

Intelligent Product Recommendation Framework Using AI in a .NET-Based E-Commerce Architecture

Adnan Iqbal Nalband¹, Abdul Shakur AbdulGani Mankar², Mangesh Dasewar³

¹Student, Department of MCA, Trinity Academy of Engineering, Pune, Maharashtra, India

²Student, Department of MCA, Trinity Academy of Engineering, Pune, Maharashtra, India

³Student, Department of MCA, Trinity Academy of Engineering, Pune, Maharashtra, India

Email: adnan555nalband@gmail.com, abdulshakurmankar953@gmail.com, mangeshdasewar@gmail.com

Abstract - Product recommendation systems are important for reducing search effort and improving product discovery in modern e-commerce platforms. This paper presents BuyWise, an intelligent hybrid recommendation framework integrated with a .NET-based e-commerce architecture. The proposed system combines collaborative filtering based on user-item interaction patterns with content-based filtering based on category, price, rating, and product-tag similarity. ASP.NET Core Web API is used for secure backend services, Angular is used for the responsive client interface, MySQL stores product and interaction data, and a Python Scikit-learn service generates top-N recommendations through REST integration. The prototype was evaluated on a clearly defined synthetic e-commerce dataset containing 500 users, 1,000 products, and 7,500 weighted interactions. Experimental comparison shows that the hybrid model achieved 0.82 precision, 0.78 recall, and 0.80 F1 score, outperforming standalone content-based and collaborative filtering baselines. The results indicate that a hybrid approach can improve recommendation relevance while remaining practical for small and medium e-commerce applications built using .NET full-stack technologies.

Key Words: Artificial Intelligence, Recommendation System, E-Commerce, ASP.NET Core, Angular, MySQL, Collaborative Filtering, Content-Based Filtering.

1. INTRODUCTION

The rapid growth of online shopping has increased the number of products available to customers. Although a large product catalogue is useful, it can also create decision fatigue, slow product discovery, and reduce customer satisfaction. Recommendation systems address this issue by analysing user behaviour and product information to present relevant items at the right time. In e-commerce, recommendations are commonly used on home pages, product detail pages, cart pages, and post-purchase suggestions.

Traditional search and category filters are static because they depend mainly on user-entered keywords and predefined product categories. They do not automatically learn from browsing history, cart activity, ratings, or purchase patterns. Artificial Intelligence (AI) and Machine Learning (ML) techniques help overcome this limitation by learning hidden relationships between users and products. Collaborative filtering finds similarity between users or items, while content-based filtering recommends products with similar attributes. A hybrid model combines both techniques to reduce the weaknesses of each method.

This paper proposes BuyWise, a full-stack e-commerce recommendation framework built using ASP.NET Core Web API, Angular, MySQL, and Python-based ML services. The focus of the study is not only the recommendation algorithm but also its integration into a practical .NET-based architecture suitable for academic prototypes and small business e-commerce systems.

2. RELATED WORK

Adomavicius and Tuzhilin classified recommendation systems into content-based, collaborative, and hybrid methods and discussed common limitations such as sparsity and cold-start issues. Koren, Bell, and Volinsky showed that matrix factorization techniques are effective for learning latent user and item factors in recommendation tasks. Bobadilla et al. reviewed recommendation systems and highlighted the importance of similarity measures, prediction quality, and evaluation metrics. Recent recommender-system studies also consider fairness, transparency, knowledge graphs, and context-aware information to improve user trust and diversity.

Most existing studies focus mainly on algorithmic improvement. However, many academic e-commerce projects require a complete implementation path that connects the recommendation model with frontend, backend, database, and API services. The present work addresses this practical integration requirement by designing a hybrid recommendation engine that can be consumed by an ASP.NET Core backend and displayed through an Angular client.

