Diario di lavoro

Titolo progetto	Gestione ore
Luogo	Trevano-Canobbio
Data	17 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 Model 'Activity', aggiungendo le ultime funzioni che mi rimanevano, ovvero i getter per i campi, che nella pianificazione avevo chiamato "ottieniNome", "ottieniNote", "ottieniDataInizio", "ottieniDataConsegna" e "ottieniOrePreventivate" e che ora sono state rinominate in "getName", "getNotes", "getStartDate", "getDeliveryDate", "getEstimatedHours". Essendo ogni funzione di questo tipo uguale alle altre, di seguito l'implementazione di "ottieniNome" come esempio:

```
/**
* Gets the name of this activity.
* @return string The name of this activity.
*/
public function getName(): string
{
    return $this->name;
}
```

In seguito ho cominciato a implementare la classe 'Risorsa', rinominata in 'Resource'. Ho cominciato a creare la classe, poi sono passato alla creazione della funzione "ottieniTutteRisorse", tradotta in "getAllResources". Questa funzione ritorna tutte le risorse presenti all'interno del database:

```
/**
 * Get all resources reading their data from the database.
 * @return array An array containing a object of type Resource for each line read from
the database.
 */
public function getAllResources(): array
{
    //Get the database's data thanks to superclass 'Model'.
    $models = $this->getAllModels();
    $resources = [];
    // Loop through each element read from the database and for each of them add an obj
ect of type Resource with the data from the current element from models to array activi
ties.
    foreach ($models as $model) {
        array_push($resources, new Resource($model["nome"], $model["costo_ora"]));
    }
    return $resources;
}
```

Andando avanti, ho creato anche la funzione "ottieniRisorsaDaNome", che ho rinominato in "getResourceByName", che si occupa di trovare la risorsa con il nome specificato all'interno della tabella 'risorsa' del database e ne ritorna i dati sotto forma di oggetto di tipo Risorsa. Il suo codice è quello che segue:

```
/**
 * Gets a resource with the specified name.
 * @param string $name The name for which to search for a resource in the database.
 * @return Resource An object of type Resource whose fields' values are equal to the da
ta of the line of the database's 'risorse' table whose name corresponds to the value of
parameter 'name'.
 */
public function getResourceByName(string $name): Resource
{
    //Use function getModelByKey inherited from superclass Model to get a single Resour
ce by its name.
    $models = $this->getModelByKey([$name])[0];
    //If a result has been returned
    if (count($models) > 0) {
        //Assign to a variable the result's data
        $model = $models[0];
        //Return a new object of type Resource with the result's data
        $resource = new Resource($model["nome"], $model["costo_ora"]);
        return $resource;
    }

    //If we got to this point, it means a result was not found, so return null
    return null;
}
```

Infine ho implementato pure la funzione “aggiungiRisorsa”, che è stata tradotta in “addResource” e la funzione “rimuoviRisorsa”, tradotta in “deleteResource”. Come è possibile intuire dal nome, la prima è la funzione che aggiunge una riga alla tabella ‘risorsa’ del database, la seconda è quella che la toglie. Di seguito entrambe le funzioni.

