Software Requirements & Design Specification

for

End-User Search Queries (CSV dataset)

Group - 2 Team - 12

Tathagata Raha - 2018114017
Dipanwita Guhathakurta - 2018112004
Pawan Baswani - 2021701035

Table of Contents

Software Requirements

1. Introduction

The purpose of this document is to present a detailed description of the interface for end-user search queries on CSV datasets. It will explain the purpose and characteristics of the interface, what the system will perform, the limitations under which it must function, and how the system can be integrated with the other modules/sub-systems.

2. Overall Description

2.1 Product Perspective & Features

This product is being developed currently as a standalone project for demonstration purposes. However, we are creating a well-documented RESTful API alongside, to integrate it easily with the umbrella project proposed at the start of the course.

- Accept input data in CSV or other formats
- Support for custom dataset upload (optional feature, collaboration with dataset upload and verification team needed)
- Dataset Preview and description
- Query suggestions
- Support for efficient multi-level user queries in SQL
- Support for efficient multi-level user queries in natural language
- Support for efficient multi-level user queries through filters and checkboxes
- Support for keyword-based search in images
- Render the filtered output in UI with column choosing support
- Return filtered output in downloadable CSV format (collaboration with Secure Download team needed)
- API support for easy integration and query from CLI

2.2 User Classes and Characteristics

End users are able to view and apply the search queries (SQL, filters, and natural language queries) on the available datasets. They can use the standard filters for the columns included in the CSV files to obtain the results. In addition, SQL and natural language queries will be available for retrieving results by connecting various tables using foreign keys. Once the results from the datasets have been obtained, a downloadable CSV file will be created for the user to download.

We classify users based on their levels of expertise and experience into the following three categories:

1. **Developers/SQL Experts:**

   Developers/end-users acquainted with SQL and the dataset will be allowed to enter SQL commands directly to query the dataset across multiple tables.

2. **Users familiar with the dataset:**

   Users familiar with the table columns and datatypes can directly impose conditions based on column names, and conditional filters by selecting from dropdowns, radio buttons, and checkboxes in the application UI directly.

3. **Users aiming to get a general overview of the data:**

   Users not familiar with individual data columns and datatypes can choose to merely enter the query text in natural language.

The user will be given the choice to select the mode of query input into our application.

2.3 Operating Environment

The web application will be running from a Linux Docker container and will access datasets hosted on a server.

2.4 User Documentation

- **UI docs:** We will release a detailed user manual on how to use the app UI with all its features.
- **Video walkthrough:** We will also make video walkthroughs of our application available.
- **API docs:** For developers, we plan to release Swagger documentation for our API. API docs will be available at the `/docs` endpoint of our backend API.

3. System Features

3.1 Process data in CSV format

Given that the end-user queries need to be processed on CSV, we need to convert the already uploaded datasets from other formats in CSV clusters. The dependency information and hierarchies between the SQL tables need to be stored as metadata. The metadata will also

contain data type information for the columns. Further queries will be processed on the converted data.

3.2 Custom dataset upload

We should have the support for users to upload additional datasets in CSV format. A fields form will be required to be filled by the user while uploading data to keep track of the metadata. The data will be verified to check if it follows CSV standards. The metadata can be populated automatically if the column names are uniform through all the CSVs.

3.3 Dataset preview and description

This will be a high-level overview of the dataset as a whole and the dataset columns. We will also show a preview or head of the dataset. Apart from that, the data types of columns will also be shown for acquainting the user with the dataset.

3.4 Query suggestions

For the already uploaded datasets, we will provide query suggestions. We can store frequent queries locally for each user and provide a search engine kind of features on them using keyword matching for query suggestions. Apart from that, we can provide a set of example queries for each dataset outputs for which will be cached already for quick download.

3.5 Support for efficient user queries in SQL

For developers acquainted with SQL, our application provides support for entering queries directly as SQL. The SQL queries will be processed using the csvkit library. Apart from that, the query will be validated to check if it is a valid SQL query.

3.6 Support for efficient user queries in natural language

Users who are not acquainted with SQL can query the database using natural language. We can do TF-IDF matching or table question answering to identify the relevant columns. However, this might have limited support for a single table.

3.7 Support for efficient user queries through filters and checkboxes

Users who are not acquainted with the dataset itself can query the dataset using the relevant filters. This will be a GUI-based tool to provide primary, secondary and tertiary levels of the filter.

People can choose particular columns from the dropdown and query the database recursively up to 3 times.

