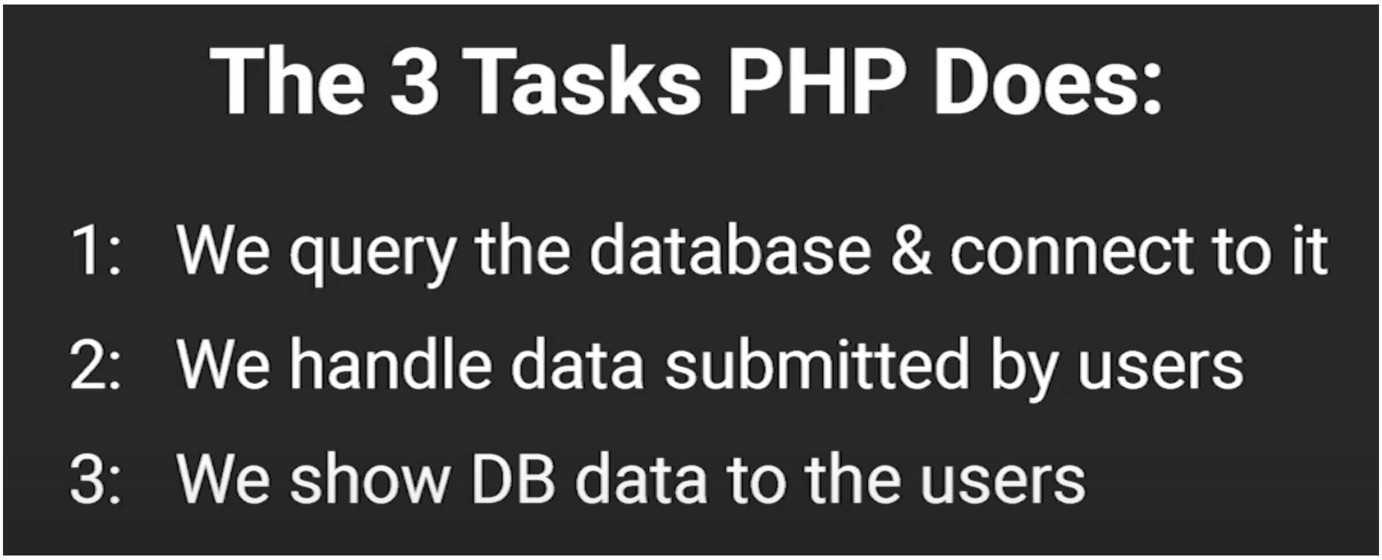
My part of project Explanation

Important Thoeries to cover:
1:



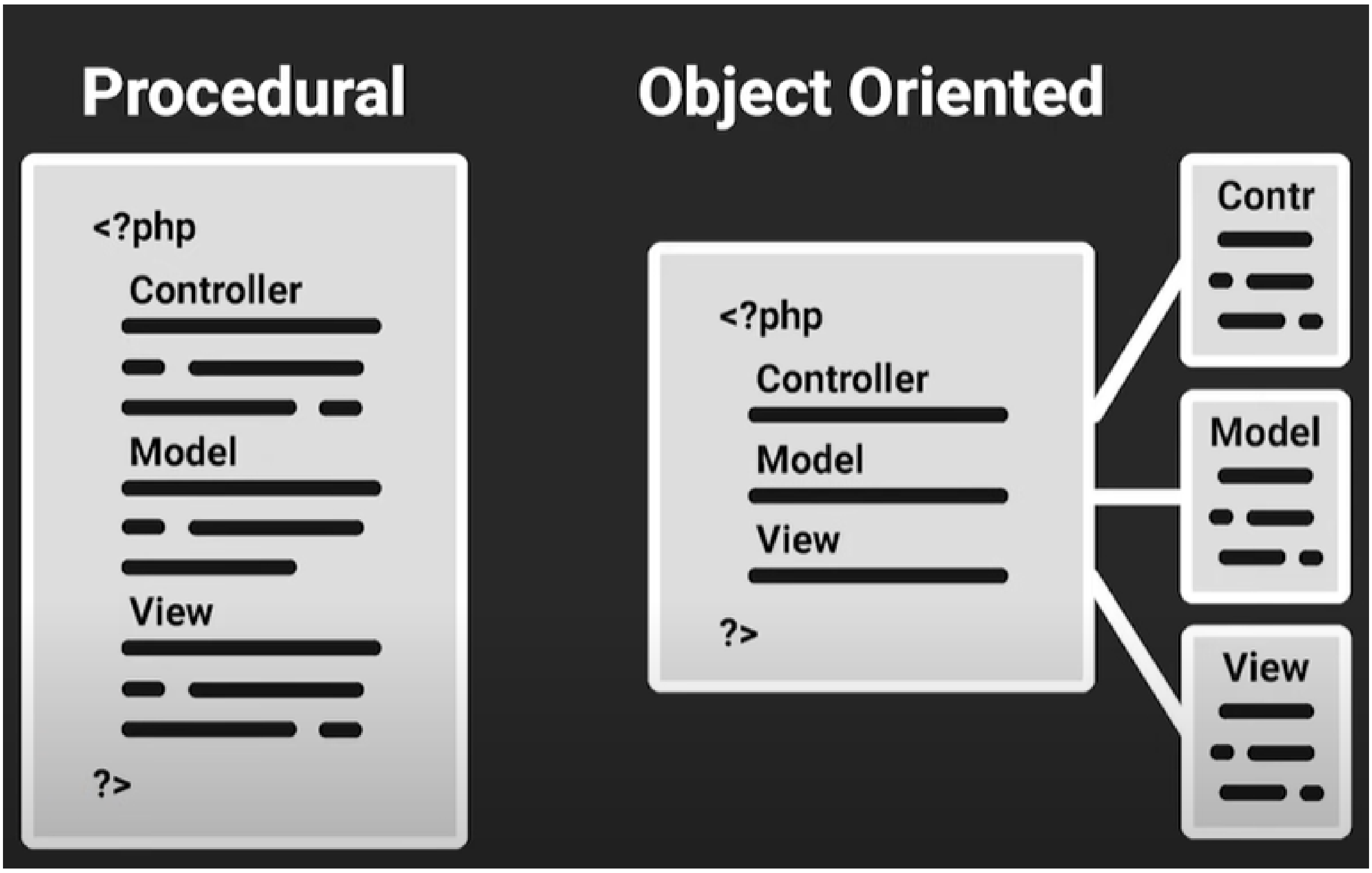
The 3 Tasks PHP Does:

1: We query the database & connect to it

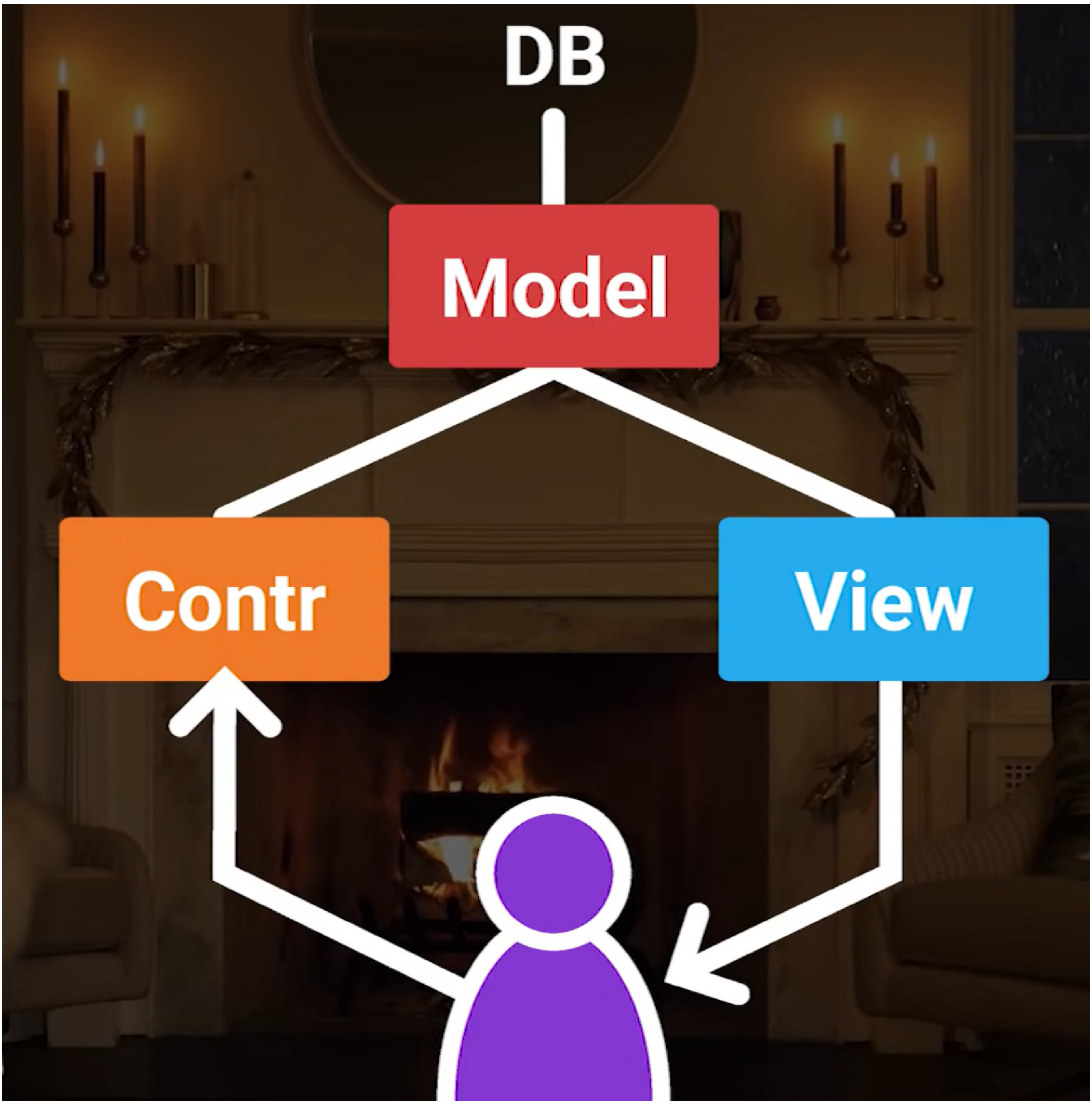
2: We handle data submitted by users

3: We show DB data to the users

2: Know the difference:
Reference[1]: [https://www.youtube.com/watch?v=3OKOe7CraGY]

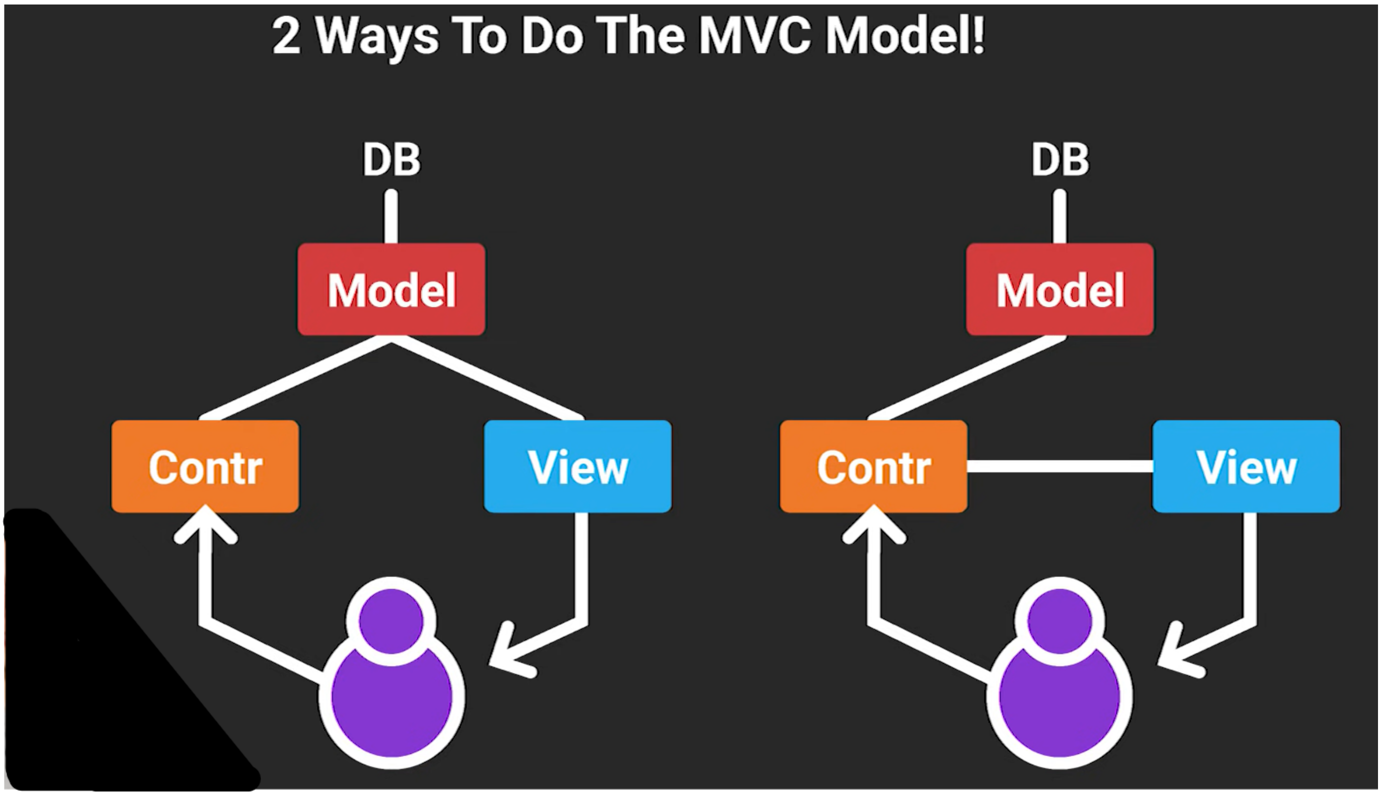


3: MVC = C-M-V, In Simple Terms [Brain-Memory-Visual]



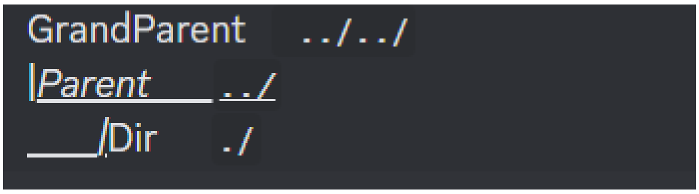
[!!->For MID we are using [JSON] as the database(not real database). And DATABASE will be made later [in the final term] using: MySQL.

[2-Options]



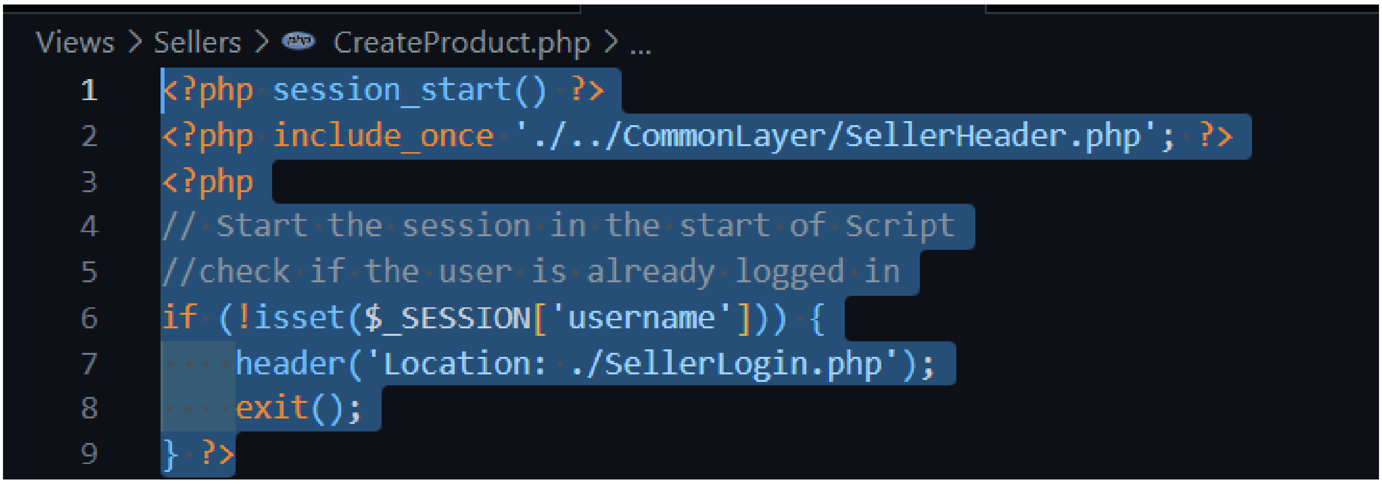
[redacted]

4: [Directory Analogy Explained]



```
GrandParent   ../../
|Parent___ ../
___|Dir    ./
```

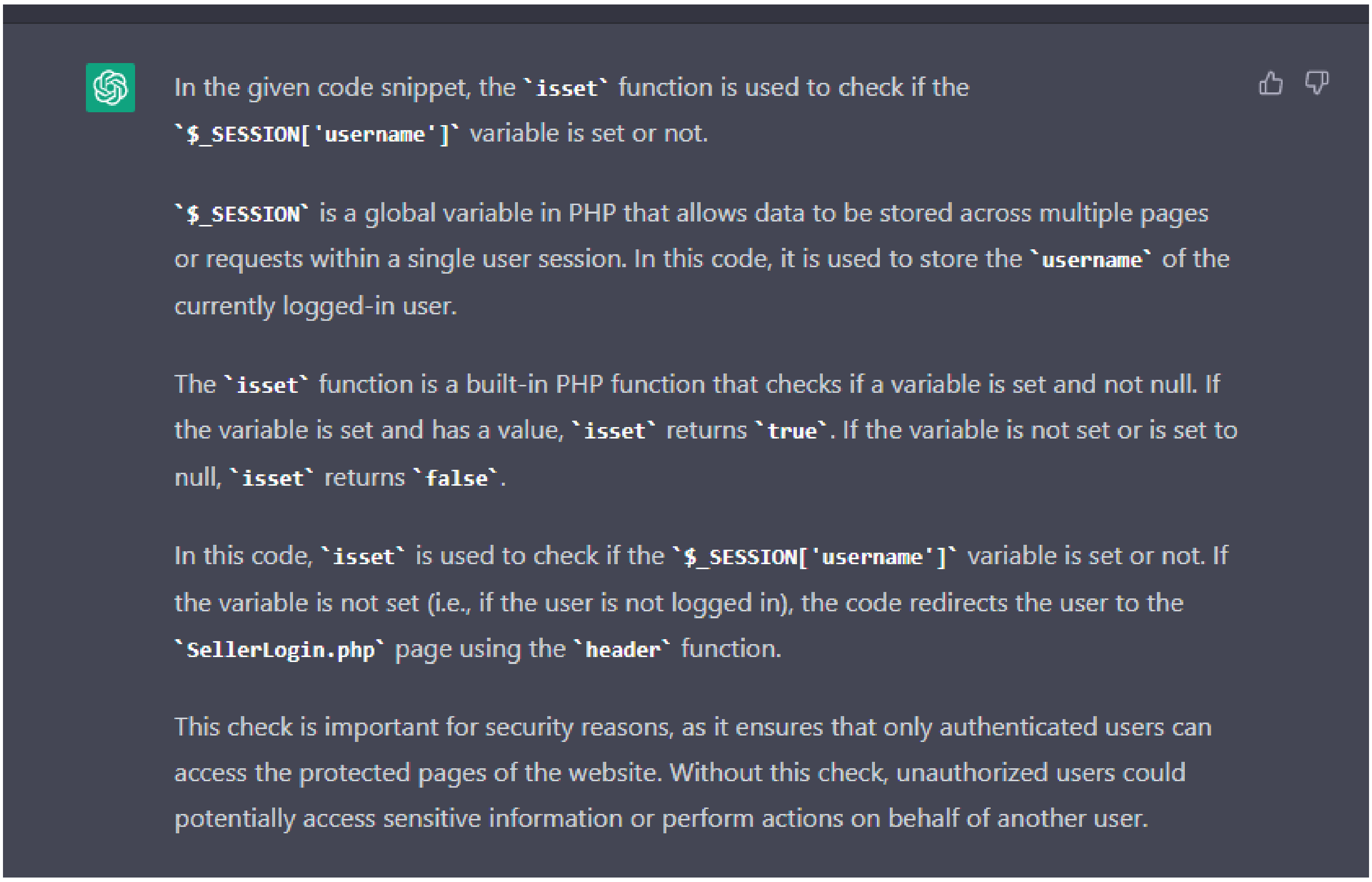
5: [Simple Code Snippet Analysis]



Views > Sellers > CreateProduct.php > ...

```php
<?php session_start() ?>
<?php include_once './../CommonLayer/SellerHeader.php'; ?>
<?php
// Start the session in the start of Script
//check if the user is already logged in
if (!isset($_SESSION['username'])) {
    header('Location: ./SellerLogin.php');
    exit();
} ?>
```

Analysis:



In the given code snippet, the `isset` function is used to check if the `$_SESSION['username']` variable is set or not.

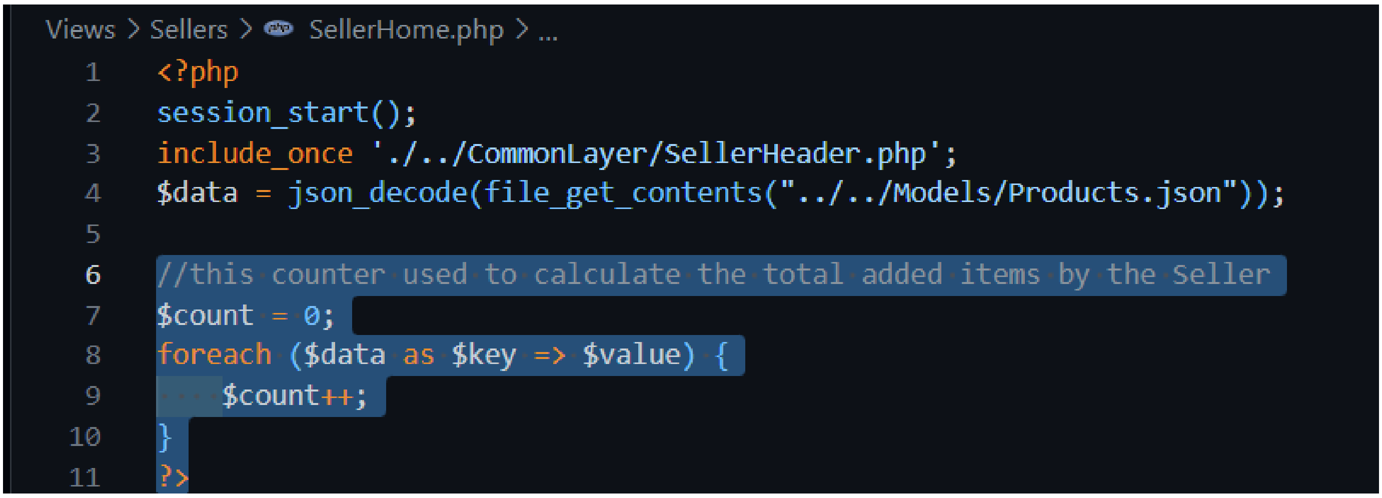
`$_SESSION` is a global variable in PHP that allows data to be stored across multiple pages or requests within a single user session. In this code, it is used to store the `username` of the currently logged-in user.

The `isset` function is a built-in PHP function that checks if a variable is set and not null. If the variable is set and has a value, `isset` returns `true`. If the variable is not set or is set to null, `isset` returns `false`.

In this code, `isset` is used to check if the `$_SESSION['username']` variable is set or not. If the variable is not set (i.e., if the user is not logged in), the code redirects the user to the `SellerLogin.php` page using the `header` function.

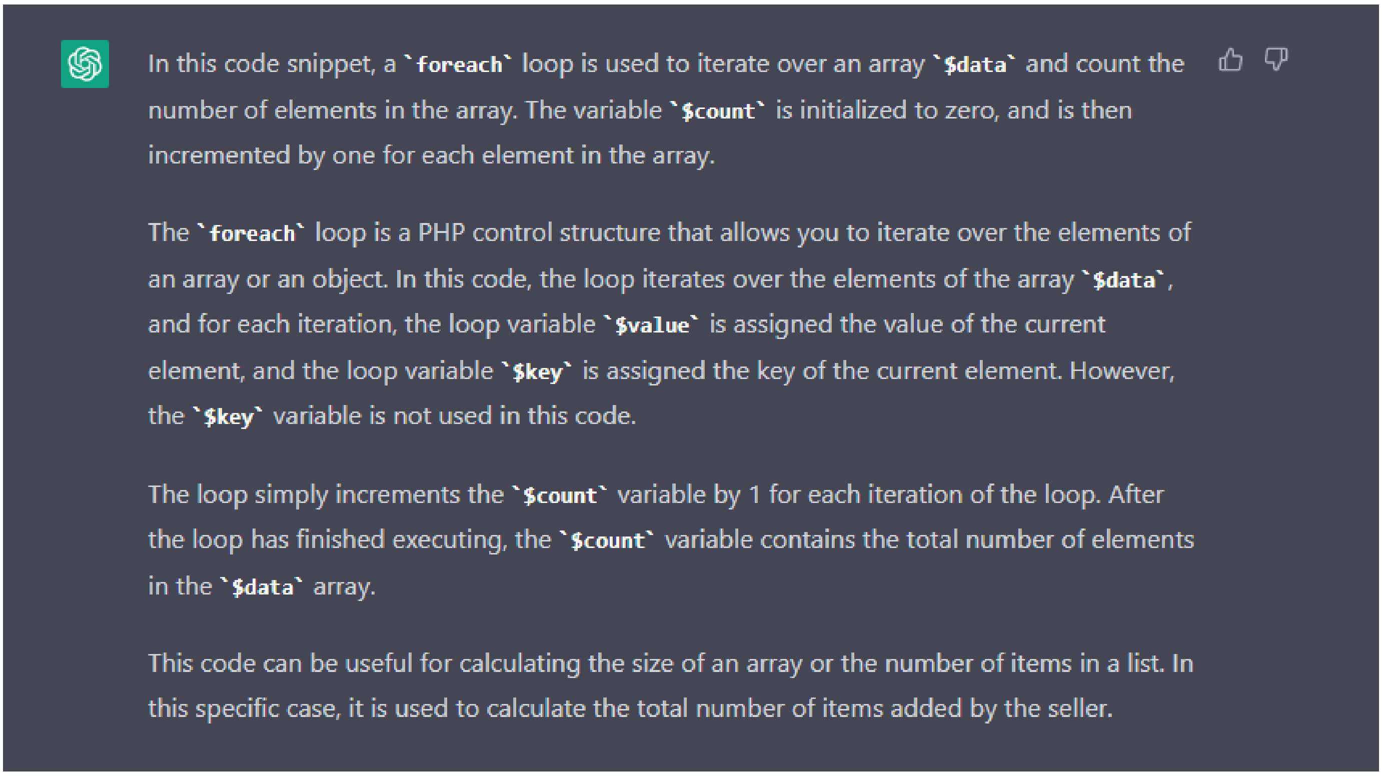
This check is important for security reasons, as it ensures that only authenticated users can access the protected pages of the website. Without this check, unauthorized users could potentially access sensitive information or perform actions on behalf of another user.

[Added Items By Seller Code Snippet Analysis]



```php
<?php
session_start();
include_once './../CommonLayer/SellerHeader.php';
$data = json_decode(file_get_contents("../../Models/Products.json"));

//this counter used to calculate the total added items by the Seller
$count = 0;
foreach ($data as $key => $value) {
    $count++;
}
?>
```

Explanation:



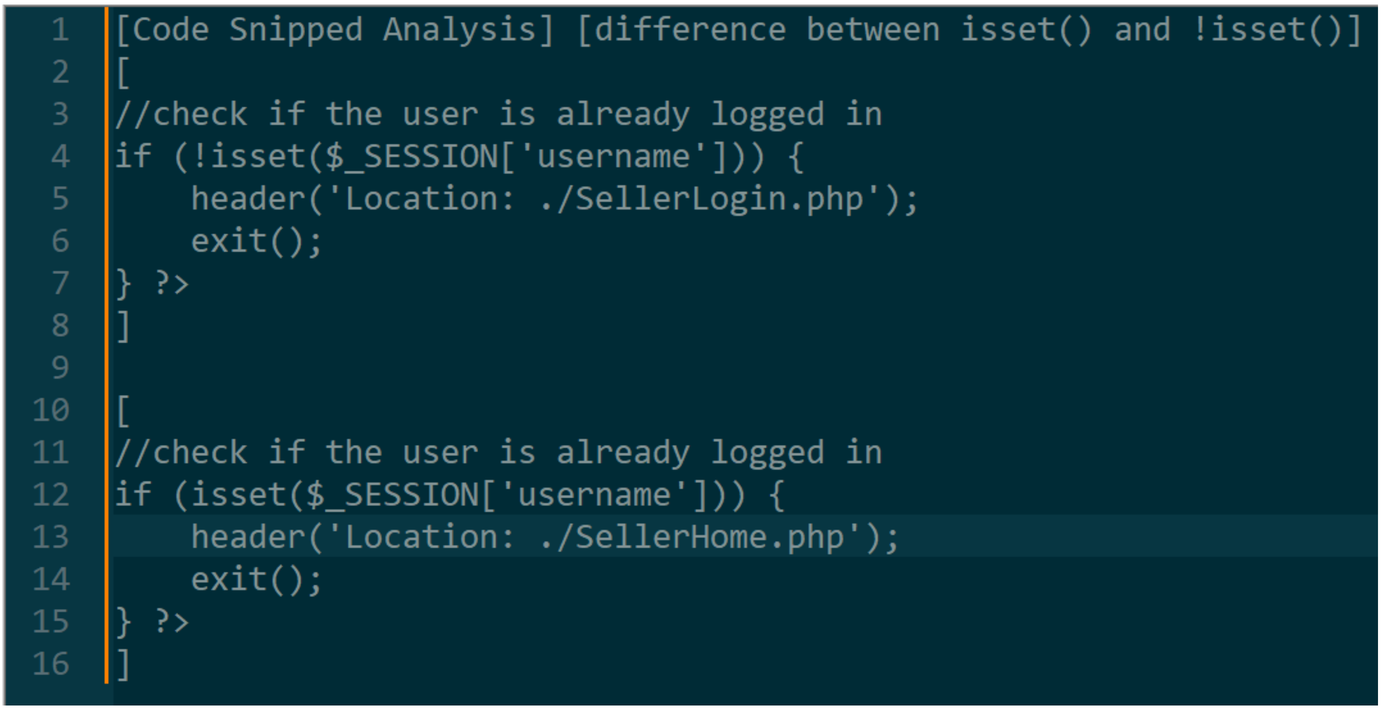
In this code snippet, a `foreach` loop is used to iterate over an array `$data` and count the number of elements in the array. The variable `$count` is initialized to zero, and is then incremented by one for each element in the array.

The `foreach` loop is a PHP control structure that allows you to iterate over the elements of an array or an object. In this code, the loop iterates over the elements of the array `$data`, and for each iteration, the loop variable `$value` is assigned the value of the current element, and the loop variable `$key` is assigned the key of the current element. However, the `$key` variable is not used in this code.

The loop simply increments the `$count` variable by 1 for each iteration of the loop. After the loop has finished executing, the `$count` variable contains the total number of elements in the `$data` array.

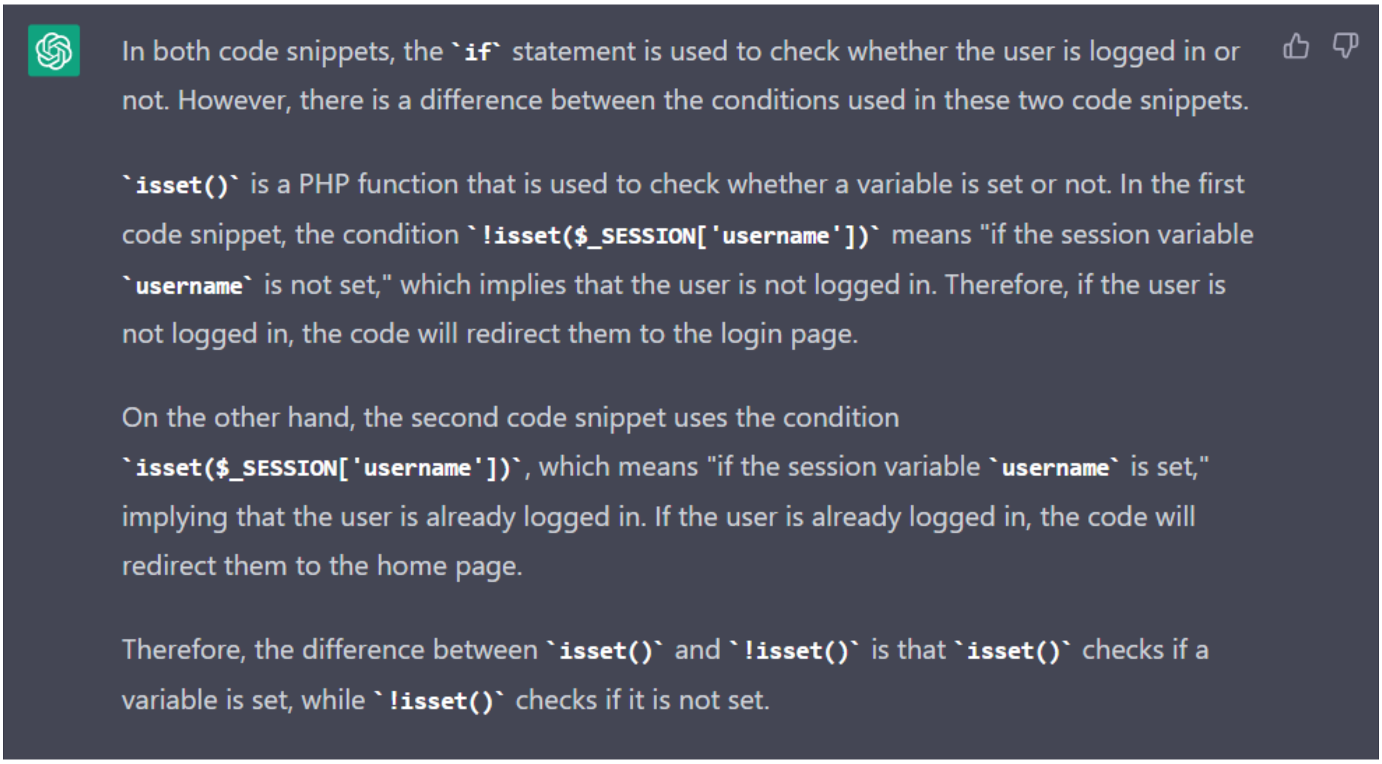
This code can be useful for calculating the size of an array or the number of items in a list. In this specific case, it is used to calculate the total number of items added by the seller.

