**A PROJECT REPORT**

**On**

**PRUS: Product Recommendation System Based on User Specification and Customer Reviews**

**Submitted in the partial fulfilment of requirements to**

**CS – 363 – Term Paper**

**By**

**Batch – 19**

**Oruganti Monik Paparao (Y22CS139)**

**Pendyala Skanda Bhagavan (Y22CS145)**

**Tulam Sai Sudheer (Y22CS184)**



**R.V.R. & J.C. COLLEGE OF ENGINEERING (Autonomous)**

**(NAAC ‘A+’ Grade)**

**Approved by AICTE :: Affiliated to Acharya Nagarjuna University**

**Chandramoulipuram::Chowdavaram**

**GUNTUR – 522 019, Andhra Pradesh, India.**

**Contents:**

**1. Goal of the Project**

**2. Mechanism to be Followed**

**3. Tools, and algorithms to be Followed**

**4. Architecture diagram**

**5. Architecture Overview**

**6. Data Set Description**

**7. Functional Requirements**

**8. Non -Functional requirements**

**Goal of the Project**

The research paper titled **"PRUS: Product Recommender System Based on User Specifications and Customers Reviews"** introduces a new and improved way to recommend products to users by understanding their specific needs more accurately. Most existing recommendation systems only focus on the overall sentiment of product reviews or highlight only the positive aspects. However, this approach can overlook important negative feedback that may affect a user’s decision. To solve this problem, the proposed system, called **PRUS**, takes into account both **positive and negative sentiments** found in customer reviews.

One of the main features of PRUS is that it allows users to **specify exactly which product features they care about**, such as camera quality, battery life, or screen resolution. Instead of looking at the entire review as one opinion, the system breaks the reviews down into **individual sentences** and identifies opinions related to each feature. It then performs **aspect-level sentiment analysis**, which means it understands the user’s opinion on each feature separately, rather than as a whole.

To rank the products effectively, the authors introduce a method called **RANK-ify**, which assigns scores to each product based on how often and how positively or negatively each feature is mentioned. This scoring method also gives users the flexibility to decide whether they want to prioritize positive feedback, negative feedback, or both equally.

Overall, the goal of the paper is to create a more **personalized, accurate, and useful product recommendation system** that helps users make better decisions by understanding detailed customer opinions, not just average ratings or general reviews.

**Mechanism to be Followed**

Here’s the **step-by-step mechanism** of how the PRUS system works, explained in simple words:

**Step 1: Collect Product Reviews**

The system starts by collecting a large number of **customer reviews** from platforms like Amazon, focusing on products such as mobile phones.

**Step 2: Clean and Prepare the Reviews**

The reviews are **cleaned** by removing useless data like short or incomplete reviews, stopwords, punctuation, and numbers. Words are also simplified using techniques like **lemmatization** so that similar words are treated as the same.

**Step 3: Break Reviews into Sentences**

Each review is **split into sentences** to find specific opinions about different features (like battery, camera, etc.) rather than looking at the whole review.

**Step 4: Extract Features and Sentiments**

From each sentence, the system **identifies the product feature** being discussed (e.g., "battery") and the **sentiment** (positive, negative, or neutral) expressed about that feature using tools like **TextBlob**.

**Step 5: Match Features with User Query**

The user gives a **search query** specifying the features they care about (e.g., “good camera, long battery life”). The system matches these requested features with the features found in the reviews.

**Step 6: Assign Scores to Features**

Each product is assigned **scores** for each feature based on how many times that feature is mentioned positively or negatively. A formula is used to give more or less importance to positive or negative feedback, depending on what the user prefers.

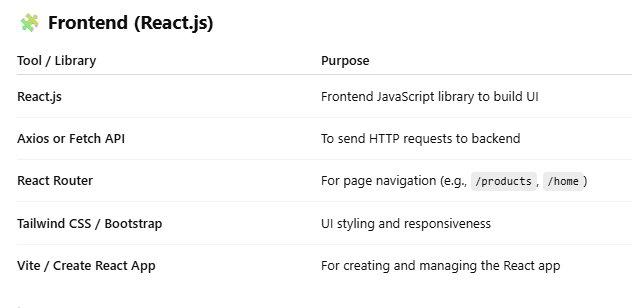
**Step 7: Rank Products (RANK-ify Algorithm)**

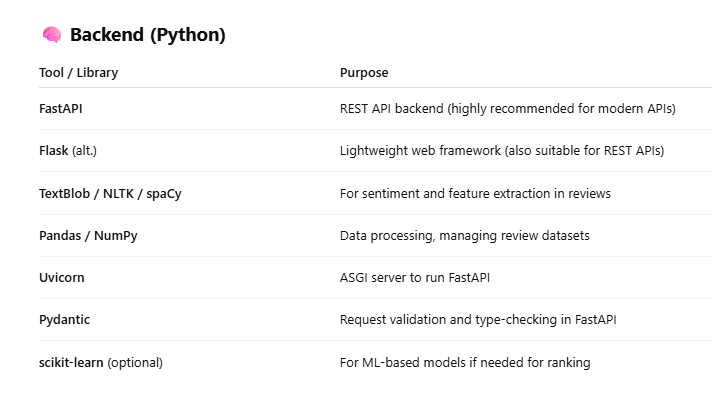
The **RANK-ify algorithm** calculates a total score for each product by combining all feature scores. Products are then **ranked** from best to worst based on how well they match the user’s needs.

