InkSight System Maintenance Guide

Project Title: ‘Tell me something I don’t know’: Generating self-insights for creative writers with NLP

Application Name: InkSight

Team: The Three Musketeers

Team Members:

- Micah Crossett (18122181)
- Montaka Khan (21472447)
- Quan Ngoc Minh Vuong (22495054)

Table of Contents:

1. Installation and Setup

Quick Setup (Automated)

Assuming you already have the required dependencies installed, you can choose one of the following methods:

1. *Run the setup command:*
   Open a Command Prompt or PowerShell terminal and enter: "**npm run full-setup-with-fasttext**". Note: for installation of everything but semantic clusters and fastext try: "**npm run full-setup**".

2. *Run the Python setup script:*
   Navigate to api/utils/ and either:
   - Double-click the "**full-installation.py**" file, or
   - Run it manually using your Python interpreter (e.g: "**python "full-installation.py**")

For Detailed Instructions and Manual Setup See the User Manual

Section 2.2 (Quick Start) and Section 3.2 (Detailed Installation)

2. Software Updates

Updating Node.js Dependencies

- Update root dependencies with the command in a windows terminal in the project root: “**npm update**”

- Update API dependencies - navigate to the project api folder and install there.
  “cd api”
  “npm update”
  “cd ..”

- Check for outdated packages:
  “npm outdated”
  “npm outdated --prefix api”

Updating Python Dependencies

- Update all packages: **“pip install --upgrade -r api/utils/requirements.txt”**

- Update individual packages:
  “pip install --upgrade nltk”
  “pip install --upgrade fasttext-wheel”
  “pip install --upgrade scikit-learn”
  “pip install --upgrade numpy”
  “pip install --upgrade scipy”
  “pip install --upgrade statsmodels”

  Check installed versions: **“pip list”**.
  Ensure pip is also updated to the latest version.

Updating NLTK Data

- Run the python script by running this command in the project root: **“python api/utils/setup_nltk.py”**

Updating or Changing FastText Models

1. Download new model gz from the FastText website
2. Extract the model bin
3. Replace existing model in models/ directory
4. Update MODEL_PATH in **“api/utils/semantic.py”** to reflect new filename

