DoDate Project — Iteration 3

MVP (Minimum Viable Product)

Warning

Do not modify `application_controller.py` or `index.html` unless you have received explicit permission from Cal. (not the TAs, not Brandon, not your congressman, not the Cal imposter that haunts your dreams)

Unauthorized changes to either file will result in a grade of **zero for the entire iteration**.

These files provide the structural foundation for the API and front-end interface. Altering them without approval can break system functionality and invalidate grading.

Installing libraries

This project uses some libraries that are not included in Python. You will need to install the libraries with the following commands:

```
pip3 install "fastapi[standard]"
```

If you are using a Windows machine you will need to use "pip" instead of "pip3".

Description

Welcome to your first implementation phase of the DoDate project!

In this iteration, you will implement the core functionality required to produce a working MVP (Minimum Viable Product). You will complete all required methods that allow the system to create, manage, and persist planner data.

All required work is marked with `TODO` comments. A `TODO` represents a clearly defined, incomplete unit of work.

Modern IDEs make these easy to track:

- **PyCharm** includes a built-in TODO tool window.
- **VS Code** supports extensions such as *TODO Tree* for similar functionality.

These tools allow you to:

- View all TODOs organized by file
- Navigate directly to each TODO
- Ensure nothing is missed

Once a TODO has been fully completed, replace `TODO` in the comment with `DONE`.

File List

Application_Controller

This file acts as the system's API (Application Programming Interface). It serves as the bridge between the front-end and the backend logic.

The Application_Controller:

- Receives requests from the front-end (index.html)
- Calls the appropriate backend methods
- Returns structured data back to the browser
- Ensures user actions are reflected in the planner data

You are not responsible for modifying its structure in this iteration. It is included here so you understand how your backend code is being used.

Planner

The Planner class is the central coordinating component of the system. It owns and manages all categories, tasks, and events. Most application behavior ultimately routes through this class.

- constructor
- getters and setters
- **to_dict:** Converts the entire Planner object (including categories, tasks, and events) into a dictionary format suitable for JSON serialization and saving to file. Used when saving our planner to a file.
- **from_dict:** Reconstructs a Planner object from dictionary data loaded from JSON, restoring categories, tasks, and events as proper objects. Used when loading our planner from a file.
- **create_task:** Creates and adds a new Task to the appropriate category within the planner.
- **set_task_status:** Updates the status of a specific task (e.g., incomplete, in progress, completed).
- **set_task_todays_focus:** Updates whether a specific task is marked as part of Today's Focus.

Task

The Task class represents an actionable to-do item. Tasks have descriptive information and a status (incomplete, in progress, completed).

- constructor
- getters and setters
- **to_dict:** Converts a Task object into a dictionary so it can be serialized to JSON.
- **from_dict:** Creates a Task object from dictionary data loaded from JSON.

Category

The Category class groups related tasks and events together. It provides organizational structure within the planner.

- constructor

- getters and setters
- **to_dict:** Converts a Category object (including its tasks and events) into dictionary form for serialization.
- **from_dict:** Rebuilds a Category object from dictionary data, restoring contained tasks and events.

Event

The Event class represents a scheduled occurrence such as a meeting, appointment, or gathering. Unlike tasks, events are typically tied to a specific date or time and do not have progress states.

- constructor
- getters and setters
- **to_dict:** Converts an Event object into dictionary form for JSON serialization.
- **from_dict:** Reconstructs an Event object from dictionary data loaded from JSON.

Data_Manager

The Data_Manager class handles persistence. It is responsible for loading planner data from storage and saving planner data back to storage.

While this functionality would typically be included elsewhere (in the main class like Planner or the API), it is separated here for simplicity and organization.

- constructor
- **open_planner:** Opens a saved planner file, loads its JSON data, and returns a fully constructed Planner object.
- **save_planner:** Converts the Planner object to dictionary form and writes it to a JSON file for persistent storage.

Running your Program

Once you are ready to run your program, you can run the application_controller.py file by clicking the run button on your IDE while the file is open or using the following command:

```
python3 application_controller.py
```

If you are on a Windows machine you will need to use "python" instead of "python3".

Submission Instructions

Open a terminal and ensure you are inside the repository directory before running any Git commands.

Stage your changes:

```
git add .
```

Commit your work with a clear message describing what you added:

```
git commit -m "Completed Iteration 3 to make the MVP"
```

Push your commit to GitHub:

```
git push
```

Open your repository in GitHub and ensure your work appears in the files.