## Base de données (Conception)

Cette partie du rapport concerne la conception de la base de données.

Voici dans un premier temps notre modèle de données conceptuel (sous forme d’un schéma UML) répondant au mieux au cahier des charges, afin de pouvoir s’y appuyer pour réaliser la base de données relationnel.

Il est à noter qu’il ne s’agit pas du schéma relationnel définitif et qu’il est sous réserve de modifications pour la phase de réalisation, mais les bases sont là.

#### Schéma UML

Voici le schéma UML de notre base de données (modèle de domaine) :

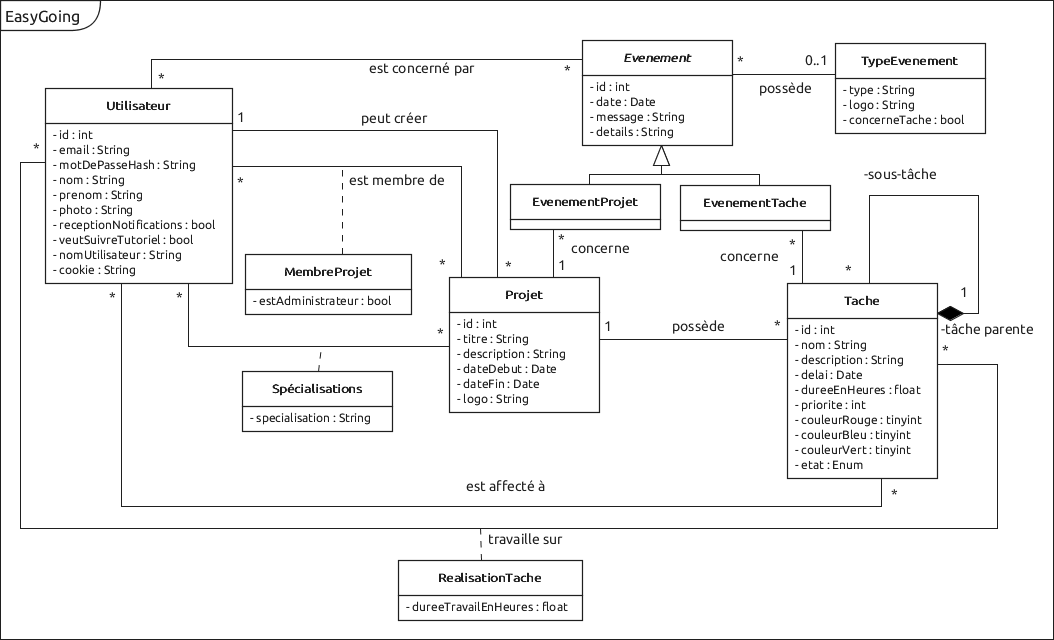


Figure - Modèle de domaine

***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 :   * Un titre 🡪 *Projet PDG 2015* * Une description 🡪 *Projet de semestre à la HEIG-VD* * Une date de début 🡪 *01.10.2015* * Une date de fin 🡪 *04.01.2016* |
| Tâche | Cette entité représente une tâche d’un projet. Elle est caractérisée par :   * Un nom 🡪 *Faire la vaisselle* * Un délai 🡪 *05.11.2015* * Une description 🡪 *Mettre toutes les assiettes dans le lave-vaisselle* * Une durée en heures 🡪 *La durée estimée de la tâche en heures* * Une priorité  🡪 *Haute* |
| Utilisateur | Cette entité représente un utilisateur. Chaque utilisateur est caractérisé par :   * Un email 🡪 [*raphael.racine@heig-vd.ch*](mailto:raphael.racine@heig-vd.ch) * Un nom d’utilisateur 🡪 *raphy* * Un mot de passé (sous forme de hash) 🡪 *ebfh451v65sd1561dfsdf1sdav5s6a* * Un nom 🡪*Raphaël* * Un prénom *🡪 Racine* * Un booléen « réception des notifications » (pour savoir si l’utilisateur désire recevoir des notifications ou pas) 🡪 *True* * Un booléen « veut suivre tutoriel » (pour savoir si l’utilisateur désire avoir le tutoriel qui apparaît) 🡪 *False* * Un cookie qui lui permettra de se reconnecter 🡪 *14561f8avsdfdsfsda4fsa* |
| Evénement | Cette entité représente un événement (d’une tâche ou d’un projet). Un événement est caractérisé par :   * Une date 🡪 *28.01.2016* * Un message 🡪 *raphaelracine a terminé la tâche X* * 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 | Cette relation représente le fait qu’un utilisateur soit membre d’un projet.  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 ») |
| Projet et Utilisateur (Spécialisations) | N - N | Cette relation représente le fait qu’un membre puisse avoir plusieurs spécialisations dans un projet. *Par exemple le membre X est spécialisé dans « Java » et dans « Base de données » pour ce projet.*  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. |
| Projet et Utilisateur | N – 1 | Cette relation représente le fait qu’un utilisateur a créé un projet (elle permettra de savoir qui a créer 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. *Par exemple le membre X a travaillé 3 heures sur la tâche Y.* |
| Tâche et Tâche | 1 – N | Cette relation représente le fait qu’une tâche peut avoir plusieurs sous-tâches*. Par exemple faire la vaisselle c’est :*   * *Nettoyer les assiettes* * *Nettoyer les verres* |
| Utilisateur et Evénement | N – N | Cette relation représente le fait qu’un utilisateur soit concerné par un événement. Un événement peut concerner plusieurs utilisateurs. |
| Evénement et TypeEvénement | N – 1 | Cette relation indique qu’un événement possède un type. |
| Evénement et EvénementProjet | Héritage | L’entité EvenementProjet concerne un projet et elle hérite de l’entité Evénement.  Elle a comme caractéristique supplémentaire le fait qu’elle concerne un projet. |
| Evénement et EvénementTache | Héritage | L’entité EvenementTache concerne une tâche et elle hérite de l’entité Evénement.  Elle a comme caractéristique supplémentaire le fait qu’elle concerne une tâche. |
| Projet et EvénementProjet | 1 – N | Cette relation représente simplement le fait qu’un événement de projet est lié à un projet. |
| Tâche et EvénementTâche | 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