3. Application Design

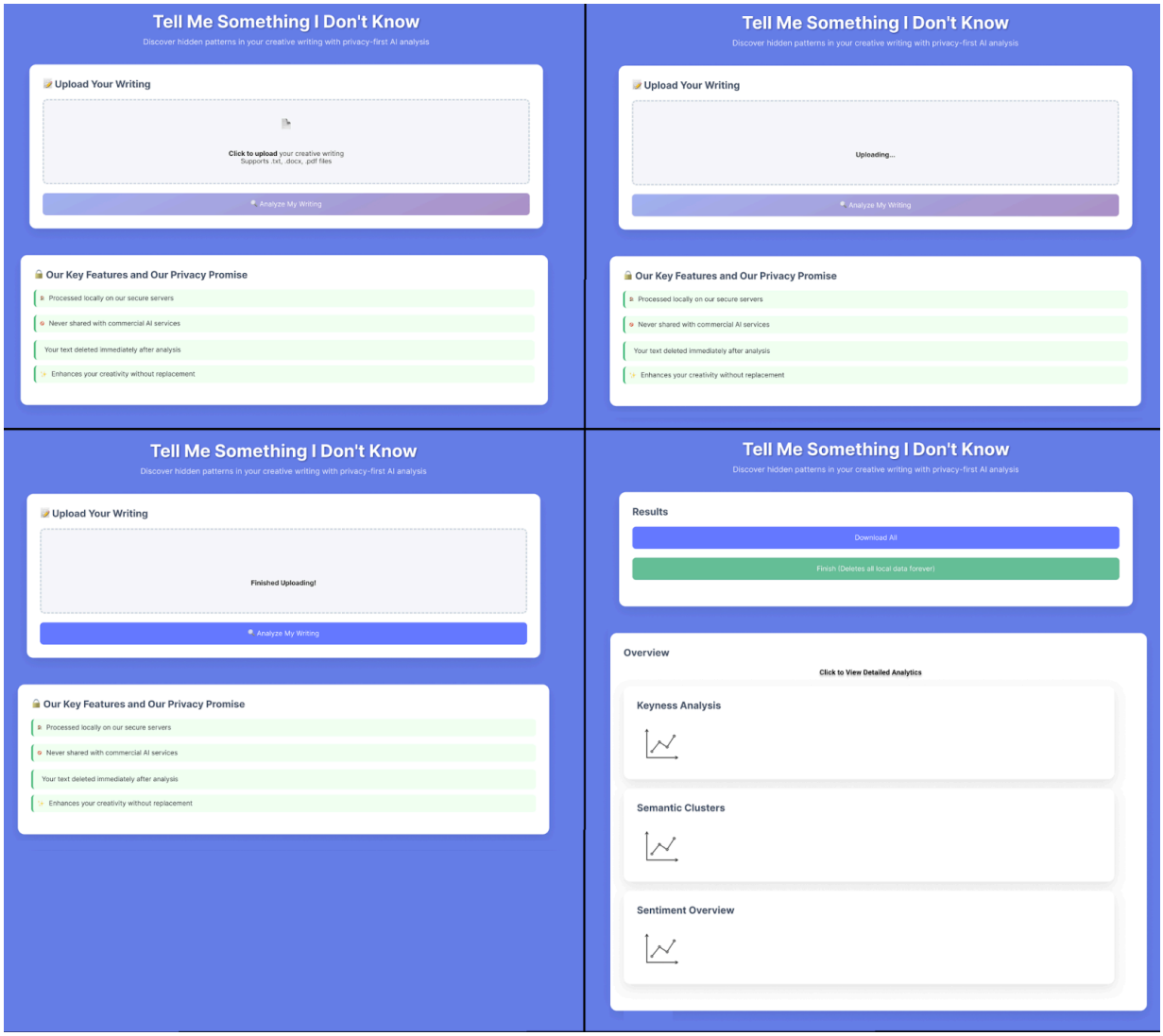
Prototype Design

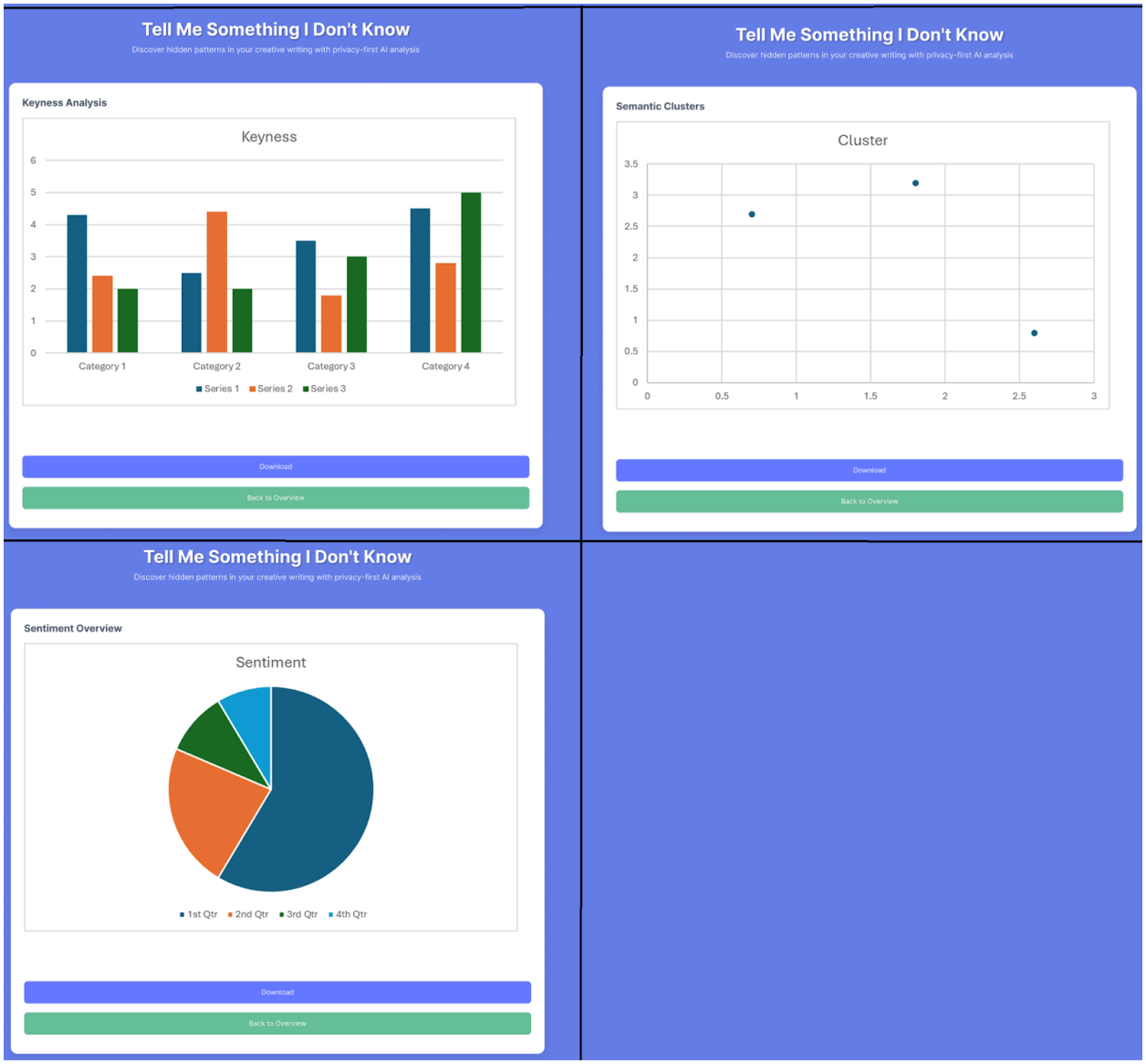
(Link attached for interactive figma prototype)

https://www.figma.com/proto/auUnYdpg0Oy5NmzlGUrWyv/Tradeshow--InkSight-?node-id=0-1&t=DEX0y66jEyrl22if-1

The Figma prototype demonstrates the core user journey and interface design for InkSight's privacy focused NLP analysis workflow.

Home Page/Initial SPA State:
The prototype shows each step of the user experience: starting with an empty upload box, then showing the file uploading, and finally displaying the results. Each screen gives clear feedback so users know exactly what's happening at every stage. The "drop to paste" zone shows which file types work, which helps prevent mistakes.





Results:

- Results Display - Analysis results are displayed as expandable cards that users can open or close individually.
- Visualization Approach - We chose to design the analysis sections to display charts for visual clarity. Bar charts show keyness comparisons, scatter plots map semantic clusters, and pie charts break down sentiment data.
- User Flow - Users follow a linear path from upload to download, with the option to return to the overview at any point to explore different analysis types.

3.1 APPLICATION DESIGN - Design Rational

**Note: We did not bundle with Webpack or Vite etc as we did not want to overengineer. Npm and python scripts handle all dependencies. Docker was also considered to maintain dependency compatibility (especially with FastTexts' python version requirements), however ultimately we decided again that too.

Front end: Javascript

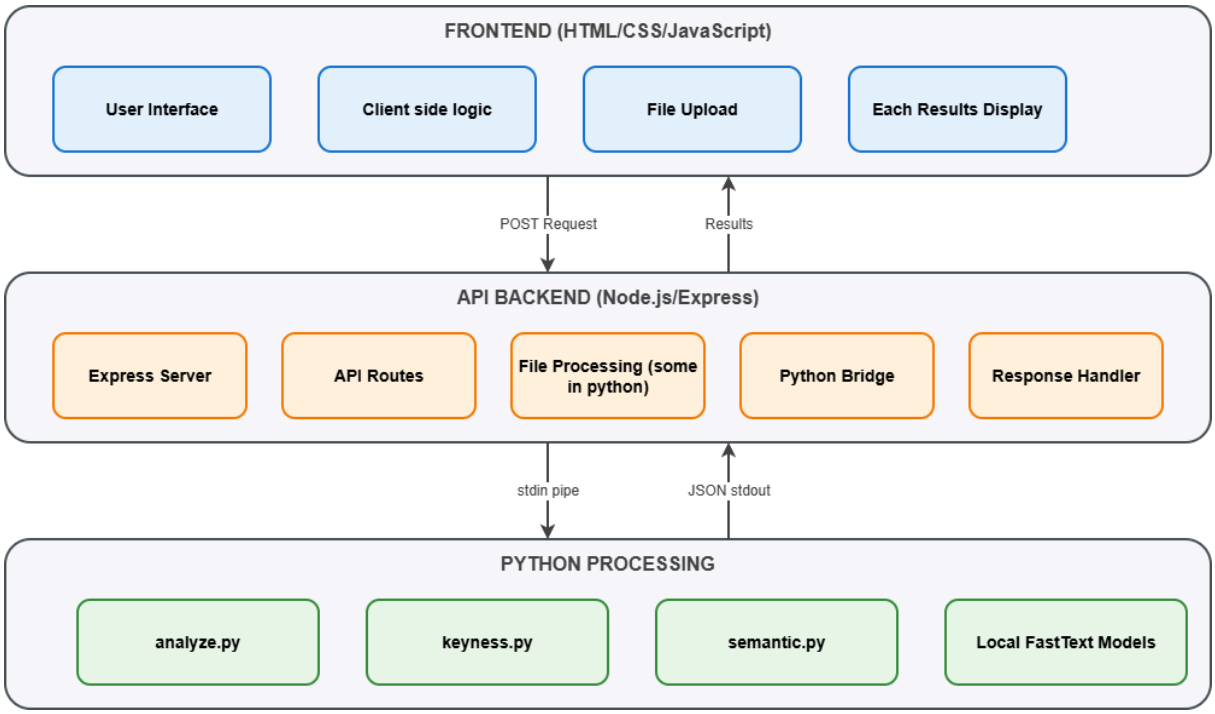
We chose vanilla JS for the front end because it was simple and Micah wanted more practice with just JS. Our design morphed into a SPA (Single Page Application), and in hindsight React might have been a better choice.

