**Phase III: Software Testing**

**Deadline: May 6, 2024, 23:59**

**Software Testing**

**Group Name: Online Bookstore**

***Introduction to Testing:***

**Software Testing**:The systematic method of assessing a software application or system to find possible defects,bugs, or differences between intended and actual performance is known as software testing. It involves running the program under controlled circumstances in order to confirm its usability, security, performance, dependability, and functionality. Software testing's main objective is to find defects early in the development process in order to guarantee that end customers receive a high-quality product. Testing can take several forms: from unit testing, which verifies the functionality of individual software components, to system testing, which evaluates the behavior of the complete system under realistic conditions. Numerous testing methods, approaches, and instruments are utilized to guarantee thorough test coverage and reduce the possibility of software failures in real-world settings.

**Importance of Testing:**Software testing is highly important in ensuring in software development reliability and correctness.Some ways in which software testing helps to achieve these objectives are:

1. **Identifying Defects Early**: Early in the software development lifecycle, testing aids in finding errors and problems in the product. When issues are found early on in the process, they may be fixed more cheaply and with less work than if they are found later on during production or after deployment.
2. **Enhancing Software Quality**: Software quality can be greatly improved by conducting extensive testing. Testing ensures that the finished product fulfills the necessary standards and criteria by verifying the software's functionality, performance, security, and usability.
3. **Improving User Experience**: Testing makes sure that the program operates as intended and offers a satisfying user experience. Testing aids in the creation of user-friendly and intuitive software interfaces by locating and resolving bugs, usability difficulties, and inconsistencies.
4. **Building trust and confidence**: Users, stakeholders, and customers place greater trust and confidence in software that is dependable and free of bugs. Testing gives users confidence in the software's functioning and performance by showcasing its correctness, stability, and dependability.
5. **Reducing Risks and Failures**: Testing assists in lowering the risks related to malfunctions and failures in software. Testing lowers the possibility of system crashes, data loss, security breaches, and other undesirable outcomes by locating and fixing potential flaws and vulnerabilities.
6. **Fulfilling Requirements and Expectations**: Testing guarantees that the program satisfies the objectives and stakeholder and end-user expectations as stated. Testing helps to provide a solution that meets customer wants and meets business goals by validating the program against set criteria and user needs.
7. **Facilitate continuous improvement**:Testing provides insightful feedback and useful insights into the software development process.
8. **Respecting Standards and Regulations**: Compliance to standards, regulations, and best practices is crucial in numerous businesses. Testing ensures that the program complies with applicable laws and regulations, including those pertaining to security, accessibility, and industry-specific rules.

***Purpose of Testing:***

The goal of testing is to find bugs, errors, or defects in software at the earliest stage of its creation. Early detection of these problems allows developers to fix them more quickly and for less money. The chance of problems spreading across the system and becoming more difficult and expensive to solve later in the development process or after deployment is reduced by early defect discovery.

Testing makes sure that systems, modules, or software operate in accordance with the requirements and specifications that were intended. It guarantees that the program operates as intended in a range of settings and circumstances. This entails verifying non-functional needs (like performance, dependability, and security) and any other software-defined criteria in addition to functional requirements (like features, functionality, and user interactions).

Testing assists in preventing defects from turning into more serious problems or system failures by detecting errors early on and confirming the accuracy of software components. Early defect detection enables developers to locate and fix problems before they affect other system components or jeopardize the software's overall quality and dependability.

Throughout the software development lifecycle, testing is an iterative process that involves regularly confirming and validating software components. Tests are carried out by developers when they write code to make sure it satisfies requirements and operates as intended. Developers improve their code, find areas for improvement, and progressively raise the software's quality and dependability through iterative testing.

One of the most important aspects of risk mitigation in software development is testing. Developers can find and fix possible risks, vulnerabilities, and issues early in the process by methodically testing software components. By using a proactive approach to risk management, software solutions are delivered that are more stable, dependable, and resistant to possible problems or failures.

***Testing a Single Component:***

The component we chose for testing is the Shopping Cart. Testing this component is important for a lot of reasons.

* **Essential Functionality:** The shopping cart component is crucial to the entire online shopping system. It's where people save products they decide and plan to buy. Any issue within the shopping cart might have a direct impact on sales and the customer experience.
* **Complexity:** Although the idea of a shopping cart appears simple, its execution may be quite challenging. It involves managing a variety of tasks such as adding and deleting books, adjusting quantities, applying shipping fees, and connecting with payment systems. Testing guarantees that all of these features operate together effectively.
* **User Experience:** A smooth and easy shopping cart experience is crucial to user satisfaction. Testing assists in the identification of any usability errors, like unclear instructions, complex designs, or unpredictable behavior, which may cause discomfort to users and cause them to leave their items on the shopping cart.
* **Data Integrity:** The shopping cart component communicates with user data, such as product details and prices. Testing makes sure that data is successfully recorded, updated, and preserved throughout the shopping experience, reducing mistakes such as incorrect prices or missing items.
* **Compatibility:** The shopping cart must function properly whether you are using a phone, PC, or tablet. Testing ensures that it functions properly on a variety of devices.
* **Security:** Because the shopping cart contains sensitive information such as user passwords and payment information, security is critical. Testing serves to guarantee that everything is secure and your personal information is secured.

