Final Year Project

Escrow System to Combat Ticket Touting

Fionn Reilly

Student No.: 20101977

Supervisor(s): Prof. Rob O'Connor

2025-12-26

Abstract

This project focuses on developing a secure and user-friendly marketplace application for buying and selling resale tickets for events. The application aims to address key issues in the ticket resale market, such as ticket touting, fraud prevention, and user reputation management. By implementing an escrow-like transaction flow, the system ensures that funds and tickets are securely held until both parties fulfill their obligations. The project leverages existing technologies like Supabase for backend services and third-party payment processors to facilitate transactions. The design incorporates a messaging system for negotiation, a reputation scoring mechanism, and strict price controls to prevent scalping. This document outlines the project's objectives, design considerations, implementation details, and future development roadmap.

Table of Contents

1 Description of Project

The main objective of this project is to build an application that will serve as a marketplace for buying and selling resale tickets for events, inspired by already established systems like Toutless[1], Trustap[2], and TicketSwap[3]. The project is focused on creating a functional and trustworthy platform by addressing several key areas, including transaction flow, fraud prevention and user reputation.

The application is designed to allow users to scroll through and search current ticket listings. Listings are for singular tickets, and all transactions will be conducted in Euro. Users will be able to sort listings based on criteria such as venue, location, price, age, and category.

A key component and goal of the app is to help prevent ticket scalping/touting; users will be unable to list tickets to sell above the original sale price of the ticket. To determine the original cost, the system will need to request information on the original purchase (things like place, time, date), and or potentially utilizing machine learning to be able to scan uploaded receipts or proof of purchase.

The process of buying a ticket involves several steps designed to facilitate negotiation between the two parties and a secure transfer of assets:

1. Negotiation & Communication: When a buyer finds a ticket they wish to purchase/investigate, they initiate a messaging system with the seller to negotiate price and other details. The project will require a chat solution that is preferably cross-platform, text-based, features logging and is free (consideration has been given to third-party services like Converse.js[4] and Supabase[5]).

2. Once a price is agreed upon, the seller sets the confirmed price within the communication tab. The system should then hold the tickets in an active transaction, meaning the seller cannot revoke the listing. The buyer proceeds to pay, with the payment transaction handled by third-party systems such as Stripe[6], PayPal[7] or

[1] "HOW TO USE TOUTLESS! PLEASE READ BEFORE POSTING. - Toutless", n.d.
[2] "What is Trustap?", n.d.
[3] "How does TicketSwap work?", n.d.
[4] "Converse", n.d.
[5] "Supabase | The Postgres Development Platform." n.d.
[6] "Stripe | Financial Infrastructure to Grow Your Revenue", n.d.
[7] "PayPal Developer", n.d.

Bluefin's[8] APIs.

3. After the system receives the money, the ticket is sent to the buyer and the funds are issued to the seller's account that they have linked to the app.

The project must account for various transaction outcomes, including failures and cancellations:

- Cancellations/Walkaways: The system must handle cases where both parties confirm that they want to cease the transaction. It also covers unilateral cancellations by either the buyer or the seller before funds are transferred, or even cancellation by the buyer after sending money, which would require a refund process.
- Reputation System: As previously mentioned, users will have a calculated reputation score linked to their account. This score is calculated using an average of ratings provided to users by other users once transactions have been ceased (including completed or failed transactions). These ratings are based on factors like politeness, ease of negotiation, etc. Ratings could be exploited by users to misrepresent other users and skew their average, but unfortunately this is a trust issue and I believe this to be the best way to get a representation of a user.

The project aims to provide a system that handles transactional consistency, user experience, and legal constraints, mirroring real-world concepts like the Escrow method where funds or assets are held to guarantee future outcomes.

2 Investigation of Project Idea

The necessity for a secure and regulated ticket resale market is informed by widespread issues including ticket touting (buying and reselling tickets for profit) in the secondary market which is a widespread problem all over the world. Key problems in the resale market include:

- Scalping: The operation of scalpers who purchase tickets to earn a premium, making it difficult for real, dedicated fans to buy tickets

[8] "API Library", n.d.

