To build a comprehensive Library Management System API with the requirements you outlined, you'll need to create several endpoints. These endpoints will handle user authentication, book management, borrowing operations, complex querying, and statistics. Here's a breakdown of the endpoints you'll need:

1. User Authentication

- **Register User**
  - **POST** `/api/register/`
  - **Description**: Register a new user.
  - **Request Body**: `{"username": "string", "password": "string"}`
- **Obtain Token**
  - **POST** `/api/login/`
  - **Description**: Obtain a token for authenticated access.
  - **Request Body**: `{"username": "string", "password": "string"}`
- **Logout (Revoke Token)**
  - **POST** `/api/logout/`
  - **Description**: Revoke the current user's token.
  - **Headers**: `Authorization: Token <token>`

2. Book Management (Admin Only)

- **List All Books**
  - **GET** `/api/books/`
  - **Description**: Get a list of all books in the library.
- **Retrieve Book Details**
  - **GET** `/api/books/<int:id>/`
  - **Description**: Get details of a specific book.
- **Add a New Book**
  - **POST** `/api/books/`
  - **Description**: Add a new book to the library.
  - **Request Body**: `{"title": "string", "author": "string", "genre": "string", "publication_date": "YYYY-MM-DD", "total_copies": integer, "available_copies": integer}`
- **Update Book Details**
  - **PUT** `/api/books/<int:id>/`
  - **Description**: Update details of a specific book.
  - **Request Body**: `{"title": "string", "author": "string", "genre": "string", "publication_date": "YYYY-MM-DD", "total_copies": integer, "available_copies": integer}`
- **Delete a Book**
  - **DELETE** `/api/books/<int:id>/`
  - **Description**: Delete a specific book from the library.

3. Borrowing Management

- **Borrow a Book**
  - **POST** `/api/borrow/`
  - **Description**: Borrow a book (decreases available copies).
  - **Request Body**: `{"book_id": integer}`
- **Return a Book**
  - **POST** `/api/return/`
  - **Description**: Return a book (increases available copies).
  - **Request Body**: `{"book_id": integer}`
- **User's Borrowing History**
  - **GET** `/api/borrow-history/`
  - **Description**: View the borrowing history of the logged-in user.
  - **Headers**: `Authorization: Token <token>`

4. Complex Queries

- **Filter Borrowed Books**
  - **GET** `/api/borrow-history/filter/`
  - **Description**: Filter borrowed books by genre, author, and date range.
  - **Query Parameters**: `?genre=string&author=string&start_date=YYYY-MM-DD&end_date=YYYY-MM-DD`
  - **Headers**: `Authorization: Token <token>`
- **Books Available for Borrowing by Genre**
  - **GET** `/api/books/available-by-genre/`
  - **Description**: Retrieve books by genre showing only those with available copies.
  - **Query Parameters**: `?genre=string`
- **Book Statistics**
  - **GET** `/api/statistics/`
  - **Description**: Retrieve statistical data about the library.
  - **Query Parameters**:
    - `?start_date=YYYY-MM-DD&end_date=YYYY-MM-DD`
  - **Response**: `{ "books_borrowed": integer, "most_borrowed_books": [{ "book": "string", "author": "string", "count": integer }], "most_active_borrowers": [{ "user": "string", "borrowed_count": integer }], "average_borrow_duration": "X days" }`

5. Admin Control

- **List All Users**
  - **GET** `/api/users/`
  - **Description**: List all users (admin only).

- **Retrieve User Details**
  - **GET** `/api/users/<int:id>/`
  - **Description**: Get details of a specific user (admin only).
- **Update User Details**
  - **PUT** `/api/users/<int:id>/`
  - **Description**: Update details of a specific user (admin only).

6. Token Expiry and Revocation

- **Revoke Token**
  - **POST** `/api/logout/`
  - **Description**: Revoke the token of the logged-in user.
  - **Headers**: `Authorization: Token <token>`

Bonus (Optional)

- **Rate Limiting (Borrow Requests)**
  - Implement logic in the borrow view to limit the number of requests per hour per user. This may require custom middleware or view logic.
- **Unit Tests**
  - Write unit tests to ensure token authentication and query correctness.

Implementation Tips

- Use Django REST Framework's `TokenAuthentication` for authentication.
- Use `django_filters` for complex querying and filtering.
- Utilize Django's ORM methods like `annotate()`, `aggregate()`, `Count()`, and `F()` to handle complex queries and analytics.
- Implement token expiration using `djangorestframework-simplejwt` or similar libraries if needed.
- Use Django's built-in `admin` interface for easy admin management or create custom admin endpoints.

This setup will cover all your requirements and ensure that your Library Management System API is robust, secure, and functional.