iExtract Developer Documentation

Source Code:

- **iExtract:** https://github.com/Gustakev/cs362-class-project
- **Photo Captioning Component:** https://github.com/BrendxnW/image-captioning-cnn-lstm

Directory Structure:

- **/cli_components/** - Contains the contents of the Presentation Layer, the CLI and GUI and their related utilities
    - **/utils/** - Helper functions for the command-line interface
    - **main_menu.py** - The command-line interface
- **/functional_components/** - Contains each of the components relevant to the Application Layer, Data Layer, and Domain Layer of the functionality of the program, as well as a document containing an explanation of the directory
    - **/backup_locator_and_validator/** - Contains the program logic for loading a backup, validating its format, and creating the BackupModel object that will be populated and used throughout the duration of the program
        - **/app/** - Contains code relevant to the Application Layer of iExtract, the code that executes program logic, like value processing
        - **/data/** - Contains code relevant to the Data Layer of iExtract, the code that reads/writes data
        - **/domain/** - Contains the code relevant to the Domain Layer of iExtract, the data structures and definitions primarily utilized by this module
    - **/conversion_engine/** - Contains the program logic that converts a file of a certain type into a more portable duplicate of a different type and returns the new asset to the Extraction Engine

        - **/app/** - Contains code relevant to the Application Layer of iExtract, the code that executes program logic, like value processing
        - **/data/** - Contains code relevant to the Data Layer of iExtract, the code that reads/writes data
        - **/domain/** - Contains the code relevant to the Domain Layer of iExtract, the data structures and definitions primarily utilized by this module
    - **/file_extraction_engine/** -Contains the code relevant the File Extraction Engine which, based on the BackupModel generated by the Backup Locator and Validator and the SQL Command Facilitator, copies a user's files from their backup into a coherent group of folders named after collections
        - **/app/** - Contains code relevant to the Application Layer of iExtract, the code that executes program logic, like value processing
        - **/data/** - Contains code relevant to the Data Layer of iExtract, the code that reads/writes data
        - **/domain/** - Contains the code relevant to the Domain Layer of iExtract, the data structures and definitions primarily utilized by this module
    - **/sql_cmd_facilitator/** - Contains the code relevant to populating the BackupModel based on SQLite queries upon the backup's databases, so that it can be used by the Extraction Engine
        - **/app/** - Contains code relevant to the Application Layer of iExtract, the code that executes program logic, like value processing
        - **/data/** - Contains code relevant to the Data Layer of iExtract, the code that reads/writes data
        - **/domain/** - Contains the code relevant to the Domain Layer of iExtract, the data structures and definitions primarily utilized by this module
    - **directory-explanation.md** - An explanation of the directory structure
- **/living_documents/** - Contains each iteration of living documents that describe the development progress of the project
- **/reports/** - Contains weekly progress reports describing current goals, progress, and next week's goals
- **/tests/** - Contains test cases
    - **/test_data/** - Contains data relevant to a test file within the 'tests' folder

- **/documentation/** - Contains user and developer documentation
- **.gitignore** - Prevents files from being committed unwillingly
- **README.md** - Explains the project and repository in quick terms
- **iExtract.py** - The launcher for iExtract, which launches either the app in either CLI or GUI mode
- **team-resources.md** - Describes the resources the team uses, such as our miscellaneous design documents, as well as all other primary project documentation
- **requirements.txt** - A list of required modules that the user must install via 'pip install -r requirements.txt', or manually for each, in order to use iExtract

- **How to build the software.** Provide clear instructions for how to use your project's build system to build all system components.

  How to Build:

  1. **Clone repo**
     a. In the terminal run the command:
        `git clone https://github.com/Gustakev/cs362-class-project.git`
  2. **Move to repo location**
     a. In the terminal, run the command: `cd cs362-class-project`
  3. **Install Python**
     a. Install Python 3.14+ for your system from this link: https://www.python.org/downloads/
  4. **Initialize virtual environment**
     a. In the terminal, run the command: `python -m venv venv`
     b. Note: If the above command, or any other command prefixed with Python fails, ensure that you have a Python path variable set on your system. It may either be 'python', 'python3', or something more specific like 'python3.14', depending on your installation.
  5. **Activate virtual environment**
     a. If your machine is running Mac or Linux, in the terminal, run the command:

`source venv/bin/activate`

    b. If you have a Windows machine, in the terminal, run the command: `venv\Scripts\activate `

6. **Install required requirements**
    a. In the terminal, run the command: `pip install -r requirements.txt`
    b. **Note:** Some Linux users may need to run the command: `sudo apt install python3-tk` in order to obtain a working installation of 'tkinter'.

- **How to test the software:**

  Online Continuous Integration: In order to run the system's test cases, the automated system that we have enabled in GitHub via the use of GitHub Actions, utilizing 'unittest' as our testing technology, a developer must simply follow the following steps:

  1. Make a push to the repository containing iExtract (or make a pull request to that repository).
  2. GitHub Actions will automatically run the test suite against the commit. Wait for it to finish.
  3. View the results of the test in the GitHub Actions tab.

  Offline Testing: Run the command 'python -m unittest discover tests' locally from the root directory of the project repo.

- **How to add new tests:** Are there any naming conventions/patterns to follow when naming test files? Is there a particular test harness to use?

  To add tests, from the project root, enter the '/tests/' directory, then create a test file adhering to the following requirements:

  1. Must be testable via the command 'python -m unittest discover tests'
  2. Files must start with 'test_'
  3. Files must end with '.py'

Example: 'test_backup_locator.py'

Conventions (Within Test Files): Test cases will start each function with 'validate' followed by the task.

- **How to build a release of the software:** Describe any tasks that are not automated. For example, should a developer update a version number (in code and documentation) prior to invoking the build system? Are there any sanity checks a developer should perform after building a release?

  Follow These Steps:

  1. Pull the latest changes from the main branch.
  2. Run the test suite offline by entering the command 'python -m unittest discover tests' into the terminal.
  3. Confirm that the tests pass in GitHub Actions for this commit.
  4. Launch iExtract to ensure that basic functionality isn't broken.
  5. Update the version number of iExtract in the 'README.md' file, under the following scheme:
     a. 'major_version.minor_version.bug_fix' (X.Y.Z) versioning scheme, iterating the correct number corresponding to the update
  6. Once the version number has been updated, tag the release/version number using Git:
     a. git tag vX.Y.Z
     b. git push origin vX.Y.Z
  7. Create a GitHub release corresponding to the version you just tagged:
     a. On the project's repo page on GitHub, under 'Releases':
        i. Select 'Create a new release'
        ii. Select the correct tag (the one you just made in *Step 6*)
        iii. Add a release summary with a title and a description
        iv. Attach relevant artifacts, such as a compressed directory containing the repository

v. Release to the correct branch

8. *Post-Release Sanity Checks:*
   a. Clone the repo to a fresh directory and ensure that the installation instructions actually work.
   b. Ensure that the release tag matches the one named in 'README.md'.
   c. Ensure that the release notes are pertinent to the changes made.