Assignment

Experiment Name: **Final Assignment**

SUBMITTED BY

Name	ID	Section
Kabid Yeiad	202-15-14440	57_A

SUBMITTED TO

Rahmatul Kabir Rasel Sarker,

Lecturer

Dept. of CSE

Daffodil International University

Submitted on November 17, 2023

GET and POST methods

1. **GET Method**
   - The GET method is used to request data from a specified resource.
   - Data is appended to the URL in the form of parameters.
   - GET requests can be bookmarked and cached as they are visible in the URL.

Example

```
<form action="process.php" method="get">
    <label for="name">Name:</label>
    <input type="text" name="name" id="name">
    <input type="submit" value="Submit">
</form>
```

In the above example, when the form is submitted, the data will be sent to the server as a query string in the URL, like this: `process.php?name=John`.

PHP Handling:

```
<?php
$name = $_GET['name'];
echo "Hello, $name!";
?>
```

2. **POST Method**
   - The POST method is used to send data to be processed to a specified resource.
   - Data is sent in the HTTP request body, not in the URL, making it more secure for sensitive information.
   - POST requests are not bookmarked or cached.

Example

```
<!-- HTML Form using POST method -->
<form action="process.php" method="post">
    <label for="email">Email:</label>
    <input type="email" name="email" id="email">
    <input type="submit" value="Submit">
```

```
</form>
```

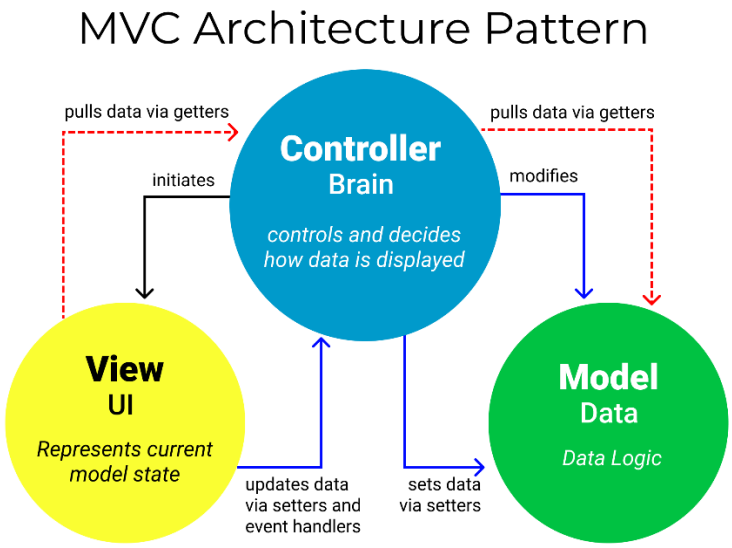
In this example, the form data will be sent to the server in the request body, not visible in the URL.

PHP Handling:

```
<?php
$email = $_POST['email'];
echo "Thank you for submitting your email: $email";
?>
```

Importance of the MVC model

The Model-View-Controller (MVC) architectural paradigm is essential to web development and other applications. It divides an application into three interrelated, purpose-specific components. The Model handles database interactions and application functionality by encapsulating data and business logic. The View renders HTML, CSS, and client-side scripting to display this data to users. The Controller handles user input, requests, and Model updates. This tripartite divide improves code modularity, maintainability, and scalability by separating issues. The MVC pattern's organised approach lets developers collaborate, adapt to needs, and construct powerful, flexible applications. Its broad use in Ruby on Rails, Laravel, and Django shows its efficiency in organising and optimising online applications.



The Model-View-Controller (MVC) architectural pattern provides software development with a number of vital advantages. The separation of concerns, which is its guiding principle, guarantees that the codebase is modular and straightforward to maintain. A clear and organised code structure is the result of the distinct responsibilities of the Model (which handles data and business logic), the View (which manages the user interface and data presentation), and the Controller (which orchestrates the flow of data between the Model and View). The aforementioned division facilitates increased code reusability, as every element can be operated autonomously. Furthermore, MVC facilitates the scaling process by allowing for the concurrent development of various components, thereby promoting scalability. Additionally, maintenance is simplified by the architecture's separation of components, which facilitates the implementation of updates or modifications without impacting other sections of the application. The testability of the model is enhanced, enabling programmers to compose autonomous unit tests for every component. A standardised framework enhances collaboration by simplifying the process for developers to comprehend and participate in initiatives. One notable strength is the capacity to acclimatise to change, as adjustments to one element can be implemented without causing disruptions to others. Furthermore, MVC facilitates the integration of multiple views that present identical data, allowing for smooth interactions between various interfaces (e.g., mobile and web) and a shared underlying data model.

Real World benefits:

Scalability: Scalability is enhanced through the concurrent work of multiple development teams on the Model, View, and Controller components. As an illustration, a team might concentrate on

enhancing the user interface (View), whereas another team might optimise the data processing logic on the backend (Model).

Maintainability: Modifications to a single element, such as the revision of the product catalogue (Model), can be executed independently of any impact on other system components. The codebase becomes more maintainable as the application undergoes further development due to this separation.

Testability: It is possible to independently evaluate each component. Unit tests may be developed to evaluate the data processing logic of the Model, the rendering of the user interface in the View, and the management of user input by the Controller.

Adaptability: Modifications to the pertinent component (Model or View) can be implemented in response to evolving business requirements or the necessity to update the user interface design, all while ensuring that the operation of the application remains uninterrupted.

The MVC design pattern facilitates the incorporation of numerous interfaces. For instance, the web interface and a mobile app interface can be powered by the identical underlying e-commerce application logic, due to the independence of the Model and View components.