Testing the shopping cart is important to ensure that it runs properly, protects your information, and makes it easy for everyone to buy books online.

We can make sure that it not only performs its basic functions reliably but also provides a seamless and secure shopping experience for users, ultimately contributing to the success of the online bookstore system.

***Preparing Test Cases:***

We need to take into account multiple scenarios for each function being tested, such as AddToCart(), GetBookDetails, UpdateQuantity, UpdateSubtotal(), and testUpdateTotal(), in order to construct test cases covering a variety of possibilities for the specified tests, including normal inputs, edge cases, and incorrect inputs:

1. testAddToCartSuccess()

* Normal Input: Check the behaviour when a valid book\_id is provided.
* Edge Case: Determine the behaviour when add\_to\_cart is not set to true.
* I nvalid Input: Check the behaviour when the book\_id is invalid.

1. testGetBookDetails()

* Normal Input: Test the behavior when a valid book\_id is entered and the book exists in the database.
* Edge Case: Evaluate the behaviour when the specified book\_id does not exist in the database.
* Invalid Input: Check the behaviour when book\_id is not a valid integer.

1. testUpdateQuantity()

* Normal Input: Test the behaviour when a valid book\_id and quantity are entered.
* Edge Case: Determine the behaviour when the current quantity is zero.
* Invalid Input: Test the behaviour where book\_id is not provided or is not a valid integer, and quantity is not a valid

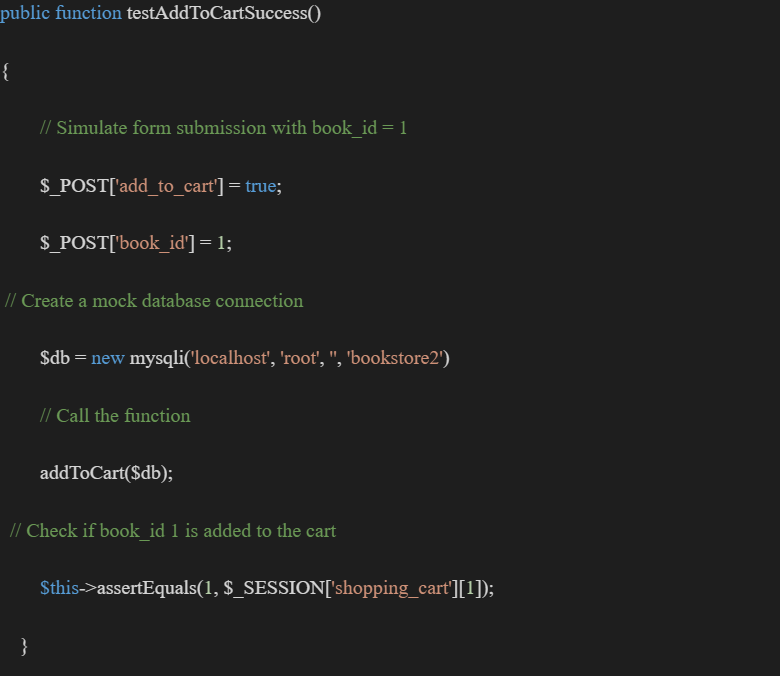
1. testUpdateSubtotal()

* Normal Input: Test the behaviour when a valid book\_id and price are entered.
* Edge Case: Determine the behaviour when the current subtotal is zero.
* Invalid Input: Determine the behaviour when book\_id is not provided or is not a valid integer, as well as when price is not a valid integer.

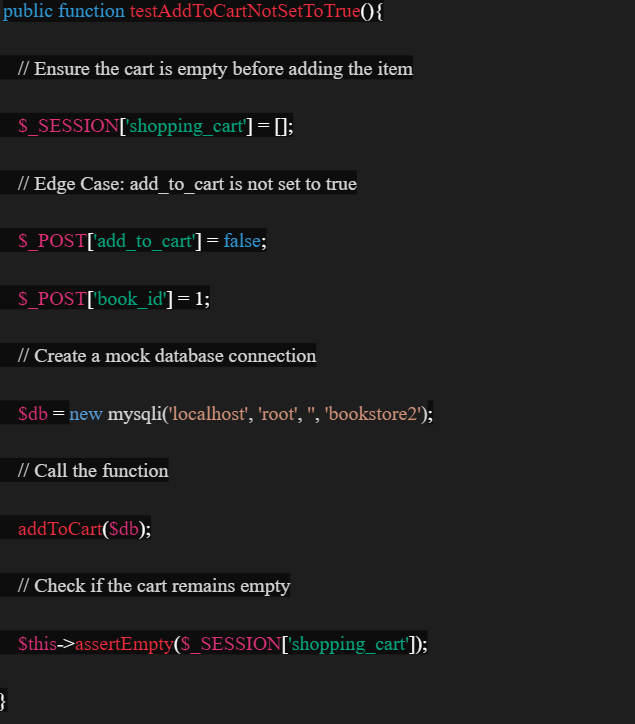
1. testUpdateTotal()