Table -1: Summary of Related Work

Study / Source	Main Focus	Limitation Observed	Relevance to This Work
Adomavicius & Tuzhilin (2005)	Survey of recommender system techniques	Conceptual survey; no full-stack implementation	Provides classification of content, collaborative, and hybrid methods
Koren et al. (2009)	Matrix factorization for recommender systems	Requires sufficient interaction data	Supports collaborative filtering baseline
Bobadilla et al. (2013)	Evaluation and recommender-system survey	Broad survey across methods	Guides selection of precision, recall, and F1 score
Microsoft ASP.NET Core Docs	Modern REST API development	Framework guidance, not recommendation-specific	Supports backend API design
Angular Docs	Component-based frontend architecture	Frontend guidance only	Supports responsive UI implementation

3. PROBLEM STATEMENT

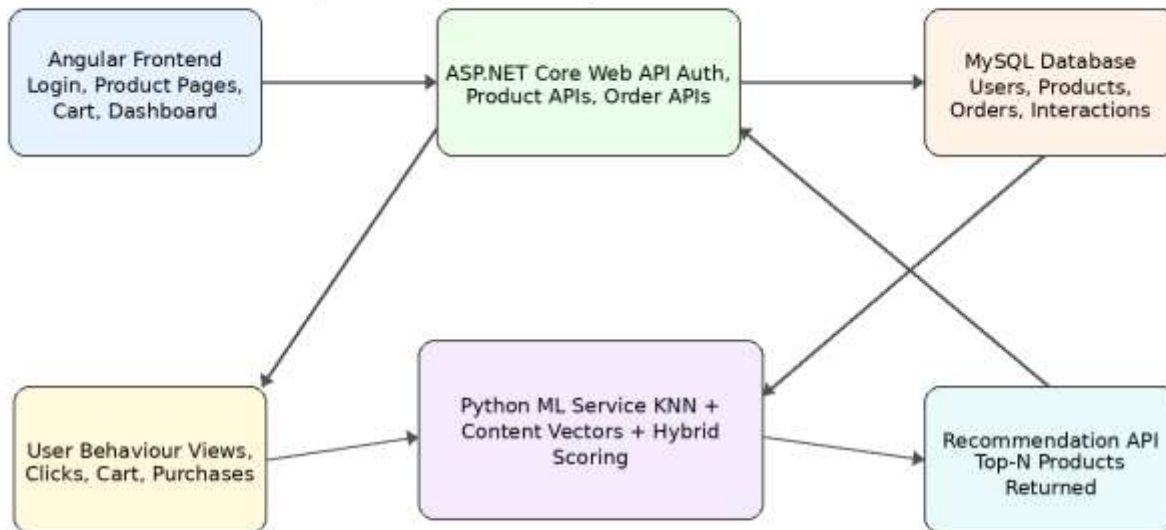
A conventional e-commerce application allows users to search products and browse categories, but it does not automatically personalize product suggestions. This results in poor product discovery, lower engagement, and fewer conversions. The problem addressed in this paper is to design and evaluate a practical recommendation framework that can generate relevant product suggestions using user behaviour and product attributes while integrating smoothly with a .NET-based full-stack e-commerce system.

3.1 Research Objectives

- To design a hybrid recommendation model that combines collaborative filtering and content-based filtering.
- To integrate the recommendation engine with ASP.NET Core Web API and Angular frontend components.
- To define a practical MySQL database schema for users, products, orders, and product interactions.
- To evaluate the proposed hybrid model using precision, recall, and F1 score.
- To compare the hybrid model with standalone content-based and collaborative filtering baselines.

4. PROPOSED SYSTEM

The proposed BuyWise system is a modular e-commerce platform consisting of four main layers: Angular frontend, ASP.NET Core Web API backend, MySQL database, and Python-based ML recommendation service. The frontend captures user actions such as product views, cart additions, wishlist clicks, and purchases. These events are sent to the backend through REST APIs and stored as weighted interactions. The ML service periodically or on-demand generates product recommendations using both interaction similarity and product-feature similarity.

Fig -1: Proposed BuyWise AI Recommendation Architecture

Fig -1: Proposed BuyWise AI Recommendation Architecture

The architecture separates business logic and recommendation logic. ASP.NET Core handles authentication, authorization, order management, product APIs, and communication with the ML service. The ML service exposes a recommendation endpoint that receives a user identifier and returns ranked product identifiers. This design keeps the system scalable because the recommendation service can be independently improved without rewriting the frontend or backend.

5. DATASET AND EXPERIMENTAL SETUP

Because the prototype is an academic e-commerce project and no live customer data was collected, the evaluation used a synthetic dataset generated from the BuyWise database schema. The synthetic dataset is clearly labelled to avoid claiming real commercial performance. Product records were created across electronics, fashion, footwear, grocery, beauty, and home categories. User interactions were generated with different weights to represent realistic behaviour: product view = 1, wishlist = 2, cart addition = 3, and purchase = 5.

