

Treemily

Rapport intermédiaire

# Henrik Akesson, juin 2017

Encadré par : Prof. Pier Donini

Sujet proposé par : Prof. Pier Donini

# Remerciements

# Introduction

## Contexte

Ce projet est développé dans le cadre du travail de bachelor, réalisé lors du 6ème semestre de la formation en ingénierie logicielle à la heig-vd.

Le but de ce travail est d’appliquer les concepts et processus de développement enseignés lors de la formation.

TODO

## Énoncé

Initialement proposé en 2016 par Prof. Pier Donini, ce projet a comme énoncé :

« Le but de ce projet est de permettre la création collaborative non seulement d'arbres généalogiques mais aussi d'enregistrer les évènements marquants survenants à un individu ou à un groupe d'individus (p.ex. mariages, séparations, déménagements, naissances, décès, changements de profession...).

Dans un premier temps il s'agira d'établir un schéma de bases de données permettant de stocker les individus et les relations familiales les liants ainsi que les événements qui leur sont associés. Ce schéma devra également permettre aux utilisateurs de définir des droits sur leurs données (privé, public, restreint à un groupe...).

Dans un deuxième temps, il faudra concevoir et développer une application web multi utilisateurs permettant de saisir de manière ergonomique ces données, de définir leurs droits et de pouvoir les visualiser sous la forme d'arbres intéractifs (p.ex. click sur un événement pour en obtenir la description textuelle ainsi que les photos ou vidéos associées).

Ce projet sera réalisé en Java au moyen du framework *Play!* en HTML5 et CSS3. »

## Problématique

# Préparation et déroulement

## Cahier des charges

### Utilité

Le cahier des charges proposé au début du travail visait à décrire et prévoir la manière dont le projet serait développé, ainsi qu’à définir les modes d’utilisation, les critères d’acceptation et fonctionnalités jugés essentiels pour considérer le logiciel comme accompli.

### Description du projet

Le but du projet est de fournir à des utilisateurs une plateforme pour décrire et partager leur généalogie. Le projet permettra d’une part aux utilisateurs de décrire leur arbre généalogique, les personnes impliquées et les évènements et média associés, d’autre part les détails doivent pouvoir être à visibilité variable, du privé au public en passant par la visibilité limitée.

D’un point de vue utilisateur, trois manières d’utiliser le logiciel seront possibles.

* Utilisateur non enregistré : Rechercher une personne, voir l’arbre généalogique de cette dernière, ses évènements et les détails associés dans les limites de ce qui est visible publiquement.
* Utilisateur nouveau et inconnu : Créer un arbre généalogique, y ajouter des personnes (créant au passage des profils fantômes dans le cas où une personne ajoutée n’est pas déjà enregistrée dans le logiciel), ajouter des évènements selon la visibilité souhaitée.
* Utilisateur nouveau mais dont un profil fantôme est déjà enregistré : Associer le profil au nouveau compte (l’utilisateur créé un compte, entre les informations de son profil et se voit proposer des profils qui pourraient être le sien), contribuer à l’arbre duquel il est membre, y ajouter des évènements et média.

### Critères d’acceptation

Afin de considérer le projet comme achevé, les critères suivants doivent être respectés :

