SeqManager: Sequencing Data Management System - Complete Documentation

Project Overview

SeqManager is a powerful and intuitive bioinformatics file management system designed to help researchers, bioinformaticians, and lab managers efficiently manage large volumes of sequencing data. The system provides a comprehensive suite of tools to scan, classify, and identify duplicate files, all through a user-friendly web interface. By offering a clear overview of storage usage and providing a safe and easy way to delete redundant files, SeqManager helps to reduce storage costs and improve data organization.

Key Features and Capabilities

- **Automated File Scanning:** Recursively scans specified directories to identify all sequencing-related files.
- **Intelligent File Classification:** Automatically categorizes files into four main groups: Raw Sequencing Data, Aligned Data, Intermediate Files, and Final Outputs.
- **Duplicate File Detection:** Identifies duplicate files based on their MD5 hash, with an optimized algorithm for large files.
- **Interactive Web Interface:** A modern and responsive user interface built with React and TypeScript for easy navigation and data visualization.
- **Safe Deletion Interface:** A secure and user-friendly interface for deleting duplicate files, with built-in safeguards to prevent accidental data loss.

- **Dashboard and Reporting:** A comprehensive dashboard provides a high-level overview of storage usage, file categories, and potential savings from deleting duplicates.
- **RESTful API:** A well-documented RESTful API allows for easy integration with other systems and workflows.

Target Users

- **Bioinformaticians:** Who need to manage large datasets and identify redundant files to free up storage space.
- **Researchers:** Who need a simple and intuitive way to manage their sequencing data without requiring advanced command-line skills.
- **Lab Managers:** Who are responsible for managing storage resources and ensuring data is well-organized.

Technical Architecture

SeqManager is built on a modern and robust technical stack, designed for scalability and ease of use. The system is containerized using Docker, making it easy to deploy and manage.

System Components

- **Python Backend:** A lightweight and efficient backend built with Flask, providing the RESTful API and core business logic.
- **React/TypeScript Frontend:** A responsive and interactive single-page application (SPA) built with React and TypeScript, providing a modern user experience.
- **Docker Containerization:** The entire application is containerized using Docker and Docker Compose, ensuring a consistent and reproducible environment.
- **File Scanner:** A Python-based file scanner that uses a combination of file extensions and regular expressions to classify files and identify duplicates.

File Classification Algorithms

The file classification engine uses a set of predefined patterns to categorize files based on their extensions and filenames. The patterns are designed to be flexible and extensible, and they cover a wide range of common sequencing file types.

The file categories are:

- **Raw Sequencing Data:** `.fastq`, `.fq`, `.fastq.gz`, `.fq.gz`, `.sra`, `.cram`
- **Aligned Data:** `.bam`, `.sam`, `.sorted.bam`, `.markdup.bam`
- **Intermediate Files:** `.bai`, `.csi`, `.fai`, `.dict`, `.idx`, `.bed`, `.vcf.gz`, `.vcf`, `.gvcf`, `.gvcf.gz`, `.m`
- **Final Outputs:** `final.vcf.gz`, `final.vcf`, `report.pdf`, `report.html`, `summary.tsv`, `summary.csv`, `summary.xlsx`, `results.tsv`, `results.csv`, `results.xlsx`, `annotation.tsv`, `annotation.csv`, `annotation.xlsx`

Duplicate Detection Methodology

Duplicate files are identified by calculating their MD5 hash. To optimize performance for large files (over 100MB), the system calculates the hash on the first and last 1MB of the file, rather than the entire file. This approach provides a high degree of accuracy while significantly reducing the time it takes to scan large datasets.

RESTful API Design and Endpoints

The backend provides a RESTful API for interacting with the system. The API endpoints are:

- `POST /api/scan`: Starts a new scan of a specified directory.
- `GET /api/scan/status`: Retrieves the current status of the scan, including progress and any errors.
- `GET /api/results`: Retrieves the results of the last completed scan, including file classifications and duplicate files.
- `POST /api/delete`: Deletes a list of specified files.