[Know the difference between isset() and !isset()]



```
[Code Snipped Analysis] [difference between isset() and !isset()]
[
//check if the user is already logged in
if (!isset($_SESSION['username'])) {
    header('Location: ./SellerLogin.php');
    exit();
} ?>
]

[
//check if the user is already logged in
if (isset($_SESSION['username'])) {
    header('Location: ./SellerHome.php');
    exit();
} ?>
]
```

Explanation:



In both code snippets, the `if` statement is used to check whether the user is logged in or not. However, there is a difference between the conditions used in these two code snippets.

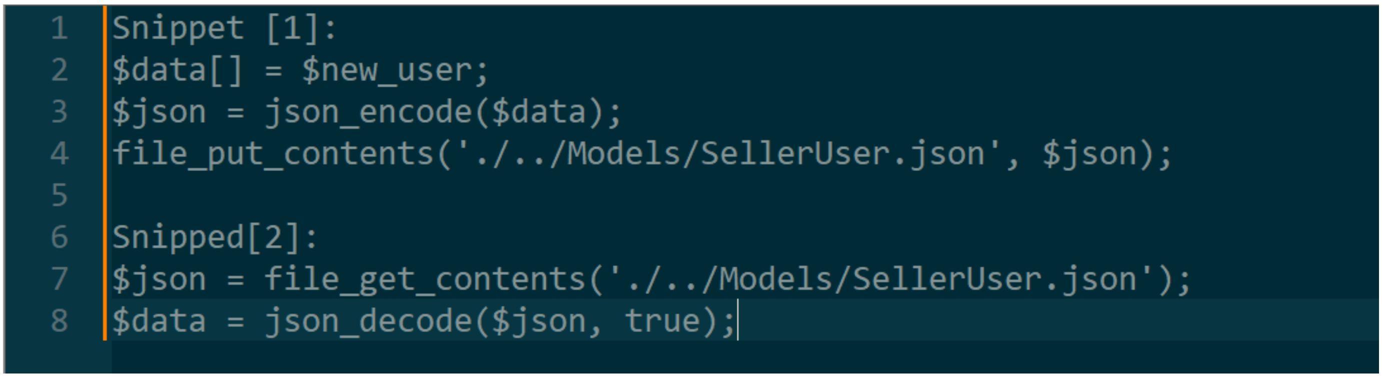
`isset()` is a PHP function that is used to check whether a variable is set or not. In the first code snippet, the condition `!isset($_SESSION['username'])` means "if the session variable `username` is not set," which implies that the user is not logged in. Therefore, if the user is not logged in, the code will redirect them to the login page.

On the other hand, the second code snippet uses the condition `isset($_SESSION['username'])`, which means "if the session variable `username` is set," implying that the user is already logged in. If the user is already logged in, the code will redirect them to the home page.

Therefore, the difference between `isset()` and `!isset()` is that `isset()` checks if a variable is set, while `!isset()` checks if it is not set.

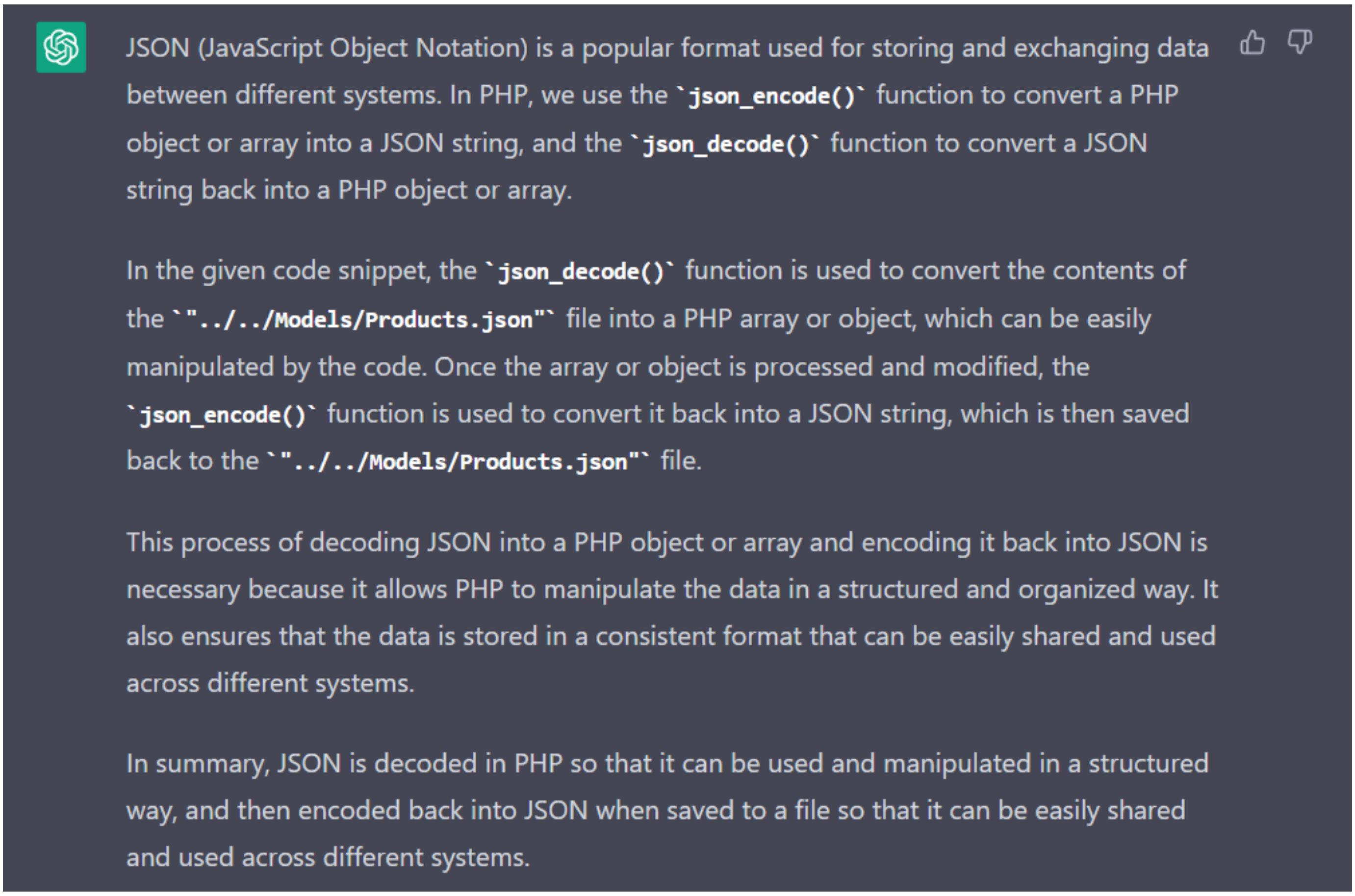
[Know the difference between JSON Encode and JSON Decode]

Code:



```
Snippet [1]:
$data[] = $new_user;
$json = json_encode($data);
file_put_contents('./../Models/SellerUser.json', $json);

Snipped[2]:
$json = file_get_contents('./../Models/SellerUser.json');
$data = json_decode($json, true);
```

Explanation:



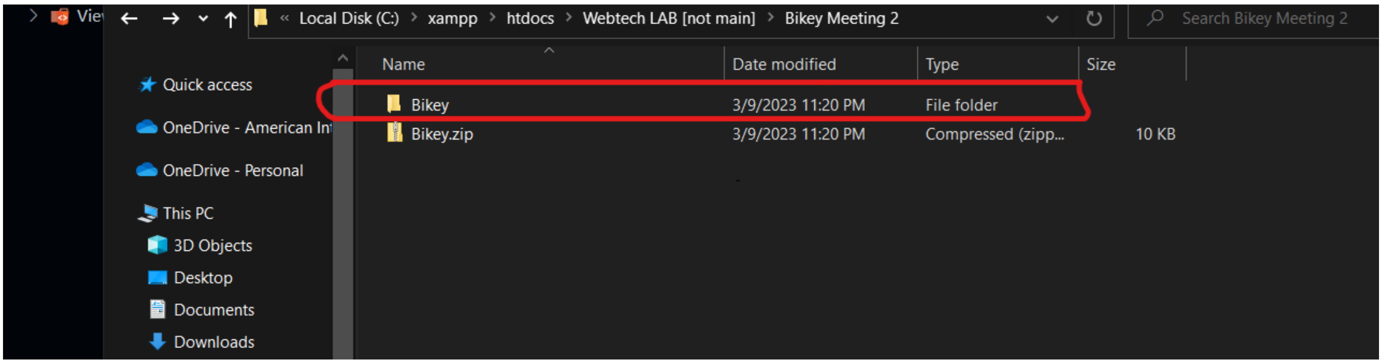
JSON (JavaScript Object Notation) is a popular format used for storing and exchanging data between different systems. In PHP, we use the `json_encode()` function to convert a PHP object or array into a JSON string, and the `json_decode()` function to convert a JSON string back into a PHP object or array.

In the given code snippet, the `json_decode()` function is used to convert the contents of the `"../../Models/Products.json"` file into a PHP array or object, which can be easily manipulated by the code. Once the array or object is processed and modified, the `json_encode()` function is used to convert it back into a JSON string, which is then saved back to the `"../../Models/Products.json"` file.

This process of decoding JSON into a PHP object or array and encoding it back into JSON is necessary because it allows PHP to manipulate the data in a structured and organized way. It also ensures that the data is stored in a consistent format that can be easily shared and used across different systems.

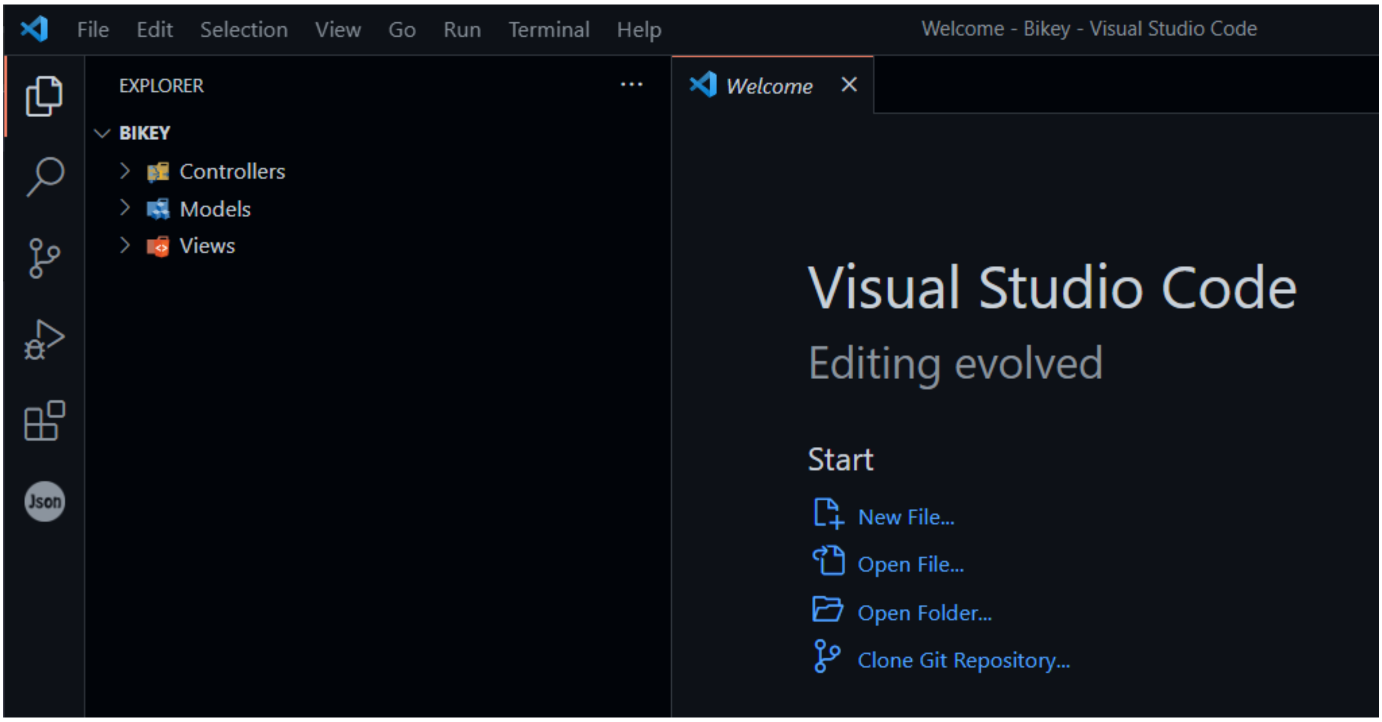
In summary, JSON is decoded in PHP so that it can be used and manipulated in a structured way, and then encoded back into JSON when saved to a file so that it can be easily shared and used across different systems.

Here, Root Boler is [Bikey]

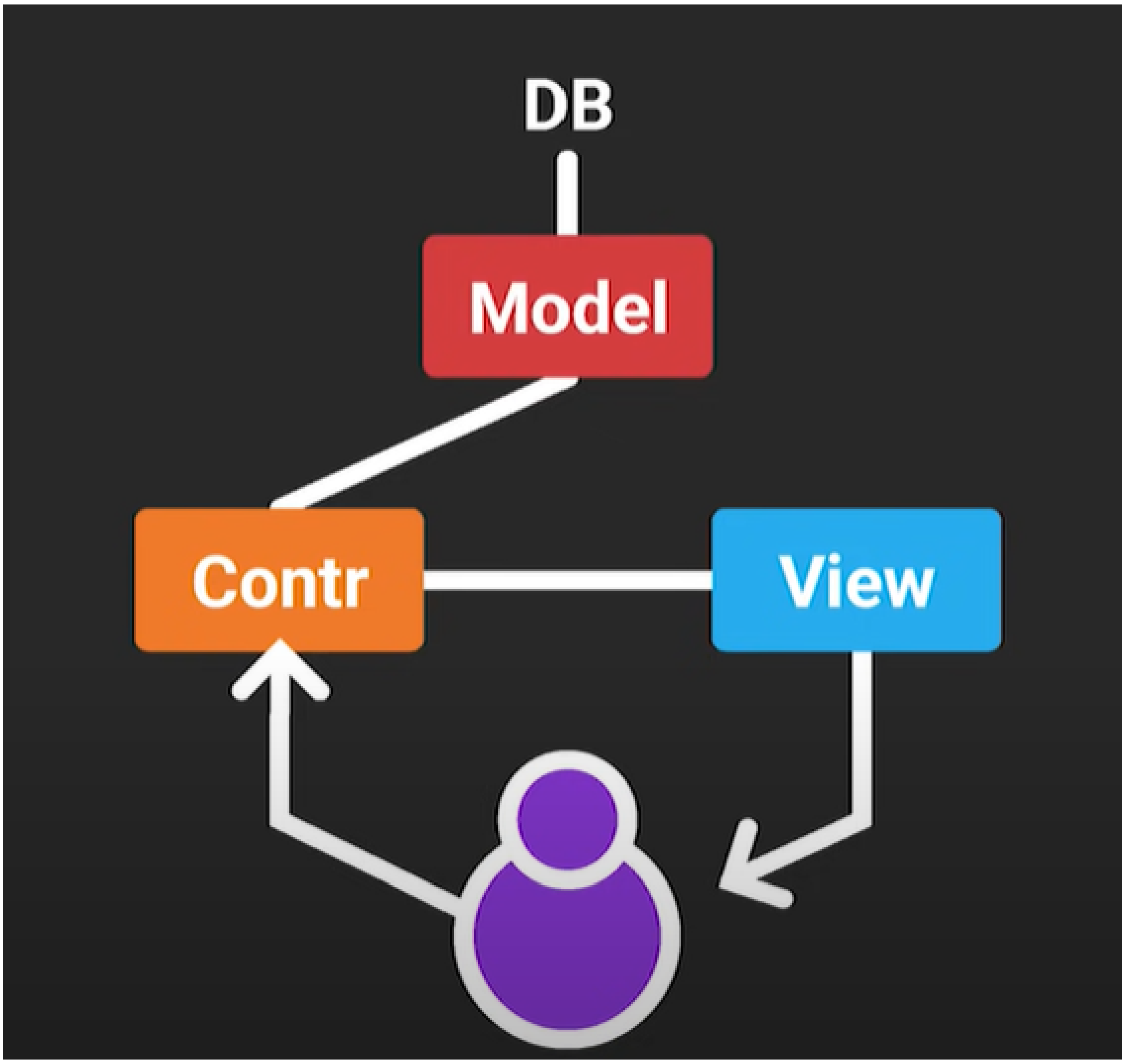


Name	Date modified	Type	Size
Bikey	3/9/2023 11:20 PM	File folder	
Bikey.zip	3/9/2023 11:20 PM	Compressed (zipp...	10 KB

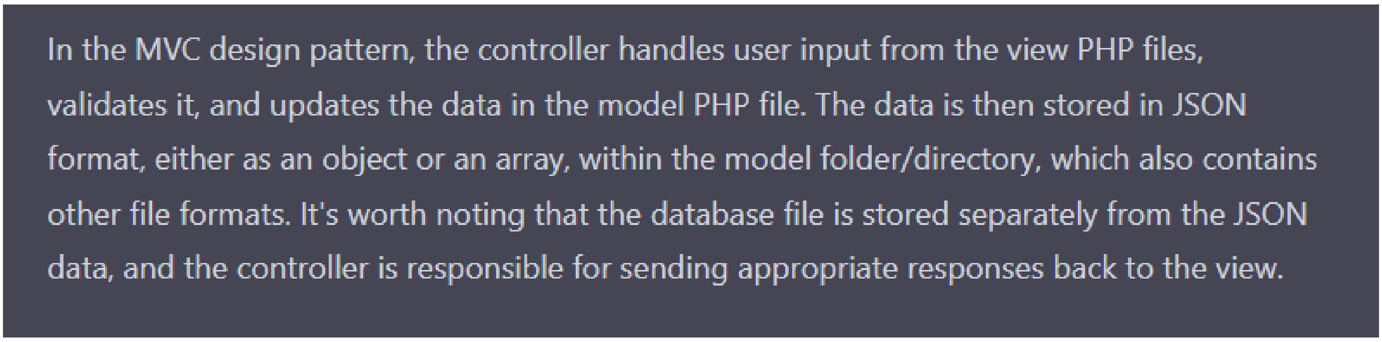
Here, this is the MVC [Architectural structure] of the Project: [Written as C-M-V] / [Same thing, MVC]



Our Seller[User] using this MVC architectural pattern:



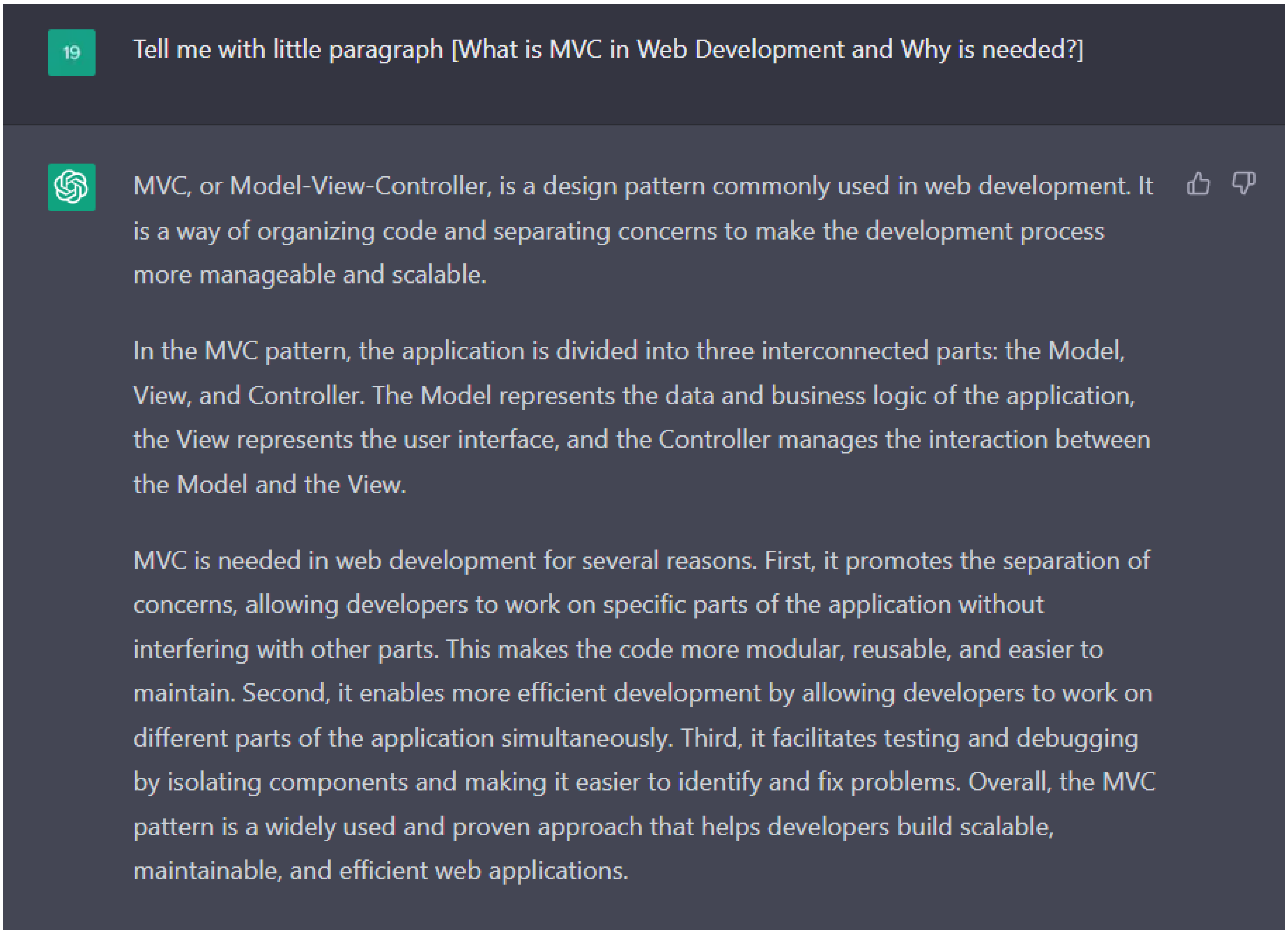
Here, in our [Seller] module:



In the MVC design pattern, the controller handles user input from the view PHP files, validates it, and updates the data in the model PHP file. The data is then stored in JSON format, either as an object or an array, within the model folder/directory, which also contains other file formats. It's worth noting that the database file is stored separately from the JSON data, and the controller is responsible for sending appropriate responses back to the view.

My part was developing the user: "Seller"

Why [M-V-C] used, you may ask:
Reference[2]: Reply For Chat-GPT



Tell me with little paragraph [What is MVC in Web Development and Why is needed?]

MVC, or Model-View-Controller, is a design pattern commonly used in web development. It is a way of organizing code and separating concerns to make the development process more manageable and scalable.

In the MVC pattern, the application is divided into three interconnected parts: the Model, View, and Controller. The Model represents the data and business logic of the application, the View represents the user interface, and the Controller manages the interaction between the Model and the View.

MVC is needed in web development for several reasons. First, it promotes the separation of concerns, allowing developers to work on specific parts of the application without interfering with other parts. This makes the code more modular, reusable, and easier to maintain. Second, it enables more efficient development by allowing developers to work on different parts of the application simultaneously. Third, it facilitates testing and debugging by isolating components and making it easier to identify and fix problems. Overall, the MVC pattern is a widely used and proven approach that helps developers build scalable, maintainable, and efficient web applications.

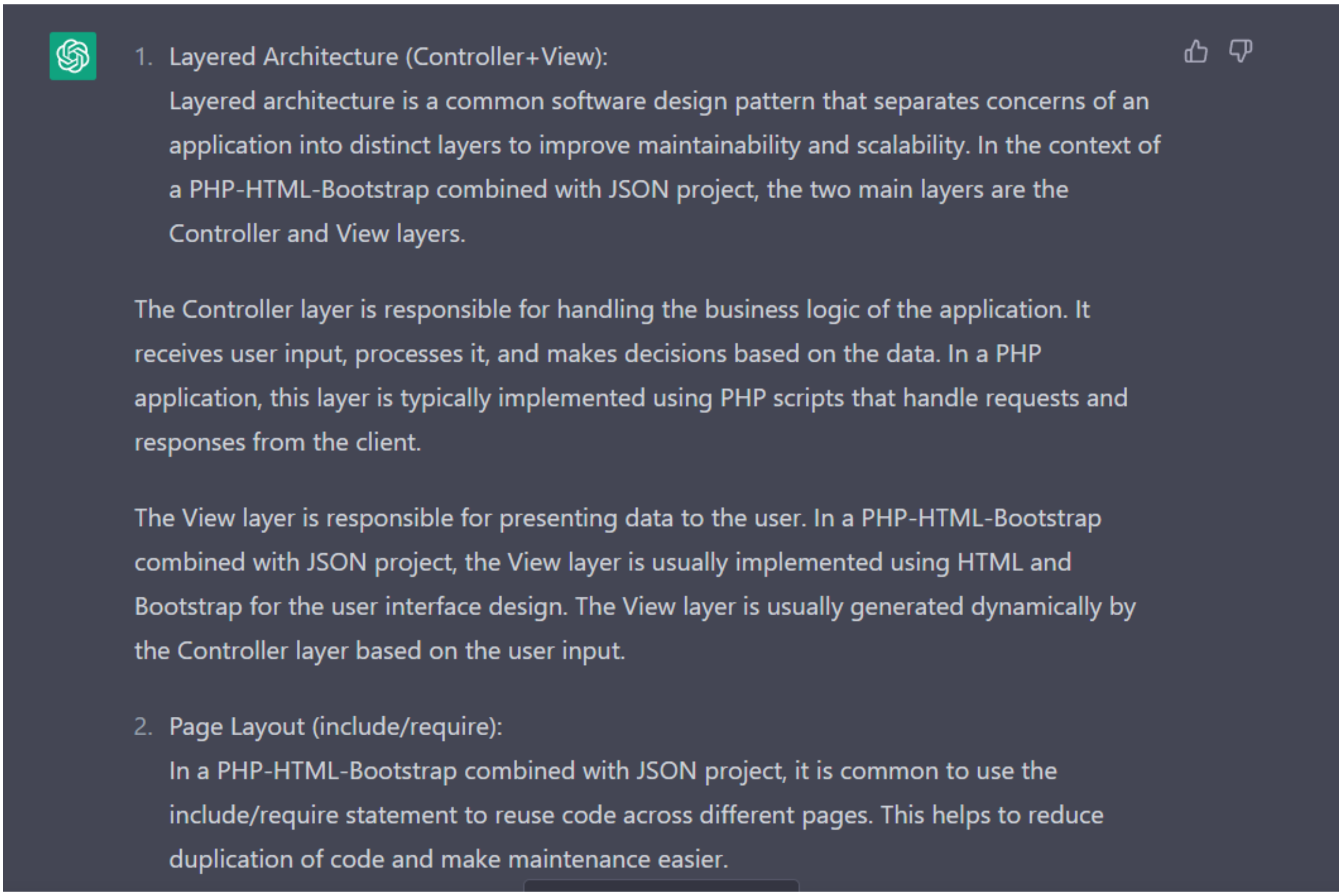
[Full MID TERM REQUIREMENT Explained:]
[Explain all these sections in detals for PHP-HTML-BootStrap Combined with JSON Project]

1 Layered Architecture (Controller+view)
2 Page Layout (include/require)
3 Validation (PHP)
4 File I/O (JSON)
5 Session/Cookies
6 Login/Registration Module

7 Database Design

8 Static Web Interfaces (All pages)

[1]



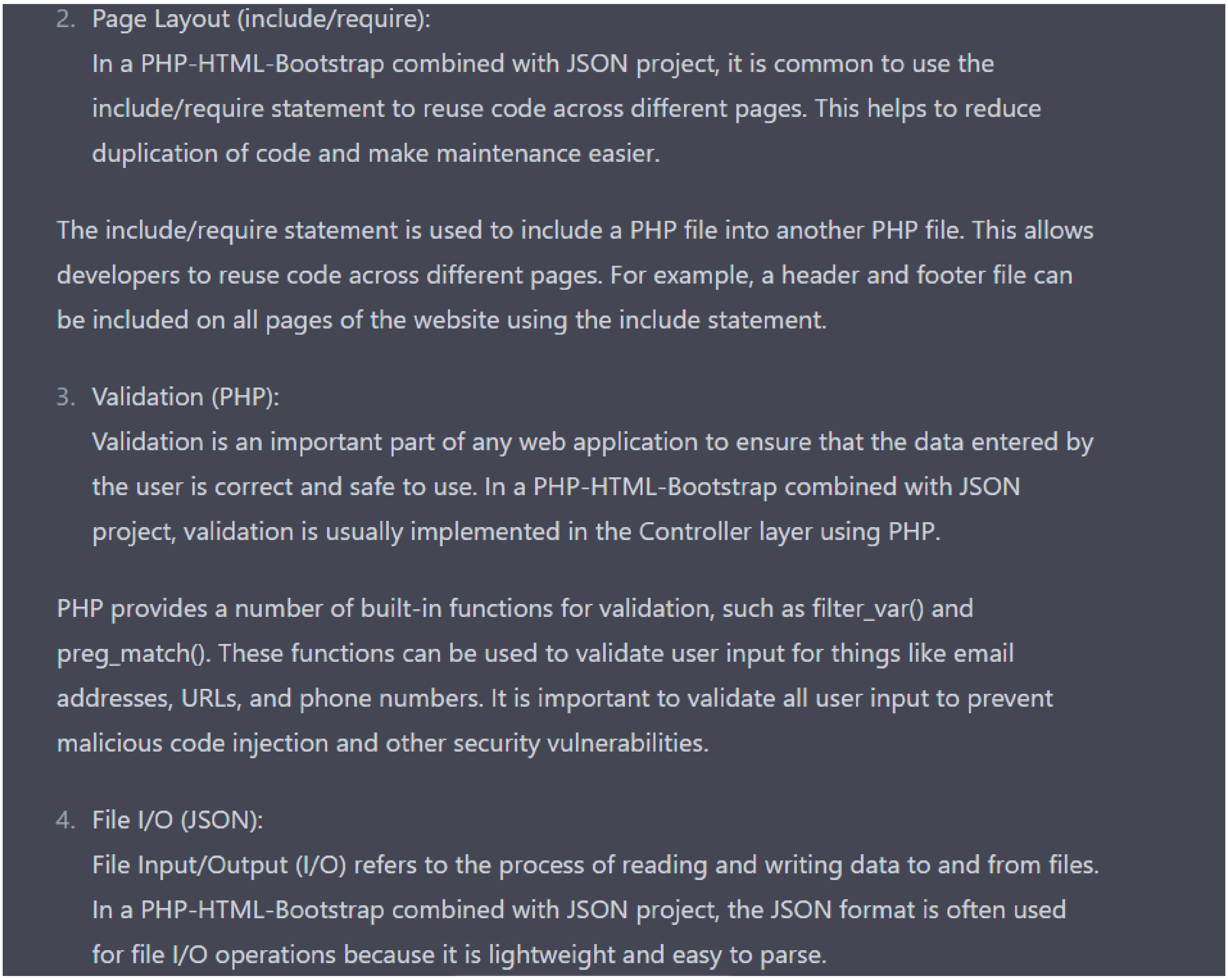
1. Layered Architecture (Controller+View):
   Layered architecture is a common software design pattern that separates concerns of an application into distinct layers to improve maintainability and scalability. In the context of a PHP-HTML-Bootstrap combined with JSON project, the two main layers are the Controller and View layers.

The Controller layer is responsible for handling the business logic of the application. It receives user input, processes it, and makes decisions based on the data. In a PHP application, this layer is typically implemented using PHP scripts that handle requests and responses from the client.

The View layer is responsible for presenting data to the user. In a PHP-HTML-Bootstrap combined with JSON project, the View layer is usually implemented using HTML and Bootstrap for the user interface design. The View layer is usually generated dynamically by the Controller layer based on the user input.

2. Page Layout (include/require):
   In a PHP-HTML-Bootstrap combined with JSON project, it is common to use the include/require statement to reuse code across different pages. This helps to reduce duplication of code and make maintenance easier.

[2]



2. Page Layout (include/require):
   In a PHP-HTML-Bootstrap combined with JSON project, it is common to use the include/require statement to reuse code across different pages. This helps to reduce duplication of code and make maintenance easier.

