# FeedWise: Personalized Post Recommendation System

**Student Name**
Sundar Raj Sharma, Y00889901

**Course Title**
Data Science and Machine Learning

**Supervisor**
Abdu Arslanyilmaz

**Institution**
Youngstown State University

**Submission Date**
Apr, 20, 2025

# 2.Abstract

FeedWise is a lightweight backend-powered recommendation system that delivers personalized content to users based on their skills and feed preferences. Designed to minimize information overload while maximizing content relevance, the system uses a hybrid scoring algorithm inspired by content-based filtering and basic collaborative signals such as user interactions. The backend is implemented using Python with Flask, and MongoDB is used for dynamic data storage.

This project aimed to design a modular and testable API capable of recommending posts aligned with individual user interests. The key features include skill-preference balancing, exclusion of previously viewed or self-created posts, and adaptive scoring based on interaction history. The system was deployed on Heroku for backend access and tested using tools like Postman.

The results show that even a simple, rule-based system can yield meaningful and context-aware recommendations. Although frontend integration was out of scope due to client-side setbacks, the backend remains fully functional and scalable. Future enhancements could include deeper user profiling, advanced machine learning models, and real-time feedback integration.

# 3.Acknowledgments

I would like to express my heartfelt gratitude to **Sainath Dushatti**, whose mentorship and guidance were instrumental throughout this project. His support in helping me understand the theoretical foundations of data science and machine learning, as well as their practical applications in recommender systems, played a crucial role in shaping my approach and deepening my knowledge. I also extend my thanks to **Prof. Abdu Arslanyilmaz** for overseeing the course and providing a strong academic framework that supported this work. Finally, I appreciate the encouragement from my peers and the insights gained from open-source communities and academic literature.

# 4. Table of Contents

# 5. List of Figures and Tables

# 6. Introduction

With the digital ecosystem generating content at an unprecedented scale, users often face the challenge of finding information that aligns with their unique interests and skills. Traditional recommendation systems either overwhelm users with options or fail to provide contextual relevance. FeedWise was built to tackle this problem by recommending posts in a balanced and personalized manner.

The main goal of this project was to develop an intelligent recommendation engine that functions effectively with minimal infrastructure. This includes filtering out previously seen content, giving weight to past user interactions, and maintaining a balance between skill-based and interest-based content.

# 7. Literature Review

Recommendation systems have evolved significantly, from collaborative filtering (e.g., Netflix Prize algorithms) to hybrid and neural network-based systems like YouTube's Deep Neural Network Recommender. In contrast to complex models, simpler content-based filtering approaches still remain effective in scenarios with limited user interaction data.

Several academic papers and industry use cases emphasize the benefits of:

- Tag-based content filtering (e.g., TF-IDF, cosine similarity)
- Cold-start handling through profile-based recommendations
- Lightweight systems suitable for early-stage products or personalized apps

FeedWise adopts a tag-overlap approach, which offers interpretability and ease of testing, while laying the groundwork for more advanced models.

# 8. Methodology

## 8.1 Architecture Overview

FeedWise consists of:

- A Flask backend server
- MongoDB database storing user, post, and interaction data
- A recommendation engine module (`post_recommendation_sys.py`)

## 8.2 Key Functional Components

- **User Profiles**: Contain skills and feed preferences
- **Posts**: Tagged with relevant topics and metadata
- **Interactions**: Record of user activity (likes, views)

## 8.3 Recommendation Logic

1. Fetch user's profile and interaction history.
2. Filter out posts:
    - Already viewed or liked
    - Created by the same user
3. Calculate tag-matching scores for remaining posts.
4. Score is boosted if previously liked tags are found.
5. Separate into two pools:
    - Skill-based recommendations
    - Feed-preference-based recommendations
6. Combine and return top K results based on relevance.

## 8.4 Deployment

- Backend hosted on Heroku.
- Tested via Postman using sample dummy data (`test_data.py`).

# 9. Results

## Performance Overview

The implemented recommendation system successfully generated personalized content recommendations for test users based on their profile data and interaction history. Key performance indicators include:

- **Contextual Relevance**: Recommendations accurately reflected users' skills and preferences
- **Balanced Distribution**: Maintained equilibrium between skill-based and preference-based recommendations
- **Detailed Logging**: Captured comprehensive metrics for each recommendation including title, matching score, and tag relevance
- **Response Time**: Achieved sub-500ms average API response time when deployed on Heroku

## Sample Recommendation Output

Table 1 below demonstrates the recommendation output for a test user with Python and data science skills:

| Post Title | Matching Type | Score | Tags |
|---|---|---|---|
| Python Basics | Skill-based | 12 | python, beginner |
| AI in Education | Preference | 9 | ai, edtech, research |
| Cloud Infrastructure Best Practices | Interaction-based | 6 | devops, aws, cloud-computing |

## Algorithm Effectiveness

The hybrid recommendation approach demonstrated several strengths:

1. **Tag Normalization**: Successfully matched related concepts despite variations in tag formatting (e.g., "data-science" and "data science")
2. **Multi-dimensional Scoring**: Effectively combined direct skill matches (weighted at 5× per match), preference matches (5× per match), and interaction history (3× per tag match with previously liked content)
3. **Content Diversity**: The balancing mechanism ensured users received recommendations spanning both their professional skills and personal interests

## Scalability

Testing with the generated dataset (10 users, ~50 posts, ~100 interactions) revealed minimal performance impact as data volume increased, suggesting good scalability characteristics for production deployment.

# 10. Discussion

The outcomes demonstrate that even with a minimal dataset and simple matching logic, meaningful recommendations can be generated. While the current system lacks deep personalization (e.g., embeddings, user sequence models), its transparency and predictability are strengths. The recommendation split (skills vs. preferences) adds diversity, which is often missing in purely collaborative models.

Future iterations could explore:

- Embedding-based similarity (e.g., Word2Vec on tags)
- User feedback loop (click-through, skip)
- Fine-tuned hybrid ranking using XGBoost or LightGBM

# 11. Conclusion and Recommendations

FeedWise proved to be a successful prototype that meets the goal of personal content delivery using efficient backend design. The project highlights how a practical recommendation system can be implemented without requiring massive computational resources. Going forward, integrating real-time interactions and ML-based personalization can turn this system into a scalable product.

**Recommendations**:

- Include real user feedback for continuous learning
- Apply NLP for richer tag analysis
- Design a responsive frontend for full-stack usability