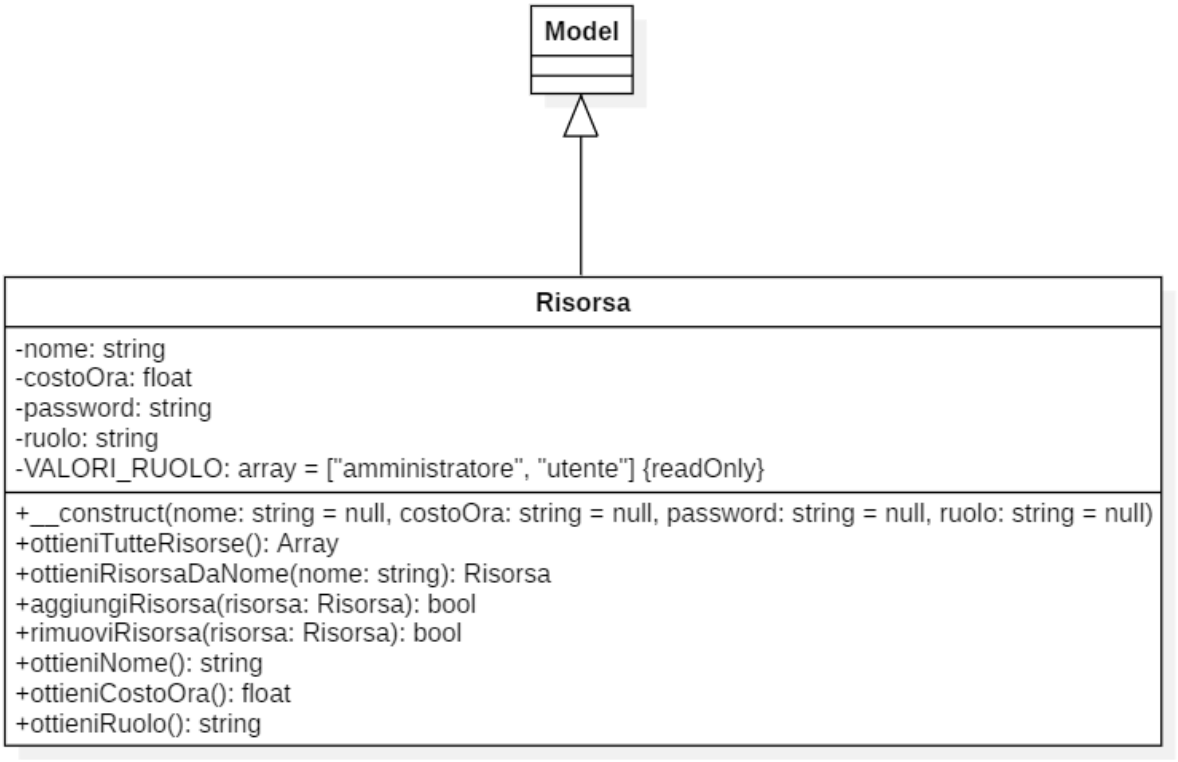
```
/**
 * Deletes a record from the MySQL table 'risorsa' where the name is equal to the name
of an object of type Resource.
 * @param Resource $resource The data of the activity to delete.
 * @return bool true if the deletion is successful, false otherwise.
 */
public function deleteResource(Resource $resource): bool
{
    return $this->deleteModel([$resource->name]);
}

/**
 * Inserts a new row into table 'risorsa' of database with data passed as parameter.
 * @param Resource $resource An object of type Resource that contains the data to add t
o the database.
 * @return bool true if the insert operation is successful, false otherwise.
 */
public function addResource(Resource $resource): bool
{
    //Check if the resource to be added is set, its name and cost per hour have been se
t and either
```

```
    // both its password and role (with a value that is contained in array ROLE_VALUES)
 are set or neither.
    if (isset($resource) &&
        isset($resource->name) &&
        strlen(trim($resource->name)) > 0 &&
        isset($resource->hourCost) &&
        $resource->hourCost >= 0.0 &&
        (
            (
                isset($resource->password) &&
                strlen(trim($resource->password)) > 0 &&
                isset($resource->role) &&
                strlen(trim($resource->role)) > 0 &&
                in_array($resource->role, self::ROLE_VALUES)
            ) ||
            (
                (
                    !isset($resource->password) ||
                    strlen(trim($resource->password)) == 0
                ) &&
                (
                    !isset($resource->role) ||
                    strlen(trim($resource->role)) == 0
                )
            )
        )) {
        //Write the query that will write to the database.
        $query = "INSERT INTO risorsa(nome, costo_ora, password, ruolo) VALUES (:name,
:hourCost, :password, :role)";
        //Prepare the query.
        $statement = $this->database->prepare($query);
        //Bind placeholders to their respective values
        //Crypt password
        $securePassword = password_hash($resource->password, PASSWORD_DEFAULT);
        //Replace placeholder ':name' with resource name taken from field 'name' of 're
source' parameter.
        $statement->bindParam(":name", $resource->name);
        //Replace placeholder ':hourCost' with resource name taken from field 'hourCost
' of 'resource' parameter.
        $statement->bindParam(":hourCost", $resource->hourCost);
        //Replace placeholder ':password' with resource name taken from field 'password
' of 'resource' parameter after being encrypted.
        $statement->bindParam(":password", $securePassword);
        //Replace placeholder ':role' with resource name taken from field 'role' of 're
source' parameter.
        $statement->bindParam(":role", $resource->role);
        //Execute the query
        $result = $statement->execute();
        return $result;
```

```
    }
    return false;
}
```

Problemi riscontrati e soluzioni adottate

Prima di cominciare a implementare la funzione “aggiungiRisorsa” mi sono reso conto del fatto che alla classe mancassero due campi: uno per la password e uno per il ruolo. Inizialmente avevo pensato che all’interno dell’applicazione non fosse necessario nessuno dei due, perciò nella progettazione non li avevo aggiunti, ma mi accorgo ora che per aggiungere una riga al database bisogna averli per forza.
Quindi ho modificato la progettazione aggiungendo questi due campi, più il metodo getter e un array che contiene i valori validi per il campo ‘ruolo’:



È importante notare la mancanza di una funzione ‘ottieniPassword’, fatta in modo per non permettere a nessuno di vedere le password con cui accedono i collaboratori.
Poi ho notato che non eseguivo alcun tipo di controllo sui dati nella funzione “aggiungiLavoro”, quindi si poteva provare a eseguire una query con tutti i valori nulli, o con un nome uguale a ‘ ‘ (solo il carattere dello spazio), o con un numero di ore preventivate minore di 0. Ho messo a posto in questo modo:

```
if (
    isset($activity) &&
    isset($activity->name) &&
    isset($activity->startDate) &&
    isset($activity->deliveryDate) &&
    isset($activity->estimatedHours) &&
    strlen(trim($activity->name)) > 0 &&
    $activity->estimatedHours > 0) {

    //Esegui la funzione normalmente
```

```
    //.
    //.
    //.

    }

    return false;
```

In pratica controllo se i campi obbligatori abbiano un valore diverso da null, poi controllo che la lunghezza del nome, che è l'unica stringa, sia maggiore di 0, infine controllo che il numero di ore preventivate sia maggiore di 0. In caso affermativo si passa al funzionamento normale della funzione, altrimenti si ritorna direttamente false senza andare più avanti.

