

Task Test for Back-End Developer (Django)

Task : Advanced Project Management Application with Microservices Architecture, Asynchronous Task Processing, and Real-Time Notifications

Objective: Develop a project management application using Django with a microservices architecture. Implement RESTful API endpoints, Celery for asynchronous task processing, WebSocket for real-time notifications, Redis for caching. This will test the candidate's proficiency in Django, RESTful API development, Celery, WebSocket, Redis and Test-Driven Development (TDD).

Requirements:

1. Set up Django Project:

- Initialize a new Django project and create a project management app.
- Set up PostgreSQL as the primary database.

2. Models:

- Create a **Project** model with the following fields:
 - `id` (Primary Key, AutoField)
 - `name` (CharField, max_length=255)
 - `description` (TextField, optional)
 - `created_at` (DateTimeField, auto_now_add=True)
 - `updated_at` (DateTimeField, auto_now=True)
- Create a **Task** model with the following fields:
 - `id` (Primary Key, AutoField)
 - `project` (ForeignKey to **Project**, related_name='tasks', on_delete=models.CASCADE)
 - `title` (CharField, max_length=255)
 - `description` (TextField, optional)
 - `status` (CharField, choices=[('pending', 'Pending'), ('in_progress', 'In Progress'), ('completed', 'Completed')], default='pending')
 - `created_at` (DateTimeField, auto_now_add=True)
 - `updated_at` (DateTimeField, auto_now=True)
 - `due_date` (DateTimeField)
- Create a **Comment** model for task comments with the following fields:
 - `id` (Primary Key, AutoField)
 - `task` (ForeignKey to **Task**, related_name='comments', on_delete=models.CASCADE)
 - `author` (CharField, max_length=255)
 - `content` (TextField)
 - `created_at` (DateTimeField, auto_now_add=True)

3. Serializers:

- Create serializers for **Project**, **Task**, and **Comment** models.

4. Views:

- Implement the following API endpoints:
 - GET /projects/ - List all projects.
 - POST /projects/ - Create a new project.
 - GET /projects/<id>/ - Retrieve a single project by ID.
 - PUT /projects/<id>/ - Update a project by ID.
 - DELETE /projects/<id>/ - Delete a project by ID.
 - GET /tasks/ - List all tasks.
 - POST /tasks/ - Create a new task.
 - GET /tasks/<id>/ - Retrieve a single task by ID.
 - PUT /tasks/<id>/ - Update a task by ID.
 - DELETE /tasks/<id>/ - Delete a task by ID.
 - POST /tasks/<id>/comments/ - Add a comment to a task.
 - GET /tasks/<id>/comments/ - List all comments for a task.

5. Asynchronous Task Processing with Celery:

- Set up Celery with RabbitMQ as the message broker.
- Create a Celery task to send an email reminder for tasks due within the next 24 hours.
- Create a Celery task to send daily project summary reports.
- Schedule the Celery tasks using Celery Beat.

6. WebSocket for Real-Time Notifications:

- Set up Django Channels for WebSocket support.
- Create a WebSocket endpoint /ws/notifications/ to handle real-time notifications.
- Implement functionality to notify clients in real-time when a task is created, updated, deleted, or commented on.
- Notify clients when a project summary report is generated.

7. Caching with Redis:

- Set up Redis for caching frequently accessed data.
- Cache the list of tasks and projects to reduce database load.
- Implement cache invalidation strategies when data changes.

8. Testing:

- Write unit tests for each API endpoint using Django's `TestCase` class.
- Write tests for the Celery tasks to ensure they are scheduled and executed correctly.
- Implement WebSocket testing to ensure real-time notifications are working as expected.
- Write tests to verify Redis caching and invalidation logic.

9. Docker:

- Create a `Dockerfile` and `docker-compose.yml` file to containerize the application.

- The `docker-compose.yml` should include services for the Django app, PostgreSQL, RabbitMQ, Redis, and Celery workers.

10. Git Commits:

- Ensure meaningful and descriptive commit messages.
- Commit frequently to demonstrate incremental progress and document your development process.

Submission:

- Provide a link to the GitHub repository with the project code.
- Ensure the repository includes a `README.md` file with instructions on how to set up and run the project using Docker.
- Include screenshots or logs of the test results, including Celery task execution, WebSocket notifications, and Redis caching.