Immersive

Authors: Phi Le, AK Nguyen, Aimen Faiz Rehman

Status: Completed

Last Update: Dec 10, 2024

Background

In today's fast-evolving e-commerce landscape, the increasing shift towards online buying and selling has highlighted the need for effective sales data management. A sales database serves as a crucial tool for tracking essential information, allowing businesses to analyze customer behavior and efficiently manage inventory. This empowers users—whether they are retailers, business owner, sellers, or other stakeholders—to make informed decisions, enhance customer experiences, optimize operational strategies, and better understand spending behavior for greater success

Overview

A comprehensive full-stack web application will be developed, specifically tailored for sales management, using Angular for the front-end, Node.js for the back-end, and MySQL for database management. The project aims to create an efficient platform that enables retailers to manage transactions, submit product reviews, track buyer and sales activities, and monitor quarterly revenue totals.

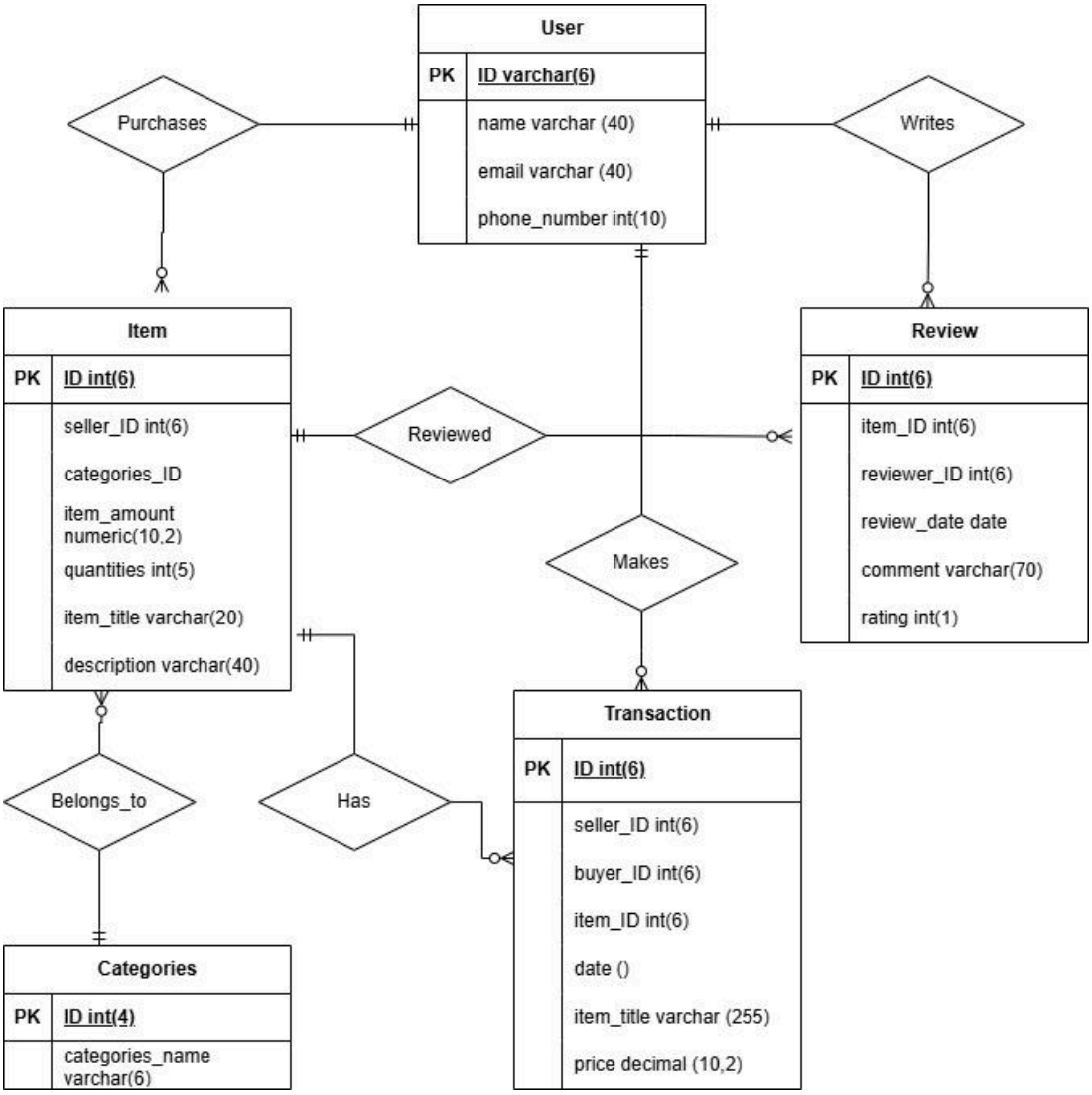
Upon project completion, an analysis of sales behavior will be conducted, including visualizations based on product categories and buying/selling patterns across different regions in the U.S. This analysis will employ various charts, such as pie charts and butterfly charts, to identify category preferences by region.