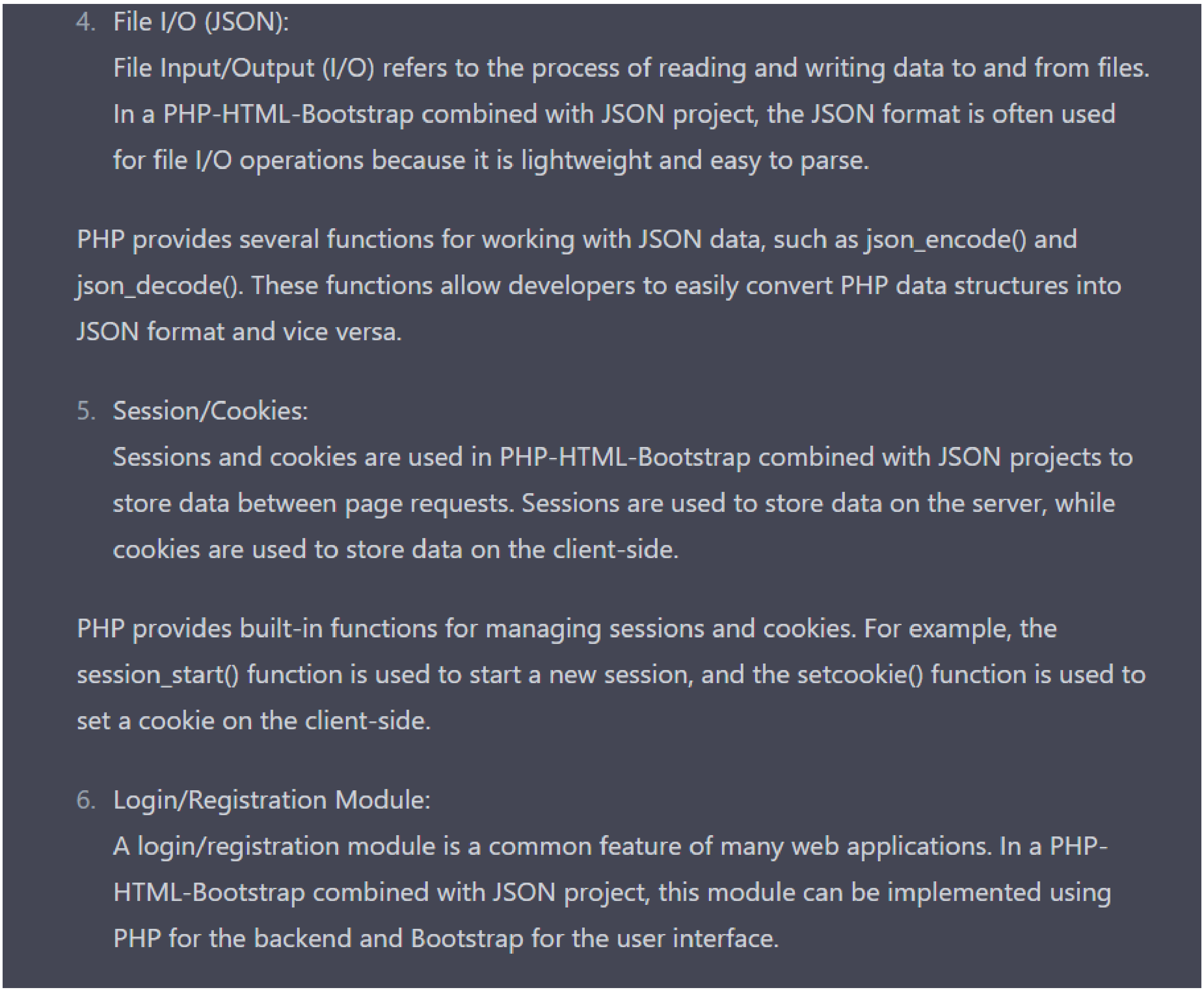
The include/require statement is used to include a PHP file into another PHP file. This allows developers to reuse code across different pages. For example, a header and footer file can be included on all pages of the website using the include statement.

3. Validation (PHP):
   Validation is an important part of any web application to ensure that the data entered by the user is correct and safe to use. In a PHP-HTML-Bootstrap combined with JSON project, validation is usually implemented in the Controller layer using PHP.

PHP provides a number of built-in functions for validation, such as filter_var() and preg_match(). These functions can be used to validate user input for things like email addresses, URLs, and phone numbers. It is important to validate all user input to prevent malicious code injection and other security vulnerabilities.

4. File I/O (JSON):
   File Input/Output (I/O) refers to the process of reading and writing data to and from files. In a PHP-HTML-Bootstrap combined with JSON project, the JSON format is often used for file I/O operations because it is lightweight and easy to parse.

[3]



4. File I/O (JSON):
   File Input/Output (I/O) refers to the process of reading and writing data to and from files. In a PHP-HTML-Bootstrap combined with JSON project, the JSON format is often used for file I/O operations because it is lightweight and easy to parse.

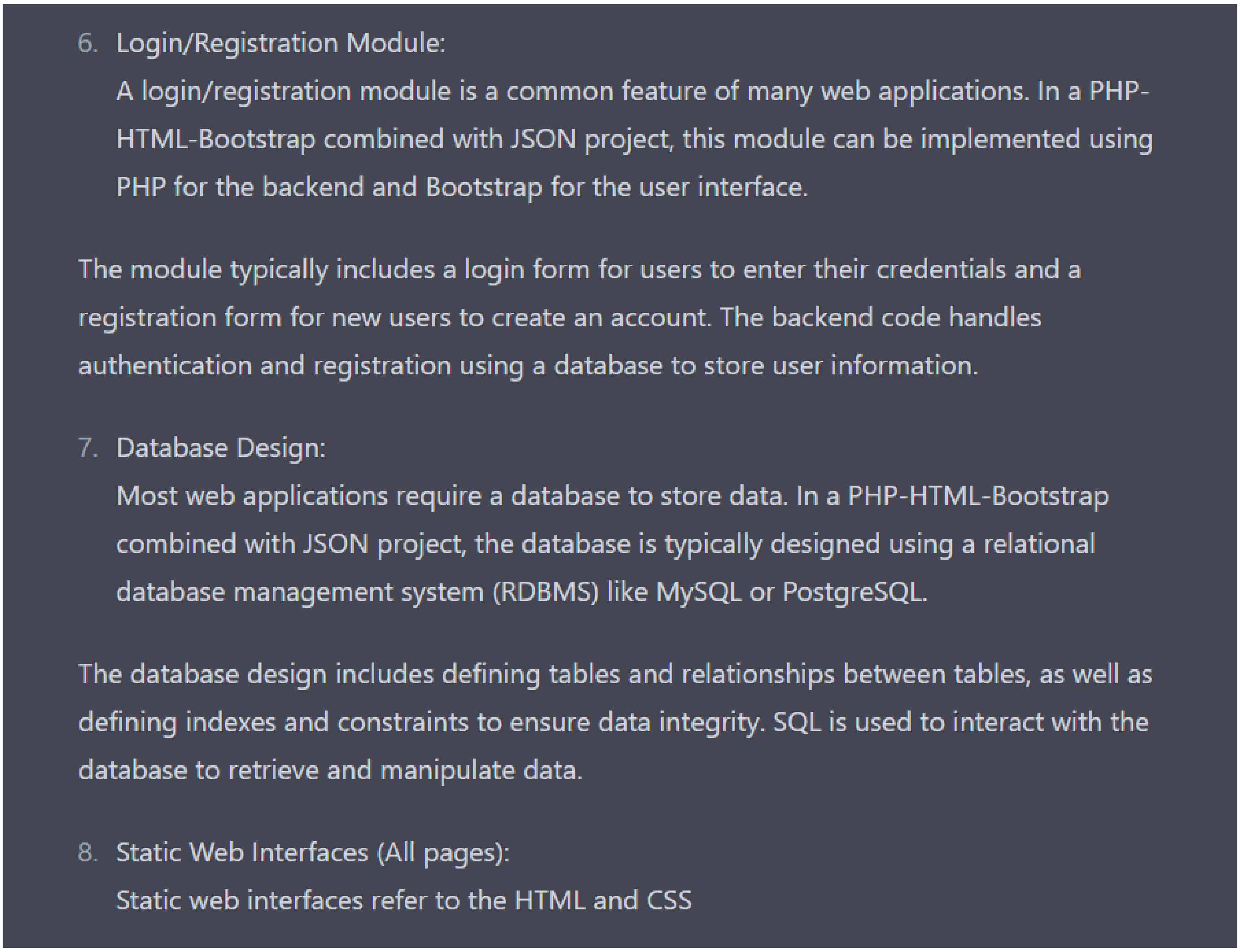
PHP provides several functions for working with JSON data, such as json_encode() and json_decode(). These functions allow developers to easily convert PHP data structures into JSON format and vice versa.

5. Session/Cookies:
   Sessions and cookies are used in PHP-HTML-Bootstrap combined with JSON projects to store data between page requests. Sessions are used to store data on the server, while cookies are used to store data on the client-side.

PHP provides built-in functions for managing sessions and cookies. For example, the session_start() function is used to start a new session, and the setcookie() function is used to set a cookie on the client-side.

6. Login/Registration Module:
   A login/registration module is a common feature of many web applications. In a PHP-HTML-Bootstrap combined with JSON project, this module can be implemented using PHP for the backend and Bootstrap for the user interface.

[4]



6. Login/Registration Module:
   A login/registration module is a common feature of many web applications. In a PHP-HTML-Bootstrap combined with JSON project, this module can be implemented using PHP for the backend and Bootstrap for the user interface.

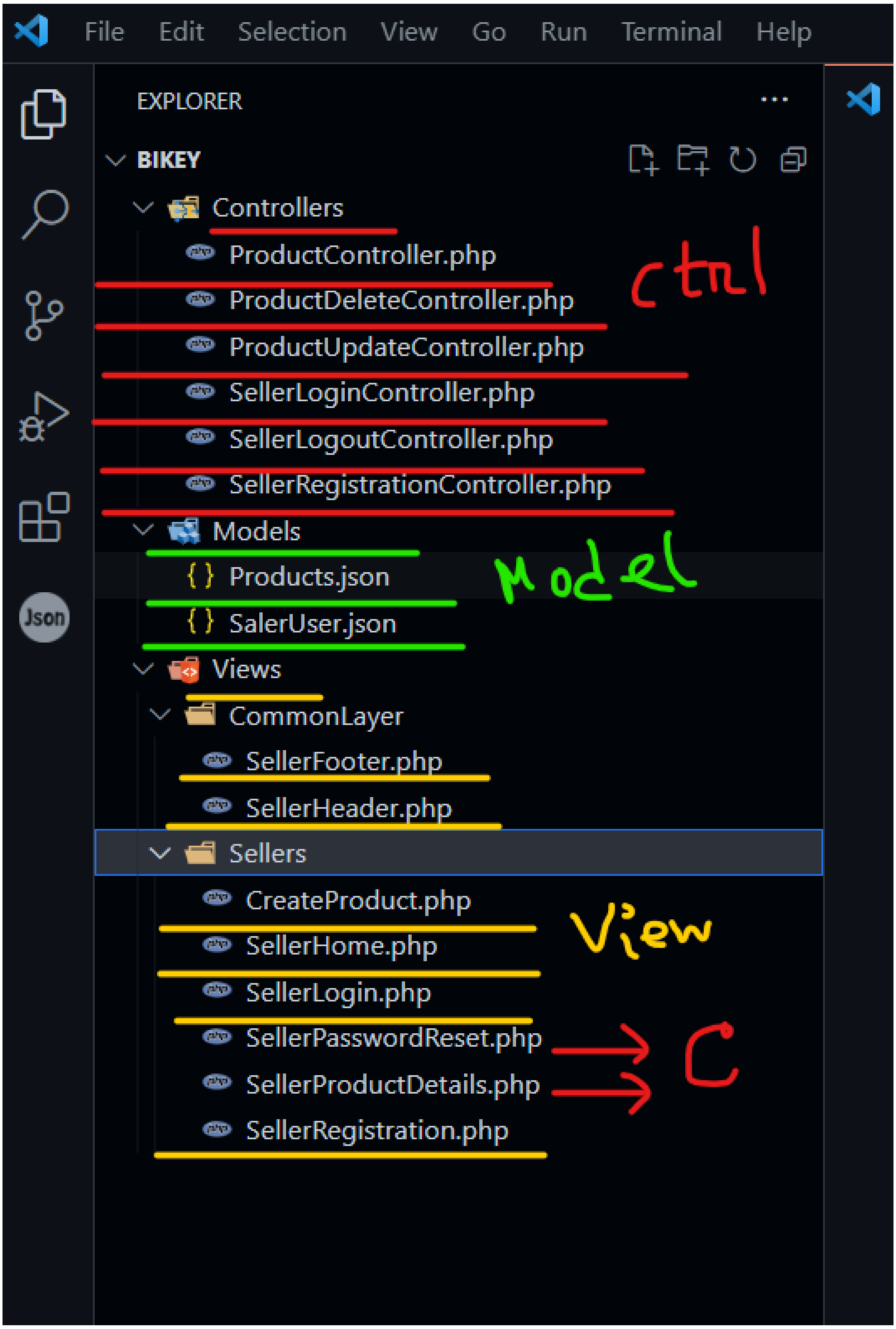
The module typically includes a login form for users to enter their credentials and a registration form for new users to create an account. The backend code handles authentication and registration using a database to store user information.

7. Database Design:
   Most web applications require a database to store data. In a PHP-HTML-Bootstrap combined with JSON project, the database is typically designed using a relational database management system (RDBMS) like MySQL or PostgreSQL.

The database design includes defining tables and relationships between tables, as well as defining indexes and constraints to ensure data integrity. SQL is used to interact with the database to retrieve and manipulate data.

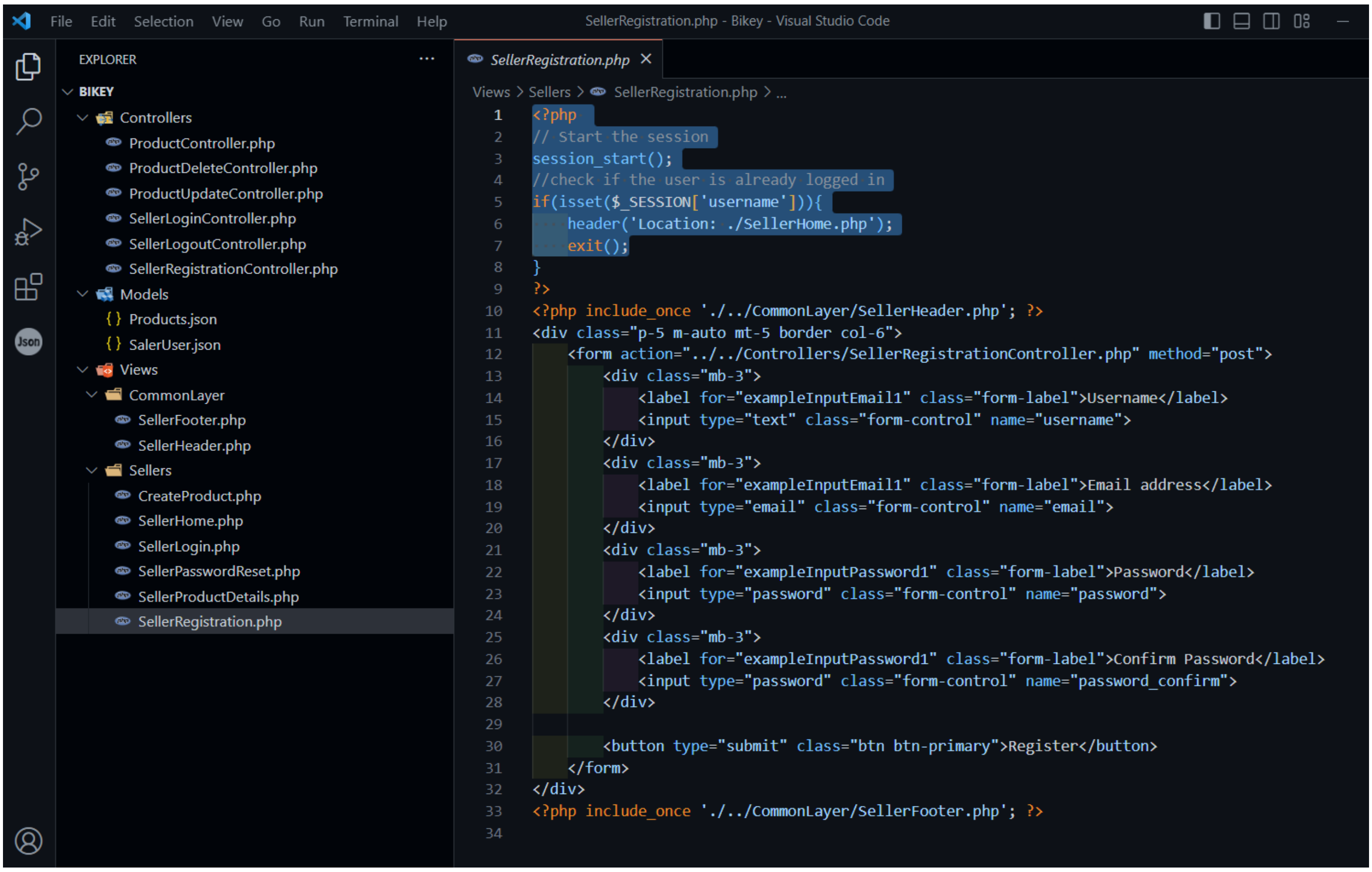
8. Static Web Interfaces (All pages):
   Static web interfaces refer to the HTML and CSS

Project's [Seller] Files highlighted in the
M-V-C Pattern:



Code Explanation:

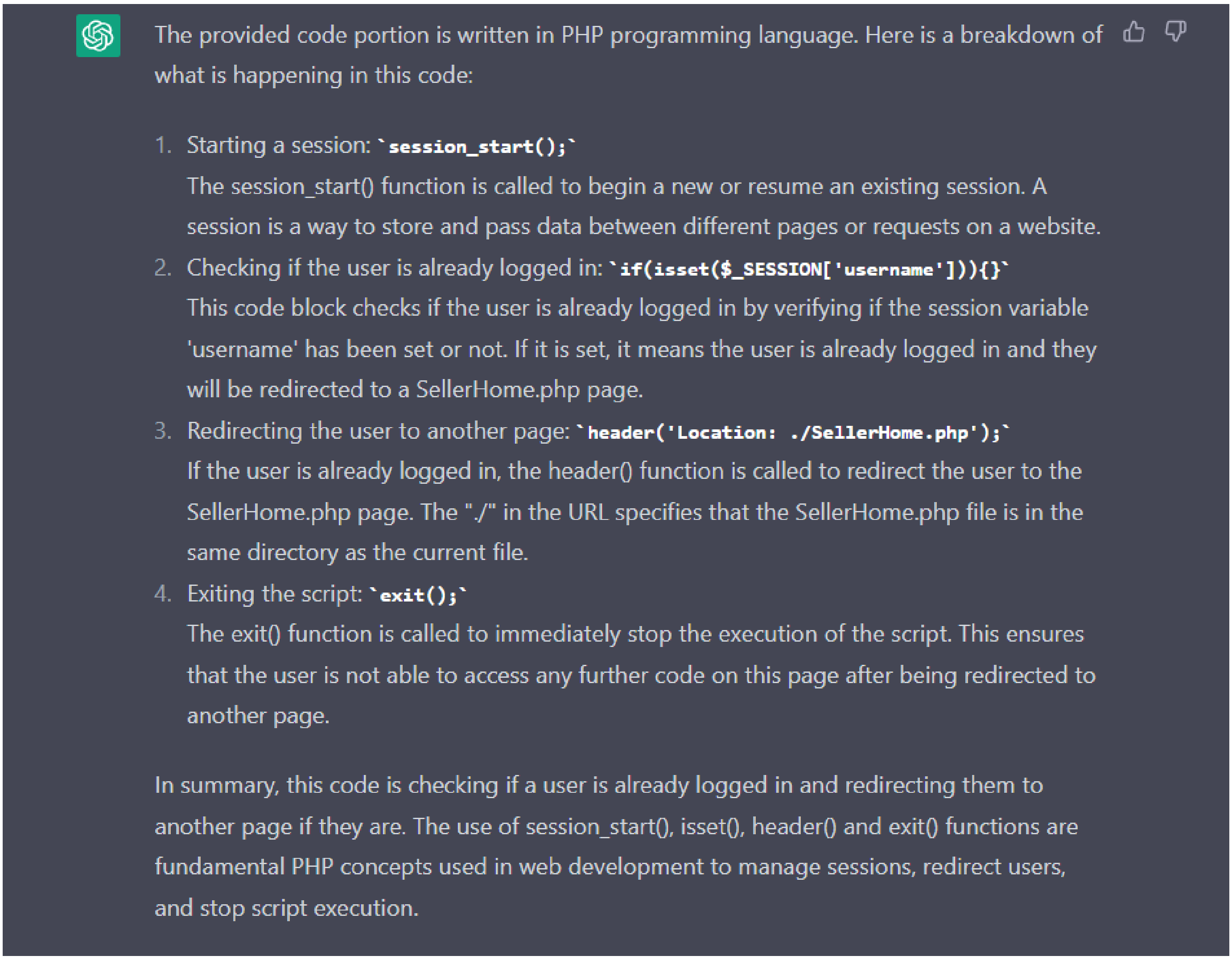
Start with Seller Account Registration[SellerRegistration.php]



```
<?php
// Start the session
session_start();
//check if the user is already logged in
if(isset($_SESSION['username'])){
    header('Location: ./SellerHome.php');
    exit();
}
?>
<?php include_once './../CommonLayer/SellerHeader.php'; ?>
<div class="p-5 m-auto mt-5 border col-6">
    <form action="../../Controllers/SellerRegistrationController.php" method="post">
        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Username</label>
            <input type="text" class="form-control" name="username">
        </div>
        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Email address</label>
            <input type="email" class="form-control" name="email">
        </div>
        <div class="mb-3">
            <label for="exampleInputPassword1" class="form-label">Password</label>
            <input type="password" class="form-control" name="password">
        </div>
        <div class="mb-3">
            <label for="exampleInputPassword1" class="form-label">Confirm Password</label>
            <input type="password" class="form-control" name="password_confirm">
        </div>

        <button type="submit" class="btn btn-primary">Register</button>
    </form>
</div>
<?php include_once './../CommonLayer/SellerFooter.php'; ?>

```

Explanation:

[Blue Highlighted Portion]



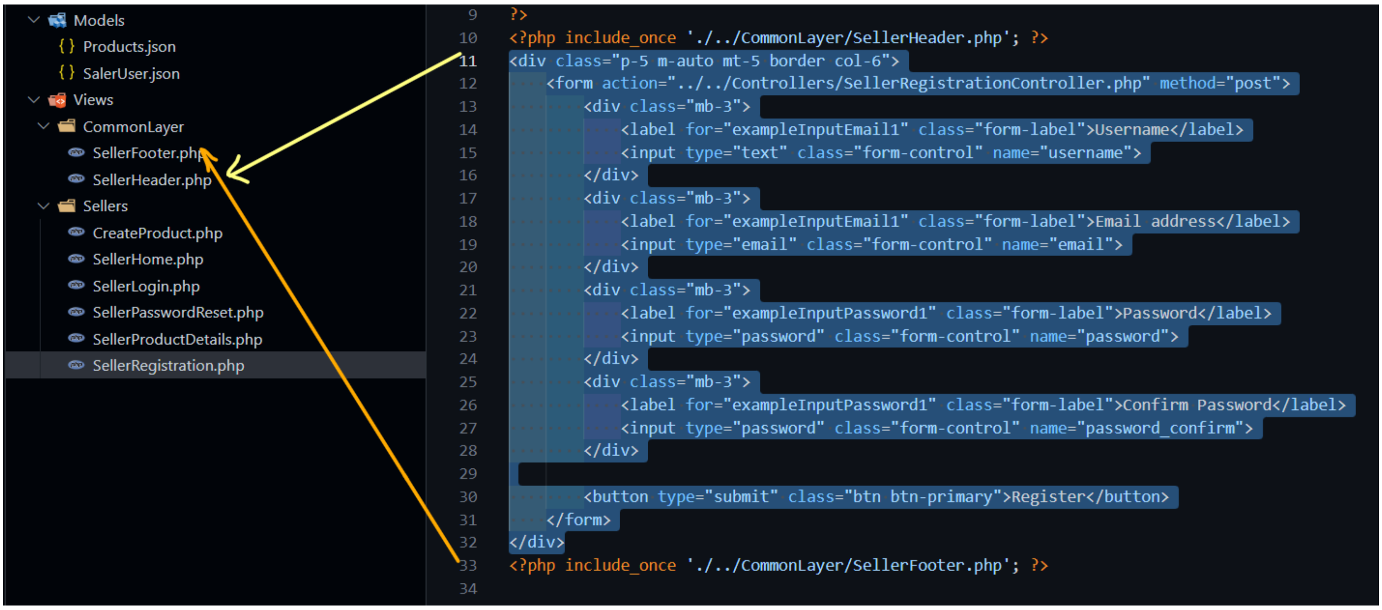
The provided code portion is written in PHP programming language. Here is a breakdown of what is happening in this code:

1. Starting a session: `session_start();`
   The session_start() function is called to begin a new or resume an existing session. A session is a way to store and pass data between different pages or requests on a website.
2. Checking if the user is already logged in: `if(isset($_SESSION['username'])){}`
   This code block checks if the user is already logged in by verifying if the session variable 'username' has been set or not. If it is set, it means the user is already logged in and they will be redirected to a SellerHome.php page.
3. Redirecting the user to another page: `header('Location: ./SellerHome.php');`
   If the user is already logged in, the header() function is called to redirect the user to the SellerHome.php page. The "./" in the URL specifies that the SellerHome.php file is in the same directory as the current file.
4. Exiting the script: `exit();`
   The exit() function is called to immediately stop the execution of the script. This ensures that the user is not able to access any further code on this page after being redirected to another page.

In summary, this code is checking if a user is already logged in and redirecting them to another page if they are. The use of session_start(), isset(), header() and exit() functions are fundamental PHP concepts used in web development to manage sessions, redirect users, and stop script execution.

->Registration part [Form Section]

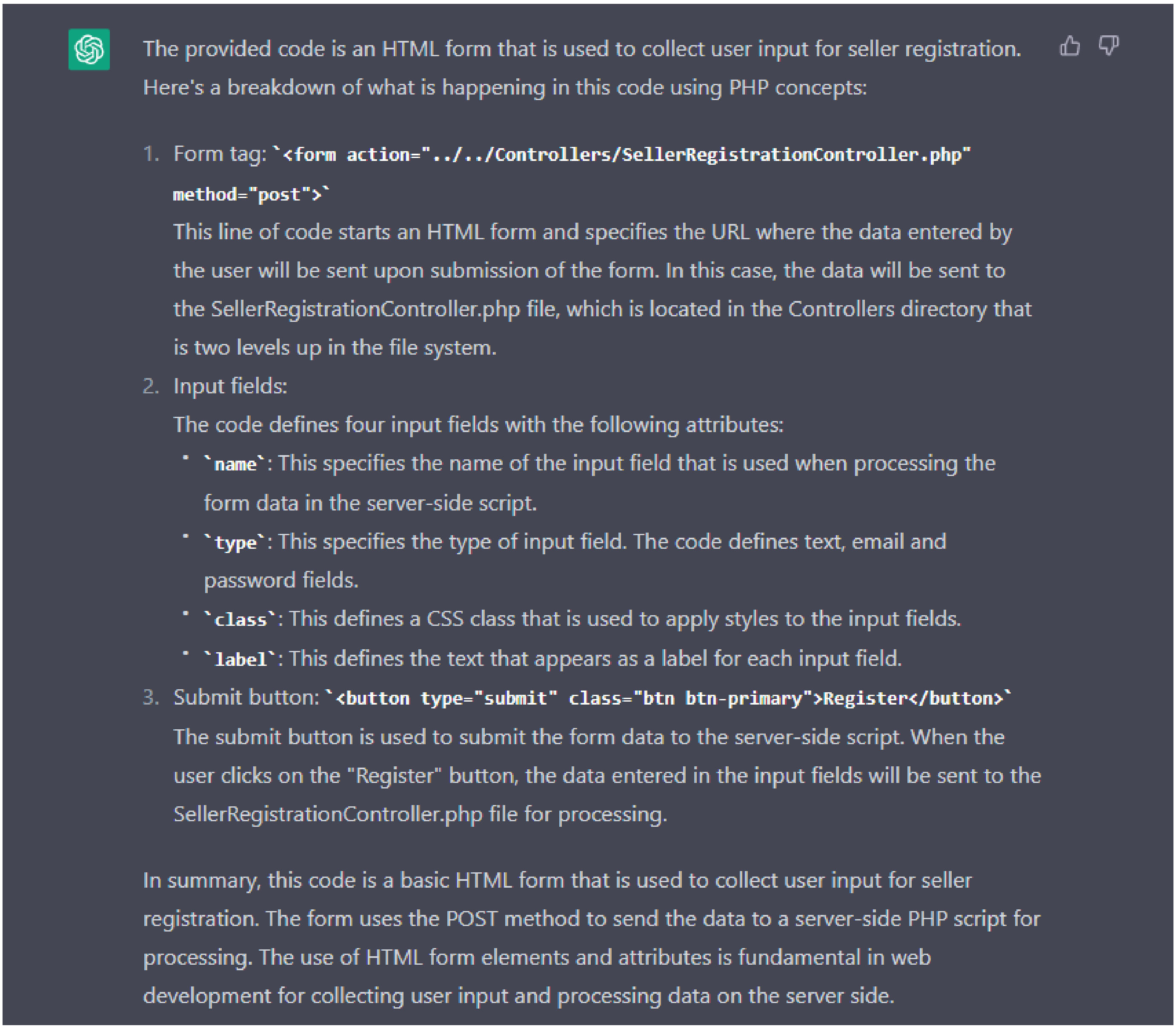
Excluding [SellerFooter.php and SellerHeader.php as they are common layer for all [USERS]]



```
?>
<?php include_once './../CommonLayer/SellerHeader.php'; ?>
<div class="p-5 m-auto mt-5 border col-6">
    <form action="../../Controllers/SellerRegistrationController.php" method="post">
        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Username</label>
            <input type="text" class="form-control" name="username">
        </div>
        <div class="mb-3">
            <label for="exampleInputEmail1" class="form-label">Email address</label>
            <input type="email" class="form-control" name="email">
        </div>
        <div class="mb-3">
            <label for="exampleInputPassword1" class="form-label">Password</label>
            <input type="password" class="form-control" name="password">
        </div>
        <div class="mb-3">
            <label for="exampleInputPassword1" class="form-label">Confirm Password</label>
            <input type="password" class="form-control" name="password_confirm">
        </div>

        <button type="submit" class="btn btn-primary">Register</button>
    </form>
</div>
<?php include_once './../CommonLayer/SellerFooter.php'; ?>
```

Explanation:

->[HTML FORM PART with PHP]

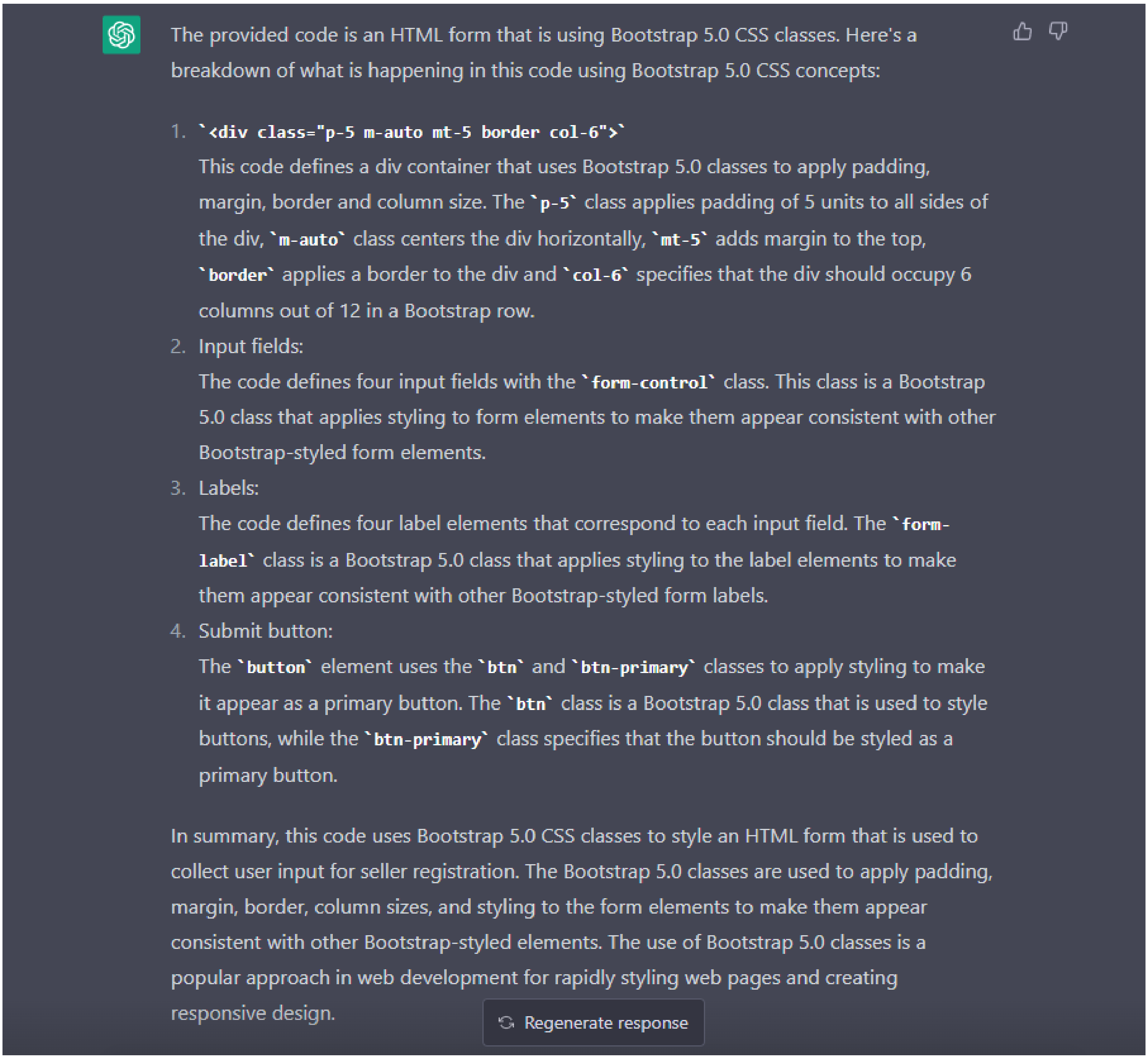


The provided code is an HTML form that is used to collect user input for seller registration. Here's a breakdown of what is happening in this code using PHP concepts:

1. Form tag: `<form action="../../Controllers/SellerRegistrationController.php" method="post">`
   This line of code starts an HTML form and specifies the URL where the data entered by the user will be sent upon submission of the form. In this case, the data will be sent to the SellerRegistrationController.php file, which is located in the Controllers directory that is two levels up in the file system.
2. Input fields:
   The code defines four input fields with the following attributes:
   - `name`: This specifies the name of the input field that is used when processing the form data in the server-side script.
   - `type`: This specifies the type of input field. The code defines text, email and password fields.
   - `class`: This defines a CSS class that is used to apply styles to the input fields.
   - `label`: This defines the text that appears as a label for each input field.
3. Submit button: `<button type="submit" class="btn btn-primary">Register</button>`
   The submit button is used to submit the form data to the server-side script. When the user clicks on the "Register" button, the data entered in the input fields will be sent to the SellerRegistrationController.php file for processing.

