Team Dogwood

Release 1

Gabriella Larrisa

Feb 28, 2025

TABLE OF CONTENTS:

This is the official documentation for **Team Dogwood's** project. Here, you'll find detailed information about the project's modules, configuration, and usage.

> **Note**
>
> This documentation is written using `reStructuredText` (reST). For more details on reST syntax, refer to the reStructuredText documentation.

CHAPTER

ONE

INSTALLATION GUIDE

To get started with the project, follow these steps:

1. **Clone the Repository**: If you haven't already, clone the repository using Git.

```
git clone https://github.com/imanzaf/ift_coursework_2024.git
```

2. **Install Python and Poetry**: - Ensure you have Python installed. You can download it from python.org. - Install Poetry by following the official Poetry installation guide.

3. **Navigate to the Project Directory**: Move into the cloned repository directory.

```
cd team_dogwood/coursework_one
```

4. **Install Dependencies**: Use Poetry to install the project dependencies.

```
poetry install
```

5. **Set Up Environment Variables**: Create a *.env* file in the root directory and populate it with the required environment variables.

6. **Run the Project**: Use Poetry to run the main script.

```
poetry run python main.py
```

CHAPTER

TWO

PROJECT STRUCTURE

The project is organized into the following modules:

- **`config`**: Contains configuration settings for the project.
- **`src`**: Contains the main source code, including data models, database interactions, and utility functions.

For more details, refer to the *src* section.

CHAPTER

THREE

API REFERENCE

3.1 src

Overview

This section provides an overview of the modules in the project. Modules are organized into packages to group related functionality, making the codebase more maintainable and reusable.

Key Features

- **Modular Design**: The project is divided into packages and modules, each handling a specific aspect of the system.
- **Reusable Components**: Modules are designed to be reusable across different parts of the project.
- **Clear Separation of Concerns**: Each module focuses on a specific responsibility, promoting clean and maintainable code.

3.1.1 src Packages

Overview

This section provides an overview of the main packages and modules in the project. Each package contains related functionality, organized into modules for better maintainability and reusability.

Key Packages

- **Data Models**: Defines the structure and validation rules for data entities.
- **Database**: Provides functionality for interacting with databases and storage services.
- **ESG Reports**: Handles searching and validation of ESG (Environmental, Social, and Governance) reports.
- **Utils**: Contains utility functions for data processing, searching, and other common tasks.

Data Models Package

This package contains the data models used in the project. These models define the structure of the data and provide validation using Pydantic. The models are designed to represent entities such as companies, ESG reports, and other related data.

Key Features

- Define the structure of data using Pydantic models.
- Validate data inputs to ensure consistency and correctness.
- Support for nested models and relationships between entities.

Data Models Module

Overview

The *company* module contains the *Company* and *ESGReport* classes, which represent companies and their associated ESG (Environmental, Social, and Governance) reports. These models are used to validate and structure data related to companies and their ESG activities.

Key Features

- **Company Class**: Represents a company with attributes such as stock symbol, name, sector, industry, location, and associated ESG reports.
- **ESGReport Class**: Represents an ESG report with attributes such as URL and publication year.

class src.data_models.company.**Company**(***data: Any*)

Bases: BaseModel

Represents a company with its associated details and ESG reports.

This class models a company, including its stock symbol, name, sector, industry, location, and any associated ESG (Environmental, Social, and Governance) reports.

Parameters

- **symbol** (*str*) – The stock symbol of the company.
- **security** (*str*) – The name of the company.
- **gics_sector** (*str*) – The GICS sector of the company.
- **gics_industry** (*str*) – The GICS industry of the company.
- **country** (*str*) – The country where the company is headquartered.
- **region** (*str*) – The region where the company is headquartered.
- **esg_reports** (*List[ESGReport], optional*) – A list of ESG reports associated with the company.

