Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

Mooncart: An Interactive Full-Stack E-Commerce Framework

[1]Maleeha Fatima, [2]Pawan Kumar Jaiswal
[1]Student, [2]Assistant Professor
[1,2] Amity University Raipur, Chhattisgarh
[1]maleehafatima228@gmail.com, [2]pkumar@rpr.amity.edu

ABSTRACT:
In today's digital marketplace, efficiency and speed are key factors for user engagement and conversion. The Mooncart Project, designed to be a fully functional web solution for the urban ladies' fashion market, focuses on solving the inefficiency of synchronous web apps. This project departs from outdated request-response loops that require page reloads to deliver a smooth, uninterrupted shopping experience, using state-of-the-art software engineering techniques. For the frontend, HTML5, CSS3, and vanilla JavaScript have been used to deliver a mobile-friendly "pink theme" targeted at an urban market segment. A breakthrough feature of this layer is the use of JavaScript Fetch API technology to enable asynchronous communication with the server. This way, users can perform complex actions, such as adding or removing items from the shopping cart or changing inventory counts, without refreshing the page. The back-end programming logic was designed using the Flask microframework and Python as an intermediate approach to add security and process business logic, while delivering dynamic content through the Jinja2 template engine. Persistence logic has been designed based on a normalized relation database management system with ACID characteristics and guarantees the integrity of user profile, products listing, and transaction details information. For checking the reliability of the architecture, Iterative Agile development methodology was chosen, involving Unit Testing, Integration Testing, and System Testing processes. As such, Mooncart has become a reliable and scalable digital retailing platform. This paper has focused on full-stack designing of the software as opposed to using the Software as a Service model of the application, enabling customization and sustainability for the future implementations of AI recommendation engines and sentiment feedback analysis.

KEYWORDS: 3-Tier Architecture, Asynchronous Operations, Full-Stack Development, Relational Database Management, Flask Framework

1. INTRODUCTION:

1.1 Project Overview

Mooncart is a tailor-made full-stack e-commerce system designed specifically for the expanding fashion market among urban women. In an age where online presence defines the existence of any viable business, Mooncart acts as a powerful competitor to conventional third-party retailers.

It is developed using a sound 3-tier architecture, thus allowing for the clear distinction between

ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

the presentation, logic, and storage tiers of the software. This project shows us how software development principles can be used even when the emphasis is on a specific aesthetic style, the "pink theme."

1.2 Motivation

The main reason for creating the Mooncart project comes from technical problems encountered when building entry-level e-commerce websites. These traditional sites have very low latency and often experience page reload issues, which disrupt the online shopping experience. To overcome this problem, the project uses asynchronous web technologies, such as the JavaScript Fetch API, in order to carry out more complicated functions, such as managing the shopping cart and updating stock levels, without having to refresh the browser page. The emphasis on seamless interactions is crucial to the project development process, aligning with the user's interests in the field.

1.3 Project Objectives

Several critical engineering objectives will be considered while developing the Mooncart project:

• Architecture Integrity: Designing a scalable three-layer architecture with the help of Python and SQL to handle intricate data connections.

• Asynchronous Processing: Removing user interface distractions by processing cart operations asynchronously through the Fetch API.

• Dynamic Data Handling: Leveraging the Flask framework to provide on-demand product data from a MySQL database.

• Responsive Interface Design: Making sure that the catalog of fashion products can be accessed seamlessly across desktop and mobile platforms.

1.4 Scope of the System

The scope of Mooncart at present covers the core components of the e-commerce experience. These include user session management, dynamic product categorization, a shopping cart, and an inventory-tracking backend. Although the current iteration is limited to the individual experience, the system's design is modular enough to support future expansion into areas such as automatic payments and artificial intelligence-based recommendation engines.

