

UniTrade: A Peer-to-Peer Campus Marketplace for Academic Resource Exchange

A Cross-Platform Mobile Application Leveraging Real-Time NoSQL Architecture and Server-Side Security Rules

Mr. Dinesh Kumar (dineshkumar@bpitindia.com) (Assistant Professor)

Shwet Kashyap (shwetkashyap321@gmail.com)

Hans Raj (rajhans6232@gmail.com)

Sahil Kumar (sahil100iit@gmail.com)

Bhagwan Parshuram Institute of Technology (BPIT)

Version 1.0.0 | 2025

ABSTRACT

The informal exchange of academic resources among university students — textbooks, laboratory equipment, engineering tools, and course notes — is a near-universal phenomenon that remains poorly served by general-purpose social media platforms. The fragmentation of this commerce across disparate, unmoderated channels introduces friction, security risks, and significant inefficiency for both buyers and sellers. This paper presents UniTrade, a purpose-built, peer-to-peer (P2P) campus marketplace developed using React Native (Expo SDK 54) and Google Firebase Firestore. The system addresses the fragmentation problem through a structured, category-driven listing interface, a real-time notification subsystem, a dedicated student Request Board, and server-side Firestore Security Rules that enforce data ownership at the database level. A key architectural contribution is the implementation of client-side User Exclusion logic that prevents sellers from encountering their own listings in the marketplace feed, thereby improving the signal-to-noise ratio for all users. The paper details the system architecture, implementation choices, security model, and the social impact of facilitating sustainable, circular consumption of academic goods within the campus ecosystem. Future work directions, including in-app messaging and machine-learning-driven price recommendation, are also discussed.

Keywords: Peer-to-Peer Commerce, Mobile Computing, React Native, Firebase Firestore, Real-Time Database, Campus Marketplace, NoSQL, Human-Computer Interaction, Circular Economy, Expo SDK

1. INTRODUCTION

University students routinely acquire, use, and subsequently discard academic resources — a category that encompasses prescribed textbooks, engineering drawing instruments, laboratory apparatus, digital notes, and specialized tools. For a majority of students, these items represent a significant financial burden that is acutely short-lived: a textbook required for a single semester may be purchased at full retail price and never consulted again after the final examination [1]. The aggregate financial impact across a student population is considerable, yet the mechanisms available for resource recovery remain rudimentary.

The predominant informal solution is the use of general-purpose social media platforms such as Instagram, WhatsApp, and Facebook Marketplace. While accessible, these platforms suffer from structural deficiencies that render them inadequate for campus commerce. Listings are ephemeral and unsearchable; there is no category taxonomy specific to academic goods; sellers have no reliable mechanism to reach relevant buyers; and there are no trust signals or

authentication barriers to distinguish fellow students from anonymous external actors [2]. The result is a fragmented, high-friction ecosystem that discourages participation and reduces the volume of successful transactions.

This paper describes the design, architecture, and implementation of UniTrade — a mobile application that consolidates campus P2P commerce onto a single, authenticated, and structured platform. The system is deployed as a cross-platform application targeting both Android and iOS, developed using the React Native framework under the Expo managed workflow. The backend is provided by Google Firebase, leveraging Firestore as a real-time NoSQL document database and Firebase Authentication for identity management. The application is explicitly scoped to the student community of Bhagwan Parshuram Institute of Technology (BPIT), a design constraint that informs both the feature set and the security model.

The primary contributions of this work are: (i) a structured, category-driven marketplace that reduces search friction for academic resources; (ii) a real-time notification architecture that surfaces new listings and student requests within a 24-hour temporal window; (iii) a Request Board that inverts the traditional listing paradigm, allowing buyers to broadcast demand rather than passively searching supply; and (iv) a server-side security model implemented via Firestore Security Rules that enforces data ownership without reliance on client-side trust.