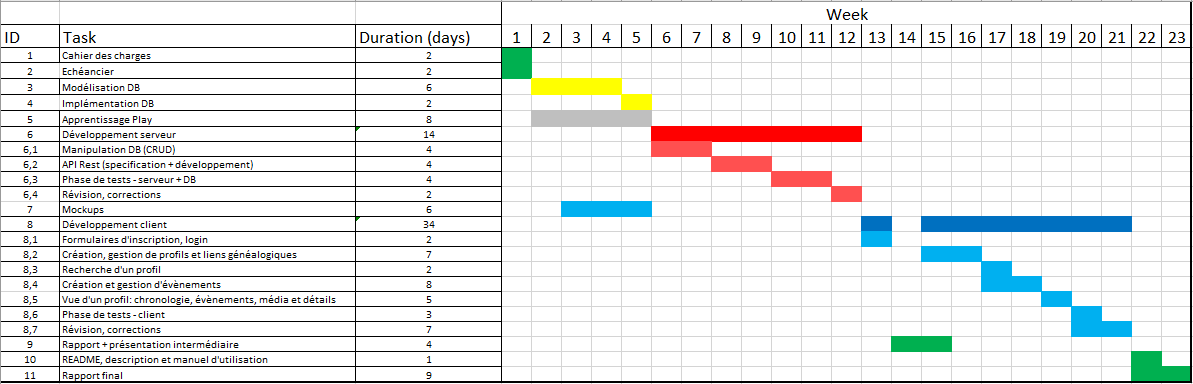
* Serveur :
  + La base de données est mise en place, la modélisation est étudiée, intuitive, propre, réutilisable et peut être développée et étendue selon de nouveaux cas d’utilisation.
  + Le serveur gère toutes les opérations CRUD associées aux cas d’utilisation et à la base de données
  + Le serveur est robuste et scalable.
  + Le serveur transmet au client toutes et seulement les données dont il a besoin selon tous les cas d’utilisation.
  + Le serveur gère les enregistrements et les connexions de manière sécurisée.
* Client :
  + Le client offre une interface graphique propre, moderne et intuitive
  + Le client permet toutes les opérations nécessaires à l’utilisation complète du logiciel.
  + L’utilisateur est tenu au courant de l’état de ses opérations (succès, erreur, nouveaux évènements qui lui sont associés…)
  + Une section d’aide permet à un utilisateur d’apprendre les possibilités du logiciel et comment les exploiter.
  + L’application cliente permet de parcourir l’arbre d’une famille et de voir une vue détaillée des membres. La vue détaillée comprend les informations générales et une chronologie des évènements.

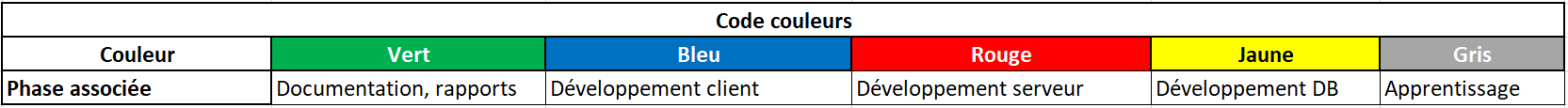
### Fonctionnalités prévues

* Créer un compte
* Créer un arbre généalogique
  + Ajouter un profil, existant au préalable ou non (date de naissance, nom complet, sexe, lieu de naissance, adresse courante, anciennes adresses, date de décès, lieu de décès, informations de contact, photo de profil)
  + Ajouter des liens entre les profils (mariage, divorce, séparation, unions civiles, enfants biologiques, adoption, PACS, partenaire, fiançailles)
* Ajouter des évènements et leur associer des média (vidéo, photo, audio, lien, document) et profils associés
  + Les évènements peuvent être d’ordre personnel (privé, limité à certains profils, public).
  + Les évènements sont associés à un ou plusieurs profils et sont visibles sur la vue détaillée de chacun après validation.
  + La définition générale d’un évènement est libre. Il peut aussi bien s’agir d’un évènement « quelconque » comme des vacances qu’un évènement important (changement de sexe, de nom, d’emploi, de lieu de vie, naissance, adoption)
  + Une modification du profil pourra être enregistré comme un évènement, permettant de conserver un historique du profil. Par exemple si un utilisateur modifie sa photo de profil, un évènement décrivant cette action avec des liens vers l’ancienne et la nouvelle photo sera enregistré.
* Modifier un arbre
* Modifier un évènement
* Modifier un profil
* Parcourir un arbre généalogique : Partir d’une vue générale d’un arbre et vue détaillée d’un membre ou d’évènements.
* Approuver ou non de faire partie d’un évènement avant d’y être affiché : Un utilisateur A peut ajouter un utilisateur B dans un évènement, mais son nom ne sera pas affiché tant qu’il ne donne pas son accord.

