Task Manager API – Developer Documentation

📌 1. Project Overview

The **Task Manager API** is a production-grade, RESTful API built with **Flask** that supports secure **JWT authentication** and CRUD operations for task management.

This project demonstrates **clean architecture, robust error handling, and professional developer practices** like automated testing, detailed logging, and modular design.

🌟 2. Key Features

✅ Modular Architecture

- Separate layers for **routes**, **models**, **core logic**, and **configurations**.
- Clean, scalable structure for adding new endpoints and features.

✅ JWT-Based Authentication

- Stateless security with **Flask-JWT-Extended**.
- Required for all task endpoints (ensures secure access).

✅ Manageable Logs with Unique IDs

- Every log entry includes a **timestamp + unique event number** (from core/stamp.py).

- Centralized logging with core/logs.py, automatically writing to **info** and **error** logs.
- Helps with **debugging and audit trails** in production.

✅ **Reusable, Managed Database Queries**

- All queries are handled through a custom MySQL_connector class (core/mysql_generic.py).
- Automatic **reconnection** on failure and **fresh cursor re-creation** to avoid "cursor closed" errors.
- Parameterized queries used everywhere to **prevent SQL injection**.

✅ **Interactive Swagger UI**

- Real-time API documentation at /apidocs.
- Supports live testing of endpoints without external tools.

✅ **Comprehensive Testing Setup**

- **pytest** for unit and integration tests.
- **Coverage reports** (HTML/XML) to ensure code quality.
- Ready for **CI/CD integration**.

✅ **Environment-based Configuration**

- .env file for secrets and DB credentials.
- Easily switch between **development** and **production** modes.

✅ **Error Handling & Resilience**

- Graceful responses with meaningful error messages (404, 400, 500).
- Logs unexpected failures with unique IDs for root-cause analysis.

📁 3. Project Structure

```
/task-manager-api
├── app/
│   ├── __init__.py            # App factory, JWT, Swagger
initialization
│   ├── models.py              # Task data model and persistence
logic
│   ├── api/routes.py          # Task CRUD endpoints
│   ├── auth/routes.py         # Authentication endpoints
│   └── core/                  # Custom database and logging modules
│       ├── dbcon.py
│       ├── mysql_generic.py
│       ├── logs.py
│       └── stamp.py
├── config.py                  # Environment configuration
├── requirements.txt           # Dependencies
├── run.py                     # Entry point
└── .env                       # Environment variables
```

⚙️ 3. Setup Instructions

Step 1: Clone the Repository

```
git clone https://github.com/ZFWHospitality/r-ztm-f-d.git
cd r-ztm-f-d
```

Step 2: Create and Activate Virtual Environment

```
python -m venv venv

# Linux / MacOS
source venv/bin/activate

# Windows
venv\Scripts\activate
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
pip install pytest pytest-flask pytest-cov pytest-mock
```

Step 4: Configure Database

Run these SQL queries in **both main and test databases**:

```
CREATE DATABASE task_manager_db;
CREATE DATABASE test_task_manager_db;

USE task_manager_db;
CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE tasks (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    completed BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP
);
```

Step 5: Environment Variables

Create a `.env` file in the root directory:

```
SECRET_KEY=your-secret-key
JWT_SECRET_KEY=your-jwt-secret-key
MYSQL_HOST=localhost
MYSQL_USERNAME=root
MYSQL_PASSWORD=your-password
MYSQL_DATABASE=task_manager_db
```

🚀 4. Running the Application

```
python run.py
```

Once running, visit:
Swagger UI: http://127.0.0.1:5000/apidocs/

🧪 5. Testing with Swagger UI

Step 1: Register and Login

1. Expand **User registration and login** section.
2. Use `POST /auth/register` to create a new user:

```
{
  "username": "new_tester",
  "password": "strong_password123"
}
```

3. Use `POST /auth/login` with the same credentials to receive a JWT token.

Step 2: Authorize Swagger UI

1. Click **Authorize** (lock icon).
2. Paste the token into `BearerAuth` field (without `Bearer` prefix).
3. Now all task endpoints are unlocked.

Step 3: Perform Task Operations

- **Create Task** → `POST /api/v1/tasks`

  ```
  {
    "title": "Document API Steps",
    "description": "Write a guide for using Swagger UI."
  }
  ```

- **Retrieve Task** → `GET /api/v1/tasks/{id}`
- **Delete Task** → `DELETE /api/v1/tasks/{id}`

🧪 6. Running Automated Tests

Run All Tests

```
pytest -v
pytest --cov=app
```

Run Specific Categories

```
pytest tests/test_core/ -v
pytest tests/test_models/ -v
pytest tests/test_api/ -v
pytest tests/test_auth/ -v
```

Generate Coverage Reports

```
pytest --cov=app --cov-report=html   # Open htmlcov/index.html
pytest --cov=app --cov-report=xml    # For CI/CD
```

🛠️ 7. Troubleshooting

Issue	Solution
MySQL Connection Error	`sudo systemctl start mysql`
Import Errors	Run from project root: `python -m pytest tests/`
pytest Not Found	Activate venv and reinstall: `pip install pytest`
Cursor Closed Errors	Code already includes automatic cursor re-creation in `mysql_generic.py`

🤖 8. Test Execution Script (Optional)

Create `run_tests.sh`:

```
#!/bin/bash
source venv/bin/activate
pytest -v --cov=app --cov-report=html
```

Make it executable:

```
chmod +x run_tests.sh
./run_tests.sh
```

✅ 9. Best Practices for Developers

- **Run tests** before each commit to avoid regressions.
- **Keep test DB separate** from production.
- **Use coverage reports** to maintain quality.
- **Test specific modules** during development for faster iteration.