Example

```
>>> company = Company(
...     symbol="AAPL",
...     security="Apple Inc.",
...     gics_sector="Information Technology",
...     gics_industry="Technology Hardware, Storage & Peripherals",
...     country="United States",
...     region="North America",
...     esg_reports=[]
... )
>>> print(company.symbol)
AAPL
```

class src.data_models.company.**ESGReport**(** (Keyword-only parameters separator (PEP 3102))*, *url: str | None = None*, *year: str | None = None*)

Bases: BaseModel

Represents an ESG (Environmental, Social, and Governance) report.

This class models an ESG report, including its URL and the year it was published.

Parameters

- **url** (*Optional[str]*) – The URL of the ESG report.
- **year** (*Optional[str]*) – The year the ESG report was published.

Example

```
>>> report = ESGReport(
...     url="https://example.com/esg-report-2023.pdf",
...     year="2023"
... )
>>> print(report.url)
https://example.com/esg-report-2023.pdf
```

class src.data_models.company.**SearchResult**(***, *title: str*, *metatag_title: str | None = None*, *author: str | None = None*, *link: str*, *snippet: str*)

Bases: BaseModel

Represents a search result from a query.

This class models a search result, including its title, metatag title, author, link, and snippet.

Parameters

- **title** (*str*) – The title of the search result.
- **metatag_title** (*Optional[str]*) – The metatag title of the search result.
- **author** (*Optional[str]*) – The author of the page.
- **link** (*str*) – The link to the search result.
- **snippet** (*str*) – The snippet of the search result.

Example

```
>>> result = SearchResult(
...     title="Example Search Result",
...     metatag_title="Example Meta Title",
...     author="John Doe",
...     link="https://example.com",
...     snippet="This is an example search result snippet."
... )
>>> print(result.title)
Example Search Result
```

Database Package

This package contains modules for interacting with databases and storage services, including **PostgreSQL** for relational data and **MinIO** for object storage.

Database Minio Module

Overview

The *minio* module provides functionality for interacting with **MinIO**, an open-source object storage service. It includes methods for uploading, downloading, and managing files stored in MinIO buckets.

Key Features

- Upload files to MinIO buckets.
- Download files from MinIO buckets.
- Manage bucket policies and configurations.

class src.database.minio.**MinioFileSystem**(*bucket_name*, *, *user: str = 'ift_bigdata'*, *password: str = 'minio_password'*, *endpoint_url: str = 'localhost:9000'*)

Bases: `MinioFileSystemRepo`, `BaseModel`

Overwrite file read and file write methods in MinioFileSystemRepo to add functionality to process PDF files.

Variables

- **bucket_name** (*str*) – The name of the MinIO bucket.
- **user** (*str*) – The username for MinIO.
- **password** (*str*) – The password for MinIO.
- **endpoint_url** (*str*) – The endpoint URL used to connect to MinIO, consisting of the MinIO host address and port.

property client: Minio

Initializes and returns a Minio client.

Returns
A Minio client instance.

Return type
Minio

create_bucket(*bucket_name: str*)

Ensures the bucket exists. Creates it if it doesn't exist.

Parameters
bucket_name (*str*) – The name of the MinIO bucket.

Example

```
>>> minio = MinioFileSystem()
>>> minio.create_bucket("my-bucket")
# Creates a bucket named "my-bucket" if it doesn't exist.
```

download_file(*file_name: str*, *dest_path: str*)

Downloads a file from MinIO to a local path.

Parameters

- **file_name** (*str*) – The name of the file in the bucket.
- **dest_path** (*str*) – The local path to save the file (e.g., "./downloaded.pdf").

list_files_by_company(*company_id*)

Lists all files for a specific company by prefix 'company_id/'.

Parameters
company_id (*str or int*) – The company ID.

Returns
A list of object names belonging to that company's folder.

Return type
list

upload_pdf(*local_file_path: str*, *company_id: str*, *report_year: str*)

Uploads a PDF into a subfolder structure: company_id/year/filename.pdf.

