**Smart Recipe Generator with Pantry Tracker**

1. **Abstract**

The paper outlines a Smart Recipe Generator with Pantry Tracker, a application aimed at helping to reduce food waste while also enabling easier meal planning. The application maintains a digital pantry with MySQL to store the pantry, keeps track of the food items and their expiry dates, and provides recipe suggestions based on available ingredients and dietary restrictions. One of the most significant features is the option to suggest recipes based on the ability to use leftover food items to create unique dishes. Ultimately, this application aims to improve cooking efficiency, and it assists users in making healthy and affordable meal choices typical of everyday planning and consumption. Personalization is further enhanced through recommendation strategies such as content-based filtering, collaborative filtering, and hybrid models, which refine suggestions using user feedback data, such as article clicks and favorites. This design not only ensures fast retrieval and integrity of data but also adapts recommendations over time, addressing common challenges like the cold-start problem, and fairness in recommendations.

1. **Introduction**

In today’s digital world, people consume news from multiple platforms such as websites, apps, and social media. Every minute, thousands of new articles are published online, leading to **information overload**, where users are exposed to more content than they can process [1,2]. This overload often causes stress, fatigue, and in some cases results in **news avoidance**, where people disengage from news altogether [3,4].

A promising solution to this problem is the development of **personalized news recommended systems**. Instead of showing every available article, these systems learn individual user preferences—such as interest in politics, sports, or health—and suggest only the most relevant articles [5]. Techniques used include **content-based filtering**, **collaborative filtering**, and **hybrid approaches**, which combine both strategies [6]. These methods analyze user interactions such as clicks, reading time, or saved items to improve recommendations over time.

To collect news articles, **RSS feeds** (Really Simple Syndication) are widely used. Many news portals provide RSS feeds in XML format, allowing systems to automatically aggregate and update news content from multiple sources in real time [7]. A relational database then organizes this data, storing information about users, preferences, and article history, which enables both **trending news discovery** and **personalized recommendations** [8].

Despite their usefulness, recommender systems face challenges such as the **cold-start problem** (difficulty recommending new users or new articles) [9], **popularity bias** (over-representing widely read articles), and lack of diversity or fairness in recommendations [10]. These issues are active research areas, with many techniques being explored to ensure recommendations remain relevant, diverse, and trustworthy.

Overall, combining RSS-based news aggregation, relational database management, and intelligent recommendation algorithms creates a powerful system. Such systems not only reduce information overload but also make news consumption faster, smarter, and more enjoyable.

[1] Eppler, M. J., & Mengis, J. (2004). The concept of information overload: A review of literature from organization science, accounting, marketing, MIS, and related disciplines. *The Information Society*, 20(5), 325–344.

[2] Bawden, D., & Robinson, L. (2009). The dark side of information: Overload, anxiety, and other paradoxes and pathologies. *Journal of Information Science*, 35(2), 180–191.

[3] Skovsgaard, M., & Andersen, K. (2020). Conceptualizing news avoidance: Towards a shared understanding of different causes and potential solutions. *Journalism Studies*, 21(4), 459–476.

[4] Nguyen, A. (2013). Online news audiences: The challenges of web metrics. *Palgrave Macmillan*.

[5] Karimi, M., Jannach, D., & Jugovac, M. (2018). News recommender systems–Survey and roads ahead. *Information Processing & Management*, 54(6), 1203–1227.

[6] Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734–749.

[7] Treloar, A. (2001). RSS: A brief review and future directions. *Ariadne*, (27).

[8] Elahi, M., Ricci, F., & Rubens, N. (2016). A survey of active learning in collaborative filtering recommender systems. *Computer Science Review*, 20, 29–50.

[9] Lam, S. K., & Riedl, J. (2004). Shilling recommender systems for fun and profit. *Proceedings of the 13th International Conference on World Wide Web (WWW’04)*, 393–402.

[10] Zhu, Z., Xia, Y., & Zhang, Y. (2021). Popularity bias in recommendation: A multi-stakeholder perspective. *ACM Transactions on Information Systems*, 39(1), 1–27.

**III. Methodology**

The methodology for developing the **Smart Recipe Generator** follows a structured approach, combining relational database design, backend logic in Flask, and an interactive frontend interface. The system is designed to help users quickly discover recipes based on available ingredients while ensuring simplicity, efficiency, and usability.