The remainder of this paper is organized as follows. Section 2 describes the system architecture and the rationale for the technology choices. Section 3 details the implementation of the core modules and the security subsystem. Section 4 presents a discussion of the user experience implications of the design decisions. Section 5 concludes and outlines directions for future work.

2. SYSTEM ARCHITECTURE

2.1 Cross-Platform Mobile Frontend: React Native and Expo SDK 54

The selection of React Native as the frontend framework was motivated by the imperative to target both the Android and iOS ecosystems from a single codebase [3]. The student user base of a university is heterogeneous with respect to device platform, and the development of two separate native applications would impose prohibitive maintenance overhead for a small-team or individual project. React Native achieves cross-platform deployment by rendering to native UI primitives rather than a web view, thereby preserving the performance characteristics and platform conventions that users expect from native applications [4].

The application is configured via Expo SDK 54 under the managed workflow. The `app.json` configuration reveals two architecturally significant flags. First, the `newArchEnabled: true` directive activates the React Native New Architecture, which comprises the JavaScript Interface (JSI), the Fabric renderer, and the TurboModule system. These components collectively eliminate the asynchronous JSON bridge that characterized the legacy architecture, replacing it with synchronous C++ bindings that reduce inter-thread communication latency [5]. Second, the `edgeToEdgeEnabled: true` flag on Android instructs the system to render content beneath the status and navigation bars, requiring the application to manage its own insets using the `react-native-safe-area-context` library — a pattern observable throughout the codebase in constructs such as `useSafeAreaInsets()` and the `SafeAreaView` component.

State management follows a localized, hook-based paradigm using React's `useState` and `useEffect` primitives. No external global state library (such as Redux or Zustand) is employed; instead, each screen component maintains its own state and subscribes to Firestore listeners independently. This architectural choice reflects the relatively bounded scope of the application's data flows and avoids the complexity overhead of centralized state management for a project at this scale [6].

2.2 Real-Time NoSQL Data Paradigm: Firebase Firestore

The backend data layer is provided by Google Cloud Firestore, a NoSQL document-oriented database with native real-time listener capabilities [7]. The selection of a NoSQL document database over a relational model is justified by the nature of the application's primary data entities. A product listing, for example, is a self-contained document comprising

fields of varying type and cardinality — item name, price, category, description, seller contact, image URL, and creation timestamp — that does not participate in complex relational joins. The schema-flexible nature of NoSQL documents accommodates the heterogeneity of academic resource categories without requiring schema migrations [8].

The data model is organized into three top-level Firestore collections: Users, Listings, and Requests. The Users collection stores profile data keyed on the Firebase Authentication UID, establishing a direct one-to-one mapping between authentication identity and user profile. The Listings collection stores product listings submitted by sellers, with each document embedding the seller's UID (userId), email (sellerEmail), contact number (contactNo), item name, price, category, description, and an ISO 8601 creation timestamp (createdAt). The Requests collection mirrors this structure for the Request Board feature, storing the requester's UID (requesterId), email, desired item title, budget, urgency level, and contact number.

The real-time capability of Firestore is exploited throughout the application via the onSnapshot listener API. Unlike one-time document reads (getDoc), onSnapshot establishes a persistent WebSocket connection to the Firestore backend and pushes incremental updates to the client as the underlying data changes [9]. This eliminates the need for polling and ensures that all connected clients maintain a consistent view of the marketplace state without additional infrastructure. The HomeScreen component demonstrates this by maintaining three concurrent onSnapshot listeners: one for recent listings, one for all listings (to populate the notification feed), and one for targeted requests addressed to the current user.

