Objective

This technical challenge is designed to evaluate your backend development skills, specifically in the areas of authentication, API design, containerization, and inter-service communication using Docker.

You are required to build a simple system composed of:

- Two simple backend services with linked databases
- One simple frontend (minimal UI)
- All components containerized with Docker and orchestrated using Docker Compose

System Requirements

1. User Service Backend with a database
    - Acts as a centralized login/sign-up endpoint.
    - Users should be able to register and log in.
    - Upon successful login, the user receives a JWT (JSON Web Token).
2. Todo List Backend with a database
    - A separate backend for managing a personal todo list.
    - Must expose CRUD endpoints (Create, Read, Update, Delete) and corresponding tests.
    - JWT has to be validated before requests are processed. To validate the JWT, use the same secret with user service.
3. Frontend (Optional)
    - Minimal pages to demonstrate login and interaction with the todo service.
    - Feel free to use plain HTML or any lightweight framework (optional).

Deliverables

- A git repo including source code for all services.
- Dockerfiles and docker-compose.yml.
- Clear instructions to build and run the system.
- API documentation (Postman/bruno collection or openapi documents).
- Unit test cases for all user stories.

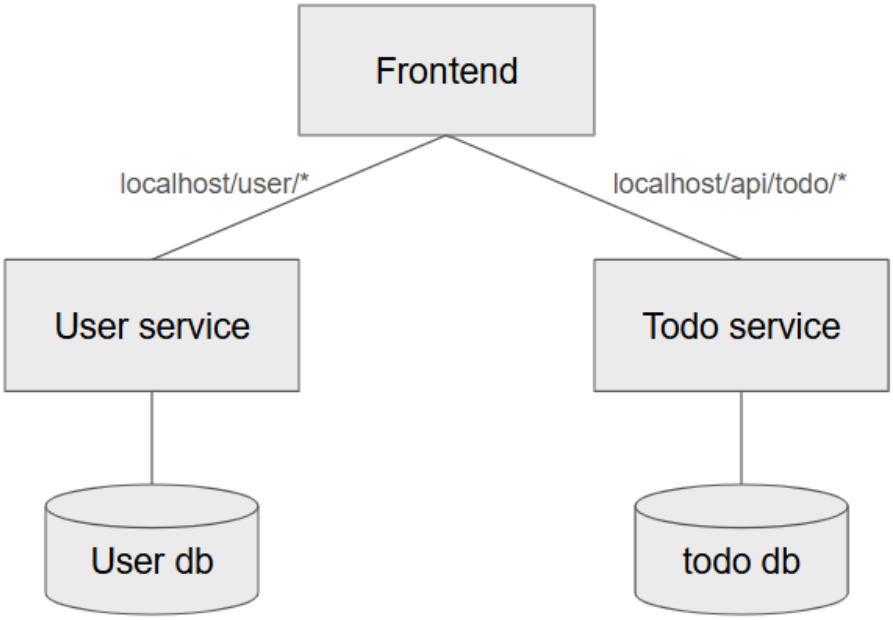
Technical Stacks

- **Backend:** Node.js with TypeScript
- **Database:** MySQL or PostgreSQL
- **Deployment:** Docker and Docker Compose

Optional:
You are welcome to incorporate any additional open-source tools or frameworks to enhance your solution.Choose technologies that demonstrate your experience and best practices.

Implementation Details

1. System design



2. Database design

   Note: The uuid field in the users table and the user_uuid field in the todos table should have a **foreign key** relationship. This ensures that each todo item is properly associated with a specific user.

User table		
field	type	explanation
id	number	This is the default id by postgresql
uuid	string	The unique id of user
user_email	string	User email address
User_pwd	string	User password

Todo table		
field	type	explanation
id	number	This is the default id by postgresql
uuid	string	The unique id of todo item
content	string	The content of todo item
user_uuid	string	The related user uuid

3. User stories

User service

User Story 1: User Registration

As a new user,
I want to register for an account with my email and password,
So that I can securely log in and access my personal todo list.

Acceptance Criteria:

1. The system must accept a valid email and password (e.g., password min length 6).
2. If the email is not already registered, a new user account is created.
3. The password must be hashed and stored securely.
4. If the user registers successfully, return 201 created.
5. If the email is already in use, return a 409 Conflict or appropriate error message.
6. Invalid or missing fields (e.g., password length < 6) must result in a 400 Bad Request.

User Story 2: User Login

As a registered user,
I want to log in using my email and password,
So that I can receive a JWT to authenticate future requests.

Acceptance Criteria:

1. The system must accept valid email and password credentials
2. The JWT must include user identification (e.g., user ID or email) in the payload.
3. If credentials are correct, login success. The system returns 200 with a valid JWT.
4. If the credentials are invalid, the system must return a 401 Unauthorized response.
5. Passwords must never be exposed in any response or logs

Todo Service

User Story 1: Create Todo

As a logged-in user,
I want to create a new todo item,
So that I can track tasks I need to complete.

Acceptance Criteria:

1. The request must include a valid JWT in the Authorization header.
2. The created todo must be associated with the authenticated user.
3. The response must return a 201 Created status and the created todo item.
4. If the JWT is missing or invalid, return a 401 Unauthorized error.

User Story 2: Read Todos

As a logged-in user,
I want to fetch my list of todos,
So that I can see what tasks I have.

Acceptance Criteria:

1. The request must include a valid JWT in the Authorization header.
2. The response must return only the todos belonging to the authenticated user.
3. The response should return a 200 OK with a list of todos in JSON format.
4. If the user has no todos, return an empty array.
5. If the JWT is missing or invalid, return a 401 Unauthorized.

User Story 3: Update Todo

As a logged-in user,
I want to update an existing todo item,
So that I can change its title, description, or status.

Acceptance Criteria:

1. The request must include a valid JWT in the Authorization header.
2. Only the owner of the todo can update it.
3. If the update is successful, return a 200 OK with the updated todo item.
4. If the todo does not exist or does not belong to the user, return a 404 Not Found or 403 Forbidden.
5. If the JWT is invalid or missing, return a 401 Unauthorized.

User Story 4: Delete Todo

As a logged-in user,
I want to delete one of my todo items,
So that I can remove tasks that are no longer needed.

Acceptance Criteria:

1. The request must include a valid JWT in the Authorization header.

2. Only the owner of the todo can delete it.
3. If deletion is successful, return a 204 No Content status.
4. If the todo does not exist or does not belong to the user, return a 404 Not Found or 403 Forbidden.
5. If the JWT is invalid or missing, return a 401 Unauthorized.