In summary, this code is a basic HTML form that is used to collect user input for seller registration. The form uses the POST method to send the data to a server-side PHP script for processing. The use of HTML form elements and attributes is fundamental in web development for collecting user input and processing data on the server side.

->[CSS: BootStrap 5.0 Portion]



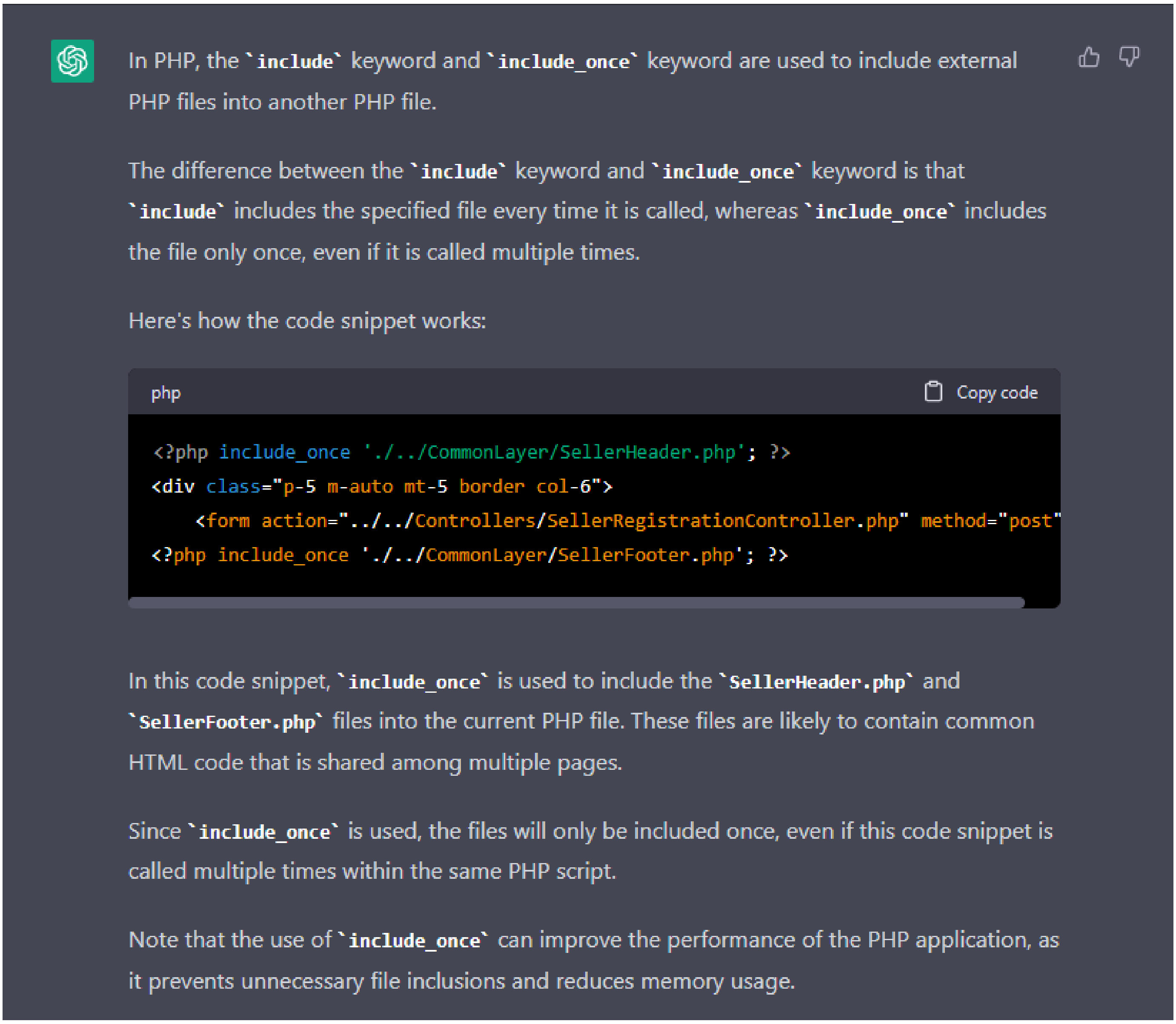
The provided code is an HTML form that is using Bootstrap 5.0 CSS classes. Here's a breakdown of what is happening in this code using Bootstrap 5.0 CSS concepts:

1. `` `<div class="p-5 m-auto mt-5 border col-6">` ``
   This code defines a div container that uses Bootstrap 5.0 classes to apply padding, margin, border and column size. The `` `p-5` `` class applies padding of 5 units to all sides of the div, `` `m-auto` `` class centers the div horizontally, `` `mt-5` `` adds margin to the top, `` `border` `` applies a border to the div and `` `col-6` `` specifies that the div should occupy 6 columns out of 12 in a Bootstrap row.
2. Input fields:
   The code defines four input fields with the `` `form-control` `` class. This class is a Bootstrap 5.0 class that applies styling to form elements to make them appear consistent with other Bootstrap-styled form elements.
3. Labels:
   The code defines four label elements that correspond to each input field. The `` `form-label` `` class is a Bootstrap 5.0 class that applies styling to the label elements to make them appear consistent with other Bootstrap-styled form labels.
4. Submit button:
   The `` `button` `` element uses the `` `btn` `` and `` `btn-primary` `` classes to apply styling to make it appear as a primary button. The `` `btn` `` class is a Bootstrap 5.0 class that is used to style buttons, while the `` `btn-primary` `` class specifies that the button should be styled as a primary button.

In summary, this code uses Bootstrap 5.0 CSS classes to style an HTML form that is used to collect user input for seller registration. The Bootstrap 5.0 classes are used to apply padding, margin, border, column sizes, and styling to the form elements to make them appear consistent with other Bootstrap-styled elements. The use of Bootstrap 5.0 classes is a popular approach in web development for rapidly styling web pages and creating responsive design.

->[include VS include_once]
In PHP, include and include_once are called "include directives". They are used to include external files into a PHP script.

[Btw, [include] korle kinto jei file a include kora hoy oi file er prerspective a think kora lagbe]



In PHP, the `include` keyword and `include_once` keyword are used to include external PHP files into another PHP file.

The difference between the `include` keyword and `include_once` keyword is that `include` includes the specified file every time it is called, whereas `include_once` includes the file only once, even if it is called multiple times.

Here's how the code snippet works:

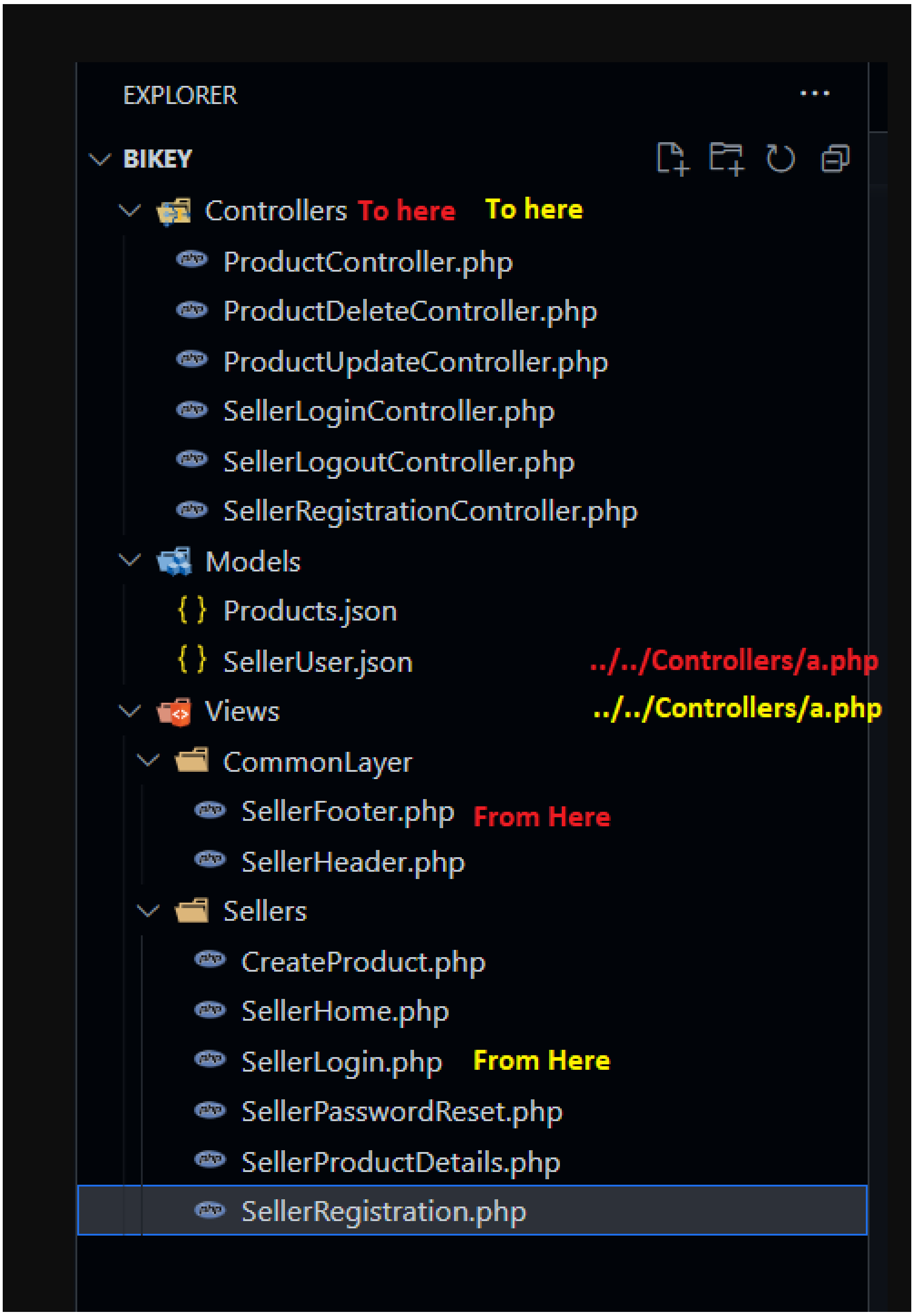
```php
<?php include_once './../CommonLayer/SellerHeader.php'; ?>
<div class="p-5 m-auto mt-5 border col-6">
    <form action="../../Controllers/SellerRegistrationController.php" method="post"
<?php include_once './../CommonLayer/SellerFooter.php'; ?>
```

In this code snippet, `include_once` is used to include the `SellerHeader.php` and `SellerFooter.php` files into the current PHP file. These files are likely to contain common HTML code that is shared among multiple pages.

Since `include_once` is used, the files will only be included once, even if this code snippet is called multiple times within the same PHP script.

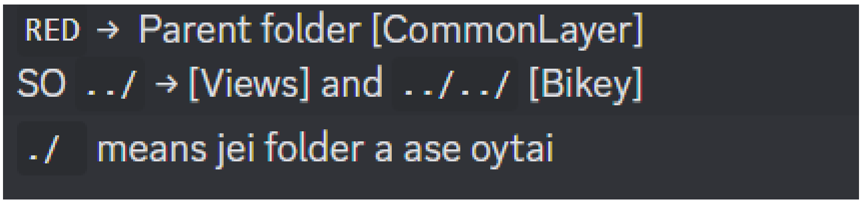
Note that the use of `include_once` can improve the performance of the PHP application, as it prevents unnecessary file inclusions and reduces memory usage.

->[Directory Analogy] explained



```
BIKEY
  Controllers To here    To here
    ProductController.php
    ProductDeleteController.php
    ProductUpdateController.php
    SellerLoginController.php
    SellerLogoutController.php
    SellerRegistrationController.php
  Models
    { } Products.json
    { } SellerUser.json            ../../Controllers/a.php
  Views                            ../../Controllers/a.php
    CommonLayer
      SellerFooter.php  From Here
      SellerHeader.php
    Sellers
      CreateProduct.php
      SellerHome.php
      SellerLogin.php   From Here
      SellerPasswordReset.php
      SellerProductDetails.php
      SellerRegistration.php
```

->[More explanation based on directory]



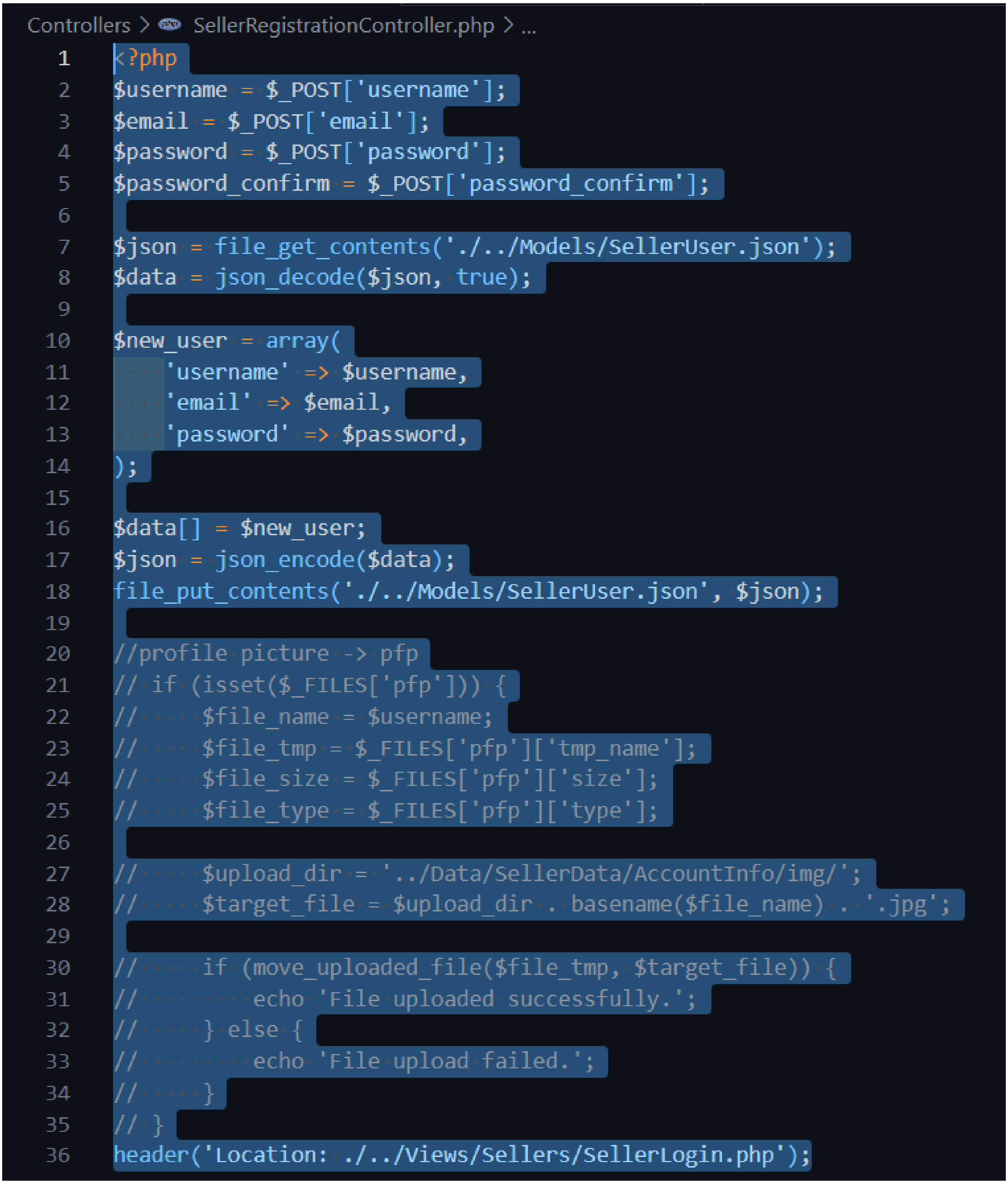
RED → Parent folder [CommonLayer]

SO `../` → [Views] and `../../` [Bikey]

`./` means jei folder a ase oytai

Registration Controller Part Explained:

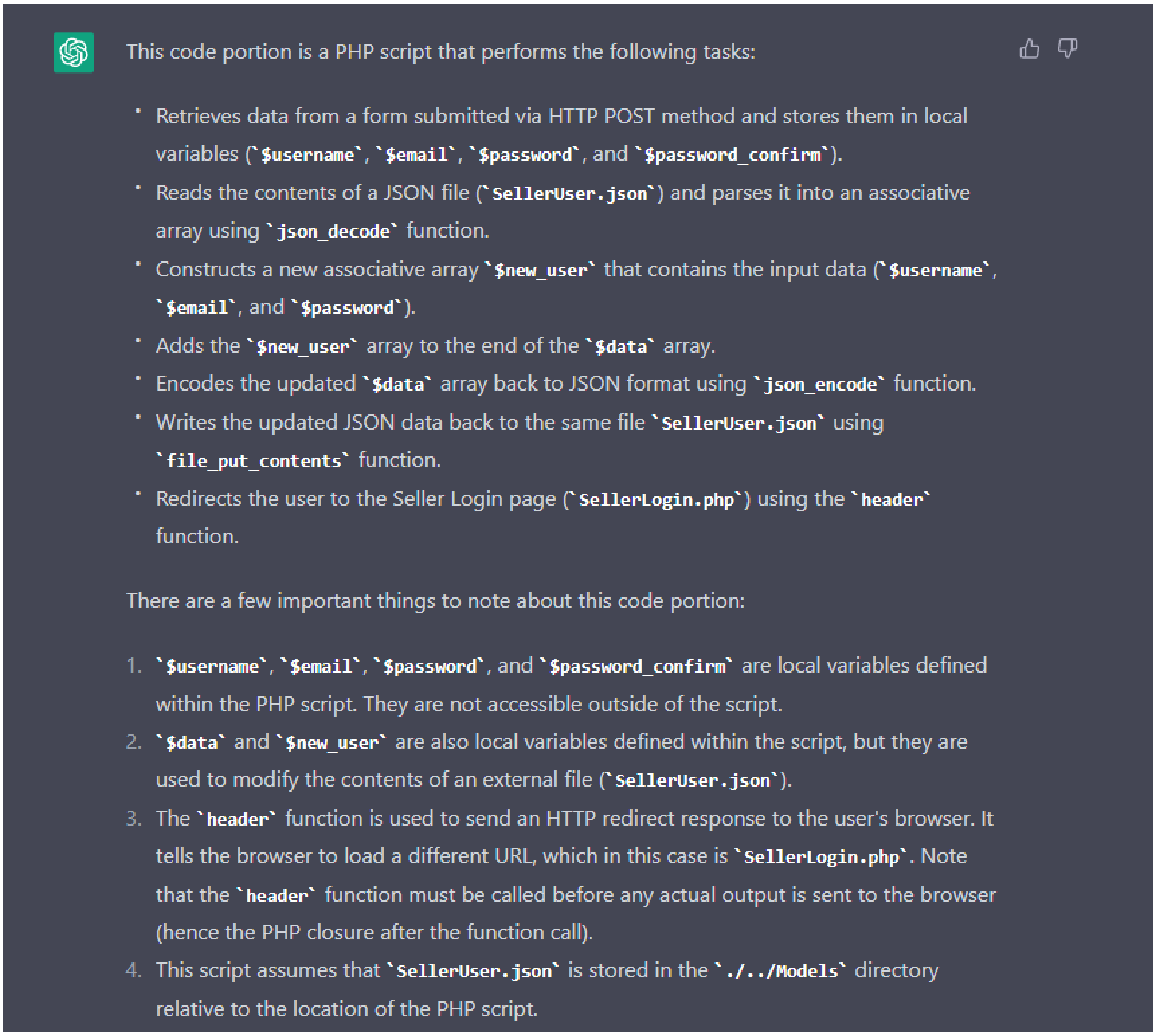
Code:



Controllers > SellerRegistrationController.php > ...

```php
<?php
$username = $_POST['username'];
$email = $_POST['email'];
$password = $_POST['password'];
$password_confirm = $_POST['password_confirm'];

$json = file_get_contents('./../Models/SellerUser.json');
$data = json_decode($json, true);

$new_user = array(
    'username' => $username,
    'email' => $email,
    'password' => $password,
);

$data[] = $new_user;
$json = json_encode($data);
file_put_contents('./../Models/SellerUser.json', $json);

//profile picture -> pfp
// if (isset($_FILES['pfp'])) {
//     $file_name = $username;
//     $file_tmp = $_FILES['pfp']['tmp_name'];
//     $file_size = $_FILES['pfp']['size'];
//     $file_type = $_FILES['pfp']['type'];

//     $upload_dir = '../Data/SellerData/AccountInfo/img/';
//     $target_file = $upload_dir . basename($file_name) . '.jpg';

//     if (move_uploaded_file($file_tmp, $target_file)) {
//         echo 'File uploaded successfully.';
//     } else {
//         echo 'File upload failed.';
//     }
// }
header('Location: ./../Views/Sellers/SellerLogin.php');
```

[LOCAL VARIABLES]



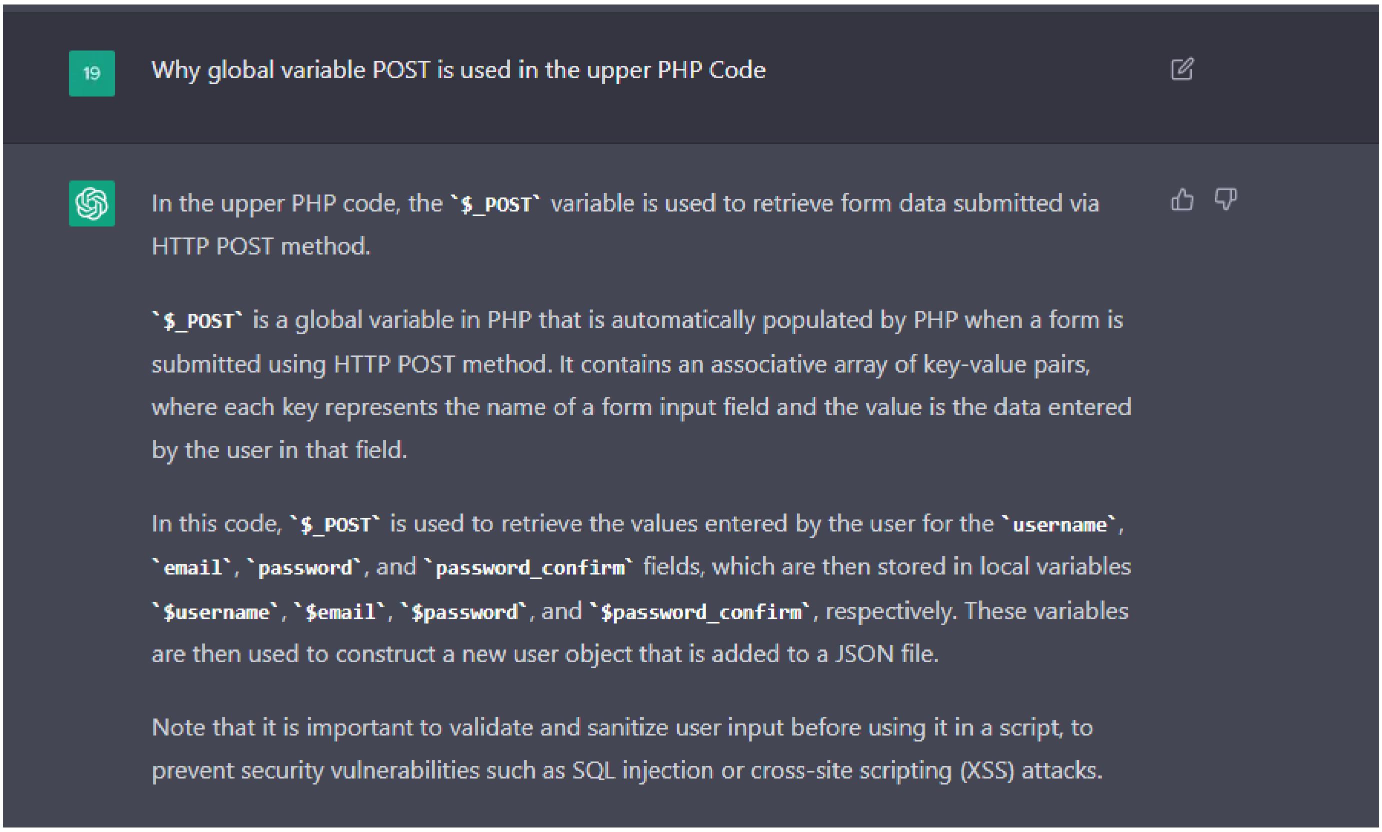
This code portion is a PHP script that performs the following tasks:

- Retrieves data from a form submitted via HTTP POST method and stores them in local variables (`$username`, `$email`, `$password`, and `$password_confirm`).
- Reads the contents of a JSON file (`SellerUser.json`) and parses it into an associative array using `json_decode` function.
- Constructs a new associative array `$new_user` that contains the input data (`$username`, `$email`, and `$password`).
- Adds the `$new_user` array to the end of the `$data` array.
- Encodes the updated `$data` array back to JSON format using `json_encode` function.
- Writes the updated JSON data back to the same file `SellerUser.json` using `file_put_contents` function.
- Redirects the user to the Seller Login page (`SellerLogin.php`) using the `header` function.

There are a few important things to note about this code portion:

1. `$username`, `$email`, `$password`, and `$password_confirm` are local variables defined within the PHP script. They are not accessible outside of the script.
2. `$data` and `$new_user` are also local variables defined within the script, but they are used to modify the contents of an external file (`SellerUser.json`).
3. The `header` function is used to send an HTTP redirect response to the user's browser. It tells the browser to load a different URL, which in this case is `SellerLogin.php`. Note that the `header` function must be called before any actual output is sent to the browser (hence the PHP closure after the function call).
4. This script assumes that `SellerUser.json` is stored in the `./../Models` directory relative to the location of the PHP script.

[POST GLOBAL's Use and Why?]



Why global variable POST is used in the upper PHP Code

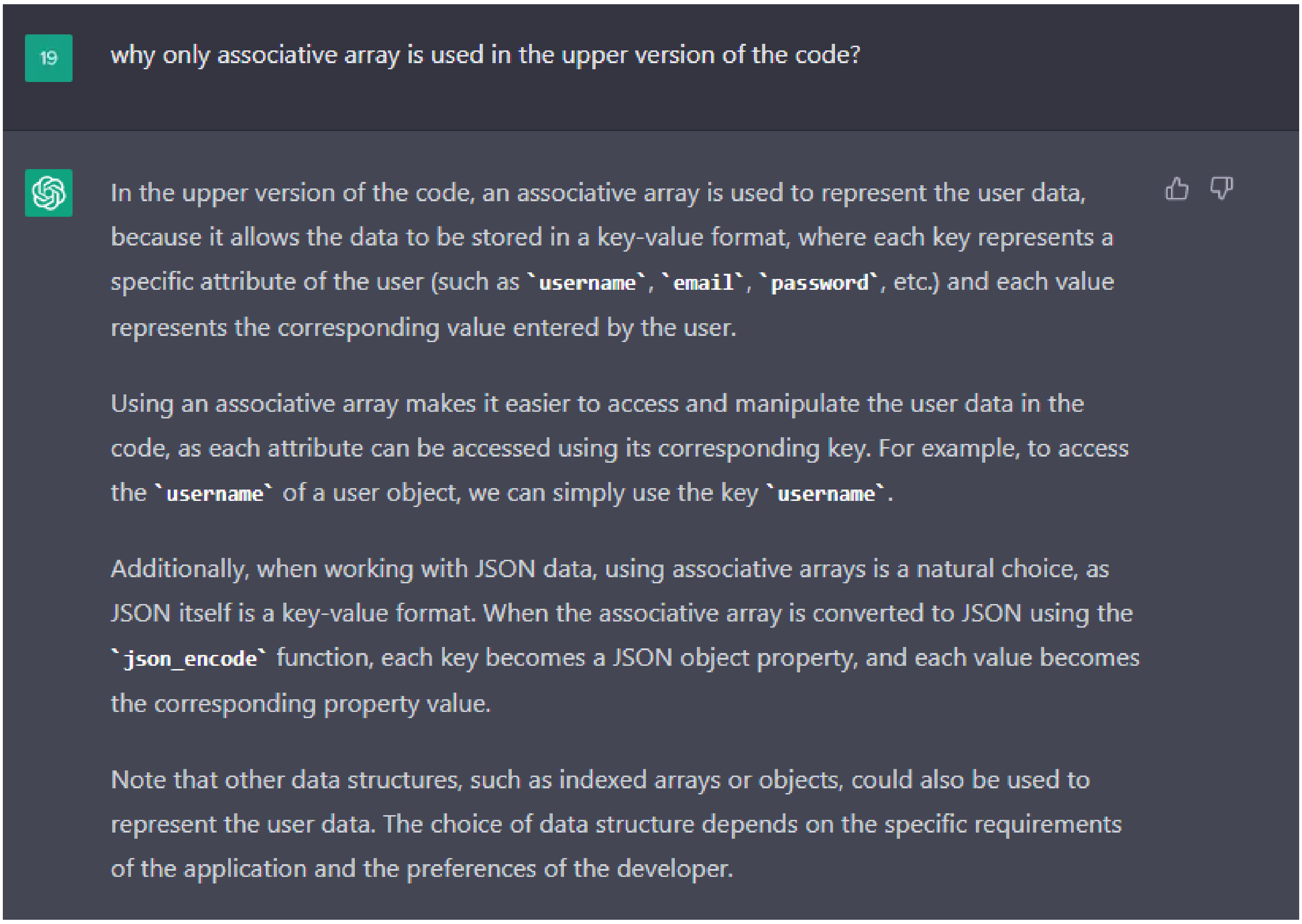
In the upper PHP code, the `$_POST` variable is used to retrieve form data submitted via HTTP POST method.

`$_POST` is a global variable in PHP that is automatically populated by PHP when a form is submitted using HTTP POST method. It contains an associative array of key-value pairs, where each key represents the name of a form input field and the value is the data entered by the user in that field.

In this code, `$_POST` is used to retrieve the values entered by the user for the `username`, `email`, `password`, and `password_confirm` fields, which are then stored in local variables `$username`, `$email`, `$password`, and `$password_confirm`, respectively. These variables are then used to construct a new user object that is added to a JSON file.

Note that it is important to validate and sanitize user input before using it in a script, to prevent security vulnerabilities such as SQL injection or cross-site scripting (XSS) attacks.

[Importance of the Associative Array]



why only associative array is used in the upper version of the code?

