1. Scenario: A Ugandan IT company wants to build a robust e-commerce platform for local farmers to sell their products online.

Question: Describe how the three layers (Presentation, Business Logic, and Data) of an ASP.NET MVC application can be used in this scenario.

Answer:

In an e-commerce platform, the three layers of ASP.NET MVC would be utilized as follows:

- **Presentation Layer (Views):** This layer would handle the user interface (UI) elements. It would display product catalogs, shopping carts, order forms, and other interactive components. The views would be responsible for capturing user input and displaying information to the user.
- **Business Logic Layer (Controllers):** This layer would act as the intermediary between the presentation and data layers. It would handle user requests, process data, and coordinate interactions with the data layer. For example, controllers would handle adding products to the cart, processing orders, and managing user accounts.
- **Data Layer (Models):** This layer would represent the data structure of the application. It would define classes for products, orders, users, and other relevant entities. The data layer would interact with the database (e.g., SQL Server) to store and retrieve data.

2. Scenario: A small Ugandan business lacks sufficient funds to maintain a private server for their web application.

Question: Explain how deploying the application on Microsoft Azure can solve their problem.

Answer:

Deploying the application on Microsoft Azure can provide several benefits to the small Ugandan business:

- **Cost-effectiveness:** Azure offers a pay-as-you-go pricing model, allowing the business to only pay for the resources they consume. This can significantly reduce infrastructure costs compared to maintaining a private server.
- **Scalability:** Azure provides scalable resources, allowing the business to easily adjust their computing power based on demand. This is crucial for handling traffic spikes during peak seasons or promotions.
- **Reliability:** Azure's infrastructure is highly reliable and redundant, ensuring that the application remains available even in case of hardware failures.
- **Security:** Azure provides robust security features, such as firewalls, intrusion detection, and data encryption, to protect the application and data from cyber threats.

3. Scenario: A Ugandan university requires a web-based student management system.

Question: Explain how a controller can validate and process student registration requests in ASP.NET MVC.

Answer:

In ASP.NET MVC, a controller can validate and process student registration requests through the following steps:

1. **Create an action method:** This method would handle the HTTP POST request from the registration form.
2. **Retrieve form data:** The controller would extract the form data, such as student name, email, and password.
3. **Validate data:** The controller would validate the input data using built-in validation attributes or custom validation logic. This ensures that the data is in the correct format and meets the required criteria.
4. **Process data:** If the validation is successful, the controller would process the data, such as creating a new user account in the database.
5. **Redirect or display a view:** After successful processing, the controller would redirect the user to a confirmation page or display a success message.

4. Scenario: A national NGO in Uganda plans to track donations via a web application.

Question: How would you use action methods in a controller to handle incoming donation data and present summary reports?

Answer:

Action methods in the controller can be used to:

- **Handle donation submissions:** Create an action method to receive donation data from the form, validate it, and store it in the database.
- **Generate summary reports:** Create additional action methods to retrieve donation data from the database, generate summary reports (e.g., total donations, donations by category), and display them to authorized users.
- **Filter and sort data:** Implement filtering and sorting options in the action methods to allow users to customize the reports based on their needs.
- **Present data:** Use appropriate view models to pass data to the views, which would then display the reports in a user-friendly format.

5. Scenario: A Ugandan startup wants a website that dynamically shows real-time stock levels.

Question: Discuss how Razor view syntax and partial views can be used to display live updates without refreshing the page.

Answer:

Razor view syntax and partial views can be used to implement real-time updates without page refreshes through techniques like:

- **JavaScript and AJAX:** Use JavaScript to make AJAX calls to the server to fetch updated stock levels.
- **Razor syntax:** Use Razor syntax to dynamically render the updated stock levels in the view.
- **Partial views:** Create partial views for the stock level components, making it easier to update them without affecting the entire page.

6. Scenario: A Ugandan government health portal aims to improve accessibility for visually impaired users.

Question: How can you ensure that views rendered by Razor are accessible and follow best practices?

Answer:

To ensure accessibility of Razor views, consider the following practices:

- **Semantic HTML:** Use appropriate HTML elements with semantic meaning to structure the content.
- **ARIA attributes:** Add ARIA attributes to provide additional information to screen readers.
- **Keyboard navigation:** Make sure the website is fully navigable using the keyboard.
- **Color contrast:** Use sufficient color contrast between text and background.
- **Alternative text for images:** Provide descriptive alternative text for images.
- **Testing with assistive technologies:** Test the website with screen readers and other assistive technologies to identify and fix accessibility issues.

