Blood Pressure Tracker

A PyQt5 application to log and evaluate blood pressure data using an SQLite database. PDF export is enabled through HTML rendering.

Project Structure

- **`blood_pressure_entry.py`**
  Defines the `BloodPressureEntry` datatype for representing blood pressure entries.
- **`database_manager.py`**
  Handles database operations, including
  - creating tables
  - adding entries
  - fetching filtered data (depending on the selected time period for data retrieval).
- **`days_option.py`**
  Provides enum-like datatype, for filtering data by predefined time periods in the combobox.
- **`evaluation_dialog.py`**
  Implements the evaluation dialog, displaying blood pressure data in a table with colored categories and PDF export feature. PDF export generates HTML, which is used because of easy formatting and styling options. `res/styles.css` contains CSS for this HTML
- **`main_window.py`**
  Implements the main window with input fields for blood pressure values and buttons for submitting data and opening the evaluation dialog.
- **`main.py`**
  Entry point of the application. Initializes the GUI.

How to Run

1. Install the required dependencies (`PyQt5`, `sqlite3` is included with Python).
2. Run `main.py` from `a21` folder, to launch the application.
3. Use the main window to log blood pressure data and evaluate past entries.

File: `main_window.py`

Description

This file defines the `MainWindow` class, which represents the main window of the application. It displays an image and provides a menu to open the Easter calculation dialog. The window is resizable and adjusts the image to maintain the aspect ratio when resized.

Classes

`MainWindow`

This class represents the main window of the Easter Festival Calculator application. It provides the interface for the user to select the calendar type and year to calculate Easter and related holidays. The window also displays an image and a menu bar.

Methods

- **`__init__(self)`**
  Initializes the main window and sets up the UI.
  - Sets the window title to "Osterfestberechnung."
  - Loads and displays an image (`easter.jpg`) in the window.
  - Adds a menu bar with two menus: "Datei" (File) and "Ostern bestimmen" (Determine Easter).
  - Adds actions to the menus:
    - "Beenden" (Exit): Closes the application.
    - "Ostern bestimmen" (Determine Easter): Opens the Easter calculation dialog.
  - Resizes the window to fit the image and the menu bar.
- **`open_easter_dialog(self)`**
  Opens the Easter dialog to allow the user to calculate Easter and related holidays.
- **`resizeEvent(self, event)`**
  Adjusts the window size and image display when the window is resized.
  - Enforces the aspect ratio of the image by adjusting the height based on the new width.
  - Resizes and repositions the image (`pixmap`) to fit the new dimensions of the window.
  - Calls the parent class `resizeEvent` to ensure proper handling of the resize event.

Dependencies

- **PyQt5 Modules**:
  - `QMainWindow`, `QAction`, `QVBoxLayout`, `QLabel`, `QWidget`: Used to create the window, layout, and UI components.
  - `QPixmap`: Used to load and display the image.
  - `Qt`: Provides the `Qt.KeepAspectRatio` constant for resizing the image while maintaining its aspect ratio.
- **Custom Modules**:
  - `EasterDialog` from `ui.easter_dialog`: Opens the dialog for calculating Easter dates.

Key Features

1. **Resizable Window**
   The window automatically resizes based on the window's dimensions, ensuring that the image retains its aspect ratio.
2. **Menu Bar**
   The menu bar provides two options:
   - **Beenden**: Exits the application.
   - **Ostern bestimmen**: Opens a dialog to calculate the Easter dates.
3. **Image Display**
   Displays an image (`easter.jpg`) in the main window, which is resized to fit the window while maintaining its aspect ratio.
4. **Easter Calculation**
   Allows users to open a dialog to determine Easter and related holidays for a given year and calendar type (Christian or Orthodox).

How It Works

1. The main window displays an image and a menu bar with actions.
2. The user can click "Ostern bestimmen" to open the dialog to calculate Easter.
3. The window automatically adjusts the size of the displayed image when resized, ensuring the aspect ratio remains consistent.

File: `evaluation_dialog.py`

Description

This file defines the `EvaluationDialog` class, which provides a user interface for evaluating blood pressure data. It displays a table of blood pressure entries and their corresponding categories based on systolic and diastolic values. It allows users to filter data by days and export the results to a PDF. The dialog includes a legend explaining the color-coded blood pressure categories.
The program makes use of QTs capability to render HTML. This way consistent styling of the interface, as well as the PDFs generation is given.

Classes

`EvaluationDialog`

This class represents a dialog for evaluating and displaying blood pressure data. The dialog shows a table of entries with color-coded rows indicating the blood pressure category. It provides an option to export the data as a PDF.

Constants

- `COLOR_OPTIMAL`: Represents the color for optimal blood pressure (green).
- `COLOR_NORMAL`: Represents normal blood pressure (slightly darker green).
- `COLOR_HOCHNORMAL`: Represents high-normal blood pressure (yellow).
- `COLOR_HYPERTONIE_GRAD_1`: Represents hypertension grade 1 (light red).
- `COLOR_HYPERTONIE_GRAD_2`: Represents hypertension grade 2 (darker red).
- `COLOR_HYPERTONIE_GRAD_3`: Represents hypertension grade 3 (dark red).
- `COLOR_ISOLIERTE_HYPERTONIE`: Represents isolated systolic hypertension (same as grade 3).
- `COLOR_DEFAULT`: Default color for rows with no category (white).

