Diario di lavoro

Titolo progetto	Gestione ore
Luogo	Trevano-Canobbio
Data	24 ottobre 2019

Lavori svolti

Durante la giornata di oggi mi sono occupato di implementare le classi Model che si interfacciano al database ed eseguono le query su di esso.
Più di preciso, ho terminato la classe 'Assegnazione', a cui rimaneva solo l'aggiunta delle funzioni getter per i campi 'activity' e 'resource', rispettivamente "getActivity" e "getResource". Visto che le due funzioni contengono la stessa logica, ecco "getActivity" come esempio:

```
/**
 * Gets the activity to which the resource is assigned.
 * @return Activity The activity to which the resource is assigned.
 */
public function getActivity(): Activity
{
    return $this->activity;
}
```

Poi ho aggiunto la funzione "login" all'interno della classe 'Resource', che riceve come parametro una stringa per il nome e una stringa per la password e controlla che il nome e la password corrispondano a quelli di una risorsa.

```
/**
 * Checks if a resource's name and password correspond to those read from the database.
 * @param string $name The name with which to login.
 * @param string $password The password with wich to login.
 * @return bool true if the resource's name and password correspond
 * to those of an existing resource in the database, false otherwise.
 */
public function login(string $name, string $password): bool
{
    //Get the resource whose name is equal to parameter name
    $databaseResource = $this->getResourceByName($name);
    //Check if a result was returned
    if (isset($databaseResource)) {
        //Check if the value of parameter 'password' and the hash read from the databas
e correspond
        return password_verify($password, $databaseResource->password);
    }
    //If no result was found, return false
    return false;
}
```

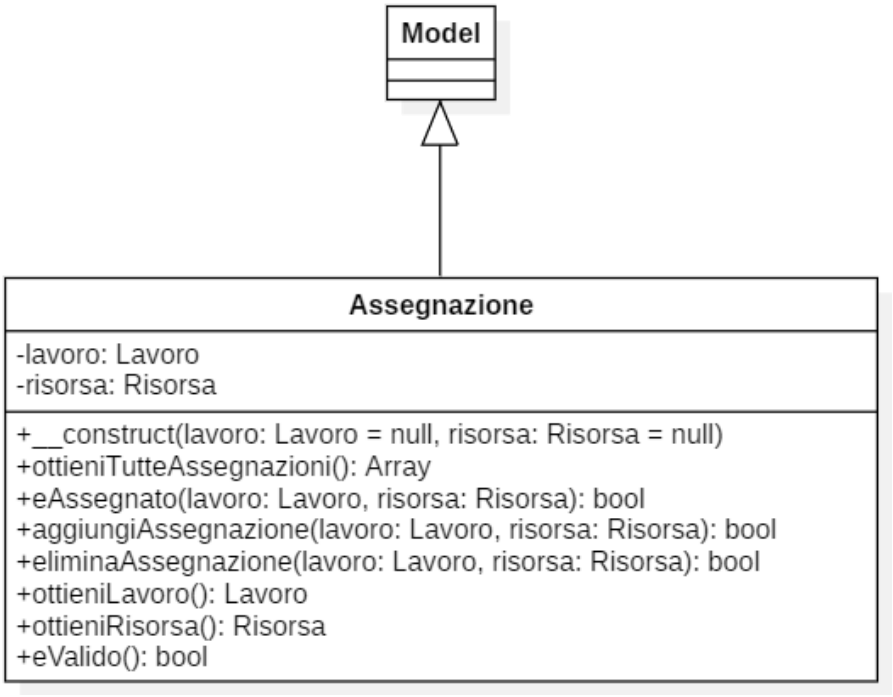
Poi ho cominciato a lavorare sulla classe 'OreLavoro', la quale è stata tradotta in 'WorkHours'. Di questa classe, per mancanza di tempo, ho potuto creare solo la funzione costruttrice, i campi 'activity', 'resource', 'date' e 'hoursNumber' e la funzione "ottieniTutteOreLavoro", che ho rinominato in

"getAllWorkHours". Questa funzione si occupa, come le precedenti funzioni "getAll…", di leggere e restituire tutti i dati della tabella 'ore_lavoro' del database:

```
/**
 * Get all work hours reading their data from the database.
 * @return array An array containing a object of type WorkHours for each line read from
 the database.
 */
public function getAllWorkHours()
{
    //Instantiates the array that will be returned.
    $workHours = [];
    //Get the database's data thanks to superclass 'Model'.
    $models = $this->getAllModels();
    //Instantiate an empty object of type Activity to use its methods
    $baseActivity = new Activity();
    //Instantiate an empty object of type Resource to use its methods
    $baseResource = new Resource();
    // Loop through each element read from the database and for each of them add an obj
ect of type Assignment with the data from the current element from models to array acti
vities.
    foreach ($models as $model) {
        //Get the activity with the same name as the current model.
        $modelActivity = $baseActivity->getActivityByName($model["nome_lavoro"]);
        //Get the resource with the same name as the current model.
        $modelResource = $baseResource->getResourceByName($model["nome_risorsa"]);
        //Only if both the model's activity and resource are not null, can we add the e
lement to the array
        if (isset($modelActivity) &&
            isset($modelResource)) {
            //Add a new object of type Assignment with the model's data to the array.
            array_push($workHours,
                new WorkHours(
                    $modelActivity,
                    $modelResource,
                    DateTime::createFromFormat("Y-m-d", $model["data"]),
                    intval($model["numero_ore"])
                )
            );
        }
    }
    return $workHours;
}
```

Problemi riscontrati e soluzioni adottate

Ho commesso un errore durante la giornata di lavoro scorsa: la classe 'Assegnazione' è stata modificata, diversamente da quanto affermato nel rapporto, solamente la sua progettazione non era stata aggiornata di conseguenza. Per rettificare, di seguito la nuova progettazione della classe 'Assegnazione':



Semplicemente è stata aggiunta la funzione "eValido", come nelle altre classi Model modificate l'ultima volta.

