**Data Model Diagram:**

To start with in this phase, we created our data model structure that will handle our data, primarily organized around the User entity which serves as a foundational element connecting various roles including buyers, sellers, and administrators. Each User has a unique identifier which is essential for distinguishing user roles, enabling interactions across the system. Users are categorized into three roles: customers, sellers, and admins, with each category linked to specific functionalities within the system.

Admins are special users with administrative privileges, identified through the Admin entity that refers back to the User entity, underscoring that all admins are users, but with additional capabilities to manage the system. Sellers have a direct relationship with the User entity as well, with attributes such as account balance and a link to the books they are selling. This setup allows sellers to manage their inventory and transactions effectively.

The Book entity is central to the system, with each book linked to its seller and transactions, facilitating the commerce of books. Books have detailed attributes like title, author, price, and genre, making it easy to manage listings and sales. Transactions capture the exchange details, linking customers, books, and the transaction specifics such as amount and date, crucial for financial management and customer service.

A screenshot of a computer

Description automatically generatedCustomer, another role derived from the User entity, have their purchasing power depicted through attributes like account balance and shipping addresses, indicating the user-centric design of the system. Each customer's interaction with the system is tracked through transactions, providing a comprehensive view of customer activity.

**Contribution:**

Omar's contribution

Our second phase commenced with rigorous discussions about our database schema. Once we finalized the design of our models, we advanced to setting up our repository. My specific contribution was to generate functions related to transactions. As we developed our API, I spearheaded the transactions API and actively resolved issues collaboratively whenever they arose. As the project progressed, I also took charge of the statistical aspects, creating charts that displayed transaction counts per seller and average quantity sold per book and finally customer count.

Khalifa's Contribution

In the crucial phase of our project, distribution posed greater challenges compared to the initial phase, largely due to the stringent schedule constraints. After completing the lecture content, we, as a group, engaged in extensive discussions to devise the most suitable database scheme for our project. Subsequently, we established our repository, where I took on the task of writing several Prisma queries. I developed key methods including getUser Login and createBook. Moving into the second stage, our strategic focus was on developing our API, starting with the login API, followed by the book API, and finally, the transaction API. For our statistical analysis, we chose to implement it using Chart.js, where I was responsible for creating charts such as the top sales book chart and average purchase amount per buyer. Despite the challenges, we successfully navigated this phase of the project.

Ebrahim's Role

The initial step involved deep discussions about our database schema until we settled on the most appropriate configuration that met our project's needs. Team collaboration was a constant throughout every stage of this project. In the repository, I managed functions such as `deleteBook`, `getBook`, and updates for `customer` and `book`. The API development was a significant phase where I mainly focused on the book API, ensuring its robustness and efficiency. In the latter part of this phase, I developed statistical insights by calculating the total revenue per seller each month, a critical metric for our business analysis.

**Database Queries:**

In our project, we used a variety of queries to handle our data as we shifted from a localStorage-based website to using a real database with Prisma. These queries were crucial for effectively managing our data. We used 'FindMany' to get multiple records that matched specific criteria and 'FindUnique' to fetch a single, distinct record. The 'Create' query helped us add new data, while 'Count' gave us useful information by tallying the number of records meeting certain conditions. 'Delete' allowed us to remove records, and 'Update' let us change existing data entries when necessary. With 'GroupBy', we could organize and analyze data based on specified attributes, and 'FindFirst' helped us retrieve the initial record meeting specific criteria. These queries empowered us to interact efficiently with our database and meet our project's functional requirements. This prompted us to transition all six use cases from phase 1, which previously relied on localStorage, to Prisma. Additionally, it facilitated the implementation of an extra use case in phase 2: statistics. We were able to gather six distinct statistics for the admin user, providing valuable insights into the activity on their website.

These are the functions we used inside our readradar\_repo that included our database queries mentioned above.

* **getUser:** This method retrieves user information from the database, including details about admins, sellers, and customers, utilizing the Prisma findUnique query.
* **getSeller:** Retrieves information about a specific seller from the database using their unique identifier, leveraging the Prisma findFirst query.
* **Login:** Validates a user's credentials (username and password) for authentication, utilizing the getUser method internally. If valid, it returns their user information; otherwise, it throws an error.
* **getBooks:** Fetches a list of books from the database, with optional additional data included, utilizing the Prisma findMany query with specified criteria and additional included data.
* **createBook:** Adds a new book entry to the database with the provided information using the Prisma create query.
* **UpdateBook:** Updates the information of an existing book entry in the database based on the provided data, utilizing the Prisma update query.
* **UpdateCustomer:** Modifies the details of a customer in the database according to the provided information, utilizing the Prisma update query.
* **updateSeller:** Updates the information of a seller in the database based on the provided data, leveraging the Prisma update query.
* **updateAddress:** Updates the shipping address information for a customer in the database, utilizing the Prisma update query.
* **deleteBook:** Removes a book entry from the database using its unique identifier, utilizing the Prisma delete query.
* **createTransaction:** Records a new transaction in the database when a user purchases a book, updating relevant data, accordingly, utilizing multiple Prisma queries including create and update.
* **getTransaction:** Retrieves transaction details from the database, including associated book and customer information, utilizing the Prisma findMany query with specified criteria and included data.
* **getTopBooksBySales:** Retrieves the top-selling books from the database based on sales volume, utilizing the Prisma groupBy query with specified criteria for ordering and limiting results.
* **getAverageQuantitySoldPerBook:** Calculates and retrieves the average quantity sold per book from the database using the Prisma groupBy query with aggregation functions like \_avg.
* **getAveragePurchaseAmountPerBuyer:** Computes and retrieves the average purchase amount per buyer from the database using the Prisma groupBy query with aggregation functions like \_avg.
* **countTransactionsPerSeller:** Counts the number of transactions associated with each seller in the database using multiple Prisma queries including findMany and aggregation functions like \_sum and another query called orderBy that we sort the data into ascending order.
* **getTotalRevenuePerMonth:** Calculates and retrieves the total revenue generated per month from transactions in the database using the Prisma groupBy query with aggregation functions like \_sum.
* **getSellerCount:** Retrieves the count of sellers stored in the database using the Prisma count query.
* **getCustomerCount:** Retrieves the count of customers stored in the database using the Prisma count query.