7. Scenario: A local Ugandan bank is implementing a customer data system.

Question: Design a model class for customer information, including properties such as ID, Name, and Account Balance.

Answer:

1. Class Name: `Customer`

The model represents a customer and their associated information, like an ID, Name, and Account Balance. The class will serve as a blueprint for creating customer objects.

2. Properties (Attributes)

These are the data fields that store specific pieces of information about the customer. In this case, we have:

- **Customer ID**:
    - **Description**: A unique identifier for each customer (e.g., a number or alphanumeric code).
    - **Type**: Integer or String (depending on the identifier format).
    - **Purpose**: Ensures each customer can be individually referenced.

- **Customer Name**:
    - **Description**: The full name of the customer (e.g., "John Doe").
    - **Type**: String (text).
    - **Purpose**: Identifies the customer by their name.
- **Account Balance**:
    - **Description**: The current balance in the customer's account (e.g., $1000.00).
    - **Type**: Decimal (to handle monetary amounts with precision).
    - **Purpose**: Stores the amount of money available in the customer's account.

3. Methods (Actions)

These are the actions or behaviors the customer object can perform. Each of these actions will manipulate or interact with the customer's properties:

- **Display Customer Information**:
    - **Description**: A method that prints out or shows the customer's details (ID, Name, and Account Balance).
    - **Purpose**: Used to retrieve and view the customer's information in a human-readable format.
- **Deposit Money**:
    - **Description**: A method that adds money to the customer's account balance.
    - **Parameters**: Amount (the amount to be deposited).
    - **Purpose**: Updates the customer's balance by adding the specified amount.
- **Withdraw Money**:
    - **Description**: A method that subtracts money from the customer's account balance.
    - **Parameters**: Amount (the amount to be withdrawn).
    - **Purpose**: Updates the customer's balance by subtracting the specified amount, but only if the balance is sufficient.

4. Relationships

This class, in a broader system, could have relationships with other models such as:

- **Transactions**:
    - A customer might be associated with multiple transactions (deposits, withdrawals, transfers).
- **Accounts**:
    - A customer might have multiple accounts (savings, checking, etc.), each with its own balance.

8. Scenario: A health organization in Uganda is digitizing patient records.

Question: Explain how strongly-typed views can be used to display and edit patient information efficiently.

Answer:

Strongly-typed views can be used to display and edit patient information efficiently by:

- **Intellisense:** The compiler can provide Intellisense for the model properties, making it easier to write code and reduce errors.
- **Type safety:** The compiler can check for type mismatches, preventing runtime errors.
- **Code reusability:** Strongly-typed views can be reused with different model instances, reducing code duplication.
- **Improved readability:** Strongly-typed views are easier to read and maintain because the model type is explicitly defined.

9. Scenario: A Ugandan courier service needs a tracking system where users can search for parcels by their IDs.

Question: Describe how you would configure routing in ASP.NET MVC to enable parcel search functionality.

Answer:

- **Routing** in ASP.NET MVC is the process of mapping incoming URLs to specific controller actions.
- To enable parcel search, you would define a **route** that matches a URL pattern like `/Search/{parcelId}`.
- This route would then direct the request to the `Search` action in the `ParcelController`.
- The `parcelId` in the URL would be passed as a parameter to the `Search` action.

10. Scenario: A travel company in Uganda wants users to access their tour details through custom URLs like /tours/kampala.

Question: Explain how custom routing rules can be defined in ASP.NET MVC to support this feature.

Answer:

- You can define a **custom route** that matches a URL pattern like `/tours/{city}`.
- This route would then direct the request to the `Details` action in the `TourController`.
- The `city` in the URL would be passed as a parameter to the `Details` action.
- This allows users to access tour details using more user-friendly URLs like `/tours/kampala` instead of default URLs like `/Tour/Details/kampala`.

11. Scenario: A Ugandan job portal wants to allow users to filter job listings by category.

Question: Illustrate how HTML Helper methods like DropDownListFor can be used to create a dynamic filter form.

Answer:

- **HTML Helper methods** in ASP.NET MVC provide a convenient way to generate HTML elements within your views.
- The `DropDownListFor` helper method is used to create a dropdown list.
- You would use this helper to generate a dropdown list of categories on the job portal page.
- Users can then select a category from the dropdown list to filter the job listings.

12. Scenario: An online shopping platform in Uganda requires a login form.

Question: Discuss the role of HTML Helper methods in designing the login form and validating inputs.