Backend Server: Javascript

- The backend server choice was NextJS/Express for familiarity.

Backend Processing: Python

- We also chose python for the Language processing (and our testing automation/setup installation scripts) because it was simple and research indicated it was a good choice for NLP.



4. DEPENDENCIES

Node.js Dependencies

- express: ^5.1.0 (MIT License)
- mammoth: ^1.11.0 (BSD-2-Clause License)
- multer: ^2.0.2 (MIT License)

Python Dependencies

- nltk ≥ 3.8 (Apache 2.0 License)
- statsmodels ≥ 0.13.0 (BSD License)
- scipy ≥ 1.9.0 (BSD License)
- fasttext-wheel ≥ 0.9.2 (MIT License)
- scikit-learn ≥ 1.0.0 (BSD License)
- numpy ≥ 1.21.0 (BSD License)

NLTK Data Packages

- punkt_tab (Apache 2.0 License)
- brown (Public Domain)
- gutenberg (Public Domain)
- reuters (Public Domain)
- inaugural (Public Domain)
- stopwords (Apache 2.0 License)

Trained Models

- FastText cc.en.100.bin (MIT License)
- Source: Facebook AI Research
- Location: models/ directory (project root)
- Size: ~1.5 GB (100-dim) or ~4.4 GB (300-dim)

License Summary

- MIT License: Permissive, commercial use allowed
- BSD License: Permissive, commercial use allowed
- Apache 2.0: Permissive, commercial use allowed
- Public Domain: No restrictions

5. MINIMUM REQUIREMENTS FOR INSTALLATION

Operating System

- Windows
- macOS / Linux (compatible, but installation may vary)

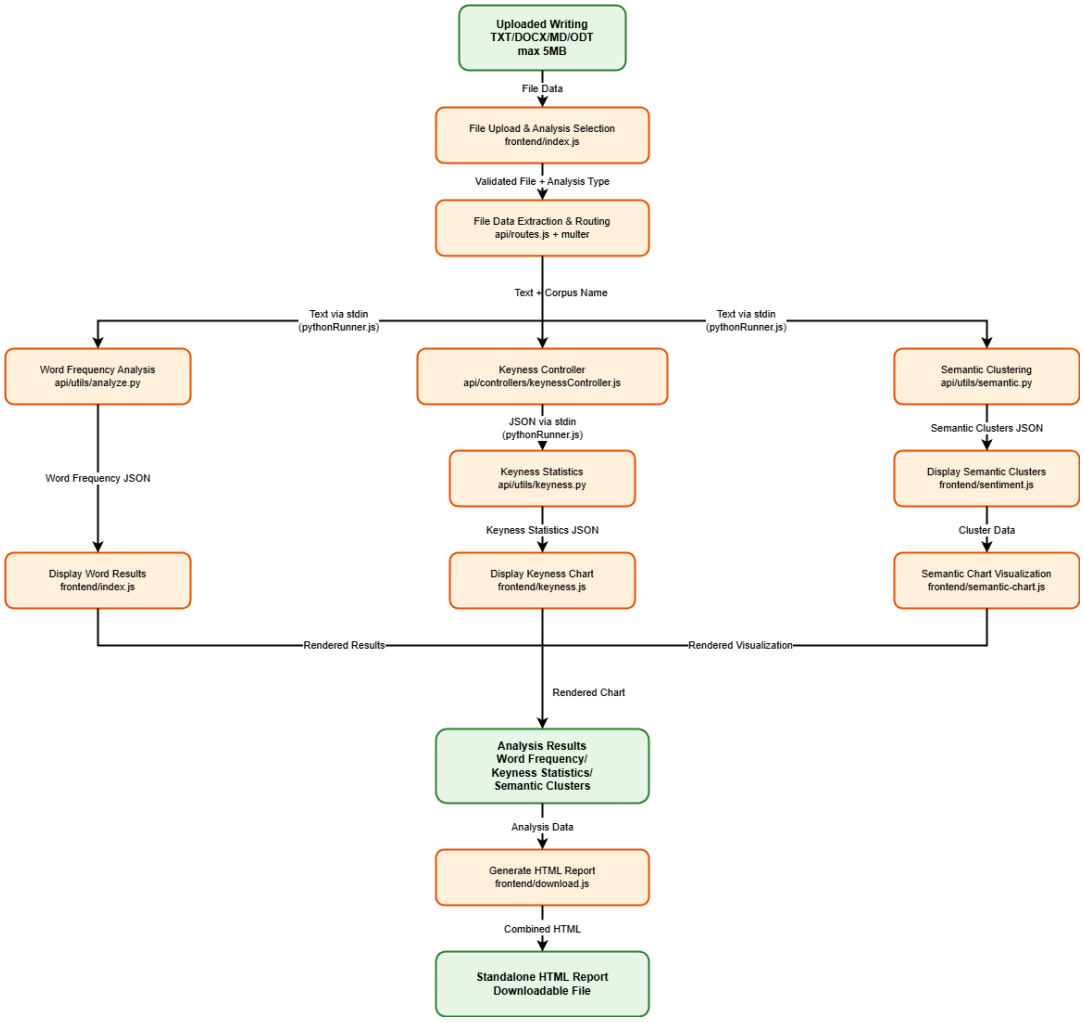
Software Requirements

- Node.js v18 or higher
- npm v9 or higher
- Python v3.11.9
- pip (latest version)

