FIT1056 – Introduction to software engineering
Semester 2
PST2

The Project Journey: An Overview **- Problem-solving tasks 2**

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 2 (PST2): The Persistence Upgrade

Your Goal: To solve the "Amnesia & Chaos" problem of PST1. We will refactor the project to save all data to a single, structured msms.json file. This makes the data permanent. We will also add more robust data management functions (full ------ create, read, update, and delete CRUD) and new features like student check-ins.

The New Structure: We will move all our logic into a new, more organized file: pst2_main.py. The old MSMS.py will no longer be used.

Fragment 2.1: The Core Persistence Engine

- **Mission:** To create the two most critical functions of this stage: saving and loading the application's entire state to and from a JSON file. This is the new "brain" that replaces the in-memory lists.

Your Task:

1. Create a new file, pst2_main.py.
2. Define a global dictionary variable named app_data. This will hold all our students, teachers, attendance records, and ID counters.
3. Implement load_data(path): This function attempts to read the JSON file. If the file doesn't exist, it initializes app_data with a default, empty structure.
4. Implement save_data(path): This function writes the current state of the app_data dictionary to the JSON file in a clean, human-readable format.

- **Check the given Template in (pst2_main.py)**

Checkpoint: Commit Your Progress
Save your file and commit the change.

```
git add pst2_main.py
git commit -m "feat(persistence): Implement core JSON data saving and loading engine"
```

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

Fragment 2.2: Refactoring and Expanding CRUD Operations

- **Mission:** To rewrite the data management functions from PST1 to work with the new app_data dictionary instead of separate lists. We will also add the missing Update and Delete functionalities.
- **Your Task:**
  1. Re-implement add_teacher to add a teacher dictionary to the app_data['teachers'] list.
  2. Implement update_teacher(teacher_id, **fields): This powerful function takes a teacher's ID and any number of keyword arguments (e.g., name="New Name") and updates their record.
  3. Implement remove_teacher(teacher_id).
  4. Implement the update_student and remove_student functions in the same way.
- **Check the given Template in (Fragment2_2.py add to pst2_main.py)**

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

```
git add pst2_main.py
git commit -m "refactor(crud): Implement full file-based CRUD for students and teachers"
```

Fragment 2.3: Implementing New Receptionist Features

- **Mission:** To add the new features specified for PST2: checking a student in and printing a physical "badge" for them.
- **Your Task:**
  1. Implement check_in(student_id, course_id, timestamp). This will create a new record in the app_data['attendance'] list.
  2. Implement print_student_card(student_id). This function will find a student and write their details to a new text file.
- **Check the given Template in (Fragment2_3.py add to pst2_main.py)**

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

```
git add pst2_main.py
git commit -m "feat(features): Implement student check-in and card printing"
```

Fragment 2.4: The Refactored Main Loop

- **Mission:** To update the main application loop to be aware of the new persistence model. It must load data at the start and save data after every change.
- **Your Task:**
  1. Create the main() function.
  2. The very first thing it does is call load_data().
  3. The menu should provide options for all the new PST2 functions.
  4. After any function that modifies data (like add_teacher, check_in, remove_student), save_data() must be called immediately to make the change permanent.
- **Check the given Template in (Fragment2_4.py add to pst2_main.py)**

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

```
git add pst2_main.py
git commit -m "refactor(app): Update main loop for persistence, completing PST2"
```

```
##########################################
# 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.