FIT1056 – Introduction to software engineering
Semester 2
PST3

The Project Journey: An Overview - **Problem-solving tasks 3**

We will use the case study of a Music School Management System (MSMS) to introduce the fundamental software engineering (SE) concepts from the perspective of a developer. Before you begin, review the **Applied#1** slides to make sure you understand how to work with the Git repository.

Music School Management System (MSMS)
(Individual Task)

Objective: You will build a Music School Management System (MSMS) over five stages. Each stage is a direct upgrade of the previous one, taking you from a simple script to a robust, professional application.

Important Notes:

- This is an **individual task**.
- You are **not** required to create a formal Software Requirements Specification (SRS) document.
- You **must** use Git to track your progress. You will make a new "commit" after completing each part.

- **PST1: The Foundation.** Build a simple, in-memory prototype.
- **PST2: The Upgrade.** Add file storage, data validation, and better organization.
- **PST3: The Architecture.** Rebuild with a professional Object-Oriented (OOP) design.
- **PST4: The User Interface.** Replace the text console with a modern Graphical User Interface (GUI).
- **PST5: The Quality Assurance.** Make the application "crash-proof" and prove it works with automated tests.

Part 0: Project Setup and Git Initialization

The same Git steps in our PST1 documentation are mentioned here. You may make the changes accordingly as you want.

Goal: Create a project directory and initialize it as a Git repository. This is the foundation for your entire project.

Make sure you are on the correct branch (individual). If you have any un-committed changes, commit them now and push. Feel free to push after each individual commit, there is no problem with doing so.

Your Task:

1. Open your terminal or command prompt (Follow the Applied-Week01 Slides).
2. Create a new folder for your project (e.g., msms-project) and navigate into it.
3. Initialize a new Git repository in this folder. This command creates a hidden .git subdirectory that will track all your changes.

```
mkdir msms-project
cd msms-project
```

```
git init
```

4. Create your main source code file. (e.g., main.py).
5. Add this new file to the Git staging area and make your first commit. A commit is a snapshot of your code at a specific point in time.

```
# Add all new/modified files to be tracked
git add .

# Save the snapshot with a descriptive message
git commit -m "Initial commit: Set up project structure"
```

Part 3 (PST3): The Architectural Redesign (OOP)

Your Goal: To solve the "Messy & Unscalable" problem of the procedural pst2_main.py file. We will refactor the project into a professional Object-Oriented application. This clean architecture is essential for handling the complex new features like courses and scheduling.

The New Directory Structure: We will now organize our code into logical directories and files.

```
.
├── app/
│   ├── __init__.py
│   ├── user.py          # Base User class
│   ├── student.py       # StudentUser class
│   ├── teacher.py       # TeacherUser class AND Course class
│   └── schedule.py      # The "brain" of the app, our ScheduleManager
├── data/
│   └── msms.json        # Our data file now lives here
└── main.py              # The main application entry point (the "View")
```

Fragment 3.1: The New Blueprints (The Model Layer)

- **Mission:** To define the core entities of our system as dedicated Python classes. These objects will not only hold data but will eventually be able to perform actions. This creates a clean "Model" layer.
- **Your Task:**
  1. Create the new app/ and data/ directories. Move your msms.json into data/.
  2. In app/user.py, create a base User class.
  3. In app/student.py, create a StudentUser class that inherits from User.
  4. Crucially: In app/teacher.py, you will define both the TeacherUser class and the Course class.
- **Check the Templates give in:**
  - **app/user.py**
  - **app/student.py**
  - **app/teacher.py**
  - **data/msms.json**

Checkpoint: Commit Your Progress

Save your file and commit the change.

```
git add app/
git commit -m "refactor(models): Establish OOP models for User, Student, Teacher, and Course"
```

(Note: "Feat" is a common convention for a commit that introduces a new feature.)

Fragment 3.2: The "Brain" of the System (The Controller Layer)

- **Mission:** To create the central ScheduleManager class. We will now explicitly add the attendance_log to its state and ensure the persistence methods handle it correctly.
- **Your Task:**
  - In app/schedule.py, open the ScheduleManager class.
  - In the __init__ method, add self.attendance_log = [] to properly initialize the attribute.
  - Implement a _load_data() method. This is the crucial link: it reads the raw dictionaries from JSON and uses them to create instances of your StudentUser, TeacherUser, and Course classes (*check the template give in the data folder. You can modify the JSON as you would like to build your data*).
  - In the _load_data() method, add logic to populate self.attendance_log from the "attendance" key in the JSON file. We will use .get() to handle old data files gracefully.
  - Implement a _save_data() method that does the reverse: it converts the lists of objects back into serializable dictionaries to be written to the JSON file.
  - In the _save_data() method, add logic to write the contents of self.attendance_log back to the JSON file.

- **Check the Template give in:**
  - **app/schedule.py**

Checkpoint: Commit Your Progress
Save your file and commit your new function.

```
git add app/schedule.py
git commit -m "controller: Fully integrate attendance_log into ScheduleManager state and persistence"
```

Fragment 3.3: Implementing Core Business Logic

- **Mission:** To implement the check_in method, which can now safely assume that self.attendance_log exists and will be persisted.
- **Your Task:**
  - Implement the check_in method as previously described. The logic for the method itself does not change, but it will now work correctly within the class structure.
  - **Check the Template in (Fragment3_3.py and add to app/schedule.py)**

Checkpoint: Commit Your Progress
Save your file and commit this important update.

```
git add app/schedule.py
git commit -m "feat(logic): Implement student check-in method with validation"
```

Fragment 3.4: The New Front Desk & Main Entry Point (The View Layer)

- **Mission:** To create a new, clean entry point for our application (main.py) that acts as the "View". It will be responsible only for user interaction and will delegate all the hard work to our ScheduleManager object.
- **Your Task:**
  1. Create the new top-level main.py.
  2. This file will import the ScheduleManager.
  3. Implement view functions like front_desk_daily_roster(manager, day) that take the manager as an argument, call its methods, and then format and print the results.
  4. The main() function will create just one ScheduleManager instance at the start and pass it around as needed.
- **Check the given Template in (main.py)**

Checkpoint: Commit Your Progress
Save your file and commit this important update.

```
git add main.py
git commit -m "refactor(app): Connect view layer to controller, completing PST3 OOP redesign"
```

```
##########################################
# 4️⃣  Push the new commit(s) up to GitHub
##########################################
git push origin individual
#          │       │
#          │       └─ the branch name on GitHub
#          └─────────── the remote repository (GitHub by default)
```

Instructions

- **Template files provided:** You will find a skeleton for every task (Fragment#_#.py)
- **Finish • Run • Test:** Complete each fragment, run it locally, and confirm it works.
- **Commit & push:** Test the complete application, then commit and push.
- **One README to rule them all:** Create **one** high-quality README.md at the root of your repo that explains
  - what each part does,
  - how to run and test the full program, any design choices or assumptions you made.
- **A clear, detailed README is worth marks, treat it like part of the assignment**

Submission Information

- Please ensure you submit your work on Moodle before the deadline to avoid any late penalties.
- Upload your task files as a single ZIP file.
- In addition, make sure to commit and push your code to Git before the same deadline.