2019

# Task Manager Project

RELAZIONE PROGETTO BASI DI DATI E WEB

LUCA FAGGION

# INDICE

# DATABASE

## ANALISI DEI REQUSITI

Il database deve gestire un'applicazione di gestione dei compiti (Task) questa e la linea guida che è stata fornita per la realizzazione:

> Il progetto descrive un sistema di gestione dei compiti (Tasks) di un'azienda nel quale ad un utente o a un Gruppo di utenti viene assegnato un o più compiti da svolgere. Per ogni compito vengono salvati la data di inizio, fine e scadenza cosi come il nome e la descrizione. Un compito può contenere oltre a queste informazioni anche una lista di altri compiti (che non contenga il compito alla quale viene associata) che deve essere completata prima di completare il compito che la contiene. Ogni Compito fa parte di un solo progetto e un progetto può avere più compiti che devono essere completati prima di dichiarare il progetto concluso. Per ogni progetto vengono anche salvati dati come nome, descrizione e le relative date di creazione, completamento e scadenza. Un utente quale e stata assegnata un compito può decidere di condividere ad un altro utente quel compito che gli è stato assegnato per potersi far aiutare e per ogni Utente vengono salvati i dati relativi a nome, cognome, nome utente, password e data di nascita.

Dalle linee guida è stata estratta la seguente tabella iniziale delle **Entità, Attributi** e **Relazioni:**

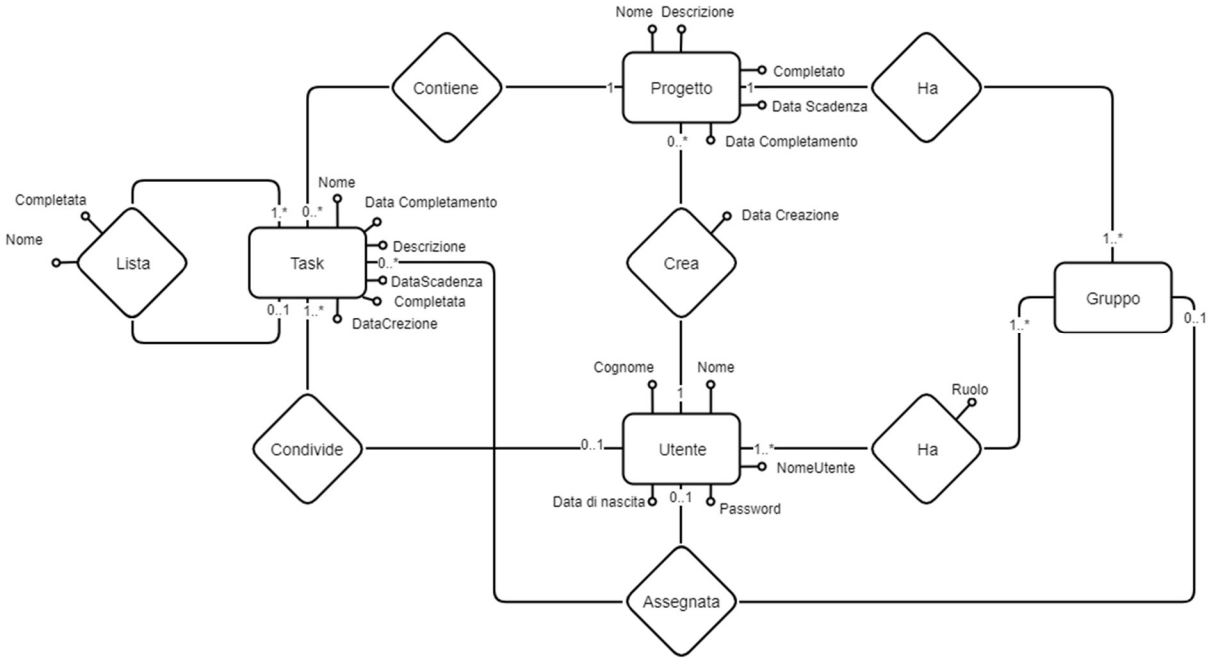
| Entita/Relazione | Attributi | Relazione |
|---|---|---|
| Utente (entità) | • Nome<br>• Cognome<br>• DataNascita<br>• NomeUtente **PK**<br>• Password | |
| Progetto (entità) | • Nome<br>• Descrizione<br>• Completato<br>• DataCompletamento<br>• DataCreazione<br>• DataScadenza | Progetto → Utente [Crea]<br>Progetto → Gruppo [Ha] |
| Gruppo (entità) | • Nome | Gruppo → Progetto [Ha] |
| Task (entità) | • Nome<br>• Descrizione<br>• Completata<br>• DataCreazione<br>• DataScadenza<br>• Progetto **FK** | Task → Progetto [Contiene] |
| Assegnamento (relazione) | • Task **FK**<br>• Utente **FK**<br>• Gruppo **FK** | Task → Utente / Gruppo [Ha Assignata] |
| Condivisione (relazione) | • Task **FK**<br>• Utente **FK** | Task → Utente [Condivide] |

| | | |
|---|---|---|
| Lista di task (entità) | • Task **FK**<br>• Task Riferimento **FK**<br>• Completata | Task→Task [Contiene] |

