Django Expense Tracker API - Intern Task

Project Overview

Goal: Build a REST API for expense/income tracking with user authentication

Core Requirements

User Access Control

- **Regular Users**: Can only manage their own expense/income records
- **Superusers**: Can access and manage all users' records
- **Authentication**: JWT tokens required for all API operations

Key Features

- User registration and login system
- Personal expense/income tracking
- Automatic tax calculation (flat amount or percentage)
- Paginated API responses
- Complete CRUD operations

Database Models

User Model

Use Django's built-in User model (no custom fields needed)

ExpenseIncome Model

Fields to implement:

- `user` → ForeignKey to User
- `title` → CharField (max 200 chars)
- `description` → TextField (optional)
- `amount` → DecimalField (10 digits, 2 decimal places)
- `transaction_type` → CharField (choices: 'credit', 'debit')
- `tax` → DecimalField (default 0)
- `tax_type` → CharField (choices: 'flat', 'percentage', default 'flat')
- `created_at` → DateTimeField (auto)
- `updated_at` → DateTimeField (auto)

Business Logic:

- Flat Tax: `Total = Amount + Tax`
- Percentage Tax: `Total = Amount + (Amount × Tax ÷ 100)`

🔗 API Endpoints

Authentication Endpoints

- `POST /api/auth/register/` → User registration
- `POST /api/auth/login/` → User login (returns JWT tokens)
- `POST /api/auth/refresh/` → Refresh JWT token

Expense/Income Endpoints

- `GET /api/expenses/` → List user's records (paginated)
- `POST /api/expenses/` → Create new record
- `GET /api/expenses/{id}/` → Get specific record
- `PUT /api/expenses/{id}/` → Update record
- `DELETE /api/expenses/{id}/` → Delete record

Expected API Response Formats

Single Record Response

```
{
    "id": 1,
    "title": "Grocery Shopping",
    "description": "Weekly groceries",
    "amount": 100.00,
    "transaction_type": "debit",
    "tax": 10.00,
    "tax_type": "flat",
    "total": 110.00,
    "created_at": "2025-01-01T10:00:00Z",
    "updated_at": "2025-01-01T10:00:00Z"
}
```

List Response (Paginated)

```
{
    "count": 25,
    "next": "http://api/expenses/?page=2",
    "previous": null,
    "results": [
        {
            "id": 1,
            "title": "Grocery Shopping",
            "amount": 100.00,
            "transaction_type": "debit",
            "total": 110.00,
            "created_at": "2025-01-01T10:00:00Z"
        }
    ]
}
```

🔧 Technical Requirements

Technologies to Use

- **Backend**: Django + Django REST Framework
- **Authentication**: JWT (djangorestframework-simplejwt)
- **Database**: SQLite (for development)
- **Python Version**: 3.8+

HTTP Status Codes

- `200 OK` → Successful GET, PUT
- `201 Created` → Successful POST
- `204 No Content` → Successful DELETE
- `400 Bad Request` → Invalid data
- `401 Unauthorized` → Authentication required
- `403 Forbidden` → Permission denied
- `404 Not Found` → Resource not found

Testing Checklist

Authentication Tests

- User registration with valid data
- User registration with duplicate email/username
- User login with valid credentials
- User login with invalid credentials
- Token refresh functionality
- Access protected endpoint with valid token
- Access protected endpoint without token

CRUD Operations Tests

- Create expense/income record
- List user's own records only
- Retrieve specific record (own only)
- Update existing record (own only)
- Delete record (own only)
- Verify superuser can access all records

Business Logic Tests

- Flat tax: Amount=100, Tax=10 → Total=110
- Percentage tax: Amount=100, Tax=10% → Total=110
- Zero tax: Amount=100, Tax=0 → Total=100

Permission Tests

- Regular user cannot access other users' records
- Superuser can access all records
- Unauthenticated requests are rejected

Deliverables

Code Structure

- Complete Django project with all models
- Serializers for data validation
- ViewSets with proper permissions
- URL routing configuration
- Database migrations

Documentation

- README with setup instructions
- API endpoint documentation
- Sample API requests/responses

Success Criteria

1. **Functionality**: All CRUD operations work correctly
2. **Security**: Users can only access their own data
3. **Authentication**: JWT tokens properly implemented
4. **Business Logic**: Tax calculations are accurate
5. **API Design**: RESTful endpoints with proper status codes
6. **Testing**: All test cases pass successfully

Tips for Success

- Start with models and migrations first
- Test authentication before implementing CRUD
- Use Django's built-in User model (don't create custom)
- Implement permissions to restrict data access
- Test with multiple users to verify data isolation
- Use Postman or similar tool for API testing