstName	lastName	filePhoto	wantTutorial	wantNotifications	cookie	user	project	isAdmin
~vd.ch	raphaelracine	e35e61fb41f672d781d24d3f5c793b754ee88b4...	Raphael	Racine	default.png	1	1	NULL	2	1	1
~vd.ch	karimghozani	e35e61fb41f672d781d24d3f5c793b754ee88b4...	Karim	Ghozani	default.png	0	1	NULL	3	1	0
sig-vd.ch	manamiz	d74ff0ee8da3b9806b18c877dbf29bbde50b5bd...	Thibault	Duchoud	default.png	0	0	NULL	4	1	0
sig-vd.ch	manamiz	d74ff0ee8da3b9806b18c877dbf29bbde50b5bd...	Thibault	Duchoud	default.png	0	0	NULL	4	2	1
neig-vd.ch	edri	d74ff0ee8da3b9806b18c877dbf29bbde50b5bd...	Miguel	Santamaria	default.png	0	0	NULL	5	1	1
neig-vd.ch	edri	d74ff0ee8da3b9806b18c877dbf29bbde50b5bd...	Miguel	Santamaria	default.png	0	0	NULL	5	2	0
sig-vd.ch	vanessameguez	e35e61fb41f672d781d24d3f5c793b754ee88b4...	Vanessa	Meguez	default.png	1	1	NULL	6	2	0

Illustration 6: exemple de vue - view_users_projects

Remarque : Certaines colonnes ne sont pas affichées ici

- **view_tasks_users**

- Cette vue permet d'afficher toutes les tâches auquel un utilisateur est affecté (elle affiche les informations de l'utilisateur, et l'id des tâches).

Exemple :

	firstName	lastName	filePhoto	wantTutorial	wantNotifications	cookie	user	task
1781d24d3f5c793b754ee88b4...	Karim	Ghozani	default.png	0	1	NULL	3	1
806b18c877dbf29bbde50b5bd...	Thibault	Duchoud	default.png	0	0	NULL	4	2

Illustration 7: exemple de vue - view_tasks_users

Remarque : Certaines colonnes ne sont pas affichées ici

- **view_users_tasks**

- Cette vue permet d'afficher toutes les tâches auquel un utilisateur est affecté (elle affiche les informations de la tâche, et l'id des utilisateurs).

Exemple :

description	deadLineDate	durationsInHours	priorityLevel	state	estheticC	estheti	esthet	parentTask	project	user	ta
Aller acheter des saucisses	2016-01-04	1	1	DONE	255	255	0	NULL	1	3	1
Créer la base de données	2016-01-16	2	2	DOING	255	255	0	NULL	1	4	2

Illustration 8: exemple de vue - view_users_tasks

Remarque : Certaines colonnes ne sont pas affichées ici

- **view_projects_details**

- Cette vue permet d'afficher les détails d'un projet.

Exemple :

projectId	name	description	startDate	deadLineDate	userId
1	Travail de Bachelor	Un projet difficile... Mais intéressant !	2015-01-26	2016-10-04	2
1	Travail de Bachelor	Un projet difficile... Mais intéressant !	2015-01-26	2016-10-04	3
1	Travail de Bachelor	Un projet difficile... Mais intéressant !	2015-01-26	2016-10-04	4
1	Travail de Bachelor	Un projet difficile... Mais intéressant !	2015-01-26	2016-10-04	5
2	TWEB Liechti Moustache Project	Description is too long and unuseful...	2015-03-06	NULL	4
2	TWEB Liechti Moustache Project	Description is too long and unuseful...	2015-03-06	NULL	5
2	TWEB Liechti Moustache Project	Description is too long and unuseful...	2015-03-06	NULL	6

Illustration 9: exemple de vue - view_projects_details

- **view_events**

- Cette vue permet d'afficher les événements qui sont arrivés sur une tâche ou un projet.

Exemple :

	type	fileLogo	id	date	message	details	username	linkedEntityId	isTaskEvent
►	Info	info.svg	30	2016-01-02	"raphaelracine" moved the task from "(manamiz..."	NULL	SYSTEM	1	1
	Tasks	task.svg	29	2016-01-02	<u>raphaelracine</u> moved task <font color...	NULL	raphaelracine	1	0
	Info	info.svg	28	2016-01-02	"raphaelracine" moved the task from "(manamiz..."	NULL	SYSTEM	1	1
	Tasks	task.svg	27	2016-01-02	<u>raphaelracine</u> moved task <font color...	NULL	raphaelracine	1	0
	Info	info.svg	26	2016-01-02	"raphaelracine" moved the task from "(manamiz..."	NULL	SYSTEM	2	1
	Tasks	task.svg	25	2016-01-02	<u>raphaelracine</u> moved task <font color...	NULL	raphaelracine	1	0
	Info	info.svg	24	2016-01-02	"raphaelracine" moved the task from "(manamiz..."	NULL	SYSTEM	2	1
	Tasks	task.svg	23	2016-01-02	<u>raphaelracine</u> moved task <font color...	NULL	raphaelracine	1	0
	Info	info.svg	22	2016-01-02	"raphaelracine" moved the task from "(manamiz..."	NULL	SYSTEM	2	1
	Tasks	task.svg	21	2016-01-02	<u>raphaelracine</u> moved task <font color...	NULL	raphaelracine	1	0
	Info	info.svg	20	2016-01-02	"raphaelracine" moved the task from "(karimgho..."	NULL	SYSTEM	2	1
	Tasks	task.svg	19	2016-01-02	<u>raphaelracine</u> moved task <font color...	NULL	raphaelracine	1	0
	Info	info.svg	18	2016-01-02	"raphaelracine" moved the task from "(raphaelr..."	NULL	SYSTEM	2	1
	Tasks	task.svg	17	2016-01-02	<u>raphaelracine</u> moved task <font color...	NULL	raphaelracine	1	0
	Info	info.svg	16	2016-01-02	"raphaelracine" created the task.	NULL	SYSTEM	2	1
	Tasks	task.svg	15	2016-01-02	<u>raphaelracine</u> created task <font colo...	NULL	raphaelracine	1	0

Illustration 10: exemple de vue - view_events

Fonctions et triggers

Voici fonctions et triggers appartenant à la base de données.

Fonctions

Voici les différentes fonctions de la base de données.

Fonction	Paramètres	Explication
taskHasParent	task INT □ Tâche dont on veut tester si elle a une tâche parente	Permet de tester si la tâche passée en paramètre possède une tâche parente. Retourne TRUE si la tâche possède une tâche parente, FALSE sinon.
checkLogin	username VARCHAR(30) □ Pseudo de l'utilisateur hashedPassword VARCHAR(64) □ Mot de passe hashé (algorithme SHA-256)	Permet de savoir si un pseudo d'utilisateur correspond à un mot de passe donné (hash). Retourne TRUE si ça correspond, FALSE sinon.
checkUserCanProduceInTask	task INT □ Tâche concernée user INT □ Utilisateur concerné	Permet de savoir si un utilisateur peut saisir des heures de réalisation pour une tâche. Retourne TRUE si c'est OK, FALSE sinon.
checkUserCanBeAffectedTo Task	task INT □ Tâche concernée user INT □ Utilisateur concerné	Permet de savoir si un utilisateur peut être affecté à une tâche ou non. Retourne TRUE si c'est OK, FALSE sinon.

Triggers

Voici les différents triggers (déclencheurs) de la base de données

Trigger	Explication
usersTasksAffectationsBeforeInsert	Ce trigger vérifie avant l'insertion dans la table UsersTasksAffectations que l'utilisateur fait partie du projet dans lequel la tâche se trouve.
usersTasksAffectationsBeforeUpdate	Ce trigger vérifie avant la modification dans la table UsersTasksAffectations que l'utilisateur fait partie du projet dans lequel la tâche se trouve.
usersTasksProductionsBeforeInsert	Ce trigger vérifie avant l'insertion dans la table UsersTasksProductions que l'utilisateur est affecté à la tâche concernée.
usersTasksProductionsBeforeUpdate	Ce trigger vérifie avant la modification dans la table UsersTasksProductions que l'utilisateur est affecté à la tâche concernée.
tasksBeforeInsert	Ce trigger vérifie avant l'insertion dans la table Tasks que la tâche parente n'ait pas aussi une tâche parente (on s'arrête à un seul niveau de sous-tâche)
tasksBeforeUpdate	Ce trigger vérifie avant la modification dans la table Tasks que la tâche parente n'ait pas aussi une tâche parente (on s'arrête à un seul niveau de sous-tâche)

Implémentation des différentes contraintes d'intégrité

Voici de quelle manière ont été implémentées les contraintes d'intégrité au niveau de la base de données :

- Membre projet : Un utilisateur ne peut pas être 2 fois membre d'un même projet, et il doit y avoir au moins un administrateur par projet.
 - Ceci est implémenté avec une contrainte UNIQUE (voir table UsersProjectsMembers). La deuxième partie de la contrainte (« au moins un administrateur par projet ») n'a pas été implémentée.
- Affectation de tâche : Un utilisateur ne peut pas être affecté à une tâche s'il n'est pas membre du projet dans lequel la tâche est inscrite.
 - Ceci est implémenté dans le trigger UsersTasksAffectationsBeforeInsert et UsersTasksAffectationsBeforeUpdate.
- Réalisation de tâche : Un utilisateur ne peut pas réaliser une tâche s'il n'y est pas affecté
 - Ceci est implémenté dans le trigger UsersTasksProductionsBeforeInsert et UsersTasksProductionsBeforeUpdate.
- Sous tâche : Une tâche qui a déjà une tâche parente ne peut pas avoir de sous-tâche. Autrement dit, on s'arrête à un seul niveau de sous-tâche.
 - Ceci est implémenté dans le trigger tasksBeforeInsert et tasksBeforeUpdate.

Le framework Zend

Pour rappel, notre application fonctionne sur une implémentation de Zend Framework 2.

Les chapitres qui suivent détaillent le but et le fonctionnement de ce framework PHP, afin de mieux le comprendre. A noter que cela concerne la version 2 du framework, qui est la plus récente.

Motivations

Zend – comme de nombreux framework existants – offre de nombreux avantages, parmi lesquels :

- la possibilité d'avoir une application MVC/MOVE (Modèle-Opérations-Vues-Événements) correctement structurée, intuitive et stable ;
- la sécurité (Zend offre de nombreuses méthodes pour empêcher les injections SQL et XSS, le vol de session / hijacking, etc.) ;
- l'augmentation de la productivité une fois le framework maîtrisé, grâce aux fonctionnalités et aux modèles de conception offerts) ;
- l'amélioration de la facilité de maintenance.

Le point faible de ce framework est la prise en main, car, avouons-le, la documentation fournie sur le site officiel n'est pas des plus explicites. Cependant, le framework étant largement utilisé par la communauté, la plupart des problèmes rencontrés peuvent être résolus à l'aide des forums.

Pour avoir travaillé quelques années avec Zend, le chef de groupe le connaît relativement bien, c'est pour cela que nous avons choisi ce framework par rapport à un autre. Nous avons décidé d'utiliser MVC, car il s'agit d'un modèle simple que nous connaissions tous.

Certaines personnes jugent le langage PHP (et donc les framework y étant liés) comme étant « désuet » comparé à de nouvelles technologies plus innovantes ; cependant il s'agit d'un langage qui a fait ses preuves, qui offre de nombreuses fonctionnalités stables, et qui ressemble à ce que nous avons pu voir durant nos cours. Nous avons donc jugé qu'il était plus facile pour les membres de l'utiliser.

Fonctionnement

Introduction

Tout d'abord, Zend se définit comme étant un framework utilisant les modèles MVC ou MOVE, à choix. Nous nous occuperons ici d'analyser MVC ; pour rappel, ce modèle consiste en trois types d'entités différentes, à savoir les :

- **Modèles** : dans Zend, une entité modèle est composée de deux classes : une classe de liaison entre la base de données et l'application (*[NomTableAuSingulier].php*), et l'autre contenant des méthodes pour traiter les données (*[NomTableAuSingulier]Table.php* contenant des méthodes comme *getUser(\$id)*, etc.).

- **Contrôleurs** : ces classes nommées « *[NomContrôleur]Controller.php* » permettent de faire le lien entre les modèles et les vues, tout en effectuant des traitements conditionnels spéciaux sur les données.