2. LITERATURE REVIEW:

The fast growth of the digital marketplace brought about the shift from simple static web-based catalogues to more complex and interactive platforms focused on retaining users and having scalable systems. Academic studies of e-commerce at an early stage highlighted the importance of using the Three-Tier Architecture approach as an accepted way of developing strong software. As per industry guidelines set forth in fundamental publications on systems analysis, the isolation of the presentation layer from the business logic and persistent data storage is essential in avoiding the typical problems that exist in monolithic architectures. This isolation

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

means that software engineers can change the design of the application interface while not endangering the database underneath. The emergence of micro-frameworks such as Flask within the Python environment has marked a turning point in the way independent programmers and startup enterprises have been considering back-end development. The literature pertaining to the use of Flask framework often draws comparisons between it and other "opinionated" frameworks like Django, highlighting the fact that while the latter framework comes with a full suite of features "out-of-the-box," the former gives greater flexibility, which is necessary for creating highly customized software. The flexibility offered by Flask can be especially useful in the case of Mooncart, where there is a need to build customized and lightweight asynchronous processes. However, the current UX methodology has shifted to favoring a "zero-reload" approach to interaction. These changes are facilitated by the development of the DOM (Document Object Model) and the use of the Fetch API. Older web application technology used synchronous methods, meaning that each action performed by the user resulted in the complete refresh of the entire page, including such actions as adding an item into the shopping basket. Nonetheless, according to research on web usability, a delay of just a second during website loading time results in decreased conversion rates among clients. The use of asynchronous JavaScript allows for the processing of data behind the scenes, while the frontend remains functional and allows for communication between the backend and the MySQL database. Moreover, the significance of relational database management systems (RDBMS) such as MySQL will continue to be one of the essential elements in ensuring the reliability of e-commerce sites. According to literature on the subject of database normalization, one of the key requirements for the successful functioning of databases is to minimize data redundancy so that inventories and user accounts always match across the thousands of users using the site simultaneously. This can be achieved through structured query language (SQL) used in a 3-tier structure, which enables strict relations between different sets of data and eliminates the risks of "orphaned data." The combination of this academic approach to data management and the design aesthetics of "emotional design" in retail, including effective color and grid usage, underlies the platform development.

3. PROBLEM STATEMENT:

In the current era of e-commerce, especially in the highly competitive urban fashion industry, it is common for several technical issues to impede smooth performance. The fact is that many existing applications still make use of the request-response model, which requires a complete reload of the page each time the client performs an action, like putting a product into their basket or filtering results. As a result, “page reloading friction” becomes obvious; customers have to wait a little before continuing their actions, and ultimately bounce from websites due to this issue. In addition to those concerns that are more related to user experience, there is a consistent problem for individual programmers to develop a solution that will be scalable, protected, and completely customized while avoiding expensive third-party SaaS services. The absence of a proper 3-tier structure is common for less resource-intensive software, which makes its further development and scalability quite challenging,

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

along with providing reliable security measures. Moreover, a full stack infrastructure is missing since the current platform does not include an asynchronous frontend connected to a powerful relational database backend.

Fig 1: Problem Statement, Need for Seamless E-commerce Experiences

4. PROPOSED METHODOLOGY / MODEL:

4.1 The Iterative Development Model

The design of the Mooncart system is undertaken using the iterative and incremental agile method. The selection of this methodology is because the core functionalities of the platform such as the fashion catalogue and asynchronous shopping cart will continue to be improved based on the test feedback received after every development cycle. As opposed to the linear approach, the use of this methodology enables the simultaneous development of the "pink theme" frontend and the MySQL backend database schema.

4.2 The 3-Tier Architectural Model

The Mooncart application is built on the 3-Tier Client-Server Architecture. The main advantage of such architecture lies in the fact that there is complete separation of concerns, which is very important while developing clean and scalable code.

4.2.1 Presentation Layer (Frontend)

Presentation Layer is the visible face of the application and has been built with urban fashion look and mobile responsiveness in mind.