## PROGETTAZIONE CONCETTUALE

### SCHEMA ENTITA'-RELAZIONE

Dalla tabella delle Entità, Attributi e Relazioni e dai requisiti del progetto è stata creata uno schema concettuale di **Entità-Relazione** per realizzare il **Modello dei Dati:**



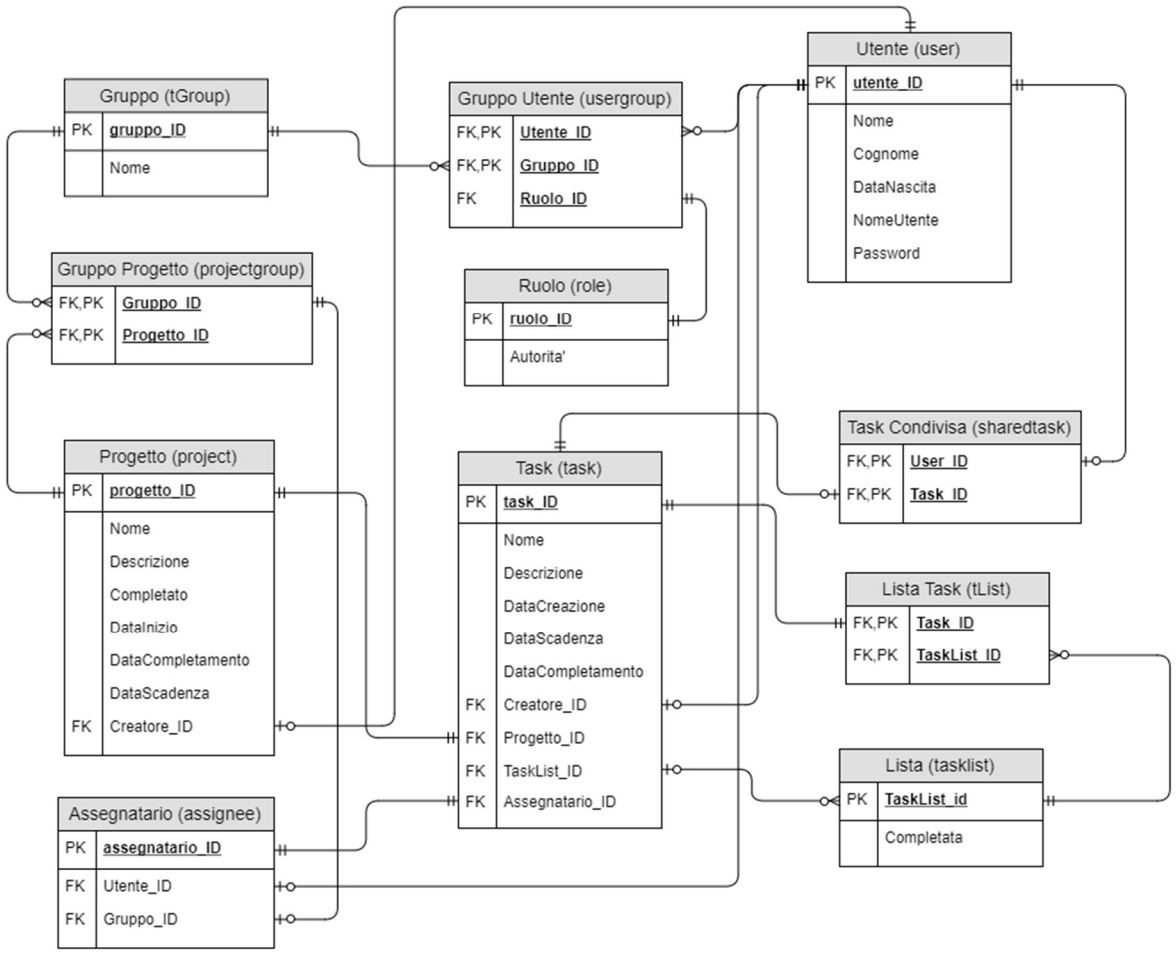

Sono stati aggiunte altre relazioni necessarie al funzionamento del database che non erano state specificate nella linea guida del progetto. L'**Entità Gruppo** ha la necessita di una relazione con **Utente** con un attributo **Ruolo (che potrà poi essere successivamente esteso per diventare una entità)** per stabilire le autorizzazioni degli utenti che verranno associati ai progetti e per la gestione delle **Task.** Per la realizzazione dello schema concettuale è stata utilizzata una **Strategia Mista** si sono da prima individuati le entità principali coinvolte per poi per raffinamenti successivi sono state integrate tra di loro e espanse con l'aggiunta di relazioni tra entità, la trasformazione di relazioni in entità, l'aggregazione di attributi in entità (come nel caso di Lista) e altre **Primitive di trasformazione** delle strategie di **Bottom-up** e **Top-Down**. Il prodotto finale ha tutte le **qualità** di correttezza, completezza, leggibilità e compattezza di uno schema concettuale.