* Normal Input: Check the behaviour when a valid subtotal and shipping cost are entered.
* Edge Case: Determine the behaviour when the current total is zero.
* Invalid Input: Evaluate the behaviour when the subtotal or shipping cost is not a valid number.
* *AddToCart() Test Cases*

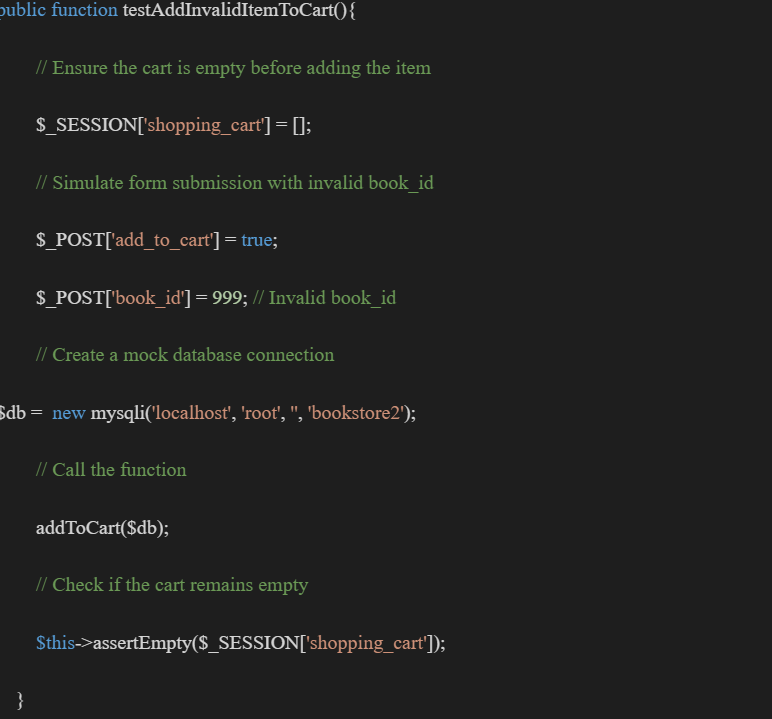
1. Normal Case:



1. Edge Case:



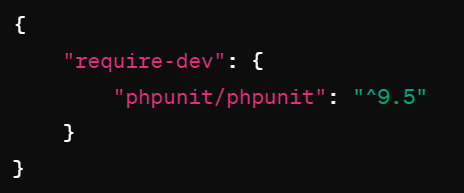
1. Invalid input:



***Choosing Testing Frameworks***

For our online bookstore,we have chosen **PHPUnit** as our testing framework. It is widely used and offers strong support for integration and unit testing. Here's how to set up a testing environment using PHPUnit.

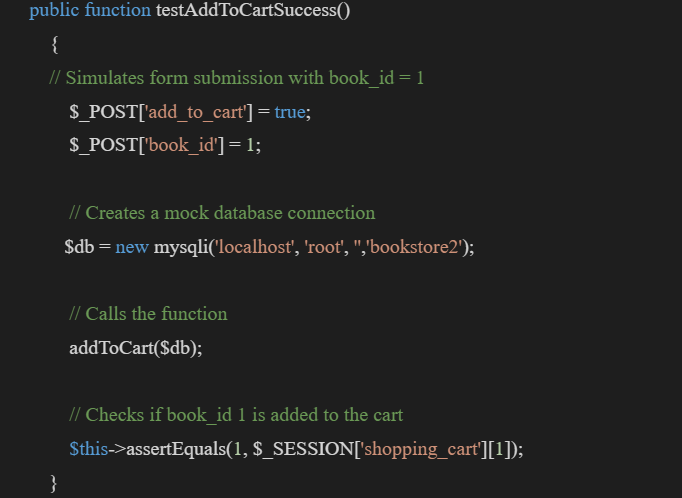
1. Installation: First we make sure that we have to install the Composer on our system which is basically a dependency manager for PHP and makes the installation of PHPunit easy.
2. The next step after it has been installed, is creating a composer.json file in our project directory for managing our project’s dependencies.
3. Then require PHPunit which is done by adding the PHPunit as a development dependency inside the composer.json file.