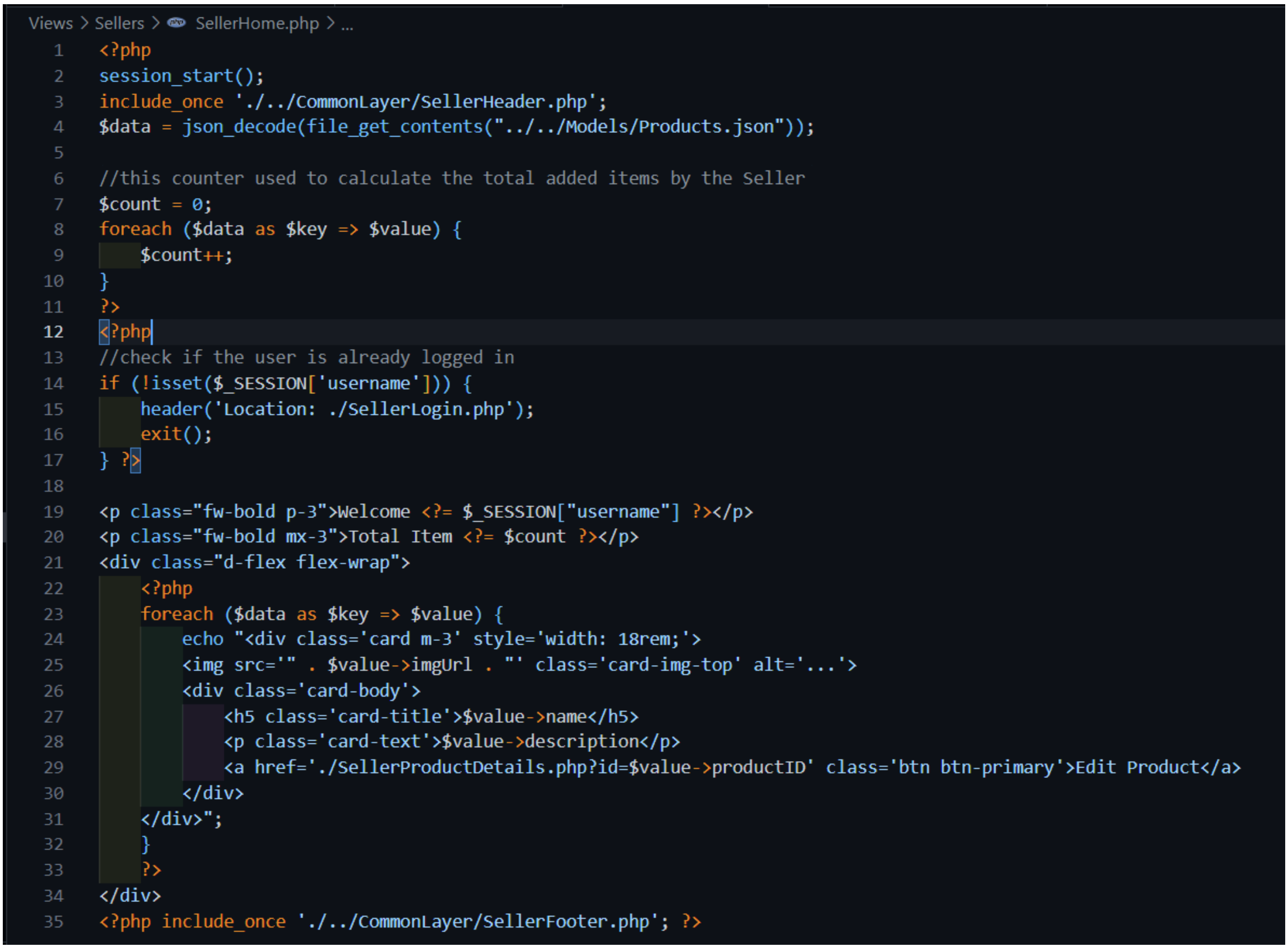
In the upper version of the code, an associative array is used to represent the user data, because it allows the data to be stored in a key-value format, where each key represents a specific attribute of the user (such as `username`, `email`, `password`, etc.) and each value represents the corresponding value entered by the user.

Using an associative array makes it easier to access and manipulate the user data in the code, as each attribute can be accessed using its corresponding key. For example, to access the `username` of a user object, we can simply use the key `username`.

Additionally, when working with JSON data, using associative arrays is a natural choice, as JSON itself is a key-value format. When the associative array is converted to JSON using the `json_encode` function, each key becomes a JSON object property, and each value becomes the corresponding property value.

Note that other data structures, such as indexed arrays or objects, could also be used to represent the user data. The choice of data structure depends on the specific requirements of the application and the preferences of the developer.

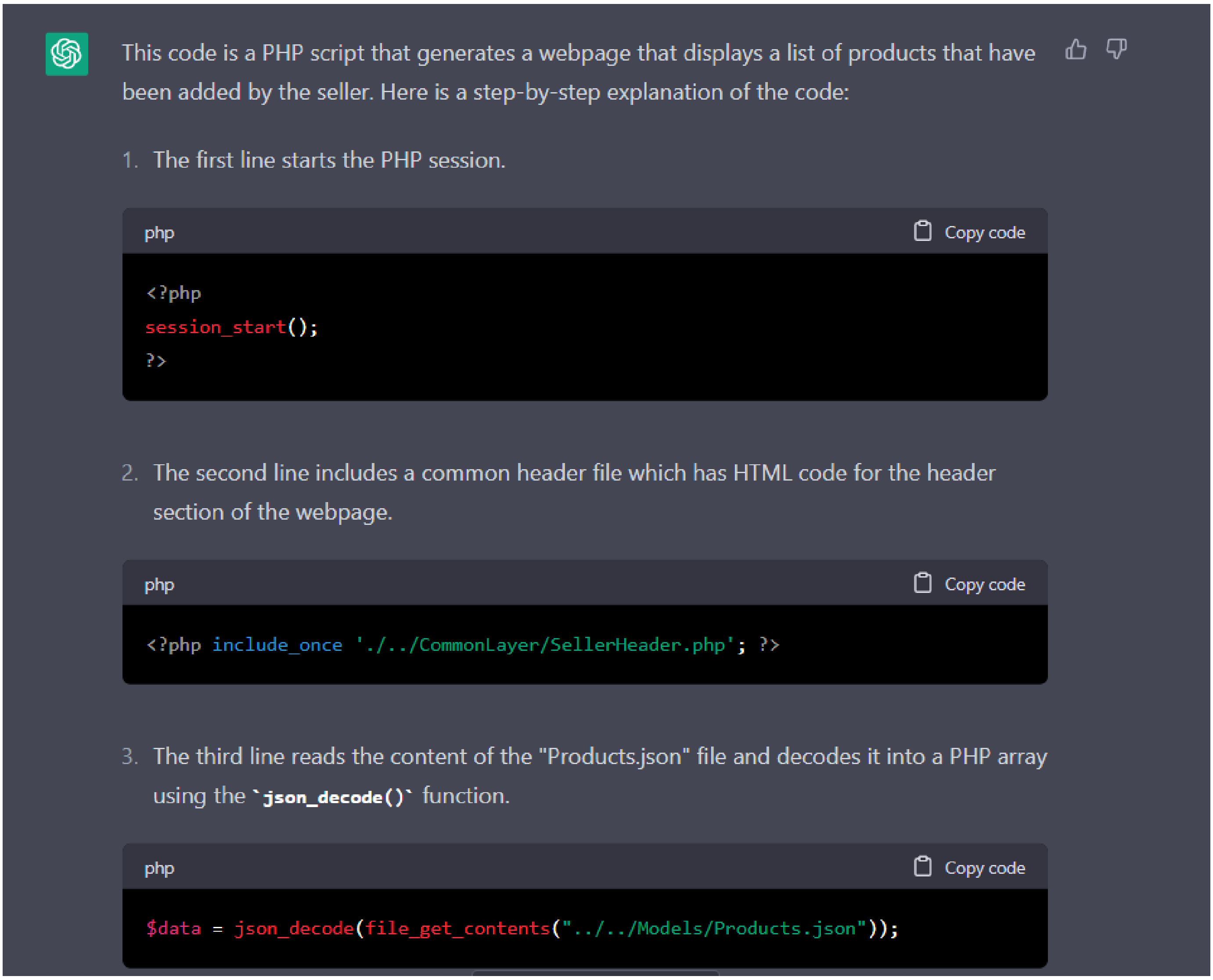
[SELLER HOME [Sellerhome.php] Explained]
Code:



```
<?php
session_start();
include_once './../CommonLayer/SellerHeader.php';
$data = json_decode(file_get_contents("../../Models/Products.json"));

//this counter used to calculate the total added items by the Seller
$count = 0;
foreach ($data as $key => $value) {
    $count++;
}
?>
<?php
//check if the user is already logged in
if (!isset($_SESSION['username'])) {
    header('Location: ./SellerLogin.php');
    exit();
} ?>

<p class="fw-bold p-3">Welcome <?= $_SESSION["username"] ?></p>
<p class="fw-bold mx-3">Total Item <?= $count ?></p>
<div class="d-flex flex-wrap">
    <?php
    foreach ($data as $key => $value) {
        echo "<div class='card m-3' style='width: 18rem;'>
        <img src='" . $value->imgUrl . "' class='card-img-top' alt='...'>
        <div class='card-body'>
            <h5 class='card-title'>$value->name</h5>
            <p class='card-text'>$value->description</p>
            <a href='./SellerProductDetails.php?id=$value->productID' class='btn btn-primary'>Edit Product</a>
        </div>
    </div>";
    }
    ?>
</div>
<?php include_once './../CommonLayer/SellerFooter.php'; ?>
```

Explanation:

[1]



This code is a PHP script that generates a webpage that displays a list of products that have been added by the seller. Here is a step-by-step explanation of the code:

1. The first line starts the PHP session.

```php
<?php
session_start();
?>
```

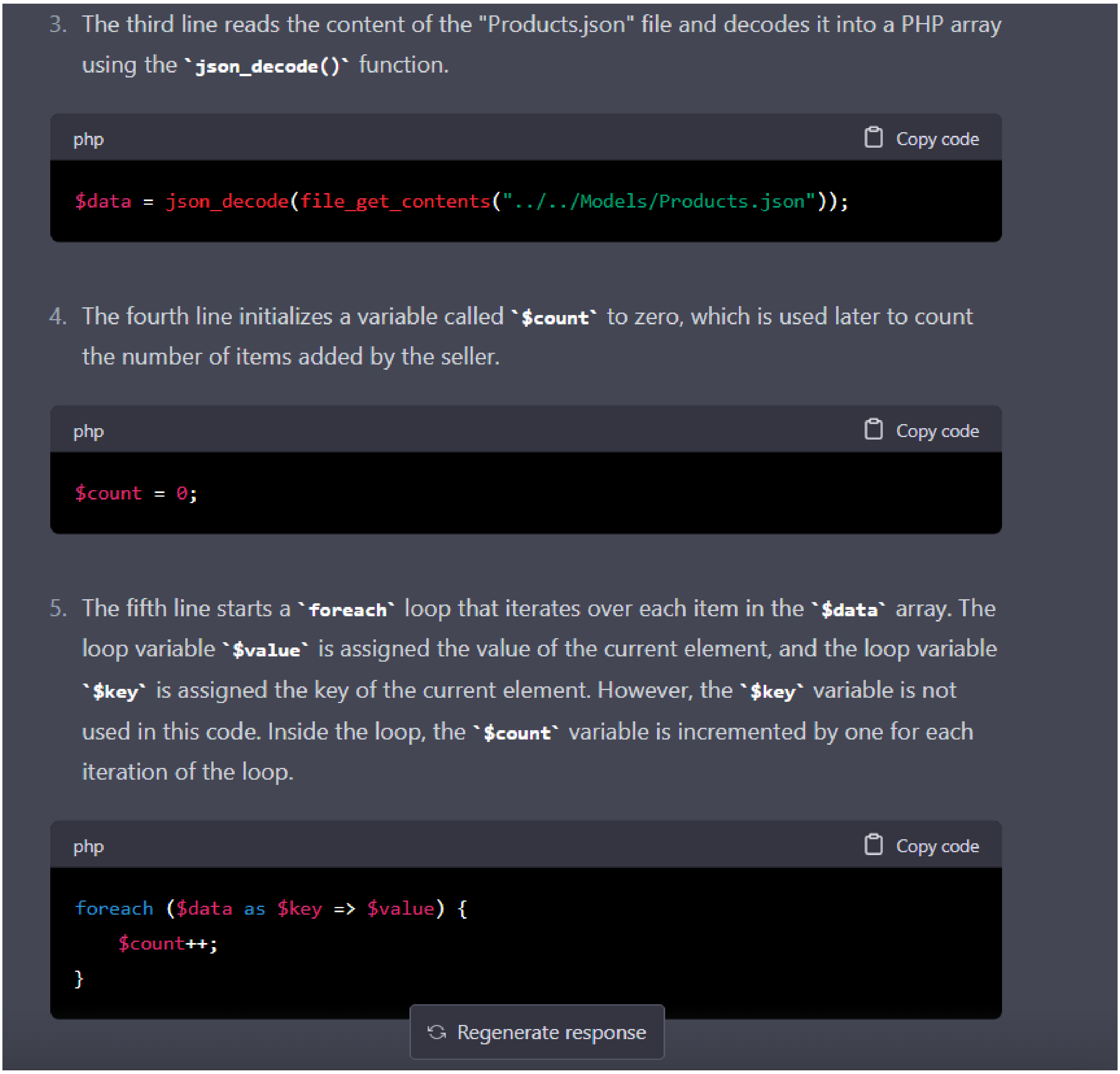
2. The second line includes a common header file which has HTML code for the header section of the webpage.

```php
<?php include_once './../CommonLayer/SellerHeader.php'; ?>
```

3. The third line reads the content of the "Products.json" file and decodes it into a PHP array using the `` `json_decode()` `` function.

```php
$data = json_decode(file_get_contents("../../Models/Products.json"));
```

[2]



3. The third line reads the content of the "Products.json" file and decodes it into a PHP array using the `json_decode()` function.

```php
$data = json_decode(file_get_contents("../../Models/Products.json"));
```

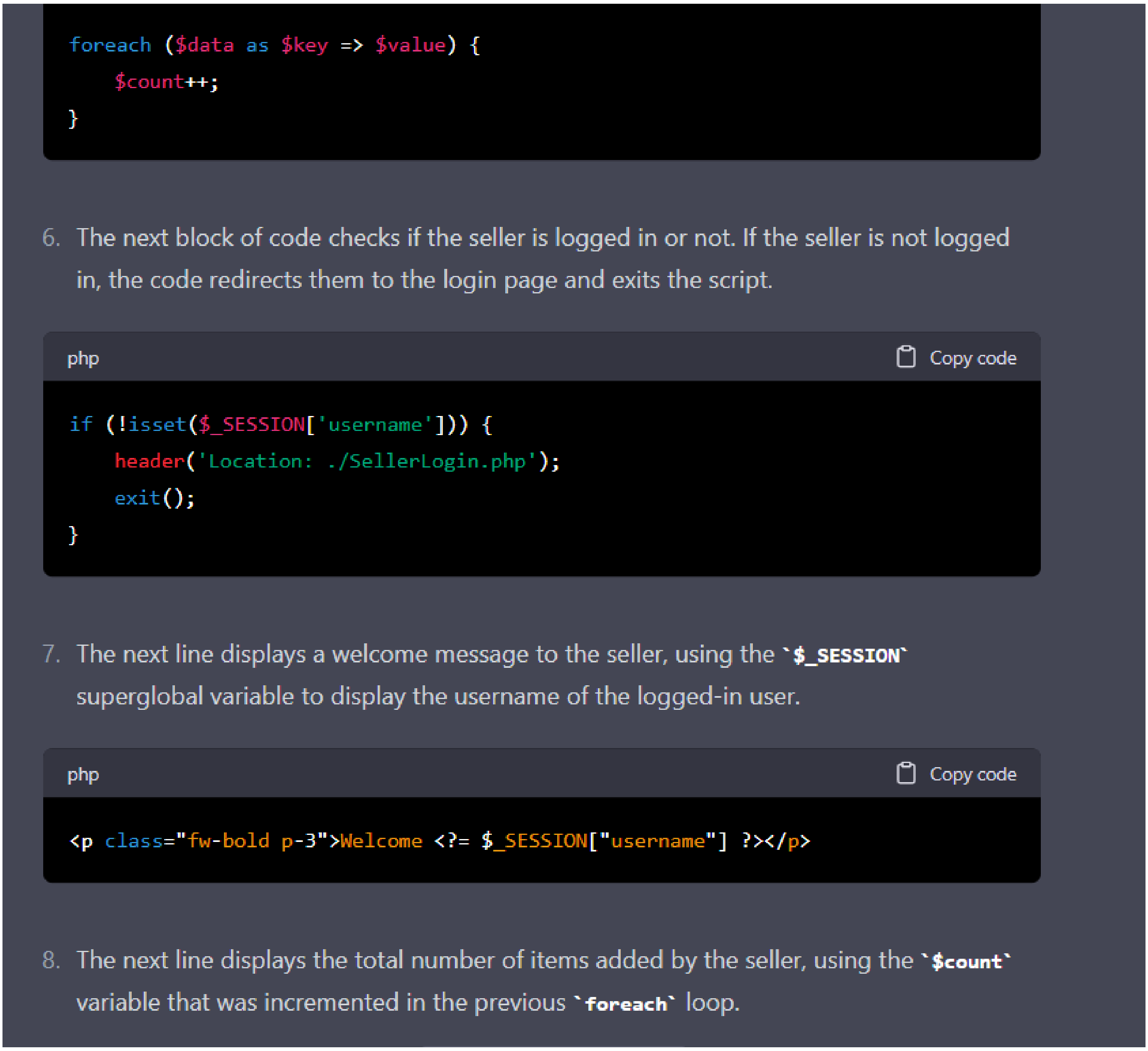
4. The fourth line initializes a variable called `$count` to zero, which is used later to count the number of items added by the seller.

```php
$count = 0;
```

5. The fifth line starts a `foreach` loop that iterates over each item in the `$data` array. The loop variable `$value` is assigned the value of the current element, and the loop variable `$key` is assigned the key of the current element. However, the `$key` variable is not used in this code. Inside the loop, the `$count` variable is incremented by one for each iteration of the loop.

```php
foreach ($data as $key => $value) {
    $count++;
}
```

[3]



```
foreach ($data as $key => $value) {
    $count++;
}
```

6. The next block of code checks if the seller is logged in or not. If the seller is not logged in, the code redirects them to the login page and exits the script.

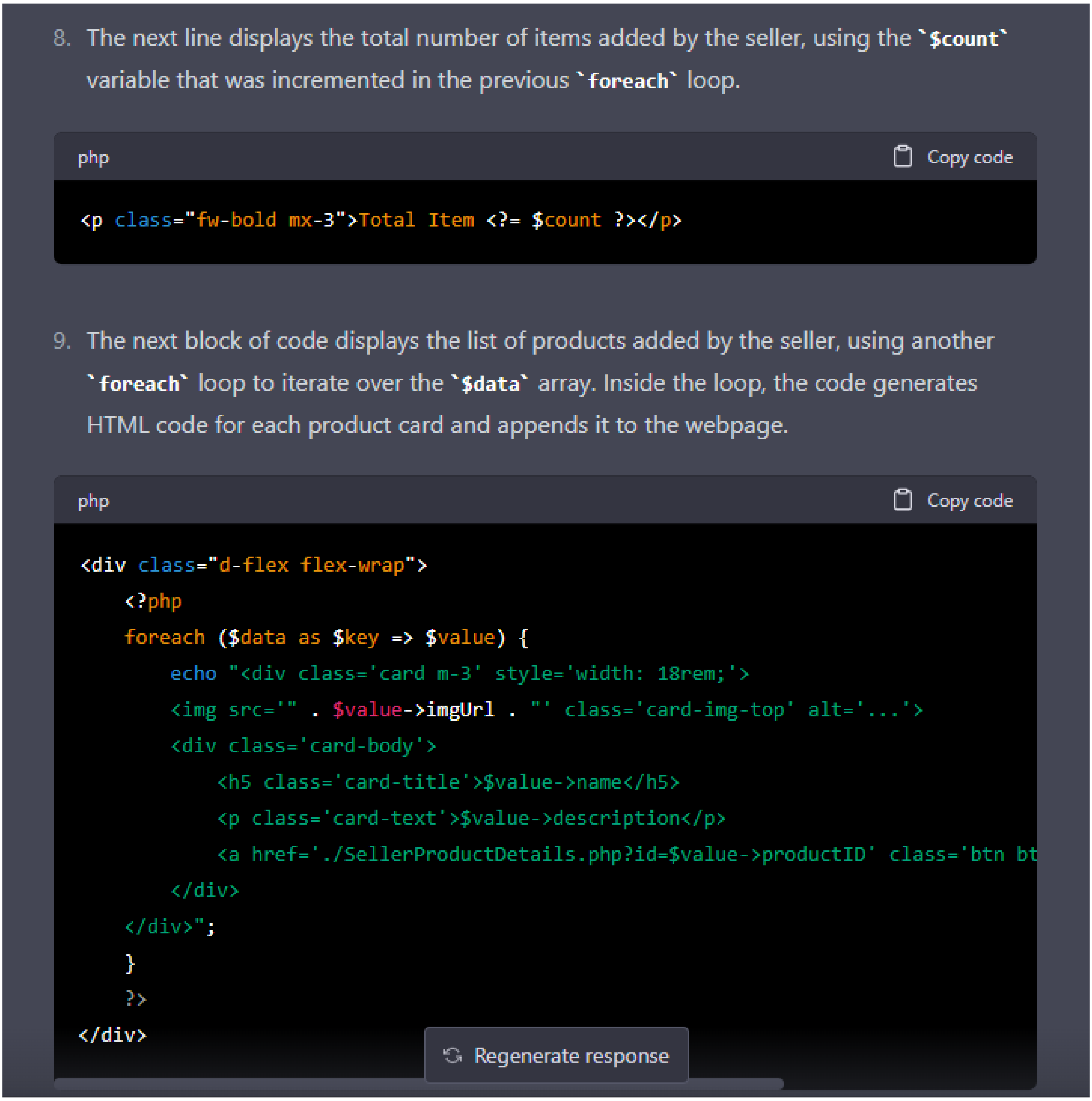
```php
if (!isset($_SESSION['username'])) {
    header('Location: ./SellerLogin.php');
    exit();
}
```

7. The next line displays a welcome message to the seller, using the `` `$_SESSION` `` superglobal variable to display the username of the logged-in user.

```php
<p class="fw-bold p-3">Welcome <?= $_SESSION["username"] ?></p>
```

8. The next line displays the total number of items added by the seller, using the `` `$count` `` variable that was incremented in the previous `` `foreach` `` loop.

[4]



8. The next line displays the total number of items added by the seller, using the `$count` variable that was incremented in the previous `foreach` loop.

```php
<p class="fw-bold mx-3">Total Item <?= $count ?></p>
```

9. The next block of code displays the list of products added by the seller, using another `foreach` loop to iterate over the `$data` array. Inside the loop, the code generates HTML code for each product card and appends it to the webpage.

```php
<div class="d-flex flex-wrap">
    <?php
    foreach ($data as $key => $value) {
        echo "<div class='card m-3' style='width: 18rem;'>
        <img src='" . $value->imgUrl . "' class='card-img-top' alt='...'>
        <div class='card-body'>
            <h5 class='card-title'>$value->name</h5>
            <p class='card-text'>$value->description</p>
            <a href='./SellerProductDetails.php?id=$value->productID' class='btn bt
        </div>
    </div>";
    }
    ?>
</div>
```

[5]

```php
<div class="d-flex flex-wrap">
    <?php
    foreach ($data as $key => $value) {
        echo "<div class='card m-3' style='width: 18rem;'>
        <img src='" . $value->imgUrl . "' class='card-img-top' alt='...'>
        <div class='card-body'>
            <h5 class='card-title'>$value->name</h5>
            <p class='card-text'>$value->description</p>
            <a href='./SellerProductDetails.php?id=$value->productID' class='btn bt
        </div>
    </div>";
    }
    ?>
</div>
```

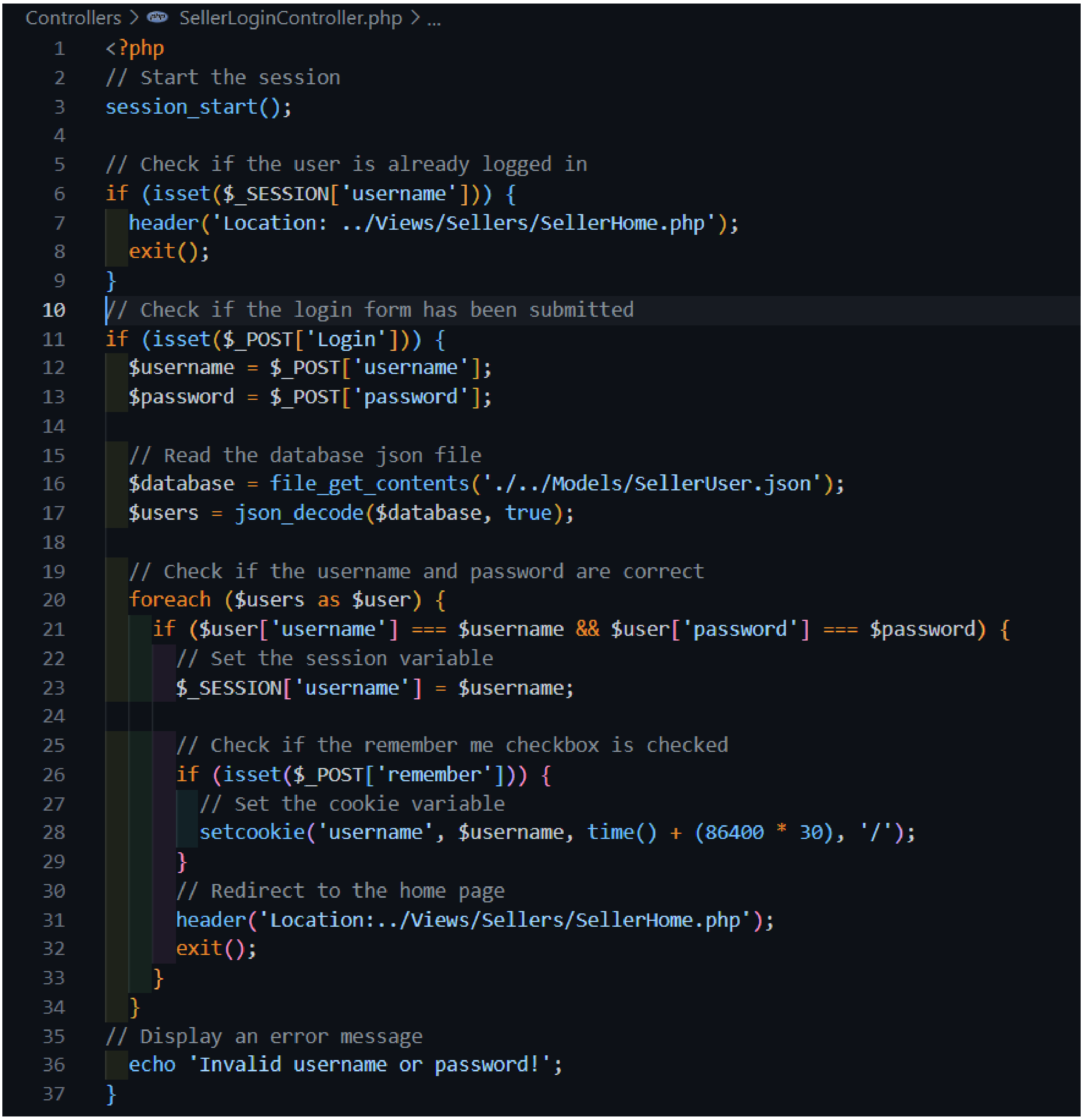
10. The last line includes a common footer file which has HTML code for the footer section of the webpage.

```php
<?php include_once './../CommonLayer/SellerFooter.php'; ?>
```

In summary, this code generates a webpage that displays a list of products added by the seller. The code reads product data from a JSON file, counts the number of products added by the seller, and displays the product list along with a welcome message

---

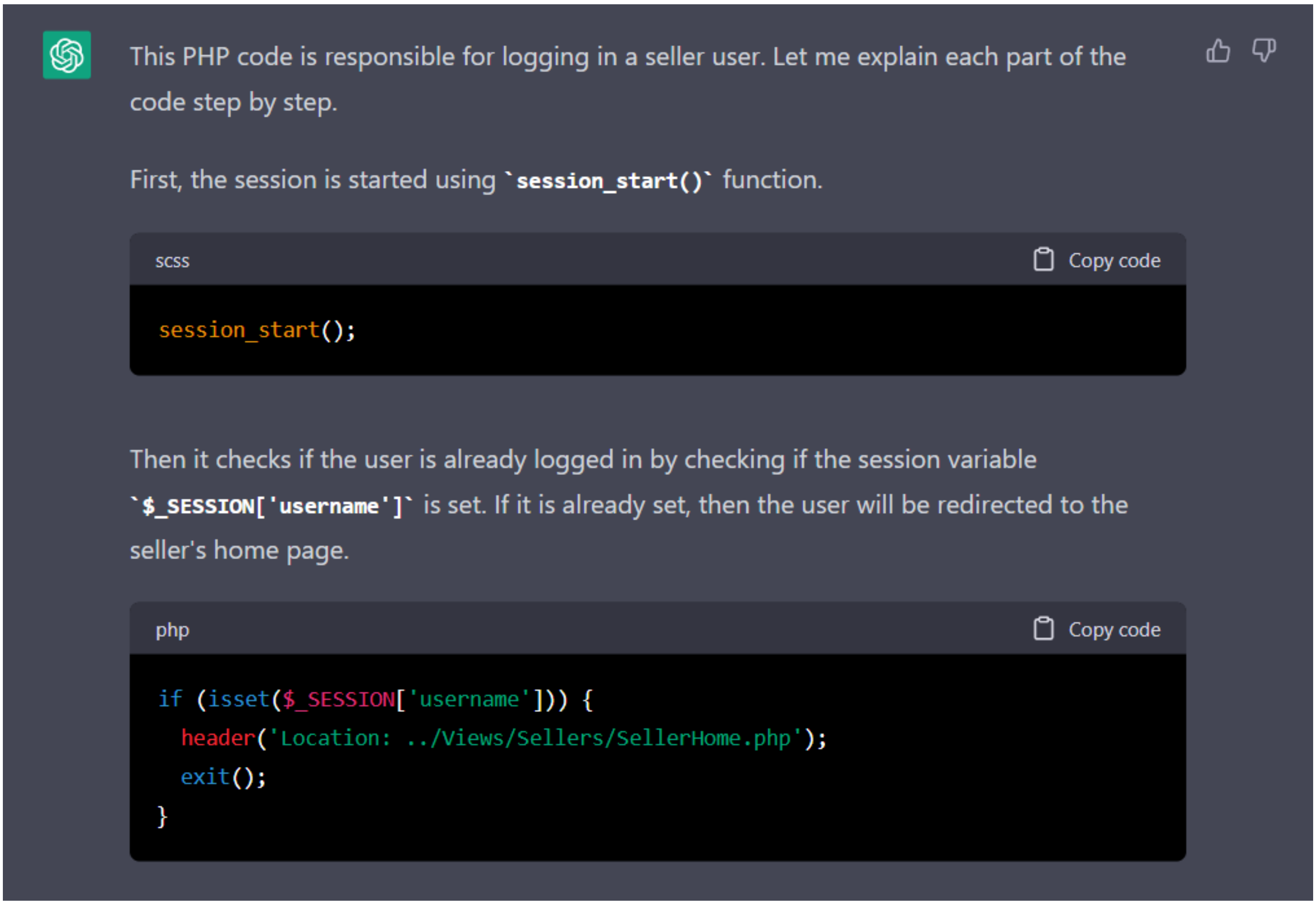
[[SellerLoginController.php] Seller's Login Controller Explained]
Code:



```php
<?php
// Start the session
session_start();

// Check if the user is already logged in
if (isset($_SESSION['username'])) {
    header('Location: ../Views/Sellers/SellerHome.php');
    exit();
}
// Check if the login form has been submitted
if (isset($_POST['Login'])) {
    $username = $_POST['username'];
    $password = $_POST['password'];

    // Read the database json file
    $database = file_get_contents('./../Models/SellerUser.json');
    $users = json_decode($database, true);

    // Check if the username and password are correct
    foreach ($users as $user) {
        if ($user['username'] === $username && $user['password'] === $password) {
            // Set the session variable
            $_SESSION['username'] = $username;

            // Check if the remember me checkbox is checked
            if (isset($_POST['remember'])) {
                // Set the cookie variable
                setcookie('username', $username, time() + (86400 * 30), '/');
            }
            // Redirect to the home page
            header('Location:../Views/Sellers/SellerHome.php');
            exit();
        }
    }
// Display an error message
    echo 'Invalid username or password!';
}
```

Explanation:

[1]



This PHP code is responsible for logging in a seller user. Let me explain each part of the code step by step.