HTML Legend: HTML string that provides a legend for the color coding of blood pressure categories.

Methods

- **`__init__(self, db_manager, parent=None)`**
  Initializes the dialog window.
  - Sets the window title to "Auswertung."
  - Displays the HTML legend explaining blood pressure categories.
  - Adds a combo box to filter entries by days.
  - Adds a table to display blood pressure data.
  - Includes a button to export the data as a PDF.
- **`display_data(self)`**
  Fetches and displays the blood pressure data based on the selected day

filter from the combo box. Populates the table with the entries.

- `insert_row(self, entry)`
  Inserts a new row in the table with the blood pressure entry data.
  - Colors the row based on the blood pressure category.
- `apply_row_color(self, row_position, color)`
  Applies a background color to the row based on the blood pressure category.
- `get_row_color(self, sys, dia)`
  Determines the color for a row based on systolic and diastolic blood pressure values.
- `export_pdf(self)`
  Exports the blood pressure data and the legend to a PDF file (`evaluation_report.pdf`).
  - Generates the HTML content for the report.
  - Uses the `QPrinter` class to create the PDF.
- `generate_html_content(self)`
  Generates the HTML content for the evaluation report.
  - Loads CSS from an external file (`res/styles.css`).
  - Builds the HTML content for the legend and the table of blood pressure entries.
  - Adds the data from the selected days filter to the table.

Dependencies

- **PyQt5 Modules**:
  - `QDialog`, `QVBoxLayout`, `QLabel`, `QComboBox`, `QTableWidget`, `QTableWidgetItem`, `QPushButton`: Used to create the dialog, layout, and UI components.
  - `QColor`, `QTextDocument`, `QPrinter`: Used for color handling and generating PDF exports.
- **Custom Modules**:
  - `BloodPressureEntry`: Represents a blood pressure entry, used to structure the data.
  - `DaysOption`: Provides day filtering options.
  - `db_manager`: Used for database operations, fetching blood pressure entries.

Key Features

1. **Blood Pressure Categories**
   Blood pressure entries are color-coded based on predefined categories (e.g., Optimal, Normal, Hypertonie Grad 1). The categories are explained in an HTML legend.

2. **Data Filtering**
   Users can select a number of days from the combo box to filter the displayed blood pressure entries.
3. **PDF Export**
   The data, including the color-coded table and legend, can be exported to a PDF file.
4. **Interactive Table**
   The table dynamically updates when the user changes the day filter. Each entry is displayed w

File: `database_manager.py`

Description

This file defines the `DatabaseManager` class, which is responsible for managing the SQLite database used to store and retrieve blood pressure data. The class provides methods to create the database table, add new blood pressure entries, and fetch entries filtered by a specified number of days.

Classes

`DatabaseManager`

This class manages the interaction with an SQLite database for storing and retrieving blood pressure data.

Methods

- **`__init__(self, db_path='db/blood_pressure.db')`**
  Initializes the `DatabaseManager` instance and establishes a connection to the SQLite database.
  - The default database path is `db/blood_pressure.db`.
  - Calls `create_table()` to ensure the database table is created.
- **`create_table(self)`**
  Creates the `blood_pressure` table in the database if it does not already exist.
  - The table has columns: `id` (primary key), `timestamp` (datetime of the entry), `sys` (systolic blood pressure), `dia` (diastolic blood pressure), and `pulse` (heart rate).
- **`add_entry(self, sys, dia, pulse)`**
  Adds a new blood pressure entry to the database.
  - The `timestamp` is automatically set to the current date and time.
  - The systolic (`sys`), diastolic (`dia`), and pulse (`pulse`) values are provided as parameters.
  - The data is inserted into the `blood_pressure` table.
- **`fetch_filtered_data(self, days)`**
  Fetches blood pressure data from the database, filtered by the number of days specified.
  - `days`: The number of days of data to retrieve (entries newer than this threshold).
  - The method returns a list of tuples containing the `timestamp`, `sys`, `dia`, and `pulse` values of the fetched entries.

Database Schema

The SQLite database contains a single table `blood_pressure` with the following schema: - `id` (INTEGER, Primary Key): A unique identifier for each blood pressure entry. - **`timestamp`** (TEXT): The timestamp when the entry was created (formatted as `YYYY-MM-DD HH:MM:SS`). - **`sys`** (INTEGER): The systolic blood pressure value. - **`dia`** (INTEGER): The diastolic blood pressure value. - **`pulse`** (INTEGER): The pulse rate.

Usage

1. **Database Initialization**: When the `DatabaseManager` is instantiated, it connects to the SQLite database (or creates it if it doesn't exist) and ensures the `blood_pressure` table is created.
2. **Adding Entries**: You can add new blood pressure entries by calling `add_entry()` with the systolic, diastolic, and pulse values. The current timestamp is automatically assigned.
3. **Fetching Data**: To fetch blood pressure entries within a specified range of days, use `fetch_filtered_data()`, passing the number of days to filter by. The method returns all matching entries within the last `n` days.

Dependencies

- **sqlite3**: Used for database management (creating tables, inserting data, and fetching entries).
- **datetime**: Used for handling timestamps and calculating date ranges for filtering.