Practical-3

Aim: Exploration and Hands-On practice on Django Views. Compare Class-Based Views and Function-Based Views.

Using concept of Django Views display current date and time of the system on a web-page.

As a part of ADF Lab session, a Social Media Application has to be designed by incorporating the concepts of Django. Decide on various modules of the application, user-types and features for the application that is to be designed.

Function Based View (FBV) and Class Based View (CBV)

In Django, both Function-Based Views (FBV) and Class-Based Views (CBV) are used to handle requests and generate responses. However, they differ in terms of implementation and usage. Here's a comparison between Function-Based Views and Class-Based Views in Django:

Function-Based Views (FBV):

1. Implementation: FBVs are implemented as simple Python functions that take a request as input and return an HTTP response.

2. Syntax: FBVs use a straightforward syntax, making them easy to define and understand.

3. Flexibility: FBVs offer more flexibility in terms of defining custom logic and handling complex scenarios.

4. Code Organization: FBVs might result in longer views.py files if there are many views, potentially making the code less organized.

5. Reusability: Code reuse might require manually copying and pasting logic between views.

6. Decorator Usage: FBVs often use decorators like `@login_required` to add functionalities like authentication and permissions.

Advantages over CBV:

Function-Based Views (FBVs) and Class-Based Views (CBVs) are both integral parts of Django, each with its own advantages and use cases. Here are some advantages of Function-Based Views over Class-Based Views:

1. Simplicity and Readability: FBVs are simpler and more straightforward in terms of syntax. They are easier to understand for developers who are new to Django or MVC frameworks in general.

2. Less Boilerplate Code: FBVs usually require less code compared to CBVs, as they do not involve class definitions, mixins, and method overrides.

3. Flexibility: FBVs provide more flexibility in defining custom logic. They allow for a direct and concise approach when handling specific cases or when your view logic is not closely aligned with the built-in behaviors of CBVs.

4. Familiarity: Developers with experience in traditional web development may find FBVs more familiar since they resemble regular functions used in programming.

5. No Inheritance Chain: FBVs do not involve inheritance, which can be advantageous in cases where inheritance might lead to complexity or confusion.

6. Simple Authentication and Authorization: For basic cases, adding decorators like `@login_required` or `@permission_required` to FBVs for authentication and authorization is straightforward.

7. Quick Prototyping: When creating a quick prototype or a minimal feature, FBVs can be a quicker choice due to their simplicity.

8. Minimal Learning Curve: Learning FBVs is generally quicker for developers who are new to Django or for simple use cases.

Class-Based Views (CBV):

1. Implementation: CBVs are implemented as classes, allowing the use of inheritance and mixins to achieve code reuse and modularity.

2. Syntax: CBVs can be more complex due to the use of class structures, but they provide a clear separation of concerns.

3. Built-in Functionality: CBVs come with built-in methods that handle common tasks like HTTP methods (GET, POST), context data, template rendering, etc.

4. Code Organization: CBVs encourage better code organization by separating different concerns into class methods.

5. Code Reusability: CBVs encourage code reusability through inheritance and mixin classes. Common behaviors can be defined in parent classes and specialized views can extend them.

6. URL Configuration: CBVs require less URL configuration, as some common functionalities are handled by default in the class-based view.

Advantage of CBV:

Class-Based Views (CBVs) offer several advantages over Function-Based Views (FBVs) in Django. Here are some of the key advantages of using CBVs:

1. Code Reusability: CBVs allow you to create base view classes with common functionalities. This enables you to reuse code across multiple views by inheriting from these base classes. With FBVs, you might need to copy and paste similar code between different views.

2. Modularity: CBVs promote a more modular code structure. Different aspects of the view, such as HTTP methods, context data, and template rendering, can be organized into separate class methods, making the code easier to understand and maintain.

3. Built-in Functionality: CBVs come with built-in methods that handle common tasks. For example, Django's generic class-based views provide methods like `get()`, `post()`, and `form_valid()` that handle HTTP methods and form validation. This reduces the amount of boilerplate code you need to write.

4. Mixins: CBVs support mixins, which are reusable components that can be mixed into view classes. This allows you to add specific functionalities to views without having to rewrite the entire view code.