## Planning

L’échéancier du projet a été défini comme suit :





## Respect des consignes et du planning

À ce stade, les fonctionnalités prévues et les modes d’utilisation sont inchangés. Dans l’état actuel du logiciel, il est possible de voir l’arbre généalogique d’une personne (ses parents, ses enfants, ses relations). Il est également possible d’ajouter des personnes, relations et parents directement depuis la vue de l’arbre. La vue détaillée d’un profil est implémentée : Il est possible de voir les informations du profil, les évènements de sa vie et les médias associés.

En revanche, l’échéancier n’a pas été respecté :

* La modélisation de la base données (modèle EA, modèle relationnel et implémentation) a pris trois semaines de plus afin de garantir un modèle approprié, complet et adapté à ce que le logiciel vise à proposer aux utilisateurs.
* La rédaction du cahier des charges et de l’échéancier a pris une semaine de plus qu’initialement prévu.
* L’échéancier prévoyait d’implémenter le serveur avant de développer l’application client. Dû au fait que le serveur sert principalement à effectuer des opérations CRUD à la base de données, son implémentation a été mise en attente afin de pouvoir consacrer une partie du semestre sur le développement du client, plus compliqué et nécessitant plus de discussion avec les encadrants du travail. Le développement du client, étant toujours en cours, a également été continué lors des semaines 14 et 15 en parallèle à la rédaction de ce rapport.

## Journal de travail

* Semaine 1 (20/02/2017 – 26/02/2017)
  + Cahier des charges
  + Échéancier
* Semaine 2 (27/02/2017 – 05/03/2017)
  + Cahier des charges
  + Échéancier
  + Modèle EA, version 1
* Semaine 3 (06/03/2017 – 12/03/2017)
  + Modèle EA, version 2
* Semaine 4 (13/03/2017 – 19/03/2017)
  + Modèle EA, version 3
  + Implémentation en MySQL du modèle EA version 3
* Semaine 5 (20/03/2017 – 26/03/2017)
  + Modèle EA, version 4
  + Implémentation en MySQL du modèle EA version 4
* Semaine 6 (27/03/2017 – 02/04/2017)
  + Modèle EA, version 5
  + Implémentation en PostgreSQL du modèle EA version 5
* Semaine 7 (03/04/2017 – 09/04/2017)
  + Modèle EA, version 6
  + Implémentation en PostgreSQL du modèle EA version 6
  + Début du backend (configurations, intégration d’Hibernate et PostgreSQL à Play)
* Semaine 8 (10/04/2017 – 16/04/2017)
  + Configuration de la communication serveur – base de données
* Vacances de Pâques (17/04/2017 – 23/04/2017)
  + Points d’entrée du serveur (RESTful API)
* Semaine 9 (24/04/2017 – 30/04/2017)
  + Test de librairies graphiques pour l’application client
    - Vis.js
    - D3.js
    - Basicprimitives
  + Début du client
  + Authentification
  + Création d’un profil
* Semaine 10 (01/05/2017 – 07/05/2017)
  + Représentation graphique de l’arbre
    - Nœuds (personnes)
    - Liens (relations et parents)
* Semaine 11 (08/05/2017 – 14/05/2017)
  + Suite à une panne d’ordinateur sans backup, le développement du client a dû être recommencé
  + Représentation de l’arbre (nœuds, liens et parents)
  + Algorithme de placement automatique des personnes
  + Insertion de la photo de profil d’une personne dans son nœud
* Semaine 12 (15/05/2017 – 21/05/2017)
  + Lecture des personnes, relations et parents depuis un fichier json, représentatif des données transmises par le serveur
  + Possibilité d'ajouter un noeud depuis l'interface graphique
* Semaine 13 (22/05/2017 – 28/05/2017)
  + Algorithme de placement automatique des personnes
  + Formulaire d’authentification
  + Barre de navigation
  + Service permettant de récupérer les données relatives à un profil selon son identificateur
  + Ajout d’un bouton « Save » lorsque des nouvelles entités (liens et nœuds) sont ajoutés afin de préciser que l’enregistrement de nouvelles données n’est pas automatique
* Semaine 14 (29/05/2017 – 04/06/2017)
  + « Dialogue » permettant d’importer l’arbre généalogique d’une personne en deux clics
  + Ajout dynamique de l’arbre généalogique d’une personne
* Semaine 15 (05/06/2017 – 11/06/2017)
  + Visualisation d’un profil
    - Informations
    - Évènements
    - Médias
  + « MediaViewer », un slider permettant de parcourir les médias associés à un évènement
  + Rapport intermédiaire
* Semaine 16 (12/06/2017 – )
  + Rapport intermédiaire
  + « Nettoyage » du code

# Structure du projet

Cette partie du rapport sert à décrire en premier lieu les technologies qui sont utilisées pour développer ce projet, puis la manière dont il fonctionne et est structuré et enfin la modélisation de la base de données, dont on présentera le modèle EA ainsi que le modèle relationnel.