Poi ho fatto fatica con i vari errori di tipo sbagliato ritornato da una funzione come "getActivityByName", perché il mio codice ritorna il valore `null` quando non viene trovata una riga corrispondente ai parametri passati alla funzione, mentre il tipo di ritorno era sempre `Activity` o `Resource` o un oggetto simile. Per sistemare mi è bastato dire che il tipo di ritorno della funzione sarebbe stato l'oggetto normale, preceduto da un `?`, in modo da dire che sarebbe anche potuto essere il valore `null`.

Esempio:

```
public function getActivityByName(string $name): ?Activity
```

Ho poi aggiunto i tipi di ritorno nullabili, come quelli appena descritti, alle funzioni getter delle varie classi più alcuni controlli per vedere se `activity` e `resource` sono nulli nella funzione "isAssigned" della classe 'Assignment':

```
//Check if activity and resource are valid
if ($activity->isValid() &&
    $resource->isValid()) {
    //Esegui la funzione normalmente...
}
```

In seguito, mentre verificavo il funzionamento delle funzioni della classe, ho notato che per due oggetti che nel database sono assegnati l'uno all'altro, quando veniva richiamata la funzione "isAssigned" veniva ritornato il valore `false`. Questo perché nella funzione "isValid" della classe 'Resource' avevo usato come riferimento una variabile chiamata 'resource', invece del normale 'this':

```
return isset($resource) &&
       isset($resource->name) && ...
```

Quindi ho rimpiazzato ogni accenno alla variabile 'resource' con un riferimento all'istanza corrente 'this'.

```
return
       isset($this) &&
       isset($this->name) &&
```

Inoltre la funzione ‘getResourceByName’ era rimasta a quando la classe aveva solo i due campi ‘name’ e ‘hourCost’, quindi ho dovuto aggiungere l’assegnazione ai campi rimanenti ‘password’ e ‘role’. Infatti, da:

```
$resource = new Resource($model["nome"], $model["costo_ora"]);
```

È diventato:

```
$resource = new Resource($model["nome"], $model["costo_ora"], $model["password"], $mode
l["ruolo"]);
```

In seguito ho visto che la funzione “deleteAssignment” cercava di eliminare una riga del database con i due oggetti invece che con i loro nomi.
Quindi questo codice:

```
return $this->deleteModel([$assignment->activity, $assignment->resource]);
```

È diventato questo:

```
return $this->deleteModel([$assignment->activity->getName(), $assignment->resource-
>getName()]);
```

Più tardi ancora ho constatato che la funzione “addModel” della superclasse ‘Model’ non funziona perché io stavo usando le chiavi primarie della tabella come se fossero tutti i suoi campi. Per risolvere ho dovuto aggiungere un campo “fields” alla classe:

```
/**
 * @var array The name of all table fields.
 */
private $fields;
```

Questo campo viene usato nella funzione “addModel” in modo da poter sapere il nome dei vari campi della tabella nei quali scrivere:

```
/**
 * Inserts a model into the MySQL table whose name corresponds to the value of field ta
bleName.
 * @param array $values An array that contains an element of type string for each value
 defined inside array fields.
 * @return bool true if the insertion was successful, false otherwise.
 */
protected function addModel(array $values): bool
{
    if (count($this->fields) == count($values)) {
        //Write the query that will write into the database
        $query = "INSERT INTO $this->tableName(" . implode(", ", $this-
>fields) . ") VALUES (";
        //Loop each element of array keys and create a placeholder for each of them
        for ($i = 0; $i < count($values) - 1; ++$i) {
            $query .= ":value$i, ";
        }
        $query .= ":value" . (count($values) - 1) . ")";
        //Prepare the statement
        $statement = $this->database->prepare($query);
        //Bind each placeholder to its respective key
        for ($i = 0; $i < count($values); ++$i) {
            $statement->bindParam(":value$i", $values[$i]);
        }
        //Execute the statement
        return $statement->execute();
    }
    return false; }
```

Mi sono poi accorto che nella funzione “getAllAssignments” della classe ‘Assignment’ non veniva eseguito alcun controllo sul ritorno della chiamata ai metodi “getActivityByName” e “getResourceByName”, quindi c’era il rischio che venissero ritornati valori nulli in caso di errore di esecuzione della query, valori che sarebbero stati aggiunti al database tranquillamente. Ho quindi aggiunto il seguente controllo:

```
//Only if both the model's activity and resource are not null, can we add the element t
o the array
if (isset($modelActivity) &&
    isset($modelResource)) {
        //Aggiungi all’array normalmente
```

Punto della situazione rispetto alla pianificazione

Sono in orario con la pianificazione.

Programma di massima per la prossima giornata di lavoro

Portare avanti la documentazione della pianificazione e la realizzazione delle classi Model che si interfacciano al database.