DIGITOPIA CASE STUDY

TARGET CANDIDATE

Late Junior / Mid Backend Developer

PURPOSE OF THE STUDY

Assess the candidate's knowledge of the listed technologies and libraries. The case study covers common backend topics using Kotlin (or Java, since Kotlin is based on Java) with Spring Boot, relational data structures, and web services.

Candidates can have questions about the case and contact the recruiter. Purpose and structure of these questions can also be used as acceptance criteria during evaluation.

KNOWLEDGE CHECK

Required

- Kotlin (preferred) or Java 12+
- Spring Boot with annotations
- JPA or Hibernate
- Gradle or Maven
- Microservice Architecture
- Cloud Services

Optional

- Unit Testing and API Tests with JUnit
- Open API with Swagger
- Event Driven Architecture & Queues

DELIVERY

- GitHub or BitBucket repository share
- 1-on-1 study session after code review to explain solution approach

ACCEPTANCE CRITERIA

Candidates should have the required knowledge and cover,

- Case requirements specified below.
- Proper, detailed documentation for the data structure and endpoints.
- Some operations and implementation details (such as data types, relations, input validation, and sanitation) are purposely left unspecified. This is to evaluate how candidates analyse requirements and make design decisions.
- There are several ways to achieve the given requirements. Candidates are expected to deliver a high performing case study and have consistency across all code.

Optional knowledge and functionalities are nice to have and are **not required** to complete the study. However, they can be considered as a plus during evaluation.

Case Requirements

You can find below the main requirements for this case study. Optionally specified criteria are not required during submission but will be taken into account when evaluating candidates.

Main Data Structure and Services

Create microservices to cover all CRUD operations for the given data structure below.

- Sanitize all free texts and validate standard inputs such as email
- Create indexes for searchable fields

ALL entities must have ID: UUID, createdAt: date, updatedAt: date, createdBy: UUID (creator user id) and updatedBy:UUID (updating user id) fields.

ALL microservices should have an endpoint to check their health status, such as "/healtz".

USER

- Email
- Status: Enum [ACTIVE, PENDING, DEACTIVATED, DELETED]
- Full Name: letters only (no numbers or special characters)
- Normalized Name: lower case, alphanumeric, ASCII-only (English characters)
- Role: Enum [ADMIN, MANAGER, USER] → OPTIONAL. See Optional Functionalities
- Requirements:
  - A user can be in multiple organizations
  - Each email must be unique across all users, regardless of status
- Additional Endpoints:
  - Return all organizations that a user belongs to
  - Search by normalized name to return matching users
  - Search by email to return a single user

INVITATION

- User ID
- Organization ID
- Invitation Message
- Status: Enum [ACCEPTED, REJECTED, PENDING, EXPIRED]
- Requirements:
  - Invitations automatically expire after 7 days (based on createdAt). A daily scheduled job should update their status to EXPIRED
  - Only one pending invitation can exist per user per organization
  - A user can be reinvited if the invitation is expired
  - A user cannot be reinvited if their last invitation was rejected

ORGANIZATION

- Organization Name: alphanumeric characters
- Normalized Organization Name: lower case, alphanumeric, ASCII-only (English characters)
- Registry Number: alphanumeric characters
- Contact Email
- Company Size
- Year Founded

- Requirements:
    - A registry number can be used only once to create an organisation
    - An organization can have multiple users
- Additional Endpoints:
    - Return all users belonging to an organization
    - Search by normalized name, year, company size to return matching organizations
    - Search by registry number to return a single organization

Optional Suggestions

- You can populate createdBy and updatedBy fields using request headers
- Read the data structure carefully and create tables and indexes accordingly
- Implement pagination in search endpoints
- Document any assumptions made during design or implementation
- Reference all AI-generated code using comments

Optional Functionalities

- Use Cloud Services to create a basic Auth flow.
    - ADMINs can call all endpoints and directly create ACTIVE users
    - MANAGERs cannot call delete actions and can only create PENDING users
    - USERs can only access their own records (e.g., deactivate their account, invite users to their organizations)
- Implement a history/audit log to track record changes.
- Use OpenAPI and Swagger to auto generate base structure of your services.
- Use Junit to create unit tests for your repositories and API tests for your endpoints.
- Integrate a mail server to send invitation emails.
- Create a validation process for accepting invitation.
- Use Orchestration Pattern and events to create microservices.
- Deploy your study to cloud. You can use free tier services.