Technical Design Documentation

Project Name: Trio's Code Challenge

Date: 29/11/2022
Written By: Samuel Catalano

Introduction

This document contains information about the Trio's Code Challenge proposed.

System Overview

This project will be creating a tool that syncs contacts from MockAPI to Mailchimp. For this integration, you will need to get each contact's first name, last name, and email address from MockAPI and create them as new members of a new list on Mailchimp. The new list on Mailchimp should have your personal name (e.g.: Samuel Catalano).

System Architecture

- **Classes and new objects**
  - ☐ ContactRepository
    - This is an interface responsible for providing CRUD operations
    - *countSyncedContacts* executes a native query to count
  - ☐ BaseModel
    - This is a base class for other entity classes
    - Has pre-defined fields ***id*** and ***createAt***
    - Should be inherited by other model child classes
  - ☐ Contact
    - This is an entity class which extends the BaseModel class
    - Contains the fields described in challenge documentation
    - Will be persisted into a memory database
  - ☐ ContactDTO
    - This class is a Data Transfer Object representation of the Contact
    - It has the same fields contained in the Contact class
  - ☐ SyncedContactDTO
    - This class contains just the necessary fields for the API response
    - It will be mapped from the Contact class-related fields
  - ☐ SyncedResultDTO
    - This class represents the Object to be returned in the API call
    - Contains a synced contacts count and a list of synced contacts object

- [ ] WelcomeDTO
    - This class contains the fields for a Welcome Message
    - Should return a “Hello, World!” classic message
- [ ] ApplicationConfig
    - This class configure the ModelMapper and RestTemplate as Spring Boot managed classes
- [ ] SwaggerConfig
    - This class configure a Swagger Fox 2 Open API Rest
- [ ] MailChimp
    - This class is a util class responsible for adding or updating a new contact as a member and subscribing it to a MailChimp list.
- [ ] ContactService
    - This is an interface for contracts that should be applied to the Contac class
- [ ] ContactServiceImpl
    - This class is the implementation of the interface ContactService and implements its contracts responsible for communicating with the repository, retrieving necessary data and providing data to other classes
- [ ] BaseService
    - This is a base class for all services. It has the definition of the base URL and the base HTTP header with the content-type
- [ ] TrioBackendChallengeRESTController
    - This class is a controller responsible for a welcome message which comes from WelcomeDTO class
- [ ] ContactRESTController
    - This class is a controller responsible for receiving GET requests in a path `/contacts/sync` and response with a JSON representation defined in the SyncedResultDTO class
- [ ] ApiException
    - This is an Exception class and should be used for throwing exceptions in API classes' context
- [ ] ConfigurationException
    - This is an Exception class and should be used for throwing exceptions in Configuration classes' context
- [ ] ServiceException
    - This is an Exception class and should be used for throwing exceptions in Service classes' context

- **Database**
  - [ ] contact
    - This table will store the contacts
    - It will have the following fields:
      - [ ] id: VARCHAR(255) PK (PK = primary key)
      - [ ] created_at: TIMESTAMP
      - [ ] email: VARCHAR(255)
      - [ ] first_name: VARCHAR(255)
      - [ ] last_name: VARCHAR(255)
      - [ ] avatar: VARCHAR(255)
      - [ ] synced: BOOLEAN

Tables

contact

it will store the contacts.

Field	Type	Nullable?	Constraints
id	VARCHAR(255)	false	PK
created_at	TIMESTAMP	false	
email	VARCHAR(255)	false	
first_name	VARCHAR(255)	false	
last_name	VARCHAR(255)	false	
avatar	VARCHAR(255)	false	
synced	BOOLEAN	false	

Files

The project contains a source called `json` with a file inside called *contacts.json*. This file is used for simulating a list of contacts retrieved from a Contacts Endpoint: https://challenge.trio.dev/api/v1/contacts and it's being used in a test context.

Endpoints

Root endpoint

1. The server starts
2. The server receives the GET request on /
3. The server responds to this structure

```
{
    success: true,
    message: "Hello, World!"
}
```

Contact endpoint

1. The server receives the GET request on `/contacts/sync`
2. The server responds to this structure

```
{
  "syncedContacts": 1, // total synced contacts
  "contacts": [
      {
          "firstName": "Amelia",
           "lastName": "Earhart",
           "email": "amelia_earhart@gmail.com"
      },
  ]
}
```

Next Steps

As the next steps, we may implement new features about updating contacts, subscribing and unsubscribing then, removing them from the public or something related to that.