## PROGETTAZIONE LOGICA

Per la realizzazione effettiva del DB e del progetto si è dovuto **ristrutturare** e **traduzione** dello **schema E-R** in uno schema più completo chiamato **Schema Logico** basato sul **Modello Relazionale**

Durante la ristrutturazione si sono dovute aggiungere alcune entità e relazioni:

- Ruolo(**role)**: questa entità è stata creata dalla conversione dell'attributo ruolo sulla relazione dello schema concettuale che legava **gruppo** a **utente.**
- GruppoUtente(**usergroup**): questa relazione lega un utente a uno specifico gruppo assegnandogli anche un ruolo, facendo attenzione ai **Vincoli di Integrità** della tabella possiamo notare che essendo **utente** e **gruppo** sia **Chiavi Primarie** che **Chiavi Esterne (Foreign keys)** mentre ruolo e solo chiave esterna si possono associare ad un utente più gruppi con ruoli identici, requisito fondamentale per gestire le autorizzazioni degli utenti che useranno l'applicazione.
- GruppoProgetto(**projectgroup**): questa relazione è stata aggiunta per ovviare a un problema sorto durante la ristrutturazione ovvero non c'era nessuna tabella che legasse gli utenti ad uno specifico progetto (se non chi avesse creato quel progetto) questa relazione lega un gruppo di utenti a uno specifico progetto mantenendo le autorità di ogni utente all'interno del loro gruppo. Questa tabella verrà utilizzata per recuperare le tasks che ogni utente eredita della partecipazione al gruppo.
- L'entità **Lista** e stata scorporata in un'entità (**Lista**) e una relazione (**Lista Task**) per motivi che verranno spiegati nella sezione **2. Domain Classes**



## CREAZIONE DATABASE MYSQL

E qui sotto e riportato lo script per la creazione delle tabelle descritte nello schema logico (che è possibile trovare nel percorso **./private/domains/sql/tables.sql**)

Da notare che sono stati aggiungi dei **Constraints** sulle **chiavi esterne** che sono innescati (triggered) quando le chiavi che sono referenziate vengono eliminate, eliminando a sua volta la riga nella tabella dove e stato specificato il constraint.

**tables.sql**

```sql
CREATE TABLE IF NOT EXISTS `User` (
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `Nome` VARCHAR(32) NOT NULL,
    `Cognome` VARCHAR(32) NOT NULL,
    `DataNascita` DATE NOT NULL,
    `NomeUtente` VARCHAR(32) NOT NULL,
    Password CHAR(60) NOT NULL
);

CREATE TABLE IF NOT EXISTS `Role`(
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `Authority` VARCHAR(32) NOT NULL
);

CREATE TABLE IF NOT EXISTS `UserRole`(
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `Userid` int NOT NULL,
    `Roleid` int NOT NULL,
    FOREIGN KEY (`Userid`) REFERENCES `User`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`Roleid`) REFERENCES `Role`(`id`) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS `Project`(
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `Nome` VARCHAR(32) NOT NULL,
    `Descrizione` TEXT NOT NULL,
    `Completato` BOOL NOT NULL,
    `DataInizio` DATE NOT NULL,
    `DataCompletamento` DATE NULL,
    `DataScadenza` DATE NOT NULL,
    `Creatore` int,
    FOREIGN KEY (Creatore) REFERENCES `User`(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS `tGroup`(
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `Nome` VARCHAR(32) NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS `GroupRole`(
    `id` int NOT NULL AUTO_INCREMENT,
    `Userid` int NOT NULL,
    `Groupid` int NOT NULL,
    `Roleid` int NOT NULL,
    FOREIGN KEY (`Userid`) REFERENCES `User`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`Groupid`) REFERENCES `tGroup`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`Roleid`) REFERENCES Role(`id`) ON DELETE CASCADE,
    PRIMARY KEY (`Userid`, `Groupid`),
    UNIQUE KEY (`id`)
);

CREATE TABLE IF NOT EXISTS `ProjectGroup`(
    `id` int NOT NULL AUTO_INCREMENT,
    `tGroup` int NOT NULL,
    `Project` int NOT NULL,
    FOREIGN KEY (`tGroup`) REFERENCES `tGroup`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`Project`) REFERENCES `Project`(`id`) ON DELETE CASCADE,
    PRIMARY KEY (`tGroup`, `Project`),
    UNIQUE KEY (`id`)
);


CREATE TABLE IF NOT EXISTS `Assignee`(
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `User` int,
    `tGroup` int,
    FOREIGN KEY (`User`) REFERENCES `User`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`tGroup`) REFERENCES `tGroup`(`id`) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS `Task`(
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `Nome` VARCHAR(32) NOT NULL,
    `Descrizione` TEXT NOT NULL,
    `DataCreazione` DATE NOT NULL,
    `DataScadenza` DATE NOT NULL,
    `DataCompletamento` DATE NULL,
    `Completata` BOOL NOT NULL,
    `User` int NOT NULL,
    `Project` int NOT NULL,
    `Assignee` int DEFAULT NULL,
    FOREIGN KEY (`User`) REFERENCES `User`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`Project`) REFERENCES `Project`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`Assignee`) REFERENCES `Assignee`(`id`) ON DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS `TaskList`(
```

```
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `Task` int NOT NULL,
    `Completata` BOOL NOT NULL,
    FOREIGN KEY (`Task`) REFERENCES `Task`(`id`) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS `SharedTask`(
    `id` int NOT NULL AUTO_INCREMENT,
    `User` int NOT NULL,
    `Task` int NOT NULL,
    PRIMARY KEY (`User`, `Task`),
    UNIQUE KEY (`id`),
    FOREIGN KEY (`User`) REFERENCES `User`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`Task`) REFERENCES `Task`(`id`) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS `tList`(
    `id` int NOT NULL AUTO_INCREMENT PRIMARY KEY,
    `Task` int NOT NULL,
    `TaskList` int NOT NULL,
    PRIMARY KEY (`Task`, `TaskList`),
    UNIQUE KEY (`id`),
    FOREIGN KEY (`Task`) REFERENCES `Task`(`id`) ON DELETE CASCADE,
    FOREIGN KEY (`TaskList`) REFERENCES `TaskList`(`id`) ON DELETE CASCADE
);
```