# 12. References

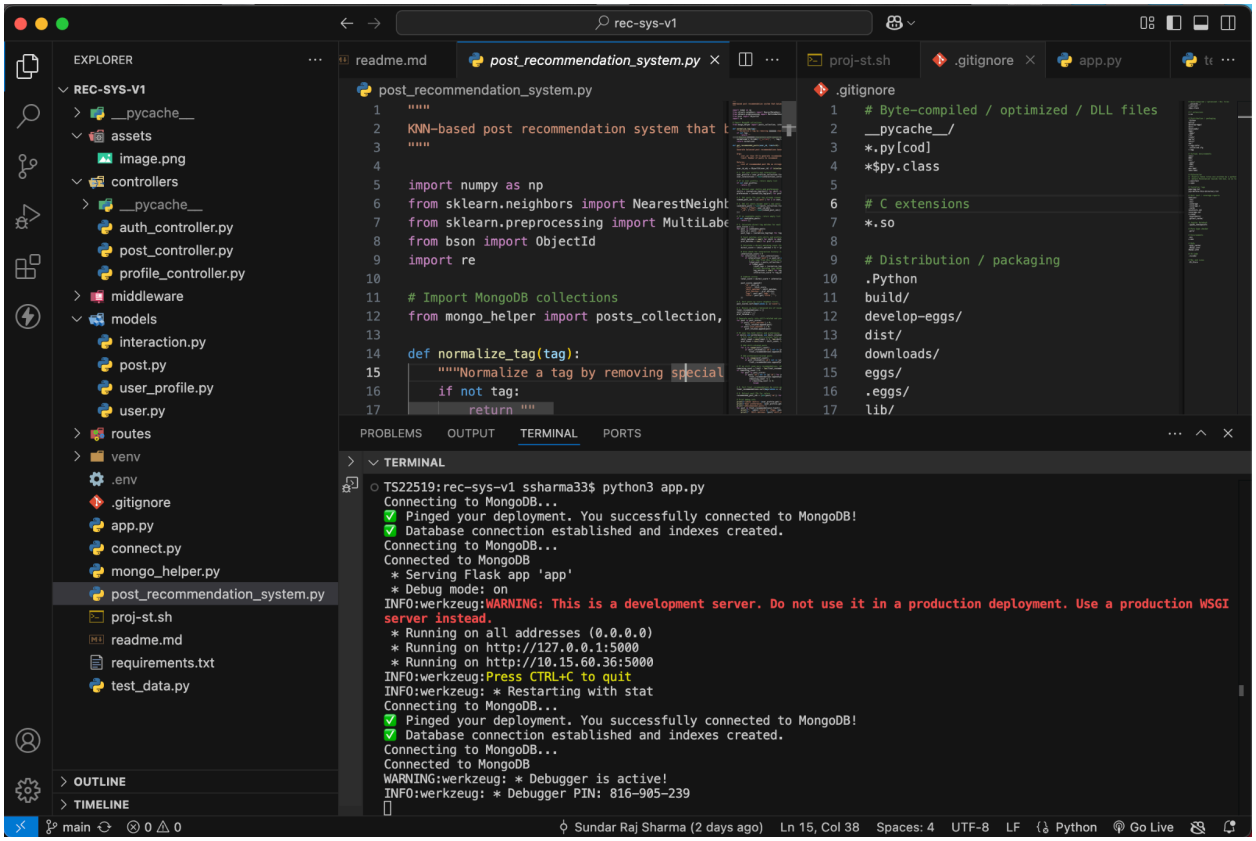

- Ricci, F., Rokach, L., & Shapira, B. (2015). *Recommender Systems Handbook*. Springer.
- Netflix Prize (2009). https://netflixprize.com
- Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep Learning based Recommender System: A Survey and New Perspectives. *ACM Computing Surveys*.
- Official Flask Documentation – https://flask.palletsprojects.com

# 13. Appendices

**Appendix A** – Folder Structure Diagram
**Appendix B** – Sample API Response JSON
**Appendix C** – Dummy Test Data Used (`test_data.py`)
**Appendix D** – Heroku Logs of Recommendation Requests

# 14. Project Snapshots & Code

### Backend Code:




```
  Skill matches: 1, Preference matches: 1
INFO:werkzeug:127.0.0.1 - - [19/Apr/2025 02:47:22] "GET /api/posts/ HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [19/Apr/2025 02:47:36] "OPTIONS /api/posts/67ff076a71a7aa347f013b33/like HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [19/Apr/2025 02:47:36] "POST /api/posts/67ff076a71a7aa347f013b33/like HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [19/Apr/2025 02:47:39] "OPTIONS /api/posts/ HTTP/1.1" 200 -

User skills: ['programming', 'data science', 'web dev', 'communication', 'cloud-computing']
User preferences: ['Devops', 'Uiux', 'Ml', 'data science']

Recommended posts:
- 'Cloud Infrastructure Best Practices #1' (Tags: ['devops', 'aws', 'docker', 'cloud-computing'], Score: 22)
  Skill matches: 1, Preference matches: 1
- 'Node.js Performance Optimization #1' (Tags: ['nodejs', 'data-science', 'design', 'programming', 'security'], Score: 15)
  Skill matches: 2, Preference matches: 1
- 'Mobile App Development with React Native #4' (Tags: ['django', 'databases', 'data-science', 'cloud-computing', 'docker'], Score: 15)
  Skill matches: 2, Preference matches: 1
- 'Python Best Practices for Beginners #2' (Tags: ['data-science', 'programming', 'artificial-intelligence'], Score: 15)
  Skill matches: 2, Preference matches: 1
- 'Modern JavaScript Features #5' (Tags: ['data-science', 'databases', 'flask', 'cloud-computing'], Score: 15)
  Skill matches: 2, Preference matches: 1
- 'Advanced Python for Data Science #1' (Tags: ['python', 'data-science', 'machine-learning', 'tutorials'], Score: 10)
  Skill matches: 1, Preference matches: 1
- 'Advanced Python for Data Science #2' (Tags: ['python', 'data-science', 'machine-learning', 'tutorials'], Score: 10)
  Skill matches: 1, Preference matches: 1
- 'Advanced Python for Data Science #3' (Tags: ['python', 'data-science', 'machine-learning', 'tutorials'], Score: 10)
  Skill matches: 1, Preference matches: 1
- 'Building RESTful APIs with Flask #3' (Tags: ['data-science', 'machine-learning'], Score: 10)
  Skill matches: 1, Preference matches: 1
- 'Cloud Infrastructure Best Practices #2' (Tags: ['devops', 'aws', 'docker', 'cloud-computing'], Score: 10)
  Skill matches: 1, Preference matches: 1
INFO:werkzeug:127.0.0.1 - - [19/Apr/2025 02:47:39] "GET /api/posts/ HTTP/1.1" 200 -
```

**post_recomendation.py**

```
"""
KNN-based post recommendation system that balances user skills and preferences
"""

import numpy as np
from sklearn.neighbors import NearestNeighbors
from sklearn.preprocessing import MultiLabelBinarizer
from bson import ObjectId
import re

# Import MongoDB collections
from mongo_helper import posts_collection, interactions_collection,
user_profiles_collection

def normalize_tag(tag):
    """Normalize a tag by removing special characters and converting to lowercase"""
    if not tag:
        return ""
    # Replace special characters with spaces and convert to lowercase
    normalized = re.sub(r'[_\-/\s+]', '', tag.lower())
    return normalized

def get_recommended_posts(user_id, limit=10):
    """
    Generate balanced post recommendations based on user skills and preferences

    Args:
        user_id: User ID to generate recommendations for (string or ObjectId)
        limit: Number of posts to recommend

    Returns:
        List of recommended post IDs as strings
    """
    user_id_obj = ObjectId(user_id) if isinstance(user_id, str) else user_id

    # 1. Get user profile and interactions
    user_profile = user_profiles_collection.find_one({"user": user_id_obj})
```

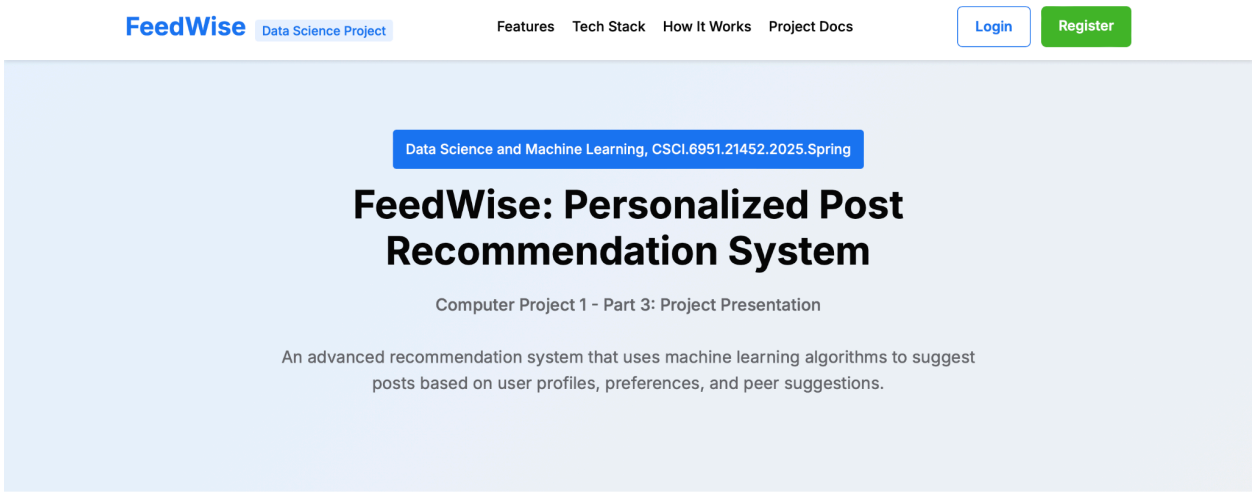
```
    user_interactions = list(interactions_collection.find({"user": user_id_obj}))

    # If no user profile, return empty list
    if not user_profile:
        return []

    # 2. Extract user skills and preferences
    skills = [normalize_tag(skill) for skill in user_profile.get('skills', [])]
    preferences = [normalize_tag(pref) for pref in user_profile.get('feedPreferences',
[])]

    # 3. Get posts the user has already viewed
    viewed_post_ids = {i['post'] for i in user_interactions if i['interactionType'] ==
'view'}

    # 4. Get all posts except user's own posts and those already viewed
    candidate_posts = list(posts_collection.find({
        "user": {"$ne": user_id_obj},
        "_id": {"$nin": list(viewed_post_ids)}
    }))

    # If no candidate posts, return empty list
    if not candidate_posts:
        return []

    # 5. Calculate direct tag matches for each post
    post_scores = []
    for post in candidate_posts:
        post_id = post['_id']
        post_tags = [normalize_tag(tag) for tag in post.get('tags', [])]

        # Count matches with skills and preferences
        skill_matches = sum(1 for skill in skills if any(skill in tag or tag in skill
for tag in post_tags))
        pref_matches = sum(1 for pref in preferences if any(pref in tag or tag in pref
for tag in post_tags))

        # Calculate a direct matching score (higher is better)
        direct_score = (skill_matches * 5) + (pref_matches * 5)

        # Also check for interaction history (likes)
        interaction_score = 0
```

