IR Assignment 4: Structured Guided Browsing and Hypertext Model

Session: 2021 – 2025

Submitted by:

M. Jawad Haider 2021-CS-149

Supervised by:

Dr. Syed Khaldoon Khurshid

Department of Computer Science

University of Engineering and Technology Lahore

Pakistan

Contents

1 Overview

This project implements **Structured Guided Browsing** and **Hypertext Navigation** for a directory of documents using Flask (Python) and Bootstrap (HTML/CSS/JS). It enables hierarchical browsing of text documents with embedded hyperlinks, enhancing navigation and information retrieval.

2 Features

- **Directory Structure Traversal**: Dynamically reads text documents organized in nested folders.
- **Keyword-Based Hyperlinks**: Automatically generates hyperlinks to relevant documents based on shared keywords.
- **Dynamic UI**: Displays hierarchical documents with collapsible sections and intuitive navigation.
- **Bootstrap Integration**: Responsive UI for better usability.

3 Project Structure

The project files are organized as follows:

```
project-root/

templates/
   index.html
   document.html

app.py
scripts.py
requirements.txt

Famous Landmarks Around the World/
   Chapter 1 - Monuments/
      The Eiffel Tower.txt
      Statue of Liberty.txt
   Chapter 2 - Nature/
       Niagara Falls.txt
       Mount Everest.txt
```

4 DFD Diagram

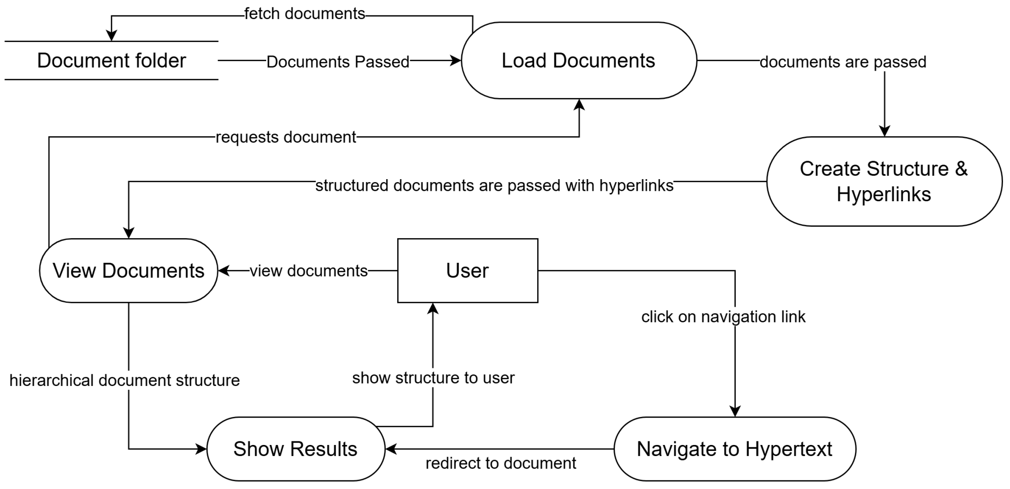

FIGURE 1: DFD - Level 0

5 Code Explanation

5.1 scripts.py

- **`readDirectoryStructure(rootDir)`**: Recursively reads the directory and loads all `.txt` files into a nested dictionary representing the file hierarchy.

```
def readDirectoryStructure(rootDir):
    """
    Reads the directory structure and returns a dictionary
    representing the hierarchy and content of the .txt
files.
    """
    hierarchy = {}

    # Traverse the directory structure
    for root, dirs, files in os.walk(rootDir):
        # Skip the root directory itself, focus on files
        if root == rootDir:
            continue

        # Build the path relative to the root directory
        pathParts = os.path.relpath(root, rootDir).split(os
.sep)
        # Find the parent node
        parent = hierarchy
```

```python
        for part in pathParts:
            parent = parent.setdefault(part, {})

        # Add content for each text file
        for file in files:
            if file.endswith('.txt'):
                # Read file content
                with open(os.path.join(root, file), 'r') as f:
                    content = f.read()

                # Store content with the filename as the key
                parent[file.replace('.txt', '')] = content

    return hierarchy