## SQL QUERIES

In questo paragrafo verranno elencante e discusse tutte le **SQL queries** utilizzate per la realizzazione del progetto.

tutte le queries banali ovvero del tipo (`SELECT * FROM tabella WHERE id=1`) sono state omesse per ovvi motivi, sono anche state omesse anche le queries di inserimento dei dati (`INSERT INTO …`) siccome sono gestite internamente dalla classe **Domain** descritta nella sezione **2. Domain Classes**

**le query riportate qui sotto avranno diciture del tipo (:id, :variabile etc..) ovvero una parola con i due punti (:) anteposti che stanno a significare che sono le variabili della query, questo e sempre spiegato più in dettaglio nella sezione 2.Domain Classes.**

```
-- Seleziona gli utenti di un progetto selezionando anche il nome il nome e ruolo nel gruppo/i a cui partecipano in un determinato progetto
SELECT u.*,r.Authority,g.Nome as GroupName
FROM User as u, Role as r, ProjectGroup as pg,GroupRole as gr, Project as p, tGroup as g
WHERE p.id = :projectid
AND p.id = pg.Project
AND pg.tGroup = gr.Groupid
AND r.id = gr.Roleid
```

```
AND g.id = pg.tGroup
AND gr.Userid = u.id GROUP BY u.id;
```

```
-- Seleziona una task e l'utente con cui e condivisa e l'id nella tabella
SharedTask se la task non ha condivisione l'attributo condivisore e messo a
null cosi come l'attributo SharedTask
```

```
SELECT t.*, IF(st.id=null,null,u.id) as Condivisore, IFNULL(st.id,null) as
SharedTask
FROM Task as t LEFT JOIN SharedTask as st ON t.id = st.Task
LEFT JOIN User as u ON u.id = st.User
WHERE t.id= :taskid;
```

Da notare l'uso di **LEFT JOIN** invece di **JOIN** normali questo perché vogliamo ottenere sempre tutte le task ma controllare quale di queste e stata condivisa nella tabella **SharedTask,** se avessimo usato normali JOIN non avremmo ritornato tutte le task ma solo quelle che erano state condivise

```
-- Seleziona i gruppi di un progetto
SELECT g.* FROM tGroup as g, Project as p, ProjectGroup as pg
WHERE p.id = pg.Project
AND pg.tGroup = g.id
AND p.id= :projectid;
```

```
-- Seleziona ogni gruppo e per ogni gruppo il numero di utenti e il numero di
progetti
SELECT g.*,(
        SELECT COUNT(gr.id) FROM
        tGroup as g2 LEFT JOIN GroupRole as gr ON g2.id = gr.Groupid
        WHERE g2.id = g.id GROUP BY g2.id
    ) as users,
    (
        SELECT COUNT(p.id) FROM
        tGroup as g1 LEFT JOIN ProjectGroup as pg ON g1.id = pg.tGroup
        LEFT JOIN Project as p ON p.id = pg.Project
        WHERE g1.id = g.id GROUP BY g1.id
    ) as projects
    FROM tGroup as g;
```

In questa query sono state usate due **Sub-Queries** per ritornare il numero di utenti di un gruppo e il numero di progetti.

```
-- Seleziona gli utenti di un gruppo e il loro ruolo
SELECT u.*,r.Authority
FROM User as u, Role as r, GroupRole as gr, tGroup as g
WHERE g.id = :groupid
```

```
AND r.id = gr.Roleid
AND gr.Userid = u.id
AND gr.Groupid = g.id;
```

```
-- Seleziona i progetti di un gruppo
SELECT p.* FROM Project as p, projectgroup as pg
WHERE pg.Project = p.id AND pg.tGroup = :groupid;
```

```
-- Seleziona il ruolo di uno specifico utente in uno specifico gruppo
SELECT r.* FROM Role as r, GroupRole as gr, tGroup as g
WHERE r.id = gr.Roleid
AND g.id = gr.Groupid
AND g.id = :groupid
AND gr.Userid = :userid;
```

```
-- Seleziona i gruppi di un progetto
SELECT g.* FROM tGroup as g, Project as p, ProjectGroup as pg
WHERE p.id = pg.Project
AND pg.tGroup = g.id
AND p.id= :projectid;
```

```
-- Seleziona tutte le task e conta quante task sono state associate
SELECT t.*, (
      SELECT COUNT(*) FROM tasklist as tl, tList as l
      WHERE l.TaskList = tl.id
      AND tl.Task = t.id
      ) as TaskListCount
      FROM task as t;
```

```
-- Seleziona tutti gli utenti e il numero di progetti a cui partecipano
SELECT u.*,(
    SELECT COUNT( DISTINCT p.id)
    FROM User as u1,Project as p,ProjectGroup as pg, GroupRole as gr
    WHERE p.id = pg.Project AND gr.Groupid = pg.tGroup AND u.id = gr.Userid
AND u1.id = u.id
    ) as projects FROM User as u
```