Our management system supports five categories, which can be expanded based on business needs: Clothing, Technology Products, Beauty & Grooming, Books, and Health Product.

ER Diagram

- **Category**(catergory_id, catergory_name)
    - **Primary Key:** catergory_id
- **Review** (review_id, item_id, reviewer_id, review_date, comment, rating)
    - **Primary Key:** review_id
    - **Foreign Key :** item_id reference item, reviewer_id reference user
- **Transaction**(trans_id, item_id, buyer_id, seller_id, date, item_title, price)
    - **Primary Key:** trans_id
    - **Foreign Key**: seller_id & buyer_id reference user, item_id reference item
- **User** (id, name, email, phone number)
    - **Primary Key:** id
- **Item**(item_id, item_amount, description, seller_id, catergory_id, quantity, item_title**)**
    - Primary Key: item_id
    - Foreign Key: seller_id reference user, catergory_id reference category

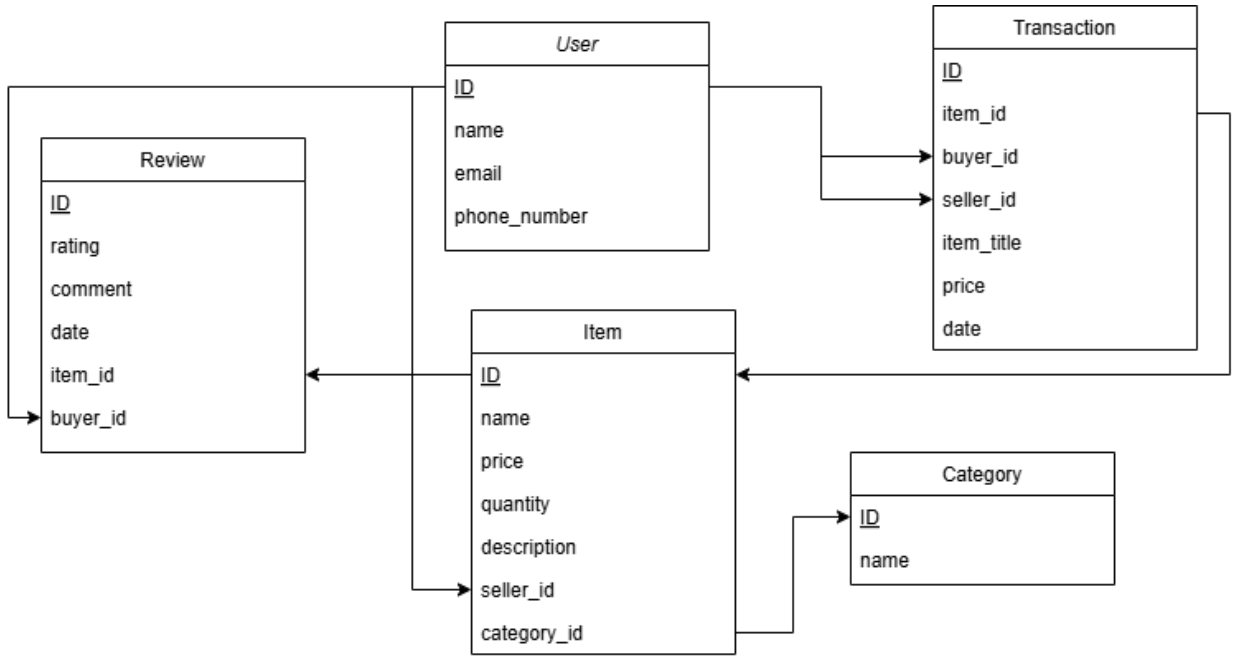
  Define Relationship:
    - **User** one to zero or many **Item**
    - **User** one to zero or many **Transaction**
    - **User** one to zero or many **Review**
    - **Item** one to zero or many **Transaction**
    - **Item** one to zero or many **Reviews**
    - **Item** one or many to one **Categories**



Database Tables

- **User**: stores information related to the user of our application
- **Item**: stores data relating the products being sold
    - The name will be indexed so that users can search for products
- **Review**: stores data on the review of products that customers purchase
- **Transaction**: stores data related to the sale of a product
    - Stores the item title and price so that editing or deleting an item does not affect transactions
- **Category**: stores data about the category of the items being sold
    - The name will be indexed so that users can search for items based on category

Relational Schema:



Backend

The backend development focuses on creating a secure and efficient server environment to handle business logic, data processing, and communication with the frontend and database. By leveraging Express.js, the backend will provide a flexible framework for building RESTful APIs that streamline data operations and enhance scalability.