Un contrôleur possède plusieurs actions, chacune d'elles pouvant être interprétée comme une page physique du site. Une action est en réalité une méthode du contrôleur qui est nommée selon la spécification « *[nomAction]Action()* » et qui possède dans la majorité des cas une vue qui lui est liée (dans la majorité, car certaines fois nous ne souhaitons pas de rendu graphique pour une action : déconnexion d'un utilisateur, actions appelées dynamiquement par de l'Ajax, etc.). Depuis ces actions, il est possible d'appeler le rendu d'une vue en lui passant des paramètres, à l'aide d'un tableau PHP.

L'architecture par défaut des URLs du site sera du type « *monsite/contrôleur/action* » ; par exemple pour l'action *addAction()* du contrôleur *ProjectsController*, l'URL sera « *monsite/projects/add* ». A noter que pour les actions par défaut, nommées *indexAction()*, le nom de l'action peut être omis (« *monsite/projects/index* » → « *monsite/projects* »).

Il est possible de configurer différemment cette architecture à l'aide des routes déclarées dans le fichier « *module/Application/config/module.config.php* » ; plus de détails sont donnés par la suite.

- **Vues** : les vues sont des fichiers de type « *.phtml* » contenant principalement du code HTML. Pour pouvoir utiliser les variables envoyées depuis le contrôleur, il faut ouvrir une balise PHP et utiliser la variable à l'aide du symbole-clé '\$' (par exemple *\$maVariable*). A noter que dans de nombreuses applications Zend (la nôtre ne faisant pas exception à la règle), le rendu graphique d'une page est composé d'un layout (« *module/Application/view/layout/layout.phtml* ») et du code HTML d'une ou plusieurs vues. Le layout est un modèle de page qui sera chargé par défaut sur toutes les pages du site, et qui contiendra un contenu dans lequel le code HTML de la vue est inséré.

Ajouter un contrôleur

Dans cet exemple, nous créerons un contrôleur appelé "**TestController**" possédant les actions "**indexAction**" et "**testAction**" (sur le site, il sera donc possible d'accéder aux pages « *monsite/test* » et « *monsite/test/test/* »).

Notre contrôleur sera donc identifié dans le framework Zend par l'appellation "**test**" ; si nous avions fait un contrôleur "**LoginController**", il aurait été identifié par l'appellation "**login**", etc.

1. Créer le fichier contrôleur dans "*module\Application\src\Application\Controller*", en s'inspirant du fichier par défaut *IndexController.php*. Attention à renommer la classe (ici, **TestController**).
2. Ajouter les actions désirées dans le contrôleur (ici, **indexAction** et **testAction**). Rappel : les URLs sont de la forme "*controller/action*".

3. Créer un dossier dans "module\Application\view\application", du nom du contrôleur (ici, **"test"**).
4. Créer une vue par action, du nom de l'action, et portant l'extention ".phtml" : phtml peut être interprété comme la version propre à Zend des fichiers HTML, un peu à la manière des fichiers .jsp dans Java EE.
Le framework Zend fera donc automatiquement le lien entre le nom et l'action et le nom du fichier, à l'aide de l'appel "new ViewModel();" dans le contrôleur.
Dans notre exemple, nous créerons deux fichiers **"index.phtml"** et **"test.phtml"**.
5. Se rendre dans le fichier "module\Application\config\module.config.php", et inscrire le contrôleur (rechercher le commentaire "// Add new controllers here." et l'ajouter dans le tableau "invokables".
Dans notre exemple, nous ajouterons la ligne :
`'Application\Controller\Test' => 'Application\Controller\TestController',`
6. Finalement, toujours dans le fichier "module\Application\config\module.config.php", ajouter une route juste en-dessous du commentaire "// Add new routes thereafter." s'il s'agit du premier contrôleur créé, ou en-dessous des routes déjà existantes sinon. S'inspirer de l'exemple ci-dessous pour créer la route :

```
'test' => array(
    'type' => 'segment',
    'options' => array(
        // Creating the route, identified by the controller's name.
        'route' => '/test[/[:action]]',
        'constraints' => array(
            // Regular expression for the action's name ; should not be modified.
            'action' => '[a-zA-Z][a-zA-Z0-9_-]*',
        ),
        'defaults' => array(
            'controller' => 'Application\Controller\Test', // Controller's name.
            // Default action ; should not be modified.
            'action' => 'index',
        ),
    ),
)
```

Pour transférer des données du contrôleur à la vue, il faut passer un tableau en paramètre du ViewModel créé, contenant les données. Si nous souhaitons par exemple passer 2 variables, il faudra écrire :

```
return new ViewModel(array(
    'var1' => $var1,
    'var2' => $var2
));
```


Il est ensuite possible de les récupérer dans la vue à l'aide de PHP, en écrivant :

```
<?php echo $this->escapeHtml($var1); ?>
```

A noter que nous aurions simplement pu mettre `<?php echo $phone; ?>`, mais la méthode `escapeHtml` permet d'éviter d'éventuelles injections html, et renforce donc la sécurité de l'application.

Finalement, il est possible d'intercepter les requêtes HTTP transitant via le contrôleur à l'aide de la méthode **onDispatch** (exemple notamment dans `ProjectController.php`), afin d'y effectuer des opérations avant de la renvoyer plus loin. Ce cas est par exemple typiquement utilisé pour contrôler l'accès à des pages qui requiert une connexion de la part de l'utilisateur.

Ajouter un modèle

Pour communiquer avec la base de données, il faut tout d'abord configurer le projet pour s'y connecter (à ne faire qu'une seule fois) :

1. Dans le fichier « `config/autoload/global.php` » : dans 'dsn', configurer le nom de la DB, ainsi que l'host.
2. Dans « `config/autoload/local.php` » : configurer le nom d'utilisateur ainsi que le mot de passe.

Pour ajouter une nouvelle table/vue/etc. au framework (nous prendrons pour cette exemple la table **"users"**, possédant les champs **"id"**, **"username"** et **"password"**) :

1. Il faut tout d'abord créer un nouveau modèle ; créer les deux nouveaux fichiers dans « `module/Application/src/Application/Model` », nommés `[NomTable].php` et `[NomTable]Table.php`, avec une majuscule pour le nom de la table, ainsi qu'une utilisation du singulier. Ces noms ne sont pas obligatoires, mais il s'agit d'une convention utilisée ; il est donc conseillé de les nommer ainsi. Ici nous aurons donc deux fichiers **User.php** et **UserTable.php**.
2. Créer une classe du nom du fichier dans `[NomTable].php` ; cette classe va servir à transformer les entités de la table en objets. Pour se faire, il suffit de déclarer un attribut dans la classe pour chaque champ de la table que nous souhaitons utiliser. Ici, nous aurons donc les attributs `"$id"`, `"$username"` et `"$password"`. Utiliser ensuite la méthode `exchangeArray($data)` pour transformer les données en objets : `"$this->id = (!empty($data['id'])) ? $data['id'] : null;"`, etc.
De nombreux exemples sont présents dans notre code.
3. Le fichier `[NomTable]Table.php` va contenir toutes les différentes méthodes qui nous seront utiles pour travailler sur la table (par exemple : *`bool checkCredentials($username, $password)`* pour contrôler la validité du nom d'utilisateur et du mot de passe donnés). Il faut déclarer la classe du même nom que le fichier ; pour ajouter une méthode, s'inspirer des fichiers existants. A noter que Zend utilise un ORM qui lui est propre ; de nombreux exemples sont présents sur le net.

4. Aller dans le fichier « module/Application/Module.php », puis inclure dans la classe les deux fichiers créés auparavant. Par exemple :

```
use Application\Model\User;
```

```
use Application\Model\UserTable;
```

5. Rechercher la fonction *getServiceConfig()* située dans « module/Application/Module.php ».
6. En s'inspirant de ce qui existe déjà, déclarer une passerelle entre l'entité de la base de données (ici la table "**users**") et la classe de transformation [NomTable] (en prenant attention à changer les valeurs déjà existantes), puis l'utiliser pour passer la passerelle à la classe [NomTable]Table.
7. La modèle est à présent prêt à être utilisé.

Pour pouvoir utiliser dans un contrôleur le modèle créé auparavant (dans cet exemple, le modèle **User** ; se baser sur ce qui est fait dans le contrôleur UserController) :

1. Dans le contrôleur, déclarer un attribut privé qui représentera le modèle (*private \$userTable*).
2. Créer une méthode privée *get[NomTable]Table*, qui va agir comme un singleton et va retourner une instance de la table depuis le modèle.

```
// Get the user's table's entity, represented by the created model.  
// Act as a singleton : we only can have one instance of the object.  
private function getUserTable()  
{  
    // If the object is not currently instantiated, we do it.  
    if (!$this->userTable) {  
        $sm = $this->getServiceLocator();  
        // Instantiate the object with the created model.  
        $this->userTable = $sm->get('Application\Model\UserTable');  
    }  
    return $this->userTable;  
}
```

3. Voilà, il est désormais possible d'utiliser le modèle à l'aide de *\$this->get[NomTable]Table()*.
Exemple :

```
$areCredentialsCorrect = $this->getUserTable()->checkCredentials("jean-louis",  
"jlaacawm39cjaéwsà23cnoiqw");
```

A noter que si un contrôleur utilise beaucoup de modèles différents, nous avons pensé à un moyen de factoriser le code, afin d'éviter de devoir déclarer manuellement chaque modèle :

```
// Get the given table's entity, represented by the created model.
private function _getTable($tableName)
{
    $sm = $this->getServiceLocator();
    // Instanciate the object with the created model.
    $table = $sm->get('Application\Model\\'.$tableName);
    return $table;
}
```

A utiliser de la manière suivante dans le code :

```
$this->_getTable('UserTable')->checkCredintentials("jean-louis",
"jlaacawm39cjaéwsà23cnoi qw");
```

Détails d'implémentation de l'application EasyGoing

Schéma UML

Voici le schéma UML de notre application :

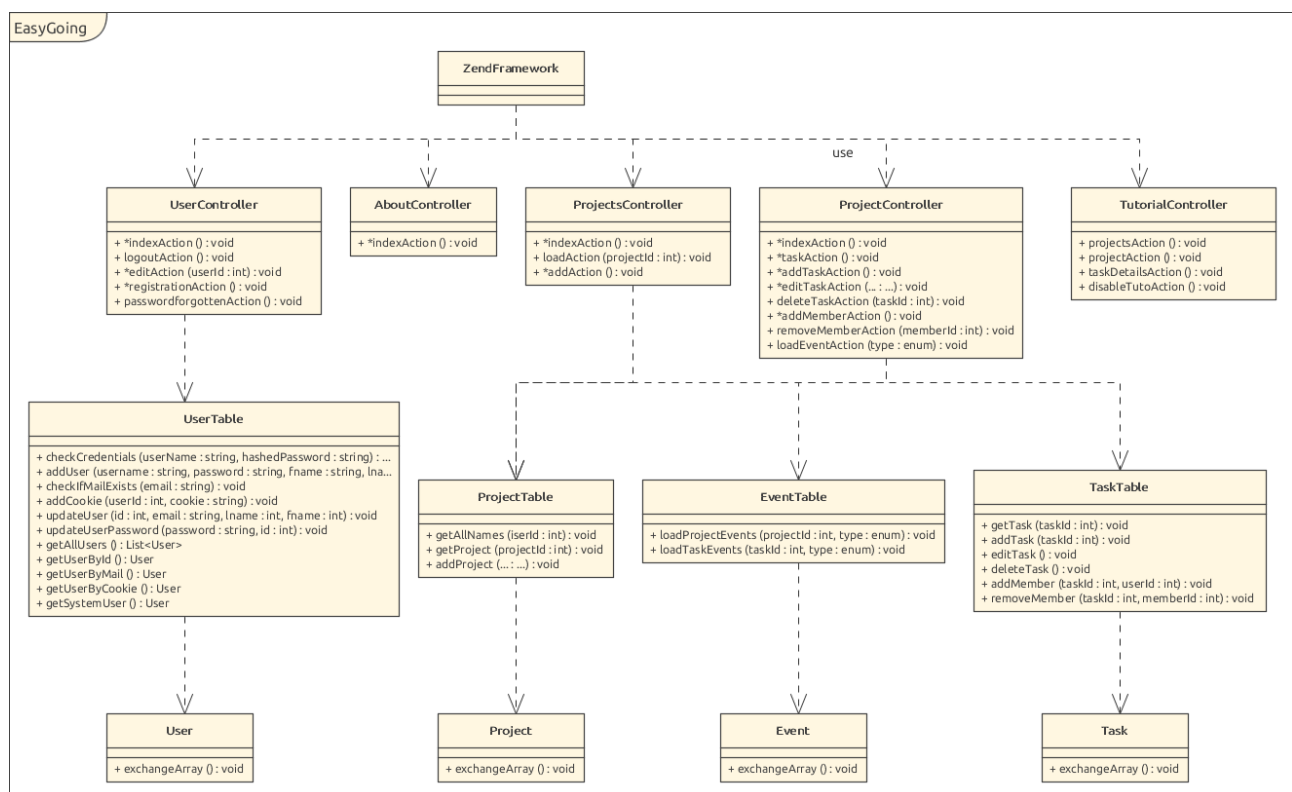


Illustration 11: schéma UML des classes de l'application.

