Lab 8
Detailed Design
Tasks

Issues

1. Teachers

   Issue 1: Architecture Design Diagram Review

   Issue 2: Teacher explains the Detailed Design template

2. Students

   Task 1: Students prepare Detailed Design document

Issue 1: Architecture Diagram Review

- Feedback:
  - 1.2 The main references are the SRS document and the long description.
  - 1.3 Purpose of architecture design is to divide the system into subsystems, so they can be developed individually.
  - 2.1 Use case diagram and class diagram
    - Change the class diagram into architectural design
    - In UI layers of the subsystems, consider only the use cases from the use case diagram.
    - Some teams put the use case and class diagrams in 2.1 and then ignored them in the rest of architectural design.
  - 2.2 Design model includes both MVC and three-layered models

Issue 1: Architecture Diagram Review

- Feedback – 2.3 whole system architecture
  - Whole system architecture diagram is missing in some
  - Whole system architecture must be consistent with the class diagram in 2.1
    - Every class in 2.1 must appear in diagram in 2.3.
    - Every class in 2.3 must appear in diagram in 2.1.
    - The whole system architecture in 2.3
      - Shows how the system is divided into subsystems
      - Shows how the subsystems are coupled with each other
      - Contains class names, but no attributes, operations, UI or database; those things belong to Section 3 subsections.
    - A class cannot appear in 2 subsystems.
      - Otherwise those 2 subsystems are highly coupled together.

Issue 1: Architecture Diagram Review

- Feedback – 3.X Subsystems UI's
  - The UI's in the subsystems must be from the use case diagram in 2.1.
    - Except for MainUI or HomeUI, which contains home pages.
  - A UI for a use case cannot appear in 2 subsystems
    - Otherwise the 2 subsystems are highly coupled together.

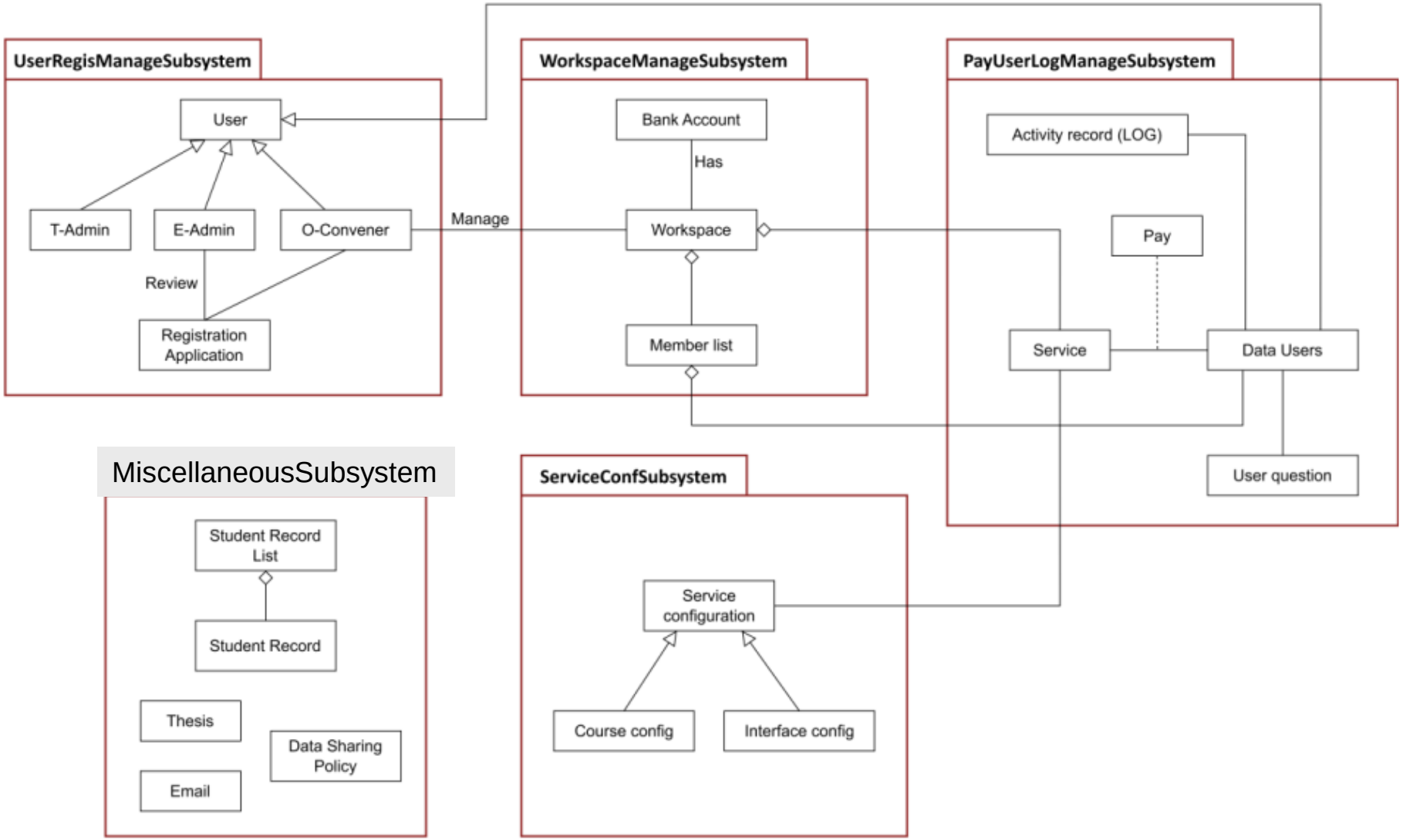
Issue 1: Architecture Diagram Review

- Feedback – 3.X.2 database tables
  - A table's attributes must match the class' attributes.
    - Especially so in the detailed design we do this week.
  - DBMS class thoroughly covered how to use tables to represent associations between entities.
    - We briefly covered this in SE Lecture 10.
    - Recall the 10 cases in DBMS class; following 3 are most important.
    - For 1 to many relation, the foreign key from the "1" table is "marked for association" in the "many" table.
    - For many to many relation, the foreign keys for both tables are "marked for association" in the association table.
    - For inheritance, we can extend the table to accommodate both the super and sub classes.