Parameters

- **local_file_path** (*str*) – The path to the local PDF file.
- **company_id** (*str*) – The ID of the company for which the PDF is being uploaded.
- **report_year** (*str*) – The year of the CSR report.

Returns
The object name (MinIO path), e.g., "123/2024/report.pdf".

Return type
str

view_pdf(*object_name: str*, *expiry_hours: int = 1*)

Generates a presigned URL to view the PDF in a web browser.

Users can open the link in their browser without explicitly downloading.

Parameters

- **object_name** (*str*) – The MinIO path (e.g., "123/2024/report.pdf").
- **expiry_hours** (*int, optional*) – The expiry time for the presigned URL in hours. Defaults to 1.

Returns
A presigned URL string. Returns None if an error occurs.

Return type
str

write_pdf_bytes(*pdf_bytes: bytes*, *company_id: str*, *report_year: str*, *file_name: str*)

Uploads a PDF (as bytes) into a subfolder structure: company_id/year/filename.pdf.

Parameters

- **pdf_bytes** (*bytes*) – The PDF file as bytes.
- **company_id** (*str*) – The ID of the company for which the PDF is being uploaded.
- **report_year** (*str*) – The year of the CSR report.

- **file_name** (*str*) – The name of the file to be saved.

Returns
The object name (MinIO path), e.g., "123/2024/report.pdf".

Return type
str

Database Postgres Module

Methods for interacting with postgres database.

Overview

The *postgres* module provides functionality for interacting with **PostgreSQL**, a powerful open-source relational database. It includes methods for querying, inserting, updating, and deleting data in PostgreSQL tables.

Key Features

- Execute SQL queries.
- Insert, update, and delete records.
- Manage database connections and transactions.

class src.database.postgres.**PostgreSQLDB**

Bases: BaseModel

Methods for connecting to and interacting with the PostgreSQL database.

This class provides methods for connecting to a PostgreSQL database, executing SQL operations, and managing database sessions. It supports both read and upsert (update/insert) operations.

Parameters
BaseModel – Inherits from Pydantic's BaseModel for data validation and settings management.

Example

```
>>> db = PostgreSQLDB()
>>> with db:
...     db.execute("read", sql_statement="SELECT * FROM companies")
```

__enter__()

Enter the runtime context related to this object.

Returns
The instance of the PostgreSQLDB class.

Return type
PostgreSQLDB

__exit__(*exc_type*, *exc_value*, *traceback*)

Exit the runtime context and close the database connection.

Parameters

- **exc_type** – The exception type (if any).
- **exc_val** – The exception value (if any).

- **exc_tb** – The traceback (if any).

static _conn_postgres()

Create a connection engine for PostgreSQL.

Returns
A SQLAlchemy engine object.

Return type
sqlalchemy.engine.Engine

Raises
Exception – If an error occurs while creating the engine.

static _conn_postgres_psycopg2()

Establishes a raw psycopg2 connection to PostgreSQL.

delete_csr_report(*report_id*)

Deletes a CSR report record from the database by report_id.

execute(*query*, *params=None*)

Fetches data (SELECT) and returns a list of dictionaries.

get_csr_report_by_id(*report_id*)

Fetch a single CSR report by its primary key (report_id). (Assumes you have a 'report_id' column in your table.)

get_csr_reports_by_company(*company_id*)

Retrieve all CSR reports for a specific company, ordered by year desc.

property session

Create and return a SQLAlchemy session for database interactions.

Returns
A SQLAlchemy sessionmaker object.

Return type
sqlalchemy.orm.sessionmaker

store_csr_report(*company_id*, *report_url*, *report_year*)

Inserts a new CSR report record into the database. If (company_id, report_year) is unique, we use ON CONFLICT to avoid duplicates. Adjust if you have a primary key like 'report_id serial primary key'.

update_csr_report(*report_id*, *new_url=None*, *new_year=None*)

Updates a CSR report's URL and/or year based on report_id. Only updates fields that are provided.

