Design Choices and Project Structure

for

Personal Document Management API using Django REST Framework

Introduction

The Personal Document Management API is designed to provide users with a platform to manage their personal documents securely. It allows users to upload, store, retrieve, update, and delete their documents through a RESTful API built with Django REST Framework. This report outlines the design choices and project structure made during the development of the API.

Design Choices

1. Django REST Framework (DRF)

Django REST Framework was chosen as the primary framework for building the API due to its robustness, ease of use, and extensive documentation. DRF provides various tools and features for creating a secure, scalable, and maintainable API.

2. Authentication

To ensure secure access to user-specific documents, token-based authentication was implemented using Django's Simple JWT Token Authentication or a more advanced authentication method like OAuth2. This ensures that only authenticated users can interact with the API.

3. Permissions

Granular permissions were used to restrict access to specific endpoints and actions. Users have read-only access to their documents, while authenticated and authorized users have read-write access to their own documents.

4. Document Storage

The API allows users to upload and store documents securely. For this purpose, a separate file storage solution such as Amazon S3 or Google Cloud Storage can be integrated to handle document storage, ensuring data durability and scalability.

5. API Versioning

API versioning was implemented to allow future updates and changes while maintaining backward compatibility with existing clients. Versioning is included in the URL structure, such as here I use Django rest framework version 3.14.0.

6. Error Handling

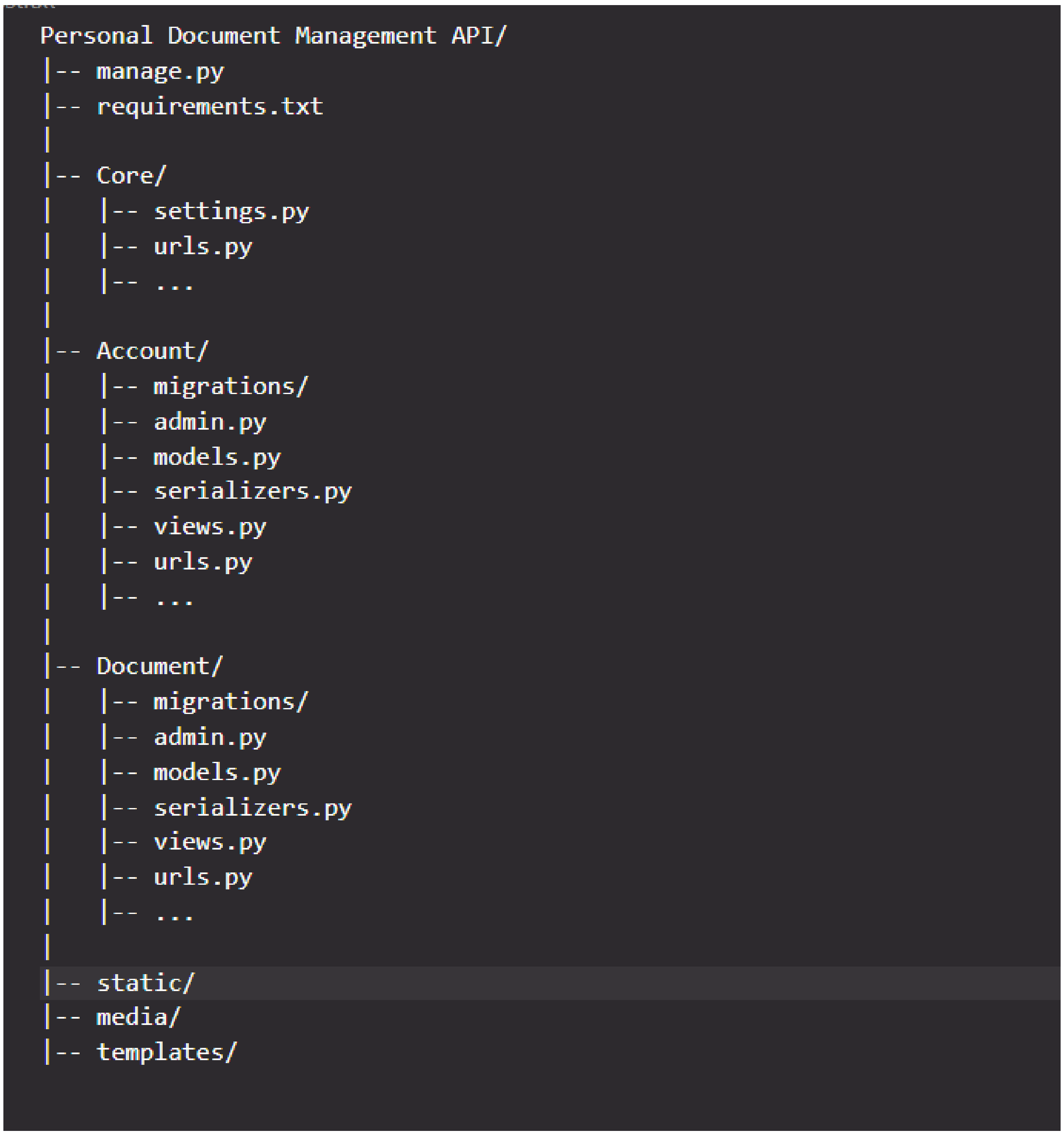
A comprehensive error-handling mechanism was implemented to provide meaningful error messages to clients in case of exceptions or issues with API requests. Custom error responses were created for common scenarios.