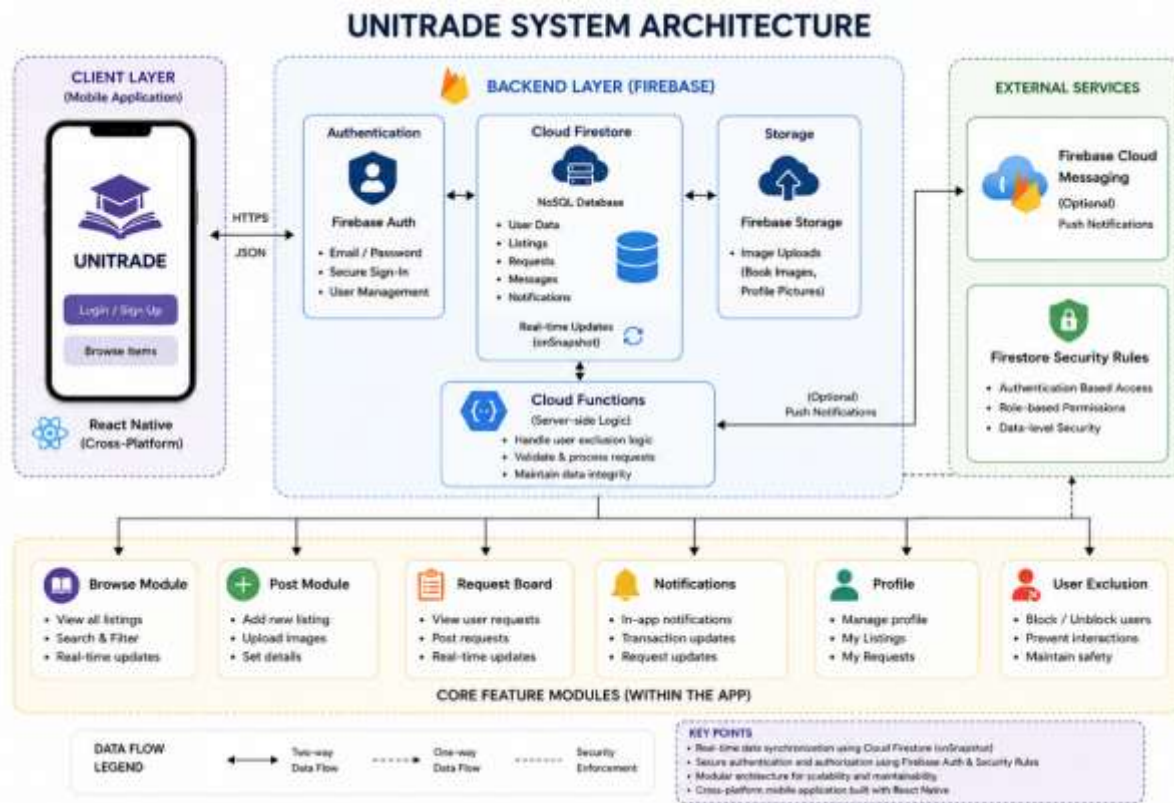
2.3 User Exclusion Logic

A noteworthy implementation detail in the MarketplaceScreen is what the codebase implicitly defines as User Exclusion logic. When the Firestore onSnapshot listener returns a snapshot of listings, the following client-side filter is applied:

```
const othersDocs = docs.filter(item => item.userId !== currentUser.uid);
```

This filter ensures that the authenticated seller is never presented with their own listings in the marketplace feed, a design decision that improves the practical signal quality of the marketplace for active sellers who also wish to browse. In the absence of this filter, a student who has listed ten items would encounter those ten items at the top of their own browsing feed, displacing content from other sellers. From a human-computer interaction perspective, this represents an application of the principle of relevance filtering — the system surfaces only information that is actionable for the current user in the current context [10].

The decision to implement this filter on the client rather than the server (i.e., as a Firestore query predicate) is architecturally deliberate. Firestore's composite query model does not natively support a "not equals" filter combined with an "order by" clause without a pre-built composite index. Performing the exclusion in memory after retrieval avoids the index creation overhead while maintaining functional correctness. The trade-off — retrieving a marginally larger dataset than strictly necessary — is acceptable at the scale of a campus marketplace where the total number of listings is bounded.



