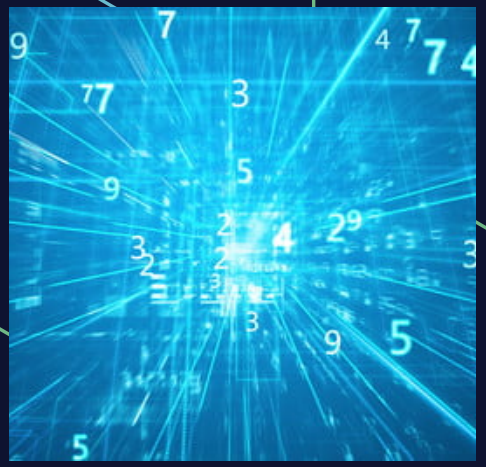
Intelligent Agent

File Segregation System Powered by Intelligent Agents

Challenges with digital data
File Classification - organisation and segregation
Automate using intelligent agents



Background

Intelligent Agents in AI
Architecture
Communication
Collaboration

Perceive
Reason
Extract knowledge
Inference
Derive conclusion
Autonomous decisions

Architecture: Reactive

Communication: protocols, languages and frameworks

Collaboration with users and other agents to achieve shared goals

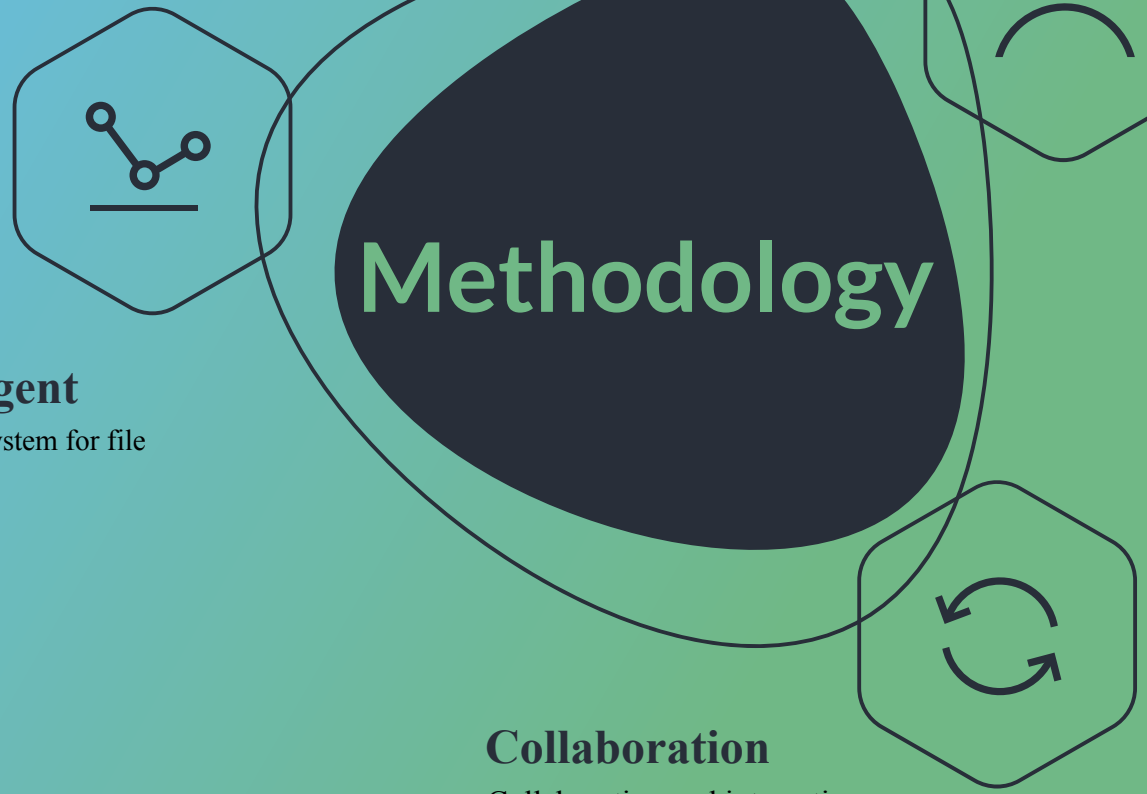
Objectives

To develop intelligent agents - to analyse file name extensions for classification - > 10 different file types - accuracy rate > 95% - timeline 10 days.