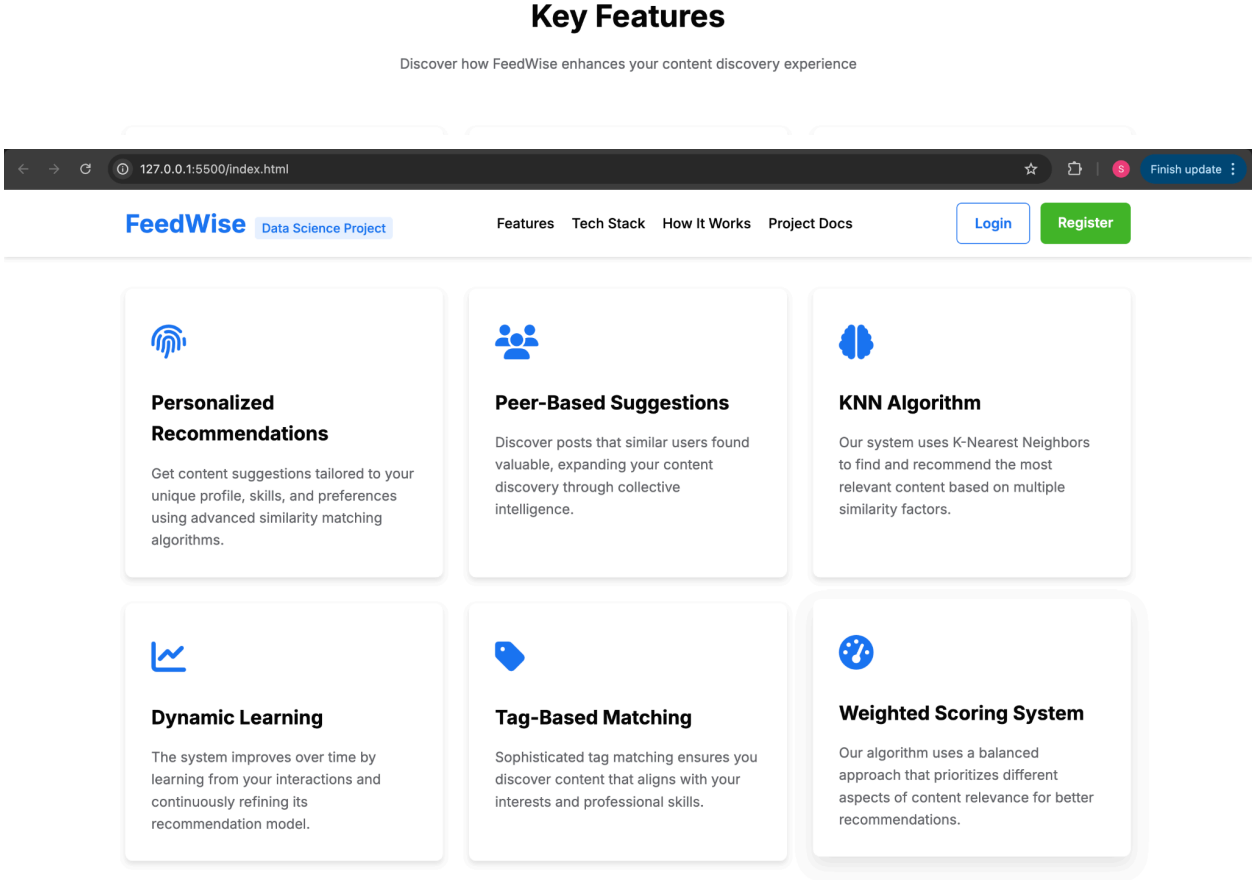
```
        for interaction in user_interactions:
            if interaction['post'] == post_id and interaction['interactionType'] ==
'like':
                # Get tags from this liked post to compare with current post
                liked_post = posts_collection.find_one({"_id": interaction['post']})
                if liked_post:
                    liked_tags = [normalize_tag(tag) for tag in liked_post.get('tags',
[])]
                    # Count matching tags between liked post and current post
                    tag_matches = sum(1 for tag in post_tags if tag in liked_tags)
                    interaction_score += tag_matches * 3

        # Combine scores
        total_score = direct_score + interaction_score

        post_scores.append({
            'id': post_id,
            'score': total_score,
            'skill_matches': skill_matches,
            'pref_matches': pref_matches,
            'tags': post.get('tags', []),
            'title': post.get('title', '')
        })

    # 6. Sort posts by score (highest first)
    post_scores.sort(key=lambda x: x['score'], reverse=True)

    # 7. Ensure we have a balanced mix of recommendations
    final_recommendations = []
    skill_related = []
    pref_related = []

    # Separate posts into skill-related and preference-related
    for post in post_scores:
        if post['skill_matches'] > 0:
            skill_related.append(post)
        if post['pref_matches'] > 0:
            pref_related.append(post)

    # If user has both skills and preferences, ensure both are represented
    if skills and preferences and skill_related and pref_related:
        # Fill half with skill-related, half with preference-related
```

