Badea Stefan-Vasile (324CC)
Pirvu Tatiana-Andreea-Daria (324CC)

Software architecture description
Task Planner App
(TaskUP)

TaskUP is a task management application developed for use on desktop, with a visual interface based on Tkinter. Users can add tasks, setting deadlines for each of them. The app includes a machine learning (ML) module that, based on a dataset containing information about various types of tasks, will estimate the required duration for each task and generate a personalized schedule. The goal of the application is to help users plan and organize their time efficiently, particularly in academic settings.

Main features:

- **User authentication** - Users can create accounts to save progress and maintain stored data over time.
- **Task management** - Users can create, edit, and delete tasks. Task attributes include a title, description, priority, deadline, and estimated completion time.
- **Integrated calendar** - A calendar view will allow users to see tasks distributed by day, and potentially by week or month. Data visualization with charts or diagrams can help illustrate time allocation and deadline proximity.
- **Task duration estimation using ML** - A machine learning model, leveraging a dataset of task types, difficulty, and time, will estimate the time needed for tasks. Using *scikit-learn*, this model could employ a regression or classification algorithm to predict task durations.
- **GUI** - Built with Tkinter to ensure usability on desktop devices. Designed to keep the UI clean and intuitive, facilitating smooth task creation and calendar navigation.

TaskUP will be a fully Python-based application, utilizing *Tkinter* for the graphical interface, *Pandas* for data processing, and *scikit-learn* for machine learning algorithms. This setup will support the app's data handling and machine learning functionalities while providing a smooth user interface for managing tasks and schedules.

System Overview:

- Graphical Interface (*Tkinter*):
  - Provides a user-friendly interface for task management, allowing users to add, edit, and delete tasks.
  - Designed to keep interactions smooth and intuitive, optimized for both laptop and desktop environments.
- Database (*SQLite*):
  - Acts as the storage layer, holding data on task types, their difficulties, and estimated completion times.

    - Ensures data persistence across sessions and is lightweight, fitting the local-only architecture.
- Machine Learning Module:
    - Implements regression or classification models, using linear regression or k-nearest neighbors (KNN) to estimate task durations.
    - Pulls data from SQLite to train and make predictions, leveraging historical data to improve task duration estimates over time.

This centralized, local architecture ensures privacy and efficiency, with all components running seamlessly on the user's device.

Design Patterns: Model-View-Controller (MVC) Pattern

The Model-View-Controller (MVC) pattern structures **TaskUP** into three distinct components, optimizing modularity and separation of concerns:

- **Model:** Acts as the application's data layer, handling task-related data operations, storage, and logic. *SQLite* is utilized to manage the persistent storage of tasks and associated attributes, supporting operations such as adding, modifying, and deleting tasks. Data-processing functions, including task duration estimation via machine learning, are integrated within the Model to encapsulate all data management responsibilities.
- **View:** Provides the graphical user interface (GUI) that enables users to interact with the application. *Tkinter* is employed for rendering the View, facilitating task management, calendar visualization, and ML-based suggestions in a cohesive interface. The View is updated dynamically by the Controller based on user interactions.
- **Controller:** Acts as the intermediary, connecting user actions in the View with data operations in the Model. It handles application logic by responding to user inputs, triggering data updates within the Model, and refreshing the View accordingly. This separation allows for an organized flow of information and maintains a clear boundary between the user interface and data processing layers.

Detailed Component Design

1. Graphical Interface (Tkinter):
    - Task Panel: This component is responsible for presenting the user with an interactive interface for task management. It allows users to create new tasks, modify existing ones, and delete tasks. For each task, the panel includes fields such as:
        - Deadline: A date picker or input field where users specify when the task is due.

    - Difficulty: A dropdown or slider allowing users to select the difficulty level of the task, which could influence time estimation.
    - Description: A text field for users to enter additional details related to the task.
- Calendar: The calendar view is designed to display the tasks assigned to specific days. Users can interact with the calendar to see the tasks scheduled for each day, a visual representations such as colored blocks or icons denoting different task types or deadlines.The calendar serves as a primary tool for time management, helping users visualize their task schedules and deadlines effectively.

2. Database (*SQLite*):
    - Structure: The database is built around several key tables that organize task-related data:
    - Task Table: Stores basic task information such as task ID, name, description, deadline, difficulty, and estimated duration.
    - Task Type Table: Stores predefined types of tasks, each with associated characteristics that could be used for classification or estimation purposes (e.g., research task, coding task).
    - Completion Data Table: Logs task completion times and success rates to help train the ML model. This data will be updated as users complete tasks.
    - Difficulty Level Table: Stores predefined difficulty levels, which can be linked to tasks to refine predictions.
    - ML Interaction: The database serves as the source of training data for the machine learning module. It continuously collects data about task types, user performance, difficulty, and estimated vs. actual task durations. This data helps refine the ML model's accuracy in estimating task durations.

3. Machine Learning Module:

- ML Algorithms: The module employs algorithms to predict the duration of tasks based on data and task characteristics. The primary algorithms used are:
- Linear Regression: A simple model used to predict the task duration based on numerical input such as difficulty or type. It assumes a linear relationship between features (e.g., difficulty, task type) and the task duration.
- Classification Algorithms: These will be used for categorizing tasks into different types, with each category having an associated predicted time duration. Algorithms such as *k-Nearest Neighbors (KNN)* or *Decision Trees* may be employed for this purpose, enabling the model to group tasks based on similarity and predict times based on similar tasks.

Pandas and scikit-learn:

- Pandas: Pandas is used to handle and process data. It will load the data from the SQLite database into DataFrames, preprocess it (e.g., handle missing values, normalize or scale features), and prepare it for training.
- Scikit-learn: This library is used to implement the machine learning algorithms. It will train the model on the data from the database and periodically update it based on new task completion data. The model will use features such as task type, difficulty, and previous completion data to predict the expected duration of new tasks.
- Model Training and Adjustment: The machine learning model is trained on historical data and updated periodically to reflect real-world performance. As users log new tasks and provide feedback (completion times), the model will adapt, recalibrating to offer more accurate predictions over time. This continuous learning process is key to improving the task duration estimates as the app gathers more data.

Deployment and Testing

- Run Requirements: The application requires Python 3.8+ and the mentioned libraries (Tkinter, Pandas, scikit-learn, SQLite3).
- Distribution: The application can be distributed as a Python executable using PyInstaller for portability.
- Testing: Testing will include verifying the Tkinter interface and the ML algorithms with different input data to ensure the accuracy of duration estimates. Manual testing will be performed for the UI, and automated testing for the ML functions.

Conclusions

Developing TaskUP in Python provides an efficient and scalable solution, benefiting from the direct integration of machine learning algorithms with SQLite for data storage and Tkinter for building the graphical interface. The application will require approximately 80 hours for development, adjustments, and testing. The complexity of the project increases as optimizations are made to the ML model and as the interface is refined to ensure it is user-friendly and has cross-platform support.