Rôles des contrôleurs

Afin de rendre la compréhension du schéma UML plus aisée, veuillez tout d'abord lire le chapitre dédié au framework Zend, si cela n'est pas déjà fait. Suite à cela, voici quelques explications supplémentaires sur les différents contrôleurs de l'application.

UserController

Rôle

Ce contrôleur sert à gérer toutes les actions propres à l'utilisateur directement, à savoir:

- l'inscription d'un utilisateur au site
- la connexion/déconnexion d'un utilisateur sur le site
- l'édition du compte d'un utilisateur préalablement inscrit
- la possibilité d'envoyer un mail à un utilisateur ayant oublié ou perdu son mot de passe.

Usage

Il est utilisé dans plusieurs vues:

- page d'accueil (connexion)
- page d'inscription
- page d'édition
- page d'envoi d'un nouveau mot de passe
- sur toutes les pages, le bouton "Log out" situé dans l'en-tête

Page d'accueil

C'est le point d'entrée de notre application. L'utilisateur peut s'inscrire, entrer ses identifiants ou demander un mot de passe. Chacune de ses actions va le rediriger vers une page dédiée. Si la session de l'utilisateur est active ou qu'il a préalablement cliqué sur le bouton "Remember Me" lorsqu'il s'est connecté, l'utilisateur est alors redirigé sur la page "Projects". Les notions de session et de cookie sont développés dans une section à part.

Page d'inscription

Sur cette page, l'utilisateur entre ses informations personnelles. L'unicité est garantie sur le nom d'utilisateur ainsi que sur l'adresse e-mail. L'utilisateur a la possibilité de suivre ou non un tutoriel, ainsi que de recevoir éventuellement des notifications. L'enregistrement se fait uniquement si tous les champs obligatoires sont remplis et ne comportent pas d'erreur (voir page d'édition)

Page d'édition

L'utilisateur, une fois connecté, peut modifier ses informations à l'exception du nom d'utilisateur. Il peut également configurer un nouveau mot de passe. Tout comme dans la page d'inscription, des messages d'erreur apparaissent si l'utilisateur entre des données erronées comme:

- un format incorrect d'adresse e-mail
- deux mots de passe qui ne correspondent pas
- un format incorrect de la photo de profil
- l'adresse e-mail est déjà utilisée par un autre utilisateur
- (uniquement pour l'inscription) le nom d'utilisateur est déjà utilisé par un autre utilisateur

Page d'envoi d'un nouveau mot de passe

Lorsque l'utilisateur a perdu ou oublié son mot de passe, il a la possibilité de le réinitialiser. Il entre alors son adresse e-mail et un e-mail contenant un nouveau mot de passe lui est envoyé sous la réserve que l'adresse entrée est bien stockée en base de données.

Bouton « Logout »

Le bouton « Logout » est accessible dans l'en-tête du site, et donc à chaque page. Il réinitialise la session et supprime l'éventuel cookie d'inscription existant sur le navigateur de l'utilisateur.

ProjectsController

Le contrôleur « ProjectsController » est relatif à l'affichage des différents projets auxquels appartient un utilisateur, ainsi qu'à l'ajout d'un nouveau projet. Il est composé de deux actions :

1. **indexAction** : cette action s'occupe de récupérer la liste de tous les projets auxquels appartient un utilisateur, puis de les envoyer à la vue qui les affichera.
2. **addAction** : s'occupe d'ajouter un nouveau projet.
Elle réagit différemment selon si l'utilisateur a effectué une requête POST (c'est-à-dire qu'il a appuyé sur le bouton « Create! » depuis la vue) ou non :
 - dans le premier cas, elle validera les différents champs entrés par l'utilisateur et, si ceux-ci sont valides, ajoutera le projet dans la base de données puis redirigera l'utilisateur ; s'ils ne sont pas valides, elle enverra un message à la vue qui affichera une erreur ;
 - si l'utilisateur n'a pas encore réalisé de POST, c'est qu'il vient d'accéder à la page de création ; nous appelons simplement la vue qui affichera le formulaire vide.

ProjectController

Le contrôleur « ProjectController » est relatif à un projet en particulier. Il gère toutes les actions qui sont relatives à un projet. Il va, entre autres, afficher les informations relatives à un projet, les détails d'une tâche qui se trouve dans un projet, la liste des membres qui peuvent rejoindre ce projet. Il gère également l'ajout, suppression d'un membre dans le projet et l'ajout, suppression et édition d'une tâche ou sous-tâche dans le projet.

Voici les différentes actions qui le composent :

- **indexAction*** : cette action s'occupe de récupérer les différentes informations relatives au projet courant (le projet, les tâches et sous-tâches qui s'y trouvent, les membres, les événements et d'autres informations utilisées dans la vue). Elle va ensuite demander à la vue d'afficher ces différentes informations.
- **editAction*** : cette action va mettre à jour les informations d'un projet, après une édition de celui-ci. Elle réagit différemment selon si l'utilisateur a effectué une requête POST (c'est-à-dire qu'il a appuyé sur le bouton « Save changes ») ou non :
 - Dans le premier cas, elle validera les différents champs entrés par l'utilisateur et les droits de celui-ci. Si les champs sont corrects et que l'utilisateur a les droits nécessaires pour modifier les propriétés d'un projet, elle met à jour les valeurs dans la base de données et redirige l'utilisateur sur la page d'index des projets ; s'ils ne sont pas valides ou qu'il n'a pas les droits, elle renvoie une erreur.

- Dans le second cas, l'utilisateur n'a pas fait de requête POST et donc elle appelle simplement la vue qui affichera le formulaire d'édition d'un projet.
- **addTaskAction*** : cette action s'occupe d'ajouter une nouvelle tâche au projet. Cette action va, comme la précédente, faire deux choses lorsqu'elle reçoit une requête POST ou non. Dans les 2 cas on va vérifier, si la tâche qu'on veut ajouter est une sous-tâche, que celle-ci est valide.
 - Dans le cas d'une requête POST, elle vérifie les champs entrés par l'utilisateur. S'ils sont corrects on ajoute la tâche dans la base de données et, si ce n'est pas une sous-tâche, on l'affecte à son créateur. Si les champs sont incorrects elle renvoie une erreur.
 - Si ce n'est pas une requête POST, elle va afficher le formulaire permettant l'ajout d'une tâche.
- **taskDetailsAction*** : cette action va afficher les détails d'une tâche. Elle récupère les différentes informations à afficher (les informations relatives à une tâche) et les envoie à la vue qui va nous être affichées.
- **editTaskAction*** : cette action s'occupe de l'édition d'une tâche. Elle fait également deux choses lorsqu'elle reçoit une requête POST ou non.
 - Dans le cas d'une requête POST, elle vérifie les champs entrés par l'utilisateur. S'ils sont corrects elle met à jour les informations dans la base de données et redirige l'utilisateur sur la page d'index d'un projet.
 - Dans le second cas, elle affiche la vue d'édition d'un projet en passant les informations relatives au projet qui y seront affichées.
- **boardViewMembersAction** : cette action va retourner les informations relatives au « board » qui affiche les membres dans la colonne de gauche. Elle va récupérer les membres du projet (et des informations les concernant), leurs spécialisations et les tâches affectées à chaque membres. Elle va ensuite créer une vue en lui envoyant les informations récupérées et elle va enlever le « layout » de cette vue car celle-ci est chargée, du côté client, en ajax (requête « load »). La vue représente un board.
- **boardViewTasksAction** : cette action va retourner les informations relatives au « board » qui affiche les tâches dans la colonne de gauche. Elle va récupérer les tâches principales (celles qui ne sont pas des sous-tâches) du projet et les membres affectés à chaque tâche. Elle va ensuite faire la même chose que l'action « boardViewMembersAction » mais c'est une autre vue qui est créée.
- **assignTaskAction*** : cette action gère l'attribution d'une tâche à un membre d'un projet. Elle est appelée lorsqu'un utilisateur « drag-drop » une tâche non assignée pour la mettre dans le « board » (il assigne la tâche à un membre). Les tâches sont assignées en déplaçant les liens qui se trouvent en dessous du « board » dans la liste des tâches du projet. Elle va vérifier les droits de l'utilisateur et si la tâche est déjà assignée à l'utilisateur cible ou non. Dans le cas où l'utilisateur a les droits nécessaires et la tâche n'est pas déjà assignée à l'utilisateur cible, on ajoute une affectation dans la base de données. Sinon elle affiche une erreur.
- **moveTaskAction*** : cette action gère le déplacement d'une tâche dans le « board ». Lorsqu'un utilisateur déplace une tâche du « board » vers un autre utilisateur ou dans une autre colonne. Elle va également vérifier les droits de l'utilisateur et, si l'utilisateur déplace la tâche vers un autre utilisateur, que celle-ci ne lui est pas déjà assignée. Dans le cas où tout est ok, elle met à jour les informations dans la base de données. Sinon elle affiche une erreur.

- `unassignTaskAction*` : cette action gère la désaffectation d'une tâche à un utilisateur. Elle est appelée lorsqu'un utilisateur fait un clic droit sur une tâche et sélectionne « Unassign ». Elle va vérifier les droits de l'utilisateur et s'il a les droits, elle supprime l'affectation correspondante dans la base de données. Sinon elle affiche une erreur.
- `deleteTaskAction*` : cette action gère la suppression d'une tâche. Elle est appelée lorsqu'un utilisateur fait un clic droit sur une tâche et sélectionne « Delete ». Elle va vérifier les droits de l'utilisateur et si ceux-ci sont suffisants elle supprime la tâche de la base de données et les affectations correspondantes et redirige l'utilisateur à la page d'index du projet. Sinon elle affiche une erreur.
- `addMemberAction*` : cette action gère l'ajout d'un utilisateur dans un projet. Elle a deux comportements différents en fonction de si la requête reçue est de type POST ou non. Quel que soit le type de requête, elle va vérifier les droits de l'utilisateur et si celui-ci n'a pas les bons elle le redirige vers la page d'index du projet.
 - Dans le premier cas, elle va récupérer les informations postées et celles-ci correspondent au(x) utilisateur(s) à ajouter. Elle va parcourir ses informations et ajouter chaque utilisateur au projet.
 - Dans le second cas, elle affiche la vue qui permet d'ajouter des utilisateurs au projet (affiche la liste des utilisateurs qui ne sont pas membres du projet et l'utilisateur peut les sélectionner afin de les ajouter).
- `removeMemberAction*` : cette action gère le fait de retirer un membre d'un projet. Elle va vérifier les droits de l'utilisateur et s'ils sont suffisants elle va mettre à jour la base de données en supprimant l'affectation du membre au projet et les spécialisations relatives à celui-ci.
- `detailsAction` : cette action retourne, sous format JSON, les informations relatives à un projet. Elle est utilisée lorsqu'un déroule un projet dans la liste des projets.
- `postNewFeedAction*` : cette action gère l'ajout d'une nouvelle « news » correspondant à une tâche. Quand on est sur la page qui affiche le détail d'une tâche l'utilisateur a la possibilité d'ajouter une « news » sur cette tâche. Cette action est appelée lorsqu'un utilisateur ajoute une nouvelle « news » sur une tâche. Elle va récupérer les informations entrées par l'utilisateur et les ajouter dans la base de données. Elle retourne un JSON contenant un champ « success » qui est à « true » s'il n'y a pas d'erreur et à « false » sinon. Lorsque la requête reçue n'est pas de type POST elle retourne se même JSON avec le champ « success » à « false ».

* Les actions suivies d'un astérisque (*) contiennent toutes une gestion des événements. Pour chacune de ces tâches, des événements sont créés et insérés dans la base de données. Ces événements correspondent à ce qui est fait dans la tâche en question. Par exemple lorsqu'on retire un membre du projet (*removeMemberTask*) un événement de type utilisateur est créé avec la date courante et contenant le texte : « xxx removed user yyy from project, bye bye ! ». Le texte et le type d'événement change, bien évidemment, en fonction de l'action appelée. Ces actions envoient, pour chaque événement, une requête au serveur qui gère les événements afin que les autres utilisateurs présent sur le projet en soit informé en temps réel. (Pour plus d'informations c.f. le chapitre « Serveurs secondaires d'événements »).

