Assignment: Library Management System with Token Authentication and Complex Querying

You are tasked with developing a Library Management System API where users can manage books, borrow them, and see various statistics. The system should use token-based authentication to control access, and include complex querying for filtering and analytics.

Project Requirements:

1. **Models Setup: You need to define the following models:**
    - **Author: Represents the author of a book with fields such as `name`, `birthdate`, and `nationality`.**
    - **Book: Represents a book with fields such as `title`, `author` (ForeignKey to `Author`), `genre`, `publication_date`, `available_copies` (number of copies available for borrowing), and `total_copies`.**
    - **Borrow: Represents the borrowing of a book with fields like `user` (ForeignKey to Django's `User` model), `book` (ForeignKey to `Book`), `borrow_date`, `return_date`, and `is_returned`.**
2. **Each model should include the appropriate relationships and constraints.**
3. **Token Authentication:**
    - **Implement token-based authentication so that each user has their own token for accessing the system.**
    - **A user should be able to borrow and return books, but only when authenticated using their token.**
    - **Only authenticated users can borrow books, and users must be able to view their borrowing history using complex queries.**
4. **Borrowing Logic:**
    - **A user should not be able to borrow a book if all copies are already borrowed. Decrease the `available_copies` count when a book is borrowed, and increase it when the book is returned.**
    - **Add validation to ensure that users cannot borrow more than one copy of the same book at the same time.**
    - **Allow users to view only the books they have borrowed using complex queries that also show whether they have returned the books or not.**
5. **Advanced Querying:**
    - **Filter Borrowed Books: Create an API that allows users to filter the books they have borrowed by `genre`, `author`, and `borrow date range`. Use complex queries to retrieve this data efficiently.**
    - **Book Statistics: Create an endpoint that returns statistical data about the library:**
        - **Number of books borrowed in a given time period.**
        - **The most borrowed books and their authors.**

        - The most active borrowers (users who have borrowed the most books).
        - Average time a book stays borrowed before being returned.
6. Admin Control:
    - An admin should have the ability to add or update books, including adjusting the `total_copies` and `available_copies`.
    - Admins should be able to view a list of all books and the users who have borrowed them, along with the `borrow_date` and `return_date`.
7. Token Expiry and Revocation:
    - Implement token expiration logic (e.g., tokens should expire after 1 hour).
    - Add an API endpoint for users to manually log out (revoke their tokens).
    - Return a 401 Unauthorized status if a user tries to use an expired token.
8. Complex Query: Books Available for Borrowing by Genre:
    - Create an API that retrieves all books by genre, showing the books that currently have available copies for borrowing. This query should also show the total number of available copies for each genre.
9. Bonus (Optional):
    - Implement a rate-limiting feature to limit the number of borrow requests a user can make per hour.
    - Add unit tests for token authentication and the complex queries to ensure that unauthorized access is denied and the correct data is returned for authenticated users.

Hints:

- For token-based authentication, use `TokenAuthentication` from `rest_framework.authtoken`.
- Use advanced Django ORM queries such as `annotate()`, `aggregate()`, `Count()`, and `F()` for efficient querying and filtering.
- For token expiration, you can use a custom implementation or integrate with `djangorestframework-simplejwt`.

This assignment is designed to test your ability to handle complex token-based authentication scenarios combined with advanced database querying, validation logic, and performance optimization. Let me know if you need any additional clarification!