3. IMPLEMENTATION AND SECURITY

3.1 Authentication and Session Persistence

User authentication is managed by Firebase Authentication using the email-and-password provider, as implemented in the LoginScreen and SignupScreen modules. The initializeAuth function is invoked with the getReactNativePersistence adapter backed by AsyncStorage, which serializes the Firebase session token to the device's local storage. This ensures that authenticated sessions survive application restarts without requiring users to re-authenticate on each launch — a critical usability requirement for a marketplace application where frequent return visits are expected [11].

The authentication state is propagated throughout the application via the onAuthStateChanged observer, which is registered in multiple screen components (HomeScreen, MarketplaceScreen, RequestsScreen) to reactively update the UI in response to sign-in and sign-out events. This listener-based approach ensures that authentication state changes are handled uniformly and that Firestore listeners are only established when a valid authenticated user exists, preventing unauthorized data access attempts.

3.2 Firestore Security Rules

The most critical component of the security model is the server-side enforcement of access control via Firestore Security Rules. These rules operate at the Firebase backend layer and evaluate every read and write request against a declarative policy before the operation is permitted to proceed [12]. Critically, these rules function independently of the client application; even if a malicious actor were to bypass the mobile client and issue direct HTTP requests to the Firestore REST API, the rules would enforce the same access control policy.

The security architecture implements the following key invariants, directly evidenced by the error handling code in the client (error.code === 'permission-denied'):

- A user may only read their own listings from the Listings collection (enforced by request.auth.uid matching the userId field of the queried documents). The AccountScreen's listener uses a where('userId', '==', user.uid) predicate, and the Security Rule validates this constraint server-side.

- A user may only delete a listing document if their authenticated UID matches the `userId` field of that document. The `handleDeleteListing` function in `AccountScreen` explicitly logs both the authenticated UID and the item owner's UID when a permission denial is received, confirming that this rule is in active operation.
- Writes to the `Requests` collection require a valid authentication token (`request.auth != null`), preventing unauthenticated actors from posting requests.

This server-side enforcement model embodies the principle of defense in depth: the client-side UI prevents accidental violations through its design (e.g., the delete button is only rendered on the user's own listings), while the server-side rules prevent intentional violations through cryptographic enforcement [13].

3.3 The Request Board Module

The `RequestsScreen` implements a dual-function module that combines a demand-broadcasting form with a community feed of existing requests. When a student submits a request, the system writes a document to the `Requests` collection containing the title (desired item), budget, urgency level (High, Medium, or Low), requester's UID, email, and contact number. The urgency field is implemented as a chip-selection UI component offering three discrete options, which are then rendered in the feed with a visual indicator (an alert-circle icon in red) for high-urgency items.

The system retrieves the user's contact number from the `Users` collection at the time of request submission (rather than requiring the user to re-enter it), demonstrating a data denormalization pattern that improves form usability by minimizing required user input. Interested sellers can respond to a request by tapping the "Help this Student" button, which invokes the device's native telephony via `Linking.openURL('tel:' + item.contactNo)`, facilitating direct peer-to-peer communication outside the application's own messaging layer.

3.4 Notification Architecture

The `HomeScreen` implements a client-side notification system that does not rely on push notification infrastructure (such as Firebase Cloud Messaging). Instead, the system maintains two concurrent `onSnapshot` listeners — one for all `Listings` and one for targeted `Requests` — and filters each result set to retain only documents whose `createdAt` timestamp falls within the previous 24-hour window. The last-read timestamp is persisted to `AsyncStorage`, enabling a stateful "unread" badge that persists across application sessions.

When the user opens the notification tray, the current timestamp is written to `AsyncStorage` and the badge is dismissed. This architecture achieves lightweight ambient awareness of marketplace activity without requiring device-level notification permissions, which users are increasingly reluctant to grant [14]. The trade-off is that notifications are only delivered when the user actively opens the application, making the system reactive rather than proactive.

