Despike: Thermal Expansion Data Analysis Tool

Technical Documentation

Table of Contents

Project Overview

Despike is a specialized web application designed for analyzing thermal expansion coefficient (CTE) data. It enables researchers and engineers to upload Excel files containing CTE measurements at different heating rates, visualize the data with interactive Plotly graphs, and download processed results.

Main Features

- **File Upload**: Process Excel files containing thermal expansion data at multiple heating rates (1K/min, 3K/min, 6K/min, and 10K/min)
- **Data Visualization**: View original (raw) and smoothed data in interactive side-by-side graphs for easy comparison
- **Data Table**: Explore uploaded data in a paginated table with navigation controls
- **Data Processing**: Apply customizable smoothing algorithms to raw data to reduce noise and outliers
- **Export Capability**: Download processed data as Excel files with adjustable smoothing parameters

Target Users

- Materials scientists and thermal engineers analyzing CTE data
- Researchers working with thermal expansion measurements
- Laboratory technicians processing thermal data

Architecture

The application uses a combination of Flask (backend) and Dash (for interactive components) with a responsive HTML/CSS frontend. Below is the high-level architecture:

```
┌─────────────────┐     ┌──────────────────┐     ┌─────────────────┐
│                 │     │                  │     │                 │
│  Client Browser │◄────┤  Flask (Server)  │◄────┤  Temporary      │
│                 │     │                  │     │  File Storage   │
└────────┬────────┘     └────────┬─────────┘     └─────────────────┘
         │                       │                        ▲
         │                       │                        │
         │                       │                        │
         │                       │                        │
         │              ┌────────▼─────────┐              │
         └─────────────►│                  ├──────────────┘
                        │   Dash App       │
                        │  (Data Table)    │
                        │                  │
                        └──────────────────┘
```

Technology Stack

1. **Backend**: Flask (Python) for routing, file handling, and template rendering
2. **Data Processing**: Pandas for data manipulation and analysis
3. **Visualization**: Plotly for interactive graphs
4. **Interactive Components**: Dash for the data table with pagination
5. **File Handling**: Temporary file storage with UUID-based identification
6. **Frontend**: HTML/CSS with responsive design

Key Design Patterns

- **Model-View-Controller (MVC)** pattern with clear separation of concerns
- **Stateless architecture** with minimal session data (only file references)
- **Component-based design** with Dash embedded in Flask

File Structure

```
app/
├── app.py                  # Main application file with all logic
├── templates/              # HTML templates
│   ├── index.html          # Home page
│   ├── about.html          # About page
│   ├── app.html            # Main application page with visualization
│   └── contact.html        # Contact page
├── static/                 # Static assets (stylesheets, JavaScript, images)
└── temp/                   # Temporary file storage (created at runtime)
```

Key Files

- **app.py**: Core application file containing all Python logic including:

    - Flask application setup and configuration

- Dash integration for interactive data table
- Data processing functions
- Route handlers
- File management

- **templates/app.html**: Main template for the dashboard containing:
    - File upload form
    - Dash iframe for data table
    - Graph containers
    - Download section

Function Relationships

The diagram below shows the relationship between key functions in the application:

```
┌────────────────────┐    ┌─────────────────────┐    ┌──────────────────┐
│ Flask Routes       │    │ Data Processing     │    │ Dash Components  │
├────────────────────┤    ├─────────────────────┤    ├──────────────────┤
│ index()            │    │ process_excel_data()│    │ dash_app.layout  │
│ about()            │    │ smooth_data()       │    │ update_table()   │
│ contact()          │    │ allowed_file()      │    │ sync_file_id()   │
│ dashboard()        │───►│                     │◄───│                  │
│ download_file()    │    └─────────────────────┘    └──────────────────┘
└────────────────────┘              ▲                          ▲
                                    │                          │
                                    │                          │
                                    │                          │
                      ┌─────────────┴────────┐                 │
                      │ File Management      │                 │
                      ├──────────────────────┤                 │
                      │ Temporary File Storage│────────────────┘
                      │ Session Management   │
                      └──────────────────────┘
```