AboutController

Ce contrôleur permet d'afficher les pages en rapport avec l'aide et la page « about » du site.

- aboutAction : décrit le cadre du projet, ainsi que les différents acteurs ; donne aussi le lien GitHub du projet.
- helpAction : nous avons créé cette section afin de présenter de manière sommaire le site et permettre aux utilisateurs de savoir à priori ce qu'ils peuvent y retrouver ; il s'agit de champs de texte qui apportent quelques éléments de réponses aux questions que pourraient se poser les utilisateurs.

TutorialController

Pour rappel, notre application offre un système de tutoriels aux nouveaux utilisateurs (ainsi qu'aux anciens qui le désireraient). Ce contrôleur permet donc de questionner le modèle qui contient les informations à afficher pour les tutoriels.

Il est à noter que ce contrôleur est utilisé dans le cadre de requêtes AJAX du côté client (voir plus loin dans le chapitre « Tutoriel ») et que ces actions retournent du contenu JSON et non une vue.

Voici les différentes actions de ce contrôleur :

- projectsAction : cette action retourne l'ensemble des informations du tutoriel concernant l'affichage de tous les projets dont un utilisateur fait partie, autrement dit quand on est sur la page avec l'url : « /projects ». Cette action fait appel au modèle Tutorial (méthode projects).
- projectAction : cette action retourne l'ensemble des informations du tutoriel concernant l'affichage d'un projet, autrement dit quand on est sur la page avec l'url : « /project/{idProjet} ». Cette action fait appel au modèle Tutorial (méthode project).
- taskDetailsAction : cette action retourne l'ensemble des informations du tutoriel concernant l'affichage des détails d'une tâche, autrement dit quand on est sur la page avec l'url : « /project/{idProjet}/taskDetails/{idTache} ». Cette action fait appel au modèle Tutorial (méthode taskDetails).
- disableTutoAction : cette action permet de désactiver le tutoriel pour l'utilisateur actuellement connecté. Elle est déclenchée lorsque celui-ci clique sur le lien « Disable Tutorial » de la fenêtre de tuto.

Architecture des fichiers du projet

Voici l'architecture des fichiers de notre projet (plus de détails sont donnés plus bas) :

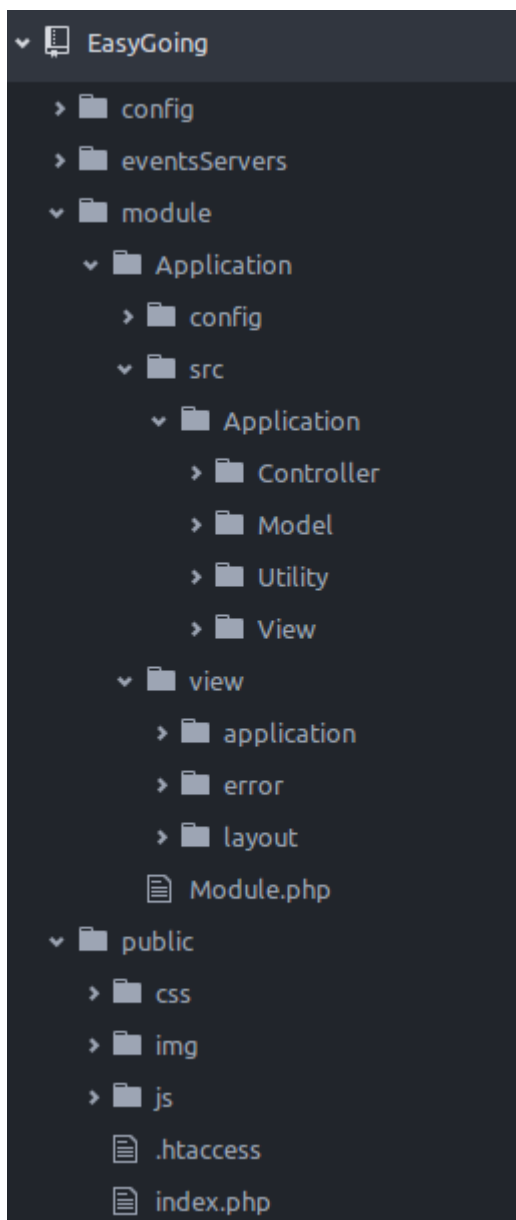


Illustration 12: architecture des fichiers de l'application.

- **config** : contient les configurations générales et globales de l'application (chargement des modules, connexion à la base de données, etc.).
- **eventsServers** : contient le code Node.js des serveurs d'événements (plus d'explications dans « **Serveurs secondaires d'événements** »).
- **module** : contient le code à proprement parlé de l'application.
 - **config** : contient la configuration des routes de l'application et des alias de contrôleurs.
 - **src/Application** : contient les modèles et contrôleurs.
 - **Controller** : contient les contrôleurs de l'application.
 - **Model** : contient les modèles de l'application.
 - **Utility** : contient des classes utilisées dans plusieurs contrôleurs pour factoriser le code (liste énumérée, création de miniatures d'images, envoi de mails, etc.), ainsi que les variables globales du projet (adresse des serveurs d'événements et URL du site).
 - **View** : contient les différentes vues HTML.
 - **Application** : contient les vues des contrôleurs.
 - **Error** : contient les vues des différentes erreurs pouvant survenir.
 - **Layout** : contient le layout des pages.
- **public** : contient les fichiers publics du serveur, comme le code CSS, les images, le code JavaScript client, le fichier « index.php » de lancement de l'application ainsi que son fichier de configuration « .htaccess ».

Interface graphique

Pour notre site, le rendu du layout se présente de la façon suivante :



Illustration 13: architecture graphique de l'application.

1. Il s'agit du contenu du layout, qui est composé des points 2, 3 et 4.
2. L'en-tête du site, qui se place tout en haut. A noter qu'il s'agit ici de sa représentation lorsque l'utilisateur est connecté. Dans le cas contraire, seul le titre apparaît.
3. Le contenu à proprement parler de la page ; il s'agit du code HTML de la vue. Cette partie possède un ascenseur de scrolling si elle est trop grande pour la page ; lorsque l'utilisateur scroll la page vers le bas, le header disparaît.
4. Un pied-de-page toujours présent en bas de page.

A noter que le rendu de la page d'accueil est légèrement différent, car la barre d'en-tête n'y apparaît pas, étant inutile.

Inscription

Pour pouvoir effectuer n'importe quelle activité dans le site, chaque utilisateur doit posséder un compte. La création de compte consiste à remplir les champs d'un formulaire en indiquant obligatoirement les champs marqués comme obligatoires à l'aide d'une astérisque. Aussi, chaque utilisateur doit posséder une adresse e-mail correcte et unique pour pouvoir créer un compte. Tous les membres des projets doivent posséder une photo de profil ; nous avons donc prévu une image par défaut pour les utilisateurs qui n'en fourniraient pas. Les images sont validées uniquement si elles possèdent les dimensions et les extensions de fichier supportées et indiquées sur le site à côté des champs correspondants. Un utilisateur doit également spécifier s'il veut ou non que des tutoriels lui soient proposés tout au long de sa navigation sur le site afin de faciliter son utilisation, et s'il veut recevoir les notifications e-mail. Les notifications lui seront envoyées lorsqu'il aura été ajouté à un projet, ou lorsqu'un administrateur l'en aura retiré ; à noter qu'il recevra dans tous les cas une notification lors de la modification de son compte, pour des raisons de sécurité.

Lorsqu'un utilisateur entre une donnée incorrecte, un message d'erreur s'affiche au dessus du formulaire.

Pour sécuriser les données des utilisateurs, les mots de passe des utilisateurs sont hashés avant d'être stockés dans la base de donnée.

Session

La notion de session évoquée dans la page d'accueil et dans le bouton « Logout » sont définies par PHP, ainsi que par une surcouche du framework Zend.

Essentiellement, la session s'appuie sur un cookie nommé PHPSESSID contenant un ID de session et étant stocké sur les navigateurs des clients. Le cookie est alors utilisé par le serveur pour "reconnaître" l'utilisateur lors de sa navigation sur EasyGoing. En effet, lors de la connexion de l'utilisateur au site, sa session devient active et se désactivera dans les 5 minutes qui suivent la fermeture de la page EasyGoing (configuré dans le fichier « /module/Application/Module.php »).

L'activation de la session a pour effet d'octroyer à l'utilisateur toutes les pages qui le concernent, à savoir:

- ses projets et toutes les pages qui en découlent ;
- l'édition de son compte.

Lorsque la session n'est plus active et que l'utilisateur veut accéder à ses pages personnelles, il est redirigé sur la page d'accueil où il doit alors entrer à nouveau ses identifiants. La session est désactivée soit par l'expiration du temps après que l'utilisateur ait fermé sa page EasyGoing, soit lorsqu'il clique sur le bouton « Logout ».

Les pages "About" et "Help" sont accessibles en tout temps et ne requièrent donc pas l'activation d'une session.

Cookies

En plus du cookie utilisé par la session PHP, deux autres cookies sont utilisés.

Cookie d'inscription

Lors de la connexion au site, l'utilisateur a la possibilité de cocher la case "Remember Me". Lorsqu'il le fait, un cookie est stocké sur la machine de l'utilisateur. La présence de ce cookie active la session et confère donc le droit à la personne en possession de ce cookie d'accéder à toutes les informations relatives à l'utilisateur. Un cookie est donc lié à un utilisateur.

Ce cookie constitue alors une ressource sensible et requiert de ce fait une gestion sécuritaire. Sa valeur est une combinaison du nom d'utilisateur, de son mot de passe, et d'un sel généré aléatoirement. Le tout est encrypté à en SHA-256 puis stocké en base de données dans un champ dédié. De cette façon, il est impossible (ou très fortement improbable) qu'un utilisateur malveillant injecte un cookie contenant la bonne combinaison.

Tout comme pour la session, le clic sur le bouton « Logout » a pour effet de retirer le cookie. La durée de validité de ce cookie a été configurée à 30 jours. Passé ce délai, l'utilisateur devra se reconnecter, générant ainsi un nouveau cookie.

Cookie de spécialisation d'un membre

Dans la page contenant le dashboard d'un projet, il est possible d'afficher ou non les spécialisations des utilisateurs à l'aide d'un bouton poussoir. Le choix fait par l'utilisateur est mémorisé à l'aide d'un cookie, afin que le bouton soit activé ou non par défaut dans le futur. Plus de détails sont donnés plus bas.

La durée de ce cookie est aussi de 30 jours.

Tutoriel

Réalisation

Voici comment nous avons réalisé la partie des tutoriels qui s'affichent sur les pages principales de l'application, sous forme de petites fenêtres comme celle-ci (sous forme de tooltips) :

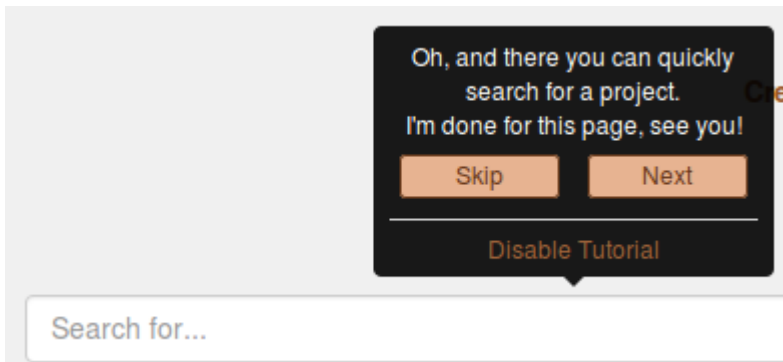


Illustration 14: rendu graphique du tutoriel.

Le but est de pouvoir à l'aide d'une balise de la forme suivante pouvoir charger un tooltip (id étant l'identifiant du tutoriel à afficher (après chargement AJAX à voir plus loin)) :

```
<div id="createProject" role="tutorial" />
```

Partie serveur

Le serveur se doit de fournir les informations de tutoriel à afficher sur les pages. Pour cela, il implémente un contrôleur et un modèle pour respecter le modèle MVC du Framework Zend.

Modèle « Tutorial »

Ce modèle contient des données statiques sur les différents tutoriels autrement dit les textes à afficher dans les différents tooltip.