- High Prices: Increased ticket prices in the resale market lead to a decrease in the event organizer's profit from other sales, like merchandise.
- Vacant Seats/Unused Tickets: If resale prices are too high, seats may remain vacant, which negatively affects the event atmosphere.
- Fraud: Customers face problems when tickets purchased in the resale market cannot be used for entry, leading to an increasing number of inquiries and complaints about online resale fraud
- Touting Methods: Touting relies on creative strategies to deceive and manipulate costumers, exploit loopholes in the primary market, and utilize widespread corruption.

The project's designed (discussed in a later section) directly addresses these concerns, notably by prohibiting users from listing a ticket above its original sale price to prevent scalping.

Research was conducted on existing ticket resale platforms to inform the application's functionality, specifically Toutless, Trustap, and TicketSwap. The goal is to build an application that combines the best features of these platforms while addressing their limitations, i.e. mirroring the marketplace function of Toutless but incorporating the secure transaction flow of Trustap.

The process of securing the ticket and payment during a sale, where the seller holds the ticket and the system holds the money until both parties fulfill their obligations, is inspired by the Escrow Transactional Method commonly used in online marketplaces. This methodology, detailed in the paper by Patrick E. O'Neil[9], provides a layer of security for both buyers and sellers, ensuring that neither party is at risk of losing their assets during the transaction process.

Escrow was developed specifically for long lived transactions in database management. This is crucial for this project as ticket transactions may not be instantaneous, with human interaction and negotiation potentially taking hours or days.

The Escrow method guarantees the recoverability of funds and tickets in the event of a system failure, ensuring that both parties can complete their transactions without loss. This also helps to prevent high concurrency items from acting as a bottleneck.

[9] "Escrow Transactional Method", n.d.

The idea if the anti-scalping rule essentially transfroms the app into a Centralized Exchange (CE) system, allowing users to transfer ticekts at face value or below. The analysis of CE systems in the paper by Kimitoshi Sato[10] provides the economic justification for this design. In this paper, numerical experiments demonstrate that implementing a CE system is effective in lowering the resale prices of tickets, making events more accessible to genuine fans, and improving social welfare, particularly in popular events where demand exceeds supply.

3 Design of Project

The project's design and operational integrity are heavily influenced by two main types of resources: analogous ticket resale platforms and academic transactional methods.

Research was conducted on existing ticket resale platforms to inform the application's functionality (Toutless, Trustap, TicketSwap).

The project is designed and will be developed as a cross platform mobile application. The app will use managed backend services to reduce the infrastructure overhead and allow for greater focus on the application logic, security, and user experience.

The mobile client is developed with Reach Native[11] and the Expo framework[12], allowing for a single code base to be deployed across both Android and iOS platforms. The Expo Router is used to structure the app into different screens, including authentication, message selection and messaging interfaces for the demo for this semester, with more screens able to be added easily when the final product is in development.

Supabase was chosen as the selected backend platform due to its integrated support for authentication, relational data storage, and real time database updates - all this with no cost. User authentication is handled via Supabase Auth using email and password credentials, ensuring that all users interacting with the app are authenticated and uniquely identifiable. Using Supabase Auth allows the same credentials to be across all parts of the app, including user profiles, conversations, and listings in the future.

To establish the database on Supabase, I ran a few SQL queries to create tables, add security, create Postgres functions, and enable real time functionality.

[10] Sato, 2023.
[11] "React Native · Learn once, write anywhere", n.d.
[12] "Expo", n.d.

