**Day 1: Data Collection, Preprocessing, and Model Development**

**Morning:**

1. **Define the Problem and Gather Data**
   * Finalize your problem statement and objectives.
   * Download relevant datasets (e.g., e-commerce data, product reviews).
   * Simulate data if real datasets are not available.
2. **Data Preprocessing**
   * Clean the data (handle missing values, remove duplicates).
   * Perform basic feature engineering (e.g., create user-item interaction matrix).
   * Normalize numerical features (e.g., product ratings, prices).
   * Encode categorical features (e.g., product categories, user IDs).

**Afternoon:**

1. **Implement Collaborative Filtering**
   * Set up **Matrix Factorization** (SVD/ALS) using **Surprise** or **Scikit-learn**.
   * Train the model with customer purchase history data and evaluate using metrics like **RMSE**.
2. **Build Content-Based Filtering**
   * Vectorize product descriptions using **TF-IDF** or **Word2Vec**.
   * Calculate **Cosine Similarity** between products.
3. **Hybrid Model**
   * Combine the Collaborative and Content-Based models for enhanced recommendations.

**Evening:**

1. **Test Models Locally**
   * Run some sample predictions and evaluate the model’s performance.
   * Fine-tune models based on accuracy (use **GridSearchCV** or hyperparameter tuning).

**Day 2: Flask API Development and Frontend Setup**

**Morning:**

1. **Set Up Flask**
   * Install **Flask** and set up a basic Flask project.
   * Create a folder structure: app.py for routes, templates/ for HTML, static/ for CSS/JS.
2. **Create Flask Routes**
   * Develop a route (/recommend) to handle requests for product recommendations.
   * Accept input (user data) and output recommendations via API.

**Afternoon:**

1. **Integrate Models into Flask**
   * Load your trained recommendation models into Flask using **pickle**.
   * Set up the recommendation function to take user input and return suggested products.
2. **Build Frontend (Basic UI)**
   * Create an HTML page where users can input data (e.g., user ID) to get recommendations.
   * Use basic **CSS** to style the page.
   * Use **JavaScript** to dynamically fetch recommendations from the backend.

**Evening:**

1. **Test Flask API Locally**
   * Test your API on **localhost** (e.g., http://127.0.0.1:5000/recommend).
   * Ensure that recommendations are displayed properly based on input.
2. **Refine the Interface**
   * Add input validation and error handling in Flask.
   * Make sure the interface is intuitive.

**Day 3: Deployment and Final Touches**

**Morning:**

1. **Prepare for Deployment**
   * Make sure all dependencies are listed in requirements.txt (e.g., Flask, Scikit-learn, Pandas).
   * Use Flask for local development; prepare for deployment on **Heroku** or **Render**.
   * Set up any configuration files (e.g., Procfile, runtime.txt for Heroku).
2. **Deploy Flask App**
   * Deploy your Flask app to **Heroku** or **Render** (free hosting platforms).
   * Follow the instructions to set up your repository for deployment.
   * Push the code to **GitHub**, then link it to your Heroku/Render account.

**Afternoon:**

1. **Test on Cloud**
   * Test the deployed application on the cloud (e.g., https://your-app-name.herokuapp.com/recommend).
   * Ensure everything works as expected, including API calls and the frontend interface.
2. **Refinements**
   * Optimize performance for faster recommendations (e.g., caching).
   * Improve the UI/UX (e.g., show product images or brief details).

**Evening:**

1. **Documentation and Final Review**
   * Add detailed documentation in the **GitHub repository** about the setup and usage.
   * Ensure clear README instructions (e.g., how to run locally, deployment steps).
   * Include model explanation, API endpoints, and frontend integration in your documentation.
2. **Final Testing & Deployment**
   * Test the deployed application one last time.
   * Ensure that all the functionalities work (recommendations, UI, API).

**Tools & Technologies:**

* **Programming Languages**: Python, HTML, CSS, JavaScript
* **Frameworks**: Flask, Scikit-learn, Pandas, NumPy
* **Deployment**: Heroku or Render
* **Libraries**: Surprise (Collaborative Filtering), Scikit-learn (Content-Based Filtering), Flask
* **Database**: SQLite or any simple file storage for user-product interactions (if necessary)

For a **Personalized Product Recommendation System** like the one you're building for **Kilimall**, you would need data related to both **users** and **products**, as well as their interactions. Here's a breakdown of the **datasets** you need and **considerations** for the project:

**1. User Data**

You need information about the users who will receive the recommendations. This data helps in personalizing suggestions based on their behaviors, preferences, and interactions.

