I'll help you create a detailed documentation of the GST Reconciliation Application. Let me first explore the codebase to gather all the necessary information.

Ran tool

Let me examine the main application file and other relevant files to understand the functionality in detail.

Read file: app.py

Let me check the reconciliation utility file to understand the reconciliation logic.

Read file: utils/reconciliation.py

Based on my analysis of the codebase, I'll create a comprehensive documentation of the GST Reconciliation Application. Here's the detailed documentation:

# GST Reconciliation Application Documentation

## 1. Overview

The GST Reconciliation Application is a sophisticated tool designed to reconcile GST data between a company's books and GSTR-2A records. It provides a user-friendly interface for data upload, processing, and analysis of GST reconciliation results.

## 2. Application Features

### 2.1 User Interface

- Modern web-based interface built with Streamlit

- Wide layout with expandable sidebar

- Custom styling for better user experience

- Progress tracking with visual indicators

- Interactive data tables with editing capabilities

### 2.2 Data Input

- Supports Excel file upload (.xlsx, .xls)

- Required data columns:

- Source Name (Books/GSTR-2A)

- Supplier GSTIN

- Supplier Legal Name

- Supplier Trade Name

- Invoice Date

- Books Date

- Invoice Number

- Total Taxable Value

- Total Tax Value

- Total IGST Amount

- Total CGST Amount

- Total SGST Amount

- Total Invoice Value

### 2.3 Data Filtering

- Source-based filtering (Books/GSTR-2A)

- Date range filtering

- Interactive data editing capabilities

- Real-time data validation

## 3. Reconciliation Process

### 3.1 Data Cleaning and Standardization

- GSTIN format standardization

- Invoice number standardization

- Date format conversion

- Numeric value standardization

- Handling of missing values

### 3.2 Matching Process

The application performs reconciliation in multiple steps:

#### 3.2.1 Exact Matches

- Matches based on:

- Supplier GSTIN

- Invoice Number

- Tax amounts (within ₹1 tolerance)

- Invoice dates (within 1 day tolerance)

#### 3.2.2 Tax-based Matches

- Secondary matching based on:

- Total tax amounts

- Taxable values

- Date proximity

- Supplier information

#### 3.2.3 Enhanced Matching

- Fuzzy matching for invoice numbers

- Supplier name matching

- Amount-based matching

- Date-based matching

### 3.3 Tolerance Levels

- Tax amount tolerance: ₹1

- Date tolerance: 1 day

- Amount matching tolerance: Configurable

## 4. Reconciliation Results

### 4.1 Match Categories

1. Exact Matches

- Perfect matches with all criteria

- Minimal differences in dates and amounts

2. Partial Matches

- Matches with some discrepancies

- Tax amount differences

- Date differences

- Supplier information differences

3. Mismatches

- Records present in one source but not in the other

- Significant differences in amounts or dates

- Invalid or missing data

### 4.2 Reporting

- Detailed reconciliation report

- Summary statistics

- Tax-wise breakdown

- Supplier-wise analysis

- Date-wise analysis

### 4.3 Integrity Checks

- Data completeness verification

- Amount validation

- Date validation

- GSTIN validation

- Duplicate check

## 5. Output and Analysis

### 5.1 Reports Generated

1. Raw Summary

- Total records by source

- Amount totals

- Tax-wise totals

2. Reconciliation Summary

- Match statistics

- Mismatch details

- Partial match analysis

3. Tax Summary

- IGST analysis

- CGST analysis

- SGST analysis

- Total tax reconciliation

### 5.2 Visualizations

- Progress tracking

- Match statistics

- Tax distribution

- Date-wise trends

## 6. Technical Specifications

### 6.1 Dependencies

- Python 3.x

- Streamlit

- Pandas

- NumPy

- FuzzyWuzzy

- Plotly

- Openpyxl

### 6.2 Performance Features

- Progress tracking

- Time elapsed monitoring

- Memory optimization

- Batch processing

- Error handling

## 7. Error Handling and Validation

### 7.1 Data Validation

- Required column verification

- Data type validation

- Format validation

- Value range checks

### 7.2 Error Types Handled

- Missing data

- Invalid formats

- Data type mismatches

- Duplicate records

- Inconsistent values

## 8. Best Practices

### 8.1 Data Preparation

- Ensure all required columns are present

- Validate data formats before upload

- Check for duplicate records

- Verify GSTIN formats

- Validate date formats

### 8.2 Reconciliation Process

- Start with exact matches

- Review partial matches carefully

- Investigate mismatches thoroughly

- Document reconciliation decisions

- Maintain audit trail

### 8.3 Result Analysis

- Review summary statistics

- Analyze mismatch patterns

- Check tax calculations

- Verify date differences

- Document findings

## 9. Limitations and Considerations

### 9.1 Known Limitations

- Maximum file size restrictions

- Processing time for large datasets

- Memory constraints

- Browser compatibility

### 9.2 Performance Considerations

- Large dataset handling

- Memory management

- Processing optimization

- Browser performance

## 10. Security and Data Privacy

### 10.1 Data Security

- Local processing

- No data storage

- Secure file handling

- Input validation

### 10.2 Privacy Considerations

- Data confidentiality

- Access control

- Audit trail

- Compliance requirements

This documentation provides a comprehensive overview of the GST Reconciliation Application, its features, processes, and best practices. The application is designed to streamline the GST reconciliation process while maintaining accuracy and providing detailed analysis capabilities.