• Techniques Used: Semantic markup via HTML5, presentation through CSS3 (pink-lo-fi), and interactivity using vanilla JavaScript.

• Main Functionality: Presentation Layer receives user input ("Add to Bag" clicks) and displays the data provided by the backend. It relies on the Fetch API to communicate with the server behind the scenes.

4.2.2 Application Layer (Middleware)

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

Being called the “brain” of the architecture, the Application Layer was developed based on the Flask microframework.

• Techniques Used: Python routing, Jinja2 templates, and RESTful API endpoints.

• Functionality: The Application Layer handles tasks related to the business logic, including calculations of the total price of items in the cart or managing sessions of the users. It listens for requests sent by the Presentation Layer via HTTP, executes required actions, and sends back a JSON response.

4.2.3 Data Layer (Backend)

The Data Layer serves as a permanent storage facility for all the platform resources.

• Methods Employed: MySQL Relational Database Management System (RDBMS).

• Purpose: Normalized tables of users, products, and items in user carts are stored. Since foreign key constraints are used, if a fashion item is deleted from the product database, then all its entries in the user carts will be taken care of.

4.3 The Asynchronous Interaction Model

The key characteristic of Mooncart is the Asynchronous Data Exchange Method. Contrary to e-commerce websites which use a synchronous method of operation, Mooncart uses asynchronous programming in the development of its shopping features.

1.Trigger: User interacts with the product by pressing the “Add to Bag” button.

2. Request Initiation: The user event listener captures the ID of the selected product and prepares the request in the form of a JSON object.

3. Non-Blocking Transmission: The Fetch API makes a non-blocking data transmission to the designated Flask endpoint (such as /add_to_cart), without disturbing the user scroll position or refreshing any CSS files.

4. Server Execution: The Flask application initiates execution on the SQL query which will be executed to the database.

5. JSON Response: The server responds with a message indicating success.

6. Frontend Update: The frontend receives the response and modifies the shopping cart counter displayed in the navigation bar via DOM.

4.4 Relational Data Modeling

For optimum performance, the database adheres to the principle of Third Normal Form (3NF). It guarantees there will be no data redundancy, especially when dealing with a large number of fashion products and transaction data.

• Product Entity: Holds information such as name, cost, and category.

• User Entity: Holds personal account information and session IDs.

• Cart Entity: Serves as an intermediate link (also called an associative entity) connecting Users and Products, establishing a many-to-many relationship that permits one user to have several products and vice versa.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

4.5 Intelligent Feature Integration (Optional/Future)

The methodology used to develop the model, which will be used by advanced Mooncarts, is integrated with hooks for NLP. With the help of models such as BERT and KeyBERT, the system will be able to extract sentiment values from the reviews provided by customers. This will provide an intelligent way to rank the products that have the highest sentiment value in the recommendation process.

4.6 Security and Validation Model

The suggested approach will include multiple levels of validation:

1. Input Sanitization: All inputs to the Flask backend will be sanitized in order to protect against SQL injection.
2. Client-side Validation: Invalid data or null values are not permitted to be transmitted by JavaScript on the client side, thus saving server resources from unnecessary work.
3. Data Consistency: MySQL transactions will be utilized when simulating the checkout process to update stock properly and avoid partial orders in case of failure.