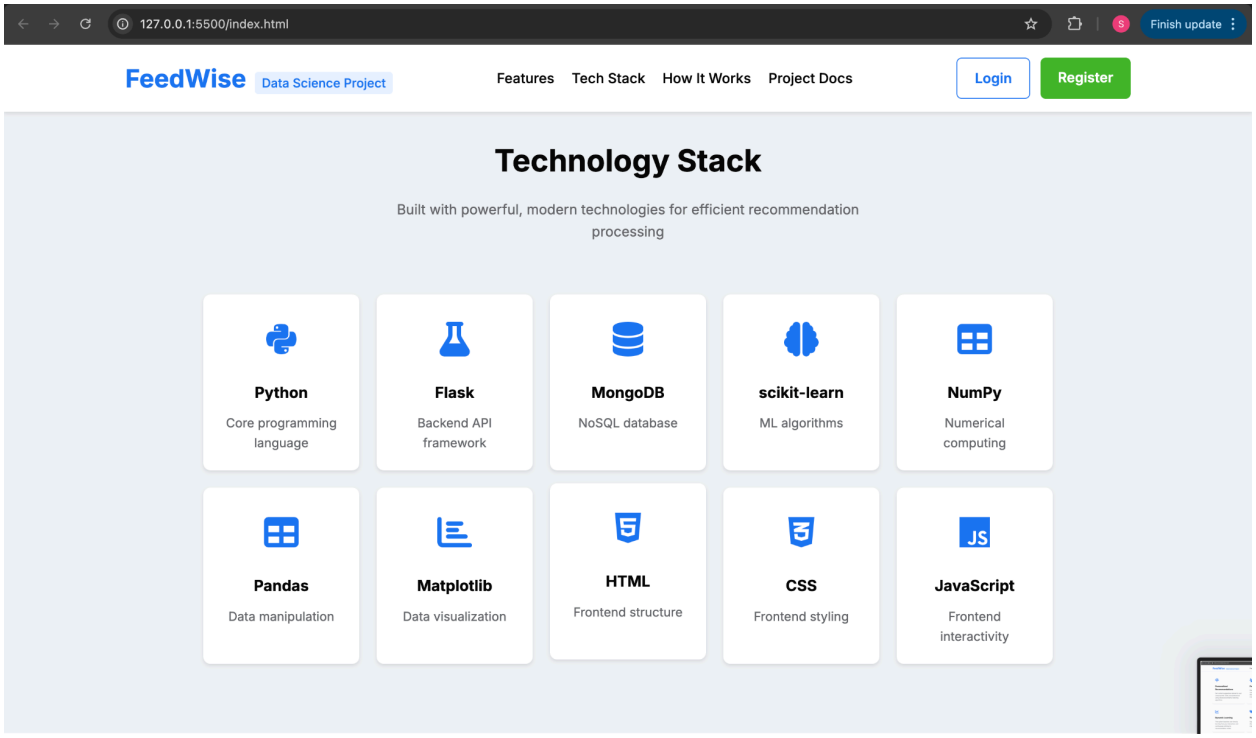
```
        skill_count = min(limit // 2, len(skill_related))
        pref_count = min(limit - skill_count, len(pref_related))

        # Add skill-related posts
        for i in range(skill_count):
            if skill_related[i]['id'] not in [p['id'] for p in final_recommendations]:
                final_recommendations.append(skill_related[i])

        # Add preference-related posts
        for i in range(pref_count):
            if pref_related[i]['id'] not in [p['id'] for p in final_recommendations]:
                final_recommendations.append(pref_related[i])

    # If we still need more recommendations, add from the remaining top-scored posts
    remaining_count = limit - len(final_recommendations)
    if remaining_count > 0:
        for post in post_scores:
            if post['id'] not in [p['id'] for p in final_recommendations]:
                final_recommendations.append(post)
                remaining_count -= 1
                if remaining_count == 0:
                    break

    # 8. Sort final recommendations by score again
    final_recommendations.sort(key=lambda x: x['score'], reverse=True)

    # 9. Extract post IDs for return
    recommended_post_ids = [str(post['id']) for post in final_recommendations[:limit]]

    # Print debug info
    print(f"\nUser skills: {user_profile.get('skills', [])}")
    print(f"User preferences: {user_profile.get('feedPreferences', [])}")
    print("\nRecommended posts:")
    for post in final_recommendations[:limit]:
        print(f"- '{post['title']}' (Tags: {post['tags']}, Score: {post['score']})")
        print(f"  Skill matches: {post['skill_matches']}, Preference matches:
{post['pref_matches']}")

    return recommended_post_ids
```

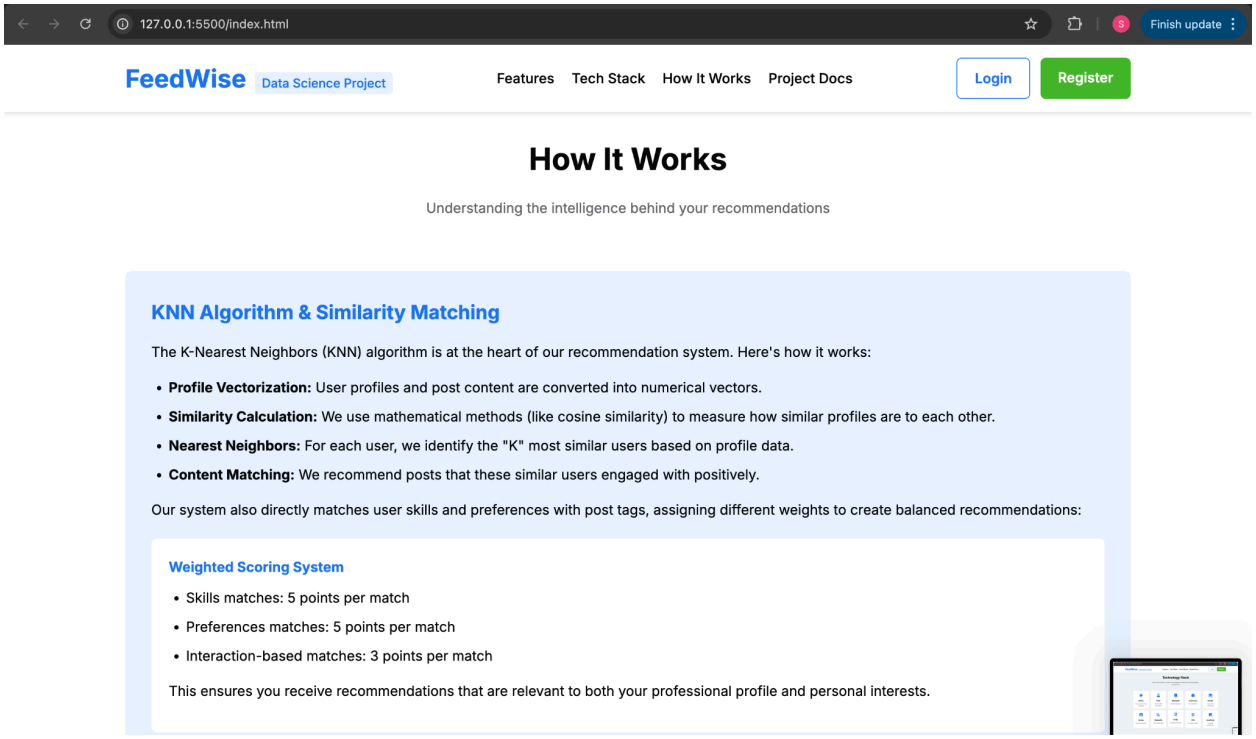
**Clilent UI Snapshots:**

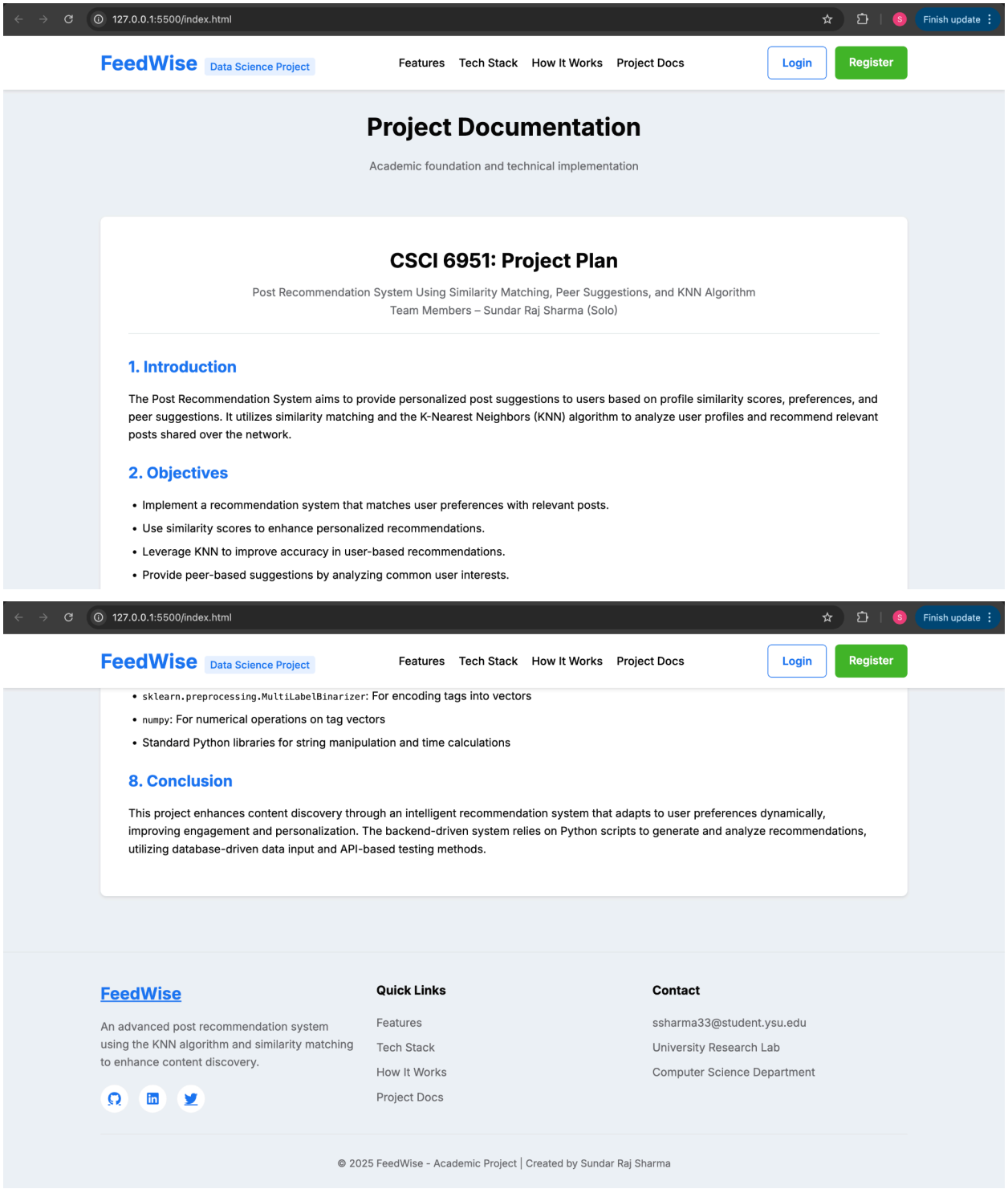
Hosted Link: https://mcisysusrs25.github.io/post-recom-system/