To create a user-friendly interface - users to review and validate the classification results - provide feedback on any misclassifications - timeframe 10 days.

To conduct testing and validation processes - dataset of 1,000 files representing a diverse range of file types - To achieve an accuracy rate of >90% - ensure classification process < 5 seconds per file - timeline 5 days

Methodology



Multi Agent

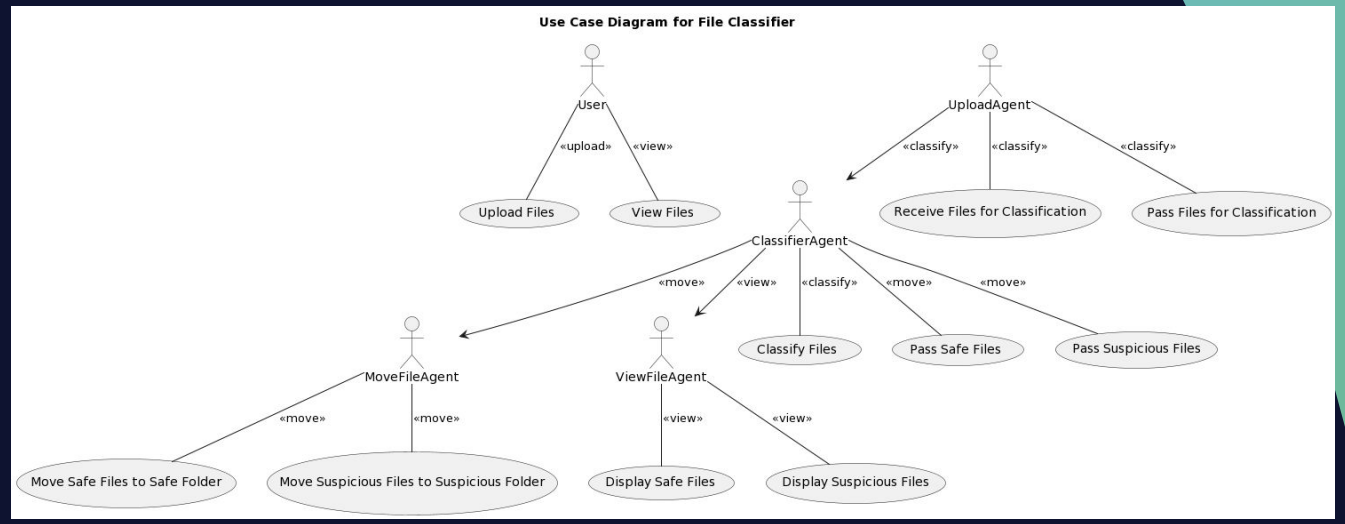
Multi-agent system for file classification