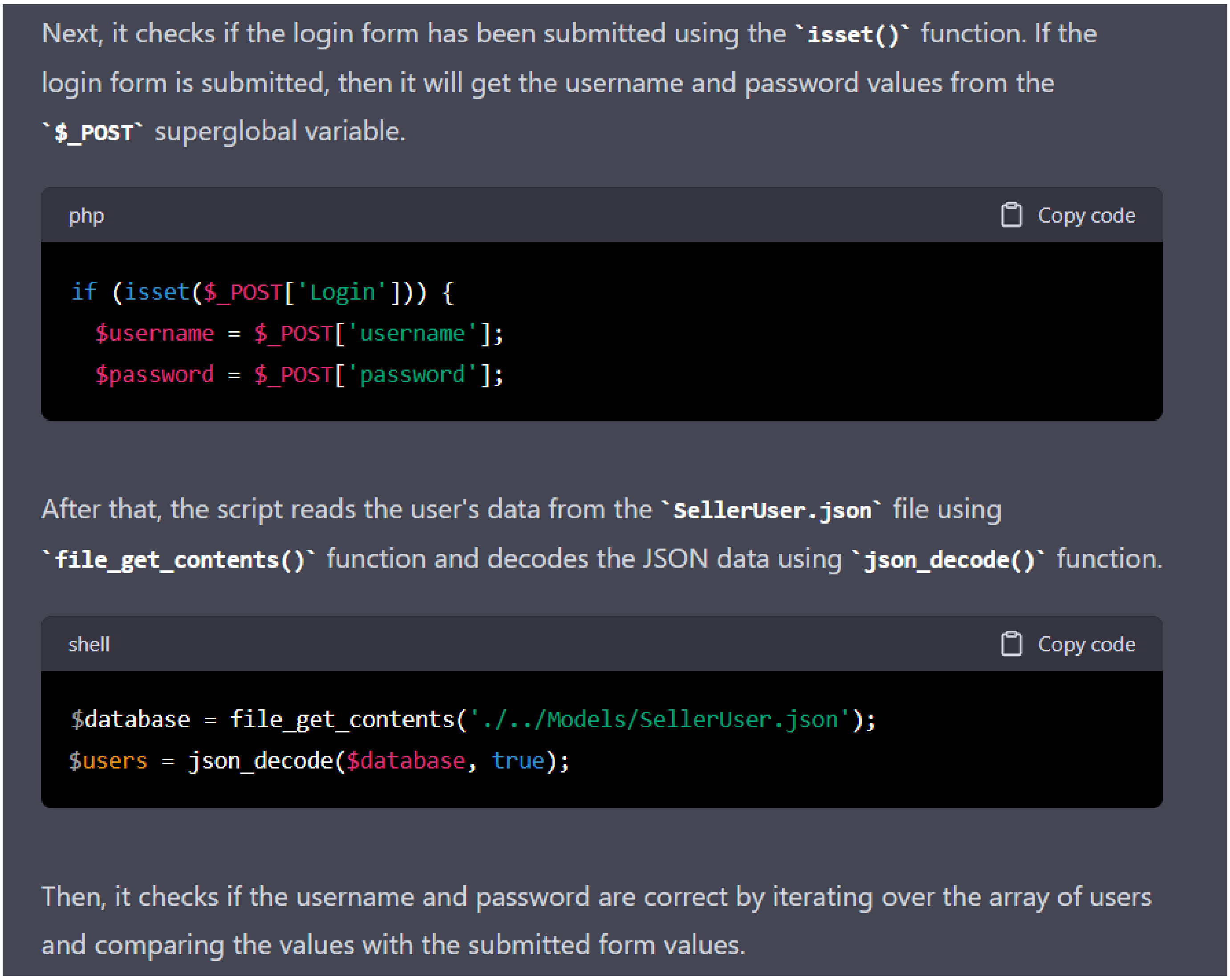
First, the session is started using `session_start()` function.

```scss
session_start();
```

Then it checks if the user is already logged in by checking if the session variable `$_SESSION['username']` is set. If it is already set, then the user will be redirected to the seller's home page.

```php
if (isset($_SESSION['username'])) {
  header('Location: ../Views/Sellers/SellerHome.php');
  exit();
}
```

[2]



Next, it checks if the login form has been submitted using the `isset()` function. If the login form is submitted, then it will get the username and password values from the `$_POST` superglobal variable.

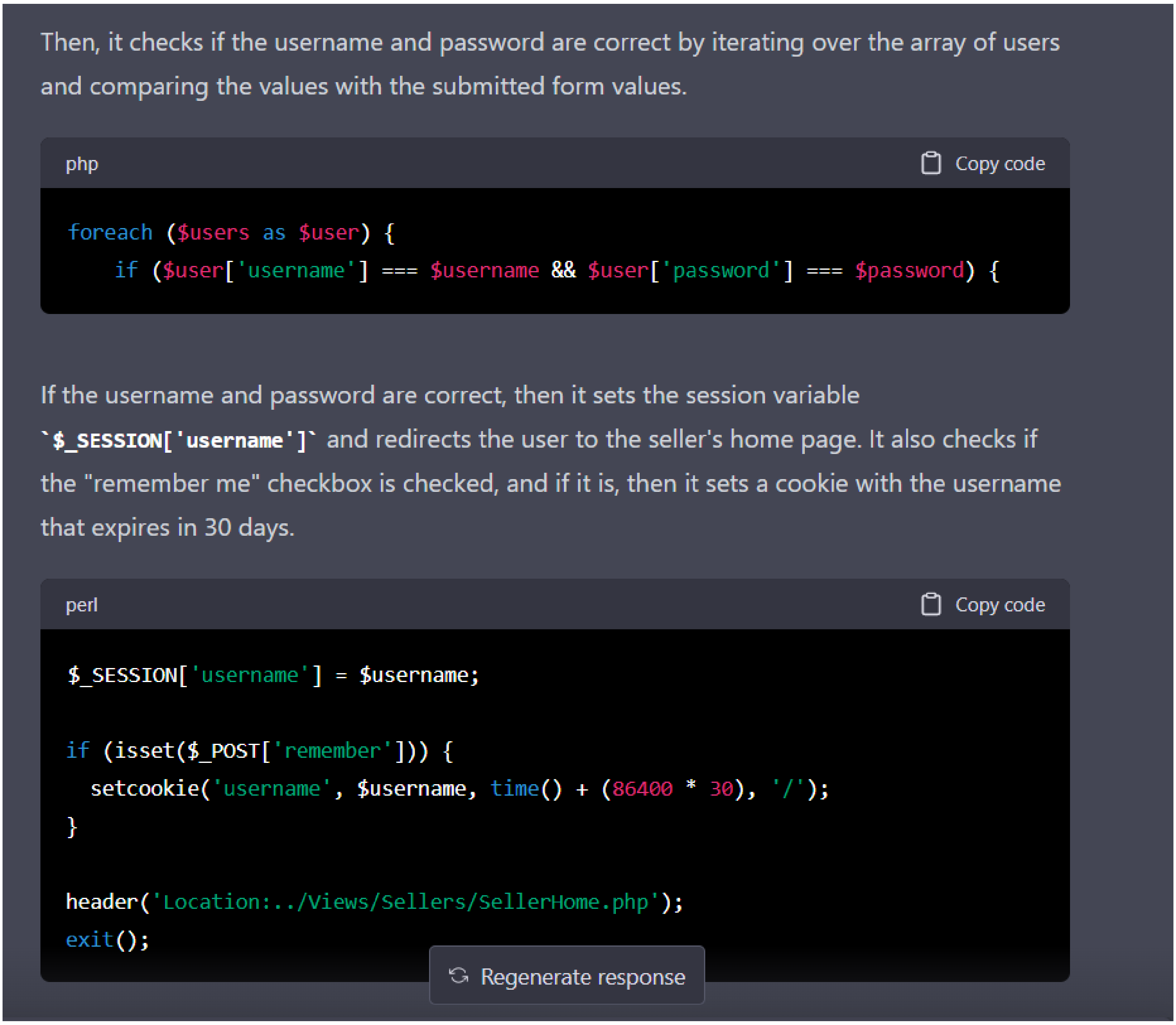
```php
if (isset($_POST['Login'])) {
  $username = $_POST['username'];
  $password = $_POST['password'];
```

After that, the script reads the user's data from the `SellerUser.json` file using `file_get_contents()` function and decodes the JSON data using `json_decode()` function.

```shell
$database = file_get_contents('./../Models/SellerUser.json');
$users = json_decode($database, true);
```

Then, it checks if the username and password are correct by iterating over the array of users and comparing the values with the submitted form values.

[3]



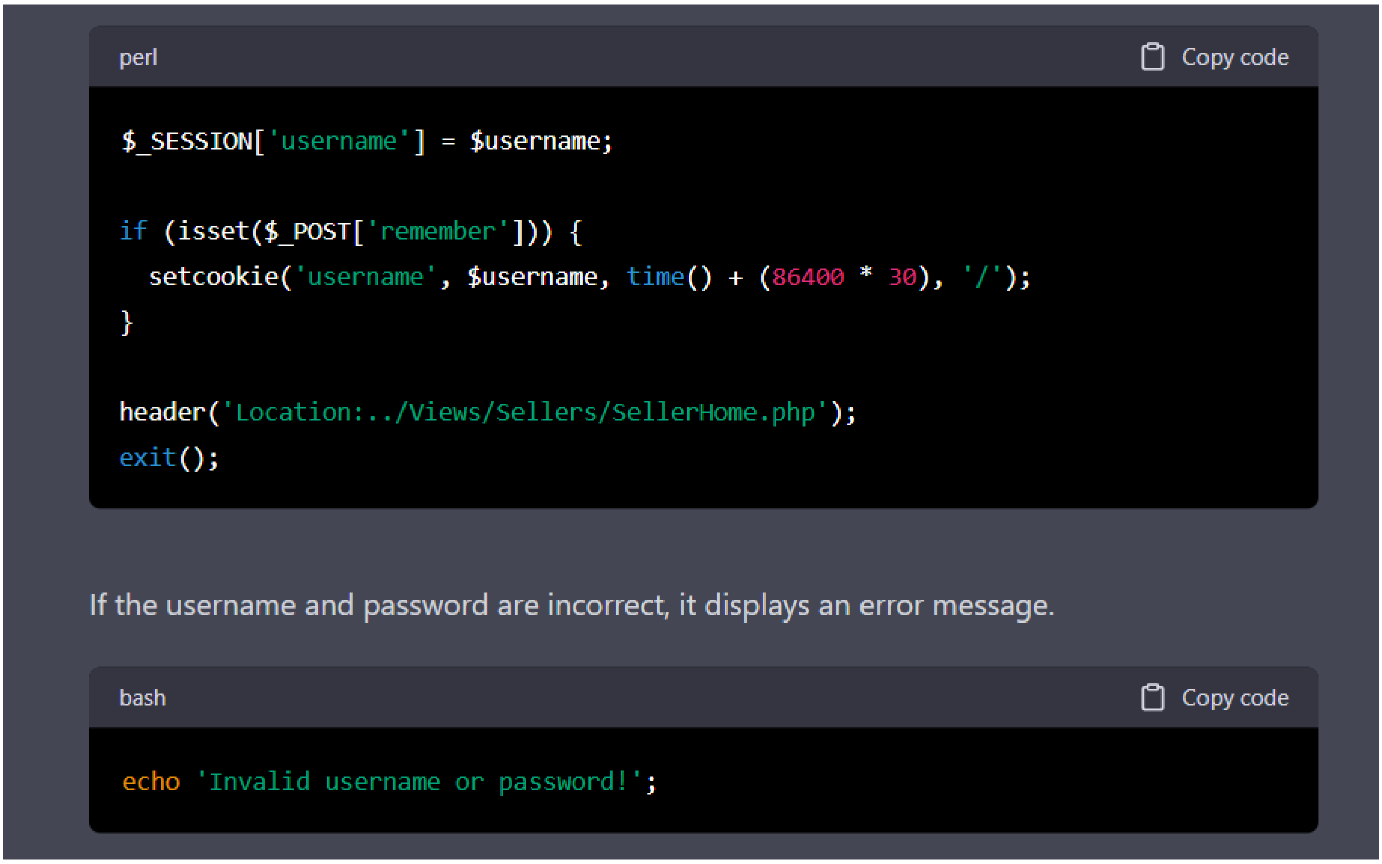
Then, it checks if the username and password are correct by iterating over the array of users and comparing the values with the submitted form values.

```php
foreach ($users as $user) {
    if ($user['username'] === $username && $user['password'] === $password) {
```

If the username and password are correct, then it sets the session variable `` `$_SESSION['username']` `` and redirects the user to the seller's home page. It also checks if the "remember me" checkbox is checked, and if it is, then it sets a cookie with the username that expires in 30 days.

```perl
$_SESSION['username'] = $username;

if (isset($_POST['remember'])) {
  setcookie('username', $username, time() + (86400 * 30), '/');
}

header('Location:../Views/Sellers/SellerHome.php');
exit();
```

[4]

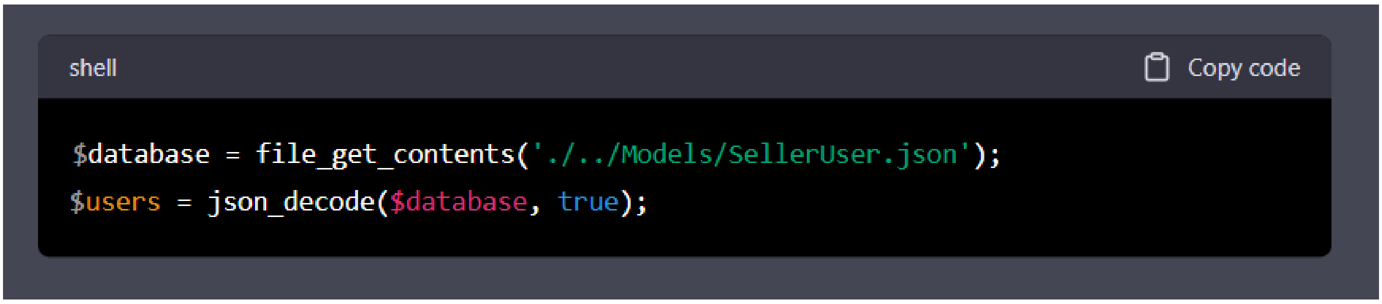


```perl
$_SESSION['username'] = $username;

if (isset($_POST['remember'])) {
  setcookie('username', $username, time() + (86400 * 30), '/');
}

header('Location:../Views/Sellers/SellerHome.php');
exit();
```

If the username and password are incorrect, it displays an error message.

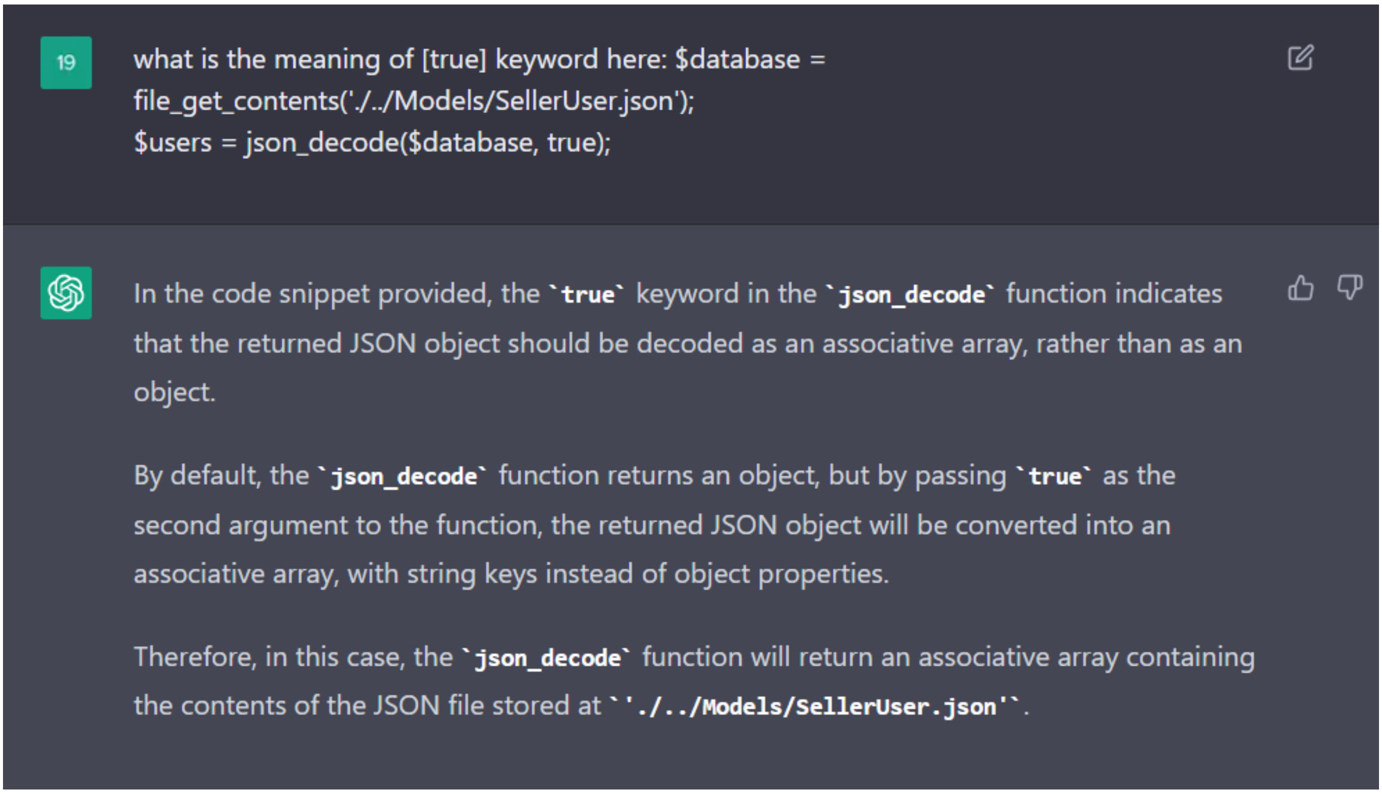
```bash
echo 'Invalid username or password!';
```

[5]



```shell
$database = file_get_contents('./../Models/SellerUser.json');
$users = json_decode($database, true);
```

[6]



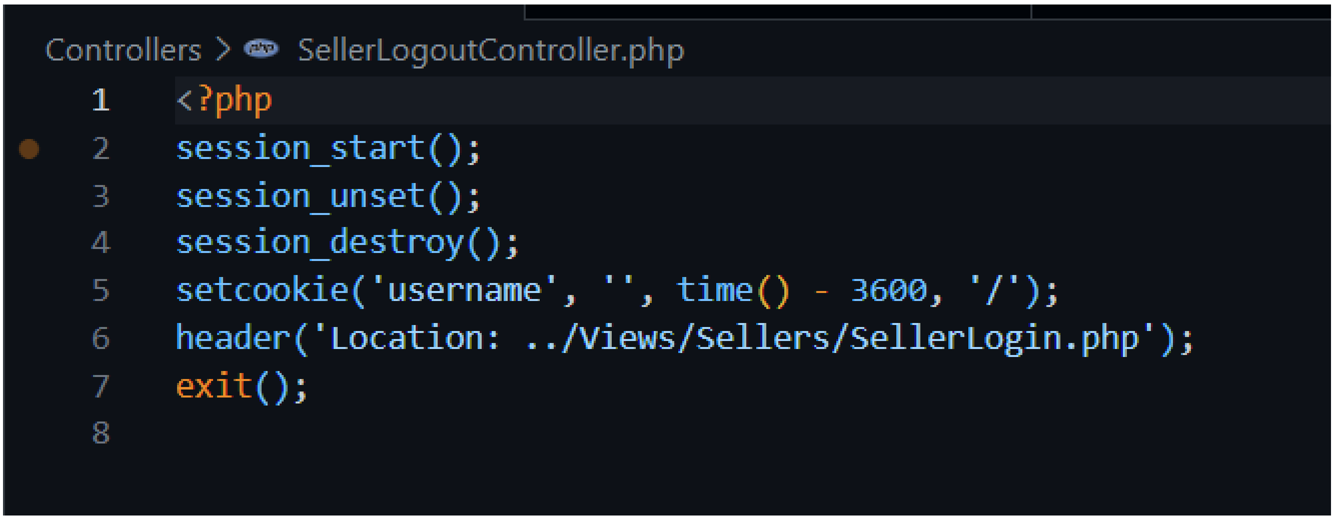
what is the meaning of [true] keyword here: $database = file_get_contents('./../Models/SellerUser.json');
$users = json_decode($database, true);

In the code snippet provided, the `true` keyword in the `json_decode` function indicates that the returned JSON object should be decoded as an associative array, rather than as an object.

By default, the `json_decode` function returns an object, but by passing `true` as the second argument to the function, the returned JSON object will be converted into an associative array, with string keys instead of object properties.

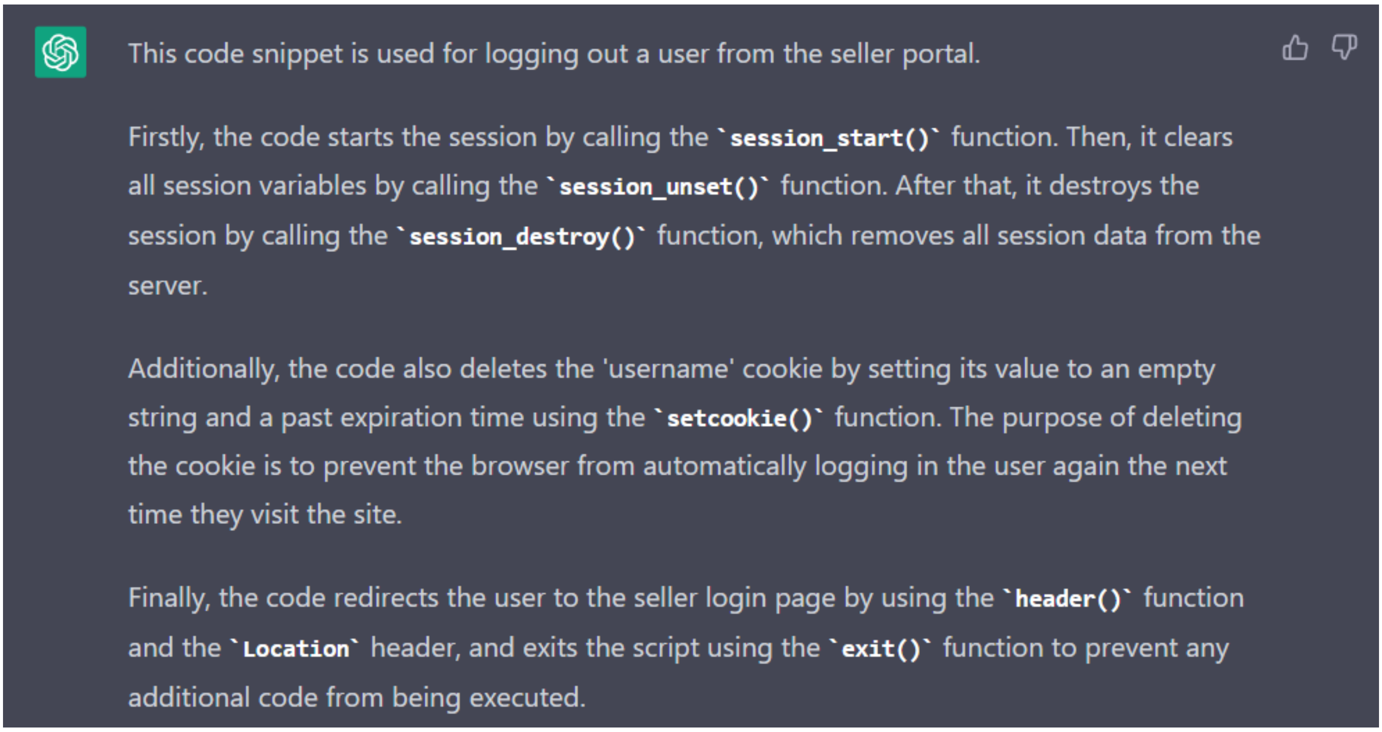
Therefore, in this case, the `json_decode` function will return an associative array containing the contents of the JSON file stored at `'./../Models/SellerUser.json'`.

[[SellerLogoutController.php] Seller's Logout Controller Explained]
Code:



```php
<?php
session_start();
session_unset();
session_destroy();
setcookie('username', '', time() - 3600, '/');
header('Location: ../Views/Sellers/SellerLogin.php');
exit();

```

Explanation:



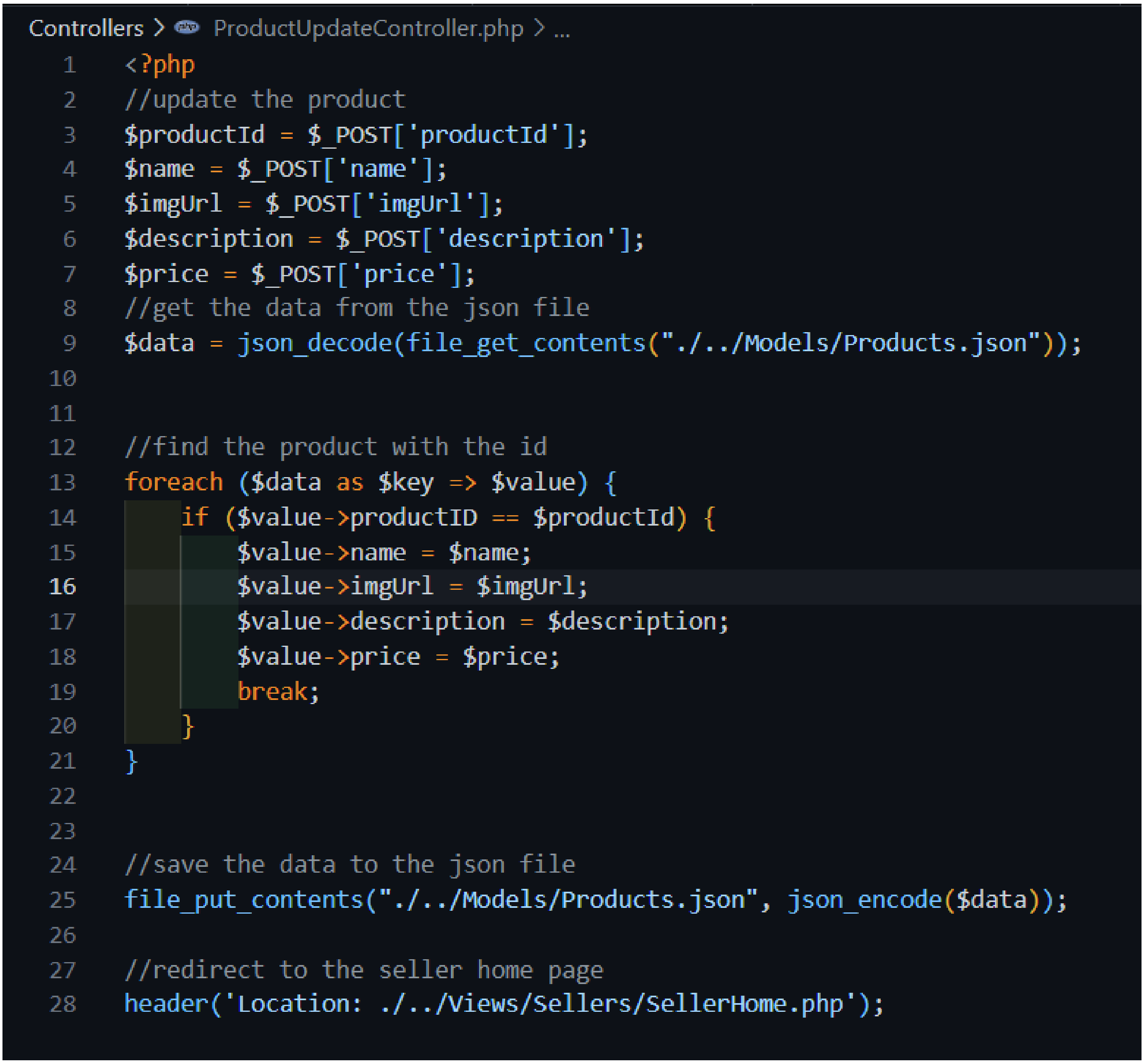
This code snippet is used for logging out a user from the seller portal.

Firstly, the code starts the session by calling the `` `session_start()` `` function. Then, it clears all session variables by calling the `` `session_unset()` `` function. After that, it destroys the session by calling the `` `session_destroy()` `` function, which removes all session data from the server.

Additionally, the code also deletes the 'username' cookie by setting its value to an empty string and a past expiration time using the `` `setcookie()` `` function. The purpose of deleting the cookie is to prevent the browser from automatically logging in the user again the next time they visit the site.

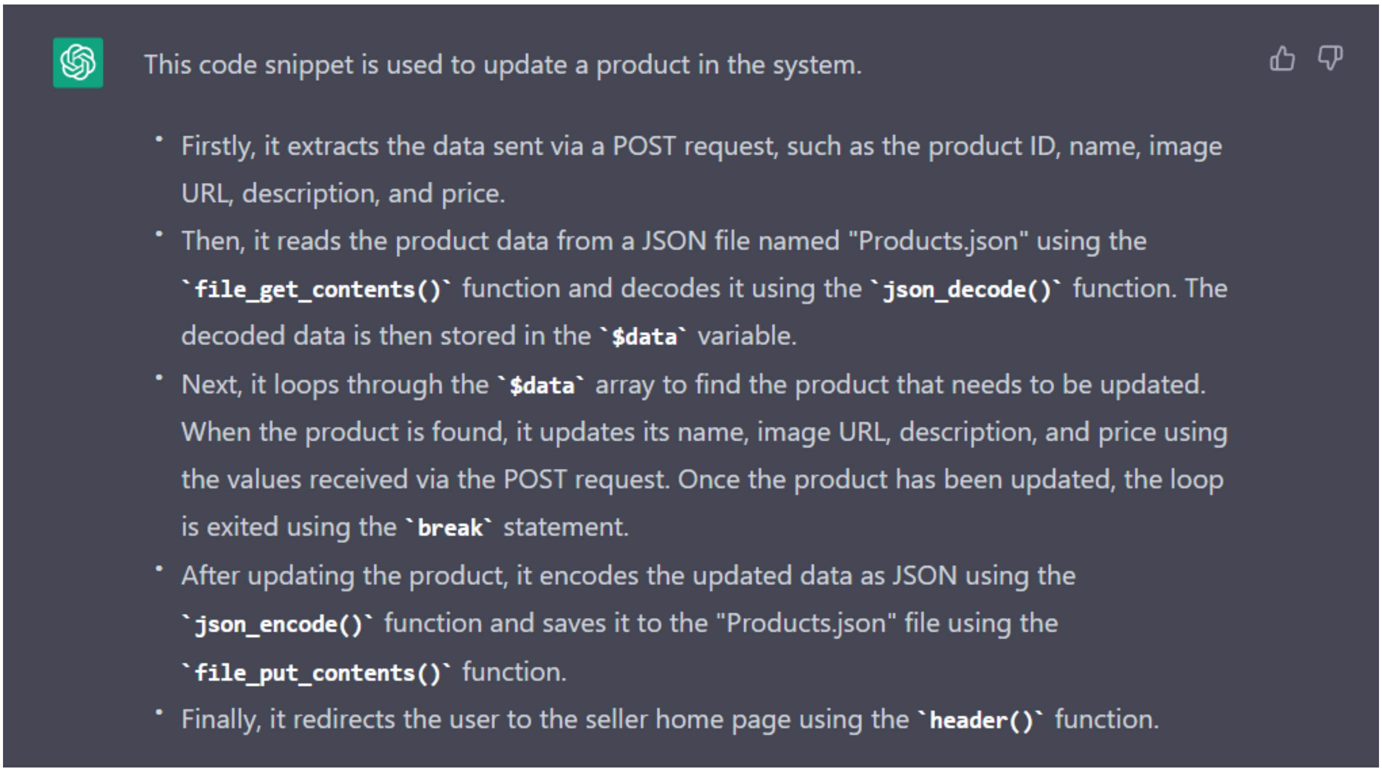
Finally, the code redirects the user to the seller login page by using the `` `header()` `` function and the `` `Location` `` header, and exits the script using the `` `exit()` `` function to prevent any additional code from being executed.

[[ProductUpdateController.php] Seller's Product edit module explained here]
Code:



```php
<?php
//update the product
$productId = $_POST['productId'];
$name = $_POST['name'];
$imgUrl = $_POST['imgUrl'];
$description = $_POST['description'];
$price = $_POST['price'];
//get the data from the json file
$data = json_decode(file_get_contents("./../Models/Products.json"));


//find the product with the id
foreach ($data as $key => $value) {
    if ($value->productID == $productId) {
        $value->name = $name;
        $value->imgUrl = $imgUrl;
        $value->description = $description;
        $value->price = $price;
        break;
    }
}


//save the data to the json file
file_put_contents("./../Models/Products.json", json_encode($data));

//redirect to the seller home page
header('Location: ./../Views/Sellers/SellerHome.php');
```

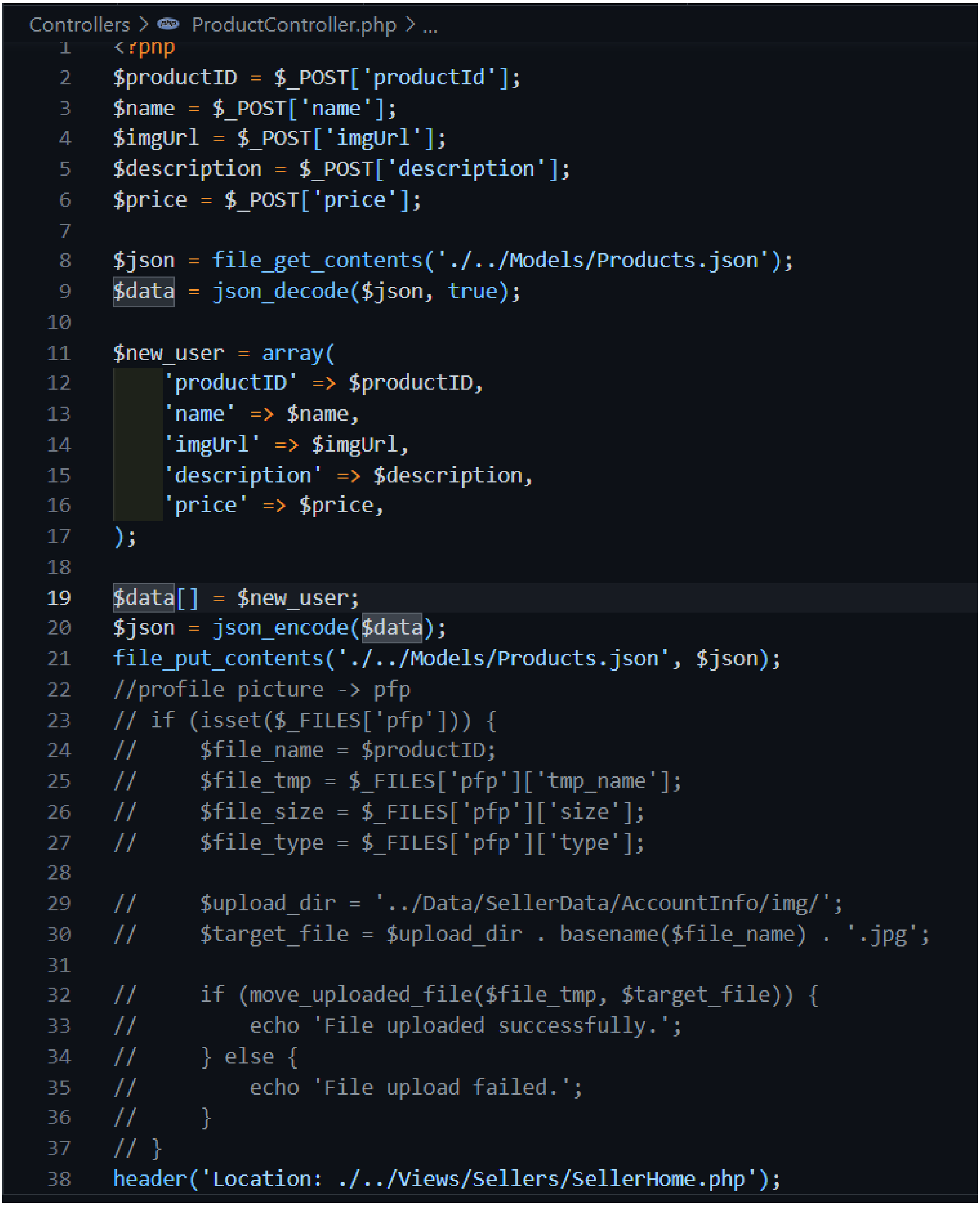
Explanation:



This code snippet is used to update a product in the system.