Il est à préciser, dû au fait que le serveur n’est pas encore concrètement utilisable depuis le client, que la communication entre les deux logiciels ainsi que la structure du serveur pourront être étendues et modifiées

## Technologies utilisées

### Client

L’application client est développée avec HTML5, CSS3 et les technologies suivantes :

#### Angular & TypeScript

Angular[[1]](#footnote-1) est un Framework développé par Google, open-source, permettant de développer des applications web. Contrairement à son prédécesseur, AngularJS, Angular (ou Angular2) est développé pour être utilisé en TypeScript[[2]](#footnote-2).

TypeScript est un langage développé par Microsoft, défini comme un sur-ensemble de JavaScript, y ajoutant les concepts de classes, interfaces, et typage. Le langage est ensuite transcompilé en JavaScript pour pouvoir être interprété par les moteurs JavaScript des navigateurs[[3]](#footnote-3). Un avantage considérable acquis grâce à la compilation du code est le fait que, contrairement à JS, l’application ne sera pas rendue tant que des erreurs figurent dans le code. De plus, le typage des variables (qui reste optionnel) permet de rendre le logiciel plus robuste et sécurisé.

Angular permet la programmation orientée objets, l’injection de dépendances, l’utilisation de directives (attributs personnalisés d’éléments du DOM), l’animation de composants et la création de classes injectables.

#### Angular-cli

Angular-cli[[4]](#footnote-4) est une interface en ligne de commande permettant de gérer un projet Angular. Il sert à compiler, créer des composants, interfaces, services, types énumérés, modules et les inclure dans les modules Angular automatiquement, servir l’application et créer des fichiers de test.

#### D3js

D3js[[5]](#footnote-5) est une librairie JavaScript qui permet de créer, d’attribuer des comportements et d’interagir avec des éléments du DOM de manière fluide. Son API liée aux éléments SVG s’avère particulièrement pratique dans ce projet, notamment pour adapter dynamiquement la vue de l’arbre généalogique.

#### Angular2 Material

Angular2 Material[[6]](#footnote-6) fournit des composants Angular prédéfinis qui respectent les spécifications de Material Design[[7]](#footnote-7) défini par Google, tels que les barres de navigation, menus, dialogues, boutons, formulaires, icônes, tableaux et bien d’autres.

### Serveur

Le serveur est développé en Java 1.8 et utilise les technologies suivantes :

#### Play Framework

Le Framework Play[[8]](#footnote-8) est un Framework open-source développé en Scala[[9]](#footnote-9), mais compatible avec d’autres langages compilés en Bytecode[[10]](#footnote-10).

Conçu pour être utilisé dans le cadre du développement d’applications web, Play a la particularité d’être :

* RESTful[[11]](#footnote-11)
* Stateless[[12]](#footnote-12)
* Convention plutôt que configuration[[13]](#footnote-13)

Ces particularités le rendent adapté à ce projet, puisque le serveur ne répondra qu’à des évènements enclenchés par l’application client.

Lorsqu’une requête http est reçue sur le port d’écoute de l’application, un fichier de configuration lui permet de la rediriger vers un contrôleur adapté qui se chargera de traiter la requête puis de formuler une réponse.

#### SBT

SBT[[14]](#footnote-14), acronyme de Scala Build Tool, est un outil permettant de compiler du code Scala ou Java, gérer des dépendances et télécharger les librairies nécessaires. Similaire à Maven, SBT est le moteur de production préconisé par Play. Les fichiers de configuration de SBT sont écrits en Scala plutôt qu’en XML pour Maven.

