ETL PROJECT

Ariel Tamzegui Leventhal

Introduction

This report details the impletation of the ETL (Extract, Transform, Load) project in OCaml. The project is designed to calculate the total price and total taxes of orders loaded from a CSV file. The csv files are downloaded from a given URL, parsed, and the results are saved in a new CSV file called `order_total.csv`.

The project is structured to filter orders by order statuses and origins, allowing for flexible data processing.

The project is managed by Dune, a build system for OCaml projects, and uses the following libraries for CSV handling and HTTP requests:

- **csv**: For reading and writing CSV files.
- **Lwt**: For asynchronous operations.
- **cohttp-lwt-unix**: For HTTP requests.

Observation: The file links in this report only work properly when visualizing the markdown version in github.

Project Structure

All of the source code is located in the `bin` directory. The main components are:

- **main.ml**: The main entry point for the application. Contains the general logic of the ETL process.
- **io.ml**: Handles impure I/O functions, including downloading CSV files.
- **fileParsing.ml**: Provides impure functions to parse CSV file into OCaml types.
- **parsing.ml**: Provides pure functions to parse data from CSV files into OCaml types.
- **types.ml**: Defines the data types (e.g., `order`, `fullOrder`, `orderTotal`, `processedOrderedItem`).

Building and Running the Project

To build the project, ensure you have Dune and the libraries used by this project installed. From the root of the project, run:

```
dune buld
```

To run the project, use the following command:

```
dune exec etl
```

Parameters

The project accepts the following command-line parameters to filter and customize the ETL process:

- `-status <order_status>`
  Specify the order status to filter by (e.g., `Cancelled`, `Pending`, `Complete`).
- `-origin <order_origin>` Specify the order origin to filter by (e.g., `P` for Physical or `O` for Online). You can combine these parameters to refine your filtering. For example, the following command processes only cancelled orders from Physical origin:

```
dune exec etl -- -status Cancelled -origin P
```

If no parameters are provided, the project processes all orders.

Output

The output of the project is saved in a new CSV file called `order_total.csv`. The file contains the following columns:

- `order_id`: The ID of the order.
- `total_amount`: The total value of the order items.
- `total_taxes`: The total tax amount for the order.

Implementation Details

Download CSV File

The CSV file is downloaded from a given URL using the `Cohttp_lwt_unix` library. The `download_csv` function in `io.ml` handles the HTTP request and saves the file locally.

Reading and Parsing CSV File

```
The CSV file is read using the `Csv` library.
```

The `fileParsing.ml` module is responsible for converting CSV rows into the project's domain types.

- The function `parse_order_row` takes a CSV row (expressed as a list of key-value pairs) and extracts the `id`, `status`, and `origin` fields to create an `order`.
- Similarly, `parse_orderItem_row` parses a row into an `orderItem` by extracting the `order_id`, `quantity`, `price`, and `tax` values.
- A helper function, `unwrapResult`, is used to extract values from the `Result` types returned by the pure parsers (such as `parse_id`, `parse_orderStatus`, and `parse_orderOrigin`), raising exceptions on errors.
- Additionally, command-line arguments are parsed by `parse_args` to provide optional filtering criteria.

Data Processing: Grouping, Item Processing, and Filtering

Once the CSV data has been read and parsed into domain types (e.g., `order`, `orderItem`) defined on types.ml, the next phase involves several processing steps to generate meaningful results.

- **Processing Order Items**
  The raw `orderItem` records, obtained from parsing individual CSV rows, are transformed into `processedOrderedItem` records. This transformation is performed by the function `processItem`, which computes the total price and total tax for each order item. Specifically, it calculates:
  - `item_total` as the product of the item's price and its quantity.
  - `total_tax` as the product of the item's tax rate and the calculated `item_total`.
- **Grouping Order Items**
  After processing, the list of `processedOrderedItem`s is grouped by `order_id` using the function `group_processed_items`. A hash table is used to collect all items belonging to the same order. For each unique `order_id`, a corresponding `fullOrder` record is created that aggregates these items along with additional order details (such as `status` and `origin`).
- **Filtering Orders**
  With the orders grouped into `fullOrder` records, the function `filter_orders` applies filtering based on command-line parameters (such as order status and origin). This step ensures that only orders matching the specified criteria are retained for further processing. In the absence of parameters, all `fullOrder` records are processed.
- **Calculating Totals**
  Finally, the `calculateTotal` function computes aggregate data for each filtered `fullOrder`. It sums the individual `item_total` and `total_tax` values from the processed items, producing an `orderTotal` record. This record contains the order ID, the overall total amount, and the total taxes, and is ultimately output to the final CSV file.

Optional requirements implemented

[x] Document functions using docstrings. [x] Implement groupby [x] Use dune [x] Get CSV file from a URL

Generative AI usage

The project use generativa AI tools in the following ways:

- Debug and Fix Errors: The project used LLMs to debug multiple errors in the code.
- Code Generation: The project used LLMs to generate code for the parse_args function. Code generation was also used to learn the basic usage of certain ocaml language or libraries features such as HashTables and Results. The basic code generated by LLMs was then modified to fit the project needs in functions such as the parsing functions and the grouping function.
- Documentation: The docstrings used for function documentation were mostly generated by LLMs.
- Text: Generative AI was also used to improve and format the markdown README file and this report.