**1. Introduction to Testing:**

**● Define software testing as the process of evaluating software to identify defects or bugs. ● Highlight the importance of testing in software development for ensuring reliability and correctness.**

Define software testing as the process of evaluating software to identify defects or bugs.

* Software testing is essential for evaluating the functionality, reliability, and security of software applications. It involves the systematic execution of test cases to identify defects or bugs.

● Highlight the importance of testing in software development for ensuring reliability and correctness.

1. Ensuring Product Quality: Thorough testing ensures that the software meets the required quality standards. It helps in validating that the software functions as expected, performs well under different conditions, and provides a satisfactory user experience.
2. Enhancing User Satisfaction: Quality software results in higher user satisfaction. Testing helps in uncovering usability issues, performance bottlenecks, and other factors that may affect the user experience. By addressing these issues, the software becomes more reliable and user-friendly.
3. Reducing Risks: Testing helps in mitigating risks associated with software development. It ensures that critical functionalities work as intended, data integrity is maintained, and security vulnerabilities are identified and addressed. This reduces the risk of system failures, data breaches, and other adverse events.

**2. Purpose of Testing:**

**● Explain that testing aims to identify defects early in the development process and verify that software components perform as intended.**

Testing ensures that the "My Home" website operates as intended, providing a seamless experience for users, admins, and service providers. It helps in identifying and fixing issues early in the development process.

1. Early Detection of Defects: Testing is essential for identifying defects or bugs in the software as early as possible in the development process. By detecting issues early, developers can address them promptly, minimizing the cost and effort required for bug fixing later in the development lifecycle. Early defect detection also helps in maintaining project timelines and meeting delivery deadlines.
2. Verification of Software Components: Testing verifies that individual software components, such as functions, modules, or classes, perform according to their specifications. It ensures that each component behaves as intended and produces the expected output for a given set of inputs. By validating the functionality of individual components, testing contributes to the overall reliability and correctness of the software system.
3. Validation of Requirements: Testing plays a crucial role in validating that the software meets its specified requirements. By executing test cases based on functional and non-functional requirements, testers can verify that the software fulfills user expectations, business objectives, and regulatory requirements. This validation ensures that the software delivers the intended value to its stakeholders.

**3. Focus on Testing a Single Component:**

● Choose a component **(class, module, function, etc.) from your codebase for testing.** ● Explain why testing this component is important, considering its role, complexity, and impact on the system.

–Valerian:

I chose to test the login admin part of the "My Home" web application which is crucial for ensuring smooth operation and efficient management of the platform. The login part of admin is important in order to be logged only the admin and the fact that the admin credentials are part of the code. Testing for functionality assurance makes sure that the login process functions as it should. It confirms that users who enter the wrong credentials are not allowed access and that they can successfully authenticate with the right ones. This guarantees your application's dependability and security.

Testing aids in the discovery of errors or problems with the login component. You can find potential problems that might occur during user authentication by methodically checking various circumstances (valid credentials, invalid credentials, edge cases).

–Erina:

The UserTest for editing is crucial for testing the functionality related to editing user information in the application. This component is chosen for testing because user information editing is a fundamental feature of most web applications, including updating profiles, changing passwords, and modifying contact details. Ensuring that this feature works as expected is vital for user satisfaction and system reliability. By testing this component, we can verify that the application correctly handles user information editing, guaranteeing that user data remains accurate and up-to-date. The testSuccessfulEditInformation() method specifically ensures that the user's name and email are updated correctly, providing confidence in the application's ability to manage user data effectively.

–Griselda:

The DatabaseConnectionTest class is critical for ensuring the reliability and robustness of the application's database connection functionality. Testing the database connection is crucial because it is a foundational component of the application. A failed database connection can lead to various issues, including data corruption, loss of data integrity, and service disruptions. By simulating a connection failure and checking for the appropriate error message, this test ensures that the application responds correctly to database connection failures, thereby preventing potential issues and ensuring uninterrupted service for users. I chose to test this component because a reliable database connection is essential for the proper functioning of the entire application.

**4.Preparing Test Cases:**

● Discuss how to prepare test cases covering various scenarios, including normal inputs, edge cases, and invalid inputs.

**Login:** Normal Inputs:

Valid Credentials: Test the login functionality with correct username and password. Ensure that the user is successfully authenticated and granted access.

Invalid Inputs: Incorrect Username or Password: Test the login functionality with incorrect username or password combinations. Verify that the system rejects these credentials and displays appropriate error messages indicating login failure.

**Edit:** Edge case:

The UserTest class contains test methods to ensure the correct editing of user information. The testSuccessfulEditInformation method verifies that user information is successfully updated when provided with valid name and email inputs. Additionally, edge cases are covered to handle various scenarios: testEditInformationWithEmptyName and testEditInformationWithEmptyEmail check for exceptions when either the name or email input is empty.

**Database connection:** Invalid:

The DatabaseConnectionTest class ensures the application handles database connection failures correctly. By defining TEST\_ENVIRONMENT as true, it simulates a database connection failure. The testConnectionFailure() method includes the connect.php file, capturing the output to check for the presence of the string 'Failed to connect with database'. This test validates the application's ability to handle invalid inputs, specifically the failure to establish a database connection. It ensures that the application responds appropriately to such scenarios, contributing to its robustness and reliability.