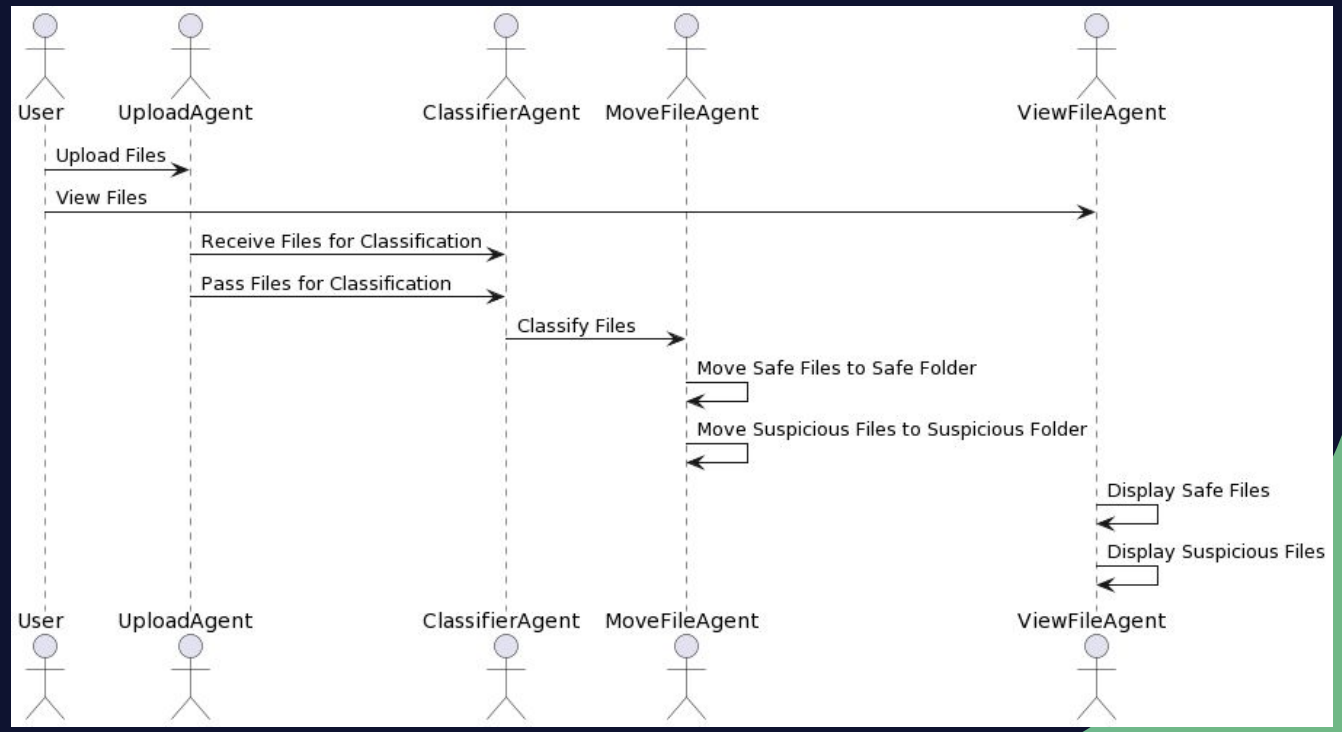
Reactive Agent

Use of reactive agents for fast and efficient response

Collaboration

Collaboration and interaction among agents





Project Design

Use Case UML

Use case diagrams serve to illustrate the broad functionalities and scope of a system.

Sequence UML

Sequence diagrams employed to highlight specific interaction patterns within the context of the use case.

Developments

Upload Agent

Secure file uploads

```
@app.route('/')
def upload_file():
    return render_template('upload.html')

@app.route('/upload', methods = ['GET', 'POST'])
def upload_files():
    if request.method == 'POST':
        files = request.files.getlist('files[]')
        for file in files:
            UploadAgent(file)
        return redirect(url_for('view_files'))
    else:
        return 'No file uploaded!'
```

```
def UploadAgent(file):
    filename = secure_filename(file.filename)
    if not os.path.exists('uploads'):
        os.makedirs('uploads')
    file.save(os.path.join('uploads', filename))
    ClassifierAgent(filename)
```

Classifier Agent

Classify files based on extensions

```
def ClassifierAgent(filename):
    # Checking if file extension is in the list of suspicious file types
    if any(filename.endswith(ft) for ft in suspicious_file_types):
        MoveFileAgent(filename, 'suspicious')
    else:
        MoveFileAgent(filename, 'safe')
```

Move File Agent

Move files to designated folders

```
def MoveFileAgent(filename, classification):
    if not os.path.exists(classification):
        os.makedirs(classification)
    shutil.move(os.path.join('uploads', filename), os.path.join(classification, filename))
```

View File Agent

Display files to the user

```
@app.route('/view', methods = ['GET'])
def view_files():
    safe_files = os.listdir('safe') if os.path.exists('safe') else []
    suspicious_files = os.listdir('suspicious') if os.path.exists('suspicious') else []
    return render_template('view.html', safe_files=safe_files, suspicious_files=suspicious_files)
```