Hardware Requirements

- Disk Space: 2 GB minimum

6. System Architecture and Data Flow



Data Flow

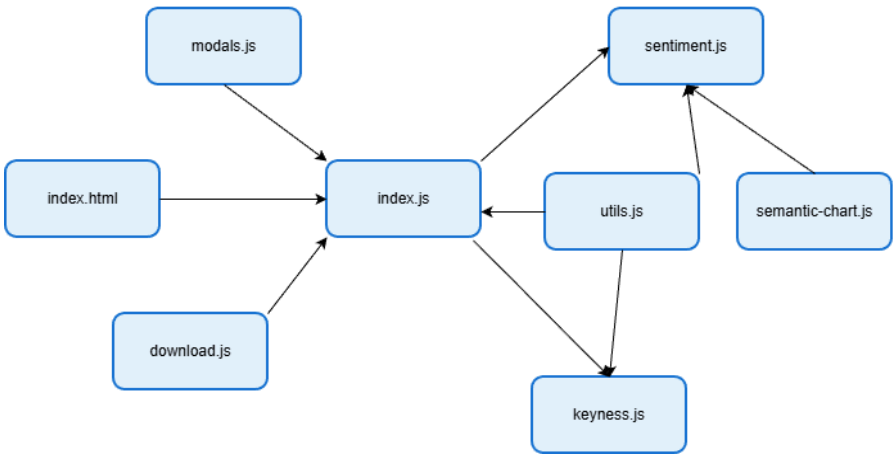
1. User uploads file → Frontend validates (max 5MB)
2. POST to /api/analyze-file, /api/keyness-stats, or /api/sentiment
3. Express extracts text in-memory (no disk writes)
4. pythonRunner.js spawns Python subprocess, pipes text via stdin
5. Python processes with local models, outputs JSON to stdout
6. Node.js returns JSON → Frontend renders results

Key Files

Frontend (frontend/)

File / Folder	Purpose
index.html	Main user interface (UI) entry point
index.js	Handles file uploads, API calls, and rendering of analysis results
modals.js	Modal management (defines Modal class and loadModals function)
utils.js	Shared utility functions (escapeHtml, toggleCardContent, toggleFormulas, etc.)
keyness.js	UI and visualization for keyness statistics
sentiment.js	Handles semantic analysis UI
semantic-chart.js	Chart visualization for semantic clustering (under development)
download.js	Generates downloadable HTML analysis reports
styles.css	Frontend styling (CSS)
modals/	Contains modal HTML files: About, How It Works, Privacy, Clear confirmation

Frontend Dependencies

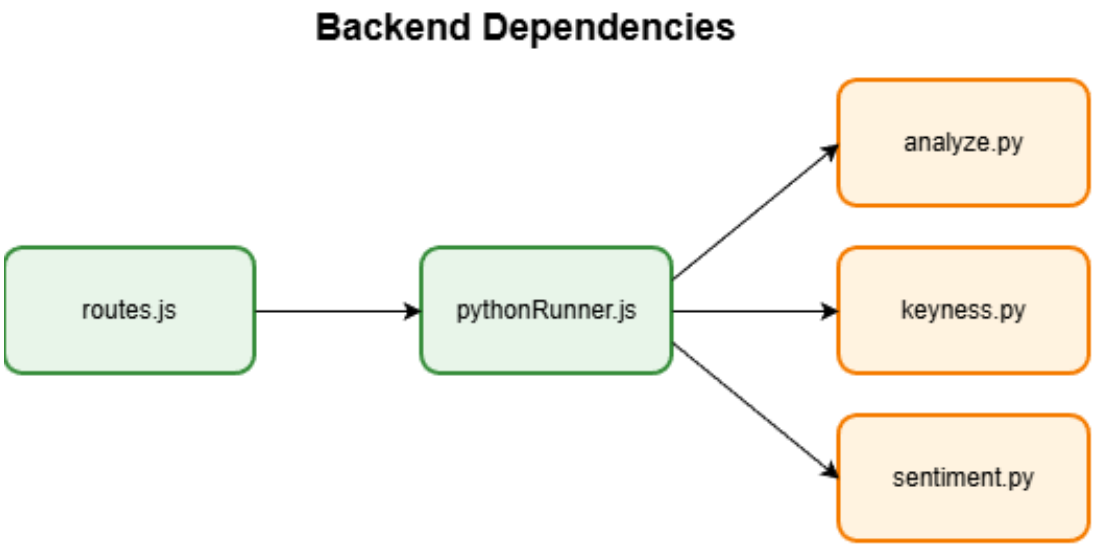


Backend Server (api/)

File / Folder	Purpose
app.js	Express.js server setup (runs on port 3000)
routes.js	Defines API routes and endpoints, and route handling
controllers/keynessController.js	Wraps the route handling and stores corpus data + functions
utils/pythonRunner.js	Utility to run Python subprocesses (for analysis tasks)

Python Processing (api/utils/)

File	Purpose
analyze.py	Performs word frequency analysis using regex tokenization
keyness.py	Calculates keyness statistics via chi-squared tests with NLTK corpora
sentiment.py	Performs semantic clustering using FastText + KMeans (name is legacy)
corpora.py	Manages corpus metadata



7. Key Feature Implementation

File Upload

- Max 5MB (defined in frontend/index.js:15)
- Formats: .txt, .md (UTF-8), .docx (mammoth library)
- Soft support for .odt (UTF-8 fallback)
- In-memory processing via multer

Python Communication (via pythonRunner.js)

- Utility function used by all routes accessing the python processing.
- Runs the python subprocess
- Text piped to stdin, JSON read from stdout

Frontend State

- lastResult - word analysis data
- sentimentData - semantic analysis data
- keynessData - keyness data
- Cleared on page refresh or manual clear

HTML Download

- download.js generates standalone HTML reports
- Options: Download All or individual section downloads

7.1 Keyness Statistics

What It Does
Keyness analysis identifies words that are statistically distinctive in your text compared to a reference corpus using chi-squared tests and effect size calculations (Cohen's h).

