Python Developer Assignment

Objective:

This assignment assesses your proficiency in building a secure and scalable API using modern Python tools. You'll create a REST API with user authentication, CRUD operations, reporting, and various modern development practices.

Technologies:

- FastAPI
- Async MongoDB
- Poetry
- Pydantic
- Async Pytest
- Celery
- Docker
- Pre-Commit Hooks
- JWT Authentication
- Error Monitoring (e.g., Sentry)

Tasks:

1. Project Setup:

- Establish the project using Poetry for dependency management.
- Configure the FastAPI application, defining relevant middleware (e.g., error handling, logging).
- Integrate the Async MongoDB driver and define models using Pydantic for data validation.
- Implement JWT authentication with secure password hashing and token generation/validation.
- Configure Celery for asynchronous tasks.
- Set up a Dockerfile for containerization.
- Implement Pre-Commit hooks for linting and code formatting.

2. API Endpoints:

- **GET /healthendpoint:** Create a simple health check endpoint returning basic API health information.
- **POST /userendpoint:** Develop an endpoint for registering new users in the `usercollection`, ensuring secure password handling and validation.
- **CRUD /candidateCRUD:** Implement endpoints for:

    - **POST /candidates:** Create a new candidate profile in the candidate collection.
    - **GET /candidates/{id}:** Retrieve a specific candidate's information.
    - **PUT /candidates/{id}:** Update an existing candidate's information.
    - **DELETE /candidates/{id}:** Delete a candidate profile.

- **GET /all-candidates:** Implement an endpoint to retrieve all candidates from the candidatecollection, with the following features:
    - Search across all candidate data fields.
    - Global search using keywords.
    - Pagination to handle large datasets.

- **GET /generate-report:** Generate a CSV report containing all candidate information, optimized for efficiency with large datasets.

3. Authorization:

- Protect /candidate and /all-candidates endpoints with JWT authentication.
- Require users from the usercollection to access these protected endpoints.

4. Testing:

- Write unit tests for all API endpoints using Async Pytest.
- Utilize mock objects for external dependencies where appropriate.
- Consider integration tests for Celery tasks.

5. Error Monitoring:

- Integrate an error monitoring service like Sentry to capture and report errors and exceptions.
- Configure appropriate error levels and context information.

6. Documentation:

- Provide API documentation (e.g., Swagger docs) describing endpoints, their parameters, and expected responses.

Requirements:

- Prioritize code quality, readability, and maintainability.
- Adhere to Object-Oriented Programming principles.
- Implement proper logging and exception handling.
- Include clear comments and docstrings (using Google format) explaining key functionalities.
- Package your code in a Docker container for easy deployment.

Additional Notes:

- Implement all technologies mentioned in the **Technologies** section.
- Feel free to choose specific libraries or implementations for each technology.
- Explain your design choices and any additional features you implemented.
- Always be mindful of security best practices throughout your development process.

Good luck with the assignment!

Further Enhancements:

- Consider including a sample `.env` file to facilitate project setup.
- Provide instructions for running the project and using the API.
- Offer guidance on testing and deployment workflows.