Installation and Setup

SeqManager is designed to be easy to install and run using Docker.

Prerequisites and System Requirements

- Docker and Docker Compose must be installed on your system.
- A modern web browser (Chrome, Firefox, Safari, or Edge).
- Sufficient storage space for the sequencing data you intend to manage.

Quick Start Guide

1. Clone the repository to your local machine.
2. Navigate to the root directory of the project.
3. Run the following command to start the application:
   ```
   bash ./run.sh
   ```
4. Open your web browser and navigate to `http://localhost:3000` to access the SeqManager interface.

Environment Configuration Options

The application can be configured using environment variables. You can create a `.env` file in the root directory of the project to set the following variables:

- `FLASK_ENV`: The Flask environment (e.g., `development` or `production`).
- `FLASK_APP`: The entry point of the Flask application (e.g., `code/app.py`).
- `REACT_APP_API_URL`: The URL of the backend API (e.g., `http://localhost:5000/api`).

Usage Guide

Step-by-Step User Interface Walkthrough

The SeqManager user interface is divided into four main sections:

1. **Dashboard:** Provides a high-level overview of your sequencing data, including a summary of file categories, total storage usage, and a list of the largest files.

   (Screenshot: A view of the SeqManager dashboard, showing summary charts and statistics of the scanned data.)

2. **Scanner:** Allows you to start a new scan of a directory and monitor its progress.

   (Screenshot: The scanner view, with an input field for the directory path and a progress bar showing the scan status.)

3. **File Categories:** Displays a detailed breakdown of your files by category, allowing you to see how much space each category is consuming.

   (Screenshot: The file categories view, with a breakdown of file types and their corresponding storage usage.)

4. **Duplicate Files:** Lists all duplicate files found during the scan, allowing you to select and delete them.

   (Screenshot: The duplicates view, showing a list of duplicate file groups with checkboxes for selection and deletion.)

File Scanning Procedures

To start a new scan:

1. Navigate to the **Scanner** page.
2. Enter the full path to the directory you want to scan in the input field.
3. Click the "Start Scan" button.

The progress of the scan will be displayed in real-time. Once the scan is complete, you can view the results on the **Dashboard**, **File Categories**, and **Duplicate Files** pages.

Understanding Classification Results

The **File Categories** page provides a detailed breakdown of your files by category. You can use this information to understand how your storage is being used and to identify which types of files are consuming the most space.

Safe File Deletion Procedures

The **Duplicate Files** page lists all duplicate files found during the scan. To delete duplicate files:

1. Select the files you want to delete by checking the box next to each file.
2. To make this easier, you can use the "Select All Duplicates (Keep First)" button to automatically select all but the first file in each duplicate group.
3. Once you have selected the files you want to delete, click the "Delete Selected Files" button.
4. A confirmation dialog will appear, asking you to confirm that you want to delete the selected files.
5. Click the "Delete" button to permanently delete the files.

Managing Large Datasets Efficiently

SeqManager is designed to handle large datasets efficiently. The optimized duplicate detection algorithm and the ability to scan directories in the background allow you to manage your data without impacting the performance of your system.

Supported File Types

SeqManager supports a wide range of common sequencing file types, which are identified using a combination of file extensions and regular expression patterns.

Raw Sequencing Data

- **Extensions:** `.fastq`, `.fq`, `.fastq.gz`, `.fq.gz`, `.sra`, `.cram`

- **Patterns:**
  - `r'.*\.f(ast)?q(\.gz)?$'`
  - `r'.*\.sra$'`
  - `r'.*\.cram$'`
  - `r'.*_R[12]_.*\.f(ast)?q(\.gz)?$'` (for paired-end reads)

Aligned Data

- **Extensions:** `.bam`, `.sam`, `.sorted.bam`, `.markdup.bam`
- **Patterns:**
  - `r'.*\.bam$'`
  - `r'.*\.sam$'`
  - `r'.*sorted.*\.bam$'`
  - `r'.*markdup.*\.bam$'`
  - `r'.*aligned.*\.bam$'`