**5.Choosing Testing Frameworks:**

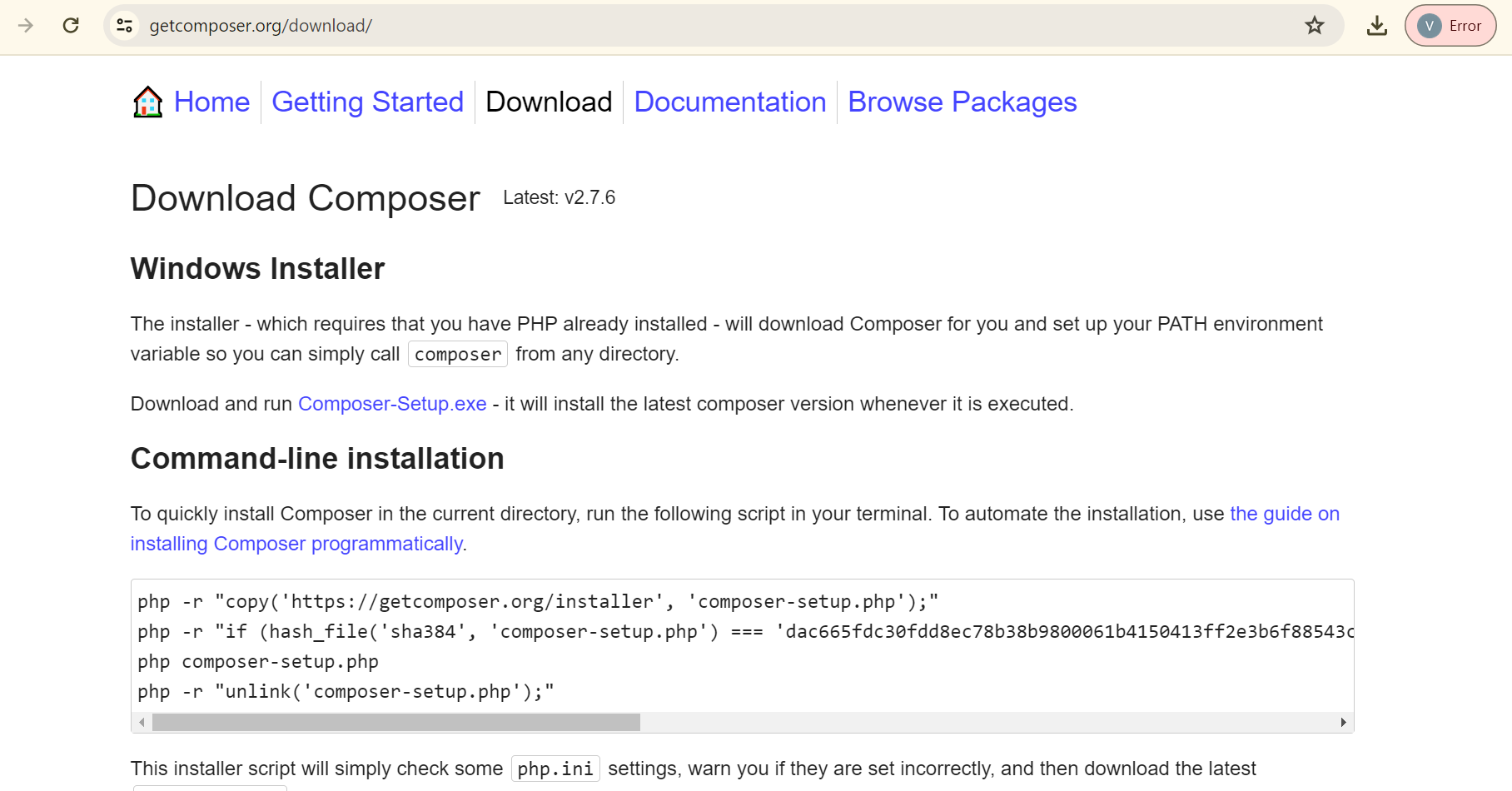
● Recommend suitable **testing frameworks or tools for** unit , integration testing etc… based on your programming language or environment (e.g., JUnit for Java, pytest for Python)

● Provide guidance on setting up the testing environment with the chosen tools.

For achieving this test we have used the Composer and Phpunit.

You can go at this link and find composer: [getcomposer.org/download/](http://getcomposer.org/download/)

At the web you can find : Download and run [Composer-Setup.exe](https://getcomposer.org/Composer-Setup.exe) - it will install the latest composer version whenever it is executed.



Download the Composer Setup executable file (.exe).

Double-click the downloaded file to start the installation process.

Follow the on-screen instructions to complete the installation.

Once the installation is complete, you can verify the Composer installation by opening a

new Terminal (or Command Prompt on Windows) and typing:

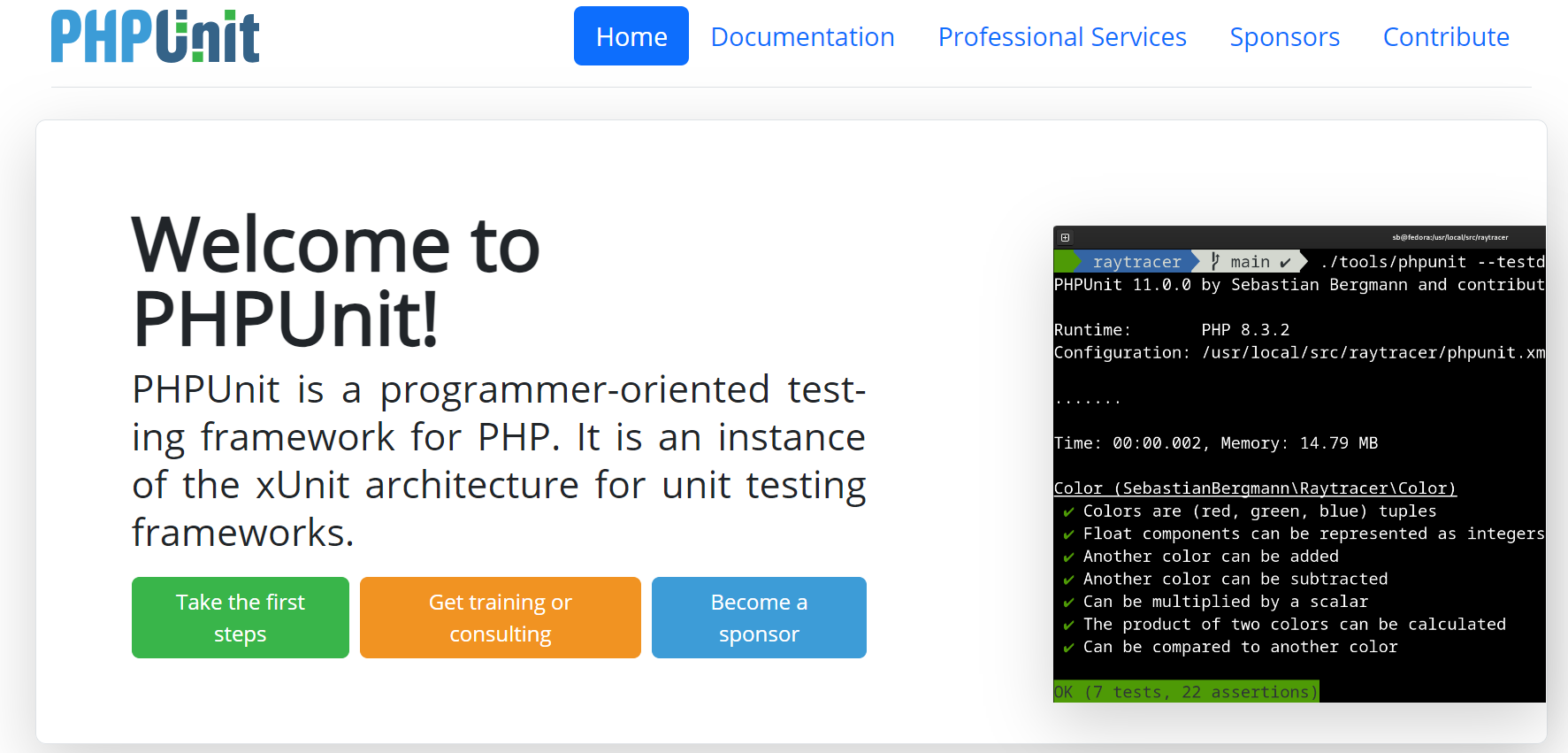
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash\_file('sha384', 'composer-setup.php') === 'dac665fdc30fdd8ec78b38b9800061b4150413ff2e3b6f88543c636f7cd84f6db9189d43a81e5503cda447da73c7e5b6') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP\_EOL;"