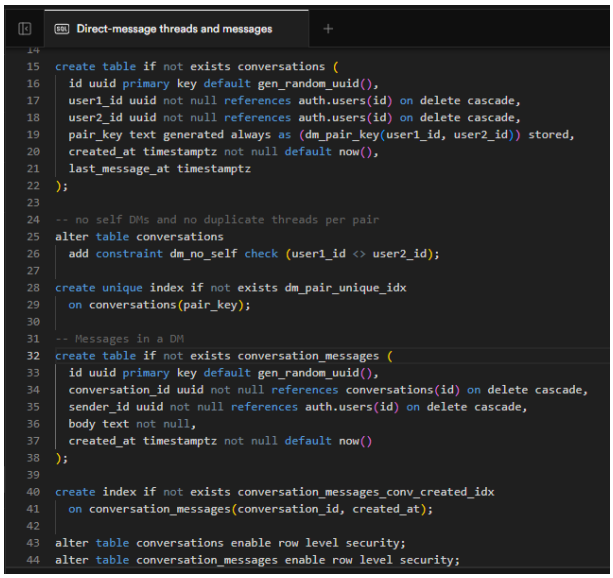

Figure 1: SQL code to create Conversation and Message tables, including constraints and enabling Row Level Security (RLS)

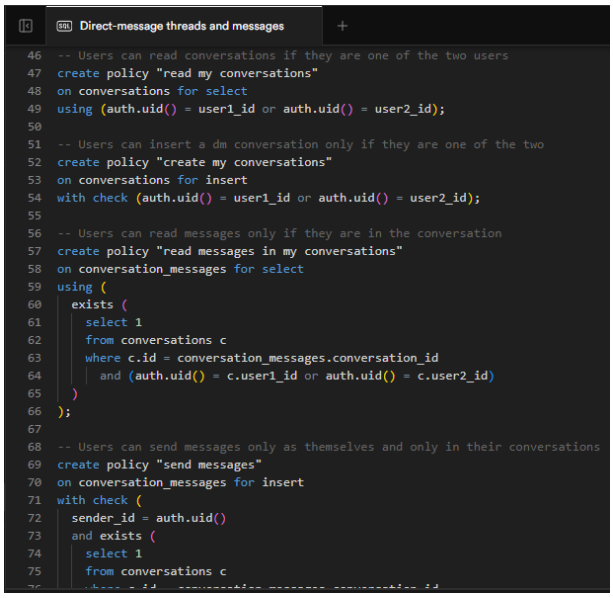

Figure 2: PostgreSQL policies implementing Row Level Security (RLS) for conversations and messages

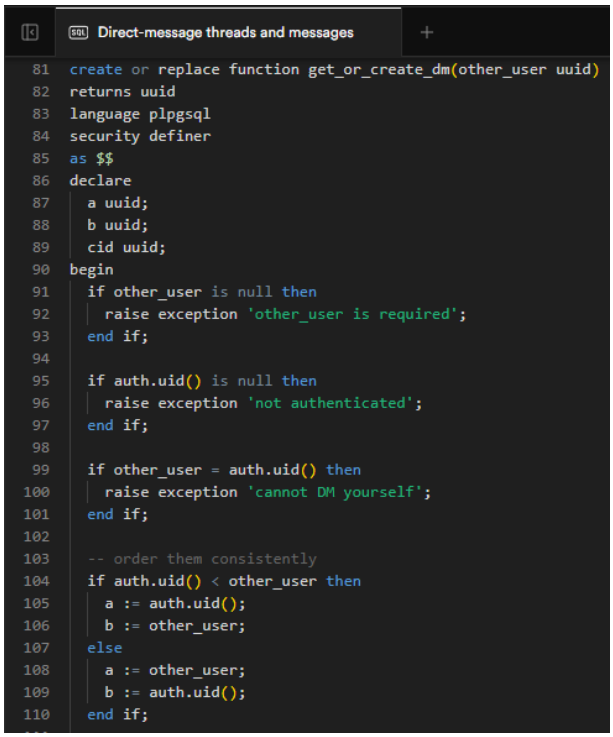

Figure 3: PostgreSQL function to retrieve or create a unique Direct Message (DM) conversation (get_or_create_dm)

This query does a few things. First, it creates a conversation record for two users (pair_key guarantees that the two users always end up in the same conversation, no matter what way the pair is formatted), then it adds constraints and indexes to the conversations tables to prevent users from messaging themselves and to ensure that there is only one conversation per user pair.

Next, the conversation_messages table is created to store every message, linked to a conversation and sender, to allow review and inspect of messages sent between users on the app.

Row Level Security is then enabled to prevent any authenticated users to be able to read any conversation in the database, allow users to read their own conversations, allow users to create conversations that they are part of, allow users to read messages in their conversations, and to allow users to send messages only as themselves.