Intermediate Files

- **Extensions:** `.bai`, `.csi`, `.fai`, `.dict`, `.idx`, `.bed`, `.vcf.gz`, `.vcf`, `.gvcf`, `.gvcf.g`

- **Patterns:**
    - `r'.*\.bai$'`
    - `r'.*\.csi$'`
    - `r'.*\.fai$'`
    - `r'.*\.dict$'`
    - `r'.*\.idx$'`
    - `r'.*\.bed$'`
    - `r'.*\.vcf(\.gz)?$'`
    - `r'.*\.gvcf(\.gz)?$'`
    - `r'.*\.metrics$'`
    - `r'.*\.log$'`
    - `r'.*\.tmp$'`

Final Outputs

- **Extensions:** `.vcf.gz`, `.vcf`, `.tsv`, `.csv`, `.xlsx`, `.pdf`, `.html`
- **Patterns:**
    - `r'.*final.*\.vcf(\.gz)?$'`
    - `r'.*report.*\.(pdf|html)$'`
    - `r'.*summary.*\.(tsv|csv|xlsx)$'`
    - `r'.*results.*\.(tsv|csv|xlsx)$'`
    - `r'.*annotation.*\.(tsv|csv|xlsx)$'`

API Documentation

Complete REST API Reference

The SeqManager API provides a set of endpoints for interacting with the system programmatically.

`POST /api/scan`

Starts a new scan of a specified directory.

- **Request Body:**
  - `directory_path` (string, required): The full path to the directory to scan.
- **Response:**
  - `200 OK`: If the scan was started successfully.
  - `400 Bad Request`: If the directory path is missing or invalid.
  - `409 Conflict`: If a scan is already in progress.

`GET /api/scan/status`

Retrieves the current status of the scan.

- **Response:**
  - `200 OK`: With a JSON object containing the scan status.
    - `scanning` (boolean): Whether a scan is currently in progress.
    - `progress` (integer): The progress of the scan, from 0 to 100.
    - `current_directory` (string): The directory currently being scanned.
    - `error` (string): Any error message if the scan failed.

`GET /api/results`

Retrieves the results of the last completed scan.

- **Response:**
  - `200 OK`: With a JSON object containing the scan results.
  - `404 Not Found`: If no scan results are available.

`POST /api/delete`

Deletes a list of specified files.

- **Request Body:**
  - `file_paths` (array of strings, required): A list of full paths to the files to delete.
- **Response:**
  - `200 OK`: With a JSON object containing the results of the deletion.
    - `total_deleted` (integer): The number of files successfully deleted.
    - `space_freed_gb` (float): The amount of space freed, in gigabytes.
  - `400 Bad Request`: If the file paths are missing or invalid.

Request/Response Examples

Start a Scan:

```
curl -X POST -H "Content-Type: application/json" -d
'{"directory_path": "/path/to/your/data"}' http://localhost:5000/
api/scan
```

Get Scan Status:

```
curl http://localhost:5000/api/scan/status
```

Get Scan Results:

```
curl http://localhost:5000/api/results
```

Delete Files:

```
curl -X POST -H "Content-Type: application/json" -d
'{"file_paths": ["/path/to/your/data/file1.bam", "/path/to/your/
data/file2.bam"]}' http://localhost:5000/api/delete
```

Error Handling and Status Codes

The API uses standard HTTP status codes to indicate the success or failure of a request.

- `200 OK`: The request was successful.
- `400 Bad Request`: The request was invalid or could not be understood by the server.
- `404 Not Found`: The requested resource could not be found.
- `409 Conflict`: The request could not be completed due to a conflict with the current state of the resource.
- `500 Internal Server Error`: An unexpected error occurred on the server.