## Base de données (Réalisation)

#### 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

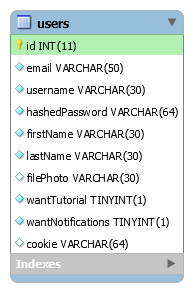


Figure - Schéma relationnel - Utilisateurs

|  |  |  |  |
| --- | --- | --- | --- |
| **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’utilisater |
| 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



Figure - 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

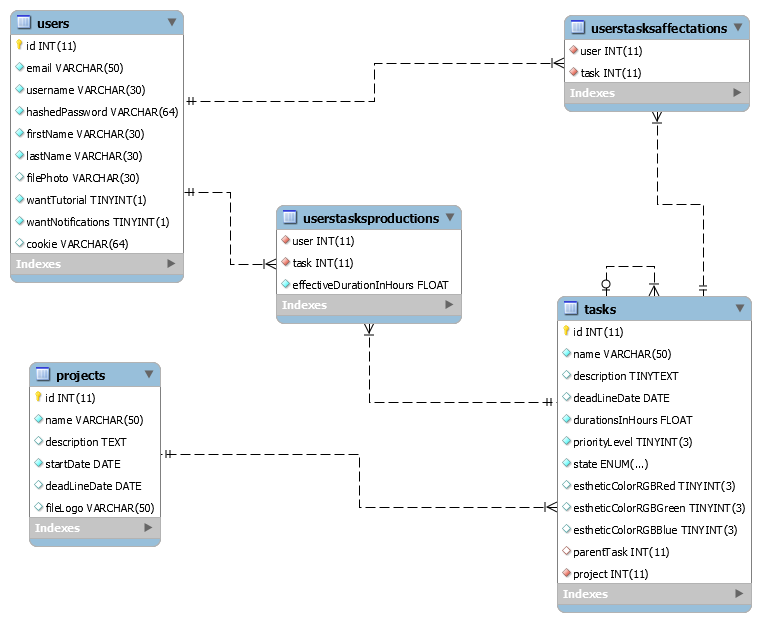


Figure - Schéma relationnel - Tâches

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Champs** | **Type** | **Explication** |
| Tasks | Id | INT NOT NULL PRIMARY KEY AUTO\_INCREMENT | Clé primaire de la table |
| name | VARCHAR(50) NOT NULL | Nom du 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.* | | |

##### Evénements

Ici sont expliquées les tables concernant les événements :

* Events
* EventTypes
* EventsOnProjects
* EventsOnTasks
* EventsUsers

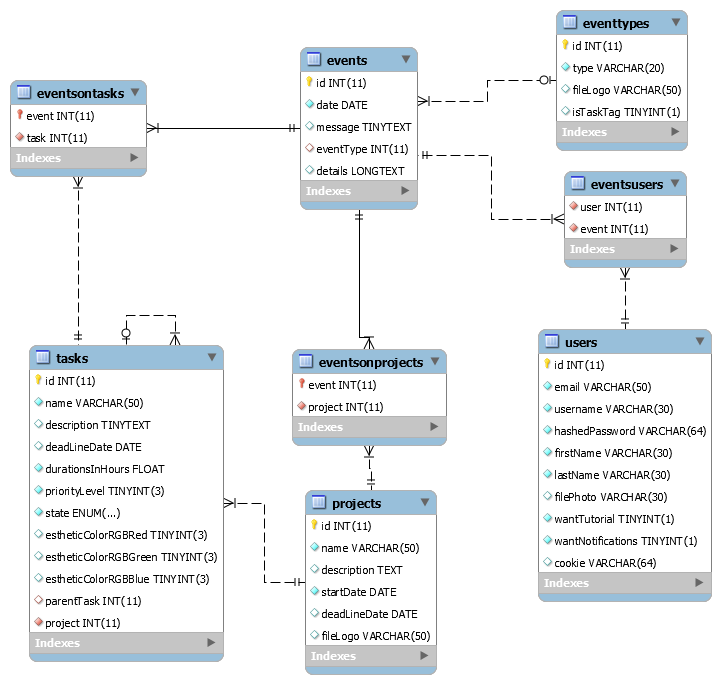


Figure - Schéma relationnel - Evénements

|  |  |  |  |
| --- | --- | --- | --- |
| **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 :