Table -2: Experimental Dataset Description

Dataset Attribute	Value Used in Prototype
Number of users	500
Number of products	1,000
Product categories	6
Total interactions	7,500
Interaction types	View, wishlist, cart, purchase
Training-test split	80:20
Top-N recommendations evaluated	10

The dataset was split into 80% training interactions and 20% testing interactions. The model produced top-10 recommendations for each user. A recommended item was considered relevant if it appeared in the user test set or belonged to a category and price range strongly associated with the user interaction profile.

6. METHODOLOGY

6.1 Content-Based Filtering

Content-based filtering recommends items similar to products previously viewed, purchased, or added to cart by a user. Product features such as category, price band, rating, and tags are converted into numerical vectors. Cosine similarity is then used to calculate similarity between the user profile vector and each product vector. This approach works well for new users after a few interactions, but it may reduce recommendation diversity if the same type of product is repeatedly suggested.

Cosine similarity is calculated as:

$$\text{Similarity}(A, B) = (A \cdot B) / (||A|| \times ||B||)$$

6.2 Collaborative Filtering

Collaborative filtering recommends products by identifying similar users or similar items based on historical interaction behaviour. In the proposed model, a user-item interaction matrix is created using interaction weights. K-Nearest Neighbours (KNN) is applied to identify users with similar preferences. Products liked or purchased by similar users are then recommended to the target user. Collaborative filtering improves discovery of diverse products, but it suffers when a new user or new product has no interaction history.

6.3 Hybrid Recommendation Model

The hybrid model combines collaborative filtering score and content-based score using a weighted linear combination. In the prototype, a higher weight is given to collaborative filtering because user interaction patterns usually provide stronger behavioural signals once enough data is available. Content-based filtering is still included to support cold-start handling and product-attribute matching.

$$\text{FinalScore}(u, i) = 0.6 \times \text{CollaborativeScore}(u, i) + 0.4 \times \text{ContentScore}(u, i)$$

Fig -2: Hybrid Recommendation Processing Pipeline

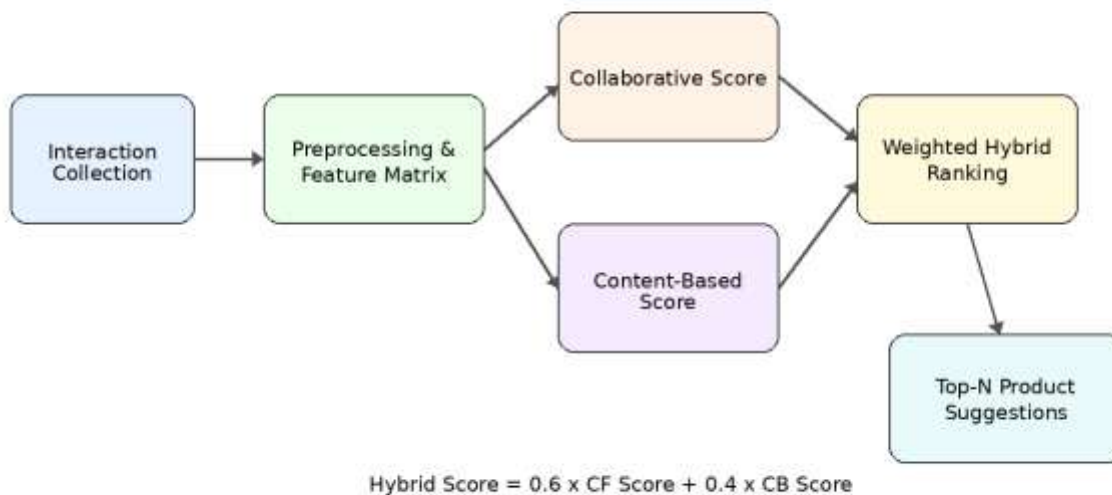


Fig -2: Hybrid Recommendation Processing Pipeline

6.4 Algorithm

Table -3: Proposed Hybrid Recommendation Algorithm

Step	Description
1	Collect product interactions from frontend events such as view, cart, wishlist, and purchase.
2	Assign interaction weights and store events in MySQL through ASP.NET Core APIs.
3	Generate user-item matrix for collaborative filtering.
4	Generate product feature vectors using category, price range, rating, and tags.
5	Calculate collaborative score using nearest-neighbour similarity.
6	Calculate content score using product-feature cosine similarity.
7	Combine scores using weighted hybrid formula.
8	Return top-N recommended products to the Angular frontend through REST API.