**Attributes to Include:**

* **User ID**: A unique identifier for each user.
* **User Profile Information** (optional): This could include age, gender, location, etc.
* **Purchase History**: Products purchased by the user in the past (could include product IDs and timestamps).
* **Browsing History**: Products viewed by the user (could include product IDs and timestamps).
* **Ratings/Reviews** (optional): Ratings given by users to products or textual reviews.

**2. Product Data**

This data helps to understand the product catalog and the features that can be used to make recommendations.

**Attributes to Include:**

* **Product ID**: A unique identifier for each product.
* **Product Name**: The name of the product.
* **Product Category**: The category or type of product (e.g., electronics, clothing, etc.).
* **Product Description**: Detailed text describing the product.
* **Product Features**: Features such as price, brand, color, size, etc.
* **Price**: Price of the product.
* **Product Images** (optional): Can be used for image-based recommendations.
* **Product Ratings/Reviews**: User ratings and textual reviews to aid in the content-based filtering.

**3. Interaction Data (User-Product Interactions)**

This is crucial for generating personalized recommendations. It connects users to products based on their interactions.

**Attributes to Include:**

* **User ID**: A unique identifier for each user.
* **Product ID**: A unique identifier for each product.
* **Interaction Type**: Could be a purchase, view, or add to cart.
* **Timestamp**: When the interaction happened (important for time-based recommendations).
* **Rating/Feedback** (optional): If users rate products, this can enhance your recommendation engine.

**4. Behavioral Data (Optional)**

This data can help refine your recommendation engine by tracking more granular user behavior, which can be used for context-aware recommendations.

**Attributes to Include:**

* **Clickstream Data**: Tracks the user's clicks, searches, and other interactions on the website.
* **Session Data**: Time spent on product pages, cart interactions, etc.
* **Search History**: Past search queries made by the user.

**Sources of Data:**

* **Kilimall Database**: If you have access to the company's database, it would be ideal to use their real user and product data.
* **Public Datasets**: You can use publicly available datasets as placeholders, especially when you’re testing your model. Examples:
  + **MovieLens Dataset**: While this dataset is for movie recommendations, it shares similar structure and can be adapted for products.
  + **Amazon Product Data**: Contains user reviews, product metadata, and ratings that you can repurpose for an e-commerce recommendation system.
  + **Instacart Dataset**: Available on Kaggle, this dataset contains shopping history and could be used for building a recommendation system.
  + **UCI Repository**: Offers various datasets for collaborative filtering, like the **Retail Market Basket** dataset.

**Key Considerations for the Project:**

1. **Data Privacy & Ethics**:
   * Ensure that the user data you use complies with **data privacy** laws (e.g., GDPR in Europe, Kenya’s Data Protection Act).
   * Anonymize user data wherever possible to ensure privacy.
2. **Data Sparsity**:
   * **Collaborative filtering** models often suffer from sparsity, meaning not all users rate all products. Techniques like matrix factorization (SVD) can help mitigate this issue, but you should plan to handle sparse matrices efficiently.
3. **Cold-Start Problem**:
   * New users or products with no interaction history present a challenge. **Content-based filtering** or **popularity-based recommendations** can address the cold-start problem.
4. **Scalability**:
   * Your system should be scalable to handle large numbers of users and products. As your dataset grows, using **matrix factorization** or **deep learning-based models** may become necessary to keep up with the increasing complexity.
5. **Evaluation Metrics**:
   * Choose relevant metrics to evaluate your model’s performance:
     + **Precision** and **Recall** for evaluating relevance.
     + **RMSE** or **MAE** for measuring accuracy in predicted ratings.
     + **Diversity** of recommendations (to avoid recommending the same items repeatedly).
6. **Model Interpretability**:
   * It’s essential for users or stakeholders to understand how recommendations are made. Consider the interpretability of your models, especially if using black-box models like neural networks.
7. **User Experience**:
   * Your recommendation system should be integrated smoothly with the frontend of the platform (Kilimall). Ensure the recommendations are presented in a user-friendly way.
   * Provide clear options to users to interact with recommendations (e.g., "Show more like this").
8. **Performance and Latency**:
   * Recommendations should be fast and not block the user experience. Consider using **caching** mechanisms or pre-computed models if the system requires real-time interaction.

**Final Thoughts:**

* Ensure that your datasets are properly **cleaned** and preprocessed before feeding them into the recommendation model.
* Try combining **Collaborative Filtering** and **Content-Based Filtering** to build a **hybrid system** that performs better in various scenarios.
* **Monitor** and **update** the model periodically as new data arrives to keep the system relevant.

By considering these datasets and key points, you will build a robust and scalable **personalized recommendation system** for your project.