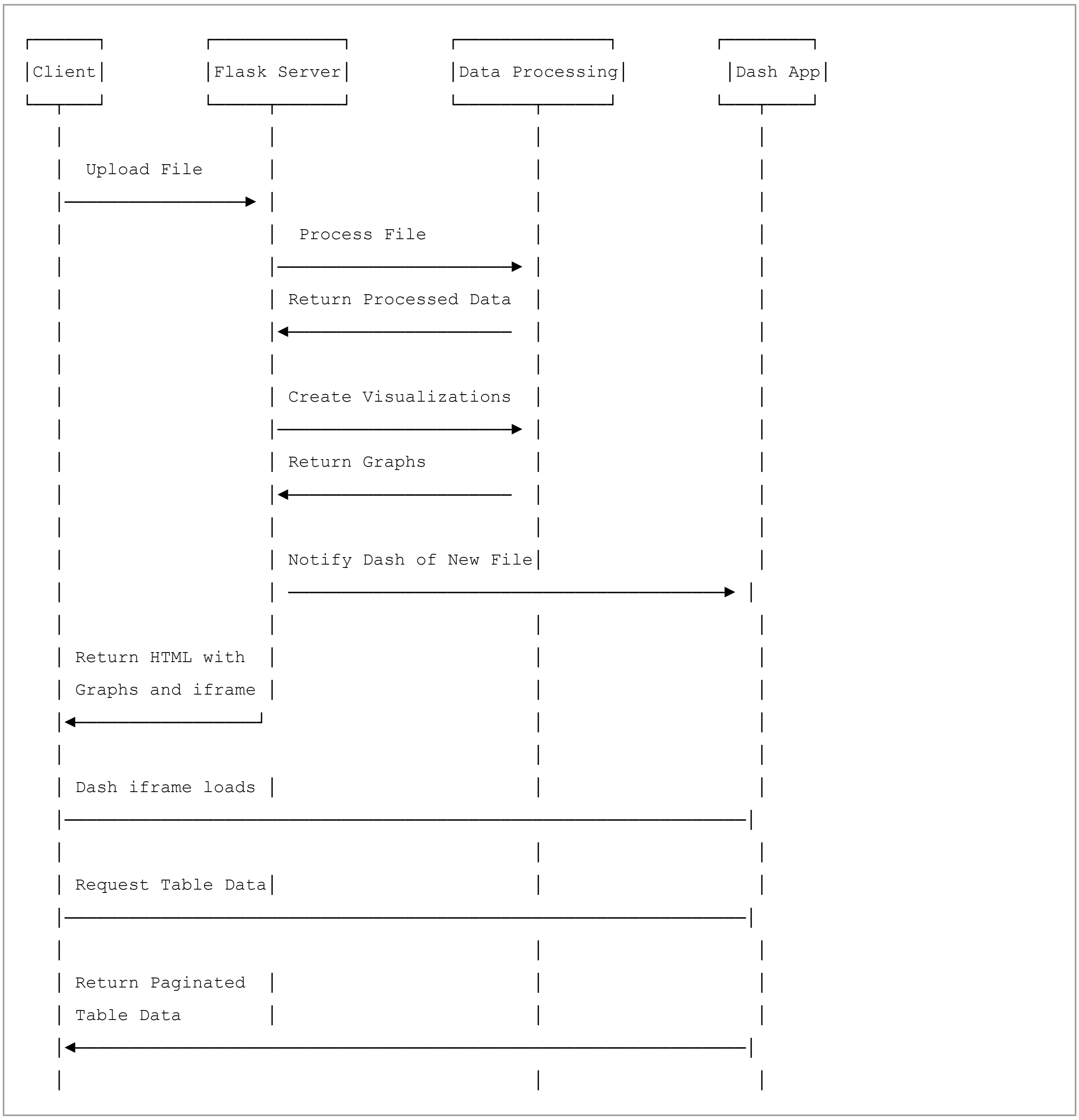
Core Function Dependencies

1. **`dashboard()` route**
    - Handles file upload and visualization
    - Depends on `allowed_file()`, `process_excel_data()` and `smooth_data()`
    - Generates visualizations using Plotly
    - Stores file ID in session for Dash to access
2. **`download_file()` route**
    - Processes uploaded file with custom parameters
    - Depends on `process_excel_data()` and `smooth_data()`
    - Generates downloadable Excel file

3. **`update_table()` callback**
    - Dash callback for paginated data table
    - Accesses file using ID from session
    - Processes data for tabular display
4. **`process_excel_data()` function**
    - Reads and processes Excel data
    - Handles column renaming and data cleaning
    - Central data processing function used by multiple components
5. **`smooth_data()` function**
    - Applies configurable smoothing algorithm
    - Used for both visualization and data export

Data Flow

The sequence diagram below illustrates the data flow when a user uploads and processes a file:



Key Data Flow Steps

1. **File Upload Process**
    - User uploads Excel file via form submission
    - File is validated and saved to temporary storage with UUID
    - File ID is stored in session for later reference
2. **Data Processing Pipeline**
    - Excel data is read and processed
    - Columns are renamed to standard format
    - Data is cleaned and converted to appropriate types
    - Smoothing algorithm is applied to data for visualization
3. **Visualization Generation**

- Individual graphs are created for each heating rate
- Both original and smoothed data are displayed
- Graphs are converted to HTML for embedding in page

4. **Data Table Integration**

    - Dash component loads in iframe
    - File ID is retrieved from session