php composer-setup.php

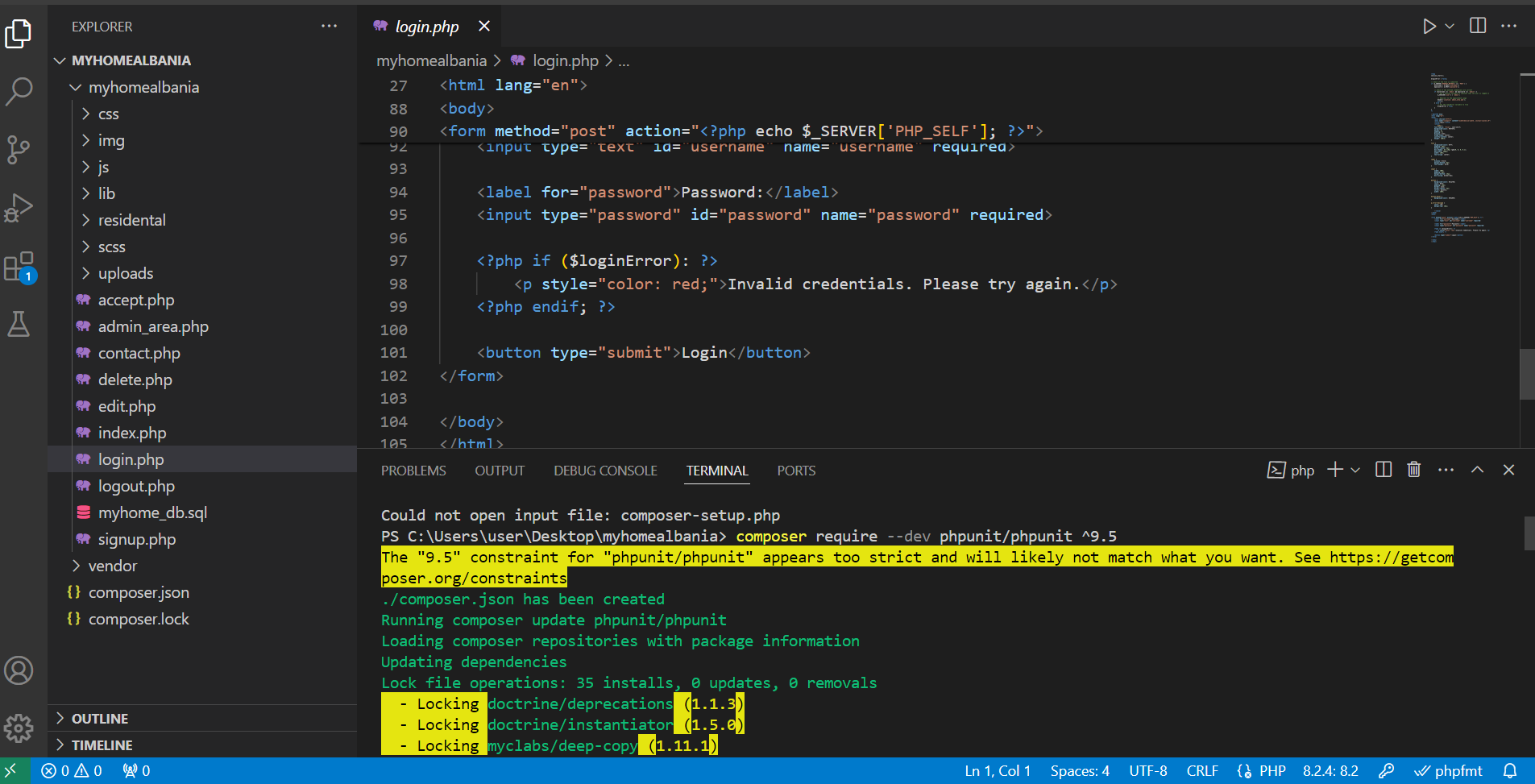
php -r "unlink('composer-setup.php');"

After this you have to make the PHP Unit Testing:



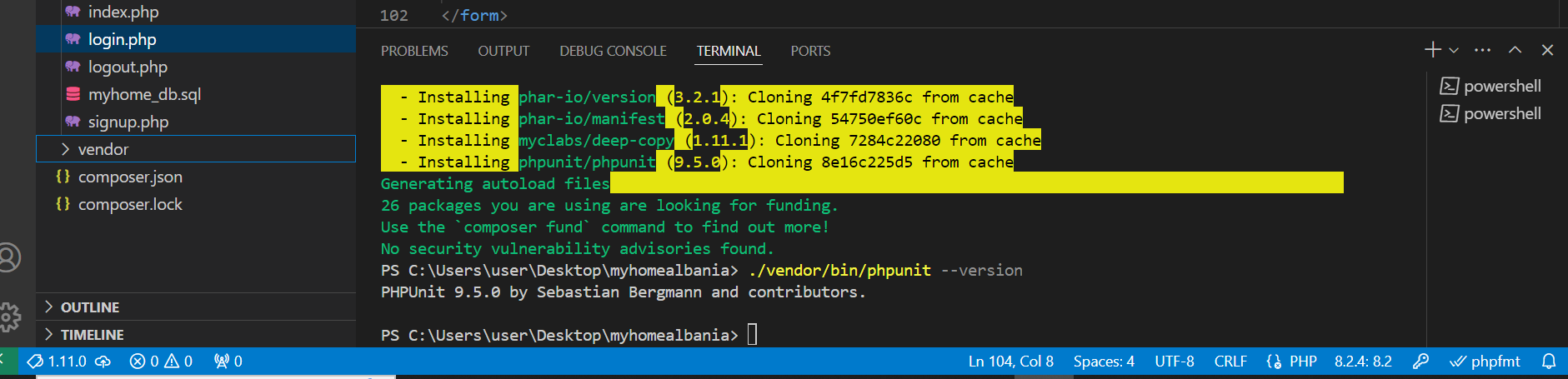
1.Install PHPUnit: Open your terminal and navigate to your project directory. Then, run the following command to install PHPUnit as a development dependency using Composer:

composer require --dev phpunit/phpunit ^9.5

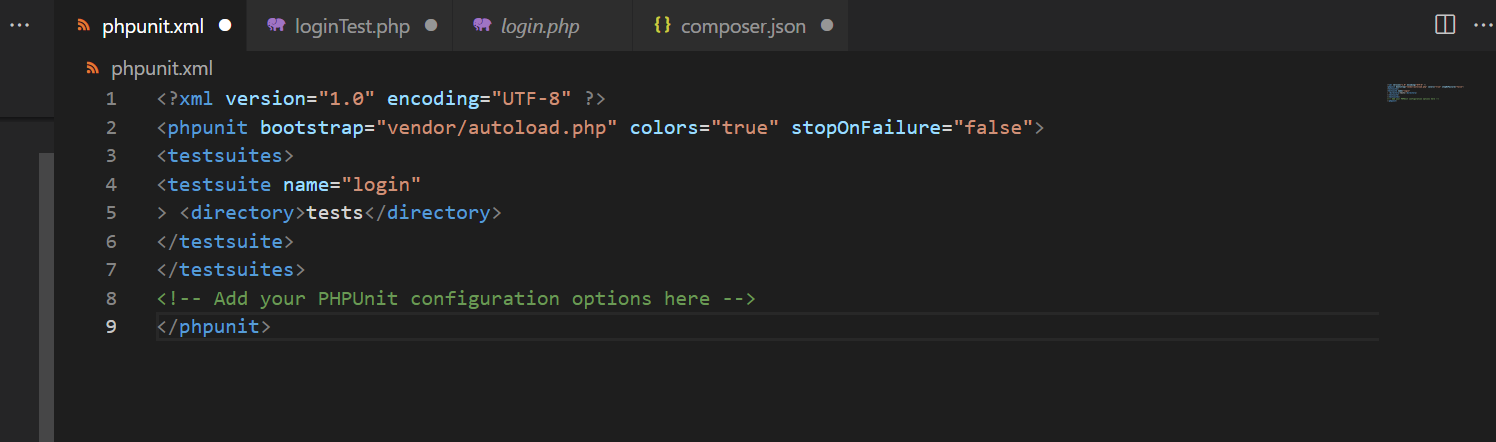


Inspect PHPUnit Version: Before writing test cases, it's a good idea to verify the version of PHPUnit installed in your project. You can do this by running the following command in your Terminal:

./vendor/bin/phpunit --version



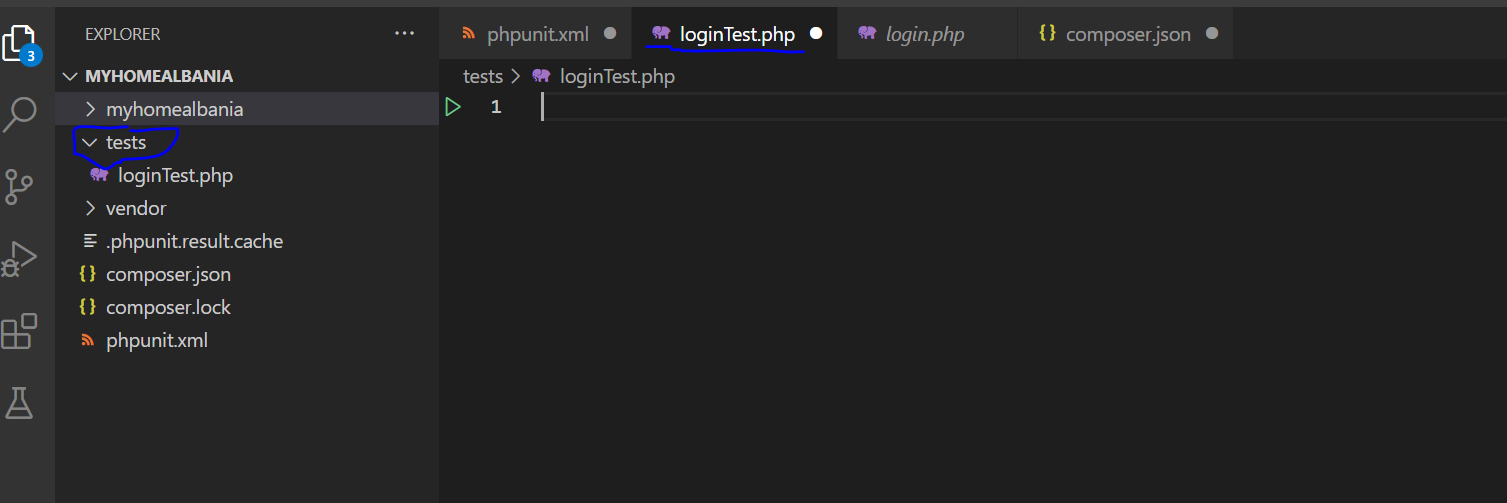
3.Write PHPUnit Configuration File: PHPUnit can be configured using an XML configuration file, which allows you to customize various aspects of the testing process. Create a new file named phpunit.xml in your project's root directory, and add the necessary configuration settings.Here's a basic example of a phpunit.xml configuration file:



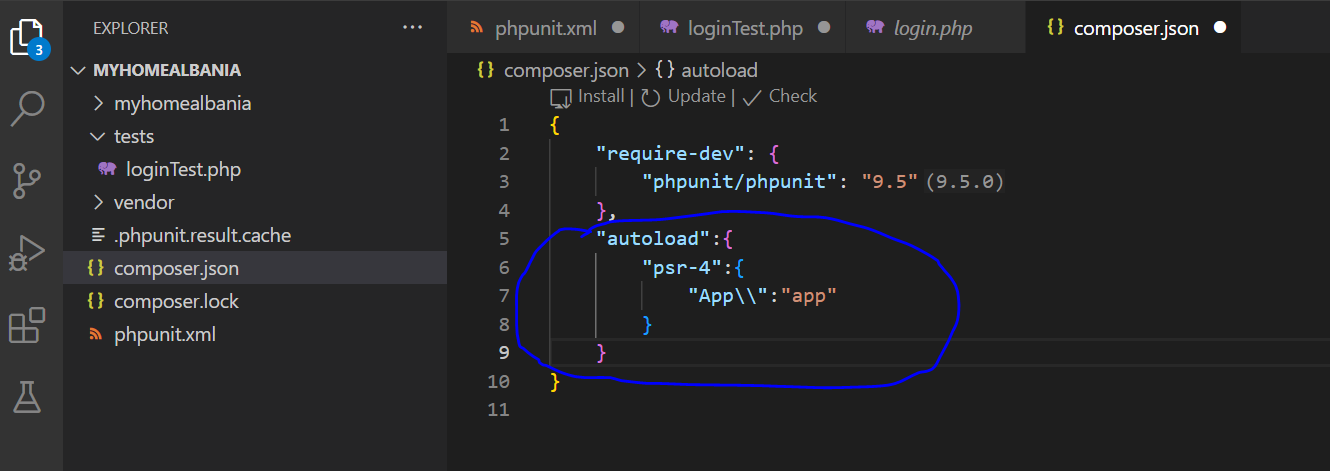
In this example, we've specified the bootstrap file (vendor/autoload.php), enabled colored output, and set stopOnFailure to false, which means PHPUnit will continue running tests even if one fails.You can add additional configuration settings as needed, such as specifying test directories, including or excluding specific files or directories, configuring code coverage settings, and more. Refer to the PHPUnit documentation for more information on available configuration options. By writing a PHPUnit configuration file, you can customize PHPUnit's behavior to suit your project's requirements and streamline the testing process.

4.Create Tests Folder and ….Test Class: In your project directory, create a folder

named tests. Inside the tests folder, create a new PHP file named ……Test.php. This filemwill contain your PHPUnit test class for testing the ……… class.



5.We have to modify composer json



**6.Writing Test Code:**

● Demonstrate how to write test code for the selected component, creating test methods to exercise different functionalities.

● Provide examples of **assertions** to validate expected outcomes.

Valerian: Failing test code:

<?php

use PHPUnit\Framework\TestCase;

class LoginTest extends TestCase

{

private function authenticate($username, $password)

{

$adminUsername = 'admin';

$adminPassword = 'admin';

return ($username === $adminUsername && $password === $adminPassword);

}

public function testSuccessfulLogin()

{

$this->assertTrue($this->authenticate('admin', 'password'));

}

}

Successful :

<?php

use PHPUnit\Framework\TestCase;

class LoginTest extends TestCase

{

private function authenticate($username, $password)

{

$adminUsername = 'admin';

$adminPassword = 'admin';

return ($username === $adminUsername && $password === $adminPassword);

}

public function testSuccessfulLogin()

{

$this->assertTrue($this->authenticate('admin', 'admin'));

}

}

**Erina:**

Here's an explanation of the test case:

· setUp() method: This method is called before each test method. It sets up the initial user information for testing.

· $this->userInfo: This variable stores the user information, including name, email, and password.

· editInformation($name, $email) method: This method simulates editing user information by updating the $userInfo array with the new name and email.

· testSuccessfulEditInformation() method: This method tests the successful editing of user information. It calls the editInformation() method to update the user information with new values. Then, it asserts that the user's name and email have been updated correctly.

· $this->assertEquals('Erina Vladi', $this->userInfo['name']);: This assertion checks if the user's name has been updated to 'Erina Vladi' successfully.

· $this->assertEquals('vladi@example.com', $this->userInfo['email']);: This assertion checks if the user's email has been updated to 'vladi@example.com' successfully.

<?php

use PHPUnit\Framework\TestCase;

class UserTest extends TestCase

{

private $userInfo;

public function setUp(): void

{

// Simulate user information

$this->userInfo = [

'name' => 'Erina Vladi',

'email' => 'erina@example.com',

'password' => 'password'

];

}

private function editInformation($name, $email)

{

// Simulate editing user information

$this->userInfo['name'] = $name;

$this->userInfo['email'] = $email;

}

public function testSuccessfulEditInformation()

{

$this->editInformation('Erina Vladi', 'vladi@example.com');

$this->assertEquals('Erina Vladi', $this->userInfo['name']);

$this->assertEquals('vladi@example.com', $this->userInfo['email']);

}

}

?>

**Griselda:**

This PHPUnit test case is designed to test the database connection by simulating a connection failure. Here's an explanation of the test case:

· testConnectionFailure() method: This method tests the database connection failure scenario.

· define('TEST\_ENVIRONMENT', true);: This line defines a constant TEST\_ENVIRONMENT as true. This constant can be used in the included connect.php file to distinguish between the test and production environments.

· ob\_start(): This function turns on output buffering. It means that all output (e.g., echo statements) generated after this point will be stored in an output buffer instead of being sent to the browser.

· include 'connect.php';: This line includes the connect.php file, which contains the database connection logic.

· $output = ob\_get\_clean();: This line retrieves the contents of the output buffer and cleans (clears) the buffer.

· $this->assertStringContainsString('Failed to connect with database', $output);: This line asserts that the output generated by the connect.php file contains the string 'Failed to connect with database'. If the database connection fails, this string should be present in the output.