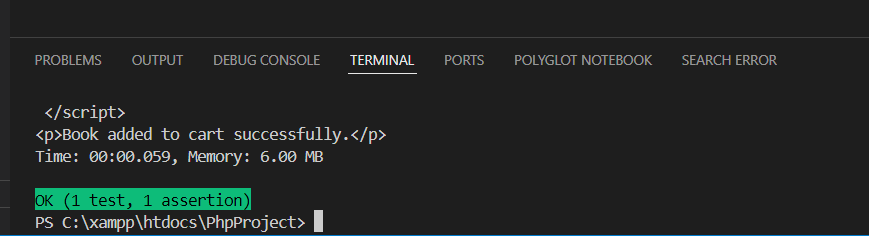


1. Now we install the dependencies by running this command “composer install” to install the PHPunit.
2. The next step is to write the tests. We write the tests for our shopping cart component here. We can create a dictionary named tests to store our test files. They can be organized based on the components or features we are testing.
3. Run tests. To run our PHPunit tests we need to run: “vendor/bin/phpunit”. This command executes all the tests in our project.

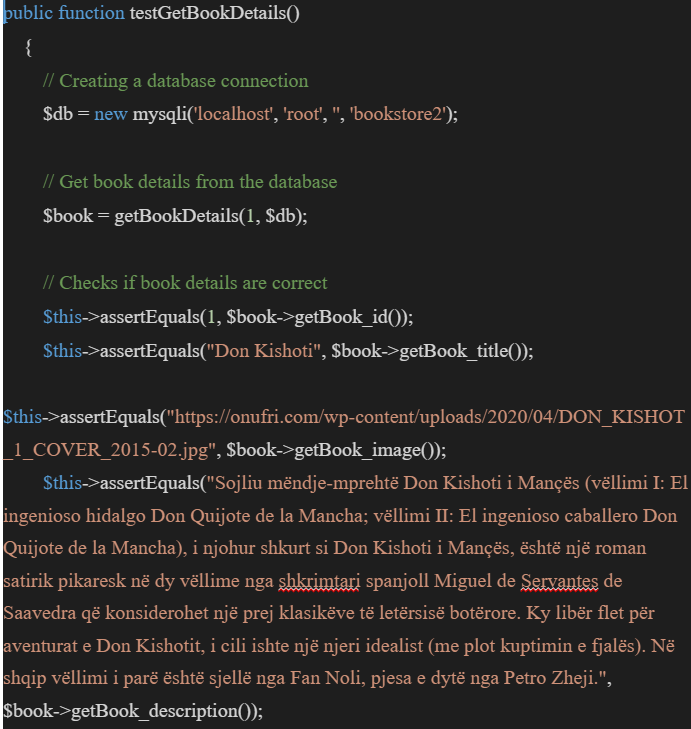
***Writing Test Code:***

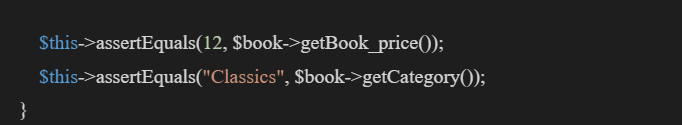
* Test ensures that the addToCart() function successfully adds a book to the shopping cart

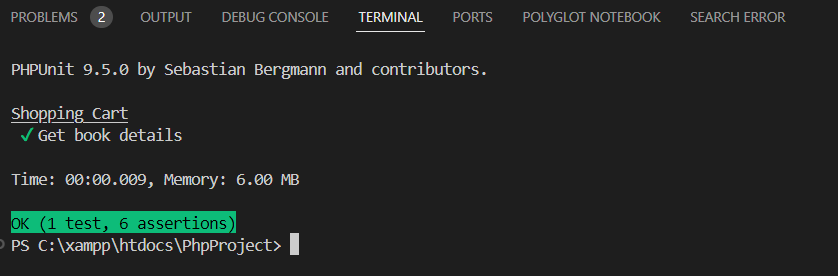




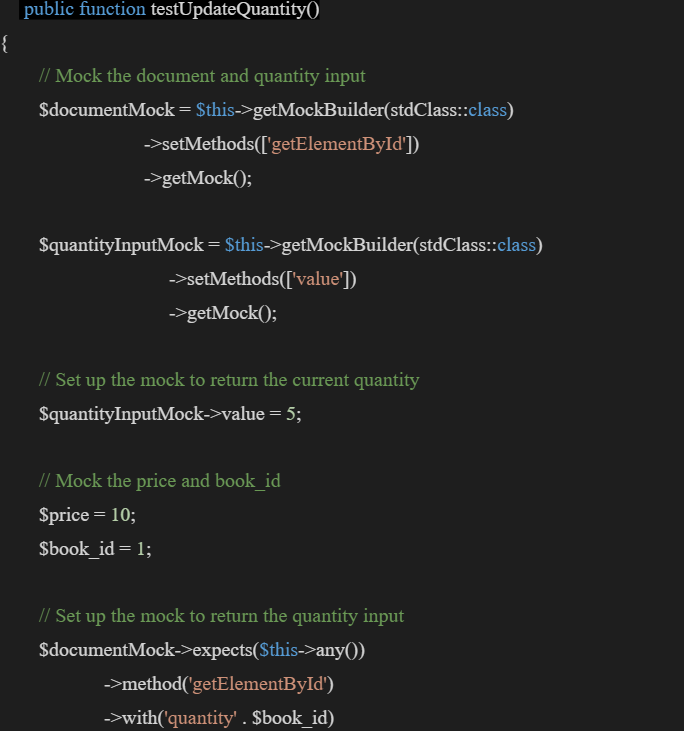
* Test ensures that the getBookDetails() function retrieves the correct book details from the database and returns them as expected.