    - Data is loaded and paginated for table display
    - User can navigate through data pages

5. **Download Workflow**

    - User sets custom smoothing window size
    - File is retrieved from temporary storage
    - Custom smoothing is applied
    - Excel file is generated and sent to browser

Key Components

1. Flask Routes

Route	Function	Description
`/`	`index()`	Renders the home page
`/about`	`about()`	Renders the about page
`/contact`	`contact()`	Renders the contact page
`/app`	`dashboard()`	Main application route; Handles file upload, processing, and visualization
`/download`	`download_file()`	Processes and returns the smoothed Excel file

2. Data Processing Functions

Function	Description
`process_excel_data(file_path)`	Reads and processes the Excel file data. Handles column renaming, data type conversion, and cleaning operations.
`smooth_data(y_values, window_size=10)`	Applies a rolling window smoothing algorithm with configurable window size. Reduces noise in measurements.
`allowed_file(filename)`	Validates if the file has the correct extension (.xlsx). Security check for file uploads.

3. Dash Components

Component	Description
`dash_app.layout`	Defines the layout for the data table and pagination controls. Contains visibility logic.
`update_table()`	Dash callback that updates the table based on file uploads and pagination. Handles data retrieval and page navigation.
`sync_file_id()`	Synchronizes file IDs between Flask session and Dash. Maintains state consistency.

4. File Management

Feature	Description
Temporary Storage	Files are saved to the system's temporary directory with a UUID identifier. Avoids permanent storage of uploaded files.

Feature	Description
Session Management	Only file IDs are stored in the session, not the entire file content. Minimizes session size.