ESG Reports Package

Overview

This package contains modules for handling ESG (Environmental, Social, and Governance) reports. It includes functionality for searching and validating ESG reports, ensuring data accuracy and consistency.

Key Features

- Search for ESG reports using various criteria.
- Validate ESG reports to ensure compliance with required standards.
- Integrate with external APIs and databases for data retrieval and validation.

ESG Reports Search Module

Overview

The *search* module provides functionality for searching ESG reports. It includes methods for querying external APIs or databases to retrieve ESG reports based on specific criteria such as company name, report year, or keywords.

Key Features

- Query ESG reports using flexible search parameters.
- Integrate with external APIs for real-time data retrieval.
- Filter and sort search results for better usability.

class src.esg_reports.search.**Search**(*, *company:* Company)

Bases: BaseModel

Search class for scraping ESG reports from various sources.

This class provides methods to search for and retrieve ESG (Environmental, Social, and Governance) reports for a given company using different sources:

1. Google Custom Search API - searches for PDF ESG reports.
2. Sustainability Reports website - scrapes reports from responsibilityreports.com.

Variables
company (Company) – The company to look for ESG reports for.

static _format_google_results(*search_results*) → list[*SearchResult*]

Formats Google search results into a list of SearchResult objects.

Parameters
search_results (*list*) – A list of search results from the Google API.

Returns
A list of formatted search results.

Return type
list[*SearchResult*]

Example

```
>>> search = Search(company=Company(symbol="AAPL", security="Apple Inc."))
>>> raw_results = [{"title": "Report 2023", "link": "http://example.com"}]
>>> formatted_results = search._format_google_results(raw_results)
>>> print(formatted_results)
[SearchResult(title="Report 2023", link="http://example.com")]
```

static _match_score(*text: str*, *search_term: str*) → int

Calculates a match score between the text and the search term.

Parameters

- **text** (*str*) – The text to match against.
- **search_term** (*str*) – The search term to match.

Returns
The match score (number of matching words).

Return type
int

Example

```
>>> search = Search(company=Company(symbol="AAPL", security="Apple Inc."))
>>> score = search._match_score("Apple Inc. Report 2023", "Apple Inc.")
>>> print(score)
2
```

google() → List[*SearchResult*] | None

Searches for ESG reports using the Google Custom Search API.

Returns
A list of search results or None if no results are found.

Return type
Union[List[*SearchResult*], None]

Example

```
>>> search = Search(company=Company(symbol="AAPL", security="Apple Inc."))
>>> results = search.google()
>>> if results:
...     for result in results:
...         print(result.title, result.link)
... else:
...     print("No results found.")
```

model_post_init(*context: Any*, */ (Positional-only parameter separator (PEP 570))*) → None

This function is meant to behave like a BaseModel method to initialise private attributes.

It takes context as an argument since that's what pydantic-core passes when calling it.

Parameters

- **self** – The BaseModel instance.
- **context** – The context.

sustainability_reports_dot_com() → *ESGReport*

Searches for sustainability reports on sustainabilityreports.com.

Returns
An ESGReport object containing the report URL and year, or None if no report is found.

Return type
ESGReport

Example

```
>>> search = Search(company=Company(symbol="AAPL", security="Apple Inc."))
>>> report = search.sustainability_reports_dot_com()
>>> if report.url:
...     print(f"Report URL: {report.url}, Year: {report.year}")
... else:
...     print("No report found.")
```

ESG Reports Validate Module

Overview

The *validate* module provides functionality for validating ESG reports. It ensures that ESG reports meet required standards and formats, and it performs checks for data completeness and accuracy.

Key Features

- Validate ESG report data against predefined standards.
- Check for missing or inconsistent data.
- Generate validation reports for further analysis.

class src.esg_reports.validate.**SearchResultValidator**(*, *company:* Company, *search_results:* *List[*SearchResult*]*)

Bases: BaseModel

Validates search results from Google for ESG reports.