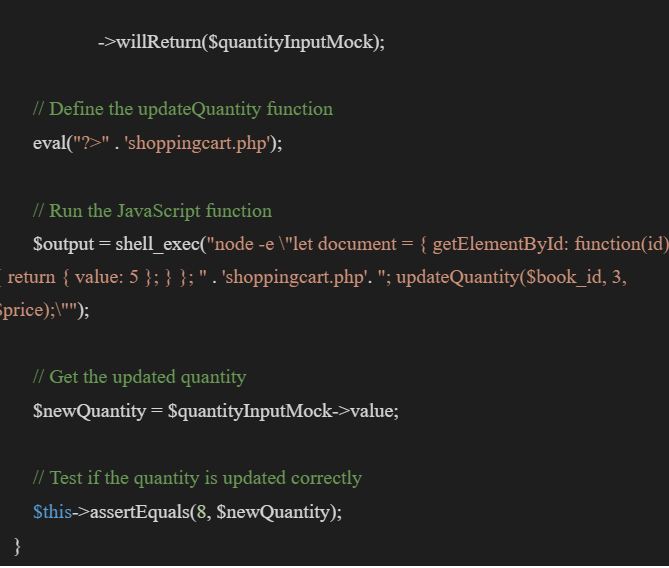


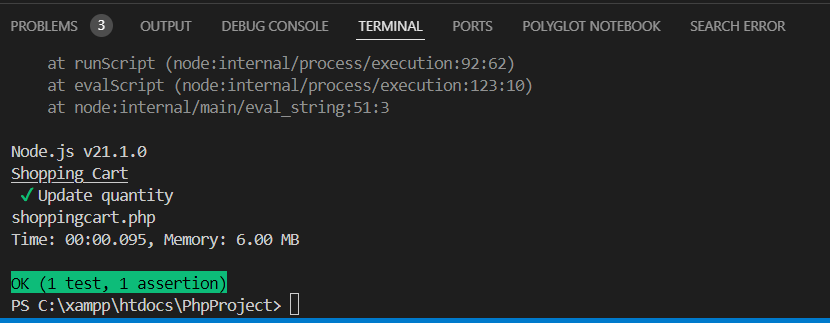




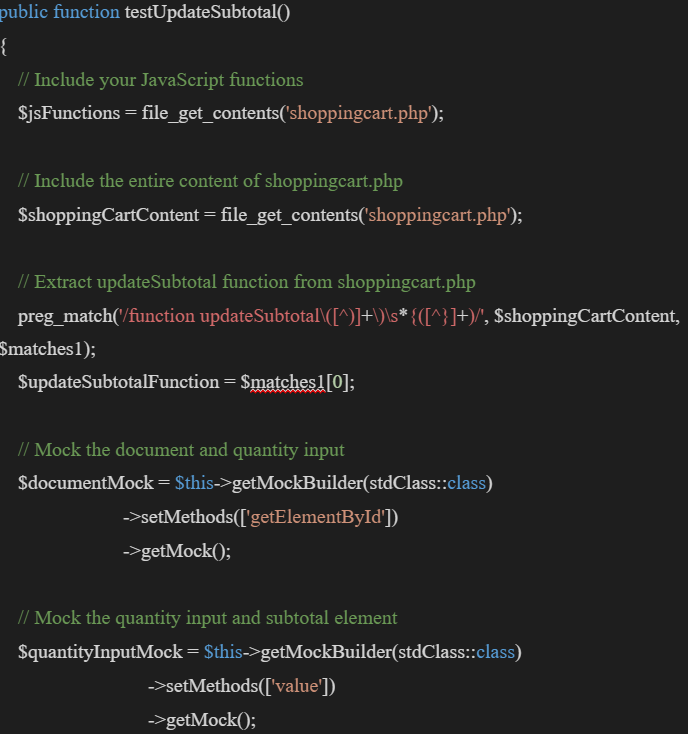
* Tests the JavaScript function updateQuantity(),ensures that it updates the quantity correctly