3.8 Support for keyword-based search in images

For the columns that will have images, we can do a search for the images. Our backend API will use object recognition models like CLIP to check if the object mentioned in the keyword is present in the dataset of images.

3.9 Render the filtered output in UI with column choosing support

The output of the query will be rendered as a table in the web app. The user will also have the option to choose which columns to view. If the output is huge, it will show a subset of the output as a preview.

3.10 Return filtered output in downloadable CSV format

After performing the query and filtering required columns, the user will be able to download the resultant table in a CSV format too.

3.11 API support for easy integration and query from CLI

For developers and anyone else who wants to query from their own app or CLI, we will also provide API support. All the above functionalities will be possible to make through respective API calls, documented with demo support using Swagger UI. The API support will also make integration with the umbrella application easier.

4. Other Nonfunctional Requirements

4.1 Performance Requirements

The natural language user query text will be parsed with a pretrained deep learning NLP model that should be able to return inferences with minimal delay. Frequent historical queries will be cached locally, along with their outputs. Since querying and filtering large volumes of data is an expensive operation, outputs for suggested and frequent queries will be cached locally for fast download. The pretrained image recognition model for image queries should also be able to return inferences efficiently and quickly. Expensive query operations will be run asynchronously in the background, allowing the user to interact with other components in the application. The

state changes of various components, including rendering dataset previews should be also quick and efficient to ensure a seamless user experience.

4.2 UI Requirements

The application needs to have an easy-to-use user-friendly interface with minimal lag during the state change of various UI components. Since the datasets may be very large in size, we will restrict the user preview to the first 15 rows of the query output, and allow for horizontal scrolling to view all columns. We also plan to provide in-app tooltips for various buttons and dropdowns to make their functionalities always available at the user's fingertips.

4.3 Security Requirements

We will collaborate with the Secure Download project team from our course for ensuring the security of filtered datasets that are to be downloaded.

4.4 Software Quality Attributes

The codebase for the backend, as well as the frontend of the application will be hosted publicly on GitLab following **efficient language standards** such as PEP8 and ES6, and industry coding practices such as dostrings, modularization and linting. Efficient **version control techniques** such as opening issues, pull requests, regular commits and well-documented README and Licenses will be followed for the entire codebase on GitHub and instructions to run the code locally will be also provided for developers. The pretrained NLP model, as well as the application will be hosted in a Docker container and deployed for direct access and demonstration, and Swagger documentation will be made available for a live demo of each API endpoint.