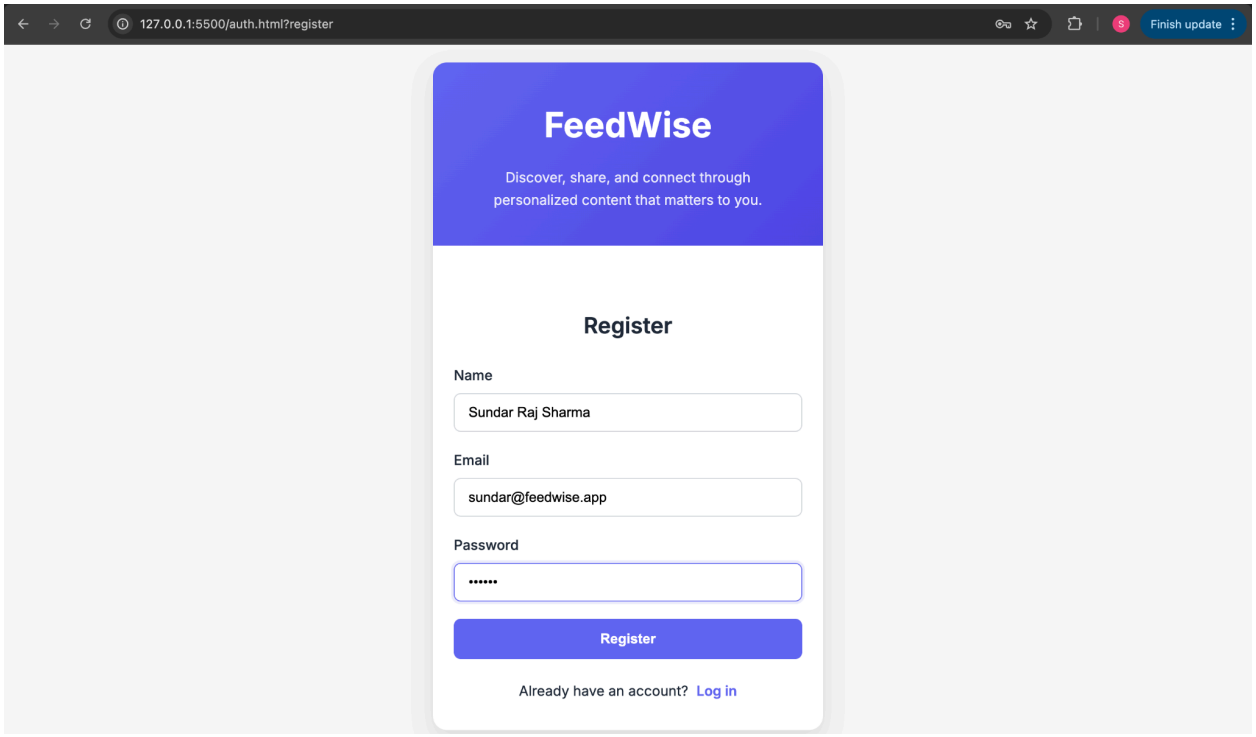


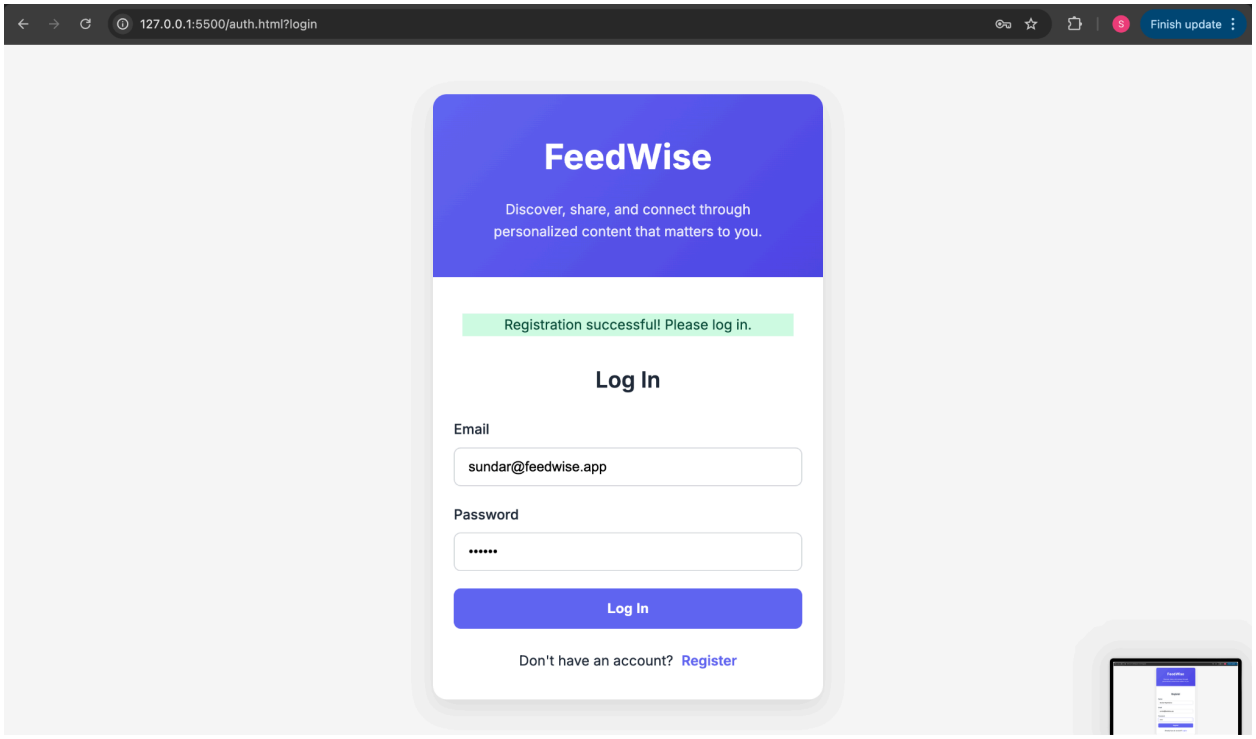



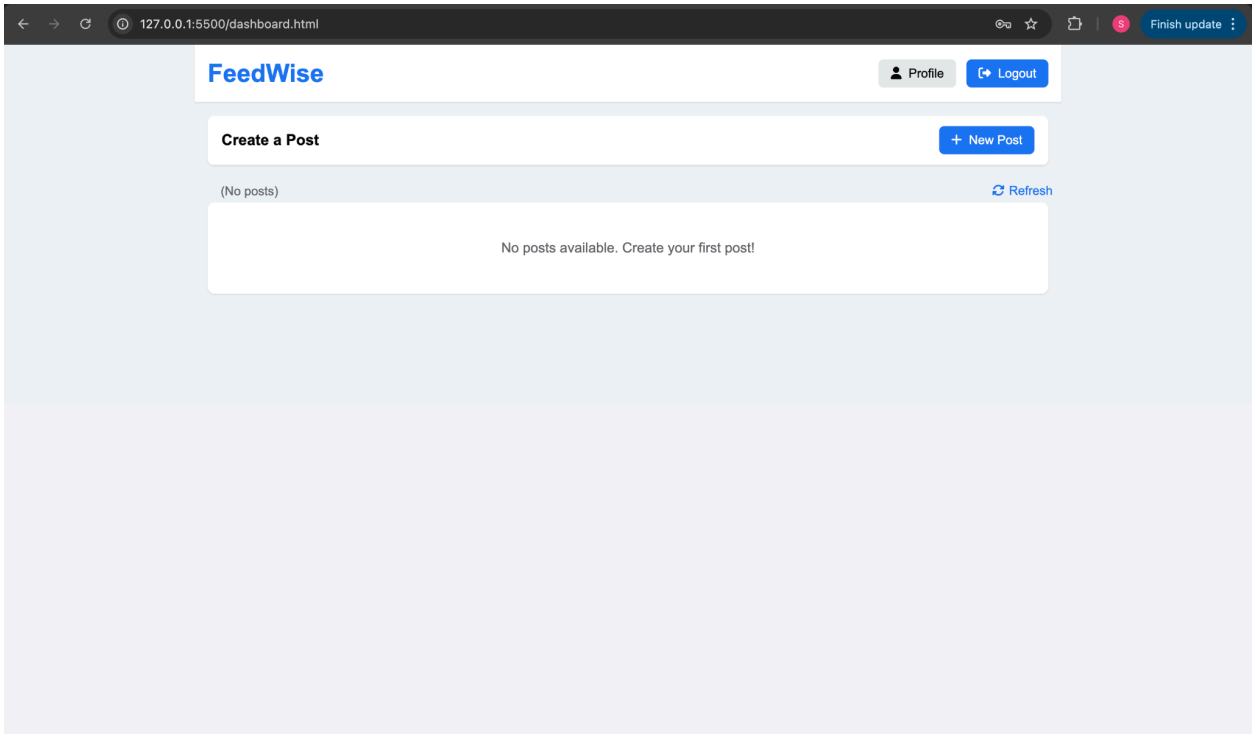

127.0.0.1:5500/index.html
Finish update
FeedWise
Data Science Project
Features
Tech Stack
How It Works
Project Docs
Login
Register
Technology Stack
Built with powerful, modern technologies for efficient recommendation processing
Python
Core programming language
Flask
Backend API framework
MongoDB
NoSQL database
scikit-learn
ML algorithms
NumPy
Numerical computing
Pandas
Data manipulation
Matplotlib
Data visualization
HTML
Frontend structure
CSS
Frontend styling
JavaScript
Frontend interactivity

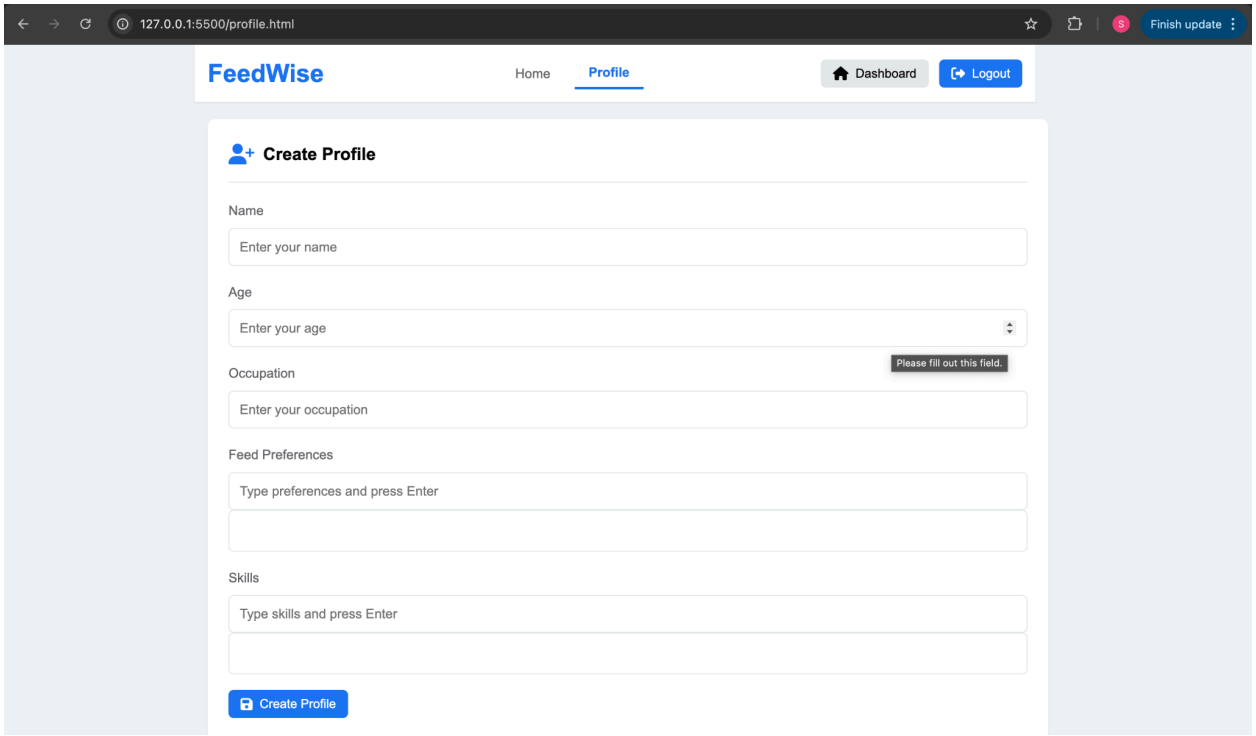

127.0.0.1:5500/index.html
Finish update
FeedWise
Data Science Project
Features
Tech Stack
How It Works
Project Docs
Login
Register
How It Works
Understanding the intelligence behind your recommendations
KNN Algorithm & Similarity Matching
The K-Nearest Neighbors (KNN) algorithm is at the heart of our recommendation system. Here's how it works:
Profile Vectorization: User profiles and post content are converted into numerical vectors.
Similarity Calculation: We use mathematical methods (like cosine similarity) to measure how similar profiles are to each other.
Nearest Neighbors: For each user, we identify the "K" most similar users based on profile data.
Content Matching: We recommend posts that these similar users engaged with positively.
Our system also directly matches user skills and preferences with post tags, assigning different weights to create balanced recommendations:
Weighted Scoring System
Skills matches: 5 points per match
Preferences matches: 5 points per match
Interaction-based matches: 3 points per match
This ensures you receive recommendations that are relevant to both your professional profile and personal interests.