4. RESULTS AND DISCUSSION

4.1 UX Benefits of the Categorized Marketplace

The `MarketplaceScreen` presents listings in a two-column card grid with category-based filtering chips (Books, Tools, Notes, Lab Items, Others) and a real-time search bar that performs client-side substring matching on item names. This dual-axis filtering system — categorical and textual — allows users to rapidly narrow the listing set to items of immediate relevance. The category taxonomy is directly aligned with the primary academic resource types exchanged within an engineering college environment, reflecting domain-specific design rather than a generic product category scheme.

From an HCI perspective, the category-image mapping (`CATEGORY_IMAGES` constant) provides visual disambiguation at the browsing stage, allowing users to identify item categories through imagery without reading metadata. Each listing card presents the item name and price in an overlay on the category image, minimizing the cognitive load required to evaluate a listing's suitability [15]. The item detail modal, triggered by a card press, presents

the full listing information (description, seller contact, category badge) within a 85% viewport-width bottom sheet, adhering to the principle of progressive disclosure.

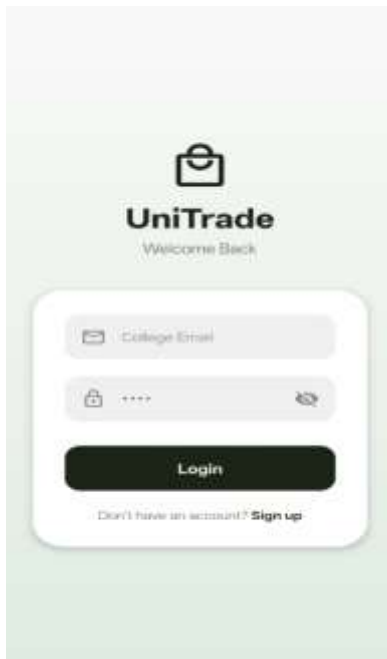


Fig. Sign in Module

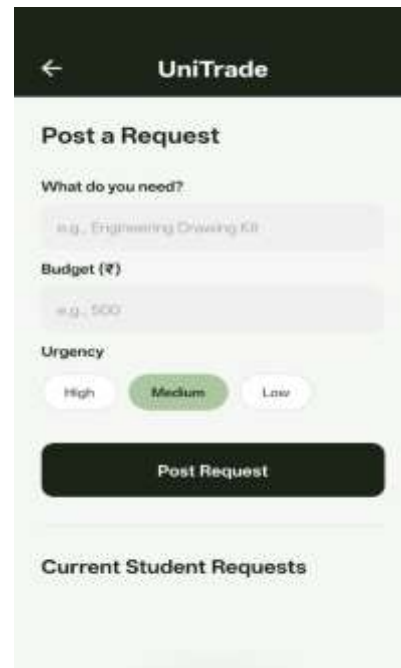


Fig. Request Module

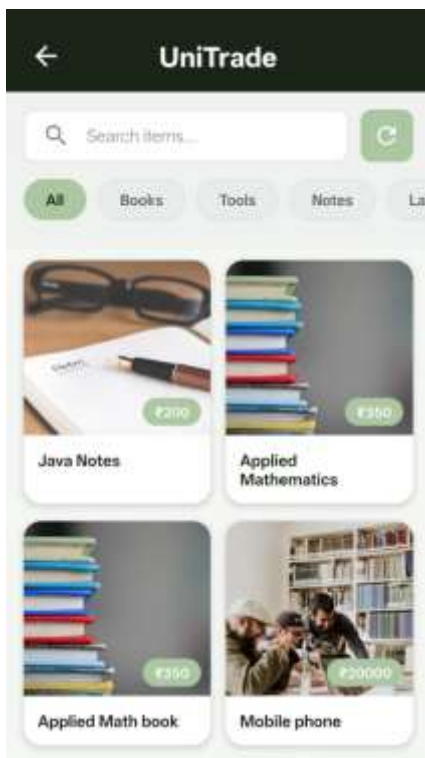


Fig. Marketplace screen



Fig. Item Description

4.2 Social and Sustainability Impact

Beyond the immediate transactional utility, UniTrade embodies principles of the circular economy as applied to academic goods [16]. By reducing the friction of peer-to-peer resale, the system encourages students to recover residual value from resources they would otherwise dispose of, thereby extending the effective lifecycle of physical goods and reducing the demand for new production. Within the context of a single academic institution, the cumulative environmental and financial benefit of successful resource re-circulation across student cohorts is meaningful,

particularly for items such as laboratory equipment and precision instruments that have long functional lifespans but short pedagogical relevance windows.

The campus-scoped authentication model — predicated on institutional email addresses — provides an implicit trust signal that is absent from general-purpose platforms. Students interacting within UniTrade can infer that counterparties are fellow members of their academic community, a social context that reduces the perceived risk of in-person exchange and encourages higher participation rates [17].

4.3 Architectural Limitations