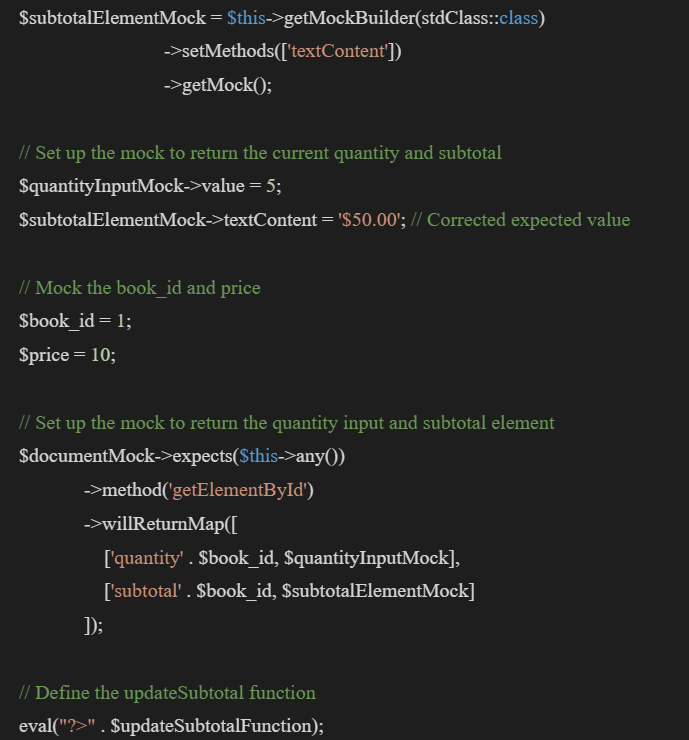


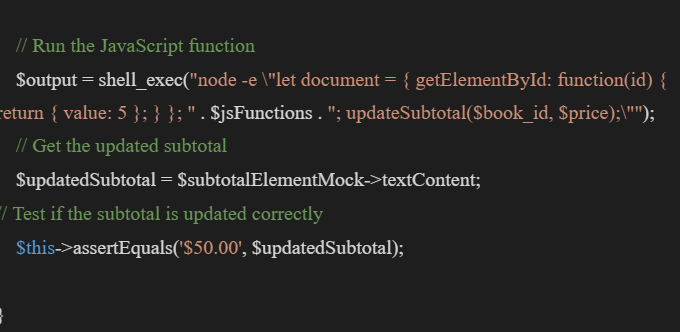


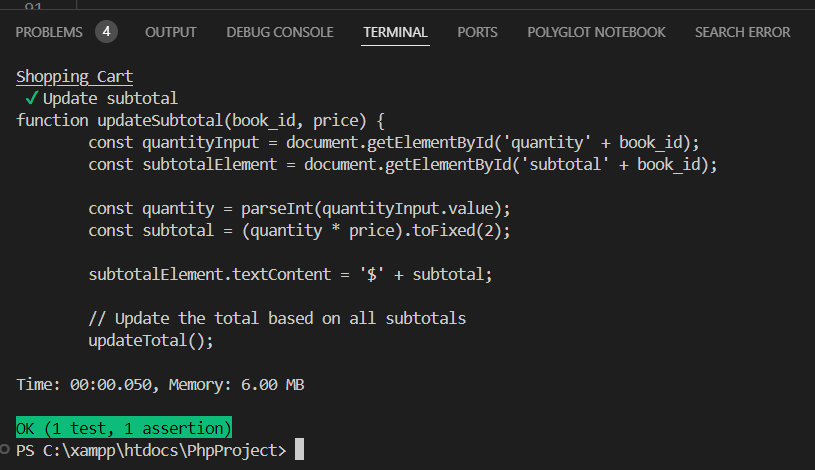


* Test ensures that the updateSubtotal function updates the subtotal correctly

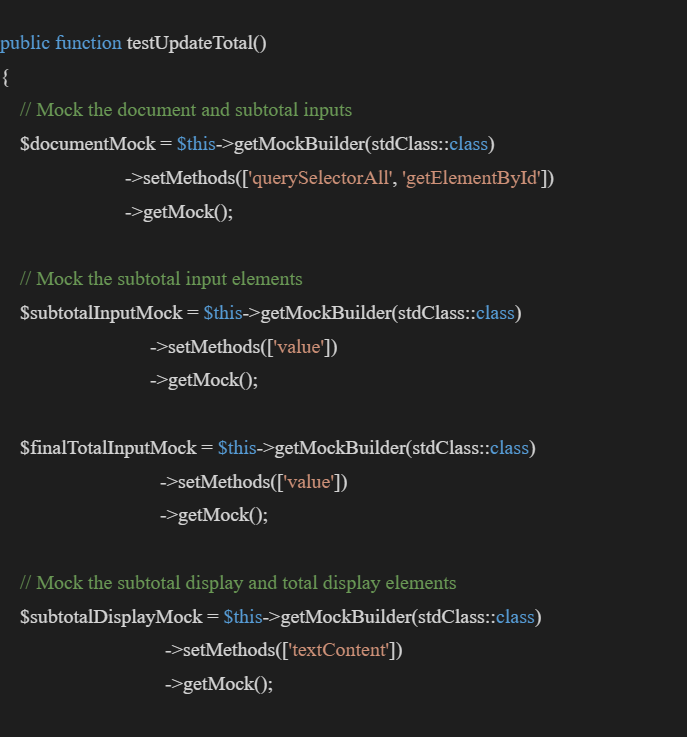




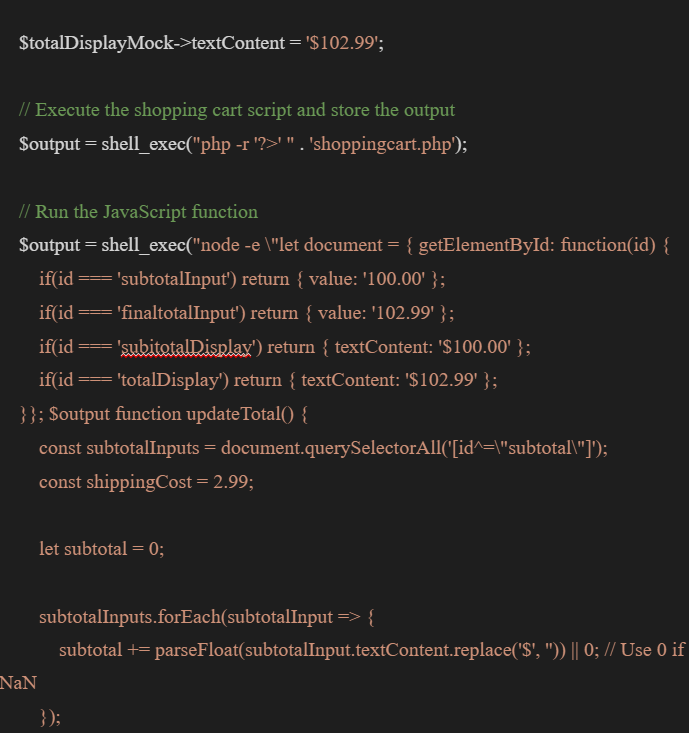


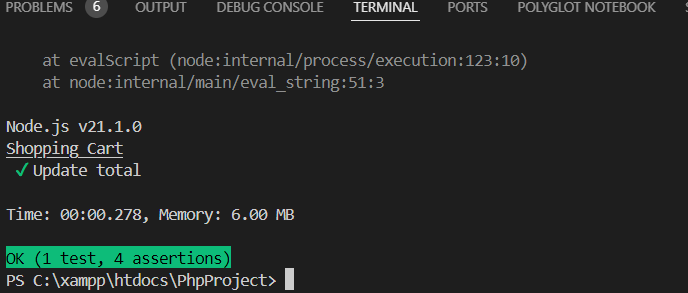


* Test ensures that the updateTotal function updates the total correctly