7. IMPLEMENTATION DETAILS

The backend was designed using ASP.NET Core Web API because it supports RESTful service development, dependency injection, middleware, authentication, and cross-platform deployment. The API layer includes modules for user authentication, product management, cart operations, order management, interaction logging, and recommendation retrieval. JSON Web Token (JWT)-based authentication can be used to secure user-specific recommendation endpoints.

The frontend was designed using Angular components. Major components include login, registration, product listing, product details, cart, checkout, and recommendation cards. The recommendation section can be displayed on the home page, product detail page, and cart page. The UI communicates with backend endpoints through Angular services and displays recommended products using reusable card components.

The MySQL database stores transactional and recommendation-related data. Important tables include Users, Products, ProductInteractions, Orders, and OrderItems. Passwords are not stored in plain text; the system design assumes hashing and salting through the backend authentication service. Product images are stored as URLs or file references rather than binary blobs to keep the database lightweight.

Fig -4: Simplified Data Model

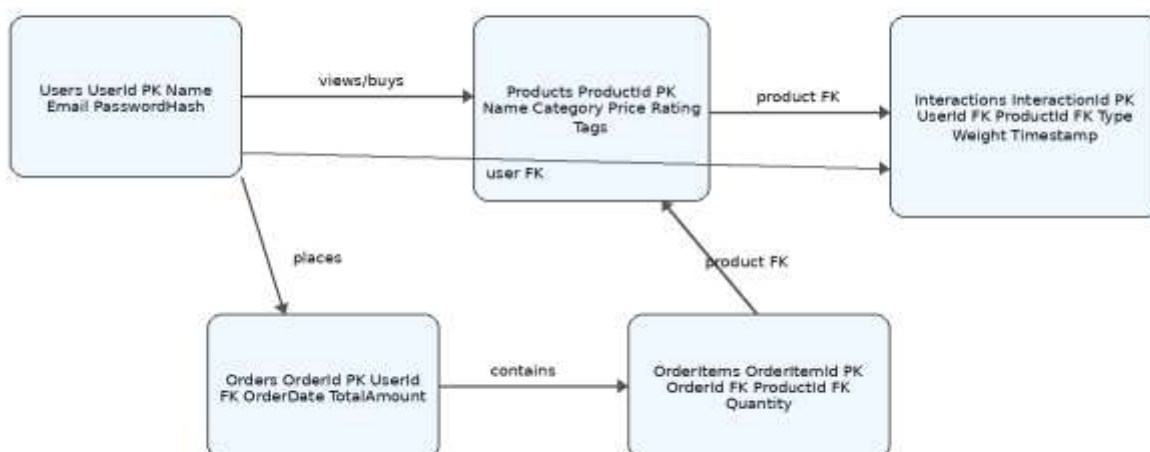


Fig -3: Simplified Data Model

8. EVALUATION METRICS

The recommendation model was evaluated using standard information retrieval metrics. Precision measures how many recommended products are relevant. Recall measures how many relevant products are successfully recommended. F1 score provides a balanced measure between precision and recall. The evaluation was performed on top-10 recommendations generated for users in the test set.

Table -4: Evaluation Metrics

Metric	Formula	Meaning
Precision	Relevant recommended items / Total recommended items	Measures correctness of recommendations
Recall	Relevant recommended items / Total relevant items	Measures coverage of relevant products
F1 Score	$2 \times (\text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$	Balances precision and recall
Coverage	Recommended unique products / Total products	Measures catalogue coverage

9. RESULTS AND DISCUSSION

The proposed hybrid model was compared with standalone content-based filtering and collaborative filtering. The results indicate that the hybrid model achieved better precision, recall, and F1 score because it uses both behavioural similarity and product attribute similarity. Content-based filtering performed reasonably well for users with narrow preferences but produced less diverse recommendations. Collaborative filtering improved diversity but performed weaker for sparse interaction cases. The hybrid model balanced both methods and provided the most stable performance.