Users can send requests to the back end directly from the frontend, allowing them to update, insert, and delete data within tables. They can also perform queries to retrieve data, ensuring accurate and efficient interaction with the database. This functionality enables smooth and seamless communication between the user interface and the underlying database, ensuring that data is managed effectively.

Database Management

MySQL will be implemented to store and organize sales data. The database will be designed to support quick access and retrieval of information, ensuring that users can analyze customer service behavior and manage inventory efficiently.

Server Operation

Express.js will handle server-side operations, managing API requests and facilitating communication with the MySQL database. This framework provides a lightweight and flexible solution, ensuring secure data handling and efficient processing of transactions.

User Authentication

Firebase Authentication will be used to manage user authentication. By integrating Firebase, we offload the complexity of securely managing sensitive data, such as passwords, to a trusted third-party service.

User Interaction Summary:

- Handles data-related requests from frontend
- Update, insert, and delete data in tables
- Query data
- User authentication for secure access

Frontend

The Angular framework will be utilized to create a user-friendly interface that allows users to navigate effortlessly through sales transactions and report generation. Features such as real-time updates and interactive dashboards will enhance the user experience.

User interaction on the platform encompasses several key features:

User Creation

Users are directed to the *Login Page* by default, where they must authenticate with their registered email and password. New users can register by providing their name, email, phone number, and password. Firebase Authentication handles user management and security, including password storage and authentication, relieving the backend of these responsibilities. After successful registration, a request is sent to the backend to add the user to the MySQL database, enabling functionality like transaction tracking and item postings.

Item Listings

After logging in, users are directed to the *Item Listings* page, where they can browse items, filter by category, and search using a query. The search prioritizes item names, ranking items with matching titles higher than those that match the category name. Each listing displays the item title, price, and available quantity, allowing users to quickly assess the products. These features ensure a seamless and efficient browsing experience, helping users find and evaluate items based on their preferences.

Item Details

Upon selecting an item from the listings, users are directed to the *Item Details* page, where they can view the item's title, description, price, quantity, seller, and rating. If the item is in stock, the user can purchase it, which triggers a backend request to update the item's quantity, create a transaction, and save the current price and title of the item to ensure that the transaction reflects the exact details at the time of purchase. Users can also view existing reviews and leave their own, which contribute to the item's overall rating.

Transactions

Users can visit the *Transaction Page* to view their transaction history, with transactions organized into two columns: Purchases and Sales. Each column displays the relevant transactions, and the subtotal for both purchases and sales is shown. At the bottom of the page, users can see a section that calculates and displays the difference between total sales revenue and total purchase expenses, providing an overview of their net balance.

User Listings

Users can visit the *My Listings* page to manage the items they are selling. They can create, edit, or delete their item listings. The page includes a search bar and category filters, similar to the *Item Listings* page, to help users easily find their own items. After making changes, the page makes a new request to the backend to retrieve and display the updated listings.

User Interaction Summary:

- Log in and register accounts
- Search for items
- Purchase items
- Leave reviews on items
- View transaction history
- Post items for sale

Data Analysis

The application will integrate with Tableau for advanced data visualization, enabling retailers to create insightful reports based on sales activities. Microsoft PowerPoint will also be utilized for presenting findings to stakeholders, enhancing strategic decision-making.

Testing

1. End User Testing

- Attempt to create a new user via the frontend. Use different email formats to check validation:
    - Valid: jane.doe@example.com
    - Invalid: jane.doeexample or @example.com
    - Duplicate email: Use an email already in the database.

- Test Login Functionality:
    - Valid credentials: alice.smith@example.com
    - Invalid credentials: Wrong email/password combination.

- Search and Filter Items:
    - Search for an item by title: **"Coffee Machine"**
    - Filter items by category: **"Electronics"**, **"Books"**.

- View Item Details:
    - Click on an item (e.g., "Samsung Smart TV") and confirm that the details, reviews, and ratings are displayed correctly.

- Submit a Review:
    - Add a review for an item and ensure it appears in the database.
    - Test edge cases for ratings (0, 6) and invalid characters in the comment.

- Perform a Transaction:
    - Attempt to buy an item, verify it reduces the item quantity, and creates a transaction record, when item stock hits zero transaction should not go through.
    - Have users create, edit and delete listings, verify that this exists within the database and on the frontend.