***Running Tests:***

* **Execute the tests**

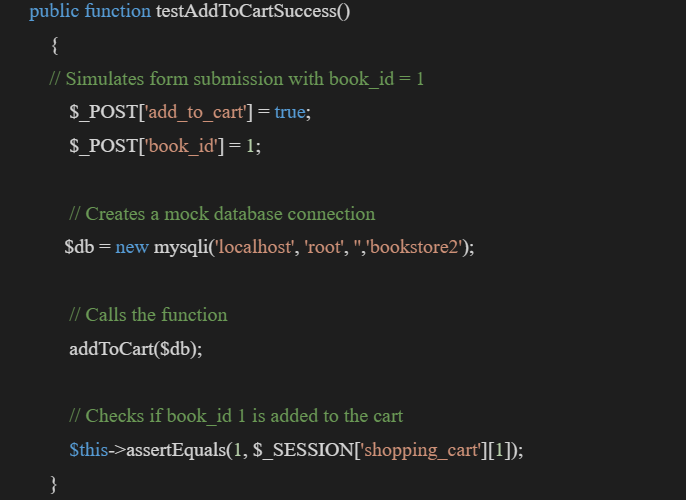
To execute test for the shopping cart component using PHPunit we use the following command:

./vendor/bin/phpunit --testdox

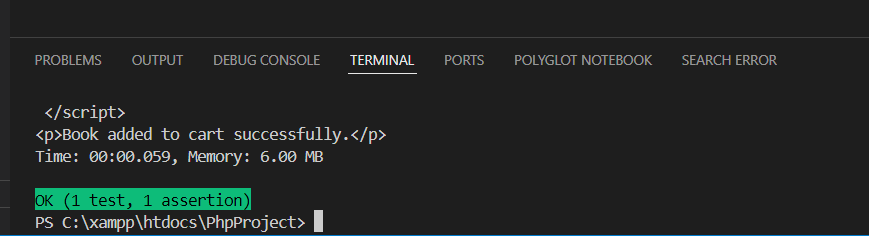
* **Interpret the results**

#### Passing Scenarios:

testAddToCartSuccess: Tests if the addToCart() function successfully adds a book to the shopping cart.