Following this, the get_or_create_dm function is created. This conversation again guarantees only one conversation per user pair and retrieves a conversation between two users if it exists, otherwise creates it.

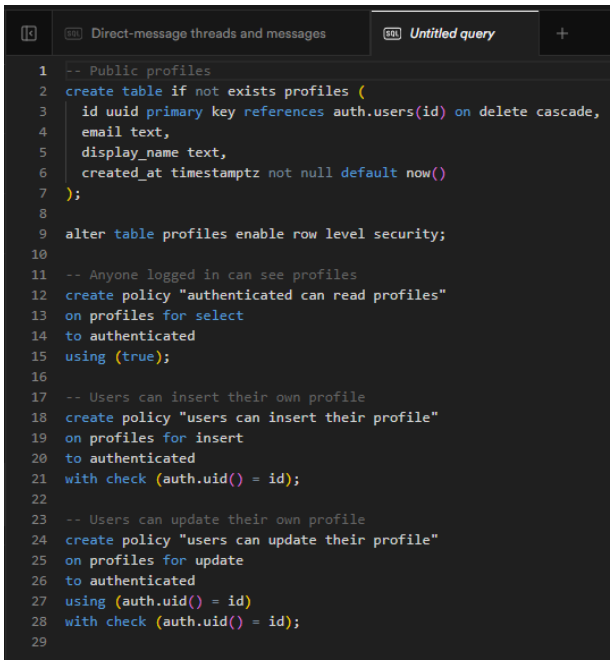

Figure 4: SQL query to create the User Profiles table and define RLS policies for profile access

The above query creates the profiles table and adds RLS policies to allow user discovery and to prevent users editing other profiles that aren't their own.

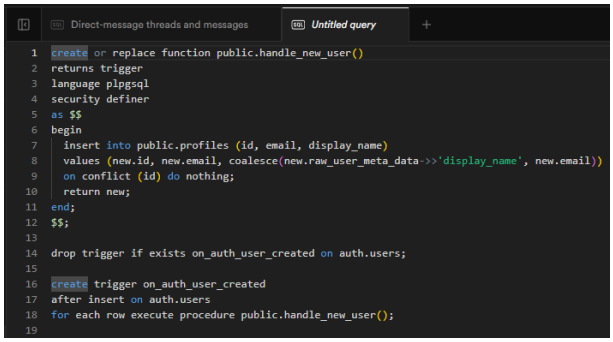

Figure 5: PostgreSQL function to automatically create a user profile upon successful sign-up

Here, a function is created to automatically create a profile for a user once they have successfully signed up to the app.

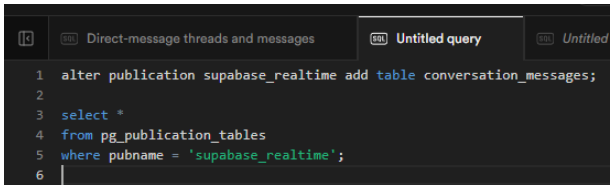

Figure 6: SQL statements enabling real-time message streaming on the conversation_messages table using Supabase Realtime

Finally, real time message streaming is enabled on the conversation_messages table to allow live message delivery. The following statements are just to confirm what tables are streamed in real time.

The messaging system was designed for 1:1 communication. Each conversation is uniquely associated with exactly two users, enforced through the database schema and constraints. Messages are stored in a dedicated relational table, with each message linked to both a conversation and a sender. This approach ensures full message logging, allowing all exchanges to be stored persistently and reviewed if/when required. The relational design aligns well with marketplace style applications where conversations may later be associated with transactions, listings or disputes.

In terms of security with the messaging system, PostgreSQL Row Level Security (RLS) policies were implemented to ensure that users can only read and send messages within conversations that they are a participant in. These access controls are enforced at the database level rather than solely in the application logic, reducing the risk of unauthorised data access.

To support real time communication, the app uses Supabase Realtime to subscribe to database insert events in the messaging table. This allows new messages to be delivered instantly to both participants without requiring the implementation of a WebSocket server.