Answer:

- HTML Helper methods simplify the creation of form elements like textboxes, password fields, and submit buttons.
- They also help with data binding, which connects the form elements to the properties of your model.
- In addition, they provide built-in validation features, such as checking for required fields or validating email addresses.
- This helps ensure that the user enters valid data before submitting the form.

13. Scenario: A school in Uganda is developing a web app for managing exam schedules.

Question: How can the ASP.NET MVC framework's separation of concerns benefit this project?

Answer:

- ASP.NET MVC follows the **separation of concerns** principle, which means that different parts of the application have distinct responsibilities.
- This leads to better **code organization** and **maintainability**.
- For example, the **model** handles data, the **view** presents the data to the user, and the **controller** handles user interactions.
- This separation makes it easier to modify or update specific parts of the application without affecting other parts.

14. Scenario: A hospital in Uganda needs to display different dashboards for doctors and patients.

Question: Explain how action filters in ASP.NET MVC can help implement role-based dashboards.

Answer:

- **Action filters** are attributes that can be applied to controller actions.
- You can create a custom action filter that checks the user's role (e.g., doctor or patient).
- This filter can then determine which dashboard to display based on the user's role.
- This ensures that only authorized users can access their respective dashboards.

15. Scenario: A startup in Uganda plans to launch a job board website.

Question: Explain how scaffolding can speed up development for creating views like job listings and application forms.

Answer:

- **Scaffolding** is a feature in ASP.NET MVC that automatically generates code for common tasks like creating views, controllers, and actions.
- This can significantly speed up the development process for tasks like creating forms for job listings and applications.
- The generated code can then be customized to fit the specific needs of the project.

16. Scenario: A community health center in Uganda wants to generate CRUD interfaces for patient data.

Question: Discuss the use of the Create and Edit scaffolding templates in implementing such a system.

Answer:

- The **Create** scaffolding template can be used to generate a form for creating new patient records.
- The **Edit** scaffolding template can be used to generate a form for editing existing patient records.
- This can quickly create the basic infrastructure for managing patient data, which can then be customized to meet the specific requirements of the health center.

17. Define and describe controllers

Answer:

- **Controllers** are the heart of ASP.NET MVC applications.
- They handle incoming requests from the client, interact with the model to retrieve or update data, and select the appropriate view to display.
- Controllers are responsible for the overall logic and flow of the application.

18. Describe how to work with action methods

Answer:

- **Action methods** are public methods within a controller that handle specific HTTP requests (e.g., GET, POST).
- They receive input parameters from the request, process the data, and return a response to the client.
- Responses can include views, redirects, or JSON data.

19. Explain how to invoke action methods

Answer:

- Action methods are invoked when a matching URL is requested.
- The routing system in ASP.NET MVC maps incoming URLs to the appropriate controller and action method.

20. Explain routing requests

- **Routing** in ASP.NET MVC is the process of mapping incoming URLs to specific controller actions.
- When a user requests a URL in an ASP.NET MVC application, the routing system analyzes the URL and determines which controller and action method should handle the request.
- This is done by comparing the URL with a set of defined **routes**.
- Routes specify patterns that URLs should match.
- For example, a route might be defined to match URLs like `/products/{id}`, where `{id}` is a placeholder for a product ID.
- If the incoming URL matches this pattern, the request would be routed to the `ProductsController` and its `Details` action method, passing the product ID as a parameter.

21. Describe URL patterns

- **URL patterns** are used in routing to define the structure of URLs that the application should handle.
- They are typically defined using a combination of literal segments and parameters.
- **Literal segments** are fixed parts of the URL, such as `/products` or `/about`.
- **Parameters** are placeholders for variable parts of the URL, enclosed in curly braces `{}`, such as `{id}`, `{category}`, or `{name}`.
- URL patterns help make URLs more user-friendly and meaningful, as they can reflect the structure of the application or the data being accessed.

22. Describe the difference between a website and a web Application.

While the terms "website" and "web application" are often used interchangeably,[1] there are some key distinctions:

- **Website:**
  - Primarily focuses on **static content** like HTML, CSS, and images.
  - Typically used for information dissemination, online presence, and branding.
  - May have limited interactivity or dynamic content.
  - Examples: company websites, personal blogs, online portfolios.
- **Web Application:**
  - Offers **dynamic content** and **user interaction**.
  - Built with technologies like ASP.NET MVC, allowing for complex logic, data processing, and user authentication.
  - Examples: e-commerce platforms, social media sites, online banking systems, project management tools.

In essence:

- A **website** is like a brochure or a digital billboard, primarily for displaying information.
- A **web application** is like a software program that runs in a web browser, enabling users to interact with the system and perform tasks.