# Documentation - Application

## Overview of the Application

This document provides a comprehensive overview of the Energy Predictor Application,   
which is designed to predict energy demand using machine learning models and display results through a graphical user interface.  
The application includes several components, such as data processing scripts, machine learning models, and a Kivy-based GUI.

## 1. main.py

**Scope:**  
The `main.py` script serves as the entry point for the Energy Predictor application. It initializes the Kivy-based GUI, manages the main window, and handles the transitions between different screens within the application.  
  
**Key Components:**  
- \*\*Kivy and KivyMD Integration\*\*: The script heavily relies on the Kivy and KivyMD frameworks to build the user interface.   
- \*\*Screen Management\*\*: Uses the `ScreenManager` to switch between different screens, such as loading screens and the main application interface.  
- \*\*Event Scheduling\*\*: Utilizes the `Clock` from Kivy to schedule events, such as changing screens after a delay and periodically deleting old conversations.  
 **Main Functions:**  
- \*\*build(self)\*\*: This function sets the application title, deletes old conversations using a utility function, and loads the main interface from `main.kv`.  
- \*\*on\_start(self)\*\*: Schedules the initial screen transition and regular cleanup tasks.  
- \*\*change\_screen(self, \*args)\*\*: Switches the current screen to the main application interface.  
  
**Execution Flow:**  
When the application is launched, `main.py` initializes the main window in fullscreen mode and starts the Kivy application loop, which keeps the application running and responsive.

## 2. loadingscreen.py and loadingscreen.kv

**Scope:**  
The `loadingscreen.py` script, in conjunction with the `loadingscreen.kv` file, defines the loading screen that is displayed when the application starts.   
This screen provides a visual indication that the application is initializing.

**Key Components:**  
- \*\*LoadingScreen Class\*\*: Inherits from Kivy's `Screen` class and represents the loading screen.  
- \*\*Kivy Language File\*\*: The `loadingscreen.kv` file contains the layout and design for the loading screen, including any animations or progress indicators.  
  
**Execution Flow:**  
The loading screen is displayed immediately after the application is launched, and it remains visible until the main application interface is ready to be displayed.

## 3. mainwindow.py and main.kv

**Scope:**  
The `mainwindow.py` script defines the main window and its associated interface elements for the application, with the `main.kv` file handling the layout and visual components.  
  
**Key Components:**  
- \*\*MainWindow Class\*\*: Implements the main interface screen, handling the display and interaction logic.  
- \*\*Custom UI Elements\*\*: Includes custom widgets such as `Dot` (for navigation indicators) and `RoundedImage` (for displaying images with rounded corners).  
- \*\*Kivy Language File\*\*: The `main.kv` file contains the structure and design elements for the main window, including button layouts, images, and other interface elements.  
  
**Execution Flow:**  
Once the application transitions from the loading screen, the main window is displayed, providing the user with various options and functionalities to interact with the energy prediction models.

## 4. secondwindow.py and secondwindow.kv

**Scope:**  
The `secondwindow.py` script manages the secondary window in the application, which provides additional functionalities such as generating predictions and visualizing data.  
  
**Key Components:**  
- \*\*SecondWindow Class\*\*: Controls the secondary screen, managing user interactions and displaying results.  
- \*\*Speech Recognition and NLP\*\*: Integrates Vosk for speech recognition and SpaCy for natural language processing, enabling voice commands.  
- \*\*Data Visualization\*\*: Generates charts and graphs based on the prediction results using Matplotlib.  
  
**Execution Flow:**  
The secondary window is displayed when users need to perform specific tasks such as generating energy demand predictions or reviewing historical data trends.

## 5. xpredict\_energy\_demand\_2.py

**Scope:**  
The `xpredict\_energy\_demand\_2.py` script is responsible for generating energy demand predictions using a pre-trained machine learning model.  
  
**Key Components:**  
- \*\*Model Loading\*\*: Loads the pre-trained model using `joblib`.  
- \*\*Data Preparation\*\*: Prepares the input data by creating features from date and time components.  
- \*\*Prediction Function\*\*: Applies the loaded model to the prepared data to generate predictions.  
  
**Execution Flow:**  
This script is called when the user requests a prediction, taking the input parameters (such as date and time range), processing them into the required format, and then using the model to predict energy demand.

## 6. xtrain\_energy\_model\_final\_biomass.py

**Scope:**  
The `xtrain\_energy\_model\_final\_biomass.py` script handles the training of the energy prediction model, specifically focusing on biomass energy data.  
  
**Key Components:**  
- \*\*Data Loading and Preprocessing\*\*: Loads historical data from CSV files, processes it to handle missing values, and extracts relevant features.  
- \*\*Model Training\*\*: Uses a machine learning algorithm (RandomForestRegressor) to train the model on the processed data.  
- \*\*Hyperparameter Tuning\*\*: Performs hyperparameter optimization using techniques like RandomizedSearchCV to improve model performance.  
- \*\*Model Saving\*\*: Saves the trained model to a file for later use in predictions.  
  
**Execution Flow:**  
This script is executed during the model training phase, preparing the dataset, tuning the model, and ultimately saving the trained model for deployment in the prediction script.

## 7. utilities.py

**Scope:**  
The `utilities.py` script contains helper functions that are used across the application. These utility functions handle common tasks, such as cleaning up old data or managing file paths.  
  
**Key Components:**  
- \*\*delete\_old\_conversations\*\*: This function deletes old conversation logs to keep the application data directory clean.  
- \*\*Other Utility Functions\*\*: The script may contain other general-purpose functions that assist in various tasks throughout the application.  
  
**Execution Flow:**  
Utility functions from this script are called as needed by other parts of the application, such as during the initialization phase in `main.py` or when managing data in the secondary window.

## 8. concatenated\_years.csv

**Scope:**  
The `concatenated\_years.csv` file contains aggregated historical energy data, combining multiple years into a single dataset. This file is used as input for training the machine learning models.  
  
**Structure:**  
- \*\*Columns\*\*: The file includes columns for date, time, and various energy metrics, such as actual energy load.  
- \*\*Rows\*\*: Each row represents a specific timestamp and its corresponding energy data.  
 **Usage:**  
This file is critical during the training phase, where it provides the historical data needed to train and validate the prediction models.

## 9. Directories

### conversations/ Directory:  
- \*\*Purpose\*\*: Stores logs of user interactions, particularly conversation histories.  
- \*\*Contents\*\*: JSON or text files containing logged conversations.  
  
### images/ Directory:  
- \*\*Purpose\*\*: Contains images used in the application, such as icons, background images, or visual assets for the UI.  
- \*\*Contents\*\*: PNG, JPEG, or other image file formats used in the UI design.  
  
### models/ Directory:  
- \*\*Purpose\*\*: Stores the pre-trained machine learning models used for making predictions.  
- \*\*Contents\*\*: Files with extensions like `.pkl`, which are model files saved using joblib.  
  
### predictions/ Directory:  
- \*\*Purpose\*\*: Stores output prediction results generated by the application.  
- \*\*Contents\*\*: CSV files or similar formats containing the predicted energy demand data.

## 10. Requirements

* torch=2.3.1+cpu
* torchvision=0.18.1+cpu
* joblib
* scikit-learn
* tensorflow
* spacy
* certifi
* numpy=1.26.4
* setuptools
* sounddevice
* utils
* spacy: en\_core\_web\_sm
* dateparser
* pandas
* matplotlib
* pyaudio
* SpeechRecognition
* Kivymd
* “kivy[base]” kivy\_examples
* vosk