**Book Keeping App Design Document**

1. **Introduction**

Bookkeeping application is crafted to manage tasks for businesses and individuals alike. Seamlessly recording transactions, tracking expenses, and generating insightful reports are just a few clicks away. Built on the robust foundation of the MERN stack (MongoDB, Express.js, React.js, Node.js), coupled with Tailwind CSS for a sleek frontend design, it offers an intuitive and efficient solution for organizing and analyzing financial data.

**2. Features**

**Customer-Facing Features:**

1. **User Registration:** Customers can easily sign up for an account on the platform, providing their basic information such as username, email, and password.
2. **User Login:** Once registered, customers can log in securely using their credentials, gaining access to their personalized dashboard and the app's features.
3. **Profile Management:** Users can manage their profiles, including updating their personal information such as name, email, and password.
4. **Book Creation:** Customers can create and manage books within their account, adding details such as title, author, and publication year.
5. **Book Listing:** The app provides customers with a comprehensive list of books, allowing them to browse and search for specific titles or authors.
6. **Book Update and Deletion:** Users can update or delete books from their catalog as needed, ensuring their inventory stays current.
7. **Dashboard:** Customers are greeted with a dashboard upon logging in, providing them with an overview of their account activity and access to key features such as book management and profile settings.
8. **Authentication and Authorization:** The app ensures secure authentication and authorization, safeguarding customer accounts and data from unauthorized access.

**3. Admin Features:**

1. **User Management:** Admins can manage user accounts, including creating, updating, and deleting user profiles as necessary.
2. **Book Management:** Admins can oversee and manage the entire catalog of books within the system, including creating, updating, and deleting book entries.
3. **Dashboard:** Admins have access to a comprehensive dashboard that provides an overview of user activity, book inventory, and other relevant metrics.
4. **User Role Assignment:** Admins can assign roles and permissions to users, granting specific privileges such as access to certain features or administrative capabilities.
5. **Data Analytics:** Admins have access to advanced data analytics tools and reports, allowing them to analyze user behavior, track book sales, and make informed decisions to optimize the platform's performance.
6. **System Configuration:** Admins can configure various settings and parameters within the system, such as email notifications, security policies, and application preferences.
7. **Audit Logs:** The system maintains detailed audit logs of all user activities and system events, providing admins with visibility into actions performed within the platform.
8. **Security Management:** Admins are responsible for overseeing the security of the application, implementing security protocols, and ensuring compliance with data protection regulations.

**4. Architecture View**

The architecture of the UrbanKicks website follows a typical MERN stack architecture:

- Frontend: React.js for building the user interface.

- Backend: Node.js and Express.js for handling server-side logic and API requests.

- Database: MongoDB for the database system, Mongoose for object modeling in Node.js.

- Deployment: The application is deployed on a cloud platform such as AWS, Azure, or Heroku.

User/Client User Interface Auth Layer Express Middleware Server Database

User DB

Book DB

**5. Database Design**

The MongoDB database is structured to support the following collections:

1. Books: Stores information about books including id, category, author, title,

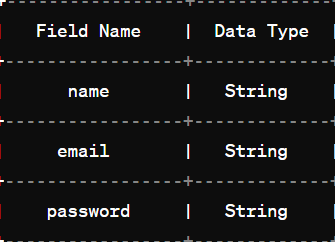
ccreatedBy, createdAt and updatedAt.

BookSchema



1. Users: Contains user account details such as username, email, password hash, and role.

UserSchema



**6. API Design**

The API routes are designed to handle various operations related to books and users. These routes are secured using authentication middleware for sensitive operations.

**BOOK Routes**

* **GET /api/books**

- Description: Get all books

- Response:

- 200 OK: Returns an array of book objects

* **GET /api/books/:id**

- Description: Get a book by ID

- Response:

- 200 OK: Returns the book object

- 404 Not Found: If the book with the given ID is not found

* **POST /api/books**

- Description: Create a new book

- Request Body:

- category: String (required)

- author: String (required)

- title: String

- Response:

- 201 Created: Returns the created book object

- 400 Bad Request: If the request body is missing required fields or has invalid data

* **PUT /api/books/:id**

- Description: Update a book

- Request Body:

- category: String

- author: String

- title: String

- Response:

- 200 OK: Returns the updated book object

- 404 Not Found: If the book with the given ID is not found

- 400 Bad Request: If the request body has invalid data

* **DELETE /api/books/:id**

- Description: Delete a book

- Response:

- 204 No Content: If the book is successfully deleted

- 404 Not Found: If the book with the given ID is not found

**User Routes**

* **GET /api/users**

- Description: Get all users

- Response:

- 200 OK: Returns an array of user objects

* **GET /api/users/:id**

- Description: Get a user by ID

- Response:

- 200 OK: Returns the user object

- 404 Not Found: If the user with the given ID is not found

* **POST /api/users/register**

- Description: Register a new user

- Request Body:

- name: String (required)

- email: String (required)

- password: String (required)

- Response:

- 201 Created: Returns the created user object

- 400 Bad Request: If the request body is missing required fields or has invalid data

* **POST /api/users/login**

- Description: Login user

- Request Body:

- email: String (required)

- password: String (required)

- Response:

- 200 OK: Returns the logged-in user object with a JWT token

- 401 Unauthorized: If the email or password is incorrect

* **PUT /api/users/:id**

- Description: Update user information

- Request Body:

- name: String

- email: String

- password: String

- Response:

- 200 OK: Returns the updated user object

- 404 Not Found: If the user with the given ID is not found

- 400 Bad Request: If the request body has invalid data

* **DELETE /api/users/:id**

- Description: Delete user

- Response:

- 204 No Content: If the user is successfully deleted

- 404 Not Found: If the user with the given ID is not found

**7. Frontend Design**

The front end of the book-keeping app is designed using React.js to create a modern and visually appealing user interface. Components are organized hierarchically to ensure reusability and maintainability.

**The directory structure for Frontend and Backend respectively**

