

# **FIT1056 – Introduction to software engineering**

## **2025 Semester 2**

### **PST1**

#### **The Project Journey: An Overview - Problem-solving tasks 1**

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 NO group submissions**.
- 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

**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.

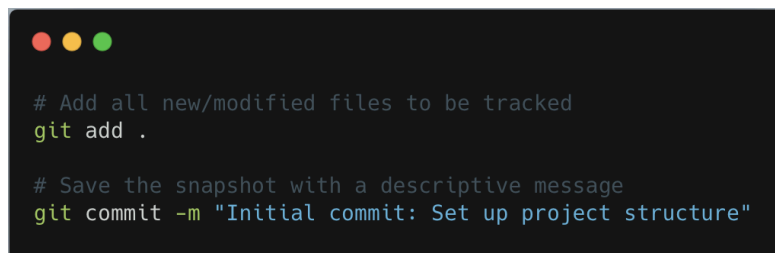


```
mkdir msms-project
cd msms-project
```



```
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 1 (PST1): The In-Memory Prototype

**Your Goal:** To build a fully functional, multi-role Music School Management System that proves the core concepts work. For this prototype, all data will be stored in memory, which means it will be erased every time the program stops. This is fast for development and lets us focus on the logic first.

**The Structure:** Everything will be built inside a single file: **MSMS.py**

### *Fragment 1.1: The Blueprints & Data Stores*

- **Mission:** To define the fundamental data structures for our system (Student and Teacher) and create the in-memory lists that will act as our temporary "databases".
- **Your Task:**
  1. Create a new directory for your project. Inside it, initialize a Git repository.
  2. Create a file named MSMS.py.
  3. In MSMS.py, define a Student class to hold a student's ID, name, and a list of the instruments they are learning.
  4. Define a Teacher class to hold a teacher's ID, name, and their instrument specialty.
  5. Create the global lists (student\_db, teacher\_db) and ID counters that will store all our data for this session.
- **Check the given Template in Fragment1\_1.py**

### **Checkpoint: Commit Your Progress**

Save your file and commit the change.

```
git init
git add MSMS.py
git commit -m "feat(models): Define data models and in-memory stores for Student and Teacher"
```

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

### *Fragment 1.2: Core Helper Functions*

- **Mission:** To implement the "backend" functions that directly manipulate our data stores. These are the fundamental building blocks of the system.
- **Your Task:**
  1. `add_teacher(name, speciality)`: Creates a new Teacher object and adds it to the `teacher_db`.
  2. `list_students()`: Prints a formatted list of all students and their enrolled courses.
  3. `list_teachers()`: Prints a formatted list of all teachers.
  4. `find_students(term)`: Searches `student_db` by name.
  5. `find_teachers(term)`: Searches `teacher_db` by name or speciality (case-insensitive).
- **Check the given Template in (Fragment1\_2.py and add to MSMS.py)**

#### **Checkpoint: Commit Your Progress**

Save your file and commit your new function.

```
git add MSMS.py
git commit -m "feat(core): Implement core helper functions for data management"
```

### *Fragment 1.3: Front Desk Functions*

- **Mission:** To create the high-level functions that a receptionist would use. These functions will provide a simpler interface by calling the more detailed core helper functions.
- **Your Task:**
  1. Create a new helper function `find_student_by_id(student_id)` which is needed for enrolment.
  2. Implement `front_desk_register(name, instrument)`. This is a "combo" function that creates a student and immediately enrolls them.
  3. Implement `front_desk_enrol(student_id, instrument)`.
  4. Implement `front_desk_lookup(term)` which searches both students and teachers.
- **Check the given Template in (Fragment1\_3.py and add to MSMS.py)**

#### **Checkpoint: Commit Your Progress**

Save your file and commit this important update.

```
git add MSMS.py
git commit -m "feat(frontdesk): Implement high-level receptionist functionality"
```

### Fragment 1.4: Main Application Menu

- **Mission:** To build the final interactive menu that a user will see. This ties all the front\_desk\_ functions together.
- **Your Task:**
  1. Create a main() function that contains the primary application loop (while True).
  2. Inside the loop, display a clear menu of options for the user.
  3. Prompt for input and use an if/elif/else block to call the correct front\_desk\_ function based on the user's choice.
- **Check the given Template in (Fragment1\_4.py and add to MSMS.py)**

### Checkpoint: Commit Your Progress

Save your file and commit this important update.

```
git add MSMS.py
git commit -m "feat(app): Build main menu and complete PST1 prototype"
```

```
#####
# 4 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
  - o what each part does,
  - o 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.