Q1 You are designing a Django **blog application** with the following requirements:

- A homepage that lists all blog posts.
- A detail page for each post, accessible via `/post/<int:id>/`.
- A form that allows users to submit new blog posts.
- Templates should use **Django Template Language (DTL)** for rendering.

Task:

1. Define the **models** for storing blog post data.
2. Create the necessary **views** to handle displaying posts and form submissions.
3. Write the **URL patterns** to map these views.
4. Design a **template structure** using template inheritance for consistent layouts.

Q2 Your Django blog application needs a **search feature** that allows users to find blog posts by entering keywords.

Task:

1. Create a **view function** that filters blog posts based on keywords found in their **title** or **content**.
2. Add a **search form** to the homepage, allowing users to enter search terms.
3. Define a **URL pattern** like `/search/?q=keyword` to display search results dynamically.
4. Modify the **template** to show the search results or a message if no posts match the query.

Q3 You are designing a Django **event management application** with the following requirements:

- A homepage that lists all upcoming **events**.
- A detail page for each event, accessible via `/event/<int:id>/`.
- A form that allows users to **create and register for events**.
- Templates should use **Django Template Language (DTL)** for rendering.

Task:

1. Define the **models** for storing event data (e.g., `title`, `description`, `date`, `location`).
2. Create the necessary **views** to handle displaying events and form submissions.
3. Write the **URL patterns** to map these views.
4. Design a **template structure** using template inheritance for consistent layouts.

Q4 You are designing a Django **e-commerce application** with the following requirements:

- A homepage that displays a list of **products**.
- A detail page for each product, accessible via `/product/<int:id>/`.
- A **shopping cart** where users can add and remove products.
- Templates should use **Django Template Language (DTL)** for rendering.

Task:

1. Define the **models** for storing product data (`name`, `price`, `description`, `image`).
2. Create the necessary **views** for listing products, displaying product details, and managing the cart.
3. Write the **URL patterns** to map these views.
4. Design a **template structure** using template inheritance for consistency.

Q5 You are building a Django **job portal** with the following requirements:

- A homepage that lists **job postings**.
- A detail page for each job, accessible via `/job/<int:id>/`.
- A form that allows users to **apply for jobs** by submitting their resume and details.
- Templates should use **Django Template Language (DTL)** for rendering.

Task:

1. Define the **models** for job postings (`title`, `company`, `location`, `description`, `posted_date`).
2. Create the necessary **views** for displaying job listings and handling job applications.
3. Write the **URL patterns** to map these views.
4. Design a **template structure** using template inheritance for consistent layouts.

Q6 You are developing a **Django contact application** where users can submit their queries via a **contact form**. The form should collect:

- **Name**
- **Email**
- **Subject**
- **Message**

After submission, the form should:

- Validate user input (e.g., required fields, valid email).
- Save the data to the database.
- Display a success message upon submission.

Task:

1. Create a **Django Form class** to handle user input.
2. Define a **view** to display the form and process submissions.
3. Write the **URL pattern** for the contact form page.
4. Design a **template** to render the form and show validation errors.
5. Implement a **success page** or redirect after successful submission.

Q7 You are developing a **Django authentication system** where users can **log in and log out** securely. The system should:

- Provide a **login form** where users enter their **username** and **password**.
- Authenticate users and create a **session** upon successful login.
- Store user-specific preferences using **cookies** (e.g., theme selection, last login time).
- Allow users to **log out**, which should end the session.
- Use **template inheritance** to maintain a consistent UI across authentication pages.

Task:

1. Create a **Django Form class** for user login.
2. Define a **view** to authenticate users and create a session.
3. Implement a **logout view** that clears the session.
4. Store and retrieve **cookies** for user preferences (e.g., theme selection).
5. Use **template inheritance** with `base.html` to ensure consistent layout for login/logout pages.
6. Define **URL patterns** for login, logout, and user dashboard pages.

Q8 You are developing a **Django blog application** where users can submit new blog posts. The blog post form should collect:

- **Title**
- **Content**
- **Author** (pre-filled for logged-in users)

After submission, the form should:

- Validate user input (e.g., required fields, non-empty content).
- Save the post to the database.
- Redirect to the blog post detail page after successful submission.
- Use **template inheritance** for a consistent layout across the blog pages.

Task:

1. Create a **Django Form class** for blog post submission.
2. Define a **view** to display the form and handle submissions.
3. Write the **URL pattern** for the blog post submission page.
4. Use **template inheritance** with a `base.html` layout and extend it in the blog post template.
5. Ensure only **authenticated users** can submit a post.

Q9 You are developing a **Django user authentication system** where new users can **register an account**. The registration form should collect:

- **Username**
- **Email**
- **Password** (with confirmation)

After submission, the form should:

- Validate user input (e.g., required fields, valid email, password match).
- Save the user to the database.
- Redirect to a success page or login page.
- Use **template inheritance** to ensure a consistent layout across pages.

Task:

1. Create a **Django Form class** for user registration.

2. Define a **view** to display the form and handle submission.
3. Write the **URL pattern** for the registration page.
4. Use **template inheritance** with a `base.html` layout and extend it in the registration template.
5. Implement validation logic and success redirection.

Q 10 You are building a **Django-based user dashboard** where only **logged-in users** can access their personalised content. The system should:

- Require users to **log in** before accessing the dashboard.
- Display a **welcome message** using session data (e.g., "Welcome, [username]!").
- Store **last login time** in a **cookie** and display it when the user logs in again.
- Provide a **logout option** that clears the session and redirects to the login page.
- Use **template inheritance** to maintain a consistent UI across authentication pages.

Task:

1. Create **views** to handle user login, logout, and dashboard access.
2. Implement **session handling** to store user authentication data.
3. Use **cookies** to store and display the last login timestamp.
4. Define **URL patterns** for login, logout, and dashboard pages.
5. Use **template inheritance** with `base.html` for consistent stylin