#### Hibernate

L’ORM[[15]](#footnote-15) utilisé pour ce projet est Hibernate[[16]](#footnote-16), un framework implémentant la spécification JPA[[17]](#footnote-17) permettant de lier des classes Java à une base de données relationnelle. Ayant son propre langage de requêtes, HQL, basé sur SQL, Hibernate gère les communications à la base de données. Dans le cadre de Treemily, la base de données est implémentée séparément ; le schéma doit donc être importé et interprété par le framework, contrairement à d’autres cas d’utilisation où les schémas et tables sont générées depuis des POJO[[18]](#footnote-18) codés en Java.

### Base de données

La base de données utilisée est une base de données relationnelle SQL.

#### PostgreSQL

PostgreSQL[[19]](#footnote-19) est un système de base de données relationnelle, offrant une syntaxe similaire mais pas équivalente au langage de requête SQL. Il propose de nombreuses fonctionnalités pratiques, telles que les clés étrangères qui réfèrent automatiquement aux clés primaires des tables associées.

### Rédaction, codage, modélisation

#### Modèle EA

Le modèle entités-associations a été dessiné avec StarUML[[20]](#footnote-20), un logiciel de modélisation

#### Serveur et base de données

Le serveur est développé avec l’IDE Intellij de Jetbrains[[21]](#footnote-21), proposant un support pour de nombreux frameworks dont Play et Hibernate. On peut également accéder à la base de données depuis le logiciel, aussi bien pour parcourir les tables que pour effectuer des requêtes.

#### Client

Le client est développé avec Visual Studio Code[[22]](#footnote-22), un éditeur de code open-source développé par Microsoft qui a de nombreux plugins et est particulièrement adapté au développement web.

#### Rapport, cahier des charges et échéancier

Le rapport, le cahier des charges et l’échéancier sont rédigés avec la suite Microsoft Office.

#### Github

Github[[23]](#footnote-23) et Git sont utilisés pour gérer le versioning et le backup du code source du projet.

## Modélisation

### C:\Users\Henrik\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Main.pngModèle Entité-Association

Figure 1- Modèle EA de Treemily

### Description des choix

#### TimedEntity

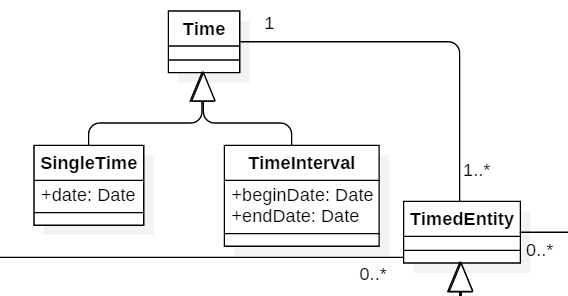


Figure 2- TimedEntity

La base de ce modèle est l’entité **TimedEntity**. Les relations, personnes et évènements sont des généralisations de cette entité.

TimedEntity décrit un objet qui est délimité dans le temps, soit par une date fixe, **SingleTime** (qui peut indiquer un jour unique ou un intervalle de temps dont la date de fin n’est pas connue).

