Event Registration System

1. Creating the Models:
    - Define the `Event` model with the following fields:
        - Title: The title of the event (CharField).
        - Description: A brief description of the event (TextField).
        - Date: The date on which the event will take place (DateField).
        - Time: The time at which the event will start (TimeField).
        - Location: The location where the event will be held (CharField).
    - Define the `Participant` model with the following fields:
        - Name: The name of the participant (CharField with `max_length` option to limit the length of participant names).
        - Email: The email address of the participant (EmailField with `unique` option to prevent duplicate email registrations for an event).
        - Phone: The phone number of the participant (CharField).
        - Event: A foreign key to the event the participant is registering for (ForeignKey relationship with the `Event` model).
2. Implementing Forms:
    - Create a Django form, `ParticipantRegistrationForm`, to allow participants to register for an event.
    - Include fields for the participant's name, email, and phone number using appropriate form field types.
    - Perform form validation to ensure all required fields are filled out and that the email and phone number are valid.
    - Save the participant's registration details to the database upon successful form submission.
3. Implementing Class-Based Views:
    - Create a class-based view, `EventListView`, to display a list of all events.
    - Inherit from Django's `ListView` and specify the `Event` model as the queryset.
    - Customize the template to display the titles, dates, and locations of the events.
    - Create a class-based view, `EventDetailView`, to display the details of a selected event.
    - Inherit from Django's `DetailView` and specify the `Event` model as the queryset.
    - Customize the template to display the event title, description, date, time, and location.
    - Create a class-based view, `EventRegistrationView`, to handle the participant registration form submission.
    - Inherit from Django's `FormView` and specify the `ParticipantRegistrationForm` as the form_class.
    - Implement the `form_valid` method to validate the form data, save the participant's registration to the database, and display a confirmation message to the participant.

Event Registration System

4. Querying and Filtering Data:
    - Create a class-based view, `RegisteredParticipantsView`, to display a list of registered participants for a specific event.
    - Inherit from Django's `ListView` and specify the `Participant` model as the queryset, filtered by the selected event.
    - Customize the template to display the participant details.
    - Create a class-based view, `EventSearchView`, to search for events based on title or date.
    - Inherit from Django's `ListView` and specify the `Event` model as the queryset.
    - Implement the `get_queryset` method to filter the events based on the search query parameters.
    - Customize the template to display the search form and the filtered events.
5. Implementing Cookies:
    - Create a cookie-based feature to store the events the user has recently viewed.
    - When a user views an event, store its relevant information (e.g., event ID, title) in the user's session or a cookie.
    - You can use Django's session framework to store the event information in the session. Ensure that you have `django.contrib.sessions` in your `INSTALLED_APPS` and `django.contrib.sessions.middleware.SessionMiddleware` in your `MIDDLEWARE` settings.
    - To set a cookie, use the `response.set_cookie()` method in your view after retrieving the relevant event information.
    - Retrieve the information from the session or cookie and display the recently viewed events on the homepage or in a dedicated section.
    - Create a view, `RecentlyViewedEventsView`, to retrieve the recently viewed events from the session or cookie and display them to the user.
    - Customize the template to display the recently viewed events.
6. User Authentication and Authorization:
    - Integrate user authentication and authorization to restrict certain functionalities to authenticated users only.
    - Use Django's built-in authentication system or Django's `User` model for user management.
    - Add login and registration views to handle user authentication and account creation.
    - Restrict certain views or functionalities, such as event registration or participant lists, to authenticated users only.
    - Use Django's decorators or mixins, such as `@login_required` or `LoginRequiredMixin`, to enforce authentication and authorization rules.