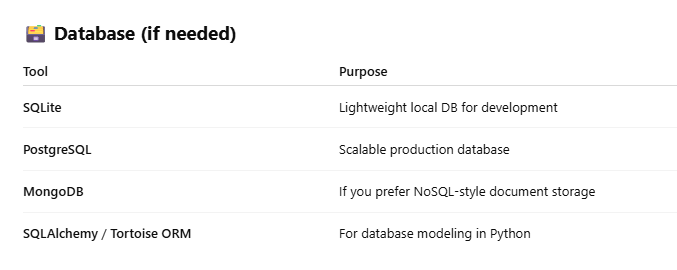
**Step 8: Display Recommended List**

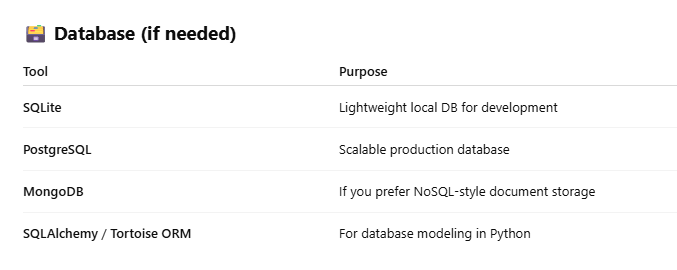
Finally, the system shows a **ranked list of recommended products** that best fit the user’s specified features and preferences, considering both good and bad reviews.

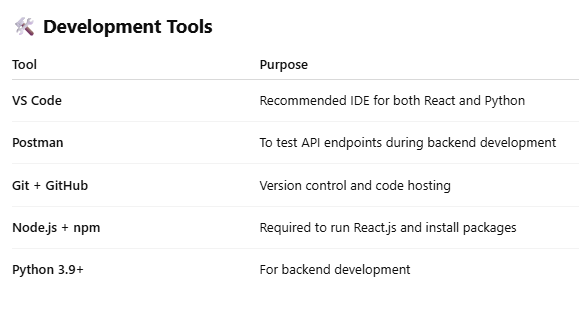
**Tools, and Algorithms needed for Implementation**