<?php

use PHPUnit\Framework\TestCase;

class DatabaseConnectionTest extends TestCase {

public function testConnectionFailure() {

define('TEST\_ENVIRONMENT', true);

// Redirect output to buffer

ob\_start();

include 'connect.php';

$output = ob\_get\_clean();

$this->assertStringContainsString('Failed to connect with database', $output);

}

}

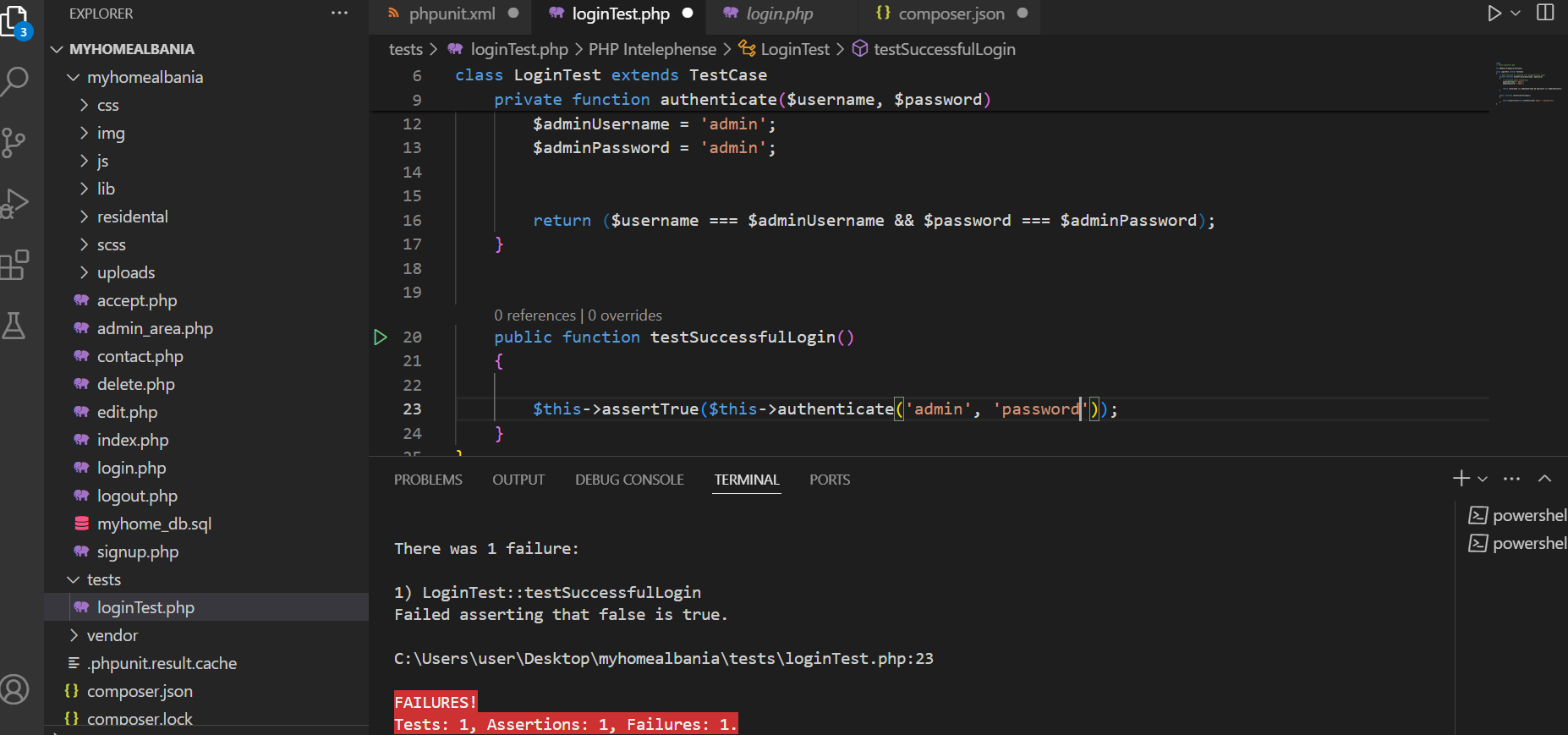
?>

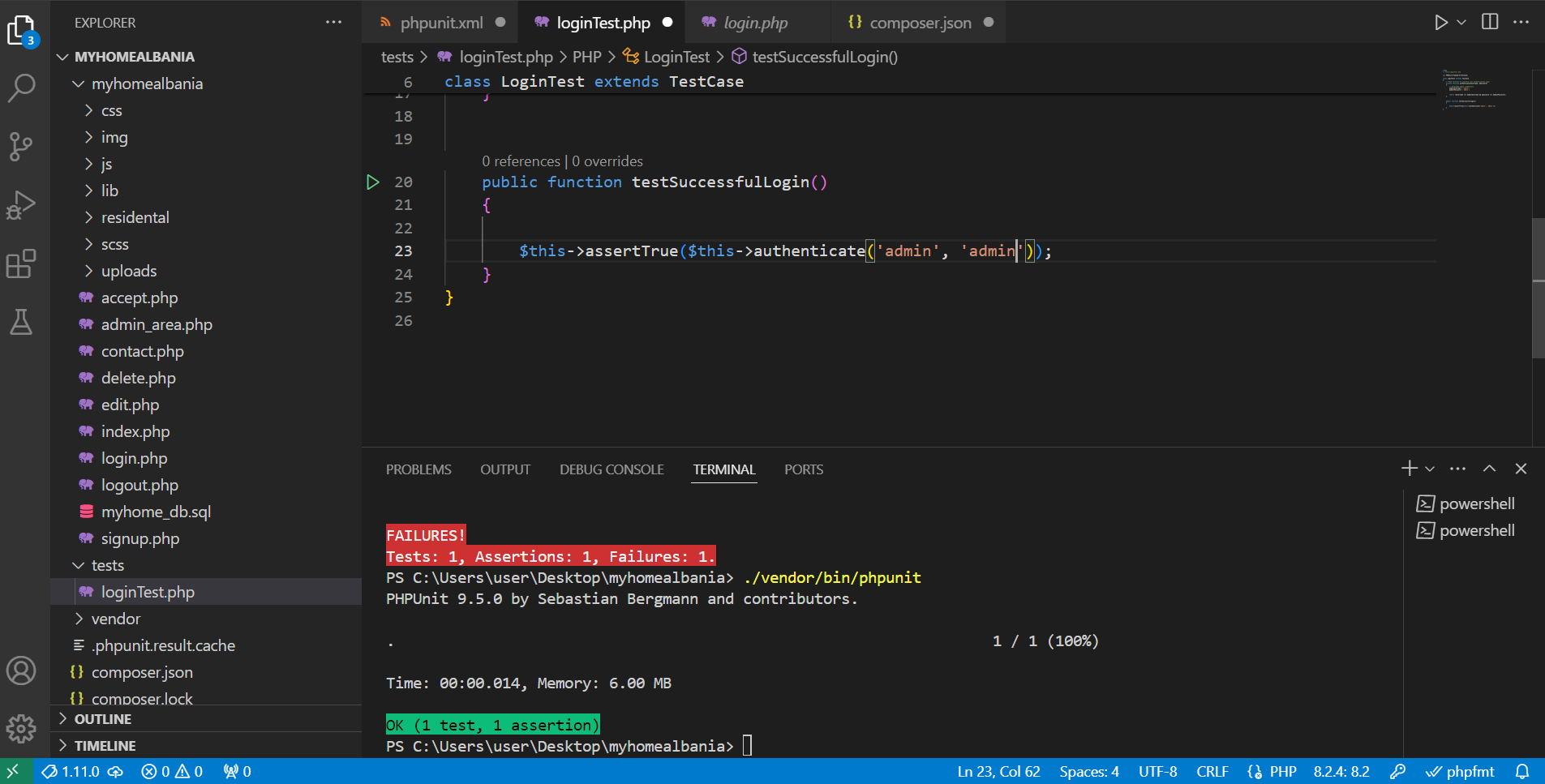
**7.Running Tests:**

**● Explain how to execute tests and interpret the results, covering passing, failing, and error scenarios.