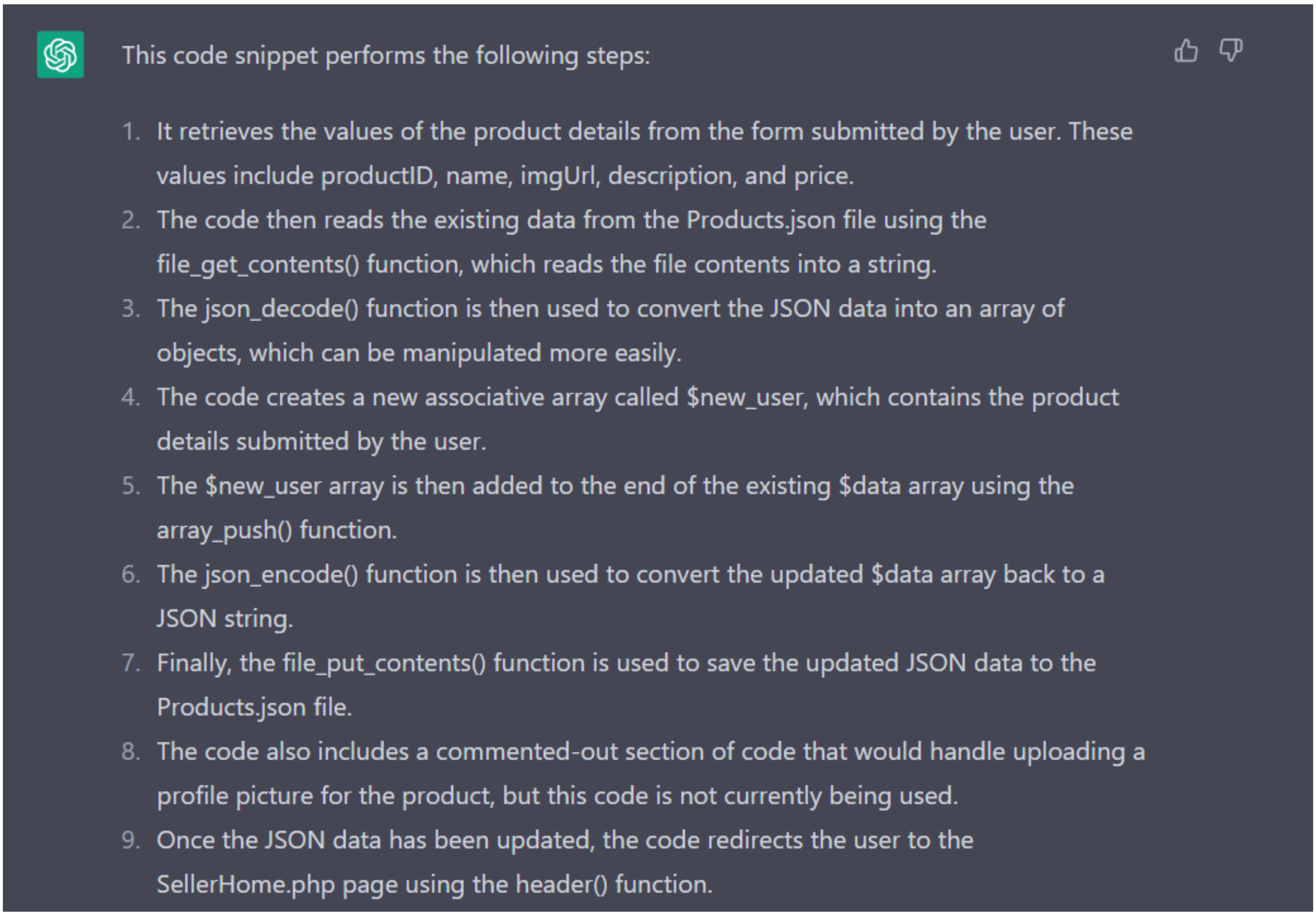
- Firstly, it extracts the data sent via a POST request, such as the product ID, name, image URL, description, and price.
- Then, it reads the product data from a JSON file named "Products.json" using the `file_get_contents()` function and decodes it using the `json_decode()` function. The decoded data is then stored in the `$data` variable.
- Next, it loops through the `$data` array to find the product that needs to be updated. When the product is found, it updates its name, image URL, description, and price using the values received via the POST request. Once the product has been updated, the loop is exited using the `break` statement.
- After updating the product, it encodes the updated data as JSON using the `json_encode()` function and saves it to the "Products.json" file using the `file_put_contents()` function.
- Finally, it redirects the user to the seller home page using the `header()` function.

[[ProductController.php] Seller's CreateProduct module that this controller handles, now will be explained]
Code:



```php
<?php
$productID = $_POST['productId'];
$name = $_POST['name'];
$imgUrl = $_POST['imgUrl'];
$description = $_POST['description'];
$price = $_POST['price'];

$json = file_get_contents('./../Models/Products.json');
$data = json_decode($json, true);

$new_user = array(
    'productID' => $productID,
    'name' => $name,
    'imgUrl' => $imgUrl,
    'description' => $description,
    'price' => $price,
);

$data[] = $new_user;
$json = json_encode($data);
file_put_contents('./../Models/Products.json', $json);
//profile picture -> pfp
// if (isset($_FILES['pfp'])) {
//     $file_name = $productID;
//     $file_tmp = $_FILES['pfp']['tmp_name'];
//     $file_size = $_FILES['pfp']['size'];
//     $file_type = $_FILES['pfp']['type'];

//     $upload_dir = '../Data/SellerData/AccountInfo/img/';
//     $target_file = $upload_dir . basename($file_name) . '.jpg';

//     if (move_uploaded_file($file_tmp, $target_file)) {
//         echo 'File uploaded successfully.';
//     } else {
//         echo 'File upload failed.';
//     }
// }
header('Location: ./../Views/Sellers/SellerHome.php');
```

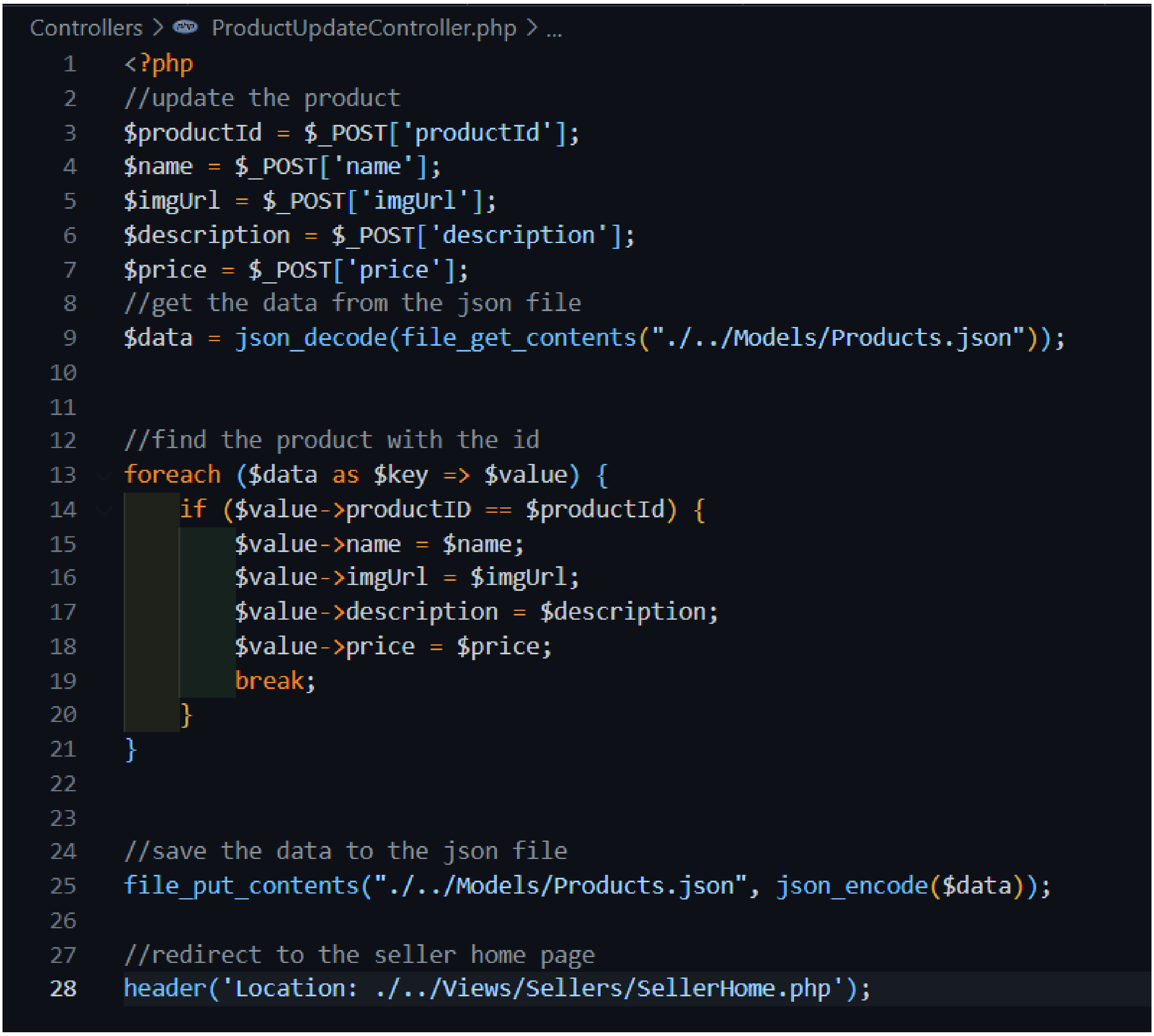
Explanation:



This code snippet performs the following steps:

1. It retrieves the values of the product details from the form submitted by the user. These values include productID, name, imgUrl, description, and price.
2. The code then reads the existing data from the Products.json file using the file_get_contents() function, which reads the file contents into a string.
3. The json_decode() function is then used to convert the JSON data into an array of objects, which can be manipulated more easily.
4. The code creates a new associative array called $new_user, which contains the product details submitted by the user.
5. The $new_user array is then added to the end of the existing $data array using the array_push() function.
6. The json_encode() function is then used to convert the updated $data array back to a JSON string.
7. Finally, the file_put_contents() function is used to save the updated JSON data to the Products.json file.
8. The code also includes a commented-out section of code that would handle uploading a profile picture for the product, but this code is not currently being used.
9. Once the JSON data has been updated, the code redirects the user to the SellerHome.php page using the header() function.

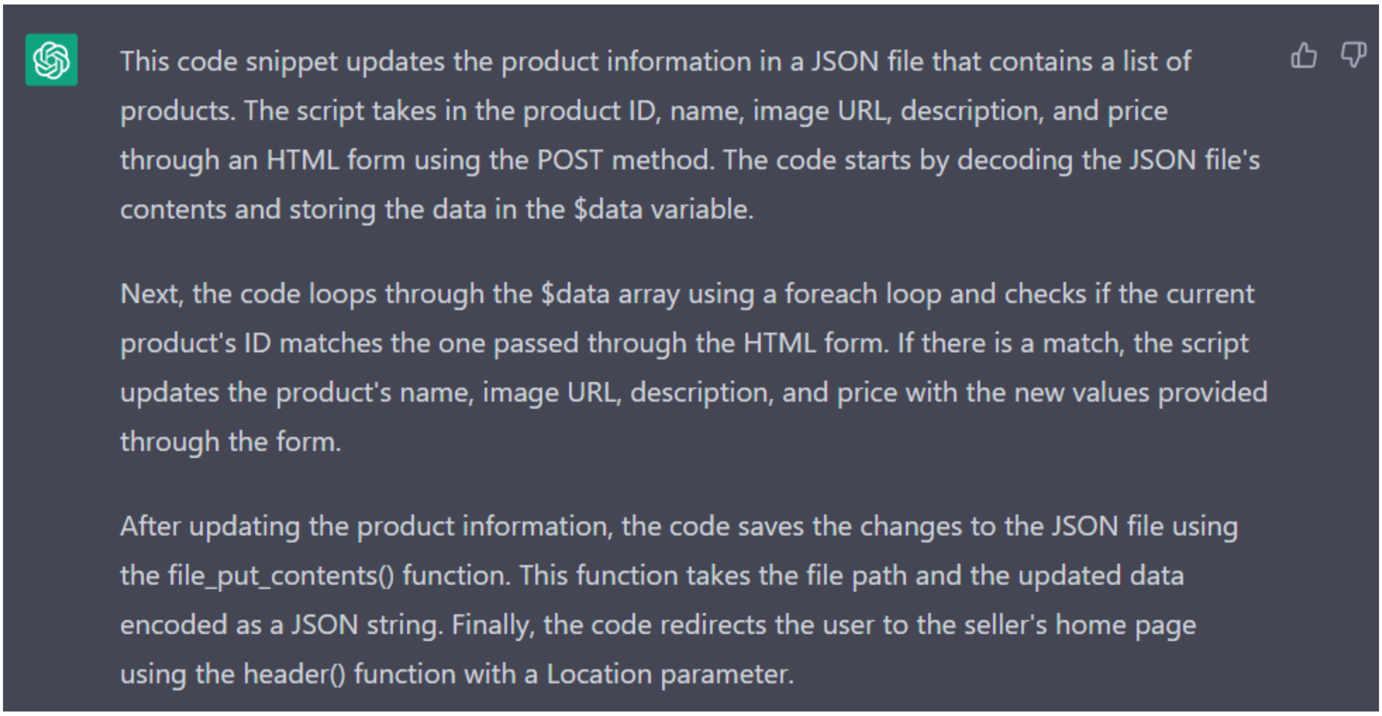
[[ProductUpdateController.php]When Seller edits the product details, this controller handles such requests]
Code;



```php
<?php
//update the product
$productId = $_POST['productId'];
$name = $_POST['name'];
$imgUrl = $_POST['imgUrl'];
$description = $_POST['description'];
$price = $_POST['price'];
//get the data from the json file
$data = json_decode(file_get_contents("./../Models/Products.json"));


//find the product with the id
foreach ($data as $key => $value) {
    if ($value->productID == $productId) {
        $value->name = $name;
        $value->imgUrl = $imgUrl;
        $value->description = $description;
        $value->price = $price;
        break;
    }
}


//save the data to the json file
file_put_contents("./../Models/Products.json", json_encode($data));

//redirect to the seller home page
header('Location: ./../Views/Sellers/SellerHome.php');
```

Explanation:

[1]

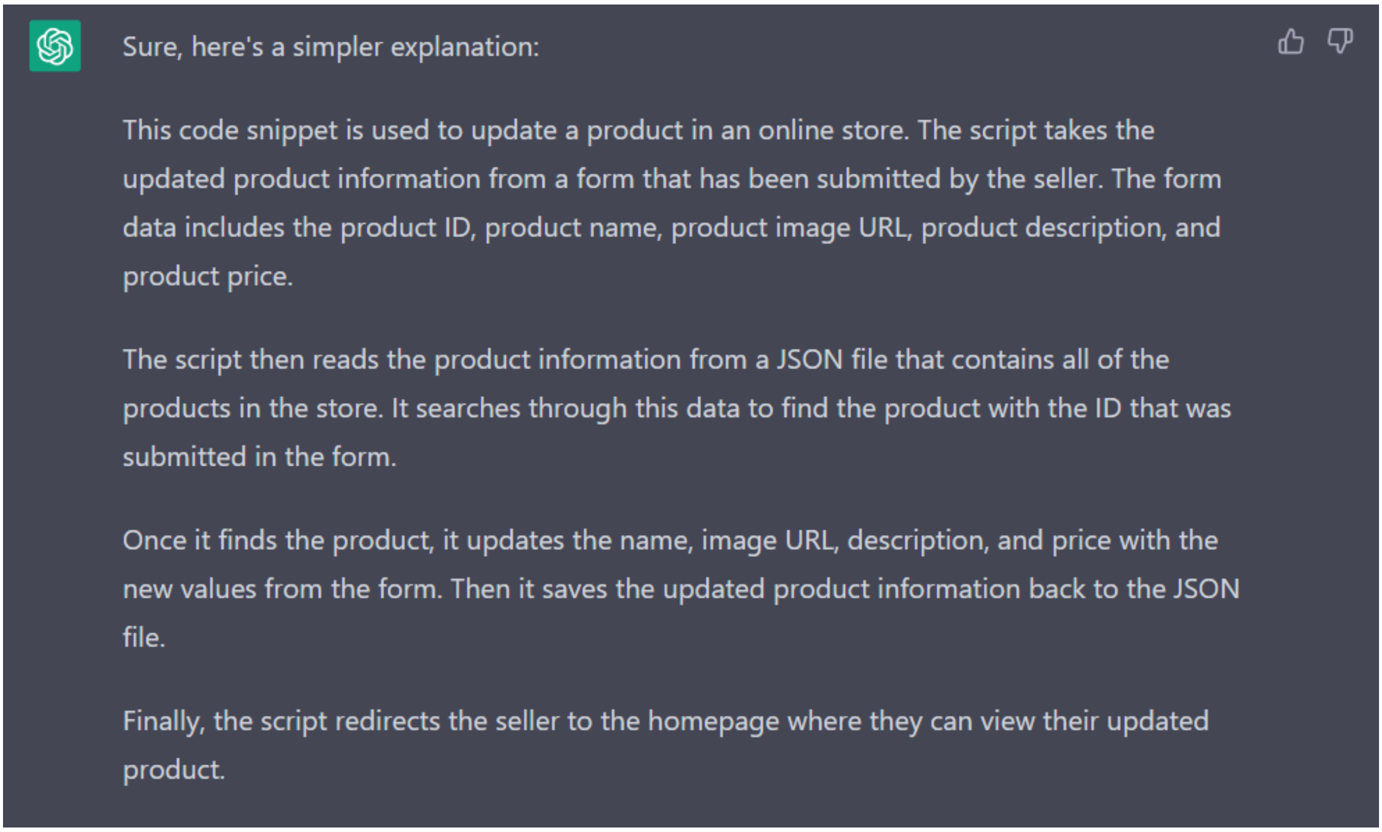


This code snippet updates the product information in a JSON file that contains a list of products. The script takes in the product ID, name, image URL, description, and price through an HTML form using the POST method. The code starts by decoding the JSON file's contents and storing the data in the $data variable.

Next, the code loops through the $data array using a foreach loop and checks if the current product's ID matches the one passed through the HTML form. If there is a match, the script updates the product's name, image URL, description, and price with the new values provided through the form.

After updating the product information, the code saves the changes to the JSON file using the file_put_contents() function. This function takes the file path and the updated data encoded as a JSON string. Finally, the code redirects the user to the seller's home page using the header() function with a Location parameter.

[2]



Sure, here's a simpler explanation:

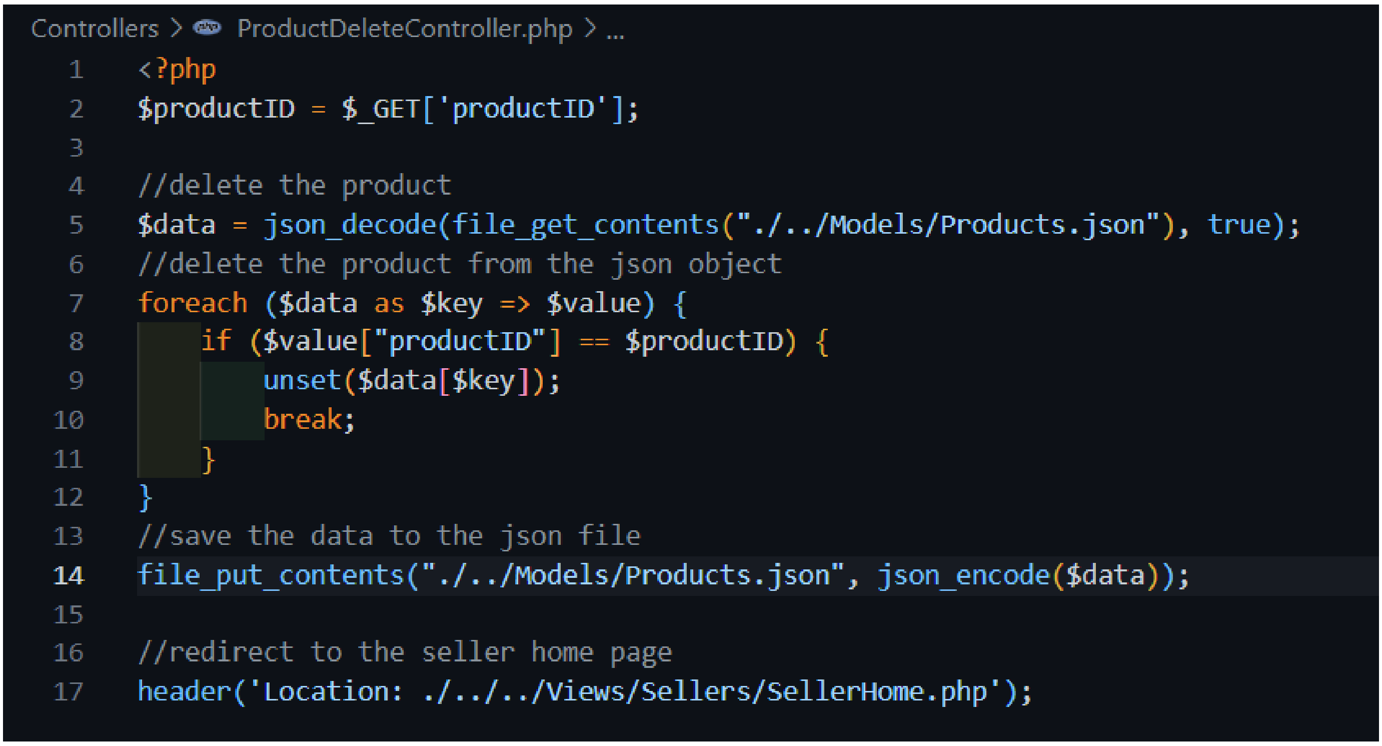
This code snippet is used to update a product in an online store. The script takes the updated product information from a form that has been submitted by the seller. The form data includes the product ID, product name, product image URL, product description, and product price.

The script then reads the product information from a JSON file that contains all of the products in the store. It searches through this data to find the product with the ID that was submitted in the form.

Once it finds the product, it updates the name, image URL, description, and price with the new values from the form. Then it saves the updated product information back to the JSON file.

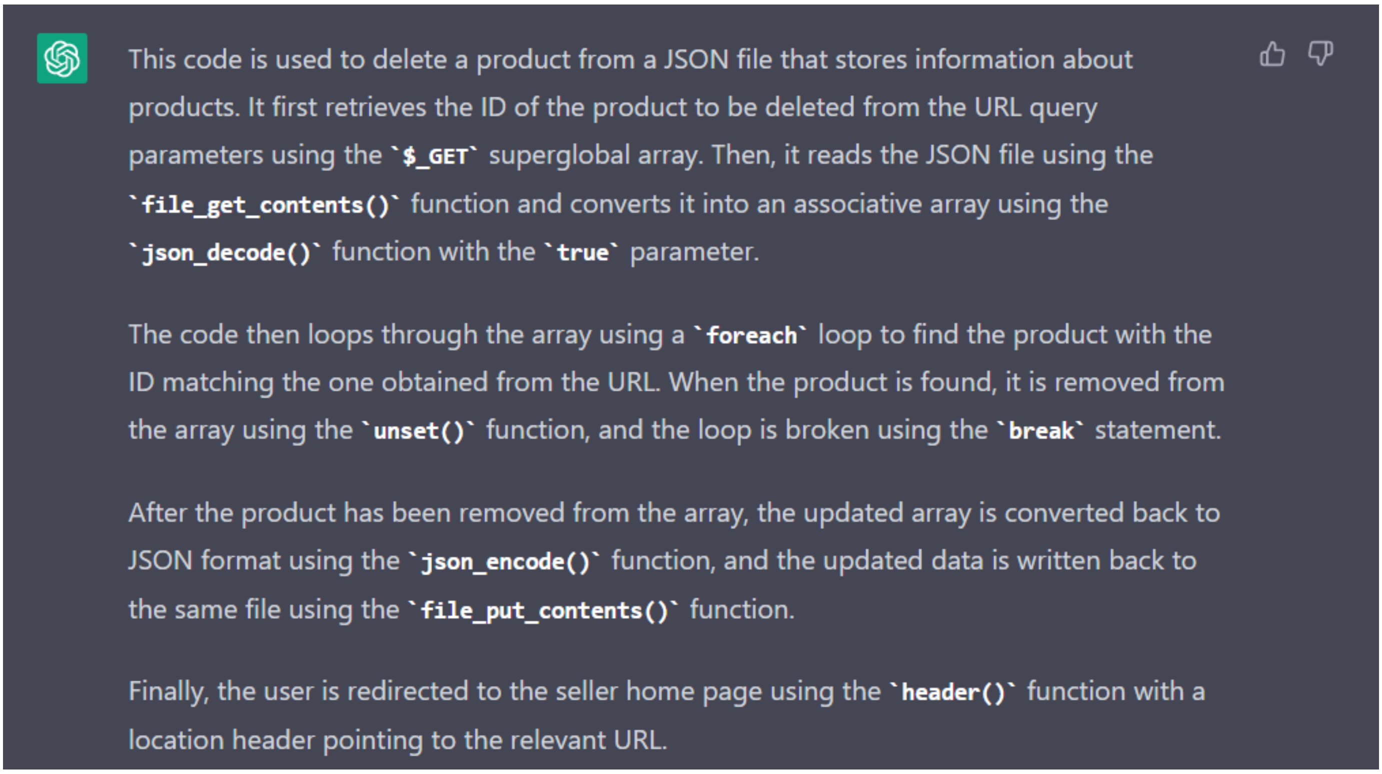
Finally, the script redirects the seller to the homepage where they can view their updated product.

[[ProductDeleteController.php] Where Seller removes a product and Controller handles the request]
Code:



```php
<?php
$productID = $_GET['productID'];

//delete the product
$data = json_decode(file_get_contents("./../Models/Products.json"), true);
//delete the product from the json object
foreach ($data as $key => $value) {
    if ($value["productID"] == $productID) {
        unset($data[$key]);
        break;
    }
}
//save the data to the json file
file_put_contents("./../Models/Products.json", json_encode($data));

//redirect to the seller home page
header('Location: ./../../Views/Sellers/SellerHome.php');
```

Explanation:



This code is used to delete a product from a JSON file that stores information about products. It first retrieves the ID of the product to be deleted from the URL query parameters using the `$_GET` superglobal array. Then, it reads the JSON file using the `file_get_contents()` function and converts it into an associative array using the `json_decode()` function with the `true` parameter.

The code then loops through the array using a `foreach` loop to find the product with the ID matching the one obtained from the URL. When the product is found, it is removed from the array using the `unset()` function, and the loop is broken using the `break` statement.

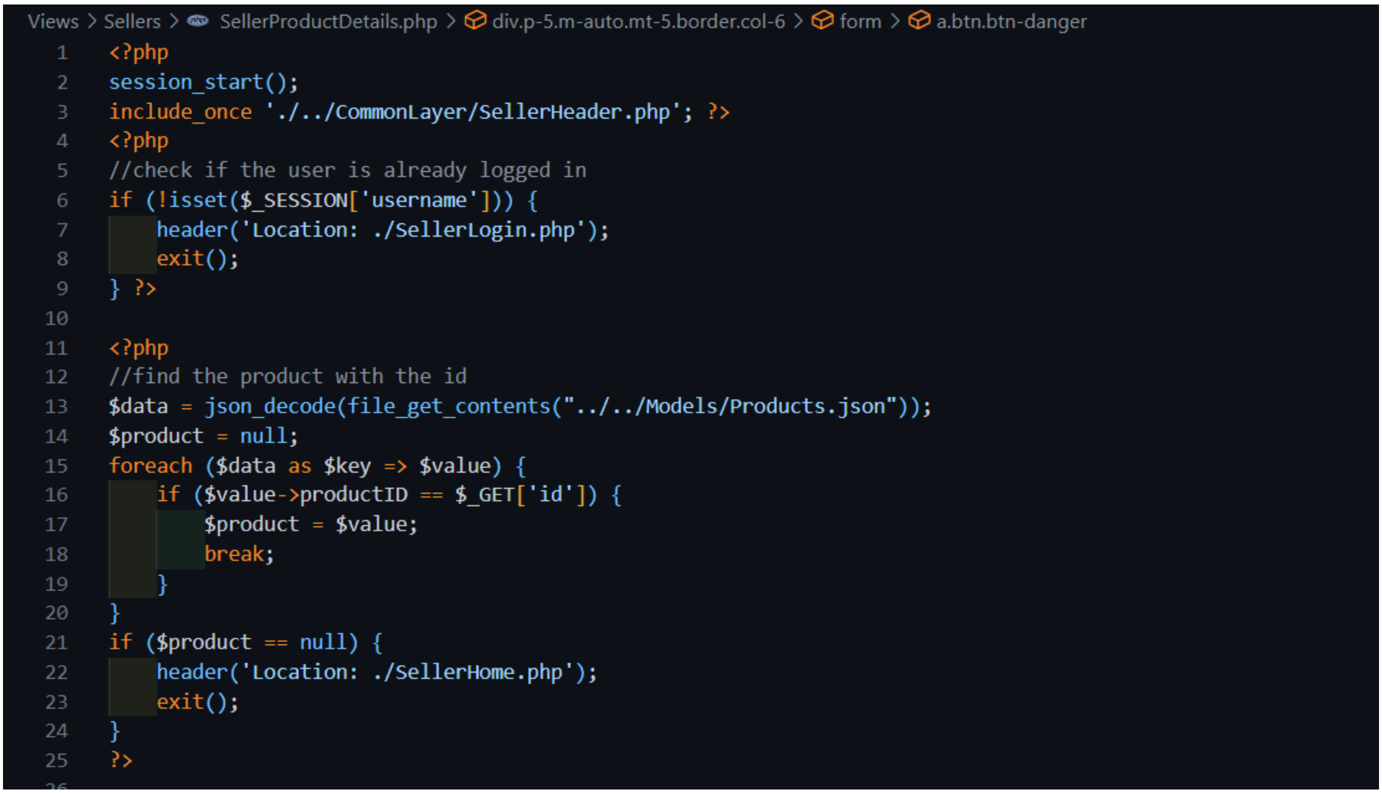
After the product has been removed from the array, the updated array is converted back to JSON format using the `json_encode()` function, and the updated data is written back to the same file using the `file_put_contents()` function.

Finally, the user is redirected to the seller home page using the `header()` function with a location header pointing to the relevant URL.