**A. System Architecture**

The system architecture is built on a **three-tier model**, ensuring modularity, maintainability, and separation of concerns.

1. **Presentation Layer (Frontend):**
   * Implemented using **HTML and CSS**, the frontend provides a clean and user-friendly interface.
   * The input form allows users to enter ingredients (comma-separated) and select between two options:
     + *Recipes With All Ingredients*
     + *Recipes With Any Ingredient*
   * Results are displayed dynamically with recipe details (name, description, ingredients, and instructions).
   * The design prioritizes readability and a smooth user experience with styled containers, hover effects, and intuitive navigation.
2. **Application Layer (Backend):**
   * Built with **Flask (Python)**, the backend handles form submissions, processes user input, and queries the database.
   * When ingredients are submitted, the system checks whether to match **all** or **any** of the entered ingredients.
   * Parameterized SQL queries ensure secure database interaction and prevent SQL injection.
   * Flask routes render the processed results back into the frontend template (frontend.html).
3. **Data Layer (Database):**
   * Implemented using **MySQL**, the database stores structured recipe information in a table (info).
   * Key attributes include:
     + name (recipe title)
     + description (short summary)
     + recipeIngredientParts (list of ingredients)
     + recipeInstructions (cooking steps)
   * SQL LIKE queries are used to search recipes efficiently based on ingredients.

This layered architecture ensures that each component (frontend, backend, database) can be modified or upgraded independently, supporting scalability and maintainability.

**B. User Input and Recipe Search**

* Users provide available ingredients in a text field (comma-separated).
* Depending on the action chosen:
  + **All Ingredients Search:** Only recipes containing *all* provided ingredients are fetched.
  + **Any Ingredient Search:** Recipes containing *at least one* of the provided ingredients are retrieved.
* A limit of 50 results is applied to ensure quick response times.

**C. Recipe Display and User Experience**

* Retrieved recipes are displayed in well-styled containers showing:
  + Recipe Name
  + Description
  + Ingredients
  + Step-by-step Instructions
* Hover effects and shadows improve interactivity, while a fallback message (“No recipes found”) guides users to refine their search.
* The design emphasizes clarity, ensuring minimal cognitive load and accessibility for all users.

**D. Security and Data Integrity**

* **Parameterized Queries:** Protect the system against SQL injection.
* **Form Validation:** Ensures that empty or malformed ingredient lists are handled gracefully.
* **Database Integrity:** Proper schema design with unique identifiers and structured fields prevents duplication and maintains consistency.

**References**

[1] M. J. Eppler and J. Mengis, *The concept of information overload: A review of literature*, The Information Society, vol. 20, no. 5, pp. 325–344, 2004.

[2] D. Bawden and L. Robinson, *The dark side of information: Overload and anxiety*, J. Inf. Sci., vol. 35, no. 2, pp. 180–191, 2009.

[3] M. Skovsgaard and K. Andersen, *Conceptualizing news avoidance*, Journalism Studies, vol. 21, no. 4, pp. 459–476, 2020.

[4] A. Nguyen, *Online news audiences: The challenges of web metrics*, Palgrave Macmillan, 2013.

[5] M. Karimi, D. Jannach, and M. Jugovac, *News recommender systems – Survey and roads ahead*, Inf. Process. Manage., vol. 54, no. 6, pp. 1203–1227, 2018.

[6] G. Adomavicius and A. Tuzhilin, *Toward the next generation of recommender systems*, IEEE Trans. Knowl. Data Eng., vol. 17, no. 6, pp. 734–749, 2005.

[7] A. Treloar, *RSS: A brief review and future directions*, Ariadne, no. 27, 2001.

[8] M. Elahi, F. Ricci, and N. Rubens, *A survey of active learning in collaborative filtering recommender systems*, Comput. Sci. Rev., vol. 20, pp. 29–50, 2016.

[9] S. K. Lam and J. Riedl, *Shilling recommender systems for fun and profit*, Proc. WWW, pp. 393–402, 2004.

[10] Z. Zhu, Y. Xia, and Y. Zhang, *Popularity bias in recommendation: A multi-stakeholder perspective*, ACM Trans. Inf. Syst., vol. 39, no. 1, pp. 1–27, 2021.