Notare che `DISTINCT p.id` è equivalente a `GROUP BY p.id` posto alla fine della query.

```sql
-- Seleziona tutti i progetti di uno specifico utente
SELECT DISTINCT p.* FROM User as u,Project as p,ProjectGroup as pg, GroupRole
as gr
WHERE p.id = pg.Project
AND gr.Groupid = pg.tGroup AND u.id = gr.Userid AND u.id= :userid;
-- Seleziona tutte le task di uno specifico utente, la query ha 2 OR siccome
ci sono 3 modi per cui un utente può essere assegnato a una task gli può
essere stata assegnata oppure e stata assegnata ad un gruppo del quale lui fa
parte oppure gli può essere stata condivisa
SELECT t.*, IFNULL(st.id,0) as Condivisa
FROM User as u, Assignee as a,Grouprole as gr, tGroup as g,
Task as t LEFT JOIN SharedTask as st ON t.id = st.Task
WHERE u.id = :userid
AND (
(a.User = u.id AND t.Assignee = a.id)
OR (t.Assignee = a.id
      AND u.id = gr.Userid
      AND a.tGroup = g.id
      AND gr.Groupid = g.id)
OR (st.User = u.id AND t.id = st.Task)
) GROUP BY t.id;
```

la query ha 2 **OR** siccome ci sono 3 modi per cui ad un utente può essere associato ad una task gli può essere semplicemente stata assegnata oppure e stata assegnata ad un gruppo del quale lui fa parte oppure gli può essere stata condivisa

---

```sql
-- Seleziona gli utenti di uno specifico progetto
SELECT DISTINCT u.*
FROM User as u, projectgroup as pg, Project as p, tGroup as g, GroupRole as gr
WHERE p.id = pg.Project
AND pg.tGroup = g.id
AND gr.Userid = u.id
AND gr.Groupid = g.id
AND p.id= :projectid;
```

---

```sql
-- Seleziona i gruppi di uno specifico utente
SELECT g.* FROM User as u, tGroup as g, GroupRole as gr
WHERE gr.Userid = u.id
AND gr.Groupid = g.id
AND u.id = :userid;
```

---

# APPLICAZIONE

## SPECIFICHE PROGETTO

Il progetto e stato realizzato utilizzando **PHP,** l'interfaccia **WEB** e stata scritta in **HTML5** e **Sass** (compilato in **CSS**) e **Javascript.**

Il progetto e stato sviluppato con il paradigma di programmazione **MVC (Model View Controller)** questo ha consentito una estrema flessibilità e modularità del codice siccome separa la **business logic** dall'**interfaccia grafica** e a sua volta dalla rappresentazione dei dati nel **DBSM.**

Sono state utilizzate le seguenti librerie per uno sviluppo più coerente e anche per il rispetto degli standard WEB:

- **Bootstrap:** libreria (Sass) sviluppata da twitter per costruire interfacce front-end mobile-friedly e responsive. (https://getbootstrap.com/)
- **Fontaweasome:** liberia di icone vettoriali (SVG) accessibili da Sass/css e da html tramite l'attributo class. (https://fontawesome.com/)
- **JQuery:** libreria multipurpose javascript, utilizzata ampiamente nel progetto per realizzare la comunicazione tra back-end e front-end (https://jquery.com/)
- **I18n:** libreria PHP per la realizzazione della internazionalizzazione del progetto permette di specificare file chiave-valore per ogni lingua e di cambiare lingua utilizzando i cookies o i parametri GET. (https://github.com/Philipp15b/php-i18n)
- **Scssphp:** compilatore di Sass in CSS usato per compilare bootstrap e tutti i file Sass scritti per l'applicazione, usato in concomitanza dei file .htacess per accedere a file CSS compilati on demand (https://github.com/leafo/scssphp)

Il progetto fa uso quindi di uno **stack WEB** di tipo **WAMP (Windows | Apache | MySQL | PHP) .**

### FILES DI CONFIGURAZIONE DI APACHE - .HTACCESS

**Permessi di accesso a cartelle**

Sono stati usati per restringere l'accesso solo a determinate cartelle del Server WEB e solo a certi documenti come e possibile vedere nel file **.htaccess principale**

```
<Files *.*>
    Order Deny,Allow
    Deny from all
</Files>

<Files ~ "\.(php|js|sql|css|svg|eot|ttf|woff|woff2|ico)$">
    Order Allow,Deny
    Allow from all
</Files>
```

Mentre in tutte le cartelle in cui non si volesse che utenti esterni potessero visualizzare i documenti (come per esempio la cartella **private** che è dove risiede il core del progetto) e stato utilizzato un file .htaccess con questa configurazione

```
Deny From All
```

**Regole di Riscrittura URL (mod_rewrite)**

Sono state anche utilizzate le regole di riscrittura degli URL per un uso più semplificato dell'applicazione da parte degli utenti

```
#css rewrite rule
RewriteRule ^stylesheets/(.*)\.css$ /stylesheets/scss.php/$1.scss
[QSA,R=301,L]

#base redirect rule /
RewriteCond %{HTTP_HOST} !^m\.
RewriteRule ^$ controllers/project/index.php?action=index [QSA, L]

# controller redirect rule
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.+)\/([^\?]+)$ controllers/$1/index.php?action=$2 [QSA, L]
```