Technical Pipeline
Endpoint: POST /api/keyness-stats

Data Flow:

1. Frontend sends file and selected corpus name
2. Express extracts text in-memory
3. pythonRunner.js spawns keyness.py and pipes text via stdin
4. Python tokenizes text (lowercase, minimum 3 characters, alphabetic only)
5. Loads NLTK corpus and tokenizes identically
6. Calculates chi-squared test and Cohen's h effect size for each word
7. Filters to $p < 0.05$ (statistically significant only)
8. Sorts by absolute effect size (most distinctive first)
9. Returns JSON and frontend renders chart in keyness.js

Calculations Performed

- **Chi-squared test** (via scipy.stats.chi2_contingency): Determines if word frequency difference is statistically significant
- **Cohen's h** (effect size): Measures magnitude of difference between user text and corpus proportions
- **Significance markers:** *** ($p<0.001$), ** ($p<0.01$), * ($p<0.05$)

Available Corpora and Why They Were Chosen

Corpus	Choice Rational
Brown	Balanced general English across multiple genres: best default comparison
Gutenberg	Classic literature: ideal for creative or literary writing
Reuters	News articles: useful for journalistic or formal writing
Inaugural	Speeches: Useful for professional or political comparisons

All corpora are pre-installed via NLTK (public domain/Apache 2.0 licensed).

Dependencies

- **scipy:** chi2_contingency for statistical testing
- **statsmodels:** effect size calculations
- **NLTK:** corpus access (brown, gutenberg, reuters, inaugural)

API

Call 1: Get Available Corpora

Request:
GET /api/corpora

Example Response:


```
{
  "corpora": [
    {
      "value": "brown",
      "label": "Brown Corpus",
      "description": "Balanced corpus of American English across multiple genres"
    },
    {
      "value": "gutenberg",
      "label": "Project Gutenberg",
      "description": "Classic literature from 19th and early 20th century"
    },
    {
      "value": "reuters",
      "label": "Reuters Corpus",
      "description": "Newswire articles from Reuters"
    },
    {
      "value": "inaugural",
      "label": "Inaugural Addresses Corpus",
      "description": "U.S. Presidential inaugural addresses"
    }
  ]
}
```


Call 2: Analyze Keyness Statistics

Request:
POST /api/keyness-stats
Content-Type: multipart/form-data

file: [uploaded file]
corpus: "brown"

Example Response:


```
{
  "total_words": 523,
  "unique_words": 287,
  "significant_keywords": 42,
  "corpus": {
    "name": "brown",
    "display_name": "Brown Corpus",
    "description": "Balanced corpus of American English",
    "total_words": 1161192
  },
  "keywords": [
    {
      "word": "cyberpunk",
      "effect_size": 2.34,
      "significance": "***"
    },
    {
      "word": "dystopian",
      "effect_size": 1.87,
      "significance": "**"
    },
    {
      "word": "neon",
      "effect_size": 1.56,
      "significance": "*"
    },
    {
      "word": "the",
      "effect_size": -0.89,
      "significance": "*"
    }
  ]
}
```


Key Files

- api/utils/keyness.py – Python processing script (reads stdin, outputs JSON)
- api/controllers/keynessController.js (stores corpus data and makes python call)
- frontend/keyness.js – UI visualization

7.2 SEMANTIC CLUSTERS

What it does

Semantic clustering identifies thematic word groups in a text using FastText word embeddings and K-Means clustering.
Instead of sentiment or frequency stats, this feature discovers patterns based on means/clusters of words that appear in similar semantic contexts.

This helps users see core ideas/themes in their writing examples like emotion, people, time, actions, etc.

Pipeline:

Endpoint: POST /api/semantic

1 Frontend uploads text file

2 Express receives file & streams text to Python via `stdin`

3 `semantic.py` tokenizes text (lowercase, alphabetic, ≥3 characters)

4 FastText loads pretrained embeddings (`cc.en.300.bin`) or trains fallback

5 Each word is converted to a vector embedding

6 K-Means dynamically determines number of clusters (~1 per 200 words)

7 Words grouped into semantic clusters based on vector proximity

8 Top 10 representative words chosen per cluster (closest to centroid)

9 Python returns JSON → frontend displays top 4 clusters

Calculations Performed

FastText Embeddings: Map each word → 300-dimensional meaning vector

Vector Averaging: Represent occurrence of semantic meaning

K-Means Clustering: Group words sharing conceptual similarity

Centroid Distance: Rank most representative terms of each cluster

Dynamic cluster sizing: 1 cluster per 200 tokens (min 2, max 15)

Output

- total_words

- number of clusters found

- top 4 largest clusters

- full cluster list (for export, hidden by default)

Why FastText model:

- Training foundation: The model used is trained on Common Crawl and Wikipedia, allow for broad semantic knowledge

- Lightweight: Not too tough on requirements, can run offline

- Licence: Open sources and allow for commercial purposes

Dependencies:

- FastText: for word embeddings
- NumPy: for vector math
- scikit-learn: for K-Means clustering
- re: for regex tokenization

API Calls and Responses

POST /api/semantic

Content-Type: multipart/form-data

file: [uploaded file]

Response

{

"overall_sentiment": "semantic_clusters",

"semantic_summary": {

"total_words": 31623,

"total_clusters": 15,

"top_clusters": [

{"label": "Cluster 9", "word_count": 1650, "words": ["something", "everything", "fortunately", "immediately", "sometimes", "completely", "possibly", "nevertheless", "remembering", "tremendously"]},

{"label": "Cluster 2", "word_count": 524, "words": ["scarecrow", "rhinoceros", "shepherdesses", "butterfly", "fountain", "cornfield", "princess", "riverbank", "riverside", "furniture"]},

{"label": "Cluster 8", "word_count": 237, "words": ["would", "might", "should", "just", "could", "better", "still", "surely", "there", "even"]}

],

"clusters": [{"label": "Cluster 9", "word_count": 1650, "words": ["something", "everything", "fortunately", "immediately", "sometimes", "completely", "possibly", "nevertheless", "remembering", "tremendously"]}, {"label": "Cluster 2", "word_count": 524, "words": ["scarecrow", "rhinoceros", "shepherdesses", "butterfly", "fountain", "cornfield", "princess", "riverbank", "riverside", "furniture"]}, {"label": "Cluster 8", "word_count": 237,

…

]}]

}

}

Key Files

`api/utils/semantic.py`: Python semantic engine

`frontend/semantic.js’: UI display & top-cluster table

Output Interpretation

top_clusters: Main themes of user's text

word_count: How many unique words belong to theme

words: Top representative terms (closest to cluster center)

8. CONFIGURATION

Port and address hardcoded in api NextJS/Express Server definition.
**Note: This is left as the future use of this project might be a local installation, and won't need a domain name/https set up etc.

FastText Model

Version and Source

Pre-trained Common Crawl model from https://fasttext.cc/docs/en/crawl-vectors.html.