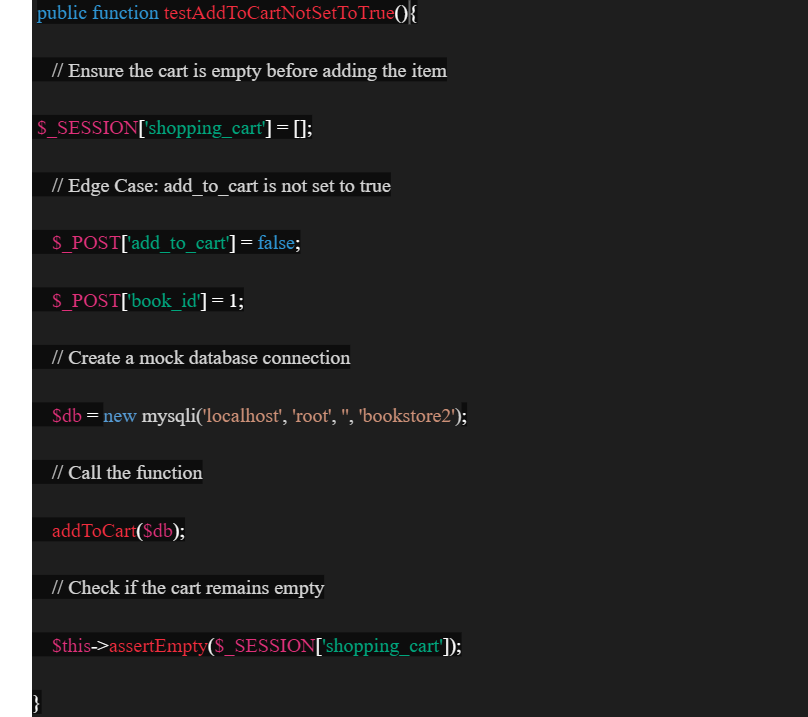


* The function passes the test successfully meaning the book is added to cart.

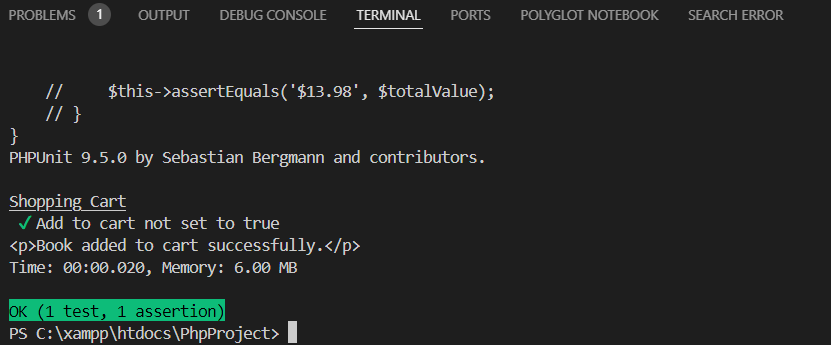


#### Failing Scenarios:

testAddToCartNotSetToTrue():If the test testAddToCartNotSetToTrue() fails, it means that the addToCart() function adds an item to the cart even when the add\_to\_cart parameter is not set to true.



* The function passes the test successfully meaning the book is not added to the cart when add\_to\_cart parameter is set to false.



***Test Coverage:***

Having a high test coverage is really important for various reasons listed below:

1. **Finding Uncovered Code Paths**: Test coverage estimates the amount of testing that has done its job on the codebase. A high level of test coverage guarantees that most, if not all, code paths are run in the testing process. This aids in locating untested or infrequently used code routes that might include flaws, logical mistakes, or vulnerabilities.
2. **Finding errors or defects**: Identifying errors, bugs, or flaws in the program is made more likely by extensive testing with good coverage. Tests are able to reveal problems that may not be noticeable during the initial stages of development or inspection by experimenting with various code pathways and circumstances. Early defect detection lowers the cost and work involved in addressing problems later in the development process.
3. **Validation of Requirements**: Extensive testing ensures that the program satisfies its requirements and operates as planned. Tests verify that the program operates as intended in a variety of scenarios and functional and non-functional criteria. This helps in confirming that the program meets the requirements and expectations of the user.
4. **Improved Trust in Software Quality**: A large test coverage raises trust in the software's dependability and quality. The majority of the code has been rigorously tested, which lowers the possibility that unidentified flaws may find their way into production. It gives developers, testers, stakeholders, and end users confidence that the program has undergone extensive testing and is unlikely to have any serious problems.
5. **Increased Refactoring and Maintainability**: Software refactoring and maintainability are made easier with high test coverage. Comprehensive test suites act as a safety net for developers who modify or replace code, ensuring that changes don't bring unwanted side effects or regressions and that functionality stays intact. This encourages code agility, giving engineers the confidence to make changes.
6. **Observance of Regulations and Auditability**: Regulatory compliance in some sectors or domains necessitates extensive testing and test coverage documentation. A high test coverage rate indicates thorough testing procedures as well as the dependability and quality of the software. By making the scope of the testing visible, it also helps with auditability.
7. **Continuous integration and deployment(CI/CD)**:High test coverage is necessary for the effective implementation of pipelines for continuous integration and deployment. The danger of introducing defects or regressions is decreased when code changes are extensively tested before being sent to production settings thanks to automated tests with high coverage.