Ashley Ang's Project Portfolio Page

Project: Travel Diary

Travel Diary is a desktop application designed to manage trips and travel memories through a Command Line Interface (CLI). It allows users to efficiently log and organize trips, photos, and personal experiences, while also automatically tracking mileage between stops. This feature ensures that users have accurate records of their travel distances, providing a more structured and customizable alternative to traditional travel journaling apps.

Contributions to the Project

New Features

Photo Displaying System

- **What it does:** Implements a system to visualize photos through a desktop window using Java Swing. It utilizes the `Photo`, `PhotoFrame`, and `PhotoPrinter` classes to coordinate the process of displaying photos in a `JFrame` window.
- **Justification:** This feature enhances the app's usability by adding a visual component for photo viewing, creating a richer experience for users as they explore their travel memories.
- **Highlights:**
  - Developed seamless interactions between `Photo`, `PhotoFrame`, and `PhotoPrinter` classes to successfully output the photo visualization window.
  - Researched and fixed compatibility issues for testing GUI elements on Linux systems by modifying `gradle.yml`.
  - Ensures portability across operating systems without compromising functionality.
- **Credits:** Ideas for visualizing photos in `JFrame` were inspired by standard Swing libraries, adapted to fit the Travel Diary application's use case.

Tracker System

- **What it does:** Provides essential utility functions to calculate distance between locations and determine the date range (period) of trips. This is done through methods like `calculateDistance` and `getPeriod`.
- **Justification:** These features streamline the user's journey logging experience by automatically generating critical travel data, minimizing manual input, and enabling efficient management of trip records.
- **Highlights:**
  - Utilized the Haversine formula for accurate distance calculation between photo locations.
  - Integrated period calculation to summarize trips, helping users organize their experiences chronologically.
  - Positioned the Tracker system as a core utility, making the app more robust and user-centric.

- **Credits:** Leveraged geographic and mathematical concepts for distance calculation, referencing Earth radius constants from scientific sources.

Code Contributed

RepoSense link

Project Management

- Managed GitHub releases: **v1.0**, **v2.0**, **v2.1**
- Coordinated team milestones and documentation updates
- Facilitated pull request reviews and issue triaging

Here's the organized and polished Markdown documentation combining all the sections:

Enhancements to Existing Features

Exception Handling

- **Added Index-Based Exceptions:**
  - Added `NotNullException` and `IndexOutOfRangeException`, both inherited from `InvalidIndexException`, to handle errors raised in select statements involving indexes.
- **Trip Selection Validation:**
  - Introduced `TripNotSelectedException` to prevent users from attempting to add photos before a trip is selected.
- **Metadata-Specific Exceptions:**
  - Added `NoDateTimeMetaDataException` and `NoGPSMetaDataException`, both inheriting from `NoMetaDataException`, to allow better narrowing down of exceptions caught during metadata extraction.

UI Enhancements

- Refined the design to make it more visually presentable and neat.
- Improved user interaction flow by reviewing interaction processes and overwriting `toString()` for enhanced representation.
  - Implemented proper `toString()` definitions for `TripManager`, `Trip`, `Album`, and `Photo`, ensuring they directly leverage the `toString()` of related components for better consistency.

New Commands

- **Close Photo Command:**
  - Added and implemented the `ClosePhotoCommand` class, allowing users to close all opened photos using the new "close" command.

Packaging Improvements

- Modified the `InputStream` to package the main CSV data file and assets into the JAR file.
    - Fixed prior issues where files were inaccessible when running the application from the JAR file.

PhotoDateTimeComparator

- **What it does:** Implements the `Comparator` interface to sort `Photo` objects based on their `datetime` attribute, ensuring proper chronological ordering.
- **Justification:** Enables more efficient organization of photos by automatically arranging them by time taken, which is crucial for chronological travel documentation.
- **Highlights:**
    - Handles edge cases where `datetime` values may be null.
    - Ensures consistent ordering, improving the user experience.

Documentation

User Guide

- **Contributions:** Created and improved the documentation for the main features: `Add`, `Select`, `Delete`, `List`

Developer Guide

- **Class Diagrams:**
    - Designed the `Photo` class diagram to illustrate the structure and relationships of the `Photo` component.
    - Created the `Tracker` class diagram, showcasing its utility functions and connections with related components.
- **Sequence Diagrams:**
    - Developed the `Add Photo Process` sequence diagram to explain the interaction flow when adding photos.
    - Created the `Tracker Calculate Distance` sequence diagram, detailing the steps involved in calculating distances between photo locations.
- **Implementation Details:**
    - Documented the implementation of both the `Photo` and `Tracker` components with clear explanations and logical workflows.