7. Rate Limiting

Rate limiting was introduced to prevent abuse and protect the API from malicious attacks. Requests from clients are limited based on a defined threshold.

Project Structure

The project follows a standard Django project structure with additional directories and files for the API app and document management functionality.



```
Personal Document Management API/
|-- manage.py
|-- requirements.txt
|
|-- Core/
|   |-- settings.py
|   |-- urls.py
|   |-- ...
|
|-- Account/
|   |-- migrations/
|   |-- admin.py
|   |-- models.py
|   |-- serializers.py
|   |-- views.py
|   |-- urls.py
|   |-- ...
|
|-- Document/
|   |-- migrations/
|   |-- admin.py
|   |-- models.py
|   |-- serializers.py
|   |-- views.py
|   |-- urls.py
|   |-- ...
|
|-- static/
|-- media/
|-- templates/
```

Explanation of Project Structure:

manage.py: Django's command-line utility for various project-related tasks.

requirements.txt: A file listing all the required dependencies for the project.

Personal Document Management API /: The main project directory containing Django settings, URLs, and other configuration files.

Account/: The app directory for the Authentication of the Personal Document Management API contains models, views, serializers, and URL configurations.

migrations/: Directory for storing database migrations.

admin.py: Configuration for Django admin panel to manage API data.

models.py: Defines the database models for the API, including the Document model to store document metadata.

serializers.py: Contains Django REST Framework serializers to convert complex data (e.g., model instances) into JSON data and vice versa.

views.py: Defines the views that handle API requests and responses.

urls.py: URL configurations for the API app, including routing of requests to appropriate views.

Document/: The app directory for Documents management the Personal Document Management API contains models, views, serializers, and URL configurations.

migrations/: Directory for storing database migrations.

admin.py: Configuration for Django admin panel to manage API data.

models.py: Defines the database models for the API, including the Document model to store document metadata.

serializers.py: Contains Django REST Framework serializers to convert complex data (e.g., model instances) into JSON data and vice versa.

views.py: Defines the views that handle API requests and responses.

urls.py: URL configurations for the API app, including routing of requests to appropriate views.

static/, media/, templates/: Directories for static files, user-uploaded media (e.g., documents), and templates if applicable.

Conclusion

The Personal Document Management API project was successfully designed with Django REST Framework, incorporating various security measures, authentication, permissions, and API versioning. The project structure follows the standard Django conventions, with the necessary additions to handle the document management functionality efficiently. The resulting API provides a secure and user-friendly interface for managing personal documents.