Voici les différentes méthodes qui se trouvent dans ce modèle :

- `_generateData` (méthode privée)
 - Cette méthode permet de construire un tableau qui représente un couple (div, text) autrement dit la valeur de div sera l'identifiant du div qui sera affiché du côté client (voir plus loin), et le text sera bien entendu celui qui sera affiché dans le tooltip à l'endroit où le div en question aura été placé sur la page (voir plus loin). Cette méthode est privée car elle permet de factoriser du code.
- `projects`
 - Cette méthode retourne l'ensemble des informations concernant le tutoriel sur les projets d'un utilisateur sous forme d'un tableau de tableaux. Chaque élément du tableau parent sera un couple (div, text). Autrement dit, on va retourner un ensemble de couples (div, text).
- `project`
 - Idem que `projects()` mais pour l'ensemble des informations concernant le tutoriel sur un projet.
- `taskDetails`
 - Idem mais pour l'ensemble des informations concernant le tutoriel sur les détails d'une tâche.

Partie client

Voici ce qui se passe du côté client pour l'implémentation de ce tutoriel sous forme de tooltips.

Javascript

Tout d'abord, le layout charge le script nommé **tuto.js** qui se trouve dans public/js/. Le layout charge ce fichier tuto.js uniquement si un utilisateur est connecté et qu'il désire avoir le tutoriel, de la manière suivante :

```
// If user want tutorials
if($sessionUser->wantTutorial)
    echo $this->headScript()->prependFile($this->basePath() . '/js/tuto.js');
```

Voici des explications sur ce fichier tuto.js.

- Variable globale « current »
 - Cette variable est l'index du tableau tutoData dont l'élément contient les informations du tutoriel qui est en train d'être affiché.
- Variable globale « tutoData »
 - Cette variable contient un tableau de couples (div, text) donc l'ensemble des données qui ont été chargées depuis la fonction loadTutorial.
- loadTutorial(tuto)
 - Cette fonction permet de charger des données d'un tutoriel dans la variable globale tutoData. Pour se faire, elle fait une requête AJAX sur l'url « /tutorial/tuto. Pour chaque couple (div, text) dans le résultat JSON de la requête, on l'ajoute dans la variable globale tutoData.
- nextTutorial()
 - Permet de charger le prochain tutorial sur la page (en commençant par current = 0). Une fois qu'elle a fait le tour des tutoriels qui ont été chargés, elle appelle la fonction skipTutorial afin de cacher tous les tooltips, sinon elle charge le prochaine tutoriel de la manière suivante :
 - Afficher avec jQuery tous les div dont l'id = tutoData[current].div et dont le role=tutorial en leur affectant du code HTML content les appels aux fonctions nextTutorial et skipTutorial correspondant aux balises <a> pour Next et Skip
 - Cacher avec jQuery tous les div dont l'id != tutoData[current].div et dont le role= tutorial

Ainsi, on affichera uniquement le tooltip correspondant au couple(div, text) actuel.
- skipTutorial()
 - Cache tous les div concernant le tutoriel (dont le role=tutorial)
- disableTutorial()
 - Permet de désactiver les tutoriels pour l'utilisateur courant, à l'aide d'une requête Ajax effectuée sur l'action *disableTuto* du contrôleur *TutorialController*.

Utilisation dans une page

Pour pouvoir utiliser les données du tutoriel dans une page, il faut tout d'abord dans la page voulu écrire le code javascript suivant (après avoir chargé tuto.js).

```
<script type="text/javascript">
    try {
        loadTutorial("projects");
    }
    catch(err) {
        // Catch to avoid undefined loadTutorial error if user doesn't
        want tutorial
    }
</script>
```

Le bloc try-catch permet d'éviter d'avoir des erreurs dans la console dans le cas où l'utilisateur ne désire pas de tutoriel (car le fichier tuto.js ne serait pas chargé) et donc la fonction loadTutorial() non définie.

Ensuite, on demande à charger le tutoriel que l'on désire (ici « projects », il est possible aussi de mettre « project » ou « taskDetails » à condition que le contrôleur TutorialController fournit une url correspondante).

Ensuite pour l'emplacement du tooltip il suffit d'écrire le code html suivant :

```
<div id="nomTuto" role="tutorial" />
```

Tooltip

Pour l'affichage d'un tooltip, nous nous sommes appuyé sur des fichiers Bootstrap CSS et jQuery (voir documentation sur : <http://getbootstrap.com/javascript/#tooltips>).

Droits dans l'application

Sur le board (et donc dans le *ProjectController*), tout le monde ne peut pas tout faire. Certaines actions requièrent certains droits. Il existe 3 types de membres à un projet :

5. Le **créateur** ou le **super administrateur**. C'est celui qui a créé le projet et qui est, de fait, super administrateur du projet. Cet utilisateur à tous les droits sur un projet
6. Le **manager** ou **administrateur**. Cet utilisateur obtient ce rôle lorsqu'il est ajouté comme membre d'un projet. Il a plus de droits qu'un membre lambda mais moins que le créateur.
7. Un membre **lambda**. C'est un utilisateur ajouté au projet sans qu'il soit manager.

Donc par ordre de grandeur il y a **Créateur** > **Manager** > **Lambda**.

Les différentes actions sur les tâches et sous-tâches ne peuvent donc pas être faite par tous.

Voici une liste des droits par action (si l'action n'y apparaît c'est qu'il n'y a pas de gestion des droits, tout le monde peut y accéder) :

- **Affichage d'un projet (indexAction)** : Tous les membres d'un projet peuvent accéder au détail de ce projet et donc au board qui lui est associé.
- **Edition d'un projet (editTaskAction)** : Pour pouvoir éditer un projet, le membre doit au moins être un Manager.
- **Création d'une tâche (addTaskAction)** : Tous les membres d'un projet peuvent créer des tâches.
- **Affichage du détail d'une tâche (taskDetailsAction)** : Tous les membres d'un projet peuvent accéder aux détails d'une tâche et à son fil d'actualité.
- **Edition d'une tâche (editTaskAction)** : Tous les membres d'un projet peuvent modifier une tâche.

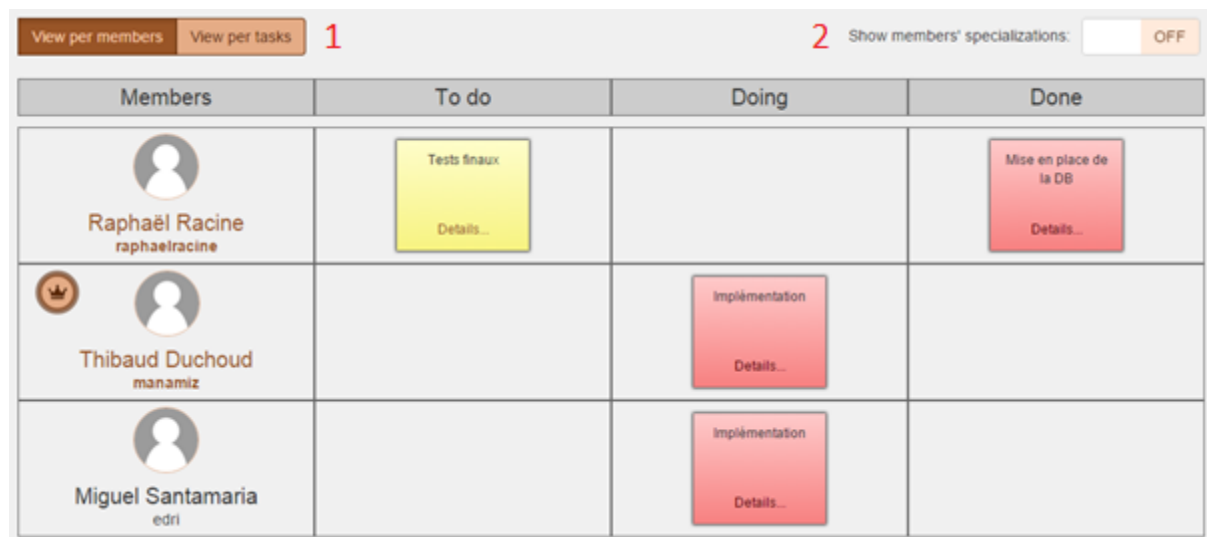
- **Assigner une tâche (assignTaskAction)** : Tous les membres d'un projet peuvent assigner les tâches à n'importe quel autre membre du projet.
- **Déplacer une tâche (moveTaskAction)** : Le créateur peut déplacer les tâches de tout le monde vers tout le monde. Le manager peut déplacer les tâches qui lui sont assignées vers n'importe qui. Il peut également déplacer les tâches des membres lambda vers n'importe qui mais il ne peut pas déplacer les tâches assignées aux autres managers ou au créateur.
- **Désassigner une tâche (unassignTaskAction)** : Le créateur peut désassigner une tâche qui est assignée à n'importe qui (y compris lui-même). Le manager peut désassigner les tâches qui sont assignées aux utilisateurs lambda ou à lui-même. Un utilisateur lambda ne désassigne aucune tâche (même une qui lui est associée).
- **Supprimer une tâche (deleteTaskAction)** : Seul le créateur peut supprimer une tâche.
- **Ajouter un utilisateur au projet (addMemberAction)** : Seul le créateur et les managers d'un projet peuvent ajouter des membres au projet et choisir si l'utilisateur ajouté est manager ou non et quels sont ses droits.
- **Retirer un utilisateur du projet (removeMemberAction)** : Le créateur peut retirer n'importe quel membre du projet (sauf lui-même). Un manager peut retirer les autres utilisateurs qui ne sont pas des managers. Les utilisateurs lambda ne peuvent rien faire.

Fonctionnement du « Dashboard »

Généralités

Le « Dashboard » est d'une grande importance dans notre projet. C'est à cet endroit que les membres d'un projet peuvent voir les différentes tâches qui existent et voir si celles-ci sont assignées à un ou plusieurs autres membres.

Ce « Dashboard » se présente sous la forme d'un « Scrum board ».



The screenshot shows a Scrum board interface. At the top, there are two tabs: 'View per members' (selected) and 'View per tasks'. To the right of the tabs is a red number '1'. Further right is a toggle switch labeled '2 Show members' specializations:' with the switch set to 'OFF'. The main area is a table with four columns: 'Members', 'To do', 'Doing', and 'Done'. The 'Members' column lists three members: Raphaël Racine (with a crown icon), Thibaud Duchoud, and Miguel Santamaria. The 'To do' column has one task for Raphaël Racine: 'Tests finaux'. The 'Doing' column has two tasks: 'Implémentation' for Thibaud Duchoud and 'Implémentation' for Miguel Santamaria. The 'Done' column has one task for Raphaël Racine: 'Mise en place de la DB'. Each task card has a 'Details...' link.

Members	To do	Doing	Done
Raphaël Racine raphaelracine	Tests finaux Details...		Mise en place de la DB Details...
Thibaud Duchoud manamiz		Implémentation Details...	
Miguel Santamaria edri		Implémentation Details...	

Illustration 15: exemple d'un "Dashboard"

On remarque que dans la colonne de gauche se trouve les membres du projet et dans les colonnes suivantes il y a les tâches assignées à chaque membre et leur avancement.

On voit également une différence de couleurs et une icône pour les membres du projet. Lorsqu'un utilisateur a son nom affiché en brun, il est manager du projet. Lorsqu'il a, en plus, une couronne cela signifie que ce membre est le créateur du projet. Lorsqu'il n'a rien de spécial, il est juste un membre lambda du projet.

Chaque tâche affichée l'est également avec une couleur correspondante à sa priorité. Pour une priorité « High » la tâche est de couleur rouge, pour une priorité « Medium » la tâche est de couleur jaune et pour une priorité « Low » la tâche est de couleur verte.

Allons voir plus en détail la composition de ce board.

(1) Au sommet se trouve un « radio button ». Celui sert à changer l'affichage du board. Lorsque l'on clique sur le bouton « View per members » c'est le board affiché ci-dessus qui est affiché et c'est également cette vue là qui permet (et elle seule) de faire des modifications depuis le board (changer l'assignement des tâches ...). C'est également cette vue qui est affichée par défaut.

Lorsque l'on clique sur le bouton « View per tasks », c'est un affichage du board par tâches.

Tasks	To do	Doing	Done
Mise en place de la DB			
Tests finaux			
Implémentation			

Illustration 16: exemple d'un « Dashboard » en vue par tâches

Comme nous pouvons le voir c'est les tâches qui sont affichées dans la colonne de gauche et les utilisateurs affectés à ces tâches qui se trouvent dans les autres colonnes.

Comme énoncé plus haut, il n'y a que la vue par membres qui permet une quelconque édition. Cette vue n'est utilisée que pour afficher les informations d'une manière différente et plus utile dans certaine situation.

Lorsque l'on clique sur l'un des boutons, on envoie une requête à l'action correspondante à l'affichage du board voulu (*BoardViewMembersAction* ou *BoardViewTasksAction*) et l'on charge la réponse (qui est le code html du board) à l'emplacement voulu.