Testing and Evaluation

- **Objective:** Ensuring functionality, identifying issues, evaluating performance, user satisfaction.
- **Test Types:**
  - **Unit:** Checks functions.
  - **Integration:** Reveals interaction faults.
  - **System:** Confirms compatibility.
  - **Performance:** Analyzes behavior under load.
  - **UAT:** User satisfaction confirmation.
- **Tools:**
  - **Python's unittest for test cases.**
  - **pytest-cov and coverage to measure test coverage**

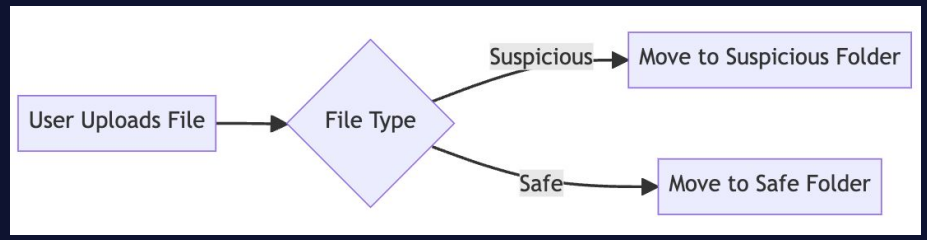


Figure 7.2 System Process

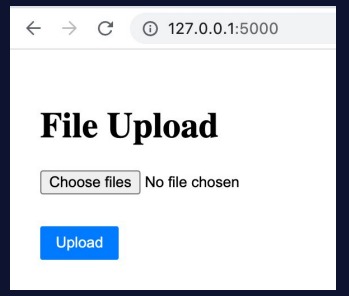


Figure 7.3 File Upload UI

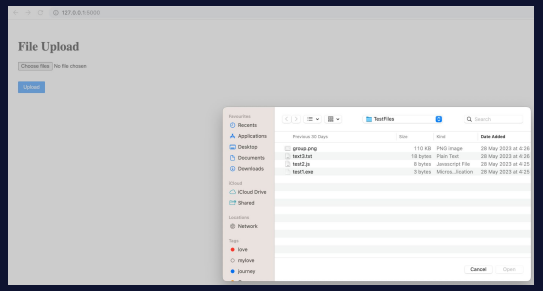


Figure 7.4 File Browser UI

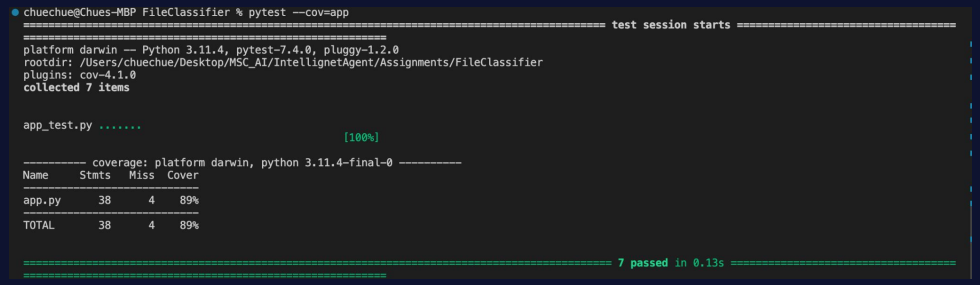


Figure 7.1 Unit Test Result

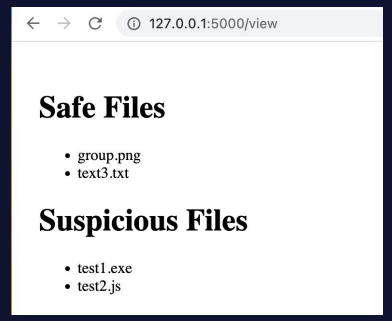
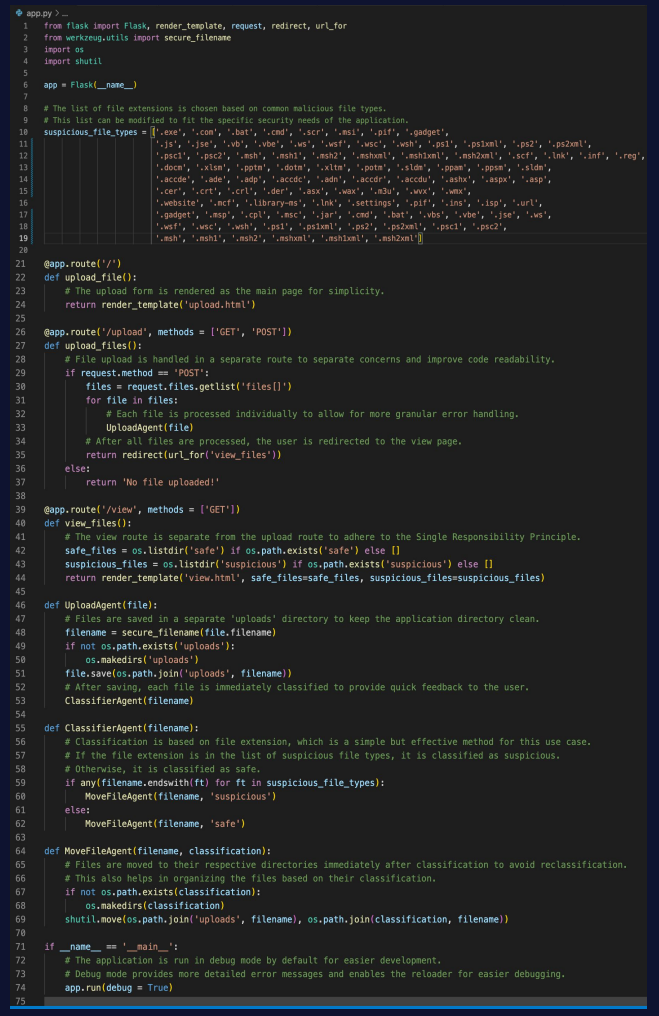


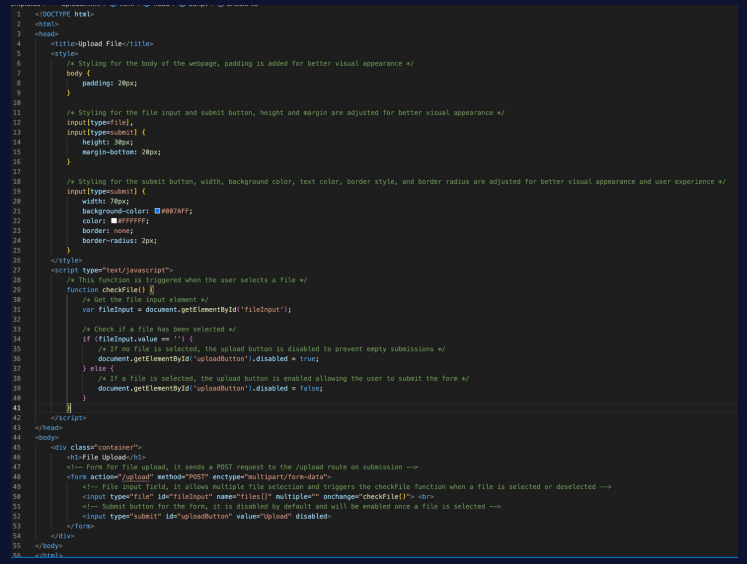
Figure 7.5 Result of Classified Files UI