Installation Guide

Prerequisites

- Python 3.7 or higher
- pip package manager

Required Packages

- Flask: Web framework
- pandas: Data processing
- plotly: Visualization
- dash: Interactive components
- openpyxl: Excel file handling

Step-by-Step Installation

1. **Clone the repository or download the source code**

   ```
   git clone https://github.com/your-username/despike.git
   cd despike
   ```

2. **Install dependencies**

   ```
   pip install flask pandas plotly dash openpyxl
   ```

3. **Run the application**

   ```
   python app.py
   ```

4. **Access the application**

   Open your browser and navigate to:

   ```
   http://127.0.0.1:5000/
   ```

Configuration Options

- **Secret Key**: Set a secure secret key in app.py

  ```
  app.secret_key = 'your_secure_key_here'
  ```

- **Debug Mode**: Enable/disable debug mode for development

```
app.run(debug=True)  # For development
app.run(debug=False) # For production
```

Customization Guide

Changing the Smoothing Algorithm

To modify the smoothing algorithm, edit the `smooth_data()` function in app.py:

```
def smooth_data(y_values, window_size=10):
    # Current implementation: Simple rolling average
    return pd.Series(y_values).rolling(window=window_size, min_periods=1).mean()

    # Alternative: Exponential weighted moving average
    # return pd.Series(y_values).ewm(span=window_size).mean()

    # Alternative: Savitzky-Golay filter (requires scipy)
    # from scipy.signal import savgol_filter
    # return savgol_filter(y_values, window_size, 3)
```

Adding New Visualization Features

To add a new type of visualization:

1. Add a new entry to the `rates` list in the dashboard route

```
rates = [
    {"title": "1K/min", "T": "T_1K", "CTE": "CTE_1K"},
    {"title": "3K/min", "T": "T_3K", "CTE": "CTE_3K"},
    {"title": "6K/min", "T": "T_6K", "CTE": "CTE_6K"},
    {"title": "10K/min", "T": "T_10K", "CTE": "CTE_10K"},
    # Add new rate here if needed
]
```

2. Create a new visualization type (e.g., a combined view)

```
# After the individual rate graphs
# Create a combined graph showing all rates together
combined_fig = go.Figure()
for rate in rates:
    combined_fig.add_trace(go.Scatter(
        x=df[rate["T"]],
        y=smooth_data(df[rate["CTE"]]),
        mode='lines',
        name=f"{rate['title']} (Smoothed)",
    ))

combined_fig.update_layout(
    title_text="All Rates Comparison",
    template="plotly_white",
    height=500,
    width=900
)

# Add to the graphs list
graphs_html.append(combined_fig.to_html(full_html=False))
```

Modifying the Page Size in the Data Table

To change the number of rows displayed per page:

1. Find the `page_size` variable in the `update_table()` function

```
# Current setting
page_size = 20

# Change to desired number
page_size = 50  # Shows 50 rows per page
```

Changing the Color Scheme

To modify the graph colors:

1. In the dashboard route, update the colors in the `fig.add_trace()` calls

```
fig.add_trace(go.Scatter(
    x=df[rate["T"]],
    y=df[rate["CTE"]],
    mode='lines',
    name=f"{rate['title']} (Original)",
    line=dict(color='#FF5252', width=1)  # Change color here
))

fig.add_trace(go.Scatter(
    x=df[rate["T"]],
    y=smooth_data(df[rate["CTE"]]),
    mode='lines',
    name=f"{rate['title']} (Smoothed)",
    line=dict(color='#4285F4', width=2)  # Change color here
))
```

2. Update the `template` parameter in `fig.update_layout()` to one of:

    - "plotly_white": Clean white background
    - "plotly": Default light gray grid
    - "ggplot2": Based on R's ggplot2
    - "seaborn": Based on Python's seaborn
    - "simple_white": Minimal white background

Adding New Excel File Columns

If your Excel files have a different structure, edit the column remapping in the `process_excel_data()` function:

```
# Current implementation:
df.columns = ["T_1K", "CTE_1K", "T_3K", "CTE_3K", "T_6K", "CTE_6K", "T_10K", "CTE_10K"]

# Modify to match your Excel structure, for example:
df.columns = ["Temp_1K", "CTE_1K", "Temp_3K", "CTE_3K", "Temp_6K", "CTE_6K", "Temp_10K", "CTE_10K"]
```