Model Specifications

- File: "cc.en.300.bin" (300-dimension English model)
- Location: models/ directory at project root
- Size: ≈ 2.26 GB (100-dim) or ≈ 6.73 GB (300-dim)
- Language: English (Common Crawl + Wikipedia)

Configuration Options

- MODEL_PATH: "../models/cc.en.100.bin"
- NUM_CLUSTERS: 4
- PCA_COMPONENTS: 10

Reducing Model Size

Run the following to reduce dimensions and improve performance:

"**cd fastText**"
"**python reduce_model.py cc.en.300.bin cc.en.100.bin 100**"

NOTE
Update MODEL_PATH and dimension settings in "semantic.py" after reduction.

9. FUTURE CONCERNS

Concern #1 - Future Scalability Concerns

Late in development we noticed that the current design creates a new Python process for each user request which could overwhelm server resources if many people use it at once.
Running this on a remote web server would need expensive auto-scaling infrastructure and conflict with the privacy design constraints (local server), while distributing it as a desktop app would keep processing local and avoid both cost and privacy concerns.

Potential solutions:

- rejecting privacy constraint
- adding a queue system for users (would be quite limiting)
- design app to be installed in full locally (would come with reasonable RAM requirements - a new phone or pc would be required)

Concern #2 - Semantic Cluster Chart

Development for the chart visualization was not fully finished but it is designed to be ready for easy updates and customization.

10. Development & Extending Functionality

This section provides a guide for developers to extend the application's functionality.

Adding New Analysis Types

- Create Python Script - Add a new script in api/utils/ that reads from stdin and outputs JSON to stdout
- Create Backend Route - Add route in api/routes.js and create corresponding controller in api/controllers/
- Add Frontend UI - Update frontend/index.html with checkbox option and results container
- Add JavaScript Logic - Create or update frontend JS file to fetch data and display results
- Add Download Support - Update frontend/download.js to generate HTML for the new analysis

Key Requirements:

- Python scripts must read from stdin, output JSON to stdout
- Use pythonRunner.runPythonScript() for all Python execution
- Document new dependencies in requirements.txt or package.json

Adding File Format Support

Update Multer Filter - Configure file filter in routes to accept new type

Adding Additional Reference Corpora

- Download Corpus - Use nltk.download() for NLTK corpora or manually add files (see website documentation from external references). Opens a graphical interface to browse.
- Update Metadata - Add corpus definition to api/utils/corpora.py CORPORA dictionary
- Update Keyness Script - Ensure api/utils/keyness.py can load the new corpus
- Update Installation - Add corpus to api/utils/setup_nltk.py required_data list

Adding Dependencies

- Node.js: npm install <package> (in api directory)
- Python: pip install <package> (add to api/utils/requirements.txt)

10.1 Code Style & Organization

JavaScript:

- ES6+
- Linted with Airbnb JS Style Guide (Uninstalled ESLint dependency to reduce package size and installation overhead)

Python:

- PEP 8 guidelines (Used Ruff to Lint)

Code organization

- Modal logic is in modals.js, and the modals content is described in frontend/modals/
- Backend utils folder contains installation python scripts, as well as the processing and helper scripts.
- Python subprocess handling is in pythonRunner.js

**Note: KeynessController.js is mis-labelled. It does NOT do route handling. It is a wrapper for the python call, and stores the corpus information.

10.2 Utility Scripts

NPM script

- npm run start-api → Start server
- npm run open-browser → Open browser
- npm run full-setup → Install deps + NLTK
- npm run full-setup-with-fasttext → Full install + FastTex

Python Utility Scripts

- python api/utils/setup_nltk.py → Download NLTK data packages
- python api/utils/full-installation.py → Automated installer (NLTK + FastText model)

Note: The other scripts (analyze.py, keyness.py, semantic.py, corpora.py) are functional processing scripts that provide core application features, not utilities.

11. API Routes / Endpoints

Route	Method	Purpose	Input	Output
/api/analyze-file	POST	Word frequency	File (multipart/form-data)	{ "wordCount": 0, "uniqueWords": 0, "topWords": [{ "word": "", "count": 0 }] }
/api/sentiment	POST	Semantic clustering	File (multipart/form-data)	{ "total_words": 0, "total_clusters": 0, "clusters": [{ "cluster_id": 0, "words": [] }] }
/api/keyness-stats	POST	Keyness statistics	File String corpus	{ "keywords": [{ "word": "", "effect_size": 0.0, "significance": 0.0 }] }
/api/corpora	GET	Available corpora	None	{ "corpora": [{ "value": "", "label": "", "description": "" }] }

12. External Resources

- NLTK – https://www.nltk.org/
- FastText – https://fasttext.cc/docs/en/crawl-vectors.html
- Express – https://expressjs.com/
- Mammoth – https://github.com/mwilliamson/mammoth.js/
- Multer – https://github.com/expressjs/multer

13. Troubleshooting

File Upload Problems

- **Problem:** “File size exceeds 5MB limit”
  Solution: Reduce your file size or split the document into smaller sections.
- **Problem:** “File type not supported”
  Solution: Supported formats are TXT, DOCX, MD. Convert the file or use a different one.

Analysis Issues

- **Problem:** “Analyze My Writing” button is disabled
  Solution: Upload or paste text AND select at least one analysis option or the button will not enable.

Server Issues

- **Problem:** Page won't load at localhost:3000
  Solution: Ensure the server is running using “npm run start-api” or “node api/app.js”.

Installation Problems

- **Problem:** Can’t install the FastText models after cloning, but you don’t want to change your python version
  Solution: You can install visual build tools for free and then download the wheel version.