**

**Valerian:** This command will run PHPUnit and execute all test cases found in your project's tests directory. PHPUnit will automatically discover and execute all test classes and methods within those classes.

./vendor/bin/phpunit

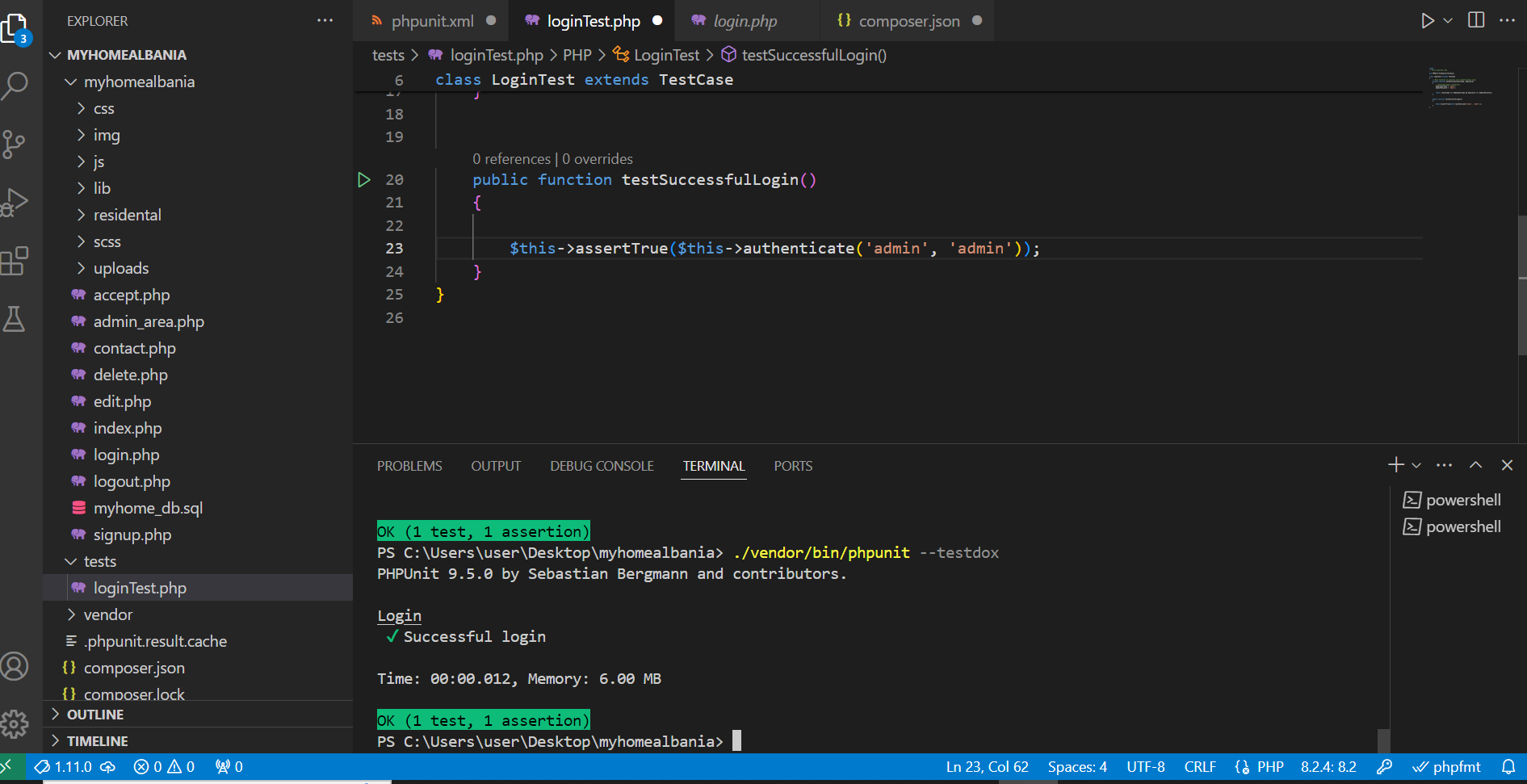
****

****

The --testdox option is used to generate human-readable test documentation. It's handy for providing an overview of your test suite and the scenarios covered by your tests. Here's how you can run PHPUnit with the --testdox option:

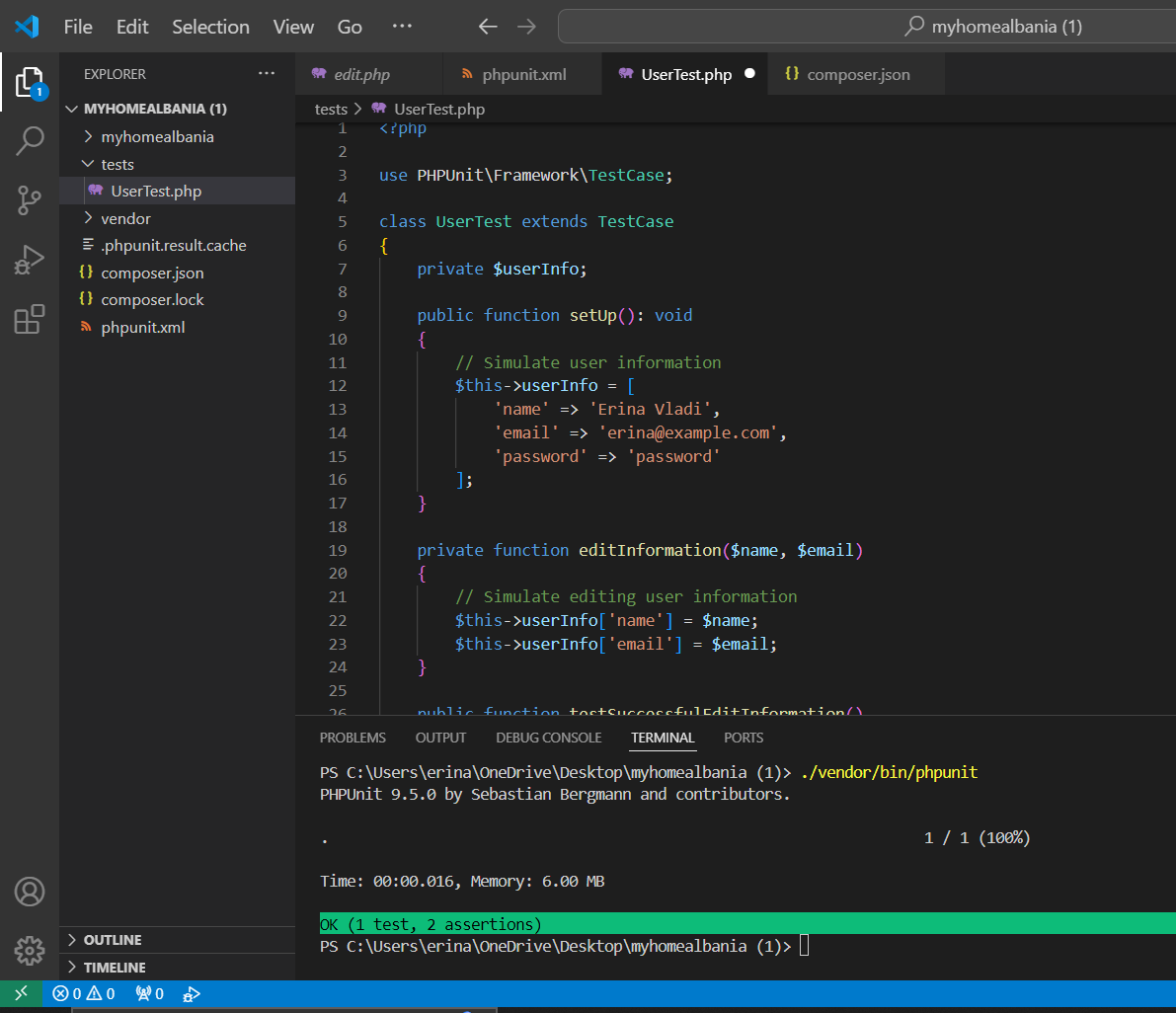
./vendor/bin/phpunit --testdox

When you execute this command in your terminal, PHPUnit will run your test suite and generate a human-readable report with test case names and their outcomes. This report will be displayed in your terminal, providing a clear overview of your tests and their results.

****

**Erina:**

Using the command ./vendor/bin/phpunit

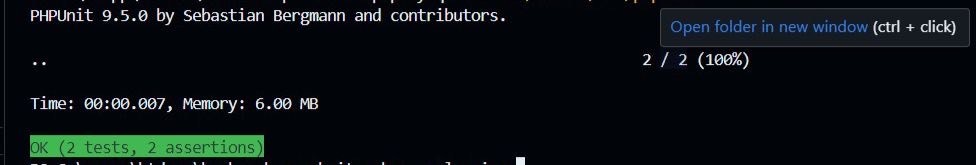


$this->assertEquals('Erina Vladi', $this->userInfo['name']);: This assertion checks if the user's name has been updated to 'Erina Vladi' successfully.

$this->assertEquals('vladi@example.com', $this->userInfo['email']);: This assertion checks if the user's email has been updated to 'vladi@example.com' successfully.

If both assertions pass, it indicates that the user information was edited successfully, and the test case will pass. If any assertion fails, the test case will fail, indicating that there was a problem with editing the user information.

**Griselda:**

****

Explanation:

The test case checks if the connection to the database succeeds, and it should generate a success message containing the text 'Connected successfully'.

If the database connection succeeds, the test case will pass. If the connection fails, the test case will fail, indicating that the database connection is not properly handled in the code.

Testing the successful case of the database connection ensures that the application can establish a connection with the database, enabling it to interact with the database and perform necessary operations, thus ensuring the proper functioning of the application.

**8.Test Coverage:**

**● Explain the importance of achieving high test coverage to ensure thorough testing of software.**

* Achieving high test coverage is paramount in software development as it ensures that a significant portion of the codebase is thoroughly tested. This means that every line of code, branch, and function is executed by the test suite, leaving minimal room for undetected defects.
* By attaining comprehensive test coverage, developers gain confidence in the reliability and correctness of their code. It provides assurance that various scenarios, including normal inputs, edge cases, and invalid inputs, have been adequately tested, thereby reducing the likelihood of bugs slipping into production.
* Moreover, high test coverage facilitates the process of refactoring and maintaining the codebase. When making changes to the code, developers can rely on the test suite to quickly identify any unintended side effects or regressions, ensuring that the system continues to function as intended.
* From a quality assurance perspective, achieving high test coverage is often a requirement for meeting industry standards and compliance regulations. It demonstrates a commitment to delivering high-quality software that meets the needs and expectations of users.