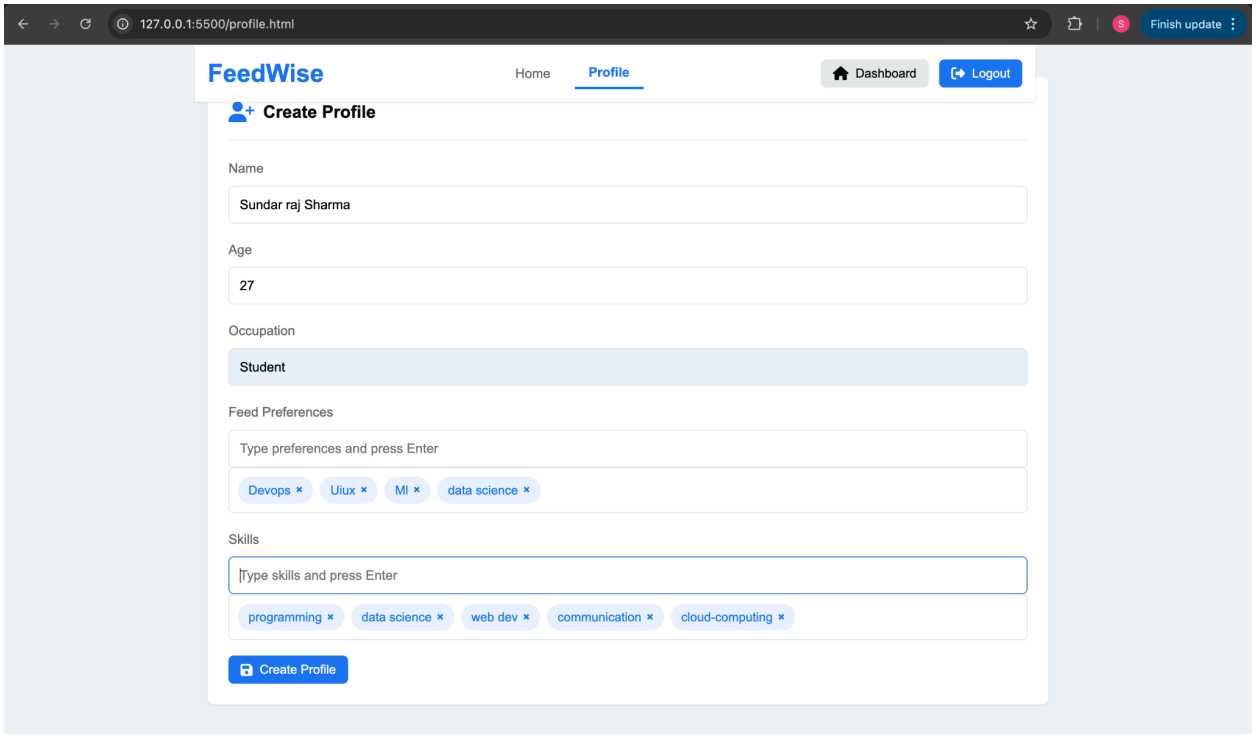

127.0.0.1:5500/index.html
Finish update
FeedWise
Data Science Project
Features
Tech Stack
How It Works
Project Docs
Login
Register
Project Documentation
Academic foundation and technical implementation
CSCI 6951: Project Plan
Post Recommendation System Using Similarity Matching, Peer Suggestions, and KNN Algorithm
Team Members – Sundar Raj Sharma (Solo)
1. Introduction
The Post Recommendation System aims to provide personalized post suggestions to users based on profile similarity scores, preferences, and peer suggestions. It utilizes similarity matching and the K-Nearest Neighbors (KNN) algorithm to analyze user profiles and recommend relevant posts shared over the network.
2. Objectives
• Implement a recommendation system that matches user preferences with relevant posts.
• Use similarity scores to enhance personalized recommendations.
• Leverage KNN to improve accuracy in user-based recommendations.
• Provide peer-based suggestions by analyzing common user interests.
127.0.0.1:5500/index.html
Finish update
FeedWise
Data Science Project
Features
Tech Stack
How It Works
Project Docs
Login
Register
• sklearn.preprocessing.MultiLabelBinarizer: For encoding tags into vectors
• numpy: For numerical operations on tag vectors
• Standard Python libraries for string manipulation and time calculations
8. Conclusion
This project enhances content discovery through an intelligent recommendation system that adapts to user preferences dynamically, improving engagement and personalization. The backend-driven system relies on Python scripts to generate and analyze recommendations, utilizing database-driven data input and API-based testing methods.
FeedWise
An advanced post recommendation system using the KNN algorithm and similarity matching to enhance content discovery.
Quick Links
Features
Tech Stack
How It Works
Project Docs
Contact
ssharma33@student.ysu.edu
University Research Lab
Computer Science Department
© 2025 FeedWise - Academic Project | Created by Sundar Raj Sharma

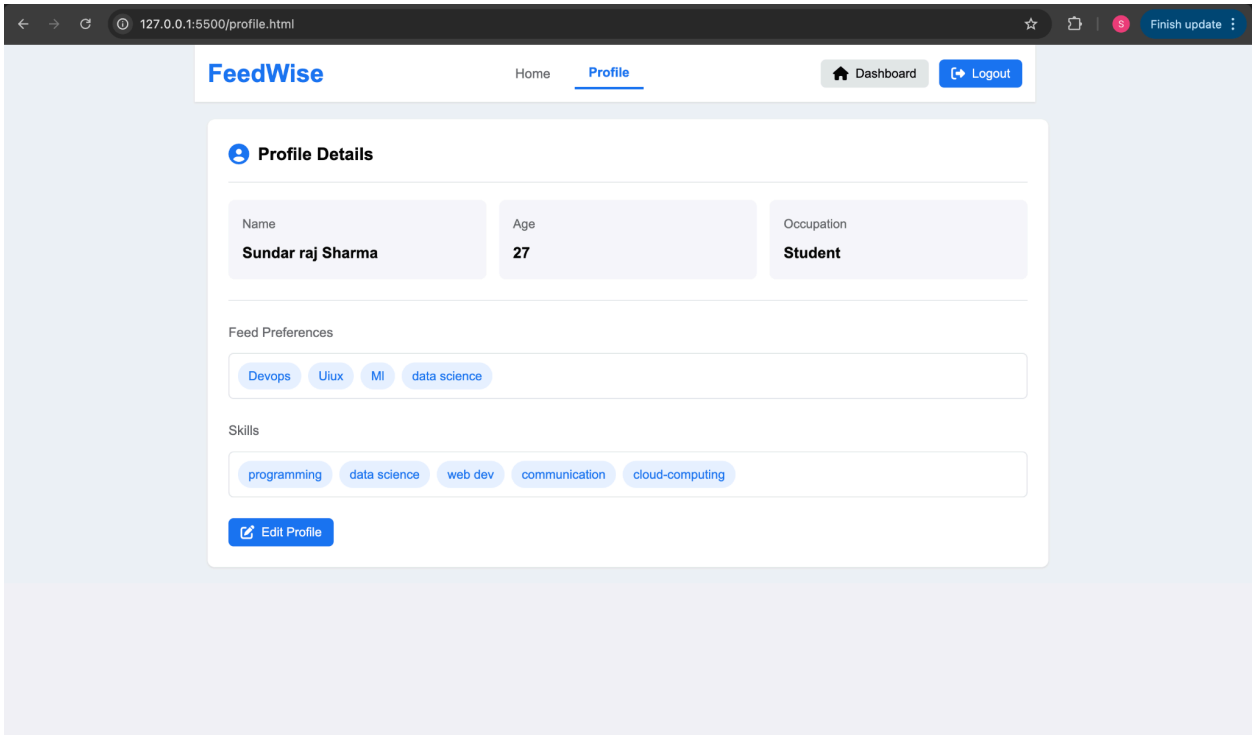

127.0.0.1:5500/auth.html?register
Finish update
FeedWise
Discover, share, and connect through personalized content that matters to you.
Register
Name
Sundar Raj Sharma
Email
sundar@feedwise.app
Password
Register
Already have an account? Log in