5. Inheritance: In CBVs, you can easily create variations of a view by subclassing and overriding specific methods. This simplifies the process of creating similar views with slight differences.

6. Consistency: CBVs provide a consistent class-based structure that can be beneficial for larger projects where a team of developers is working on the codebase. It's easier for team members to understand and contribute to the code when the structure is standardized.

7. URL Configuration: CBVs require less URL configuration compared to FBVs, as some common functionalities are handled by default in the class-based view. This can lead to a cleaner and more concise URL configuration.

8. Better Integration: CBVs integrate well with other Django features like decorators, authentication, and permissions. For example, you can easily apply a decorator to a class-based view by using the `@method_decorator` decorator.

9. Built-in Pagination: Some CBVs, such as ListView, come with built-in pagination support, making it easier to implement paginated views.

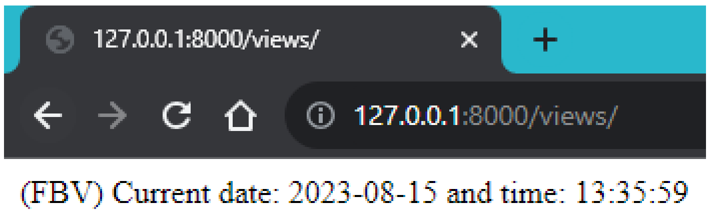
While CBVs offer these advantages, it's important to note that they might have a steeper learning curve, especially for developers who are new to object-oriented programming and class-based structures. Additionally, in cases where the logic of a view is simple and straightforward, FBVs might still be a suitable choice due to their simplicity. The decision between CBVs and FBVs depends on the specific requirements and complexity of your project.

When to Use Which:

1. Function-Based Views: FBVs are suitable for simple views with straightforward logic. If you want more control over each aspect of the view and prefer a simpler implementation, FBVs might be a better choice.

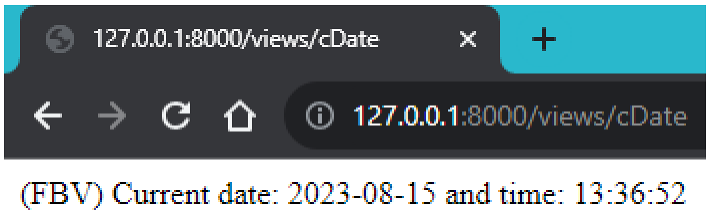
2. Class-Based Views: CBVs are well-suited for complex views and cases where you want to reuse common functionality across multiple views. They promote code modularity and organization.

Function based views output:



```
(FBV) Current date: 2023-08-15 and time: 13:35:59
```

Class based view output:



```
(FBV) Current date: 2023-08-15 and time: 13:36:52
```

<u>Modules of the application, user-types and features for the social media application</u>

Modules:

1. Authentication and Authorization:

   - User Registration

   - Login/Logout

- Password Reset

2. Profile Management:
   - Edit Profile
   - Profile Picture
3. News Feed:
   - Display Posts from Friends
   - Like, Comment, Share Posts
4. User Posts:
   - Text Posts
   - Photo/Video Posts
5. Friends and Followers:
   - Send Friend Requests
   - Accept/Reject Friend Requests
   - Follow/Unfollow Users
6. Messaging:
   - Private Messaging
   - Group Chats
   - Message Encryption
7. Search and Discovery:
   - Search for Users/Content

User Types:

1. Regular Users(logged in):

- Can create posts, like, comment, and interact with friends' posts.
- Can connect with friends and follow other users.

2. Administrator:
3. Regular user(Non Loggedin user)

Features:

1. Personalized Feed:
   - Algorithmically-generated feed based on user activity and interests.

2. Media Sharing:
   - Ability to share photos, videos, and GIFs in posts and comments.

3. Hashtags and Trends:
   - Follow trending topics and use hashtags to increase post visibility.

4. Location Sharing:
   - Tagging posts and check-ins to specific locations.

5. Privacy Controls:
   - Fine-grained control over who can see posts and personal information.

6. Analytics for Pages:
   - Page admins can access insights about their page's performance.