(2) Dans le coin en haut à droite se trouve un autre bouton. Celui permet, au choix, d'afficher ou non la spécialisation des différents membres du projet. Il n'est utile que dans la vue par membres. La vue par tâche n'est donc pas du tout affectée par un appui sur ce bouton ou non.

Members	To do	Doing	Done
Raphaël Racine raphaelracine Tests finaux Details...			Mise en place de la DB Details...
Thibaud Duchoud manamiz Node JS		Implémentation Details...	
Miguel Santamaria edri Base de données Styles CSS		Implémentation Details...	

Illustration 17: board avec l'affichage des spécialisations

Lorsque l'on choisit d'afficher la spécialisation des membres, on remarque un texte apparaître en dessous du nom des membres un texte désignant la ou les spécialisations. Si un des membres ne possède pas spécialisation, un tiret (-) est affiché à la place.

Lors d'un changement d'état du bouton, une animation est lancée et le texte est simplement affiché. Et le dernier état du bouton est également, à chaque changement, sauvegardé dans un cookie afin qu'il reste en état même si l'utilisateur continue de se balader dans les autres pages du site.

Action sur les tâches

Lorsque l'on fait une clic droit sur une tâche dans le board, un menu contextuel s'ouvre et permet de faire différentes actions. Lorsque l'on clique sur un des choix, les informations sur la tâche correspondante sont récupérées (l'id de la tâche et l'id de l'utilisateur qui y est affecté).

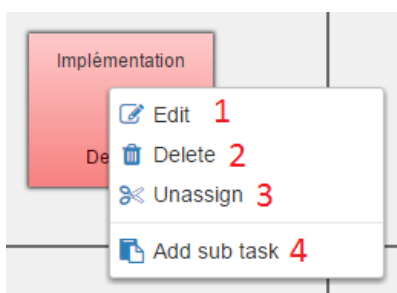


Illustration 18: menu contextuel d'une tâche

Edition (1)

L'édition se fait lorsque l'on choisit, dans le menu contextuel, le choix *Edit*.

Tout ce qui sera fait c'est que l'utilisateur sera redirigé sur la page d'édition d'une tâche (.../editTask/ « taskId »). C'est donc à l'action *editTaskAction* qu'une requête sera envoyée en y mettant à la place de *taskId* l'id de la tâche sélectionnée.

Un message sera affiché en cas de réussite ou d'erreur afin d'en informer l'utilisateur.

Suppression (2)

Pour supprimer une tâche, il faut simplement choisir *Delete* dans le menu contextuel. Il se passe la même chose que pour l'édition. Lorsque l'on sélectionne *Delete* il va simplement envoyer une requête à l'action *deleteTaskAction* (.../deleteTask/ « taskId »). L'id de la tâche sélectionnée sera mise à la place de *taskId* dans l'url.

Un message sera affiché en cas de réussite ou d'erreur afin d'en informer l'utilisateur.

Création d'une sous-tâche (4)

Pour créer une sous-tâche il suffit de choisir *Add sub task* dans le menu contextuel.

Tout ce qu'il va se passer est que l'utilisateur va être redirigé sur la page d'ajout d'une nouvelle tâche mais passant cette fois l'id de la tâche parent (.../addTask/ « parentId ») qui n'est autre que l'id de la tâche sélectionnée.. C'est donc l'action *addTaskAction* en mode sous-tâche qui est appelée et qui se charge d'ajouter la sous-tâche.




Désassignation (3)

Pour désassigner une tâche, il suffit de choisir *Unassign* dans le menu contextuel.

Comme pour la suppression, on va envoyer une requête à l'action *unassignTaskAction* (.../unassignTask/ « taskId »). L'id de la tâche sélectionnée sera mise à la place de *taskId* dans l'url.

Un message sera affiché en cas de réussite ou d'erreur afin d'en informer l'utilisateur.

Assignment

Members	To do	Doing	Done
 Raphaël Racine raphaelracine	<div>Tests finaux</div> <div>Details...</div>		<div>Mise en place de la DB</div> <div>Details...</div>
 Thibaud Duchoud manamiz Node JS		<div>Implémentation</div> <div>Details...</div>	
 Miguel Santamaria edri Base de données Styles CSS		<div>Implémentation</div> <div>Details...</div>	

Tasks

Mise en place de la DB
Tests finaux
Implémentation

1

Members

raphaelracine - remove
manamiz
edri - remove

Illustration 19: assignment d'une tâche

En dessous du board, il y a la liste des tâches du projet qui est affichée (1). Lorsqu'un utilisateur veut assigner une tâche à un autre utilisateur, il fait une cliquer-glisser du lien dans la liste vers la ligne correspondante à l'utilisateur à qui il souhaite assigner la tâche.

Lorsque l'utilisateur lâche la tâche dans le board, l'événement *ondrop* est déclenché. Il va vérifier si ce qui vient d'être lâché est une tâche pour assignation ou une tâche qui est déplacée directement depuis le board. Dans notre cas il s'agit d'une assignation.

Cet événement va récupérer les informations nécessaires à l'affectation : l'id de l'utilisateur cible et l'id de la tâche.

Il va ensuite envoyer une requête au *ProjectController* à l'action *AssignTaskAction*. Il va ensuite récupérer le code renvoyé par la tâche est affiché un feedback. Si l'utilisateur n'a pas les droits ou qu'une erreur s'est produite (déjà assignée ...) un message en rouge sera affiché.

Afficher les détails d'une tâche

Il y a la possibilité d'afficher les détails d'une tâche et donc également le fil d'actualité qu'il lui est associé.

Deux manières sont possibles. Soit en cliquant sur une tâche dans la liste se trouvant en dessous du board soit en cliquant sur le lien *Details...* qui se trouve sur la tâche dans le board.

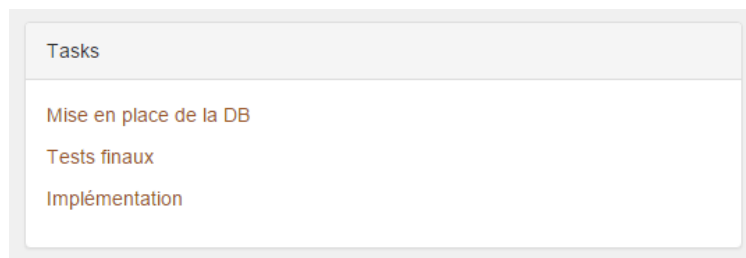


Illustration 20: liste des tâches d'un projet

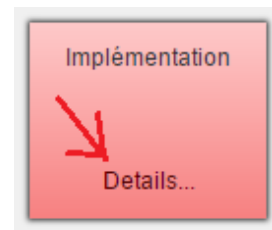


Illustration 21:
détails d'une tâche

Lorsqu'on clique sur un de ces liens, l'utilisateur est redirigé sur la page d'affichage des détails d'une tâche (.../taskDetails/ « taskId ») et l'id de la tâche est passée dans l'url à la place de *taskId*. Lorsque l'utilisateur est redirigé sur cette page c'est l'action *taskDetailsAction* qui est appelée et qui s'occupe d'afficher la vue à l'utilisateur.

Déplacer une tâche

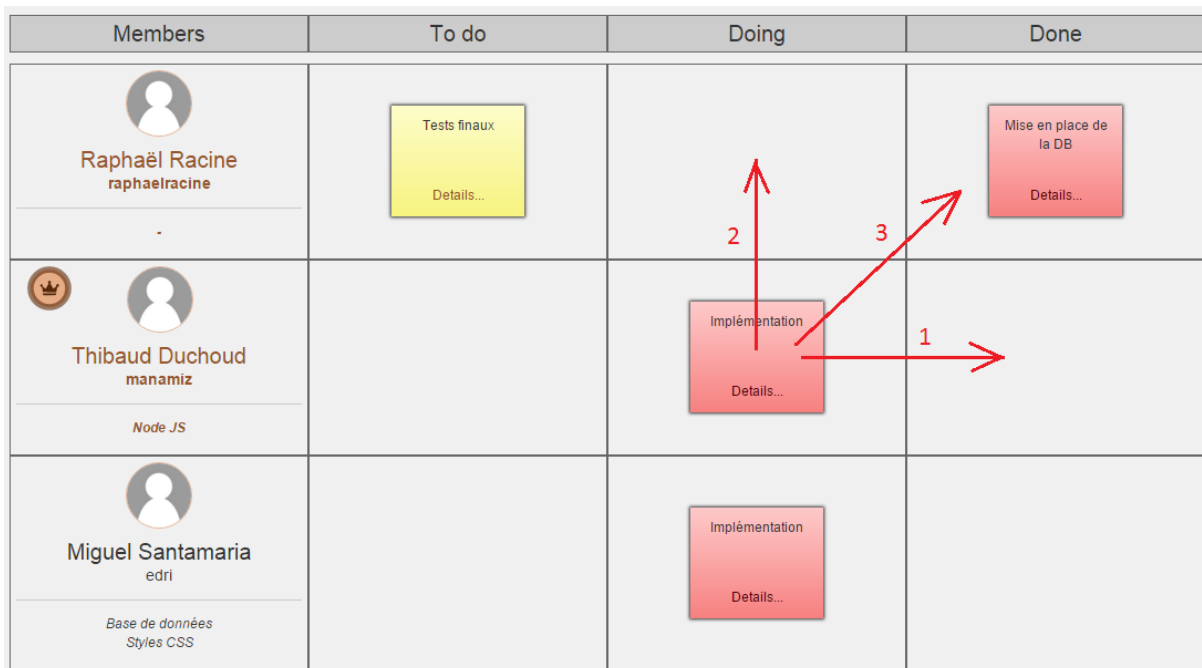


Illustration 22: board avec déplacement d'une tâche

Lors du déplacement d'une tâche dans le board, il y a plusieurs possibilités. Soit on déplace la tâche vers une autre colonne (de *Doing* à *Done* par exemple) (1). Soit on déplace la tâche vers un autre utilisateur (2). Soit les deux (3).

Comme cette action utilise le serveur d'événement, le déplacement s'opère en temps réel chez les autres membres du projet qui se trouvent également sur cette page.

1. Dans le premier cas, il n'y pas de changement de membre et donc pas de réaffectation. C'est juste l'état de la tâche qui change (qui passe de *Doing* à *Done* dans l'image ci-dessus). Ce déplacement va également faire bouger la tâche du fond car un état est lié à une tâche et non à un utilisateur qui travaille sur la tâche.
2. Dans ce second cas, l'état de la tâche ne change pas il y a une réaffectation.
3. Dans ce dernier cas, il y a une réaffectation de la tâche et un changement d'état de la tâche.

Lorsqu'un utilisateur déplace une tâche et qu'il la lâche, il appelle le même événement *ondrop* que pour l'assignation d'une tâche. Mais cette fois l'événement n'est pas en mode affectation.

L'événement va récupérer les informations nécessaires à la vérification, aux droits et aux événements concernant le déplacement.

Il récupère l'id de la tâche, le nom de la colonne de départ, le nom de la colonne d'arrivée, l'id de l'utilisateur de départ et l'id de l'utilisateur d'arrivée.

Ces informations sont ensuite envoyées à l'aide d'une requête POST à l'action *moveTaskAction* (.../moveTask). Cette action va faire les vérifications nécessaires et retourner un code de succès ou d'erreur.

Action sur les sous-tâches

Rappel sur les sous-tâches

Lorsqu'un utilisateur crée une sous-tâche, celle-ci est liée à sa tâche parent.

Les sous-tâches sont affichées sous leur parent dans la liste des tâches se trouvant en dessous du board.

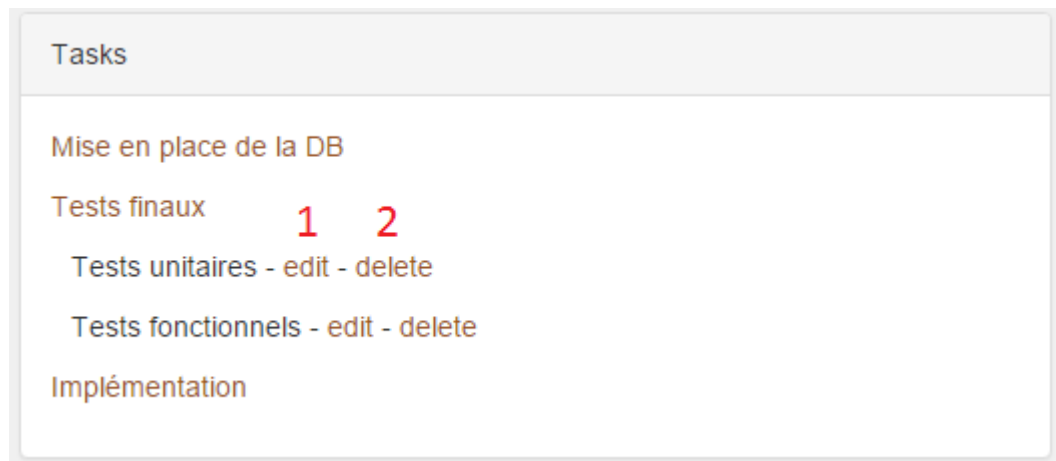


Illustration 23: liste des tâches avec sous-tâches

Les sous-tâches sont là pour pouvoir séparer et/ou différencier les différentes parties d'une tâche principale.

