SOEN 6441 Advanced Programming Practices

Fall 2022

Final Project
Quotes Application

Submitted to Dr. C. Constantinides,P.Eng

Team Members

Vamsi Krishna Reddy Munnangi (40232641)

Revanth Velagandula (40229629)

INDEX

GITHUB LINK : https://github.com/vamsivk18/APP_Project

Abstract:

Quotes application is a web application that is used to view quotes that are very popular. In this application, users can create their own account and can view quotes of all the users that are registered to this application. Users can also update, delete their own quotes. Users can view other quotes, can search for quotes written by a particular author, and can search quotes containing particular words or phrases.

Tech Stack Used:

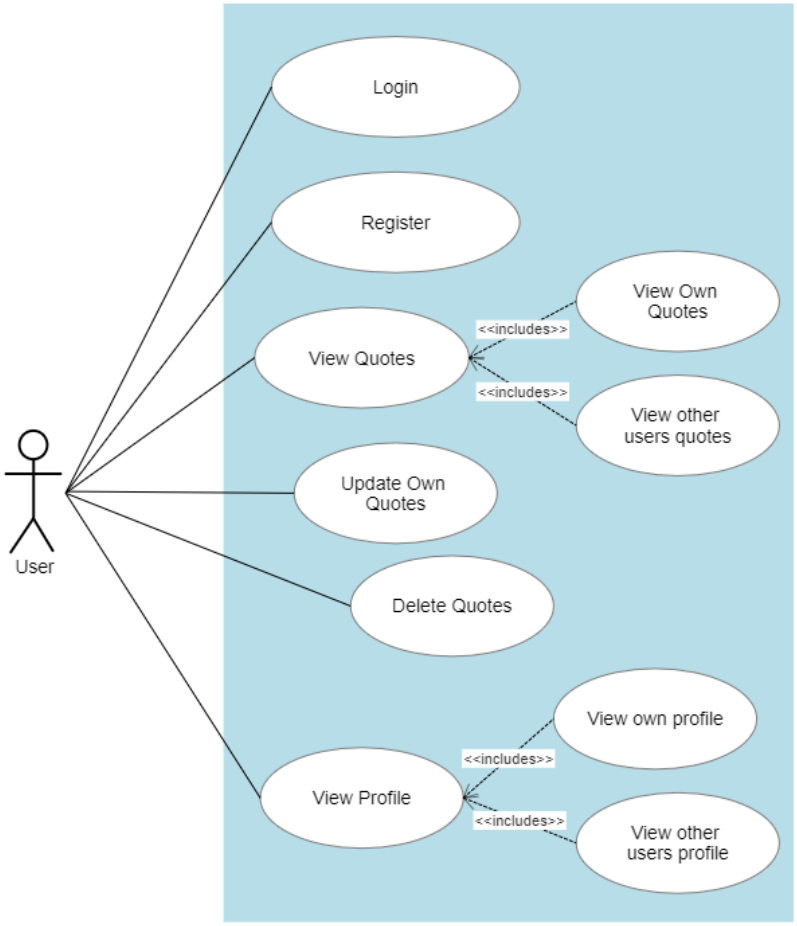
We used **PYTHON** scripts to extract the **JSON** Data from the web and stored it in a **MYSQL** database. The JSON data is converted into appropriate tables and stored.

We used **HTML**,**CSS** and **JavaScript** as frontend and **PHP** as backend to connect the database to the application.

For testing, we used the **PHPUnit** framework and performed testing on our application.

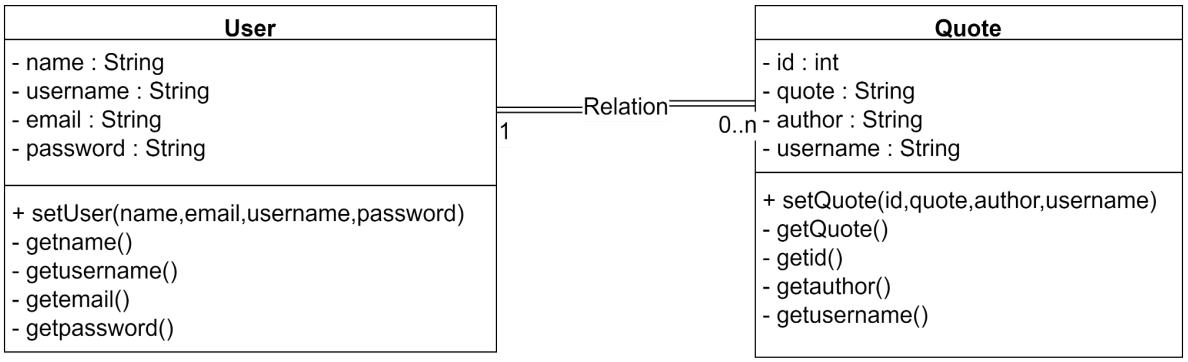
Diagrams:

UseCase Diagram:

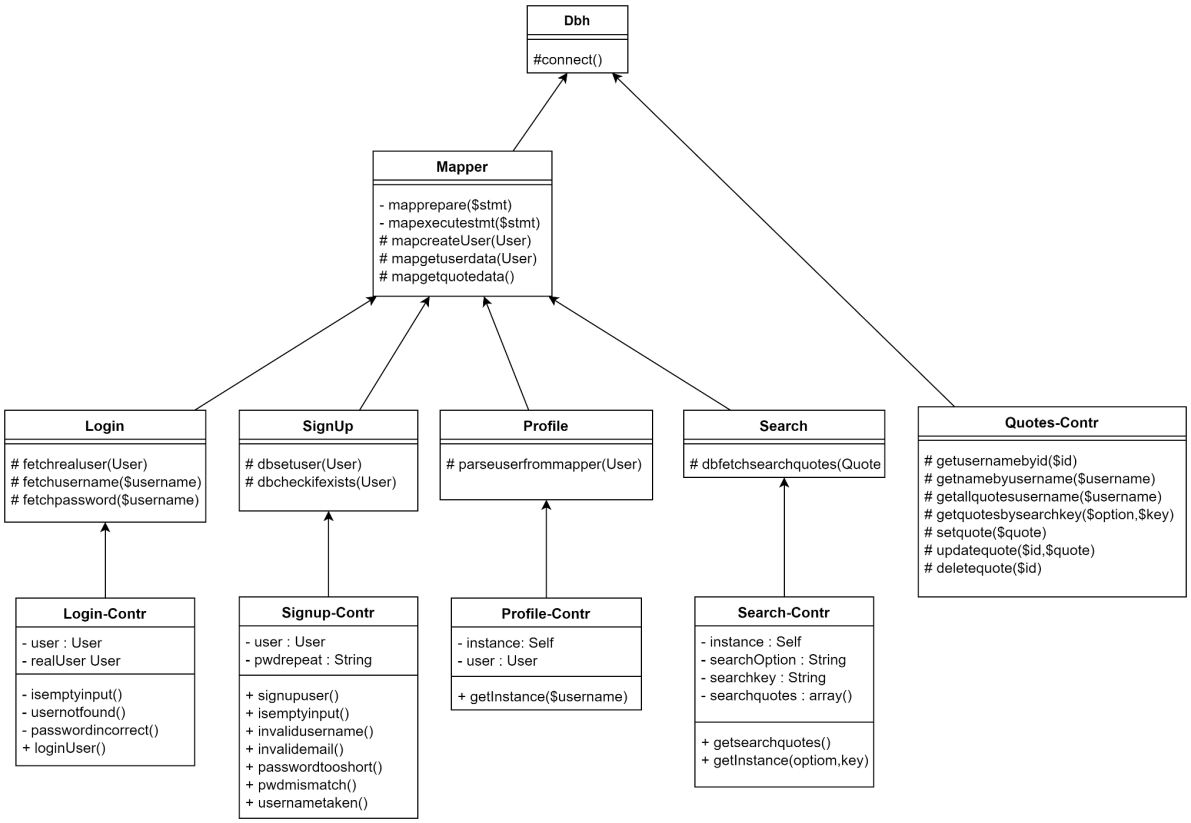


Class Diagram:

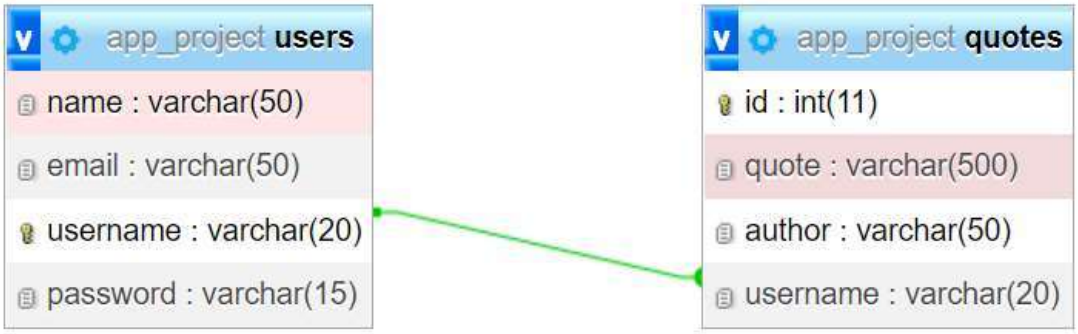
The main classes in the class diagram are User and Quote. We used a **Simple Mapping** design pattern for object relational structure. Here, individual classes map to separate database tables Users and Quotes.