#### Profils, relations et parents

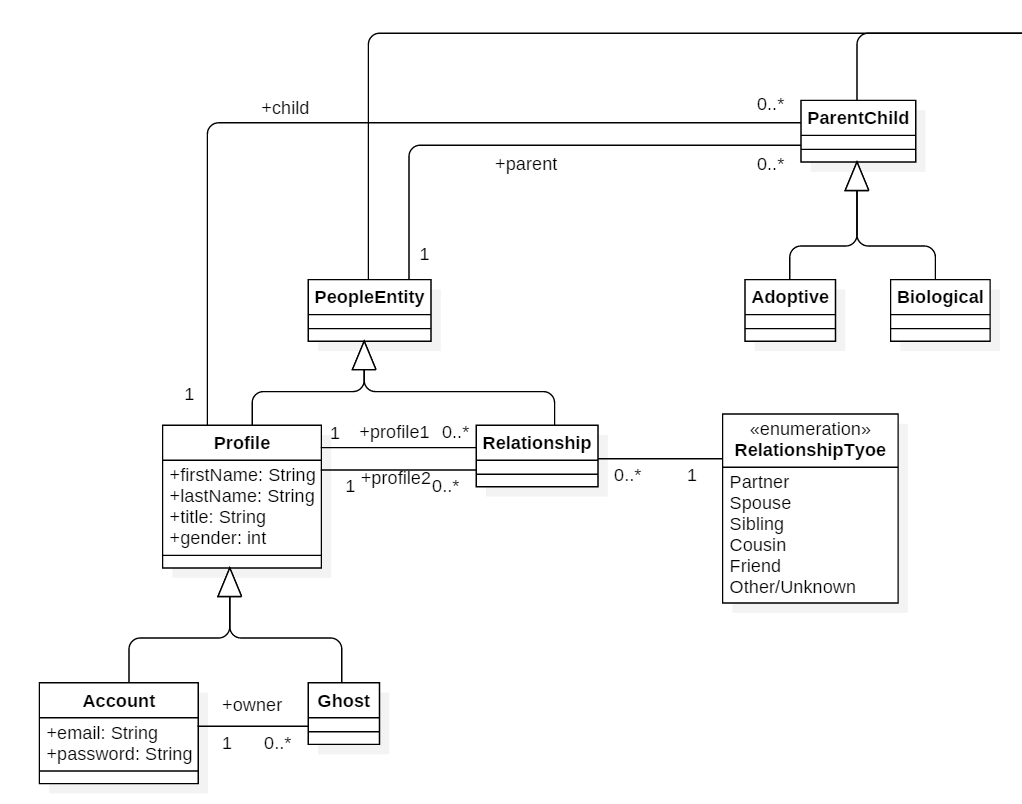


Figure 3- Relations, profils, comptes et parents

Les profils et relations, héritent d’une entité nommée **PeopleEntity**,dû au fait que l’entité **ParentChild** peut être liée soit à une personne, soit à une relation entre deux personnes. Ce choix a été fait pour que l’utilisateur puisse préciser s’il le souhaite un ou deux parents pour une personne, que ce soit pour des parents biologiques ou adoptifs.

Une relation lie deux personnes. Une personne peut être dans aucune ou plusieurs relations.

Un **ParentChild** n’admet qu’un seul parent, que ce soit une relation ou une personne seule.

Il y a deux sous-classes de **Profile**:

Un **Account** représente un utilisateur enregistré auquel un profil est lié. Un **Account** peut posséder des **Ghost**s, qui représentent des profils qui ne sont pas associés à un compte, qu’un utilisateur différent a créé. Jusqu’à ce qu’un compte soit créé pour récupérer ce profil, son **owner** a tous les droits sur ce profil. Un **Ghost** ne peut en revanche pas commenter sur un évènement ou un média.

#### Posts, Events, Media

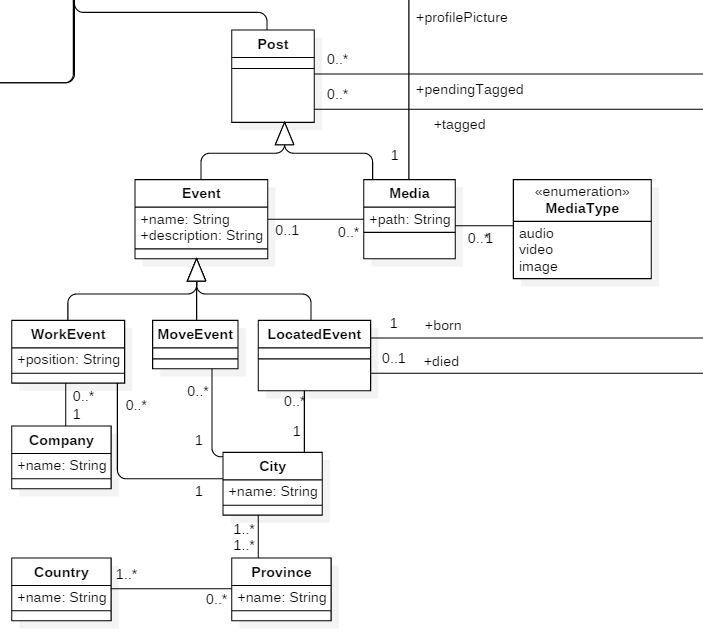


Figure 4- Posts, Events, Media, Location

Le choix de faire hériter de **Post** les entités **Event** et **Media** vient de la nécessité de pouvoir associer des commentaires, profils et une visibilité aux deux.