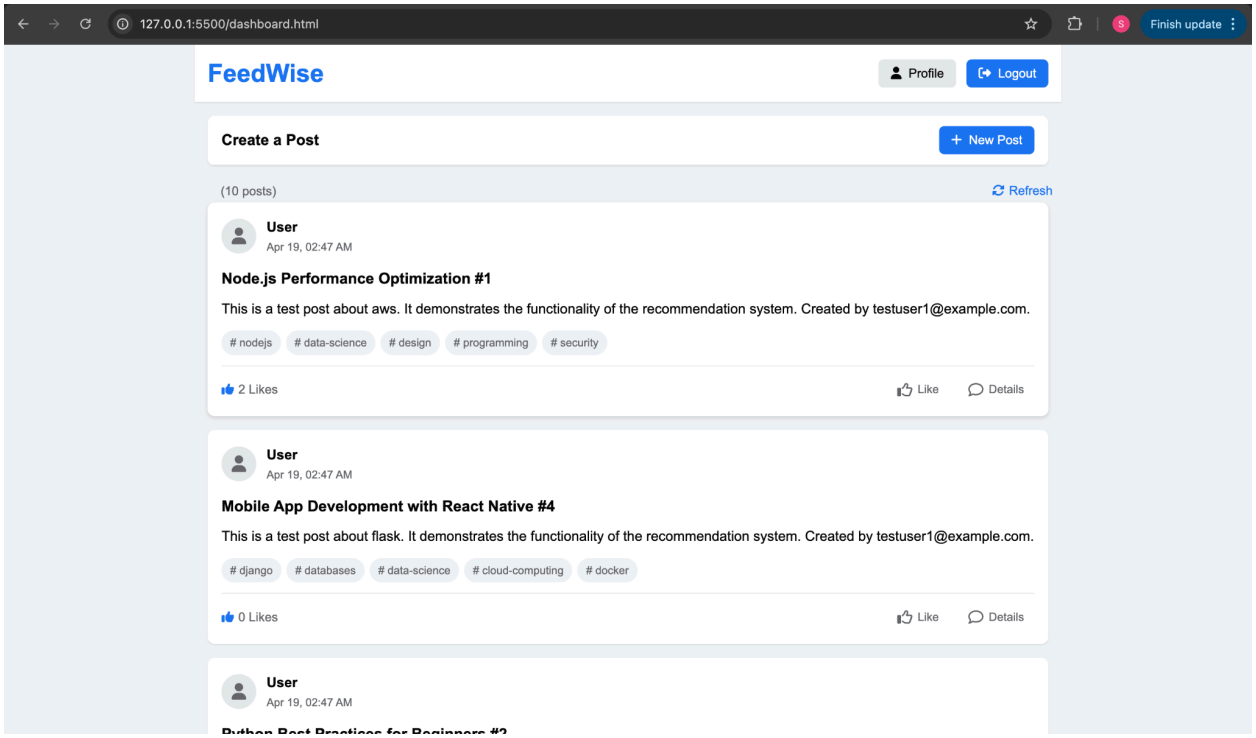

127.0.0.1:5500/auth.html?login
Finish update
FeedWise
Discover, share, and connect through personalized content that matters to you.
Registration successful! Please log in.
Log In
Email
sundar@feedwise.app
Password
Log In
Don't have an account? Register

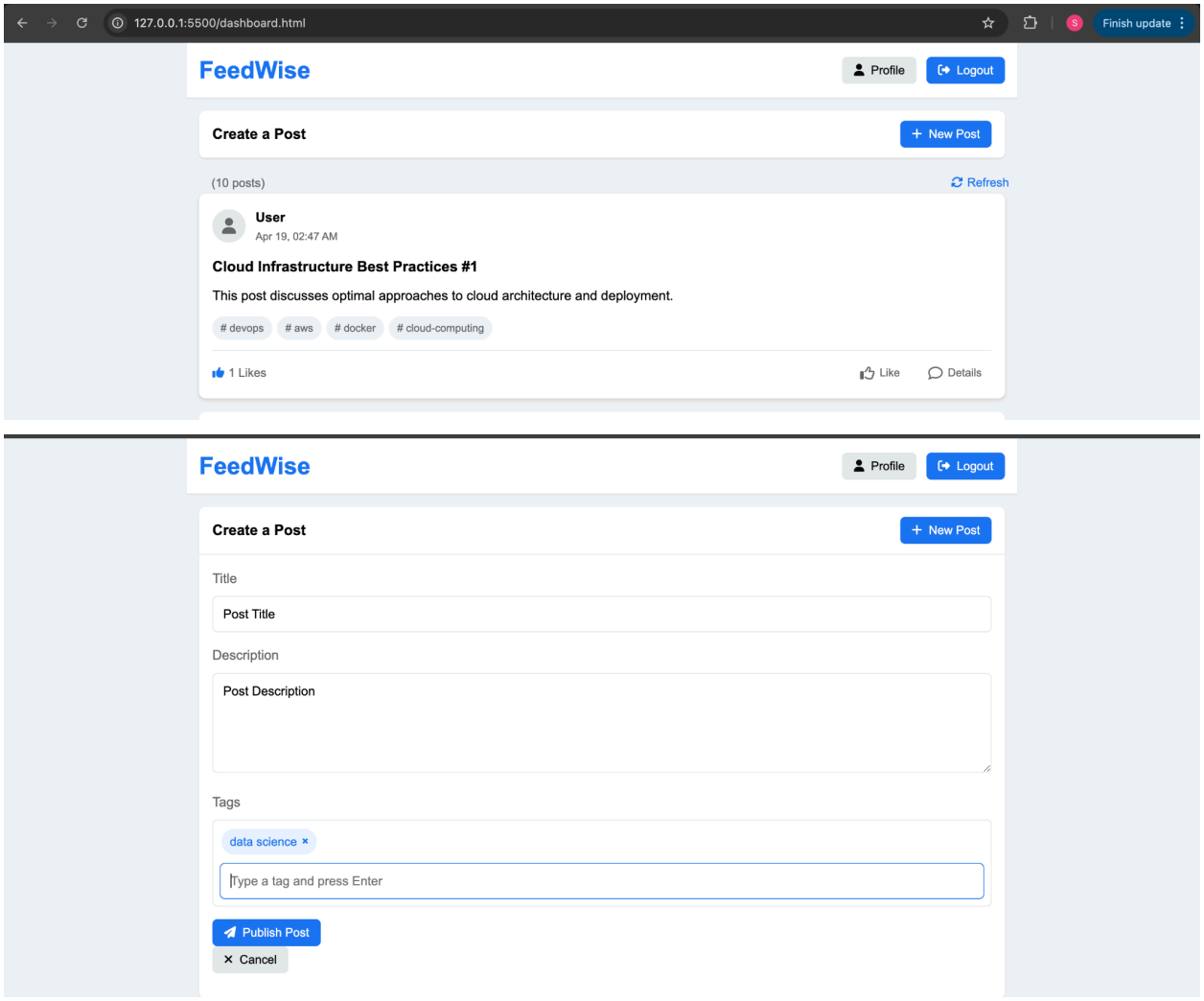

127.0.0.1:5500/dashboard.html
Finish update
FeedWise
Profile
Logout
Create a Post
New Post
(No posts)
Refresh
No posts available. Create your first post!


127.0.0.1:5500/profile.html
Finish update
FeedWise
Home
Profile
Dashboard
Logout
Create Profile
Name
Enter your name
Age
Enter your age
Please fill out this field.
Occupation
Enter your occupation
Feed Preferences
Type preferences and press Enter
Skills
Type skills and press Enter
Create Profile


127.0.0.1:5500/profile.html
Finish update
FeedWise
Home
Profile
Dashboard
Logout
Create Profile
Name
Sundar raj Sharma
Age
27
Occupation
Student
Feed Preferences
Type preferences and press Enter
Devops
Uiux
MI
data science
Skills
Type skills and press Enter
programming
data science
web dev
communication
cloud-computing
Create Profile


127.0.0.1:5500/profile.html
Finish update
FeedWise
Home
Profile
Dashboard
Logout
Profile Details
Name
Sundar raj Sharma
Age
27
Occupation
Student
Feed Preferences
Devops
Uiux
MI
data science
Skills
programming
data science
web dev
communication
cloud-computing
Edit Profile


127.0.0.1:5500/dashboard.html
Finish update
FeedWise
Profile
Logout
Create a Post
+ New Post
(10 posts)
Refresh
User
Apr 19, 02:47 AM
Node.js Performance Optimization #1
This is a test post about aws. It demonstrates the functionality of the recommendation system. Created by testuser1@example.com.
# nodejs
# data-science
# design
# programming
# security
2 Likes
Like
Details
User
Apr 19, 02:47 AM
Mobile App Development with React Native #4
This is a test post about flask. It demonstrates the functionality of the recommendation system. Created by testuser1@example.com.
# django
# databases
# data-science
# cloud-computing
# docker
0 Likes
Like
Details
User
Apr 19, 02:47 AM


127.0.0.1:5500/dashboard.html
Finish update
FeedWise
Profile
Logout
Create a Post
New Post
(10 posts)
Refresh
User
Apr 19, 02:47 AM
Cloud Infrastructure Best Practices #1
This post discusses optimal approaches to cloud architecture and deployment.
# devops
# aws
# docker
# cloud-computing
1 Likes
Like
Details
FeedWise
Profile
Logout
Create a Post
New Post
Title
Post Title
Description
Post Description
Tags
data science ×
Type a tag and press Enter
Publish Post
× Cancel