This class validates search results based on the presence of: - The current or previous year. - The company name. - ESG-related keywords.

Variables

- **company** (Company) – The company to validate the search results for.
- **search_results** (*List[*SearchResult*]*) – The search results to validate.

_company_name_in_result(*result:* SearchResult) → bool

Checks if the company name is in the author field, title, snippet, or link.

Parameters
result (SearchResult) – The search result to validate.

Returns
True if the company name is found, otherwise False.

Return type
bool

Example

```
>>> result = SearchResult(
...     title="Apple ESG Report 2023",
...     snippet="ESG Report 2023",
...     link="https://www.apple.com/esg-report-2023/",
...     author="Apple Inc.",
... )
>>> validator = SearchResultValidator(company=Company(symbol="AAPL",␣
```

```
↪security="Apple Inc."), search_results=[])
>>> is_valid = validator._company_name_in_result(result)
>>> print(is_valid)
True
```

_keywords_in_result(*result:* SearchResult) → bool

Checks if ESG keywords are in the title, snippet, or link.

Parameters
result (SearchResult) – The search result to validate.

Returns
True if ESG keywords are found, otherwise False.

Return type
bool

Example

```
>>> result = SearchResult(
...     title="Apple ESG Report 2023",
...     snippet="ESG Report 2023",
...     link="https://www.apple.com/esg-report-2023/",
... )
>>> validator = SearchResultValidator(company=Company(symbol="AAPL",␣
↪security="Apple Inc."), search_results=[])
>>> has_keywords = validator._keywords_in_result(result)
>>> print(has_keywords)
True
```

_year_in_result(*result:* SearchResult) → str | None

Checks if the current or previous year is in the title, snippet, or link.

Parameters
result (SearchResult) – The search result to validate.

Returns
The year if found, otherwise None.

Return type
Union[str, None]

Example

```
>>> result = SearchResult(
...     title="Apple ESG Report 2023",
...     snippet="ESG Report 2023",
...     link="https://www.apple.com/esg-report-2023/",
... )
>>> validator = SearchResultValidator(company=Company(symbol="AAPL",␣
↪security="Apple Inc."), search_results=[])
>>> year = validator._year_in_result(result)
```

```
>>> print(year)
"2023"
```

property clean_company_name: str

Cleans the company name by removing legal suffixes and common articles.

Returns
The cleaned company name.

Return type
str

Example

```
>>> company = Company(symbol="AAPL", security="Apple Inc.")
>>> validator = SearchResultValidator(company=company, search_results=[])
>>> print(validator.clean_company_name)
"Apple"
```

model_post_init(*context: Any*, */*) → None

This function is meant to behave like a BaseModel method to initialise private attributes.

It takes context as an argument since that's what pydantic-core passes when calling it.

Parameters

- **self** – The BaseModel instance.
- **context** – The context.

search_results: List[*SearchResult*]

property validated_results: List[*ESGReport*]

Validates search results based on the presence of year, company name, and keywords.

Returns
A list of validated ESG reports.

Return type
List[*ESGReport*]

Example

```
>>> company = Company(symbol="AAPL", security="Apple Inc.")
>>> results = [
...     SearchResult(
...         title="Apple ESG Report 2023",
...         metatag_title="Apple Inc. (AAPL) ESG Report 2023",
...         author="Apple Inc.",
...         link="https://www.apple.com/esg-report-2023/",
...         snippet="ESG Report 2023",
...     )
... ]
>>> validator = SearchResultValidator(company=company, search_
```

```
↪results=results)
>>> validated_results = validator.validated_results
>>> for result in validated_results:
...     print(result.url, result.year)
"https://www.apple.com/esg-report-2023/" 2023
```

Utils Package

Overview

This package contains utility modules that provide reusable functionality for data processing, searching, and other common tasks. These modules are designed to simplify development and promote code reuse across the project.

Key Features

- Data processing and transformation utilities.
- Search functionality for querying and filtering data.
- Reusable helper functions for common tasks.