**Tagged** et **pendingTagged** sont des associations à des profils. Elles représentent les personnes qui sont associées à un **Post**. Lors de l’ajout d’une personne associée, elle est automatiquement enregistrée comme une relation **pendingTagged**, jusqu’à ce que la personne en question (ou son propriétaire) accepte d’y être associé avant d’être migrée vers une association **tagged**.

Le propriétaire d’un évènement ou média est hérité de l’association **Profile – TimedEntity** nommée **owner**.

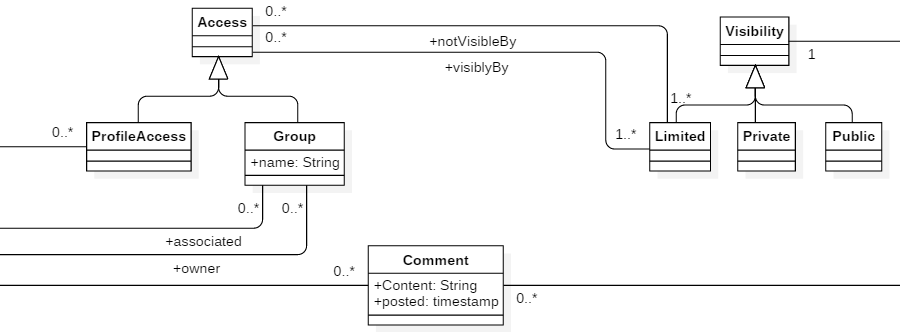
Un média est lié à un évènement ou est une photo de profil. Des médias ne peuvent pas être créés indépendamment de ces deux cas. Trois types de média sont reconnus : audio, image ou vidéo.

Un emplacement (**Location**) est décrit par une ville (**City**), une région ou **Province**, et un pays (**Country**). Dû à la possibilité d’avoir des profils datant d’époques où les pays et régions avaient des noms différents ou n’étaient pas dans le même pays (par exemple : Leipzig, qui se trouve actuellement en Saxonie, RFA se trouvait précédemment en RDA sans pour autant être une ville différente).

Il y a quatre sortes d’évènements :

* **Event**: Un évènement « classique », auquel peut être associé des médias, des commentaires, des personnes. Possède un nom, une description, un propriétaire et une date ou un intervalle de temps.
* **LocatedEvent**: Un évènement situé. En plus des attributs et associations héritées d’**Event**, est lié à un emplacement. De plus, chaque **Profile** a une association vers un **LocatedEvent**, nommé « **born** » et peut ou non avoir une autre relation nommée « **died** ».
* **MoveEvent**: Un MoveEvent représente un déménagement, et possède une association à un emplacement, la nouvelle ville de résidence de la personne.
* **WorkEvent**: Un WorkEvent représente un emploi qu’a une personne. Il possède un attribut « position » et est associé à une entreprise (**Company**). Une entreprise peut évidemment être associée à plusieurs WorkEvents.

#### Vibilité, accès, commentaires et groupes



Chaque **TimedEntity** a une visibilité. Ainsi, des profils, évènements, média, relations et liens de type parent peuvent avoir une visibilité limitée selon le choix du créateur de l’entité en question.

Il y a trois sortes de visibilité définies dans Treemily :

* Une visibilité publique signifie que tout utilisateur, enregistré ou non, a accès au contenu.
* Une visibilité privée signifie que seulement le créateur et les personnes associées ont accès à l’entité. Exemples :
  + Une relation avec une visibilité privée sera visible par les deux membres de la relation.
  + Un évènement avec une visibilité privée sera visible par son créateur et les personnes associées (aussi bien **pendingTagged** que **tagged**).
* Une visibilité limitée signifie qu’en plus de la visibilité de base (créateur et associés), celle-ci est étendue à une entité de type **Access**. Un accès peut être accordé ou empêché (**visibleBy** et **notVisibleBy**) à des personnes spécifiées individuellement ou à des groupes.Un groupe est un ensemble de personnes, ayant un propriétaire et un nom, qui est sauvegardé afin de pouvoir avec facilité le réutiliser à plusieurs reprises.