Une sous-tâche ne peut pas changer son parent. Il faut obligatoirement, pour changer le parent d'une sous-tâche, la supprimer et la recréer en sélectionnant le nouveau parent.

Dans le board, elle apparaissent dans leur tâches parentes :

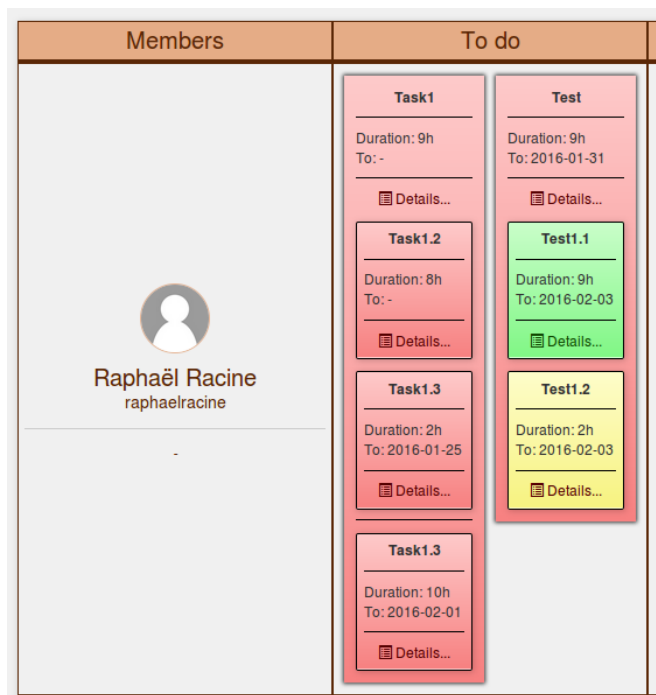


Illustration 24: sous-tâches dans le dashboard

Il n'est pas possible de les en défaire, même si nous pensions le faire au début, mais cela est dû à un manque de temps.

Edition (1)

Pour éditer une sous-tâche, il suffit de cliquer sur le lien *edit* se trouvant à côté de la sous-tâche dans la liste.

Ce lien va rediriger l'utilisateur sur la page d'édition d'une tâche et donc laisser l'action *editTaskAction* gérer la suite. Cette action ne fait pas de différence entre une tâche et une sous-tâche. On lui envoie donc une requête normale comme quoi l'on souhaite éditer une tâche avec l'id qui lui correspond. La requête est envoyée à l'url : `.../editTask/ « taskId »`, où le paramètre `taskId` est remplacé par l'id de la tâche correspondante.

Suppression (2)

Pour éditer une sous-tâche, il suffit de cliquer sur le lien *delete* se trouvant à côté de la sous-tâche dans la liste.

Ce lien envoie une requête à l'action *deleteTaskAction* à l'url : `.../deleteTask/ « taskId »`, où le paramètre `taskId` est remplacé par l'id de la tâche correspondante.

La page sera alors rafraîchie pour prendre en compte les changements.

Serveurs secondaires d'événements

Introduction

En plus du serveur PHP de base qui fait tourner l'environnement Zend, nous avons créé deux autres serveurs d'événements dans le langage Node.js (dérivé de JavaScript), permettant respectivement de gérer des connexions et de transférer des données de manière dynamique via des websockets, respectivement de gérer des requêtes HTTP de type POST.

Ils sont tous les deux situés dans le fichier « `eventsServers/index.js` », fonctionnent de manière simultanée et possèdent un environnement d'exécution et de contexte semblable, leur permettant de partager des variables.

Leur utilité au sein du projet se fait ressentir dans deux pages :

- « `/project/[idProjet]` » : pour le rafraîchissement dynamique de l'historique, ainsi que du dashboard au fil des différentes actions utilisateurs générant des événements.
- « `/project/[idProjet]/taskDetails/[idTache]` » : pour le rafraîchissement dynamique du fil d'actualité, ainsi que pour les détails de la tâche.

Diagramme de fonctionnement

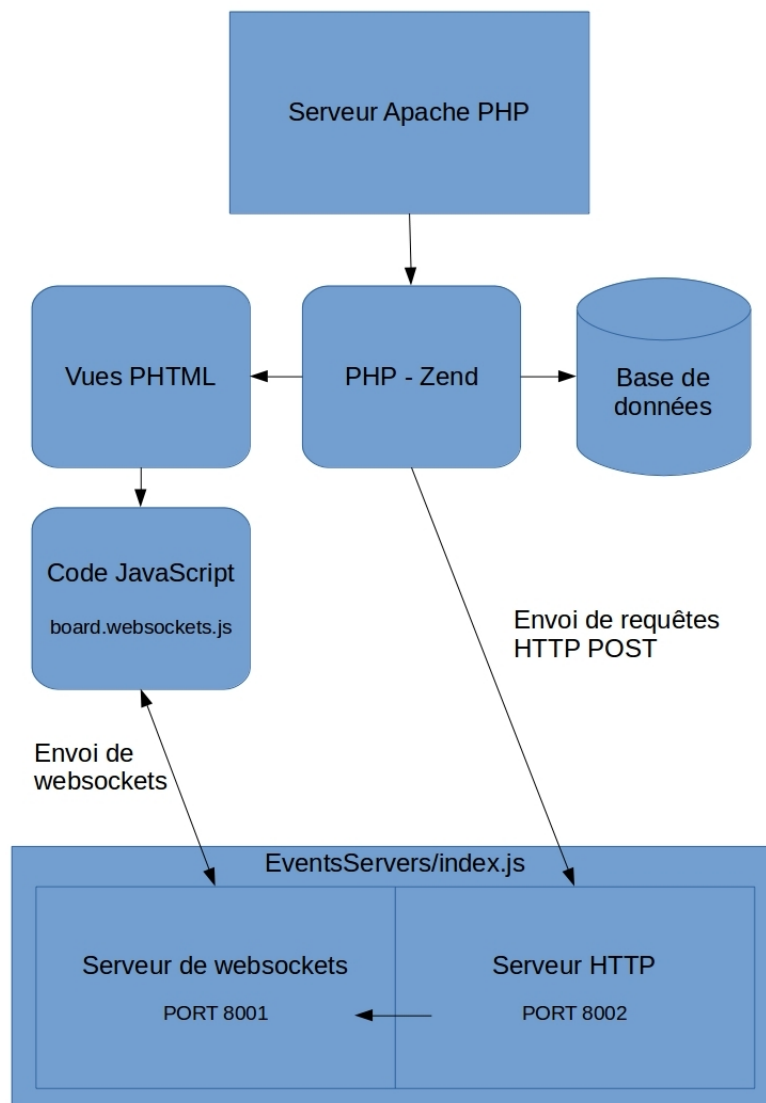


Illustration 25: diagramme de fonctionnement des différents serveurs.

Rôle des serveurs

Chaque serveur d'événements a un rôle qui lui est propre :

- **Serveur HTTP** (<http://127.0.0.1:8002>) : reçoit des requêtes HTTP de type POST de la part de diverses actions de divers contrôleurs, formaté de la manière suivante :

```
{  
    "requestType": [typeDeRequete],  
    [données relatifs au type du requête (données d'un événement, etc.)]  
}
```

Les données sont envoyées sous forme de formulaires HTTP (par exemple:

« <http://127.0.0.1:8002?userId=2&projectId=1> ») ; si un champ de la requête possède plusieurs autres champs (par exemple un événement), ceux-ci seront formatés en JSON.

Lorsque le serveur reçoit une requête, il s'occupe de traiter les données contenues à l'intérieur selon son type, puis demande au serveur de websockets d'envoyer un socket à tous les clients concernés.

Voici les différents types de requête qui peuvent être reçus par le serveur :

- **joinProjectRequest** : ce message est reçu par l'action *index* du contrôleur *Project*, et indique au serveur qu'un client souhaite se connecter au flux de messages du projet (plus de détails plus bas, dans « **Processus de connexion d'un client au serveur** »). Il contient l'ID de l'utilisateur courant, ainsi que celui du projet.
- **joinTaskRequest** : ce message est reçu par l'action *taskDetails* du contrôleur *Project*, et indique au serveur qu'un client souhaite se connecter au flux de messages de la tâche (plus de détails plus bas, dans « **Processus de connexion d'un client au serveur** »). Il contient l'ID de l'utilisateur courant, ainsi que les IDs de la tâche et du projet concernés.
- **newEvent** : ce message est reçu de la part de nombreuses actions du contrôleur *Project*. Il contient les données d'un événement précis (obtenues depuis la vue *view_events* de la base de données) sous forme de JSON, qui sera retransmis à des clients précis. Plus précisément, le serveur HTTP va envoyer un socket via le serveur websockets à chacun des clients concernés par l'événement : s'il s'agit d'un événement provenant de la page de projet, il sera renvoyé à tous les utilisateurs connectés actuellement à la page de ce projet ; s'il s'agit d'un événement provenant de la page de détails d'une tâche, il sera renvoyé à tous les utilisateurs connectés actuellement sur la page de cette tâche. Afin de déterminer la provenance de l'événement, ce dernier contient un champ « *isTaskEvent* » qui indique s'il vient de la page de tâche ou non.
- **newEvents** : même fonctionnement que **NewEvent**, mais avec un tableau d'événements qui peuvent appartenir à un projet et/ou à une tâche.

- **taskMoving** : ce message est reçu par l'action *moveTask* du contrôleur *Project* lorsqu'un utilisateur a déplacé une tâche ; il contient les nouvelles coordonnées de la tâche. Le serveur HTTP va ensuite envoyer un socket via le serveur websockets pour avertir les différents clients connectés dans le projet, afin que la tâche se déplace dynamiquement.
 - **taskDeleted** : ce message est reçu par l'action *deleteTask* du contrôleur *Project* lorsqu'un utilisateur a supprimé une tâche d'un projet ; il contient l'ID de la tâche, ainsi que le nom de l'utilisateur qui l'a supprimée. Le serveur HTTP va ensuite envoyer un socket via le serveur websockets pour avertir les différents clients connectés dans la tâche, afin qu'ils soient redirigés.
 - **taskEdited** : ce message est reçu par l'action *editTask* du contrôleur *Project* lorsqu'un utilisateur a édité une tâche ; il contient l'ID de la tâche, ainsi que ses nouvelles valeurs sous forme de JSON. Le serveur HTTP va ensuite envoyer un socket via le serveur websockets pour avertir les différents clients connectés dans la tâche, afin qu'elle soit dynamiquement éditée.
 - **memberRemoved** : ce message est reçu par l'action *removeMember* du contrôleur *Project* lorsqu'un utilisateur a été retiré d'un projet ; il contient les IDs du projet et de l'utilisateur retiré, ainsi que le nom de l'administrateur responsable de l'action. Le serveur HTTP va ensuite envoyer un socket via le serveur websockets pour avertir le client représentant l'utilisateur, afin qu'il soit redirigé.
- **Serveur de websockets** (ws://127.0.0.1:8001) : reçoit et envoie des messages de type JSON au client (situé dans le fichier « public/js/board.websockets.js »). Chacun des messages JSON est formaté de la manière suivante :

```
{  
    "messageType": [typeDuMessage],  
    [données relatifs au type du message (id de projet, id de tâche, etc.)]  
}
```

Le serveur/client s'occupe ensuite de traiter les données contenues dans le message, selon son type, puis d'agir en conséquence.

Les deux seuls sockets que peuvent envoyer les clients sont les sockets de confirmation à l'accès d'une entité : *ProjectListeningRequest* et *TaskListeningRequest*. Les autres requêtes sont effectuées à l'aide de requêtes HTTP. En effet, le client des websockets est codé en JavaScript du côté du client (contrairement aux requêtes HTTP qui sont envoyées du côté serveur par les contrôleurs), ce qui peut poser de gros problèmes de sécurité : un utilisateur malintentionné peut facilement injecter ou modifier le code JavaScript, puisqu'il se situe du côté client (donc sur son navigateur). Plus de détails sur le processus de connexion au serveur websockets sont donnés plus bas.