[[SellerProductDetails.php] View Part Explained]

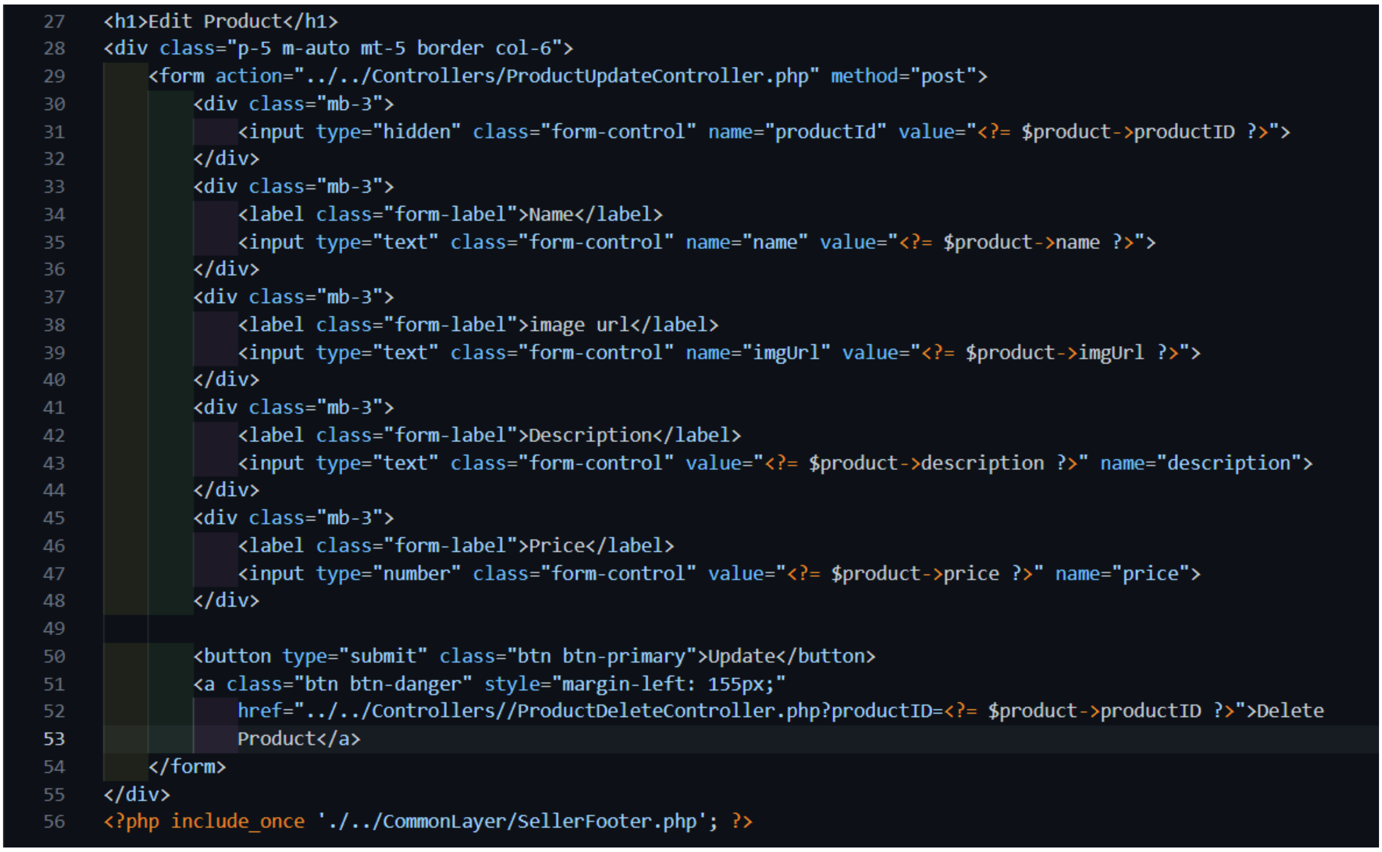
Code:

[1]


Views > Sellers > SellerProductDetails.php > div.p-5.m-auto.mt-5.border.col-6 > form > a.btn.btn-danger

```php
<?php
session_start();
include_once './../CommonLayer/SellerHeader.php'; ?>
<?php
//check if the user is already logged in
if (!isset($_SESSION['username'])) {
    header('Location: ./SellerLogin.php');
    exit();
} ?>

<?php
//find the product with the id
$data = json_decode(file_get_contents("../../Models/Products.json"));
$product = null;
foreach ($data as $key => $value) {
    if ($value->productID == $_GET['id']) {
        $product = $value;
        break;
    }
}
if ($product == null) {
    header('Location: ./SellerHome.php');
    exit();
}
?>
```

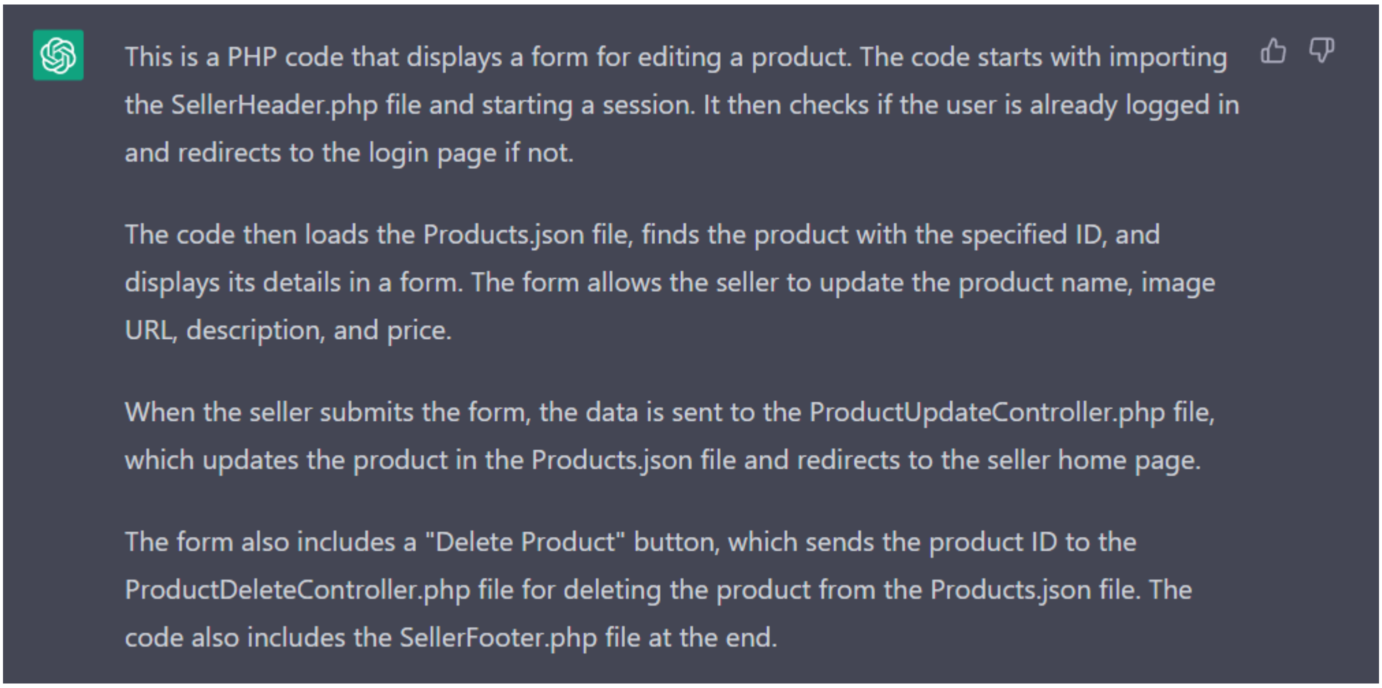
[2]



```
<h1>Edit Product</h1>
<div class="p-5 m-auto mt-5 border col-6">
    <form action="../../Controllers/ProductUpdateController.php" method="post">
        <div class="mb-3">
            <input type="hidden" class="form-control" name="productId" value="<?= $product->productID ?>">
        </div>
        <div class="mb-3">
            <label class="form-label">Name</label>
            <input type="text" class="form-control" name="name" value="<?= $product->name ?>">
        </div>
        <div class="mb-3">
            <label class="form-label">image url</label>
            <input type="text" class="form-control" name="imgUrl" value="<?= $product->imgUrl ?>">
        </div>
        <div class="mb-3">
            <label class="form-label">Description</label>
            <input type="text" class="form-control" value="<?= $product->description ?>" name="description">
        </div>
        <div class="mb-3">
            <label class="form-label">Price</label>
            <input type="number" class="form-control" value="<?= $product->price ?>" name="price">
        </div>

        <button type="submit" class="btn btn-primary">Update</button>
        <a class="btn btn-danger" style="margin-left: 155px;"
            href="../../Controllers//ProductDeleteController.php?productID=<?= $product->productID ?>">Delete
            Product</a>
    </form>
</div>
<?php include_once './../CommonLayer/SellerFooter.php'; ?>
```

Explanation:

[1]



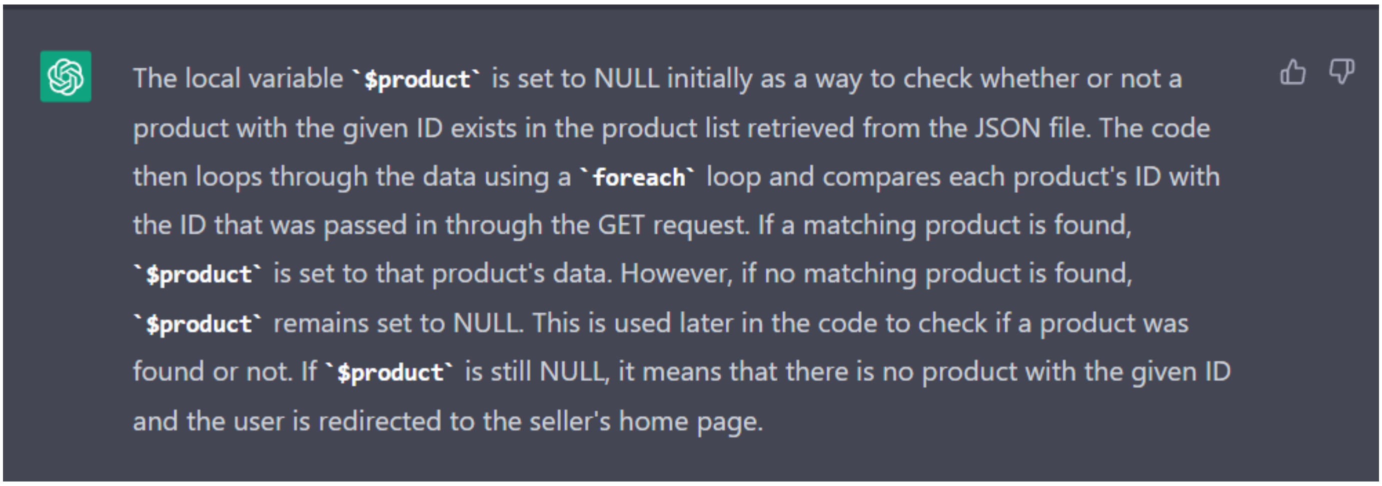
This is a PHP code that displays a form for editing a product. The code starts with importing the SellerHeader.php file and starting a session. It then checks if the user is already logged in and redirects to the login page if not.

The code then loads the Products.json file, finds the product with the specified ID, and displays its details in a form. The form allows the seller to update the product name, image URL, description, and price.

When the seller submits the form, the data is sent to the ProductUpdateController.php file, which updates the product in the Products.json file and redirects to the seller home page.

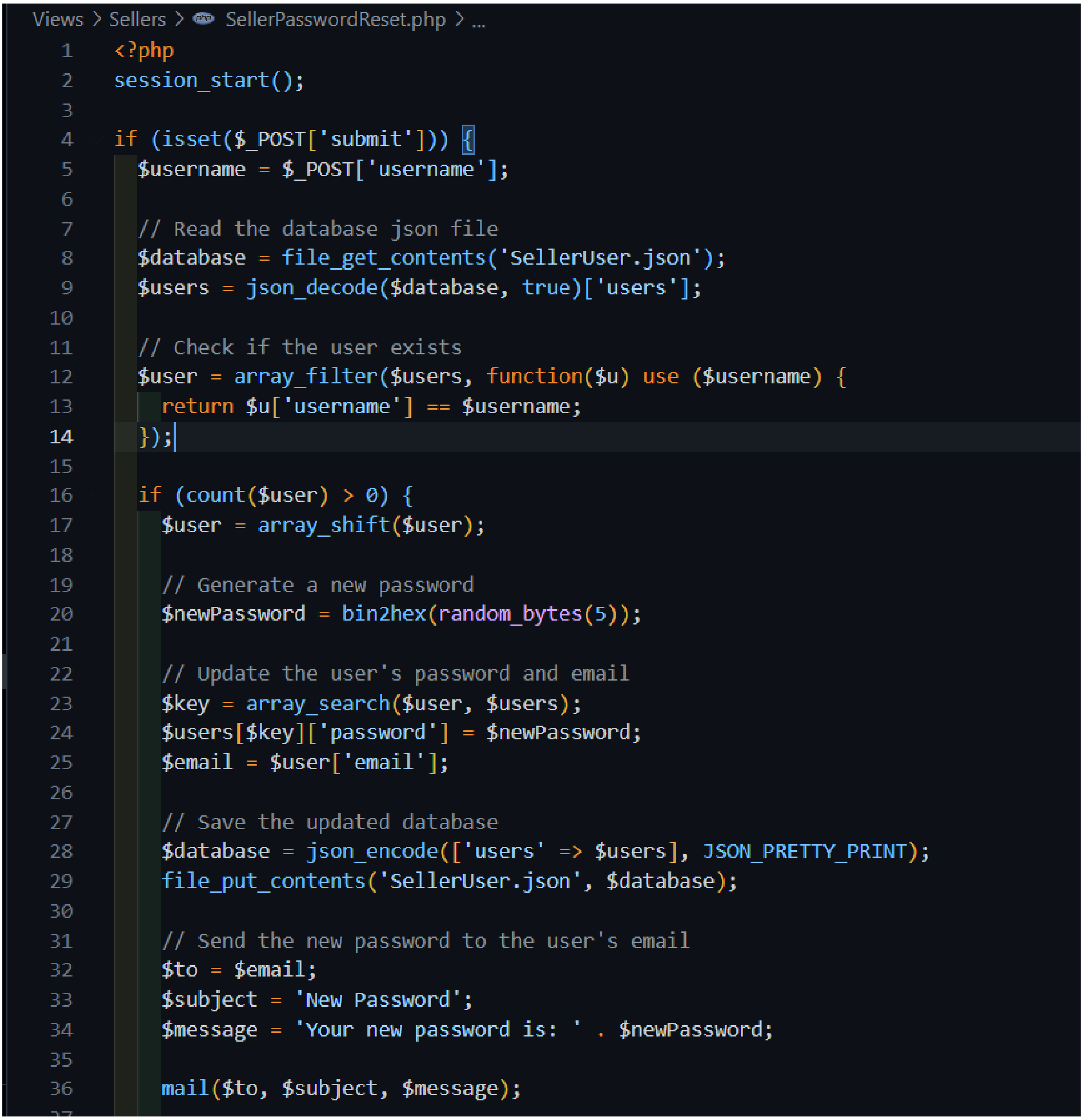
The form also includes a "Delete Product" button, which sends the product ID to the ProductDeleteController.php file for deleting the product from the Products.json file. The code also includes the SellerFooter.php file at the end.

[2]



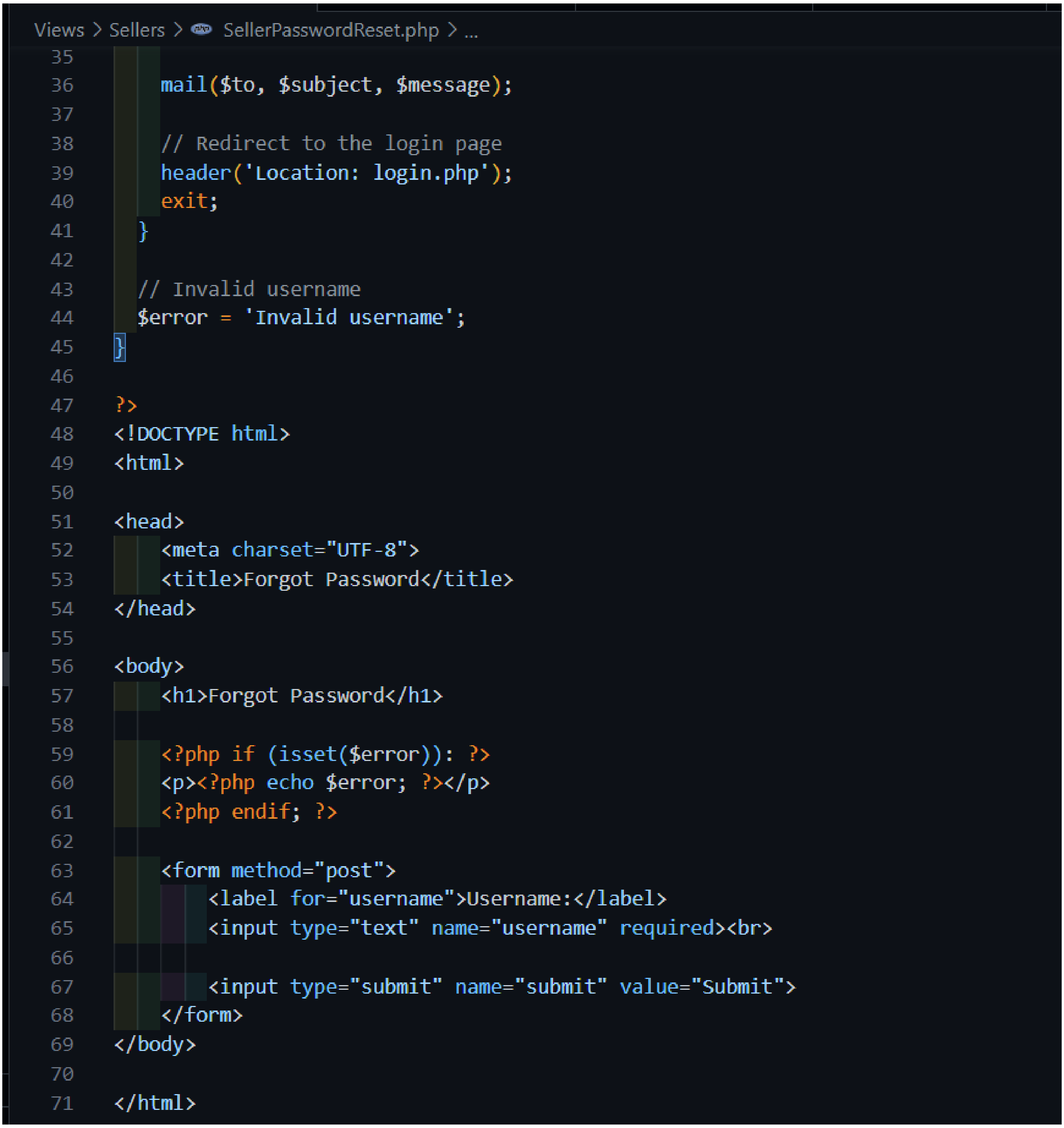
The local variable `$product` is set to NULL initially as a way to check whether or not a product with the given ID exists in the product list retrieved from the JSON file. The code then loops through the data using a `foreach` loop and compares each product's ID with the ID that was passed in through the GET request. If a matching product is found, `$product` is set to that product's data. However, if no matching product is found, `$product` remains set to NULL. This is used later in the code to check if a product was found or not. If `$product` is still NULL, it means that there is no product with the given ID and the user is redirected to the seller's home page.

[[SellerPasswordReset.php] will be explained here]
Code:
[1]



```php
<?php
session_start();

if (isset($_POST['submit'])) {
    $username = $_POST['username'];

    // Read the database json file
    $database = file_get_contents('SellerUser.json');
    $users = json_decode($database, true)['users'];

    // Check if the user exists
    $user = array_filter($users, function($u) use ($username) {
        return $u['username'] == $username;
    });

    if (count($user) > 0) {
        $user = array_shift($user);

        // Generate a new password
        $newPassword = bin2hex(random_bytes(5));

        // Update the user's password and email
        $key = array_search($user, $users);
        $users[$key]['password'] = $newPassword;
        $email = $user['email'];

        // Save the updated database
        $database = json_encode(['users' => $users], JSON_PRETTY_PRINT);
        file_put_contents('SellerUser.json', $database);

        // Send the new password to the user's email
        $to = $email;
        $subject = 'New Password';
        $message = 'Your new password is: ' . $newPassword;

        mail($to, $subject, $message);
```

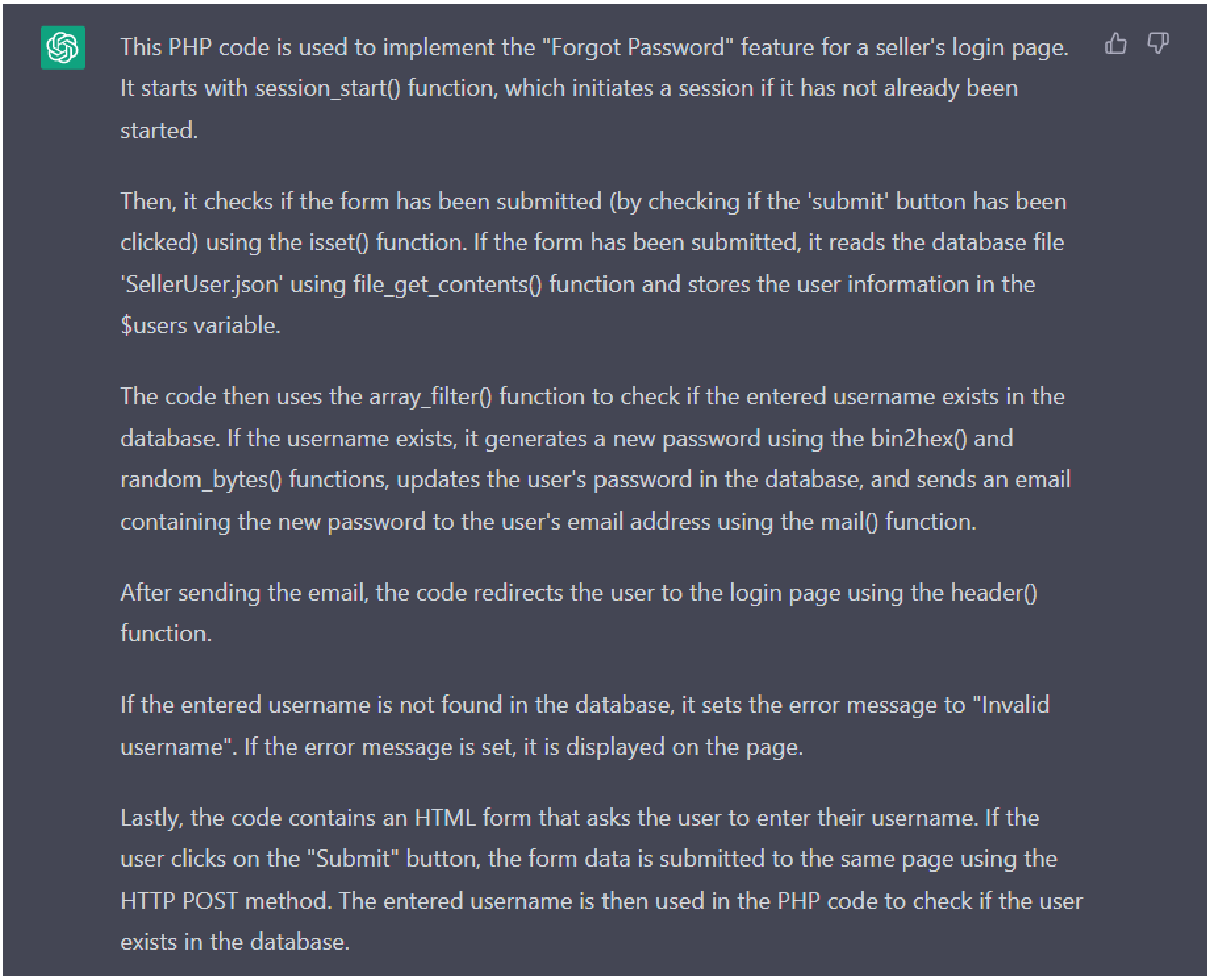
[2]



```

    mail($to, $subject, $message);

    // Redirect to the login page
    header('Location: login.php');
    exit;
  }

  // Invalid username
  $error = 'Invalid username';
}

?>
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>Forgot Password</title>
</head>

<body>
    <h1>Forgot Password</h1>

    <?php if (isset($error)): ?>
    <p><?php echo $error; ?></p>
    <?php endif; ?>

    <form method="post">
        <label for="username">Username:</label>
        <input type="text" name="username" required><br>

        <input type="submit" name="submit" value="Submit">
    </form>
</body>

</html>
```

Explanation:

[1]



This PHP code is used to implement the "Forgot Password" feature for a seller's login page. It starts with session_start() function, which initiates a session if it has not already been started.

Then, it checks if the form has been submitted (by checking if the 'submit' button has been clicked) using the isset() function. If the form has been submitted, it reads the database file 'SellerUser.json' using file_get_contents() function and stores the user information in the $users variable.

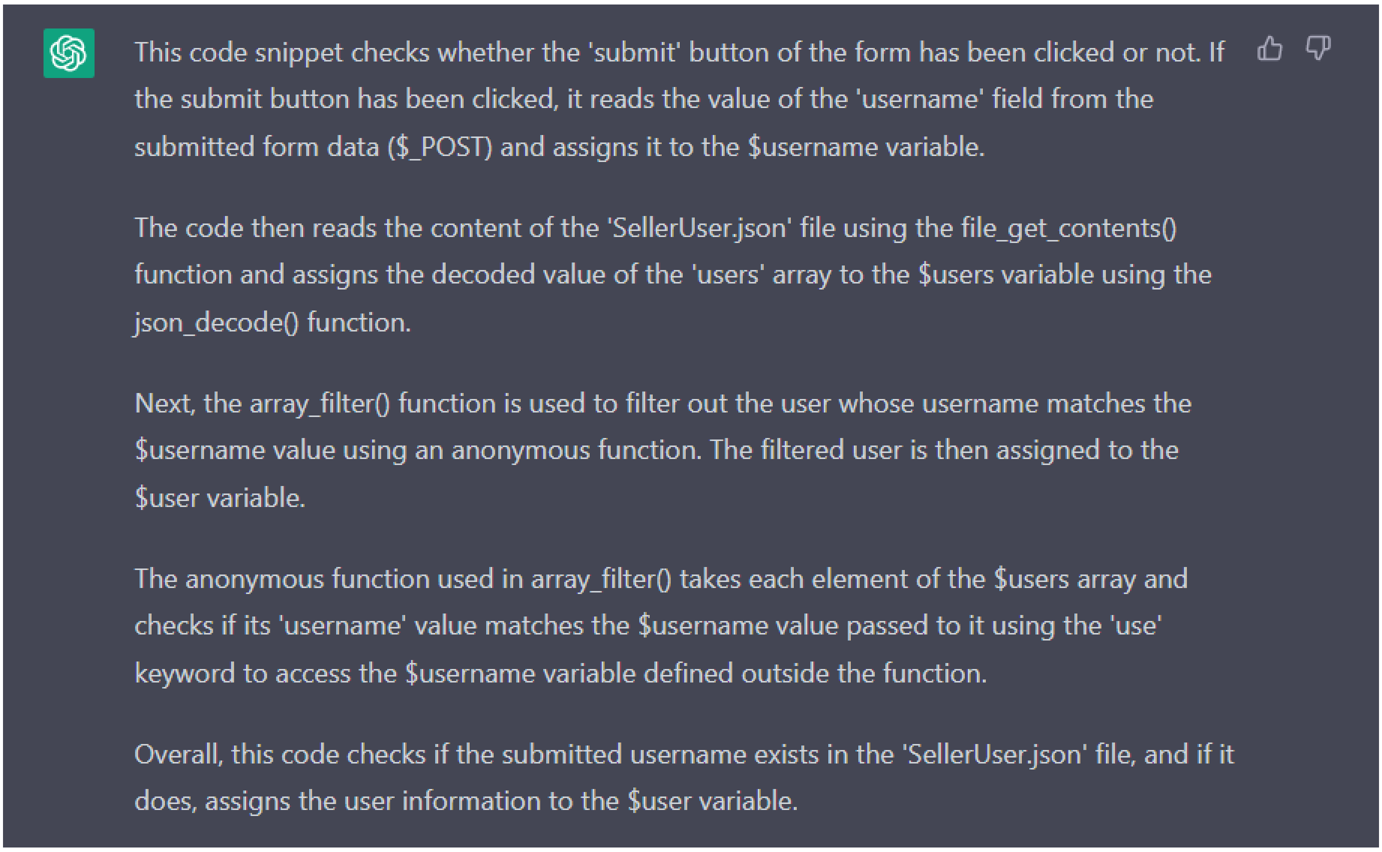
The code then uses the array_filter() function to check if the entered username exists in the database. If the username exists, it generates a new password using the bin2hex() and random_bytes() functions, updates the user's password in the database, and sends an email containing the new password to the user's email address using the mail() function.

After sending the email, the code redirects the user to the login page using the header() function.

If the entered username is not found in the database, it sets the error message to "Invalid username". If the error message is set, it is displayed on the page.

Lastly, the code contains an HTML form that asks the user to enter their username. If the user clicks on the "Submit" button, the form data is submitted to the same page using the HTTP POST method. The entered username is then used in the PHP code to check if the user exists in the database.

[2]



This code snippet checks whether the 'submit' button of the form has been clicked or not. If the submit button has been clicked, it reads the value of the 'username' field from the submitted form data ($_POST) and assigns it to the $username variable.

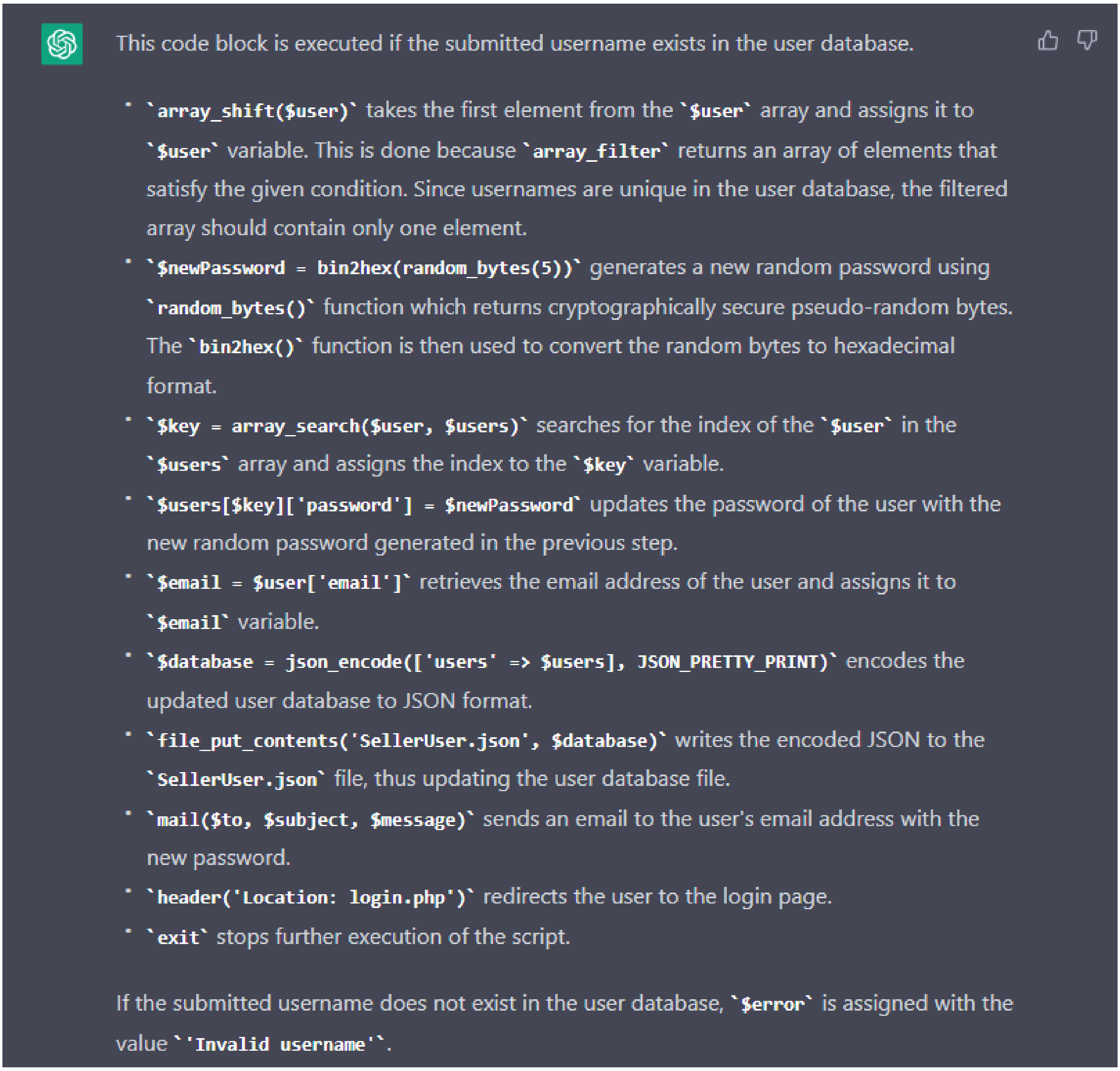
The code then reads the content of the 'SellerUser.json' file using the file_get_contents() function and assigns the decoded value of the 'users' array to the $users variable using the json_decode() function.

Next, the array_filter() function is used to filter out the user whose username matches the $username value using an anonymous function. The filtered user is then assigned to the $user variable.

The anonymous function used in array_filter() takes each element of the $users array and checks if its 'username' value matches the $username value passed to it using the 'use' keyword to access the $username variable defined outside the function.

Overall, this code checks if the submitted username exists in the 'SellerUser.json' file, and if it does, assigns the user information to the $user variable.

[3]



This code block is executed if the submitted username exists in the user database.

- `` `array_shift($user)` `` takes the first element from the `` `$user` `` array and assigns it to `` `$user` `` variable. This is done because `` `array_filter` `` returns an array of elements that satisfy the given condition. Since usernames are unique in the user database, the filtered array should contain only one element.
- `` `$newPassword = bin2hex(random_bytes(5))` `` generates a new random password using `` `random_bytes()` `` function which returns cryptographically secure pseudo-random bytes. The `` `bin2hex()` `` function is then used to convert the random bytes to hexadecimal format.
- `` `$key = array_search($user, $users)` `` searches for the index of the `` `$user` `` in the `` `$users` `` array and assigns the index to the `` `$key` `` variable.
- `` `$users[$key]['password'] = $newPassword` `` updates the password of the user with the new random password generated in the previous step.
- `` `$email = $user['email']` `` retrieves the email address of the user and assigns it to `` `$email` `` variable.
- `` `$database = json_encode(['users' => $users], JSON_PRETTY_PRINT)` `` encodes the updated user database to JSON format.
- `` `file_put_contents('SellerUser.json', $database)` `` writes the encoded JSON to the `` `SellerUser.json` `` file, thus updating the user database file.
- `` `mail($to, $subject, $message)` `` sends an email to the user's email address with the new password.
- `` `header('Location: login.php')` `` redirects the user to the login page.
- `` `exit` `` stops further execution of the script.

If the submitted username does not exist in the user database, `` `$error` `` is assigned with the value `` `'Invalid username'` ``.