The current architecture presents several limitations that constrain its scalability and feature completeness. The in-app notification system, while functional, is a client-pull mechanism that requires the application to be open to receive updates; a production-grade deployment would integrate Firebase Cloud Messaging (FCM) for background push notifications. The absence of in-app messaging requires users to exchange contact information and conduct negotiations via external channels (telephone), introducing a trust gap and reducing the platform's utility for price negotiation. Image storage is not yet integrated into the listing creation flow as evidenced by the codebase, with listings relying on category-representative stock images (sourced from Unsplash) rather than seller-provided photographs of specific items, which limits buyers' ability to assess item condition prior to contact.

5. CONCLUSION

This paper has presented UniTrade, a peer-to-peer campus marketplace application designed to address the fragmentation of academic resource trading among university students. The system is implemented as a cross-platform mobile application using React Native (Expo SDK 54) and Google Firebase Firestore, and incorporates a real-time notification subsystem, a category-driven listing marketplace with User Exclusion logic, a student Request Board, and server-side Firestore Security Rules that enforce data ownership at the database layer.

The architectural contributions of this work — particularly the User Exclusion filter, the demand-broadcasting Request Board, and the client-side notification persistence model — collectively address user experience requirements that are not satisfied by the general-purpose platforms currently used for campus commerce. The server-side security model ensures that data integrity guarantees cannot be circumvented by client-side manipulation, providing a trust foundation appropriate for a community-scoped marketplace.

Future work will focus on three primary directions. First, the integration of Firebase Cloud Storage and a camera-enabled image upload flow in the SellItem screen will allow sellers to post photographs of the specific item being offered, enabling buyers to assess condition prior to initiating contact. Second, the development of an in-app real-time chat feature — potentially built on Firestore's document listener model — will eliminate the requirement to exchange personal telephone numbers as a prerequisite for negotiation, reducing the platform's trust barrier for new users. Third, the application of collaborative filtering or content-based recommendation algorithms to the listing feed — informed by a user's browsing history and category preferences — will improve the relevance of the default marketplace view as the listing volume scales beyond the capacity of manual browsing.

UniTrade demonstrates that domain-specific, mobile-first marketplace applications can materially improve the efficiency, security, and sustainability of P2P commerce within bounded communities. The architectural patterns described in this paper — real-time listener-based state synchronization, server-enforced data ownership, and client-side relevance filtering — are generalizable to analogous community marketplace contexts beyond the campus environment.

