ECS 713P - Functional Programming

Group Coursework Report

Students' Details
Name: Thananont Chevaphatrakul - Student ID: 240737582
Name: Abdulla Sabith Fariq - Student ID: 240174846
Name: Petros Petrou - Student ID: 240910783

Course Instructors :
Professor Edmund Robinson
Dr. Paulo Oliva

School of Electronic Engineering and Computer Science
Queen Mary University of London

December 13, 2024

Contents

1 Introduction **3**

2 Project Setup **3**

3 Implementation Description **3**
3.1 Overview Modules 3
3.2 Fetched API Data and Endpoint Calls 3
3.3 Default Commands 3
3.4 Implemented Queries 3
3.5 Extra Features 3

4 Design Choices - Challenges **4**
4.1 Making Multiple API Calls 4
4.2 Database Structure and Relationships 4
4.2.1 Database Field Mismatch 4
4.3 Environment Variable Validation 4
4.4 Additional Libraries - Functions Needed 4

1 Introduction

This project is a Haskell-based command-line application that its purpose is to fetch data from the Transport for London Web API, parse the JSON responses and store them in an SQLite database. After that, users can query the stored data, generate a JSON dump and also interract with the database.

2 Project Setup

To run the project, **Stack** is required as the Haskell build tool. Version **lts-22.39** ensures compatibility with libraries such as **http-conduit** (HTTP requests), **sqlite-simple** (database management), **bytestring** (binary data), **aeson** (JSON parsing), and **text** (text manipulation).

3 Implementation Description

3.1 Overview Modules

The project is organized into modules for clarity: **Types.hs** defines data types, **Fetch.hs** handles HTTP requests, **Parse.hs** parses JSON data, **Database.hs** manages the SQLite database, and **Main.hs** serves as the main entry point.

3.2 Fetched API Data and Endpoint Calls

The following endpoints from the Transport for London (TfL) API were used:

- **Line Meta Modes**: Retrieves available transportation modes (e.g., tube).
- **Route by Mode**: Provides route details for a specific mode of transport.
- **Stoppoint Search**: Searches for stop points such as stations or bus stops.
- **Line Disruptions by Mode**: Returns disruption information for a specified mode.

3.3 Default Commands

- **Create Database Tables** - `stack run -- create`
- **Load Data from the API** - `stack run -- loaddata`
- **Export Data to JSON File** - `stack run -- dumpdata`

3.4 Implemented Queries

- **List All Modes of Transport** - `stack run -- modes`
  Description: Prints all available transportation modes (e.g., bus, tube, tram).
- **Show Routes for a Specific Mode** - `stack run -- routes <mode_name>`
  Replace `<mode_name>` with the desired mode (e.g., "bus").
  Description: Displays all routes for a given mode of transport (e.g., "bus" or "tube").
- **List Stop Points for a Specific Mode** - `stack run -- stop-points <mode_name>`
  Replace `<mode_name>` with the relevant transport mode (e.g., "tube").
  Description: Lists all stop points (e.g., stations) for a specified mode of transport.

3.5 Extra Features

The following additional commands have been implemented:

- **Drop All Database Tables** - `stack run -- drop`
  Description: Deletes all tables from the SQLite database.
- **Search for Destinations** - `stack run -- search`
  Description: The user will be prompted to enter a destination name, and the application will fetch and display the modes for up to 10 matching results. In case there are no results, then an error message is printed.
  Example: Please enter your destination: Whitechapel
  Found search data
  Name: Whitechapel - Available modes: ["elizabeth-line","bus","overground","tube"]

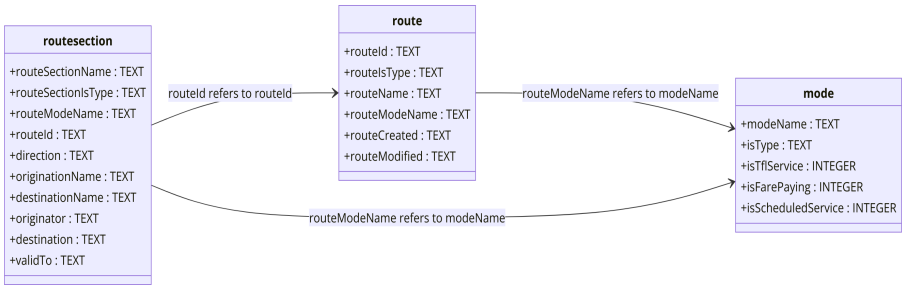
- **Show Disruptions for All Modes** - `stack run -- disruptions`
  Description: Prints real-time disruption information for all transportation modes.
  Example: Disruptions for Mode: tube - Category: RealTime - Description: Piccadilly Line: No service between Rayners Lane and Uxbridge due to difficult track conditions. Please use the Metropolitan line to complete your journey. MINOR DELAYS on the rest of the line due to a shortage of trains.

4 Design Choices - Challenges

4.1 Making Multiple API Calls

To retrieve all routes for each transportation mode, the application first fetched the available modes using an API call. The mode names were then used to dynamically generate the required URLs with the `parseURLforRoutesAPI` function. These URLs were passed to the `downloadMultiple` function, which handled multiple API requests concurrently to fetch the route data for all the modes. The parsed results were processed and inserted into the database using `mapM_`, a monadic function that performs actions sequentially.

4.2 Database Structure and Relationships



Note: From the Route by Mode API call, 9 fields were retrieved; however, the `lineStatuses` and `disruptions` fields were excluded as they were consistently empty. The remaining data was split into two tables: **route** for general route information and **routesection**, which stores the details of the `routeSections` object. This design ensures modularity and better representation of hierarchical data.

4.2.1 Database Field Mismatch

A challenge arose when dumping data from the database due to mismatches between the API response and the database schema. To resolve this, new types (`ModeDB`, `RouteDB`, and `RouteSectionDB`) were created to align with the database schema. These types excluded unused fields and incorporated foreign keys like `routeId` and `routeModeName`, ensuring seamless data dumping and maintaining a clean, normalized structure.

4.3 Environment Variable Validation

To securely manage the Transport for London API key, the application retrieves it from an environment variable using `lookupEnv "TFLAPPKEY"`. If the key is not set, the program outputs an error message and terminates execution. This approach ensures the API key is not hardcoded, enhancing security and configuration flexibility.
Note: In order for the program to run, a `.env` file containing the API key must be present in the project's root directory. The API key must be defined as `TFL_APP_KEY = "..."`

4.4 Additional Libraries - Functions Needed

Control.Monad (unless) is used to conditionally print data when it is not empty, **Data.Char (toLower)** ensures mode names are consistently converted to lowercase, and **Configuration.Dotenv (loadFile, defaultConfig)** loads environment variables from the `.env` file for secure and flexible configuration management.