Functional Class Diagram:



DataBase Schema:



Design Patterns Used:

In our Application we have used the following design patterns

Model View Controller Design Pattern

Singleton design pattern

Model-View-Controller

We used MVC design patterns to abstract and limit the functionality of each class. The View classes are used to present the data. The view has an instance of Controller. The Controller classes do all the functionality. The Model classes retrieve or update the data from the database.

Singleton design pattern

The singleton pattern is a design pattern that restricts a class's ability to instantiate multiple objects. It is simply a method of defining a class. We used a singleton design pattern for SearchController. When a user enters the search option and search key to search quotes, we will see if an instance of the same option and search key exists. If it exists, we will return the same instance.

```php
<?php
class SearchContr extends Search{
    private static $instance = null;
    private $searchoption;
    private $searchkey;
    private $searchquotes = array();

    public function __construct($searchtype,$searchfor){
        $this->searchtype = $searchtype;
        $this->searchfor = $searchfor;
        $this->searchquotes = $this->dbfetchsearchquotes(new Quote(),$searchtype,$searchfor);
    }
    public static function getInstance($searchoption,$searchkey){
        if(self::$instance==null || self::$instance->searchoption!==$searchoption || self::$instance->searchkey!==$searchkey)
        self::$instance = new SearchContr($searchoption,$searchkey);
        return self::$instance;
    }
    public function getsearchquotes(){
        return $this->searchquotes;
    }
}
?>
```

Data source architectural patterns used

We have used the following Data source architectural patterns

Data Mapper

We used DataMapper to map objects and relational schema. For instance, when a user signs up, an instance of User object is created and it is passed to the model to insert in the database, the datamapper maps/parses the created object into appropriate attributes to update the database. Similarly, when fetching the results from the database, the mapper maps the database row into appropriate User/Quote objects which can be used to present in the view.

Refactoring Strategies

In our Application we have used the following design patterns

1)Extract Method
2) Inline Method
3) Split temporary variable
4) Abstraction

Use of testing tools

We have used the PHPUnit framework for testing in our application.Following are the test cases for login page,

```php
public function testUserLogin(){
    $user = new User();
    $logindata = new LoginContr($user);
    $this->assertEquals($logindata->isemptyInput(),true);//One or more fields empty
    $user->setUsername("test");
    $user->setPassword("test");
    $logindata = new LoginContr($user);
    $this->assertEquals($logindata->userNotFound(),true);//User Doesnot Exist
    $user->setUsername("vamsi");
    $user->setPassword("test");
    $logindata = new LoginContr($user);
    $this->assertEquals($logindata->passwordIncorrect(),true);//Password Incorrect
    $user->setUsername("vamsi");
    $user->setPassword("Vamsi@1827");
    $logindata = new LoginContr($user);
    $this->assertEquals($logindata->isemptyInput(),false);//Fields not empty
    $this->assertEquals($logindata->userNotFound(),false);//User Exists
    $this->assertEquals($logindata->passwordIncorrect(),false);//Password is correct
    $this->assertEquals($logindata->loginUser(),true);//Succesfully Logged in
}
```

Following are the test cases for signup page,

```php
public function testUserSignup(){
    $user = new User();
    $signupdata = new SignupContr($user,"pwd");
    $this->assertEquals($signupdata->isemptyInput(),true);
    $user->setName("Vamsi Krishna");
    $user->setEmail("vamsi123@gmail.com");
    $user->setUsername("vamsi");
    $user->setPassword("hello");
    $this->assertEquals($signupdata->isemptyInput(),false);
    $this->assertEquals($signupdata->usernameTaken(),true);
    $user->setUsername("vamsi2");
    $this->assertEquals($signupdata->usernameTaken(),false);
    $this->assertEquals($signupdata->pwdMisMatch(),true);
    $user->setPassword("pwd");
    $this->assertEquals($signupdata->pwdMisMatch(),false);
    $this->assertEquals($signupdata->passwordtooshort(),true);
    $user->setPassword("password");
    $signupdata = new SignupContr($user,"password");
    $this->assertEquals($signupdata->passwordtooshort(),false);
    $this->assertEquals($signupdata->signupUser(),true);
}
```