The prototype app was tested using 2 devices logged into different accounts, allowing real time communication between independent instances of the app.

3.1 User Stories

As a User, I want to be able to scroll through and search current ticket listings so that I can quickly find tickets for the event I want.

As a User, I want to be able to sort listings based on criteria such as venue, location, price, etc. so that I can filter the marketplace effectively.

As a User, I want to view a reputation score on a user's profile so that I can assess their trustworthiness before engaging in a transaction.

As a User, I want to view a section for past purchases on my profile which I can toggle to be public or private so that I can keep track of my ticket history and show off to my friends.

As a Seller, I want to be able to list my ticket on the marketplace so I can resell it to another user.

As a Seller, I want to be able to upload information about the original purchase so that the system can verify the original cost and prevent me from listing the ticket above the original sale price.

As a Seller, I want to be able to edit and or delete tickets I am currently listing so that I can manage my listings if my plans change.

As a Buyer, I want to be able to open a chat with the Seller when I find a ticket so that I can negotiate a price and other details.

As a Seller, once a price is agreed upon, I want to be able to set the confirmed price within the conversation tab so that the Buyer can proceed with payment.

As a Seller, once the price is set, I want the system to hold the tickets in an active transaction so that I am unable to revoke the listing while the Buyer attempts payment.

As a Buyer, once the system has received my money, I want the ticket to be sent to me so that I have confirmed I will be able to make it to the event.

As Seller, once the system has received the money, I want the funds to be issued to my account so that I am paid for the ticket.

As a User, if negotiations fail, I want both parties to confirm they want to ceases the transaction so that the ticket listing is made available again.

As a Buyer, if I cancel the transaction after sending money, I want to be issued a refund even though I understand that it may take time to process.

As a Seller, if the Buyer is unresponsive or does not pay, I want to be able to abort the transaction so that the ticket listing is made available again.

As a User, once a transaction is ceased, I want to be able to rate the other party's performance so that the community benefits from the honest feedback.

As a User, I want my average reputation score to be displayed on my profile so that other users can assess my reliability.

3.2 Transaction Scenarios

3.2.1 Successful Transaction

This is the baseline straightforward transaction that is hoped to be achieved every time. A seller (e.g. Bob) lists a singular ticket for sale, ensuring the price is not above the original

sale price (enforced by the app). A buyer (e.g. Alice) finds the listing and initiates a messaging system with Bob to negotiate the price and other details. Once they agree on a price, Bob sets the confirmed price within the conversation tab. At this point, the system holds the ticket in an active transaction, making Bob unable to revoke the listing. Alice confirms the price and pays, with the payment handled by third-party systems like Strike or PayPal. Once the system receives the money, the ticket is sent to Alice, and the funds are issued to Bob's account. Upon completion, both parties are prompted to rate the other user based on things like politeness, ease of negotiation, etc. which then contributes to their average reputation score that is displayed on their profile.

3.2.2 Failed or Canceled Transactions

Negotiation Failure: Alice and Bob fail to agree on a price or other details during the negotiation phase. To close the message window, both parties must confirm that they want to cease the transaction. Both parties are then prompted to rate the other user.

Cancellation Pre-Payment: Either party cancels the transaction after the price has been set but BEFORE the funds have been transferred to the app. For example, Bob may wait an hour for Alice to pay and decides to abort the transaction due to unresponsiveness. The canceling party must confirm termination and provide a reason. If Bob cancels, Alice cannot communicate further, but can go back to the ticket listing and try to negotiate again.

Cancellation Post-Payment, Pre-Transfer: Alice sends the money, and the system confirms it is holding the funds. If Bob fails to send the ticket or respond, Alice may decide to abort the transaction. The system warns Alice that canceling now means she will not receive the money for a little bit while the refund is issued.

3.3 Current Limitations

- The current design of the app only allows for listings to be for a singular ticket
- All the transactions are only in Euro
- Should listings be location locked? i.e. should people from Ireland only see Irish listings and/or only be able to purchase events in Ireland?