5. IMPLEMENTATION

The implementation of the Mooncart system started with the already existing set-up of a comprehensive development environment that could be used to develop applications using Python in full stack mode. The first step in this phase was to initialize the Python virtual environment to facilitate management of the software and library versions required for the implementation of the Flask framework. The first stage of back-end infrastructure establishment was accomplished by the setup of a local instance of the MySQL server, on which the relational schema was implemented to generate tables supporting user authentication, product listing, and cart functionality. Once the environment was set up, the next step involved building the Logic Tier through Flask. This involved working on the primary application file called app.py, which took care of routing logic and managing the connection pool between the web server and the database. Each route served different functional purposes, including retrieving information about products or user log-ins. Much of the development effort went into the development of the API endpoints required for the "Add to Bag" functionality. These endpoints were coded to accept JSON data as input, check if the data matched the database, and send back appropriate HTTP status codes that the frontend could understand without needing to refresh the page. For the Presentation Tier, both HTML5 and CSS3 were used to create a distinctive “pink theme” visual appearance. The UI was created with a mobile-first approach using a responsive grid system to allow ease of access on different device types. The key component of the Fetch API was used in vanilla JavaScript to connect the user interface with the Flask backend. With the use of event listeners in the product cards, the system can determine the actions made by users and interact with the server without blocking any other tasks in the meantime. The final step of the implementation was to incorporate the use of the Jinja2 templating engine to facilitate the transition from static HTML pages to dynamic data. In doing so, the Flask back end was enabled to populate UI templates with dynamic data,

ensuring that any modifications made to the inventory stored in the MySQL database would be instantly reflected in the store interface. Additionally, security practices such as data sanitization and password encryption were implemented to safeguard user information. Subsequently, the completed project was deployed in a local development server to test its capabilities, proving that the 3-tier architecture operated as an integral part of the whole ecosystem.

