Recruitment Exercise Brief

Implementing a User Service in Python

Objective

The following exercise is a great opportunity to pick up some important knowledge while illustrating your ability to learn quickly and be effective when provided with some guidance.

Regardless of the outcome, we will provide detailed feedback over a call which will help you maximise your learning.

Task Description

This project involves the development of a RESTful API for a user registration system. The API will be implemented using Python with the Uvicorn server and OpenAPI. Poetry will be used as the package manager for managing dependencies. The project follows REST API best practices, ensuring security and consistency in responses. The code will be written in an Object-Oriented Programming (OOP) style, adhering to the best practices for clean and maintainable code.

Endpoints

- Register User
    - **Endpoint**: /register
    - **Description**: Registers a new user by accepting a valid email, password, and age.
- Get All Users
    - **Endpoint**: /users

bitloops

    - **Description**: Retrieves a list of all registered users, excluding their passwords.

Requirements

- **API Implementation**: Follow REST API best practices, including using appropriate HTTP methods, status codes, and response formats.
- **Security**: Consider security practices such as input validation and secure data handling for data which need to be secured.
- **Response Format**: All responses should be in JSON format and follow a consistent structure.
- **Database**: Choose a suitable database for storing user data.
- **Object-Oriented Design**: Implement business rules within the User class, using a rich domain model. So the business rules should live inside the User class (rich domain model instead of anaemic model)
- **Clean Code**: You should try for the code to adhere to clean code best practices.
- **Types**: The code should use Python types.
- **Testing**: Implement BDD tests to verify the registration logic, using dependency inversion to mock the database and not use the actual database for the business logic testing.
- **Bonus 1**: Set up a Dockerfile so that your application can be easily deployed in various environments with zero setup.
- **Bonus 2**: Set up a mock email service to send notifications to users after their registration (i.e. a console message that an email was sent to the user). The point here is show how you do this at the appropriate (application) layer again using dependency inversion.
- **Bonus 3**: Structure the application with Clean Architecture or Hexagonal Architecture (aka Ports and Adapters). Both architectures share the same principles (layered) so it does not matter which one you will choose.

bitloops

Submission Guidelines

Please upload your code at a public GitHub repository and just email us the URL at vasilis@bitloops.com and c.c.: antonis@bitloops.com

Good luck and we are looking forward to reviewing your work!