- Once a transaction is completed, does the transaction have the listing details? Should the listing still exist or be deleted? How will listing the past purchases on user profiles work?
- How will the ticket/proof be uploaded, if there is something to upload?

3.4 Current Model Architecture & Data Flow

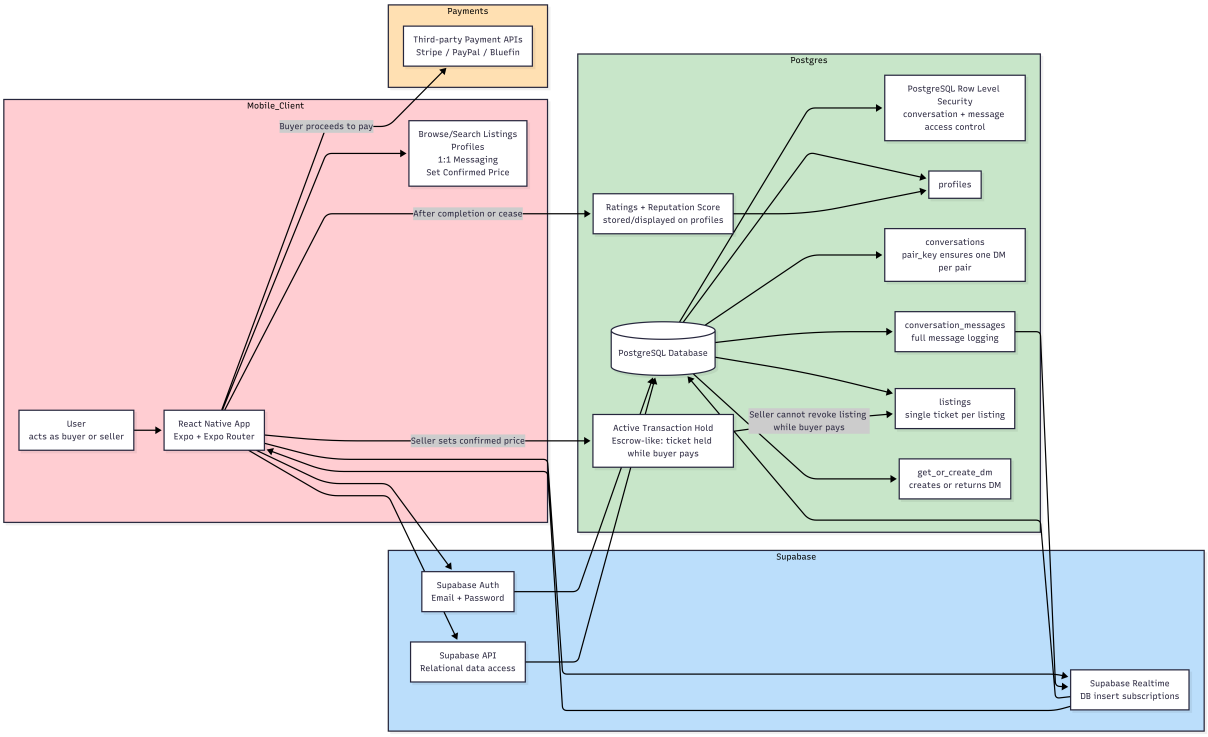

Figure 7: Database Schema Diagram showing relations between conversations, messages, and user profiles

Mobile Client:

- One type of user (can be both buyer and seller)
- Buyer and seller roles are contextual, not a different account or app - a user is a seller when they own a listing, and a buyer when they proceed to payment

Supabase:

- Handles user authentication using email and password
- Issues session tokens used in all subsequent requests to identify the user

- Supabase API is used to interact with the database for all operations including creating listings, messaging, and transactions
- Supabase Realtime is used to enable real time messaging by subscribing to database changes

PostgreSQL Database:

- Stores all application data including user profiles, listings, conversations, messages, and transactions

Payments:

- Third-party payment processors (e.g., Stripe, PayPal) are integrated to handle secure payment transactions
- The app communicates with the payment processor's API to initiate payments