2. Database System Testing

Category Table Testing:

```
-- Query all categories:
SELECT * FROM Category;
-- Insert a duplicate category name to test unique constraints:
INSERT INTO Category (category_name) VALUES ('Electronics');
```

User Table Testing:

```
-- Fetch all users:
SELECT * FROM User;
-- Attempt to insert a user with a duplicate email:
INSERT INTO User (name, email, phone_number) VALUES ('John Doe',
'alice.smith@example.com', '123-456-7899');
```

Item Table Testing:

```
-- Fetch items with low stock (e.g., quantity < 5):
SELECT * FROM Item WHERE quantity < 5;
-- Check cascading updates/deletes by deleting a seller
DELETE FROM User WHERE id = 1;
-- Upon user deletion ensure user's items are also deleted
SELECT * FROM item WHERE seller_id = 1;
```

Review Table Testing:

```
-- Fetch all reviews with ratings > 4:
SELECT * FROM Review WHERE rating > 4;
-- Check cascading updates/deletes by deleting a seller
DELETE FROM User WHERE id = 5;
-- Upon user deletion ensure user's reviews are also deleted
SELECT * FROM review WHERE reviewer_id = 5;
```

Transaction Table Testing

```
-- Validate that all transactions reference existing users and items:
SELECT * FROM Transaction
WHERE buyer_id NOT IN (SELECT id FROM User)
   OR seller_id NOT IN (SELECT id FROM User)
   OR item_id NOT IN (SELECT item_id FROM Item);
```

3. End-to-End Testing

Startup Testing

- Verify the backend starts without errors:

- In the backend folder, run ‘npm run dev’
    - Verify items are accessible http://localhost:8080/api/item
    - Verify users are accessible http://localhost:8080/api/user

- Verify the frontend starts up without an error.

- In the frontend folder, run ‘npm start’

    - Verify the frontend is accessible at http://localhost:4200/

- Verify both the frontend and backend can run together

- In the project's root folder, run 'npm start'
    - Verify items are showing up on http://localhost:4200/items

API Endpoint Testing

Use a tool like Postman to test backend endpoints. Examples:

Get All Items

GET http://localhost:8080/api/item

Add Review

POST http://localhost:8080/api/review/create

```
{
  "item_id": 1,
  "reviewer_id": 2,
  "comment": "Amazing quality!",
  "rating": 5
}
```

Perform Transaction

POST http://localhost:8080/api/transaction/create

```
{
  "itemId": 3,
  "buyerId": 2
}
```

Frontend-Backend Communication

- Modify an item in the database directly and ensure the frontend reflects these changes.

Error Handling Tests

- Try to access an API endpoint without necessary data.
- Use incorrect HTTP methods (e.g., GET instead of POST) to test error responses.

User Manual

Running the Application

1. Install NodeJS

   https://nodejs.org/en
2. Clone repository

   git clone https://github.com/Phizzle32/immersive.git
3. Navigate into project directory

   cd immersive
4. Setup database
5. Install dependencies for frontend and backend

   npm run install-all
6. Create .env file with the following content in the root directory

   ```
   MYSQL_HOST='localhost'
   MYSQL_USER='root'
   MYSQL_PASSWORD='' # Use your mysql password here
   MYSQL_DATABASE='immersive'
   ```
7. In the root folder, startup frontend and backend by running