Integration Examples for Developers

The SeqManager API can be easily integrated into your existing workflows and scripts. Here is a Python example of how to start a scan and retrieve the results:

```
import requests
import time

API_URL = "http://localhost:5000/api"

def start_scan(directory_path):
    response = requests.post(f"{API_URL}/scan",
json={"directory_path": directory_path})
    response.raise_for_status()
    return response.json()

def get_scan_status():
    response = requests.get(f"{API_URL}/scan/status")
    response.raise_for_status()
    return response.json()

def get_results():
    response = requests.get(f"{API_URL}/results")
    response.raise_for_status()
    return response.json()

if __name__ == "__main__":
    scan_response = start_scan("/path/to/your/data")
    print(scan_response["message"])

    while True:
        status = get_scan_status()
        print(f"Scan progress: {status['progress']}%")
        if not status["scanning"]:
            break
        time.sleep(5)

    results = get_results()
    print("Scan complete!")
    print(f"Found {len(results['duplicates'])} duplicate files.")
```

Security and Safety

File Deletion Safety Measures

SeqManager includes several safety measures to prevent accidental data loss:

- **Confirmation Dialog:** A confirmation dialog is displayed before any files are deleted.
- **"Select All Duplicates (Keep First)" Button:** This button automatically selects all but the first file in each duplicate group, which helps to ensure that you don't accidentally delete all copies of a file.
- **No "Select All" Button for All Files:** There is no button to select all files for deletion, which helps to prevent accidental deletion of all files.

Data Privacy Considerations

SeqManager is designed to be run on your local network, and it does not transmit any of your data to external servers. However, it is important to be aware of the following data privacy considerations:

- **File Paths:** The application stores the full paths to your files in its database.
- **File Hashes:** The application stores the MD5 hashes of your files.

Access Control Recommendations

It is recommended to run SeqManager on a secure network and to restrict access to the application to authorized users.

Backup Recommendations Before Deletion

It is strongly recommended that you back up your data before deleting any files using SeqManager. While the application includes several safety measures, it is always better to be safe than sorry.

Troubleshooting

Common Issues and Solutions

- **"Scan already in progress" error:** This error occurs when you try to start a new scan while another scan is already running. To resolve this issue, wait for the current scan to complete before starting a new one.
- **"Directory does not exist" error:** This error occurs when you enter an invalid directory path. To resolve this issue, make sure that the directory path you entered is correct and that the directory exists.
- **"Path is not a directory" error:** This error occurs when you enter a path to a file instead of a directory. To resolve this issue, make sure that you enter a valid directory path.

Performance Optimization Tips

- **Scan smaller directories:** If you have a large number of files, it is recommended to scan smaller directories individually rather than scanning the entire dataset at once.
- **Run scans during off-peak hours:** To minimize the impact on your system's performance, it is recommended to run scans during off-peak hours.

Error Message Explanations

The application provides detailed error messages to help you troubleshoot any issues that you may encounter. If you see an error message that you do not understand, please refer to the application's logs for more information.

Support and Maintenance Guidance

For support and maintenance guidance, please refer to the project's GitHub repository.

Technical Specifications

Performance Characteristics

The performance of SeqManager depends on a number of factors, including the size of your dataset, the speed of your storage, and the number of CPU cores available on your system. However, the application is designed to be efficient and to minimize the impact on your system's performance.

Scalability Considerations

SeqManager is designed to be scalable and to handle large datasets. The application's modular architecture and its use of a background thread for scanning allow it to handle a large number of files without impacting the performance of the user interface.

Resource Requirements

- **CPU:** 2 or more cores
- **RAM:** 4GB or more
- **Storage:** Sufficient storage space for the sequencing data you intend to manage.

Supported Operating Systems

SeqManager can be run on any operating system that supports Docker, including:

- Linux
- macOS
- Windows

Sources

- **[1] file_scanner.py** - High Reliability - Core backend script for file scanning and classification.
- **[2] app.py** - High Reliability - Main Flask application file containing the API endpoints.
- **[3] App.tsx** - High Reliability - Main frontend component that orchestrates the user interface.
- **[4] Scanner.tsx** - High Reliability - Frontend component for the file scanning interface.
- **[5] DuplicateFiles.tsx** - High Reliability - Frontend component for displaying and managing duplicate files.