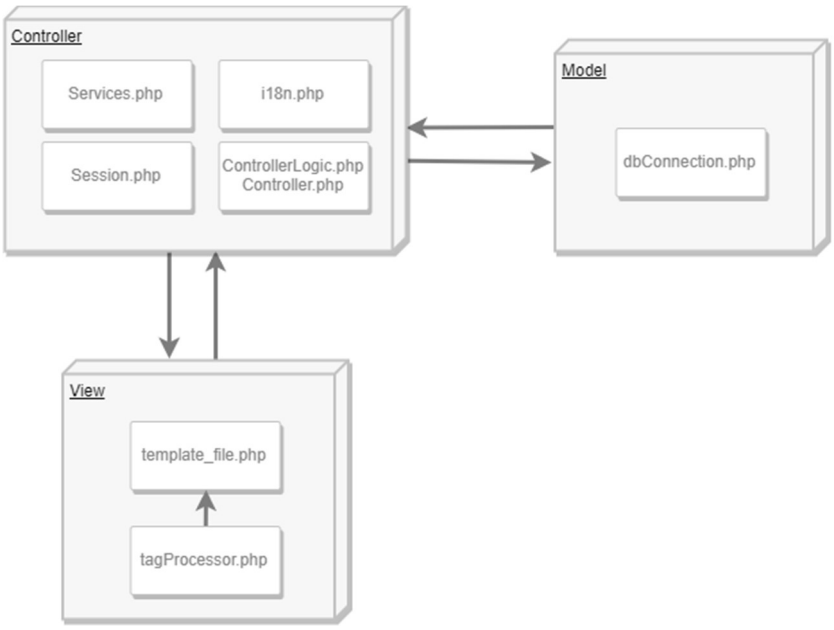
`#css rewrite rule` ogni volta che una pagina cercherà di accedere a file css verrà eseguito lo script della libreria **scssphp** per compilare (se necessario) il file **Sass** con lo stesso nome

`#base redirect rule` questa regola riscrive l'URL base del tipo http://localhost:8014/ in http://localhost:8014/controllers/project/index.php?action=index facendolo quindi puntare a un controller valido che può essere cambiato in questo caso si e deciso di visualizzare come pagina principale il controller che lista tutti i progetti.

`#controller redirect rule` questa regola invece e la regola principale per la navigazione del sito siccome riscrive gli URL del tipo http://localhost:8014/nomecontroller/azione?=queryurl che e molto semplice da ricordare e da usare e nascondendo anche l'estensione in http://localhost:8014/controllers/nomecontroller/index.php?action=azione &queryurl che e molto più lungo e difficile da usare.

## STRUTTURA PROGETTO

Questo è uno schema generale di come è strutturato il progetto e delle classi coinvolte, il progetto e strutturate interamente sul pattern **MVC**:




Nello schema sono riportati i principali script che compongono il **framework** della applicazione e quale il loro campo di operazione.

La parte **Model** e gestito dallo script dbConnection.php che mette a disposizione le classi per la comunicazione con il **DBMS** che verranno utilizzate nei servizi e nei controllers per ricevere o modificare dati presenti del database.

La parte **Controller** e composta da più script ma quelli principali sono ControllerLogic.php che predispone le classi basi dalle quali si possono creare e utilizzare i controllers e Controller.php questo script inizializza e costruisce il controller. Gli script i18n.php e Session.php mettono a disposizione rispettivamente la internalizzazione e una classe **wrapper** per gestire i cookies (fondamentali per gestire le sessioni e i dati salvati per ciascun utente).

La parte **View** e gestita dal file template_file.php che utilizza le classi messe a disposizione da tagProcessor.php per gestire i templates.il file tagProcessor.php e il core della logica per il processamento delle view e rende possibile creare file html con tag custom che vengono convertiti per poi comporre in output **DOMDocument** validi. Questo preprocessing dei template rende possibile riutilizzare le view e renderle agnostiche dai controller che le usano, rendono cosi possibile riutilizzarle.

## CONTROLLER CLASSES

Il compito dei controller e di gestire la logica di interazione tra i dati e le azioni dell'utente quindi di reindirizzare, renderizzare e popolare le pagine o interagire con i Domain