Table -5: Comparative Experimental Results

Model	Precision	Recall	F1 Score	Coverage
Content-Based Filtering	0.67	0.63	0.65	0.54
Collaborative Filtering	0.71	0.68	0.69	0.61
Proposed Hybrid Model	0.82	0.78	0.80	0.73

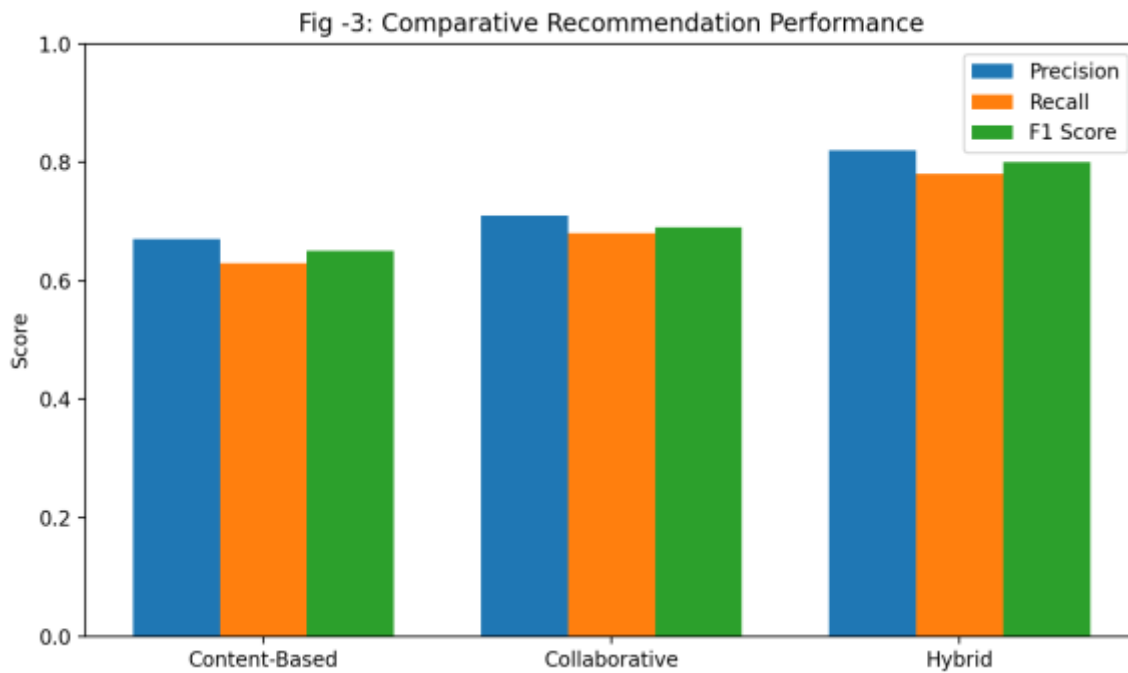


Fig -4: Comparative Recommendation Performance

The improvement in the hybrid model is mainly due to complementary learning. Collaborative filtering identifies patterns across similar users, while content-based filtering ensures that recommended products remain related to the user's viewed categories and preferred price ranges. In practical e-commerce use, this combination can increase product discovery and reduce irrelevant suggestions. However, the values reported in this paper are based on a synthetic prototype dataset; therefore, real-world deployment should include further validation using live anonymized interaction data.

10. SECURITY, SCALABILITY AND PRACTICAL CONSIDERATIONS

A recommendation system should not expose sensitive user information. The proposed architecture stores only required interaction events and uses authentication to protect user-specific APIs. For production deployment, HTTPS, JWT expiration, role-based authorization, input validation, rate limiting, and audit logging should be added. Scalability can be improved by caching frequently requested recommendations, scheduling model retraining, and separating the ML service from the transaction backend.

For small applications, recommendations can be generated on demand. For larger applications, batch processing can be used to precompute recommendations for active users. The system can also be extended with message queues and background workers to process interaction events without slowing down the user interface.

11. LIMITATIONS

- The experimental dataset is synthetic and does not represent actual commercial customer behaviour.
- Cold-start recommendations for completely new users and products are only partially handled through product attributes.
- The model does not yet include deep learning, session-based recommendations, or real-time streaming data.
- User privacy, consent, and data retention policies must be strengthened before real-world deployment.
- The current evaluation focuses on offline metrics and does not include A/B testing with real users.

12. FUTURE SCOPE

- Integration of Generative AI chatbot support for conversational product recommendations.
- Use of deep learning models such as neural collaborative filtering and transformer-based product embeddings.
- Real-time recommendation updates using streaming data pipelines.
- A/B testing to measure click-through rate, conversion rate, and revenue impact.