```

- **`addHyperlinksToContent(content, currentFilePath, fileStructure)`**: Searches for keywords in the document content and dynamically adds hyperlinks to related documents based on the index map.

```python
        def addHyperlinksToContent(content, currentFilePath, fileStructure):
    """
    Modify the content to add hyperlinks to other documents
    based on keywords found in the text.
    """
    # print(fileStructure['Chapter 1 - Monuments']['The Eiffel Tower'], 'fileStructure')

    # Extract words or terms that are used as document names in the structure
    def extractTitles(structure, prefix = ''):
        for name, substructure in structure.items():
            if isinstance(substructure, dict):
                splittedContent = filterImportantWords(name)
                for word in splittedContent:
                    addInIndexMap(word, '#' + prefix + name)

                extractTitles(substructure, prefix + name + '/')
            elif isinstance(substructure, str):
                splittedAndFilteredList = filterImportantWords(substructure)
                for word in splittedAndFilteredList:
                    addInIndexMap(word, prefix + name)

    extractTitles(fileStructure)

    # for key, value in indexMap.items():
    #     print(key, ' => ', value)

    # Create hyperlinks for the titles in the content
    for word in filterImportantWords(content):
        if word in indexMap:
            docs = indexMap[word][:]
            docs.remove(currentFilePath)
            if len(docs):
                hyperLink = f'/document/{docs[0]}' if docs[0].count('#') == 0 else f'/{docs[0]}'
                content = re.sub(r'\b' + re.escape(word) + r'\b', f'<a href="{hyperLink}">{word}</a>', content)

    return content


```

5.2 app.py

- **Routes**:
  - `/` - Home page displaying the document hierarchy.
  - `/document/<path:doc_path>` - Displays the content of a document with hyperlinks dynamically added.

```
        from flask import Flask, render_template, jsonify
        import os
        from scripts import readDirectoryStructure, addHyperlinksToContent
```

```python

app = Flask(__name__)

# Directory path where your text files are stored
rootDirectory = 'Famous Landmarks Around the World'

# Read the directory structure
fileStructure = readDirectoryStructure(rootDirectory)

@app.route('/')
def home():
    return render_template('index.html', structure=fileStructure)

@app.route('/document/<path:doc_path>')
def document(doc_path):
    # Traverse the structure based on the doc_path to find content
    docParts = doc_path.split('/')
    doc = fileStructure
    for part in docParts:
        doc = doc.get(part, {})

    # Add hyperlinks to the document content
    content = addHyperlinksToContent(doc, '/'.join(docParts), fileStructure)

    # return render_template('document.html', title=docParts[-1], content=content)
    return render_template('document.html', chapter=docParts[0], title=docParts[-1], content=content)

if __name__ == '__main__':
    app.run(debug=True)

```

- **`fileStructure`**: A nested dictionary generated by `readDirectoryStructure()`.

6 How It Works

1. **Directory Reading**: The program scans the directory `Famous Landmarks Around the World` and loads all `.txt` files into a dictionary.
2. **Keyword Indexing**: Filters meaningful words from document names and content. Keywords are mapped to document paths.
3. **Hyperlink Generation**: For each document, keywords are used to dynamically embed hyperlinks to related documents.
4. **Dynamic UI**: The home page displays the document structure, while document pages show content with embedded hyperlinks.

7 Example Workflow

1. **Home Page** (`/`): Displays a list of chapters and documents under collapsible headings.
2. **Document View** (`/document/<path>`): Displays the content of selected documents, with hyperlinks for navigation to related documents.

8 Key Components

File	**Description**
`app.py`	Main Flask app handling routing and templates.
`scripts.py`	Helper functions for reading, filtering, and hyperlinking.
`index.html`	Home page displaying document hierarchy.
`document.html`	Displays document content with hyperlinks.
`Famous Landmarks/`	Directory containing text documents.

9 Future Enhancements

- Add **search functionality** for keyword-based document retrieval.
- Include **highlighting** of keywords in document content.
- Implement **pagination** for lengthy documents.