- Voici les différents types de websockets que le serveur peut envoyer aux clients :
 - **newTaskEvent** – utilisé dans la page de détails de tâche : est envoyé au client lorsqu'un utilisateur a posté une nouvelle actualité dans le fil d'actualités d'une tâche et l'a envoyé au serveur HTTP via les requêtes « **newEvent** » ou « **newEvents** ». Ce dernier demande alors au serveur de websockets d'envoyer la nouvelle actualité à tous les clients actuellement connectés à la tâche, afin que le système l'ajoute dynamiquement au fil d'actualité. Ce message possède un deuxième paramètre « event » qui contient les données de l'événement, reçues depuis la requête HTTP.
 - **newEvent** – utilisé dans la page de projet : même fonctionnement que pour **NewTaskEvent**, mais avec l'historique d'un projet. A noter qu'à chaque événement reçu, le client rafraîchit le dashboard.
 - **taskMovingEvent** – utilisé dans la page de projet : est envoyé au client lorsqu'un utilisateur a déplacé une tâche dans le dashboard et a envoyé via le contrôleur la confirmation au serveur HTTP à l'aide de la requête « taskMoving ». Lorsque le client reçoit ce message, il déplace dynamiquement la tâche qui a été déplacée dans la page « project ». Ce message possède aussi les nouvelles coordonnées de la tâche déplacée.
 - **taskDeleted** – utilisé dans la page de détails de tâche : est envoyé au client lorsqu'un utilisateur a supprimé une tâche et a envoyé via le contrôleur la confirmation au serveur HTTP à l'aide de la requête « taskDeleted ». Lorsque le client reçoit ce message, il redirige automatiquement tous les utilisateurs qui étaient présent dans la page de détails de la tâche supprimée, en leur affichant un message d'information. Ce message possède aussi l'ID de la tâche supprimée.
 - **taskEdited** – utilisé dans la page de détails de tâche : est envoyé au client lorsqu'un utilisateur a édité les détails d'une tâche et a envoyé via le contrôleur la confirmation au serveur HTTP à l'aide de la requête « taskEdited ». Lorsque le client reçoit ce message, il modifie dynamiquement les détails de la tâche pour tous les utilisateurs qui sont présents dans la page de détails de cette dernière. Ce message possède aussi différents champs de la tâche éditée.
 - **memberRemoved** – utilisé dans la page de projet : lorsqu'un administrateur du projet en a retiré un membre, il a envoyé via le contrôleur la confirmation au serveur HTTP à l'aide de la requête « memberRemoved ». Le serveur HTTP demande ensuite au serveur de websockets d'envoyer une requête au client qui correspond à l'utilisateur retiré, s'il est connecté. Lorsque le client reçoit ce message, il en informe l'utilisateur et le redirige sur la liste des projets. Ce message possède aussi l'ID de l'utilisateur retiré, l'ID du projet concerné et le nom de l'administrateur qui l'a retiré.

Processus de connexion d'un client au serveur

Lorsqu'un utilisateur désire se connecter au serveur, le processus se déroule en deux temps :

- Premièrement, avant que l'utilisateur accède à la page, l'action concernée du contrôleur *Project* envoie une requête HTTP au serveur HTTP, lui indiquant que l'utilisateur souhaite rejoindre le flux de sockets de la page de projet/tâche courant. Cette requête contient l'ID de l'utilisateur, ainsi que l'ID de l'entité souhaitant être accédée.
Cela permet au serveur HTTP d'initialiser un objet dans un tableau d'attente, dans lequel il va indiquer que l'utilisateur dont l'ID est donné est en attente d'accès pour l'entité dont l'ID est aussi donné.
A partir de ce moment, l'utilisateur donné aura le droit d'accéder à l'entité donnée lorsqu'il effectuera la deuxième partie du processus.
- Deuxièmement, lorsque le client a accédé à la page, il envoie instantanément au serveur de websockets (via le script `board.websockets.js`) un message de type *ProjectListeningRequest* ou de type *TaskListeningRequest*, selon si l'utilisateur se situe dans la page de projet, respectivement la page de détails de tâche. Ce message contient l'ID de l'utilisateur client (provenant de la base de données), ainsi que l'ID du projet / de la tâche dans lequel/laquelle il se situe, selon le type de message. Une fois ce message reçu, le serveur de websockets va contrôler que le client dont l'ID est donné est actuellement en file d'attente du projet donné. Si ça n'est pas le cas, la connexion sera purement et simplement rejetée ; si les accès sont accordés, il va initialiser une connexion websockets permanente avec le client (jusqu'à ce que ce dernier quitte la page), en indiquant dans l'objet représentant la connexion le type de connexion (projet/tâche), l'ID de l'utilisateur, ainsi que l'ID de l'entité dans laquelle il se trouve. Ces données serviront à envoyer les sockets aux bons utilisateurs lorsqu'un événement apparaît.

Actuellement, les éventuels problèmes de concurrence ont été ignorés pour des problèmes de temps. A noter qu'ils sont finalement bénins, car dans le cas où un même utilisateur se connecte simultanément depuis deux navigateurs, l'un des deux aura accès aux réceptions websockets, tandis que l'autre non. Dans tous les cas, il pourront envoyer des requêtes HTTP d'événements, et leurs actions seront prises en compte sans avoir d'autres impacts.

Conclusions

Problèmes rencontrés

Comme dans tout projet, nous avons rencontré des difficultés tout au long du semestre, même si aucune d'entre-elles ne s'est avérée critique.

Problèmes organisationnels

- Maintenir un journal de bord en essayant de suivre au mieux possible la planification initiale.
- En tant que chef de projet, gérer correctement l'équipe de développeurs.
- Attribuer les tâches à chacun ; il n'est en effet pas aisé de répartir équitablement les charges, à cause des différences de niveaux que possèdent les membres.
- Expliquer correctement son travail aux autres membres du groupe, afin qu'ils puissent le comprendre et l'utiliser. Il est souvent difficile d'expliquer aux autres ce que nous avons implémenté, même après l'avoir bien compris.
- Le temps s'est avéré difficile à gérer sur la fin du semestre, à cause de la quantité de travail donnée dans les autres cours auxiliaires.
- Les seuls membres à relativement bien connaître les technologies Web et plus particulièrement Zend étaient Thibaud et Miguel ; il a donc fallu effectuer des formations auprès des autres développeurs, afin qu'ils puissent à leurs tours comprendre.
- Le suivi des conventions de codage, car chacun possède ses propres conventions.

Difficultés techniques

- L'installation du framework Zend s'est avérée relativement compliquée et chronophage, à cause des différences existantes entre les machines des membres (OS, versions de PHP, configuration, etc.) ; nous avons tout de même pu nous en sortir.
- L'hébergement du site sur le web nous a pris une journée complète, dû aux multiples différences présentes sur le serveur de l'hébergeur.
- La gestion de la sécurité sur le site. En effet, celle-ci est critique dans toute la partie cliente d'un site web.
- La réalisation des serveurs d'événements, dont l'implémentation s'est avérée nouvelle pour tout le monde. Nous avons tout d'abord pensé les coder en PHP ; grosse erreur de notre part !
- La gestion générale du dashboard, qui s'est avérée être une tâche complexe, même si nous nous y attendions.

Points positifs

Il n'y a évidemment pas eu que des points négatifs ! Voici une liste non-exhaustive de ce qui nous a permis de terminer ce projet avec succès :

- Chacun savait ce qu'il avait à faire, ainsi que le délai de ses tâches ; cela nous a permis de concevoir une relativement bonne organisation tout au long du semestre.
- Tous le monde a joué le jeu, ce qui a procuré une bonne ambiance au sein du groupe.
- Le projet s'est avéré extrêmement intéressant, tant au niveau des implémentations faites qu'au niveau des découvertes de nouvelles technologies.
- Nous sommes arrivés à nos fins, si ça n'est à quelques détails près, déclarés ci-dessous.

Fonctionnalités

Toutes les fonctionnalités que nous pourrions dire comme étant majeures ont été implémentées. Cela veut dire que l'utilisateur peut se connecter utiliser le site de manière agréable. Cependant, quelques fonctionnalités qui étaient prévues initialement n'ont pas pu être implémentées, ou alors pas entièrement, par manque de temps. En effet, nous avons préféré mettre la priorité sur des tâches qui étaient plus prioritaires :

- La suppression d'un projet n'a pas pu être implémentée.
- De même, aucun utilisateur ne peut quitter délibérément un projet ; il doit pour cela actuellement demander à un administrateur de le retirer.
- La vue **View per tasks** du dashboard est actuellement en lecture seule, car nous n'avons pas eu le temps d'en développer les fonctionnalités, bien que très semblables à l'autre vue.
- La gestion des sous-tâches a dû être abrégée, par manque de temps à nouveau : une sous-tâche est liée physiquement à sa tâche parente, et ne peut pas en être séparée. Elle possède cependant ses propres informations et son propre fil d'actualités.

Malgré ces points, nous pensons avoir fourni un travail de qualité, et, dans les grandes lignes, sommes parvenus à nos fins en respectant la plupart de ce qui avait été planifié. Le projet livré est propre, documenté et opérationnel.

A noter que certaines fonctionnalités développées n'étaient pas présentes dans le cahier des charges, parmi lesquels :

- L'ajout d'un droit supplémentaire dans un projet – le créateur.
- L'assignement de plus d'une spécialisation à un même utilisateur d'un même projet.
- La création d'une rubrique d'aide de type F.A.Q. pour les nouveaux utilisateurs.

Finalement, quelques changements ont été effectués par rapport au cahier des charges, car nous les avons jugés plus utiles :

- Changements aux niveau de la base de données, qui a évoluée au fil des demandes.
- Le dashboard contient désormais 3 colonnes au lieu des 4 qui étaient prévues au départ, car nous jugions la colonne « Code Review » superflue. De plus, avec 4 colonnes, le rendu du dashboard était beaucoup moins sexy.
- Seuls les administrateurs peuvent assigner des tâches aux autres membres, au lieu de tous les membres.
- Le design de certaines pages a été revu pour des questions d'ergonomie.

Améliorations futures

Nous pouvons imaginer beaucoup d'améliorations possibles pour EasyGoing! En énumérant les plus importantes d'entres-elles, outre le fait de corriger les choses qui n'ont pas pu être terminées, cela nous donne :

- La réalisation d'une version mobile du site web ; en effet, il est actuellement possible d'y accéder via un appareil mobile, mais le rendu n'est pas très élégant.
- Rendre les e-mails plus agréables à lire.
- Effectuer une inscription sécurisée à l'aide d'une validation par e-mail.
- Installer un certificat de sécurité pour pouvoir utiliser HTTPS.
- Offrir un support plus élargi aux utilisateurs.
- Repasser correctement sur tout le code pour l'optimiser.
- Etoffer un peu plus les tutoriels d'utilisation.
- Pouvoir se partager des fichiers entre membres d'un même projet.
- Pouvoir personnaliser manuellement les différents éléments graphiques du dashboard (tâches, rendu du tableau, etc.).
- Pouvoir envoyer des messages privés aux autres membres.
- Améliorer la recherche de membres, qui peut s'avérer lente et peu pratique si le site possède un grand nombre d'utilisateurs.

Conclusion finale

Tout comme l'année passée, ce projet de groupe s'est avéré extrêmement utile pour notre formation, puisqu'il nous a permis de gagner une grande expérience, tant aux niveaux organisationnels que techniques, le tout en ayant su relever notre enthousiasme. Au travers de ce projet, nous avons pu toucher à non pas un seul, mais bien quatre langages différents, comme le montre ces statistiques récupérées sur GitHub :

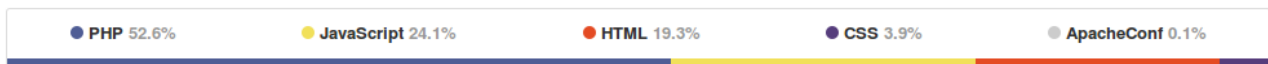


Illustration 26: statistiques d'utilisation des langages de programmation

Malgré les quelques problèmes rencontrés, se résumant surtout par le temps et la dose de travail donnée dans les autres cours, nous sommes arrivés quasiment à nos fins, et sommes heureux de pouvoir présenter une application fonctionnelle documentée. Nous avons au tout début du projet misé sur l'aspect ergonomique du programme, et nous pensons pouvoir affirmer qu'il s'agit d'un but qui a été atteint.

Nous évaluons donc notre travail comme étant bien, bien que pas parfait.