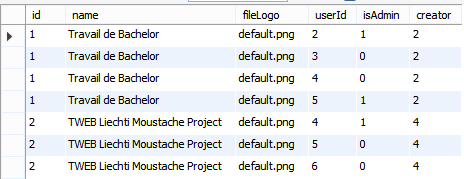


Figure - Exemple de vue - view\_projects\_min

* **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 :

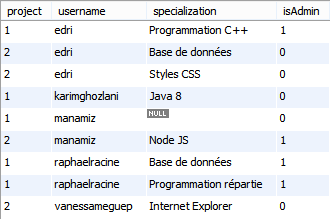


Figure - Exemple de vue - view\_projects\_members\_specializations

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 :

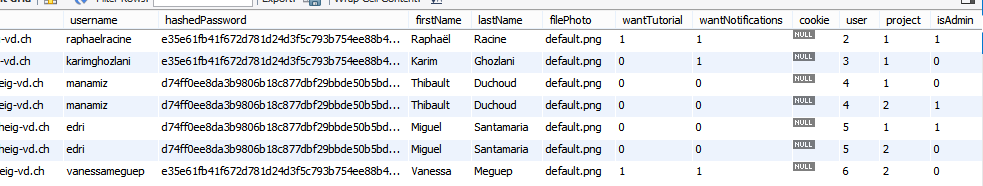


Figure - Exemple de vue - view\_users\_projects

***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’utilisateurs, et l’id des tâches)

Exemple :

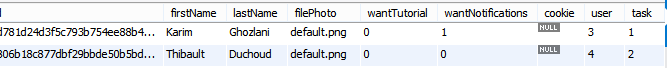


Figure - Exemple de vue - view\_tasks\_users

***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 :

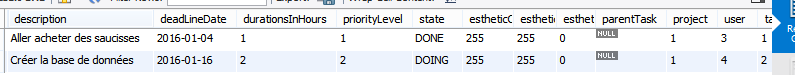


Figure - Exemple de vue - view\_users\_tasks

***Remarque****: Certaines colonnes ne sont pas affichées ici*

* **view\_projects\_details**
  + Cette vue permet d’afficher les détails d’un projet

Exemple

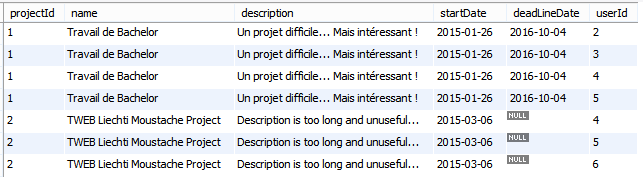


Figure - Exemple de vue - view\_projects\_details

* **view\_events**
  + Cette vue permet d’afficher les événements qui sont arrivés sur une tâche ou un projet

Exemple :

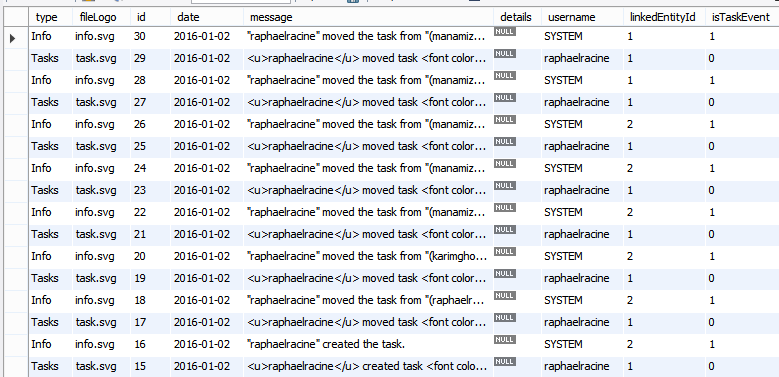


Figure - Exemple de vue - view\_events

#### 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. |
| checkUserCanBeAffectedToTask | 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

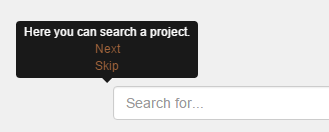
Voici de quelle manière ont été implémentées les contraintes d’intégrité au niveau de la base de données :

* 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 UsersTasksAffectactionsBeforeInsert et UsersTaksAffectationsBeforeUpdate
* 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

## Tutoriel

### Réalisation

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



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.

##### Rôle du contrôleur « TutorialController »

Ce contrôleur permet 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 long) 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 : easygoing/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 : easygoing/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 : easygoing/project/{idProjet}/taskDetails/{idTache}. Cette action fait appel au modèle Tutorial (méthode taskDetails).

#### 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 :

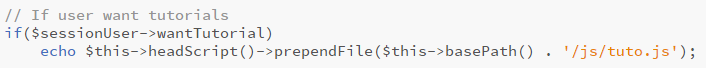


Figure - Code chargement script tutoriel

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)

##### 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> .