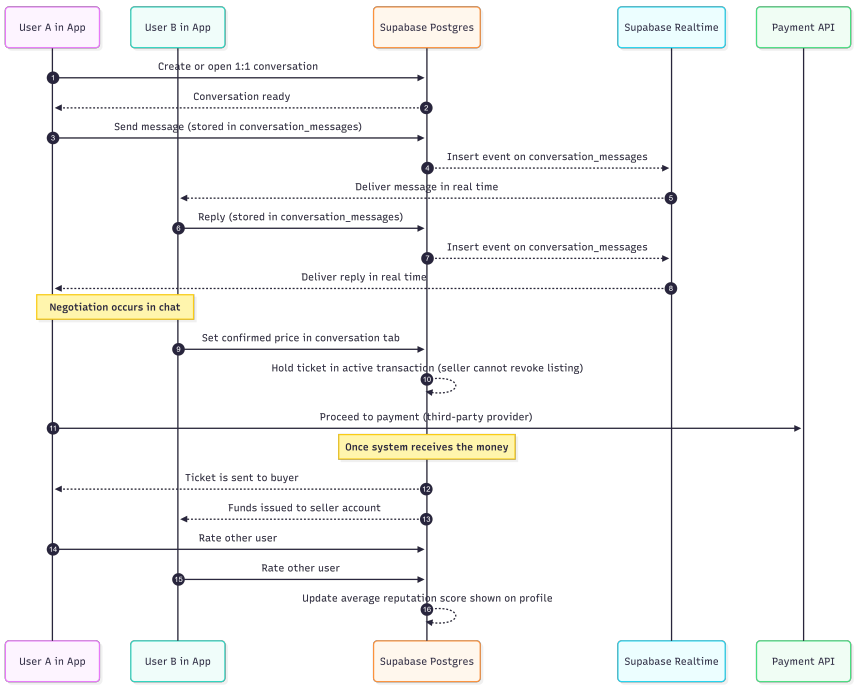

Figure 8: Data Flow Diagram illustrating the steps and components in a ticket transaction, from negotiation to payment

4 Demo Video

https://youtu.be/AzkoD4ebO8Y

5 Future Road Map

1. Core Marketplace and Listing Functionality
    - Develop listing creation, browsing, and searching features
    - Integrate price verification logic and original purchase validation
    - Implement sorting and filtering options
2. Implementing the Full Escrow Transaction Flow
    - Complete the payment integration with third-party processors
    - Develop the ticket transfer mechanism post-payment
    - Implement transaction state management (active, completed, canceled)
3. Post-Transaction Features
    - Develop the reputation system and rating mechanism
    - Implement user profiles with past purchase history
    - Resolve listing visibility and management post-transaction

References

API library [Bluefin]. (n.d.). Retrieved December 31, 2025, from https://www.bluefin.com/resources/api/

Converse. (n.d.). Retrieved December 31, 2025, from https://conversejs.org/

Escrow transactional method. (n.d.). Retrieved December 31, 2025, from https://ics.uci.edu/~cs223/papers/p405-o_neil.pdf

Expo [Expo]. (n.d.). Retrieved January 1, 2026, from https://expo.dev/

How does TicketSwap work? | is it safe to buy tickets online? [TicketSwap]. (n.d.). Retrieved December 31, 2025, from https://www.ticketswap.com/how-does-it-work

HOW TO USE TOUTLESS! PLEASE READ BEFORE POSTING. - toutless. (n.d.). Retrieved December 31, 2025, from https://www.toutless.com/viewtopic.php?f=1&t=10

PayPal developer. (n.d.). Retrieved December 31, 2025, from https://developer.paypal.com/home/

React native · learn once, write anywhere. (n.d.). Retrieved January 1, 2026, from https://reactnative.dev/

Sato, K. (2023). An analysis of a centralized exchange system for the ticket sales market. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.4665566

Stripe | financial infrastructure to grow your revenue. (n.d.). Retrieved December 31, 2025, from https://stripe.com/ie

Supabase | the postgres development platform. [Supabase]. (n.d.). Retrieved December 31, 2025, from https://supabase.com/

What is trustap? [Trustap]. (n.d.). Retrieved December 31, 2025, from https://www.trustap.com/what-is-trustap