Una cosa simile è stata fatta anche per la funzione "deleteActivity":

```
public function deleteActivity(Activity $activity): bool
{
    //Check if the value for the activity's name has been set and is not empty or a whi
tespace.
    if (isset($activity->name) &&
        strlen(trim($activity->name)) > 0) {

        return $this->deleteModel([$activity->name]);

    }

    return false;
}
```

In questo caso viene controllato solo il valore del campo 'name' perché è l'unico che viene usato dalla funzione.
Dopo ancora ho aggiunto gli stessi controlli anche alla funzione "getModelByKey" e "deleteModel", dapprima separati, poi visto che sono uguali li ho generalizzati all'interno di una funzione che ho chiamato "areKeysValid", che controlla che tutti i valori dell'array di chiavi primarie passato come parametro siano impostati e abbiano una lunghezza maggiore di 0. Ho poi rimosso il controllo aggiunto sopra alla funzione "deleteActivity" perché venivano già eseguiti nella funzione sulla quale si base: "deleteModel".
Di seguito la funzione "areKeysValid" contenuta nella classe Model:

```
/**
 * Check if an array contains only values that are not null or whitespace.
 * @param array $keys The array to check.
 * @return bool true if the array contains only values that are not null or whitespace.
 */
private function areKeysValid(array $keys)
{
    //Assume that the array is valid until it is proven invalid.
    $keysValid = true;

    //Loop through the array
    foreach ($keys as $key) {
        //If a value is null, empty or a whitespace, it is invalid, therefore the entir
e array is invalid.
```

```
        if (!(isset($key) && strlen(trim($key)))) {
            $keysValid = false;
        }
    }

    return $keysValid;
}
```

Per cui nelle funzioni che fanno uso di questa “areKeysValid” ho semplicemente aggiunto un’istruzione if per controllare se il valore del parametro è valido e, in caso affermativo, proseguire con l’esecuzione normalmente.

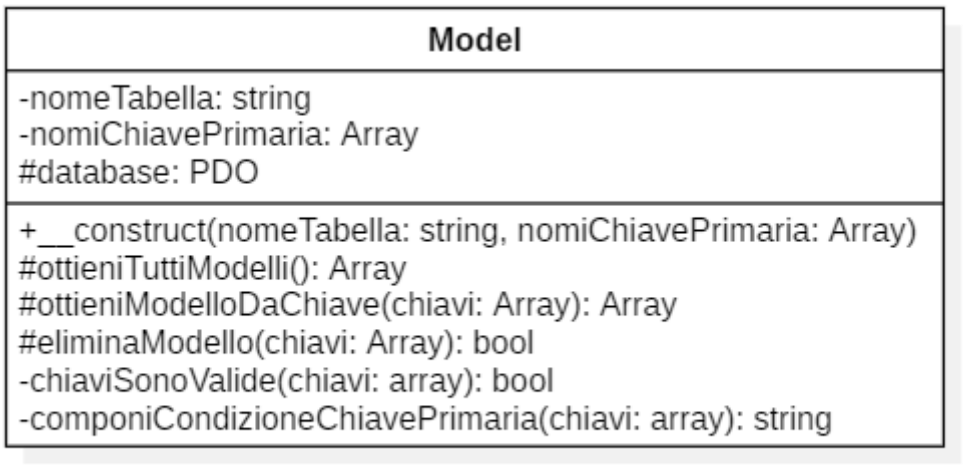
Per esempio, di seguito il nuovo codice sorgente della funzione “getModelByKey”:

```
protected function getModelByKey(array $keys): array
{
    $keysValid = $this->areKeysValid($keys);
    if ($keysValid) {
        //Esegui la funzione normalmente...
    }
    return null;
}
```

Per questo nella sottoclasse “Activity” ho dovuto aggiungere il controllo `isset($models)` al metodo “getActivityByName”:

```
public function getActivityByName(string $name): Activity
{
    //Use function getModelByKey inherited from superclass Model to get a single Activi
ty by its name.
    $models = $this->getModelByKey([$name]);

    //If a valid result has been returned.
    if (isset($models) && count($models) > 0) {
        //Esegui normalmente la funzione...
    }
}
```

Per riassumere, di seguito il diagramma UML aggiornato della classe Model:



Punto della situazione rispetto alla pianificazione
Sono in anticipo rispetto alla 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.