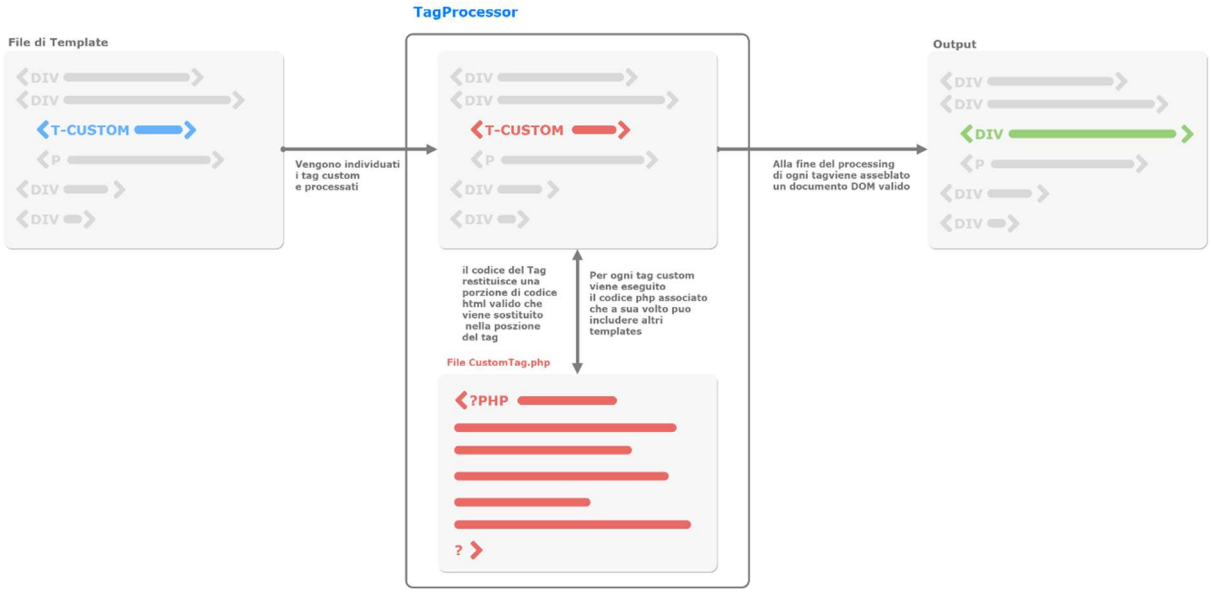
Le classi predisposte dal file ControllerLogic.php sono le seguenti:

- **Controller (abstract):** classe base astratta che deve essere estesa da ogni controller predispone alcuni metodi per semplificare e unificare lo sviluppo come:
    - `function render(String $title,template\PageModel $pageModel)` che seve per renderizzare una pagina (quindi processare i templates con i dati creati dal controller)
    - `static function redirect(String $controller,String $action="index",$params=[],$type="SESSION")` usata per reidirizzare ad altri controllers, da notare che è una funzione statica quindi può essere usata anche esternamente senza bisogno di un instance della classe.
- **ControllerDecorator(class)**: classe che viene effettivamente instanziata e ha bisogno di in ingresso una istanza di Controller valida questa classe sfrutta il paradigma di programmazione **Decorator** per aggiungere funzionalità alla istanza della classe controller si occupa di gestire i le **Annotations comments** tramite l'uso di **Code Inspection e Code Reflection** per eseguire metodi definiti nei commenti sopra ogni metodo e anche di eseguire i metodi corretti richiesti dall' URI con cui si visita lo script, si occupa anche delle **Dependency Injection.**
- **AnnotatedMethod** e **ServiceMethod:** classi usate per eseguire i metodi richiesti nelle annotazioni usate dal ControllerDecorator

## TEMPLATING CLASSES

Queste classi sono state create per consentire una modulare composizione delle pagine da mostrare all'utente, il concetto su cui si basa e il concetto di **Componente (DOM Tag)** ogni tag del template viene processato dalla classe **tagProcessor** e nel caso sia un tag custom (ovvero che rientra nel pattern t-customtag) viene eseguita la funzione render della classe corrispondente (che deve essere una classe che estende la classe **htmlTag**). Per ogni **DOM tag** (anche quelli standard html) vengono processati gli attributi che possono essere considerati gli input dei Componenti questi dati sono poi passati ad ogni tag che è figlio di quello appena processato consentendo cosi di ottenere uno scope per i dati e di poter creare un nesting logico dei tags.

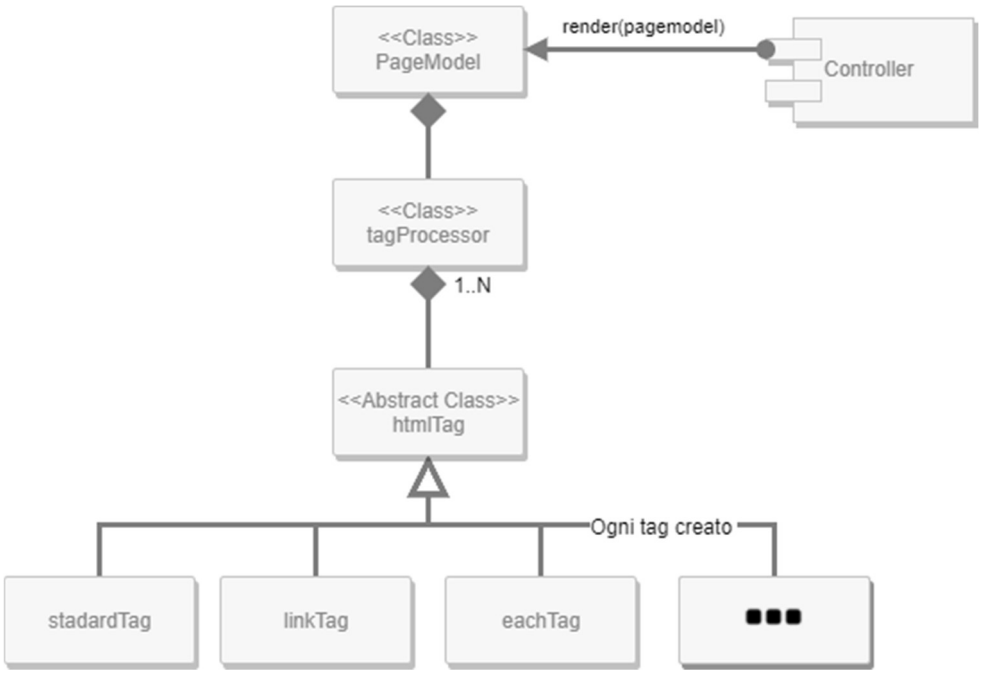
Grafico del processing dei tag DOM (tagProcessor class):