Design Documentation

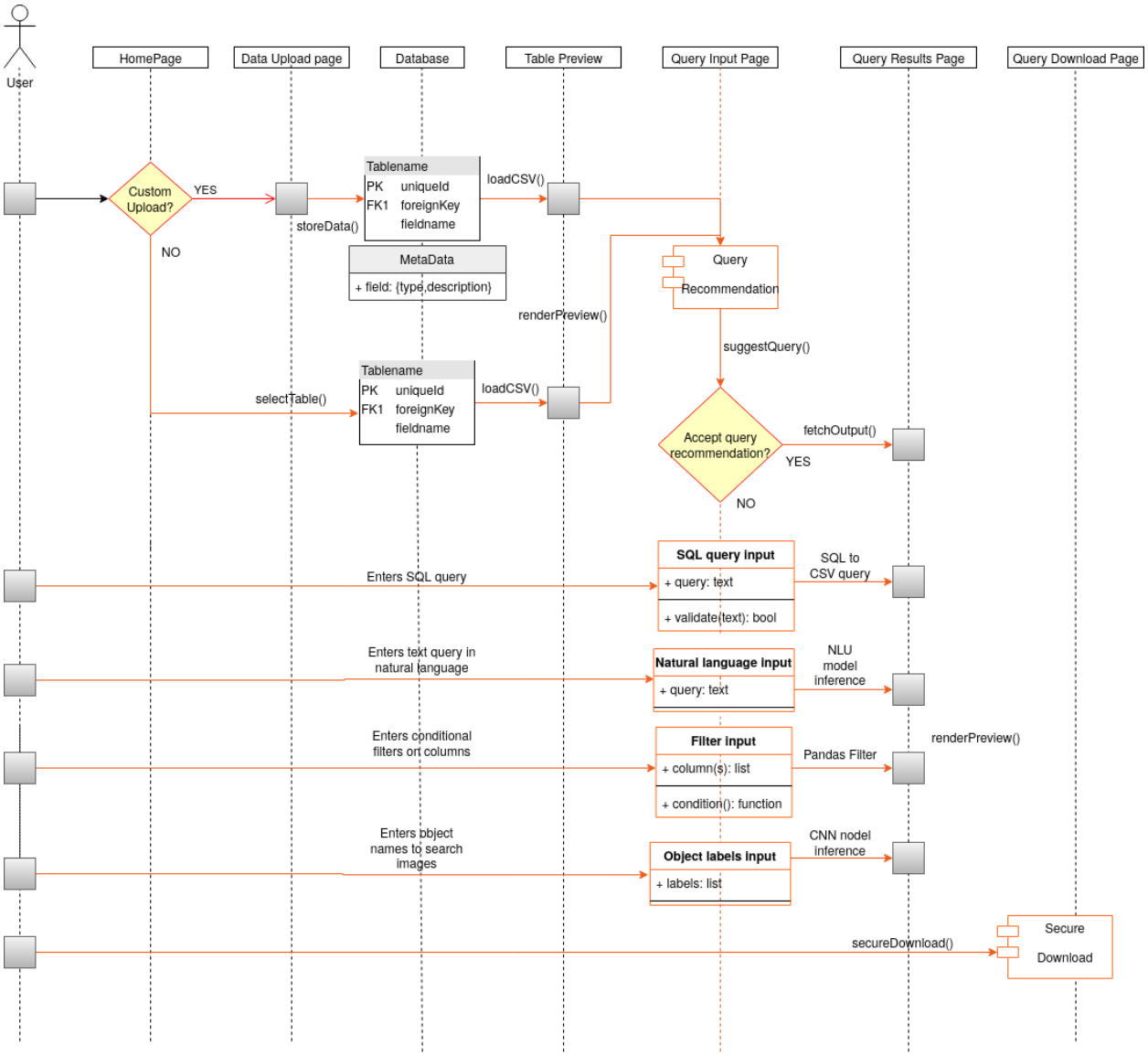
1. High Level Approach

The proposed application, as discussed in the Requirements Specification, handles queries on large datasets by users of varied experience levels and needs and must be designed to allow for a seamless user experience. On a high level, the application design consists of first the Dataloader, which converts other data formats into CSV and reads it along with the relevant metadata. Various types of queries are input using the Query Selector and handled separately by the Query Processor with support for SQL query parsing, natural language understanding and columnwise filtering of data. Keyword-based queries on images are also supported. The Dataset Preview unit embeds a preview of a truncated version of the table to be queried on, and later the Output Preview unit displays the query output in the application itself, also providing options for secure Output Download. What makes our application intelligent is its in-built Query Recommendation system, consisting of Query Logs keeping track of frequent historical queries, locally cached outputs for the most frequent queries by the user and auto-suggestions of a few standard queries.

