OptimalFlow Backend Developer Exercise

Objective

This exercise is designed to evaluate backend development skills, problem-solving ability, and code quality.

It focuses on core backend tasks while providing optional challenges for those who wish to demonstrate advanced capabilities.

Instructions

- Choose a preferred backend language and framework:
  - Golang (recommended: Fiber, Echo, or Gin)
  - Node.js (recommended: NestJS or Express)
  - Python (recommended: FastAPI)
- Complete all items in the **Core Requirements** section.
- If there is extra time or interest, work on the **Bonus (Optional)** section to demonstrate advanced skills.

Core Requirements

1. Implement these REST API endpoints:
   - `POST /users` : Create a user (name, email, password, balance)
     - Initialize the **balance** to 100 upon creation.
   - `POST /login` : Authenticate a user (check email and password match)
   - `GET /users` : List all users (do not include passwords in the response)
   - `GET /users/:id` : Retrieve user details by ID (do not include password)
2. Data Storage:
   - Use an in-memory store (map, list) or a local file (such as a JSON file).

    - A full database setup is not required for this part.

3. Password Security:

    - Hash passwords before storing them (e.g., using bcrypt or an equivalent library).
    - Do not store plain text passwords.

4. Code Structure:

    - Organize the code with a clear separation of concerns: handler/controller, service/business logic, and data layer.

5. Setup and Testing:

    - Provide clear instructions for running and testing the API. A Postman collection or curl commands are acceptable.

Bonus (Optional)

These are not required but can help demonstrate advanced knowledge and stand out.

- Implement JWT-based authentication or session token handling
- Add a `POST /transfer` endpoint to transfer **balance** between users

    - Must handle balance updates atomically
    - Validate sufficient balance and proper input

- Write unit or integration tests
- Provide a Docker setup (Dockerfile, and optionally docker-compose)
- Use a lightweight database (such as SQLite, PostgreSQL, or MongoDB)
- Include a short document or README section describing how the system would be scaled to handle 10x the current traffic

---

Time Expectation

The core section is designed to be completed in approximately 3–4 hours.
The bonus section can be completed with additional time if desired.

Evaluation Criteria

- Code correctness and secure implementation
- Clarity, maintainability, and organization of the code

- Thoughtfulness and completeness in optional improvements (if included)
- Clear README and documentation of design choices

Submission Instructions

Please submit the following:
- A link to your code repository (GitHub or similar platform) - A README file with:
- Setup instructions
- How to run and test the API (include Postman collection or curl examples)
- Any notes or explanation about your design decisions, code structure, and optional improvements (if any)

Send your submission to: **contact@optimalflow.co**
If you need any clarification or run into blocking issues, feel free to reach out.

We look forward to reviewing your work.