14. Security Model

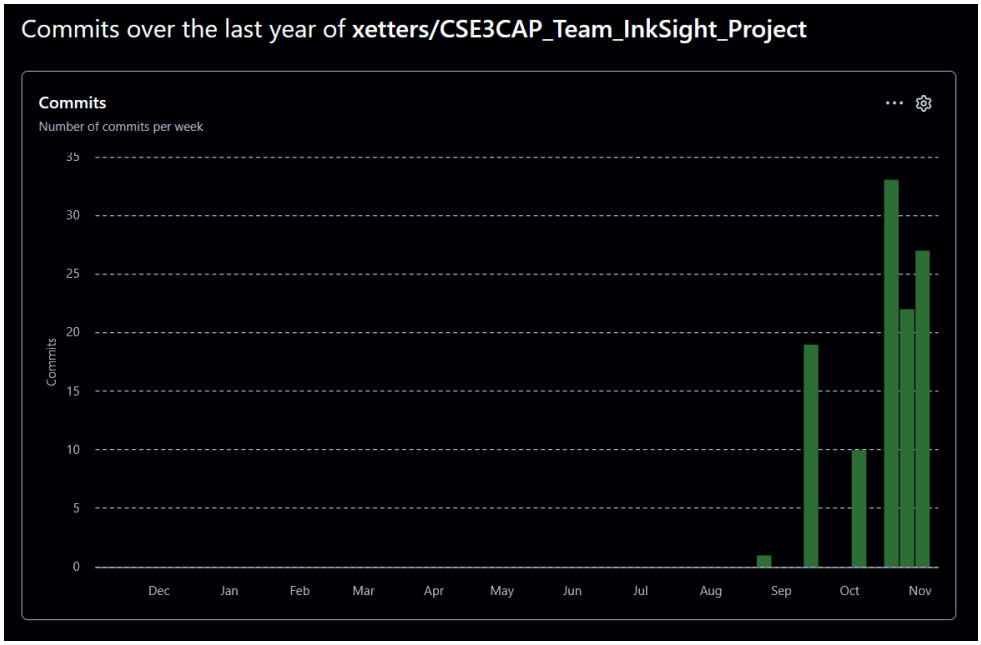
Security measures and threat mitigation strategies:

- Input validation on file size, type, and content
- In-memory buffer processing with automatic cleanup to prevent data leakage
  Subprocess isolation for Python script execution
- No external network requests to prevent data exfiltration
- Basic error handling across all python + js files

**Note: Security is an area we did not have enough development time to fully implement

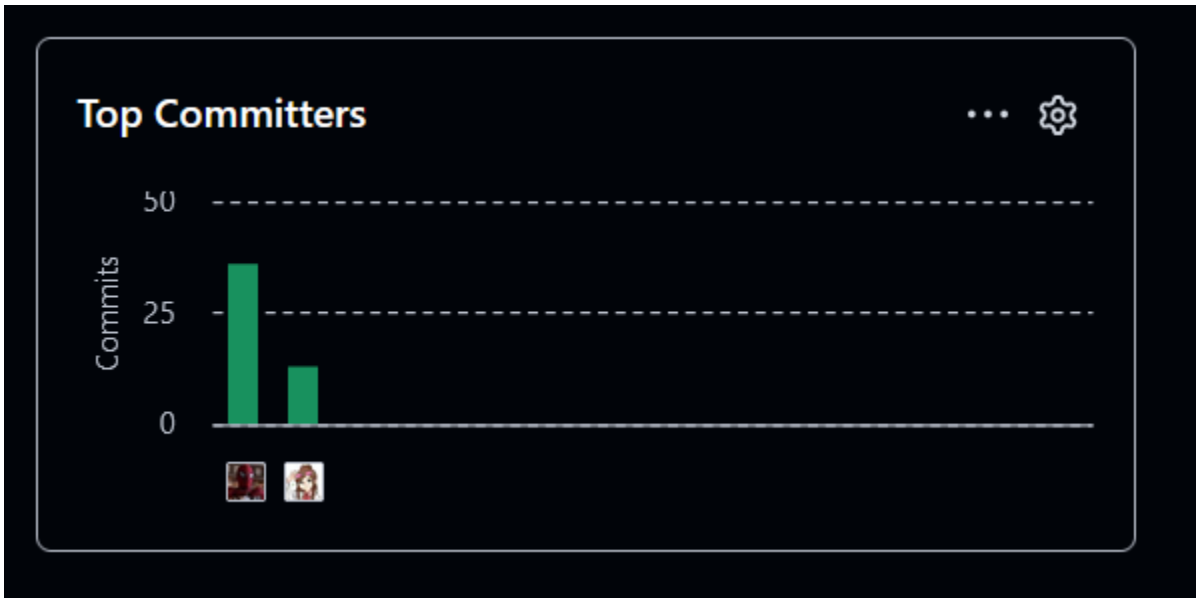
15. Git & Jira History

Git commits:



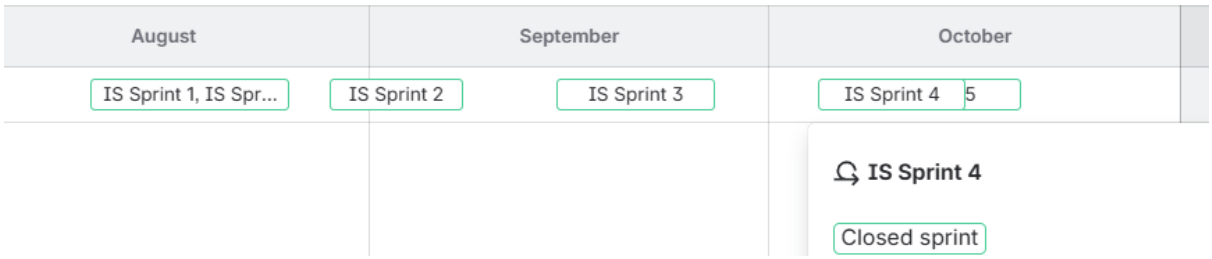
Commits went through extensive testing before merging with the master branch

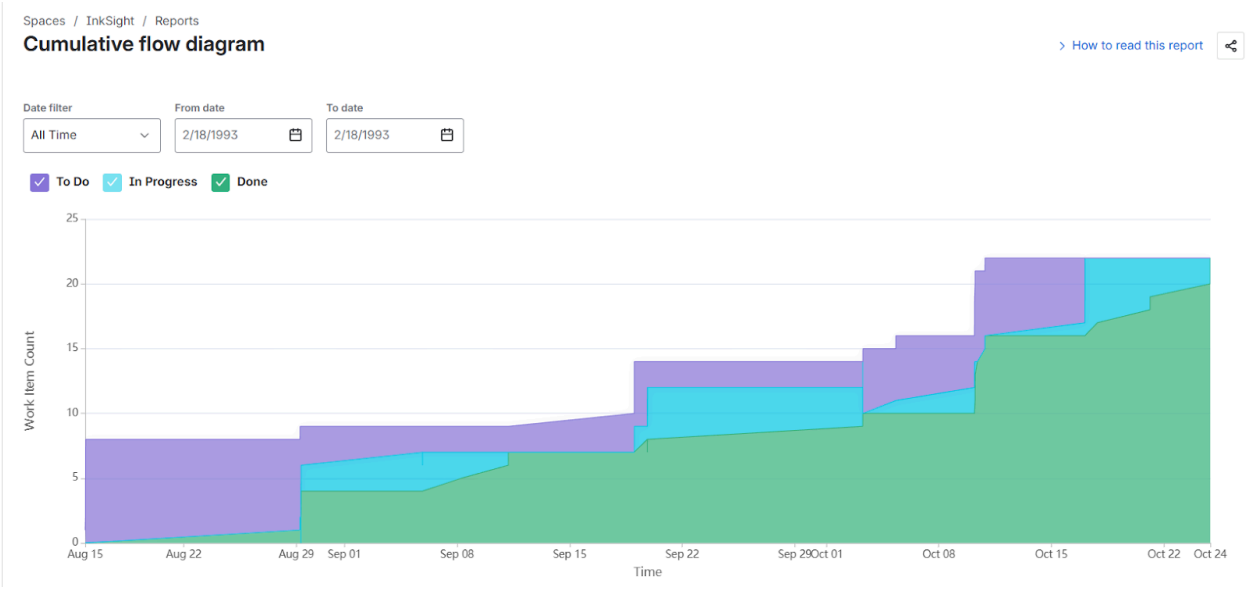
Contributors:

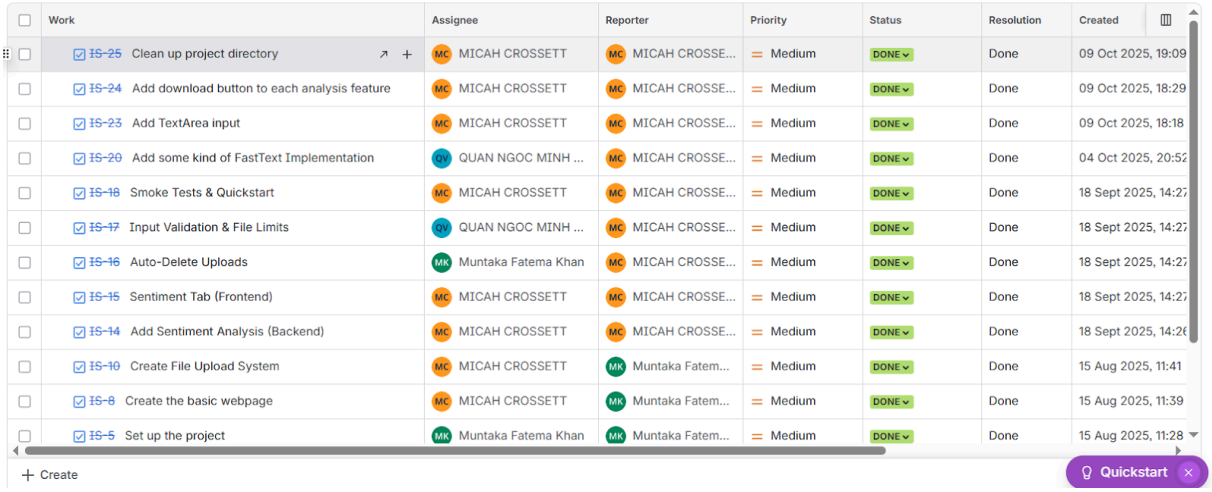


Jira

5 Sprint







16. Testing Information

User test cases

Test case	Description	Inputs	Expected output	Actual output	Status	Note
TC 01	Test valid file upload	User choose non allowed files	Cannot upload invalid files	Cannot upload invalid files	Pass	
TC 02	Test valid file limit	User choose file over limit	Error message	Error message	Pass	
TC 03	Check analysis option validation	User upload file or input text but not select any analysis option	Prompt to select an option	Prompt to select an option		Also apply if Keyness function is selected but no corpus is selected
TC 04	Check file and input text validation	User not input any text or upload any files	Analysis option block	Analysis option block	Pass	
TC 05	Check keyness and semantic function interaction with insufficient data	Single character input (a txt file or input filed a single letter	No output or error message	No output	Pass	
TC 06	Check Word Analysis function	Input file and select word analysis option	Output the word analysis result	Output the word analysis result	Pass	
TC 07	Check Keyness function (with random corpus)	Input file and select keyness analysis option with a corpus	Output the keyness analysis with the appropriate corpus result	Output the keyness analysis with the appropriate corpus result	Pass	

TC 08	Check Semantic analysis function	Input file and select semantic analysis option	Output the semantic analysis result	Output the semantic analysis result with cluster mapping	Pass	
TC 09	Select multiple analysis options	Select multiple analysis options after input	Output all selected option	Output all selected option	Pass	Slower respond time
TC 10	Use collapse option for upload window	Press the collapse button	Upload window minimalize	Upload window minimalize	Pass	This also allows user to view full cluster mapping
TC 11	Clear all output	Press the clear output button after analysis	Clear all output	Warning message and clear all output after confirmation	Pass	
TC 12	Use download function for a single analysis	Press download next to a specific analysis	Download the output data	Download a html containing the output	Pass	Semantic download option is not complete
TC 13	Press download all	Press the download all button on the top of the output window	Download the output data formatted in the order (word analysis > keyness > semantic)	Download a html that contain the output data formatted in the order (word analysis > keyness > semantic)	Pass	

Unit test cases:

In the api folder, there are 3 test cases used to test the backend logic and output. The test run on a text file in the test_text.txt file in the test_data folder in the same directory

The main purposes:
File Loading and Execution Success

Output Contract Validation

Test File	Function Tested	Purpose of Structural Check	Key Assertions
test_analyze.py	analyze_text	Verifies the output contains all basic literary metrics.	Checks for keys like **word_count**, **top**, **sentence_count**, **avg_sentence_length**, and **reading_time**.
test_semantic.py	analyze_semantic	Verifies the nested structure required for displaying clustering data.	Checks for top-level keys like **overall_sentiment** and the crucial nested dictionary **semantic_summary** containing **clusters**.
test_keyness.py	analyze_keyness	Verifies the output contains necessary statistical and corpus metadata for keyword display.	Checks for keys like **total_words**, **corpus** (which is a dictionary), and the **keywords** list, ensuring each item has word, effect_size, and significance.

Test results: all passed

```
PS C:\project\CSE3CAP_Team_InkSight_Project\api\tests> python test_semantic.py
.
----------------------------------------------------------------------
Ran 1 test in 1.480s

OK
```

```
PS C:\project\CSE3CAP_Team_InkSight_Project\api\tests> python test_keyness.py
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

```
PS C:\project\CSE3CAP_Team_InkSight_Project\api\tests> python test_analyze.py
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

End of document