   npm start
8. Navigate to the website

   http://localhost:4200/

Database setup

1. Download MySQL

   https://dev.mysql.com/downloads/mysql/
2. Open up the MySQL CLI
3. Run the following command to set up tables

```sql
CREATE DATABASE immersive;
USE immersive;
-- 1. Create the Category Table
CREATE TABLE Category (
  category_id INT AUTO_INCREMENT,
  category_name VARCHAR(100) UNIQUE NOT NULL,
  PRIMARY KEY (category_id),
  INDEX (category_name)
);
-- 2. Create the User Table
CREATE TABLE User (
  id INT AUTO_INCREMENT,
  name VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
  phone_number VARCHAR(20),
  PRIMARY KEY (id)
);
-- 3. Create the Item Table
CREATE TABLE Item (
  item_id INT AUTO_INCREMENT,
  item_amount DECIMAL(10, 2) NOT NULL,
  description TEXT,
  seller_id INT,
  category_id INT,
  quantity INT NOT NULL,
  item_title VARCHAR(255) NOT NULL,
  PRIMARY KEY (item_id),
  INDEX (item_title),
  FOREIGN KEY (seller_id) REFERENCES User(id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (category_id) REFERENCES Category(category_id)
    ON DELETE SET NULL ON UPDATE CASCADE
);
-- 4. Create the Review Table
CREATE TABLE Review (
  review_id INT AUTO_INCREMENT,
  item_id INT,
  reviewer_id INT,
  review_date DATE NOT NULL,
```

```
  comment TEXT,
  rating INT CHECK (rating BETWEEN 1 AND 5),
  PRIMARY KEY (review_id),
  FOREIGN KEY (item_id) REFERENCES Item(item_id)
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (reviewer_id) REFERENCES User(id)
    ON DELETE CASCADE ON UPDATE CASCADE
);
-- 5. Create the Transaction Table
CREATE TABLE Transaction (
  trans_id INT AUTO_INCREMENT,
  item_id INT,
  item_title VARCHAR(255) NOT NULL,
  price DECIMAL(10, 2) NOT NULL,
  buyer_id INT,
  seller_id INT,
  date DATE NOT NULL,
  PRIMARY KEY (trans_id),
  FOREIGN KEY (item_id) REFERENCES Item(item_id)
    ON DELETE SET NULL ON UPDATE CASCADE,
  FOREIGN KEY (buyer_id) REFERENCES User(id)
    ON DELETE SET NULL ON UPDATE CASCADE,
  FOREIGN KEY (seller_id) REFERENCES User(id)
    ON DELETE SET NULL ON UPDATE CASCADE
);
```

4. Insert some data into the tables

```
INSERT INTO Category (category_name) VALUES
('Electronics'),
('Fashion'),
('Home & Kitchen'),
('Books'),
('Toys & Games');

INSERT INTO User (name, email, phone_number) VALUES
('Alice Smith', 'alice.smith@example.com', '123-456-7890'),
('Bob Johnson', 'bob.johnson@example.com', '234-567-8901'),
('Charlie Davis', 'charlie.davis@example.com', '345-678-9012'),
```

```
('David Lee', 'david.lee@example.com', '456-789-0123'),
('Eva White', 'eva.white@example.com', '567-890-1234');

INSERT INTO Item (item_amount, description, seller_id, category_id, quantity, item_title)
VALUES
(299.99, '4K Ultra HD Smart TV', 1, 1, 10, 'Samsung Smart TV'),
(49.99, 'Black leather jacket', 2, 2, 5, 'Leather Bomber Jacket'),
(39.99, 'Coffee maker', 3, 3, 15, 'Deluxe Coffee Machine'),
(15.99, 'Popular mystery novel', 4, 4, 20, 'The Silent Patient'),
(29.99, 'Electric scooter', 5, 5, 8, 'Zippy Kids Electric Scooter');

INSERT INTO Review (item_id, reviewer_id, review_date, comment, rating) VALUES
(1, 2, '2024-10-01', 'Great TV, very easy to set up!', 5),
(2, 1, '2024-09-25', 'The jacket looks great, but a bit tight', 3),
(3, 4, '2024-10-05', 'Makes amazing coffee.', 4),
(4, 5, '2024-10-07', 'Loved this book! A real page-turner!', 5),
(5, 3, '2024-09-30', 'It is perfect for outdoor fun.', 4);

INSERT INTO Transaction (item_id, item_title, price, buyer_id, seller_id, date) VALUES
(1, '4K Ultra HD Smart TV', 299.99, 3, 1, '2024-10-02'),
(2, 'Black leather jacket', 49.99, 5, 2, '2024-09-28'),
(3, 'Coffee maker', 39.99, 2, 3, '2024-10-06'),
(4, 'Popular mystery novel', 15.99, 1, 4, '2024-10-08'),
(5, 'Electric scooter', 29.99, 4, 5, '2024-09-30');
```

Running the Backend Individually

1. Navigate into backend folder

   cd backend

2. Install dependencies

   npm install

3. Create .env file in the root directory with the following contents

```
MYSQL_HOST='localhost'
MYSQL_USER='root'
MYSQL_PASSWORD='' # Use your mysql password here
MYSQL_DATABASE='immersive'
```

4. Startup server

npm run dev

Running the Frontend Individually

1. Navigate into backend folder
   cd frontend
2. Install dependencies
   npm install
3. Startup server
   npm start

Contributions

In the development of **Immersive**, our team collaborated closely to ensure the project's success. Aimen Faiz Rehman's primary responsibility was designing the database, focusing on optimizing data storage and retrieval to support the application's functionality. Khoi Nguyen concentrated on creating an engaging presentation and streamlining the workflow to enhance project efficiency. Phi Le contributed by designing the interface, building the backend, establishing API endpoints and setting up communication between the frontend and backend. He also ensured the website was visually appealing and functional. Together, we all worked on preparing the final report, ensuring comprehensive documentation of our work and findings.