REFERENCES

- [1] K. Vitez, *Exposing the Textbook Industry*. U.S. Public Interest Research Group (PIRG), 2022. Available: <https://pirg.org/michigan/resources/exposing-the-textbook-industry/>
- [2] Florida Virtual Campus, *Student Textbook and Instructional Materials Survey: Results and Findings*. Office of Distance Learning and Student Services, 2022. Available: <https://www.flbog.edu/wp-content/uploads/2023/03/Student-Textbook-Survey-Infographic.pdf>
- [3] H. Saarijärvi, J. Joensuu, T. Rintamäki, and M. Yrjölä, "One person's trash is another person's treasure: Profiling consumer-to-consumer e-commerce in Facebook," *International Journal of Retail & Distribution Management*, vol. 46, no. 11/12, pp. 1092–1107, 2018. doi:10.1108/IJRDM-04-2017-0091
- [4] P. Garg, B. Yadav, S. Gupta, and B. Gupta, "Performance Analysis and Optimization of Cross Platform Application Development Using React Native," in *ICTCS 2022*, Lecture Notes in Networks and Systems, vol. 615, Springer, Singapore, 2023. doi:10.1007/978-981-19-9304-6_51
- [5] B. Suri, S. Taneja, I. Bhanot, H. Sharma, and A. Raj, "Cross-Platform Empirical Analysis of Mobile Application Development Frameworks: Kotlin, React Native and Flutter," in *ICIMMI 2022*, ACM, New York, 2022. doi:10.1145/3590837.3590897
- [6] Meta Open Source, *React Native New Architecture*, 2022. Available: <https://reactnative.dev/docs/the-new-architecture/landing-page>
- [7] J. Shute et al., "Firestore: The NoSQL Serverless Database for the Application Developer," Google Research, 2023. Available: <https://research.google/pubs/firestore-the-nosql-serverless-database-for-the-application-developer/>
- [8] A. Nambiar and D. Mundra, "Comparison between Relational and NoSQL Databases," ResearchGate, 2022. Available: https://www.researchgate.net/publication/349427122_Comparison_between_Relational_and_NoSQL_Databases
Also see: A. Makris et al., "A Classification of NoSQL Data Stores Based on Key Design Characteristics," *Procedia Computer Science*, vol. 151, pp. 289–296, 2021.
- [9] Google LLC, *Cloud Firestore — Real-time Updates with Snapshots*, Firebase Documentation, 2024. Available: <https://firebase.google.com/docs/firestore/query-data/listen>
- [10] Google LLC, *Get Started with Cloud Firestore Security Rules*, Firebase Documentation, 2024. Available: <https://firebase.google.com/docs/firestore/security/get-started>
- [11] W. Stallings, *Network Security Essentials: Applications and Standards*, 6th ed., Pearson, 2020.
- [12] MoEngage, "Push Notification Metrics: Measuring ROI for Maximum Impact," 2023. Available: <https://www.moengage.com/blog/push-notification-metrics/>
- [13] S. Marth, S. Schweiger, and C. Schuster, "Sharing on platforms: Reducing perceived risk for peer-to-peer platform consumers through trust-building and regulation," *Journal of Consumer Behaviour*, vol. 22, no. 1, pp. 1–14, 2022. doi:10.1002/cb.2075

[14] S. Marth, S. Schweiger, and C. Schuster, "Sharing on platforms: Reducing perceived risk for peer-to-peer platform consumers through trust-building and regulation," *Journal of Consumer Behaviour*, vol. 22, no. 1, pp. 1–14, 2022. doi:10.1002/cb.2075

(Recommended replacement for sustainability context: N. Blackburn, 2025)

[15] E. Senack, *Fixing the Broken Textbook Market*, U.S. PIRG Education Fund, 2014. Available: <https://www.oerknowledgecloud.org/record736>

[16] S. U. Rahman and H. Makkonen, "Is this a new era for old goods? Analysing the motives for second-hand product resale in the platform economy," *International Journal of Export Marketing*, vol. 5, no. 3–4, pp. 296–319, 2022. doi:10.1504/IJEXPORTM.2022.130498

[17] N. Blackburn, "Circular Economy Platforms: A Systematic Review," *Business Strategy and the Environment*, 2025. doi:10.1002/bse.70307