Programming Approach



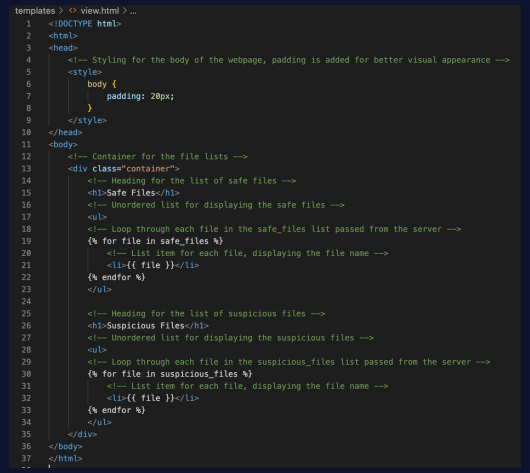
```python
from flask import Flask, render_template, request, redirect, url_for
from werkzeug.utils import secure_filename
import os
import shutil

app = Flask(__name__)

# The list of file extensions is chosen based on common malicious file types.
# This list can be modified to fit the specific security needs of the application.
suspicious_file_types = ['.exe', '.com', '.bat', '.cmd', '.scr', '.msi', '.dif', '.gadget',
                         '.js', '.jse', '.vb', '.vbe', '.ws', '.wsf', '.wsc', '.wsh', '.ps1', '.ps1xml', '.ps2', '.ps2xml',
                         '.psc1', '.psc2', '.msh', '.msh1', '.msh2', '.mshxml', '.msh1xml', '.msh2xml', '.scf', '.lnk', '.inf', '.reg',
                         '.docm', '.xlsm', '.pptm', '.dotm', '.xltm', '.potm', '.sldm', '.ppam', '.ppsm', '.sldm',
                         '.accde', '.ade', '.adp', '.accdc', '.adn', '.accdr', '.accdu', '.ashx', '.aspx', '.asp',
                         '.cer', '.crt', '.crl', '.der', '.asx', '.wax', '.m3u', '.wvx', '.wmx',
                         '.website', '.mcf', '.library-ms', '.lnk', '.settings', '.pif', '.ins', '.isp', '.url',
                         '.gadget', '.msp', '.cpl', '.msc', '.jar', '.cmd', '.bat', '.vbs', '.vbe', '.jse', '.ws',
                         '.wsf', '.wsc', '.wsh', '.ps1', '.ps1xml', '.ps2', '.ps2xml', '.psc1', '.psc2',
                         '.msh', '.msh1', '.msh2', '.mshxml', '.msh1xml', '.msh2xml']

@app.route('/')
def upload_file():
    # The upload form is rendered as the main page for simplicity.
    return render_template('upload.html')

@app.route('/upload', methods = ['GET', 'POST'])
def upload_files():
    # File upload is handled in a separate route to separate concerns and improve code readability.
    if request.method == 'POST':
        files = request.files.getlist('files[]')
        for file in files:
            # Each file is processed individually to allow for more granular error handling.
            UploadAgent(file)
        # After all files are processed, the user is redirected to the view page.
        return redirect(url_for('view_files'))
    else:
        return 'No file uploaded!'

@app.route('/view', methods = ['GET'])
def view_files():
    # The view route is separate from the upload route to adhere to the Single Responsibility Principle.
    safe_files = os.listdir('safe') if os.path.exists('safe') else []
    suspicious_files = os.listdir('suspicious') if os.path.exists('suspicious') else []
    return render_template('view.html', safe_files=safe_files, suspicious_files=suspicious_files)

def UploadAgent(file):
    # Files are saved in a separate 'uploads' directory to keep the application directory clean.
    filename = secure_filename(file.filename)
    if not os.path.exists('uploads'):
        os.makedirs('uploads')
    file.save(os.path.join('uploads', filename))
    # After saving, each file is immediately classified to provide quick feedback to the user.
    ClassifierAgent(filename)

def ClassifierAgent(filename):
    # Classification is based on file extension, which is a simple but effective method for this use case.
    # If the file extension is in the list of suspicious file types, it is classified as suspicious.
    # Otherwise, it is classified as safe.
    if any(filename.endswith(ft) for ft in suspicious_file_types):
        MoveFileAgent(filename, 'suspicious')
    else:
        MoveFileAgent(filename, 'safe')

def MoveFileAgent(filename, classification):
    # Files are moved to their respective directories immediately after classification to avoid reclassification.
    # This also helps in organizing the files based on their classification.
    if not os.path.exists(classification):
        os.makedirs(classification)
    shutil.move(os.path.join('uploads', filename), os.path.join(classification, filename))

if __name__ == '__main__':
    # The application is run in debug mode by default for easier development.
    # Debug mode provides more detailed error messages and enables the reloader for easier debugging.
    app.run(debug = True)
```