Utils Data Module

Overview

The *data* module provides utility functions for processing and transforming data. It includes methods for cleaning, formatting, and manipulating data to ensure consistency and usability.

Key Features

- Clean and preprocess raw data.
- Transform data into required formats.
- Handle missing or inconsistent data.

src.utils.data.**download_pdf_from_urls**(*urls: List[str]*, *root_path: str*)

Downloads PDF files from a list of URLs.

This function attempts to download a PDF file from each URL in the list. It stops on the first successful download and returns the path to the downloaded file. If no URLs are successful, it logs an error and returns *None*.

Parameters

- **urls** (*List[str]*) – A list of URLs to attempt downloading PDFs from.
- **root_path** (*str*) – The directory path where the downloaded PDF will be saved.

Returns

The path to the downloaded PDF file if successful, otherwise *None*.

Return type

Union[str, None]

Example

```
>>> urls = [
...     "https://example.com/report.pdf",
...     "https://example.com/report",
```

```
... ]
>>> root_path = "./downloads"
>>> downloaded_file = download_pdf_from_urls(urls, root_path)
>>> if downloaded_file:
...     print(f"Downloaded file: {downloaded_file}")
... else:
...     print("Failed to download any file.")
Downloaded file: ./downloads/report.pdf
```

Utils Search Module

Overview

The *search* module provides utility functions for searching and filtering data. It includes methods for querying datasets and retrieving relevant results based on specific criteria.

Key Features

- Query datasets using flexible search parameters.
- Filter and sort search results for better usability.
- Support for advanced search operations.

`src.utils.search.`**`clean_company_name`**(*name: str*) → str

Cleans a company name by removing legal suffixes, common articles, and extra spaces.

This function removes common legal suffixes (e.g., “Inc”, “Ltd”, “LLC”) and articles (e.g., “a”, “the”, “and”) from the company name. It also strips any leading or trailing whitespace.

Parameters
name (*str*) – The raw company name to clean.

Returns
The cleaned company name.

Return type
str

Example

```
>>> raw_name = "Apple Inc. and The Company"
>>> cleaned_name = clean_company_name(raw_name)
>>> print(cleaned_name)
"Apple Company"
```

3.2 config

Overview

This section provides an overview of the configuration modules in the project. These modules handle settings and configurations for various components, such as database connections and search functionality.

Key Features

- **Database Settings**: Manages database connection settings and configurations.
- **Search Settings**: Handles configurations for search-related functionality.

3.2.1 Database Settings Module

class config.db.**DataBaseSettings**(*_case_sensitive: bool | None = None*, *_nested_model_default_partial_update: bool | None = None*, *_env_prefix: str | None = None*, *_env_file: DotenvType | None = WindowsPath('.')*, *_env_file_encoding: str | None = None*, *_env_ignore_empty: bool | None = None*, *_env_nested_delimiter: str | None = None*, *_env_parse_none_str: str | None = None*, *_env_parse_enums: bool | None = None*, *_cli_prog_name: str | None = None*, *_cli_parse_args: bool | list[str] | tuple[str, ...] | None = None*, *_cli_settings_source: CliSettingsSource[Any] | None = None*, *_cli_parse_none_str: str | None = None*, *_cli_hide_none_type: bool | None = None*, *_cli_avoid_json: bool | None = None*, *_cli_enforce_required: bool | None = None*, *_cli_use_class_docs_for_groups: bool | None = None*, *_cli_exit_on_error: bool | None = None*, *_cli_prefix: str | None = None*, *_cli_flag_prefix_char: str | None = None*, *_cli_implicit_flags: bool | None = None*, *_cli_ignore_unknown_args: bool | None = None*, *_cli_kebab_case: bool | None = None*, *_secrets_dir: PathType | None = None*, ***, *POSTGRES_DRIVER: str*, *POSTGRES_USERNAME: str*, *POSTGRES_PASSWORD: str*, *POSTGRES_PORT: str*, *POSTGRES_HOST: str*, *POSTGRES_DB_NAME: str*, *MINIO_USERNAME: str*, *MINIO_PASSWORD: str*, *MINIO_HOST: str*, *MINIO_PORT: str*, *MINIO_BUCKET_NAME: str*)

Bases: `BaseSettings`

Configuration for database settings, including PostgreSQL and MinIO.