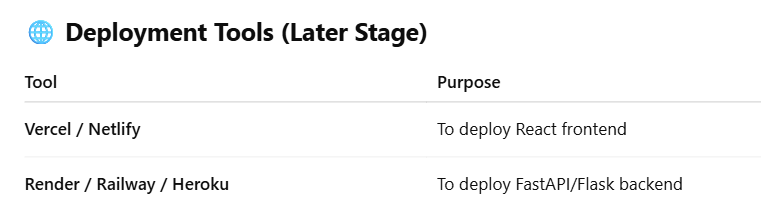




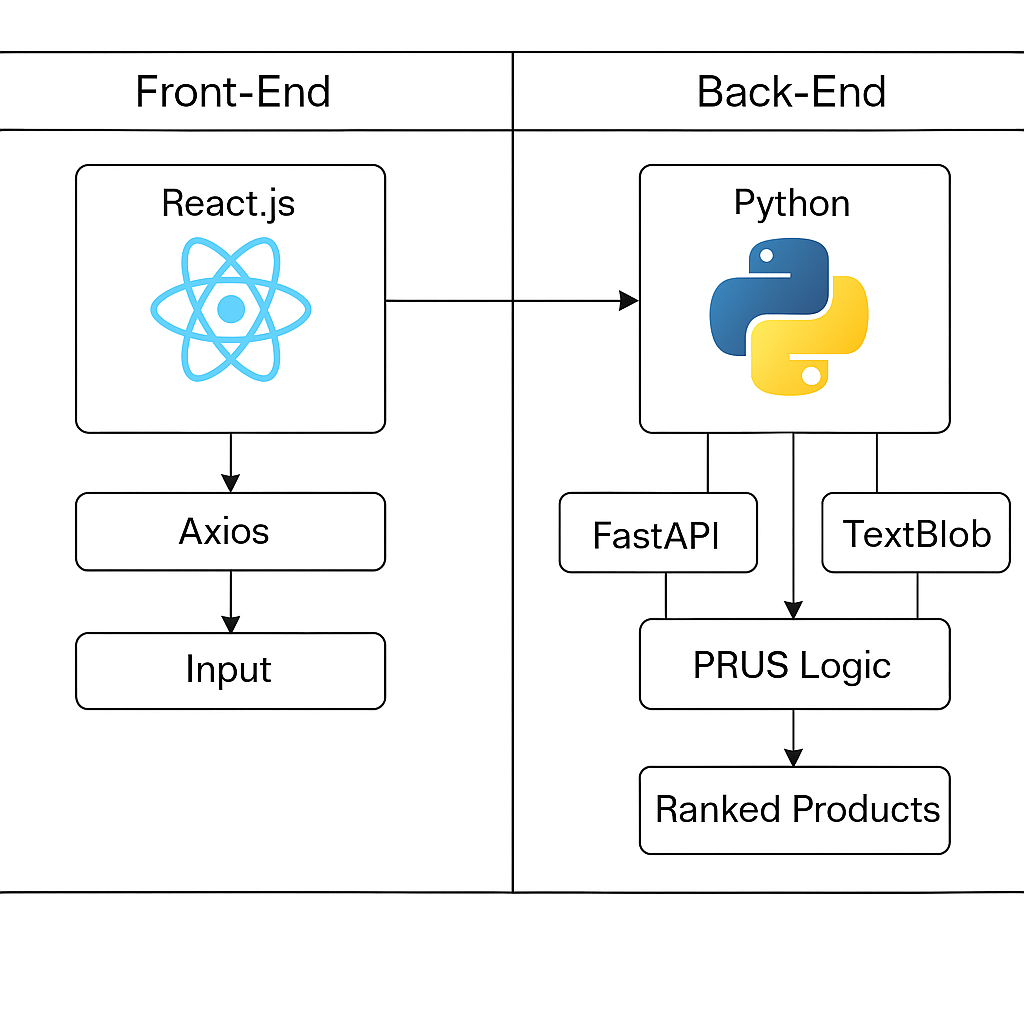








**Architecture Diagram**

****

**Architecture Overview**

The architecture of the proposed product recommendation system is divided into two main components: the **frontend** developed using **React.js**, and the **backend** built with **Python** using frameworks like **FastAPI**. On the frontend, users interact with a clean and responsive interface built in React. They enter specific product preferences, such as “long battery life” or “high-resolution screen,” through input fields or forms. Once the user submits their preferences, the React application uses a library called **Axios** to send an HTTP POST request to the backend API. This request contains the user-specified features and optionally assigned weights that indicate the importance of positive or negative sentiments. The React app is also responsible for displaying the final list of recommended products returned by the backend in a user-friendly format.

On the backend side, **FastAPI** receives the incoming request and forwards the data to the **core PRUS logic**, which is implemented in Python. The backend system uses **TextBlob** (or similar NLP tools) to process a large number of customer reviews, breaking them into sentences, extracting product features, and assigning sentiment polarity (positive or negative) to each feature. This information is then fed into the **RANK-ify algorithm**, which calculates a ranking score for each product based on how well it matches the user’s query, considering both the frequency and sentiment strength of each feature. The products are sorted accordingly, and a top-N list is created. This ranked list is then sent back through the FastAPI response to the React frontend, where it is rendered for the user. This architecture ensures a modular, scalable, and highly personalized product recommendation experience based on actual customer feedback.

**Data Set Description**

* **Name:** Amazon Unlocked Mobile Review Dataset
* **Total Entries (Rows):** 413,840
* **Total Columns (Features):**3
* *Source*: **Kaggle Data set**
* *Data\_set\_Link*: [**https://www.kaggle.com/datasets/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones/data**](https://www.kaggle.com/datasets/PromptCloudHQ/amazon-reviews-unlocked-mobile-phones/data)

**Functional Requirements**

These define **what the system should do**:

1. **User Input of Preferences**  
   Users should be able to enter specific product features (e.g., "battery life", "camera") via a web form.
2. **Submit Query to Backend**  
   The system should send the user preferences and sentiment weights (positive/negative) to the backend API using Axios.
3. **Sentiment Analysis of Reviews**  
   The backend should process reviews, perform sentiment analysis on each sentence, and identify feature-level sentiment using NLP tools like TextBlob.
4. **Feature Matching and Scoring**  
   The backend should match user-specified features with review data and calculate ranking scores using the RANK-ify algorithm.
5. **Generate Ranked Product List**  
   Based on user input and sentiment analysis, the system should return a list of ranked products.
6. **Display Results on Frontend**  
   The ranked list of recommended products should be displayed in a clear and user-friendly way (e.g., cards, list, table).
7. **Handle Errors and Invalid Input**  
   The system should validate input and gracefully handle missing data, API failures, or incorrect user queries.

**Non-FunctionalRequirements**

These define **how the system should behave**:

1. **Performance**  
   The system should return ranked results within a few seconds of receiving the user input.
2. **Scalability**  
   The backend should be designed to handle large datasets (e.g., thousands of reviews) and many users simultaneously.
3. **Security**  
   Input should be sanitized to prevent injection attacks. Secure communication (e.g., HTTPS) should be used in production.
4. **Usability**  
   The frontend should be intuitive, responsive (mobile-friendly), and easy to use for non-technical users.
5. **Maintainability**  
   The codebase should be modular and well-documented to allow future updates or improvements to the algorithm or UI.
6. **Reliability**  
   The system should remain available and correctly function even if one part (e.g., review sentiment) encounters errors.
7. **Portability**  
   The application should be deployable across various environments (local, cloud platforms like Vercel/Render).