app.py



```html
<!DOCTYPE html>
<html>
<head>
    <title>Upload File</title>
    <style>
        /* Styling for the body of the webpage, padding is added for better visual appearance */
        body {
            padding: 20px;
        }

        /* Styling for the file input and submit button, height and margin are adjusted for better visual appearance */
        input[type=file],
        input[type=submit] {
            height: 30px;
            margin-bottom: 20px;
        }

        /* Styling for the submit button, width, background color, text color, border style, and border radius are adjusted for better visual appearance and user experience */
        input[type=submit] {
            width: 70px;
            background-color: #007AFF;
            color: #FFFFFF;
            border: none;
            border-radius: 2px;
        }
    </style>
    <script type="text/javascript">
        /* This function is triggered when the user selects a file */
        function checkFile() {
            /* Get the file input element */
            var fileInput = document.getElementById('fileInput');

            /* Check if a file has been selected */
            if (fileInput.value == '') {
                /* If no file is selected, the upload button is disabled to prevent empty submissions */
                document.getElementById('uploadButton').disabled = true;
            } else {
                /* If a file is selected, the upload button is enabled allowing the user to submit the form */
                document.getElementById('uploadButton').disabled = false;
            }
        }
    </script>
</head>
<body>
    <div class="container">
        <h1>File Upload</h1>
        <!-- Form for file upload, it sends a POST request to the /upload route on submission -->
        <form action="/upload" method="POST" enctype="multipart/form-data">
            <!-- File input field, it allows multiple file selection and triggers the checkFile function when a file is selected or deselected -->
            <input type="file" id="fileInput" name="files[]" multiple="" onchange="checkFile()"> <br>
            <!-- Submit button for the form, it is disabled by default and will be enabled once a file is selected -->
            <input type="submit" id="uploadButton" value="Upload" disabled>
        </form>
    </div>
</body>
</html>
```

upload.html



```html
<!DOCTYPE html>
<html>
<head>
    <!-- Styling for the body of the webpage, padding is added for better visual appearance -->
    <style>
        body {
            padding: 20px;
        }
    </style>
</head>
<body>
    <!-- Container for the file lists -->
    <div class="container">
        <!-- Heading for the list of safe files -->
        <h1>Safe Files</h1>
        <!-- Unordered list for displaying the safe files -->
        <ul>
        <!-- Loop through each file in the safe_files list passed from the server -->
        {% for file in safe_files %}
            <!-- List item for each file, displaying the file name -->
            <li>{{ file }}</li>
        {% endfor %}
        </ul>

        <!-- Heading for the list of suspicious files -->
        <h1>Suspicious Files</h1>
        <!-- Unordered list for displaying the suspicious files -->
        <ul>
        <!-- Loop through each file in the suspicious_files list passed from the server -->
        {% for file in suspicious_files %}
            <!-- List item for each file, displaying the file name -->
            <li>{{ file }}</li>
        {% endfor %}
        </ul>
    </div>
</body>
</html>
```

view.html

app.py

- Programming approach: Python programming
- Flask: Lightweight framework for building web interfaces
- Werkzeug: Ensures security in file handling and communication
- os: Handles operating system-dependent functionalities
- shutil: Performs high-level file operations and management

upload.html

- HTML template for the file upload page.
- Allows users to select and upload files.
- Includes JavaScript code for handling file selection and enabling/disabling the upload button.

view.html

- HTML template for the file viewing page.
- Displays the list of files categorized as "safe" and "suspicious".
- Utilizes Flask template syntax to dynamically populate the file list.

Challenges
and Solution

Guaranteeing accurate file classification and securing file uploads are paramount challenges in our development process

Accuracy
Accurate file classification

Security
Secure file upload to prevent attacks

Validation
Testing and validation

CONCLUSION

Developed multi-agent system for file classification and segregation - Created user interface - Testing and validation

Limitations – filename extension basis only

Further study – classification based on content and user defined attributes – use machine learning and natural language processing to enhance performance

Thanks