This class defines the configuration settings required to connect to a PostgreSQL database and a MinIO storage service. The settings are loaded from environment variables or a *.env* file.

Variables

- **POSTGRES_DRIVER** (*str*) – The driver used to connect to the PostgreSQL database.
- **POSTGRES_USERNAME** (*str*) – The username for the PostgreSQL database.
- **POSTGRES_PASSWORD** (*str*) – The password for the PostgreSQL database.
- **POSTGRES_PORT** (*str*) – The port on which the PostgreSQL database is running.
- **POSTGRES_HOST** (*str*) – The host address of the PostgreSQL database.
- **POSTGRES_DB_NAME** (*str*) – The name of the PostgreSQL database.
- **MINIO_USERNAME** (*str*) – The username for the MinIO storage service.
- **MINIO_PASSWORD** (*str*) – The password for the MinIO storage service.
- **MINIO_PORT** (*str*) – The port on which the MinIO service is running.
- **MINIO_BUCKET_NAME** (*str*) – The name of the bucket in MinIO where files are stored.

Example

```
>>> database_settings = DataBaseSettings()
>>> print(database_settings.POSTGRES_HOST)
localhost
```

3.2.2 Search Settings Module

class config.search.**SearchSettings**(*_case_sensitive: bool | None = None*, *_nested_model_default_partial_update: bool | None = None*, *_env_prefix: str | None = None*, *_env_file: DotenvType | None = WindowsPath('.')*, *_env_file_encoding: str | None = None*, *_env_ignore_empty: bool | None = None*, *_env_nested_delimiter: str | None = None*, *_env_parse_none_str: str | None = None*, *_env_parse_enums: bool | None = None*, *_cli_prog_name: str | None = None*, *_cli_parse_args: bool | list[str] | tuple[str, ...] | None = None*, *_cli_settings_source: CliSettingsSource[Any] | None = None*, *_cli_parse_none_str: str | None = None*, *_cli_hide_none_type: bool | None = None*, *_cli_avoid_json: bool | None = None*, *_cli_enforce_required: bool | None = None*, *_cli_use_class_docs_for_groups: bool | None = None*, *_cli_exit_on_error: bool | None = None*, *_cli_prefix: str | None = None*, *_cli_flag_prefix_char: str | None = None*, *_cli_implicit_flags: bool | None = None*, *_cli_ignore_unknown_args: bool | None = None*, *_cli_kebab_case: bool | None = None*, *_secrets_dir: PathType | None = None*, ***, *GOOGLE_API_URL: str*, *GOOGLE_API_KEY: str*, *GOOGLE_ENGINE_ID: str*, *SUSTAINABILITY_REPORTS_API_URL: str*)

Bases: `BaseSettings`

Configuration for search settings, including Google API and sustainability reports API.

This class defines the configuration settings required to interact with the Google Custom Search API and a sustainability reports API. The settings are loaded from environment variables or a *.env* file.

Variables

- **GOOGLE_API_URL** (*str*) – The base URL for the Google Custom Search API.
- **GOOGLE_API_KEY** (*str*) – The API key for the Google Custom Search API.
- **GOOGLE_ENGINE_ID** (*str*) – The search engine ID for the Google Custom Search API.
- **SUSTAINABILITY_REPORTS_API_URL** (*str*) – The base URL for the sustainability reports API.

Example

```
>>> search_settings = SearchSettings()
>>> print(search_settings.GOOGLE_API_URL)
https://www.googleapis.com/customsearch/v1
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

C

S

INDEX

Symbols

C

D

E

G

L

M

P

S

U

V

W