2. Proposed Design

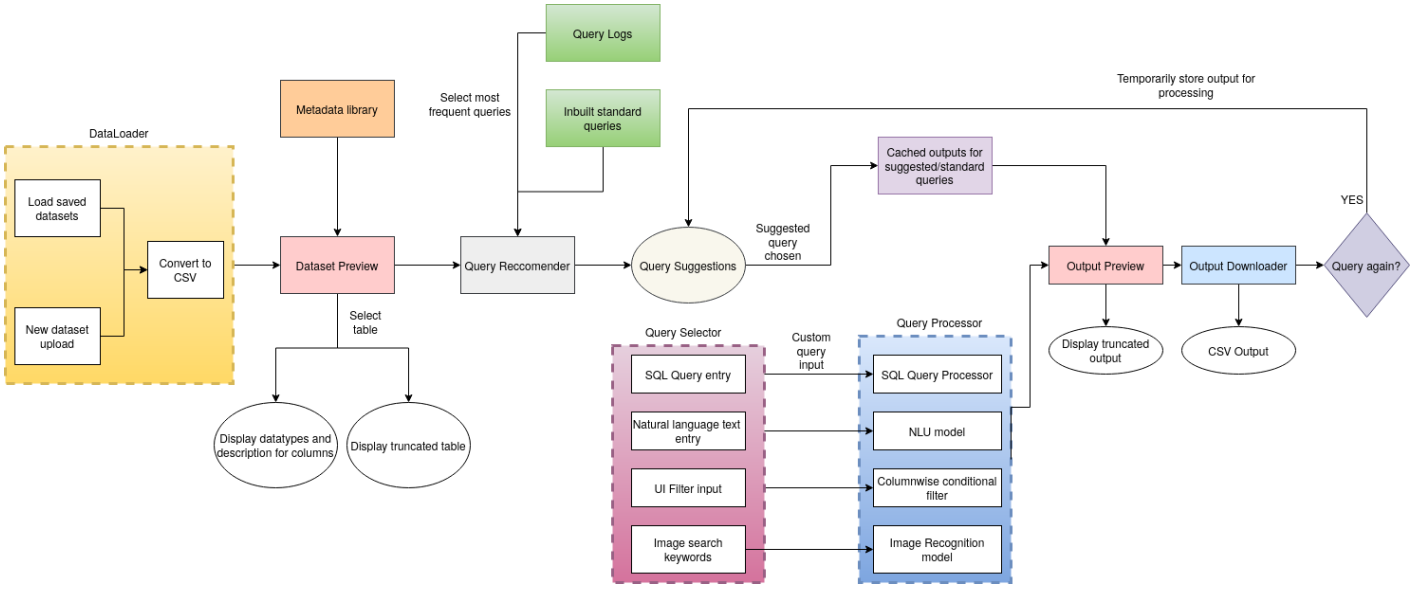
2.1. Project Workflow

The following UML diagram discusses the user sequence workflow.



2.2. Building Blocks

The connections of various functional blocks of the application are depicted in the block diagram below. Note circle blocks denote consumable outputs, and the squares indicate functional units. The functionalities of each block are also discussed below.



1. **DataLoader:**

   The dataloader incorporates two pipelines - a. Loading already hostel datasets, b. Uploading new custom dataset. In case the saved or uploaded dataset is not in CSV format, we will convert JSON/SQL datasets into CSV format for processing by our application using the `csvkit` library in Python.

2. **Dataset Preview:**

   The dataset preview component allows the user to select each table from the database, and embeds a truncated head() view of the first 15 rows of the selected table for preview to the user. Further, this component also provides information about the columns in the dataset with their individual data types and descriptions from the metadata. To do so, the dataset preview component will interface with the metadata library.

3. **Query Logs:**

   The history of previous logs by the user will be saved locally along with their frequencies so that the outputs for the most frequent queries can be cached and made available for instant download to the user. A few pre-determined standard queries will be also saved for every user, and the outputs for the same will also be cached similarly.

4. **Query Recommender:**

   For intelligent query recommendation, the query recommender component will show the standard queries, as well as the most frequent queries saved locally to the user and

allow the user to choose a suggested query. If the user is not interested in the suggested queries, the user will be redirected to the query input page.

5. Query Selector:

The query selector provides options for the user to enter a query:

- **SQL query:** Direct SQL command input
- **Natural language query:** Text input as natural language by the user
- **UI Filtering:** Filtering based on the selection of column names from dropdowns, and checkboxes to impose conditions on each column
- **Keyword-based image search**: User inputs comma separated keywords as object names to search for among the images.

6. Query Processor:

Each type of query will be handled through a separate pipeline, under the umbrella of the query processor unit. Based on the four types of queries in the query selector, we will handle the queries as follows:

- **SQL Query:** The SQL command will be applied on the CSV dataset by using the `csvkit` library in Python.
- **Natural Language query:** The text input will be sent directly to the NLU model for inference. The algorithm for the NLU deep learning model will be discussed in Section 2.3.
- **UI Filtering:** For conditions imposed on columns, we can convert the user selections on the backend directly into a Pandas filter using lambda functions. This feature supports filtering across multiple columns easily.
- **Keyword-based image search:** The comma-separated list of keywords will be parsed on the backend and sent to the Image Recognition model to return the datarows corresponding to object matches.

7. Output Preview:

Similar to the Dataset Preview component, the output preview component also lets the user get a truncated output preview embedded into the application and provides dropdowns to the user to select the required columns to view and download. Conditional filtering is not allowed at this stage.

8. Output Downloader:

The user is provided with a button to indicate whether he wishes to download the filtered dataset output. Since the size of output files may be huge, a notification will be first displayed to the user indicating the start of a large file download, and the download manager will run asynchronously allowing for other user interactions with the application. The security aspect of the Data Download phase will be handled in collaboration with the Secure Download team.

2.3. Algorithms

SQL filter processor
We will use the csvkit library for this purpose. Csvkit has its own algorithms for running SQL queries on CSV files which we can use directly to run SQL commands entered by the user on the CSV data.