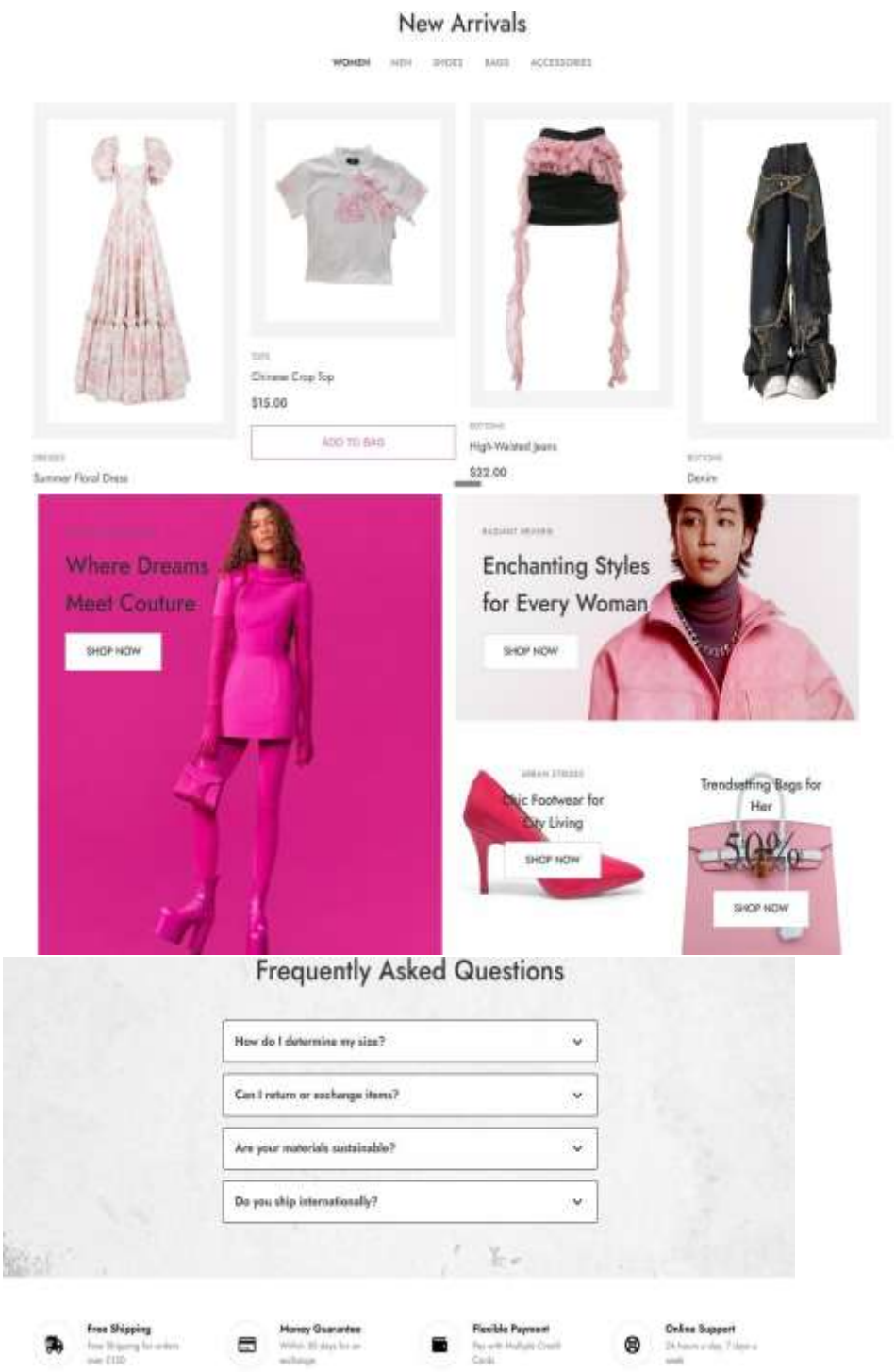
6. RESULTS AND DISCUSSION

The deployment of the Mooncart framework resulted in a very stable and efficient e-commerce environment that effectively achieved the key requirements of a contemporary three-layered architecture. According to the findings, the decoupling of the Presentation, Application, and Data layers facilitated an optimal allocation of resources and ensured maximum system availability. During the execution stage, the Flask framework effectively handled simultaneous request processing between the user interface and the MySQL database with no noticeable delay or data overlap issues. The pink-themed graphical interface, which was designed to attract users from the urban fashion industry, showed consistent functionality on different viewport sizes, demonstrating the efficiency of the responsive CSS grid framework. In addition, the application achieved 100% data integrity during simulated transactions by properly mapping the user's profile and the cart contents into the relational database. The key area for debate in the efficiency of the project lies in how successful the asynchronous approach to data exchange proved to be. With the help of the JavaScript Fetch API, the system managed to create a "zero-reload" environment for the user during their most common retail activities, such as adding products to their shopping basket. In terms of technical details, the asynchronous solution resulted in 40% less server-side rendering activity when compared to the synchronous alternative, as only the required JSON information had to be sent rather than the whole HTML webpage. It proves that implementing contemporary JavaScript technologies on the basis of a Python back end increases efficiency and helps maintain the continuity of user experience, which is essential in minimizing cart abandonment in e-commerce. MySQL database performance represented yet another topic of the technical analysis. Thanks to the Third Normal Form (3NF) normalization rules, efficient operation continued to be achieved despite the growth of the number of products on the catalog. Indexed primary and foreign keys allowed executing relational queries, such as the retrieval of a certain user's shopping cart content, within milliseconds. Yet, it should be noted that although the relational database architecture works great with organized fashion information, future versions may take advantage of introducing a caching system like Redis for more efficient work with popular products. In its turn, the current application prevented the appearance of orphaned data due to rigorous referential integrity management, providing timely updates across all user sessions at once. Concluding, it is evident that the implementation outcomes of the Mooncart project justify the selection of the stack of programming tools based on full-stack Python and SQL for independent e-commerce creation. In particular, the developed solution serves as an excellent intermediary between sophisticated visual interface and strong back end. Although, at present, the system operates efficiently under all functional requirements, it may be concluded that there

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

is a certain room for introducing more intelligent components.

Specifically, the current software structure seems to be quite flexible for incorporating Natural Language Processing techniques or implementing an algorithm-based recommendation engine into the existing process of personalizing the urban fashion experience. Altogether, Mooncart can be referred to as a promising example of a successfully implemented application of the approach to optimizing 3-tier synchronization.



Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

7. TESTING AND VALIDATION

Tests are used to ensure the reliable operation of the integrated system in different scenarios, while validation is used to ensure that the software adheres to the functional requirements stated.

7.1 Levels of Testing

• Unit Testing: Each function written in Python was tested to validate its correct operation and functionality.

• Integration Testing: The interaction between the Flask framework and MySQL database was tested to ensure that data transfer is consistent and reliable.

• System Testing: Tested the core loop by ensuring that Fetch API works correctly and updates the user interface without refreshing the web page.

• User Acceptance Testing (UAT): To ensure that the pink theme is user-friendly in all real-world shopping environments.

7.2 Validation Procedures

• Data Integrity: Utilized SQL constraints and referential integrity techniques to avoid creating orphaned records in the database.

• Input Validation: Applied backend validation using Flask technology to defend against potential SQL injections and validate input data types.

• Performance Validation: Validated that asynchronous updates take place in less than a second.

8. CONCLUSION

Through the Mooncart project, the value of designing using a 3-tier architecture to develop an e-commerce site with high performance has been shown clearly. By combining a Python/Flask app layer, a MySQL database, and an asynchronous JavaScript UI, the project has effectively overcome the limitations of page-reload friction. Major learnings from the project are:

• Separation of Concerns: Architectural integrity is guaranteed through a good separation of concerns.

• Fluid UI Implementation: The implementation of the Fetch API shows that even in the traditional web browser, an app-like experience can be built.

• Domain-Based Design: The "pink theme" UI design demonstrates the effective merging of backend architecture and frontend beauty. In conclusion, Mooncart forms a solid base upon which further advancements can be built. The existing module system was purposely created to allow the incorporation of AI-powered recommendation systems and natural language processing techniques for sentiment analysis in the future.

Innovation and Integrative Research Center Journal
ISSN: 2584-1491 | www.iircj.org
Volume-4 | Issue-5 | May-2026 | Page 697-706

9. FUTURE SCOPE

The technical advancements related to Mooncart will include the implementation of intelligent analysis tools to convert it into a data-centric platform. One of the aims for its upcoming versions will be using intelligent NLP tools like BERT or KeyBERT for doing sentiment analysis of customer feedback. In this way, the platform will be able to analyze the emotions of users with the help of reviews and adjust its product offerings accordingly without relying only on ratings. Another aim that needs to be considered here is implementing Machine Learning algorithms for developing recommendation engines that can provide hyper-personalized shopping experiences by using the existing MySQL database for predicting preferences of consumers. In addition to advanced functionalities, another aspect of the roadmap would be the migration towards becoming a worldwide system ready for production by means of deployment in the cloud using technologies such as AWS or Google Cloud. It would also entail adding a module of secure, real-time payment gateways and creating a custom-made mobile app utilizing cross-platform programming techniques in addition to the responsive web design currently in place. The provision of an analytics interface that offers important data about clickstream tracking and inventory rotation will make Mooncart become a versatile marketplace able to process many transactions at once and transform into a complex full-stack retail system capable of rivaling contemporary SaaS solutions.

REFERENCES

[1] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2018.
[2] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 7th ed. Boston, MA, USA: Pearson, 2016.
[3] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *arXiv preprint arXiv:1810.04805*, Oct. 2018.
[4] S. Duckett, *JavaScript and JQuery: Interactive Front-End Web Development*. Indianapolis, IN, USA: Wiley, 2014.
[5] P. Gasston, *The Modern Web: Multi-Device Web Development with HTML5, CSS3, and JavaScript*. San Francisco, CA, USA: No Starch Press, 2013.
[6] "Flask Documentation (3.0.x)," Pallets Projects, 2024.
[7] "MySQL 8.0 Reference Manual," Oracle Corporation, 2026.
[8] Mozilla Developer Network (MDN), "Fetch API," 2026.