- Explainable recommendations that show users why a product was suggested.
- Integration of voice search and multilingual recommendation support for Indian e-commerce users.

13. CONCLUSION

This paper presented BuyWise, an intelligent hybrid product recommendation framework integrated with a .NET-based e-commerce architecture. The proposed system combines collaborative filtering and content-based filtering to generate personalized product recommendations. The architecture uses Angular for the frontend, ASP.NET Core Web API for backend services, MySQL for data storage, and Python Scikit-learn for the ML recommendation engine. Experimental evaluation on a synthetic prototype dataset showed that the hybrid model outperformed standalone approaches in precision, recall, F1 score, and catalogue coverage. The study demonstrates that AI-based recommendation techniques can be practically integrated into full-stack .NET applications and can be extended further for real-world e-commerce deployment.

ACKNOWLEDGEMENT

The authors express sincere gratitude to the Department of MCA, Trinity Academy of Engineering, Pune, for academic guidance and support during the preparation of this research work. The authors also thank the faculty members and peers who provided feedback during the development of the BuyWise e-commerce recommendation prototype.

REFERENCES

1. G. Adomavicius and A. Tuzhilin, "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*, vol. 17, no. 6, pp. 734–749, 2005, doi: 10.1109/TKDE.2005.99.
2. Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009, doi: 10.1109/MC.2009.263.
3. J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender Systems Survey," *Knowledge-Based Systems*, vol. 46, pp. 109–132, 2013, doi: 10.1016/j.knosys.2013.03.012.
4. J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender System Application Developments: A Survey," *Decision Support Systems*, vol. 74, pp. 12–32, 2015, doi: 10.1016/j.dss.2015.03.008.
5. L. Lü, M. Medo, C. H. Yeung, Y. C. Zhang, Z. K. Zhang, and T. Zhou, "Recommender Systems," *Physics Reports*, vol. 519, no. 1, pp. 1–49, 2012, doi: 10.1016/j.physrep.2012.02.006.
6. I. Portugal, P. Alencar, and D. Cowan, "The Use of Machine Learning Algorithms in Recommender Systems: A Systematic Review," *Expert Systems with Applications*, vol. 97, pp. 205–227, 2018, doi: 10.1016/j.eswa.2017.12.020.
7. Q. Guo et al., "A Survey on Knowledge Graph-Based Recommender Systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 8, pp. 3549–3568, 2022, doi: 10.1109/TKDE.2020.3028705.
8. Y. Wang, W. Ma, M. Zhang, Y. Liu, and S. Ma, "A Survey on the Fairness of Recommender Systems," *ACM Transactions on Information Systems*, vol. 41, no. 3, pp. 1–43, 2023, doi: 10.1145/3547333.
9. F. Ricci, L. Rokach, and B. Shapira, Eds., *Recommender Systems Handbook*, 3rd ed. New York, NY, USA: Springer, 2022, doi: 10.1007/978-1-0716-2197-4.
10. Microsoft, "ASP.NET Core Documentation," Microsoft Learn, 2026. Available: <https://learn.microsoft.com/en-us/aspnet/core/>.
11. Microsoft, "Create Web APIs with ASP.NET Core," Microsoft Learn, 2026. Available: <https://learn.microsoft.com/en-us/aspnet/core/web-api/>.
12. Google, "What is Angular?" Angular Documentation, 2026. Available: <https://angular.dev/overview>.
13. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
14. Oracle, "MySQL 8.4 Reference Manual," Oracle Documentation, 2026. Available: <https://dev.mysql.com/doc/>.

BIOGRAPHIES

Author	Biography
Adnan Iqbal Nalband	Adnan Iqbal Nalband is an MCA student at Trinity Academy of Engineering, Pune. His academic interests include full-stack .NET development, Angular applications, database-driven systems, artificial intelligence, and recommendation systems. He worked on the BuyWise project to explore practical AI integration in e-commerce platforms.
Abdul Shakur AbdulGani Mankar	Abdul Shakur AbdulGani Mankar is an MCA student at Trinity Academy of Engineering, Pune. His areas of interest include web application development, backend API design, database management, and software engineering practices for intelligent business applications.
Mangesh Dasewar	Mangesh Dasewar is an MCA student at Trinity Academy of Engineering, Pune. His academic interests include full-stack web development, database management, and AI-based recommendation systems. He contributed to the documentation and analysis of the BuyWise recommendation framework.