Natural language processor
We will use the algorithm from the paper, TableQA: Question Answering on Tabular Data that will be able to parse the queries and apply them on the dataset to return relevant results. Apart from that, we will also do TF-IDF matching to identify columns and filter according to them

Image processor
We can use the CLIP model to recognise the objects in the image and run a keyword search to check if the object queried is present in any image or not. Apart from that, we can also do image question answering to run natural language queries on the image.

3. High Level API

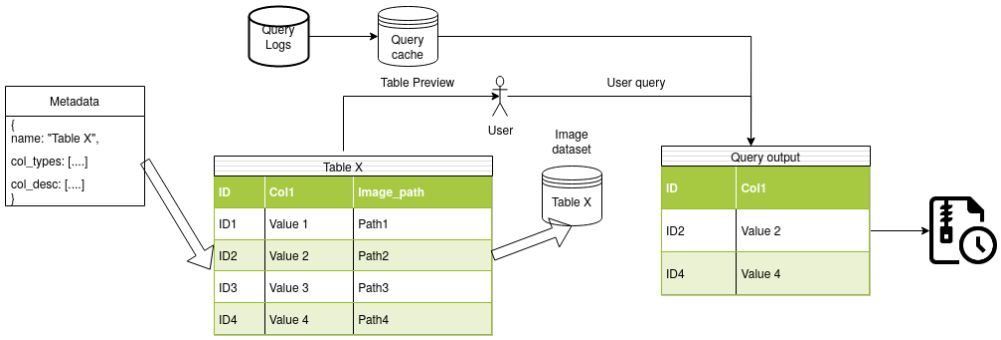
On a high level, the API will consist of the following endpoints:

- `/render_table:` Receives the table name as request and searches for the table in the dataset folder. Renders the head of the table as output. Used for both dataset preview and output preview.
- `/suggest:` Goes through query logs to suggest top 3 frequent queries to the user.
- `/sql_query:` Receives SQL query command in the request. If the user enters SQL query directly, this endpoint validates the query and uses the csvkit library to run the query on the selected tables.
- `/nl_query:` Receives the query text and table name in the request. Sends query text for processing and inference to the NLU model for running the query on the selected table.
- `/filter:` Receives the table name, column name(s) and condition in the request. Validates the condition based on the datatype of the column, and runs Pandas based query on the chosen columns.

- `/image_search`: Receives the table name and comma-separated list of object names to search for in the request. Sends the list of object labels to the Image Recognition model for inference and returns rows of the table with matching images.
- `/download`: Triggered when the user wishes to download the final output as CSV. Will interface directly with the API provided by the secure download team.

4. Database models

First of all, the models already stored as SQL in database needs to be converted to csv and all the subsequent queries will be performed on CSV files. For datasets, that contain images, the images will be stored in a folder on the server and there will be a column in the CSV files pointing to the filenames. The metadata of a dataset will be stored in JSON files. New tables created as a part of filtering will be deleted after a stipulated time after the user query completes. A query log will be maintained which will take care of the frequency of the queries. A query cache folder will be there which will store the tables of the most frequent queries for quick access. A rough diagram of the storage components is depicted in the figure below:



5. Tools/Libraries to be used

We intend to use the following tech stack for the application:

Unit	**Component**	**Library/framework**
Frontend	All UI components	ReactJS, HTML5, CSS3
Backend	Conversion to CSV	csvkit (Python)
	NLU model	PyTorch, Numpy, nltk
	Image recognition model	PyTorch, OpenCV

	Column Wise conditional filtering	Pandas
	SQL querying	csvkit (Python)
	Server and API	Flask

6. Deployment models

For Frontend - The ReactJS frontend application will be containerized using **Docker** and deployed on the suggested server. For a public demo, we can also deploy our Docker container on services like Heroku.

For Backend - The Flask backend server will also be containerized using **Docker** and deployed on the suggested server. For a public demo, we can also deploy our Docker container on services like Heroku.

Overall application - The docker containers of the frontend and backend will be bundled together using **Docker Compose** and deployed on the suggested server/Heroku.

ML models - The machine learning models will be pretrained and hosted on the suggested server for quick inference.

References:

https://csvkit.readthedocs.io/en/latest/

https://pypi.org/project/pandasql/

https://swagger.io/tools/swagger-ui/

http://aimedhub.iiit.ac.in/static/media/Gandhi_Data-20210427T035509Z-001.493f2f90.zip

https://blog.paperspace.com/tapas-question-answering/

https://towardsdatascience.com/analyze-csvs-with-sql-in-command-line-233202dc1241