Classi del templating system del progetto:

- **PageModel (class):** questa che viene instaziata e popolata con i dati da passare alla view che poi verrà processata dalla classe tagProcessor. È possibile fare il nesting di più pageModels semplicemente facendo il render di un altro pageModel nel body del padre. Questa classe si preoccupa anche di caricare gli asset (script o file di stile CSS o immagini …) definiti nella proprietà **resources** mentre i dati sono messi in un array associativo chiave valore nella proprietà **model** questi dati poi possono essere referenziati negli attributi dei **Componenti DOM** (**custom tags** e anche **stardard html** tags) con la chiave associata, possono essere passati dati anche complessi non solo stringhe o numeri ma anche array, classi, enumeratori o qualsiasi altro costrutto del **PHP**.
    - **Esempi di referenziazione dei dati:**
        - `@{variabile:[default]}` questa scrittura è possibile utilizzarla solo negli attributi dei tags DOM, significa che verrà passato il valore del template referenziato dalla nome della chiave **variabile** nell'array.
        - `${codice php valido}` questa scrittura è utilizzabile ovunque sia negli attributi dei tag sia in qualsiasi altra parte del codice html in base alla sua posizione durante il parsing verra' prodotto il corrispettivo DOM valido.
        - `@{variabile:[${return 'ciao'}]}` è possibile fare il nesting delle due scritture.
- **tagProcessor (class):** questa classe si occupa di processare tutti i tag DOM presenti nel tamplate dichiarato dalla classe **PageModel.** Utilizza la ricorsione per il processing e si occupa anche di copiare il model dei tag padri nei tag figli e di ricomporre un documento DOM valido.
- **htmlTag (abstract):** questa è la classe base che ogni tag deve estendere per essere utilizzato e riconosciuto durante il processing della classe tagProcessor. Ogni classe che estende htmlTag deve implementare il metodo getModel().

**Grafico UML:**



## DOMAIN CLASSES

la parte del MVC che riguarda i dati e trattata dal file dbConnection.php che comprende le classi:

- **dbConnection (singleton)**: questa classe si occupa di gestire la connessione con il database e provvedere a una interfaccia per l'esecuzione delle query. questa classe e un sigleton perche viene chiamata molte volte ma non è necessario che sia instanziata ogni volta riducendo così il carico sul server e anche perché si occupa di caricare le classi di dominio (che sono inheritance di **Domain**) che mappano le tabelle del database in classi

- **Domain (abstract class):** questa è la classe base di tutte le classi di Dominio si occupa di mappare le tabelle del database in classi più maneggevoli da usare su **PHP** utilizzando questa tipologia di mappatura dove bisogna specificare per ogni classe le connessioni con altre classi (implementando i metodi astratti **benlongsTo** e **hasMany**) quindi quando si esegue una query con la funzione **findAll** (es: SELECT * FROM Orders) vengono ritornate tutte le istanze della tabella Orders mappate sulla classe di Dominio Orders nel caso la tabella Orders ha una foreign key in una sua colonna questa proprietà viene mappata nella classe non come valore dell'id della foreign key ma con una istanza della classe di dominio alla quale quella tabella e mappata (es: Orders FK PersonID -> (PersonID,'Persons') viene ritornata una istanza della classe Persons) con questa tipologia di implementazione e necessario evitare tabelle cicliche **ed e per questo motivo che nella relazione tasklist e stata introdotta una rindondanza.**

**Esempio:**

```php
class task extends Domain{
    public $id;
    public $Nome;
    public $Descrizione;
    public $Completato;
    public $DataCreazione;
    public $DataCompletamento;
    public $DataScadenza;
    public $Completata;
    public $User;
    public $Project;
    public $Assignee;

    public function belongsTo(){
        return ['User'=>'User','Project'=>'Project','Assignee'=>'Assignee'];
    }

    public function hasMany(){}

    public function primaryKey(){
        return 'id';
    }
}
```

**Database query result**

```
array(1)
  0: array(12)
    id: "1"
    Nome: "Prova Task 1 with edit"
    Descrizione: "Descrizione task 1"
    DataCreazione: "2019-05-04"
    DataScadenza: "2019-05-31"
    DataCompletamento: "2019-05-05"
    Completata: "1"
    User: "1"
    Project: "1"
    Assignee: "5"
    Condivisore: null
    SharedTask: null
```

Task::find

**Domain Class output**

```
task
  id: "1"
  Nome: "Prova Task 1 with edit"
  Descrizione: "Descrizione task 1"
  Completato: null
  DataCreazione: "2019-05-04"
  DataCompletamento: "2019-05-05"
  DataScadenza: "2019-05-31"
  Completata: "1"
  User: User
  Project: Project
  Assignee: Assignee
  Condivisore: null
  SharedTask: null
```

**Gli attributi definiti in belongsTo sono stati mappati correttamente nelle loro classi di Dominio**