I'll break down the reconciliation process from reconciliation.py step by step, explaining the code in detail.

# Detailed Reconciliation Process Analysis

## 1. Initialization and Setup

```python

def \_\_init\_\_(self, df):

# Initialize progress tracking

self.total\_steps = 4 # Total number of steps in the process

self.current\_step = 1

self.step\_progress = 0.0

```

- The reconciliation process is divided into 4 main steps

- Progress tracking is initialized to monitor the reconciliation process

```python

# Store original data for comparison

self.original\_df = df.copy()

# Validate required columns

required\_columns = [

'Source Name', 'Supplier GSTIN', 'Supplier Legal Name', 'Supplier Trade Name',

'Invoice Date', 'Books Date', 'Invoice Number', 'Total Taxable Value',

'Total Tax Value', 'Total IGST Amount', 'Total CGST Amount', 'Total SGST Amount',

'Total Invoice Value'

]

```

- Creates a copy of input data for reference

- Defines required columns for validation

```python

# Split data into books and GSTR-2A records

self.books\_df = df[df['Source Name'] == 'Books'].copy()

self.gstr2a\_df = df[df['Source Name'] == 'GSTR-2A'].copy()

# Initialize processed flag

self.books\_df['processed'] = False

self.gstr2a\_df['processed'] = False

```

- Separates data into two dataframes: Books and GSTR-2A

- Adds a 'processed' flag to track which records have been matched

## 2. Data Cleaning Process

```python

def \_clean\_data(self):

# Standardize GSTIN format

self.books\_df['Supplier GSTIN'] = self.books\_df['Supplier GSTIN'].fillna('').astype(str).str.strip().str.upper()

self.gstr2a\_df['Supplier GSTIN'] = self.gstr2a\_df['Supplier GSTIN'].fillna('').astype(str).str.strip().str.upper()

```

- Standardizes GSTIN format:

- Fills missing values with empty string

- Converts to string

- Removes whitespace

- Converts to uppercase

```python

# Standardize invoice numbers

self.books\_df['Invoice Number'] = self.books\_df['Invoice Number'].fillna('').astype(str).str.strip()

self.gstr2a\_df['Invoice Number'] = self.gstr2a\_df['Invoice Number'].fillna('').astype(str).str.strip()

```

- Standardizes invoice numbers:

- Handles missing values

- Converts to string

- Removes whitespace

```python

# Convert dates to datetime

self.books\_df['Invoice Date'] = pd.to\_datetime(self.books\_df['Invoice Date'], errors='coerce')

self.books\_df['Books Date'] = pd.to\_datetime(self.books\_df['Books Date'], errors='coerce')

self.gstr2a\_df['Invoice Date'] = pd.to\_datetime(self.gstr2a\_df['Invoice Date'], errors='coerce')

```

- Converts date fields to datetime format

- Uses 'coerce' to handle invalid dates by converting them to NaT (Not a Time)

```python

# Convert numeric columns

numeric\_columns = [

'Total Taxable Value', 'Total Tax Value', 'Total IGST Amount',

'Total CGST Amount', 'Total SGST Amount', 'Total Invoice Value'

]

for col in numeric\_columns:

self.books\_df[col] = pd.to\_numeric(self.books\_df[col], errors='coerce').fillna(0)

self.gstr2a\_df[col] = pd.to\_numeric(self.gstr2a\_df[col], errors='coerce').fillna(0)

```

- Converts all numeric columns to proper numeric format

- Handles invalid numbers by converting to 0

## 3. Reconciliation Process

### 3.1 Exact Matches Processing

```python

def \_process\_exact\_matches(self):

for idx, book\_row in self.books\_df.iterrows():

if book\_row['processed']:

continue

# Find matching GSTR-2A record

gstr2a\_match = self.gstr2a\_df[

(self.gstr2a\_df['processed'] == False) &

(self.gstr2a\_df['Supplier GSTIN'] == book\_row['Supplier GSTIN']) &

(self.gstr2a\_df['Invoice Number'] == book\_row['Invoice Number'])

]

```

- Iterates through each unprocessed Books record

- Looks for matching GSTR-2A record based on:

- Unprocessed status

- Matching GSTIN

- Matching Invoice Number

```python

if len(gstr2a\_match) > 0:

gstr2a\_row = gstr2a\_match.iloc[0]

# Calculate tax differences

igst\_diff = book\_row['Total IGST Amount'] - gstr2a\_row['Total IGST Amount']

cgst\_diff = book\_row['Total CGST Amount'] - gstr2a\_row['Total CGST Amount']

sgst\_diff = book\_row['Total SGST Amount'] - gstr2a\_row['Total SGST Amount']

```

- If match found, calculates differences in tax amounts:

- IGST difference

- CGST difference

- SGST difference

```python

# Check if it's an exact match

is\_exact\_match = (

abs(igst\_diff) <= self.tax\_tolerance and

abs(cgst\_diff) <= self.tax\_tolerance and

abs(sgst\_diff) <= self.tax\_tolerance and

(pd.notna(date\_diff) and abs(date\_diff) <= self.date\_tolerance.days)

)

```

- Determines if match is exact based on:

- Tax differences within tolerance (₹1)

- Date difference within tolerance (1 day)

### 3.2 Tax-based Matches Processing

```python

def \_process\_tax\_based\_matches(self):

# Process remaining unprocessed records

unprocessed\_books = self.books\_df[~self.books\_df['processed']]

unprocessed\_gstr2a = self.gstr2a\_df[~self.gstr2a\_df['processed']]

```

- Gets remaining unprocessed records from both sources

```python

for idx, book\_row in unprocessed\_books.iterrows():

# Find potential matches based on tax amounts

potential\_matches = unprocessed\_gstr2a[

(abs(unprocessed\_gstr2a['Total IGST Amount'] - book\_row['Total IGST Amount']) <= self.tax\_tolerance) &

(abs(unprocessed\_gstr2a['Total CGST Amount'] - book\_row['Total CGST Amount']) <= self.tax\_tolerance) &

(abs(unprocessed\_gstr2a['Total SGST Amount'] - book\_row['Total SGST Amount']) <= self.tax\_tolerance)

]

```

- Looks for matches based on tax amounts within tolerance

- Considers IGST, CGST, and SGST amounts

### 3.3 Enhanced Matching

```python

def \_enhanced\_matching(self):

# Fuzzy matching for invoice numbers

for idx, book\_row in self.books\_df[~self.books\_df['processed']].iterrows():

# Calculate similarity scores

similarity\_scores = self.gstr2a\_df[~self.gstr2a\_df['processed']].apply(

lambda x: fuzz.ratio(str(book\_row['Invoice Number']), str(x['Invoice Number'])),

axis=1

)

```

- Uses fuzzy matching to find similar invoice numbers

- Calculates similarity scores using Levenshtein distance

## 4. Report Generation

```python

def \_generate\_final\_report(self):

# Combine matched and mismatch records

self.final\_report\_df = pd.concat([

self.matched\_df,

self.mismatch\_df

])

```

- Combines all matched and mismatched records into final report

```python

def \_generate\_raw\_summary(self):

return {

'total\_books\_records': len(self.books\_df),

'total\_gstr2a\_records': len(self.gstr2a\_df),

'total\_records': len(self.original\_df)

}

```

- Generates summary of raw data counts

```python

def \_generate\_recon\_summary(self):

return {

'exact\_matches': len(self.matched\_df[self.matched\_df['Match Type'] == 'Exact']),

'partial\_matches': len(self.matched\_df[self.matched\_df['Match Type'] == 'Partial']),

'mismatches': len(self.mismatch\_df)

}

```

- Generates summary of reconciliation results

## 5. Integrity Checks

```python

def \_perform\_integrity\_checks(self):

checks = {

'data\_completeness': self.\_check\_data\_completeness(),

'amount\_validation': self.\_check\_amount\_validation(),

'date\_validation': self.\_check\_date\_validation(),

'gstin\_validation': self.\_check\_gstin\_validation(),

'duplicate\_check': self.\_check\_duplicates()

}

```

- Performs various integrity checks:

- Data completeness

- Amount validation

- Date validation

- GSTIN validation

- Duplicate check

## 6. Progress Tracking

```python

def get\_results(self):

return {

'current\_step': self.current\_step,

'step\_progress': self.step\_progress,

'total\_steps': self.total\_steps

}

```

- Provides current progress information:

- Current step number

- Progress within current step

- Total number of steps

This detailed breakdown shows how the reconciliation process:

1. Cleans and standardizes data

2. Performs multiple levels of matching

3. Generates comprehensive reports

4. Performs integrity checks

5. Tracks progress throughout the process

The code is designed to be thorough and accurate while handling various edge cases and providing detailed feedback about the reconciliation process.