De manière générale, un **TimedEntity** à visibilité limitée sera visible par :

Ces formules garantissent que les personnes **tagged** et **pendingTagged** (s’il s’agit d’un évènement) et le propriétaire du **TimedEntity** auront toujours le droit de le voir. Dans le cas où l’ensemble **visibleBy** est défini, l’ensemble **notVisibleBy** lui sera soustrait. Ainsi, un utilisateur pourrait inclure un groupe tout en excluant un sous-ensemble de ce groupe sans produire d’erreur logique.

Sachant qu’un média peut avoir sa propre visibilité, la priorité des visibilités d’un média est gérée comme suit : Si un média a une visibilité plus limitée que l’évènement le contenant, il conserve sa visibilité. En revanche, si un média est défini comme ayant une visibilité plus large que son évènement, la visibilité du média sera « écrasée » par celle de son évènement.

Un média ne peut pas avoir des profils **tagged** qui ne sont pas **tagged** à l’évènement le contenant. L’utilisateur aura donc le choix entre étendre les **tagged** de l’évènement en y ajoutant ceux du média en question ou retirer ces mentions du média.

Une contrainte d’intégrité à prendre en considération lors de l’implémentation est que toute personne d’une relation ne peut pas être exclue de la visibilité de cette relation.

Enfin, un **Post** peut avoir des commentaires, qui sont associés à un **Profile**. Un commentaire a un contenu et une estampille de quand il a été soumis.

# Implémentation

## Base de données

L’implémentation de la base de données est faite depuis un ensemble de requêtes PostgreSQL. Les tables créées sont équivalentes au modèle EA, en ajoutant les associations n-aires dans leur propre table :

## Serveur

## Client

### Classes

### Hiérarchie

### Visualisation

#### Fonctionnement général des composants Angular2

#### Composants de l’application

#### Composants & SVG

#### Algorithme de placement des nœuds

#### Lazy-loading de l’arbre généalogique

#### Visualisation d’un profil et autres dialogues

1. https://angular.io/ [↑](#footnote-ref-1)
2. https://www.typescriptlang.org/ [↑](#footnote-ref-2)
3. https://fr.wikipedia.org/wiki/TypeScript [↑](#footnote-ref-3)
4. https://github.com/angular/angular-cli [↑](#footnote-ref-4)
5. https://d3js.org/ [↑](#footnote-ref-5)
6. https://material.angular.io/ [↑](#footnote-ref-6)
7. https://material.io/guidelines/ [↑](#footnote-ref-7)
8. https://www.playframework.com/ [↑](#footnote-ref-8)
9. https://www.scala-lang.org/ [↑](#footnote-ref-9)
10. https://en.wikipedia.org/wiki/Bytecode [↑](#footnote-ref-10)
11. https://en.wikipedia.org/wiki/Representational\_state\_transfer [↑](#footnote-ref-11)
12. https://en.wikipedia.org/wiki/Stateless\_protocol [↑](#footnote-ref-12)
13. https://en.wikipedia.org/wiki/Convention\_over\_configuration [↑](#footnote-ref-13)
14. http://www.scala-sbt.org/ [↑](#footnote-ref-14)
15. https://en.wikipedia.org/wiki/Object-relational\_mapping [↑](#footnote-ref-15)
16. http://hibernate.org/ [↑](#footnote-ref-16)
17. https://en.wikipedia.org/wiki/Java\_Persistence\_API [↑](#footnote-ref-17)
18. https://en.wikipedia.org/wiki/Plain\_old\_Java\_object [↑](#footnote-ref-18)
19. https://www.postgresql.org/ [↑](#footnote-ref-19)
20. http://staruml.io/ [↑](#footnote-ref-20)
21. https://www.jetbrains.com/idea/ [↑](#footnote-ref-21)
22. https://code.visualstudio.com/ [↑](#footnote-ref-22)
23. https://github.com/ [↑](#footnote-ref-23)