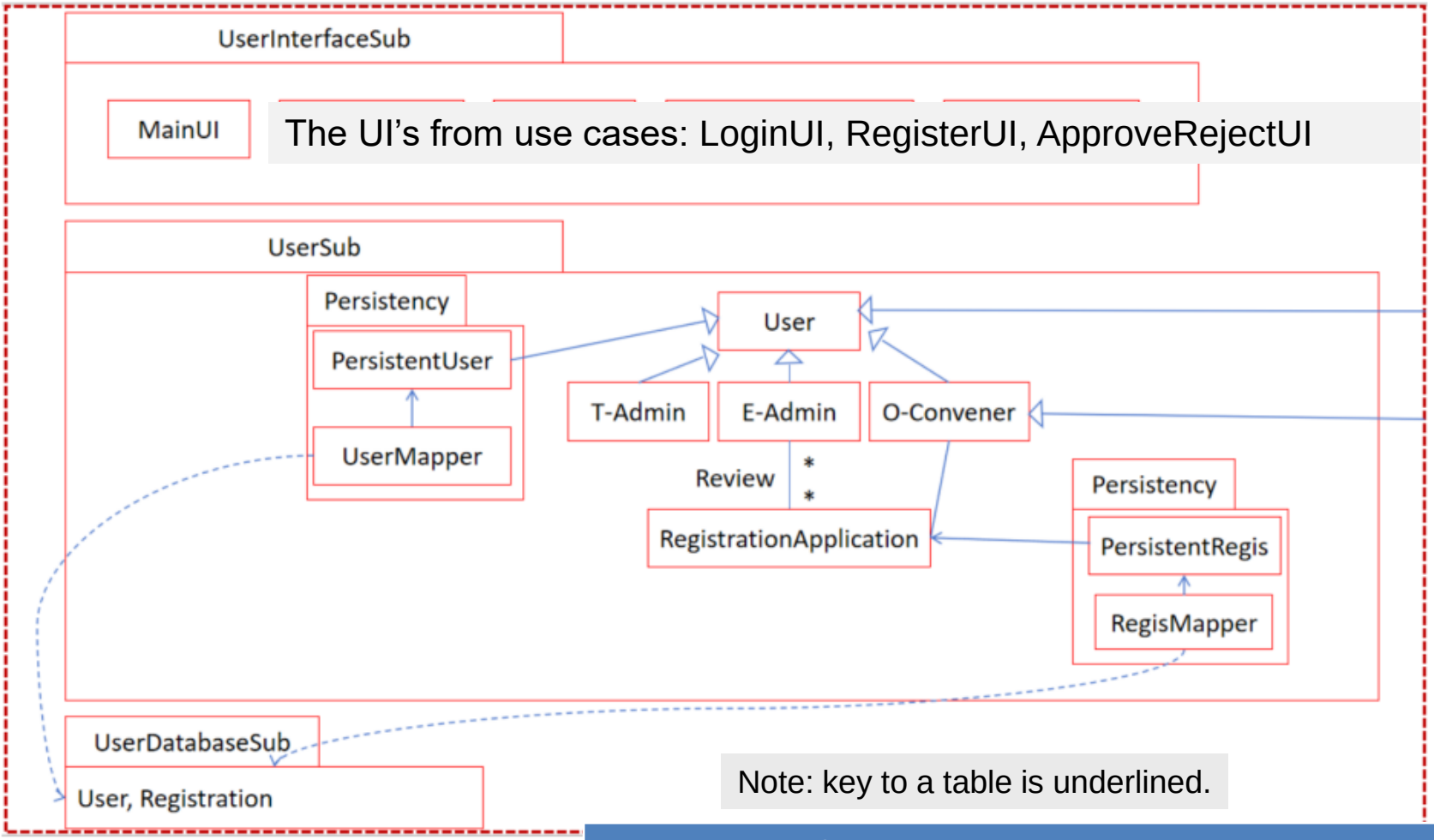
Example for struggling teams

- This solution is (mostly) workable, courtesy of Team B6. You need to
  - Fix the errors marked in the following slides.
  - Fix other unmarked errors
    - Associations are missing.
  - Redraw the diagrams using bigger text font size.

2.3 Overall System Architecture



3.1 User and Registration subsystem



Note: key to a table is underlined.

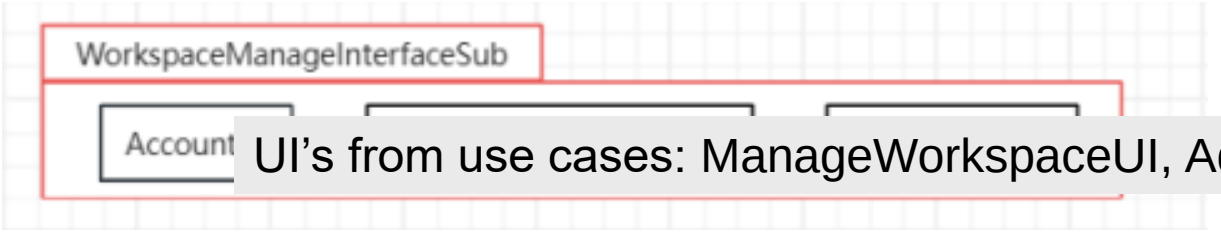
User Account

UserID	Role	EmailAdd	Password

Registration Application

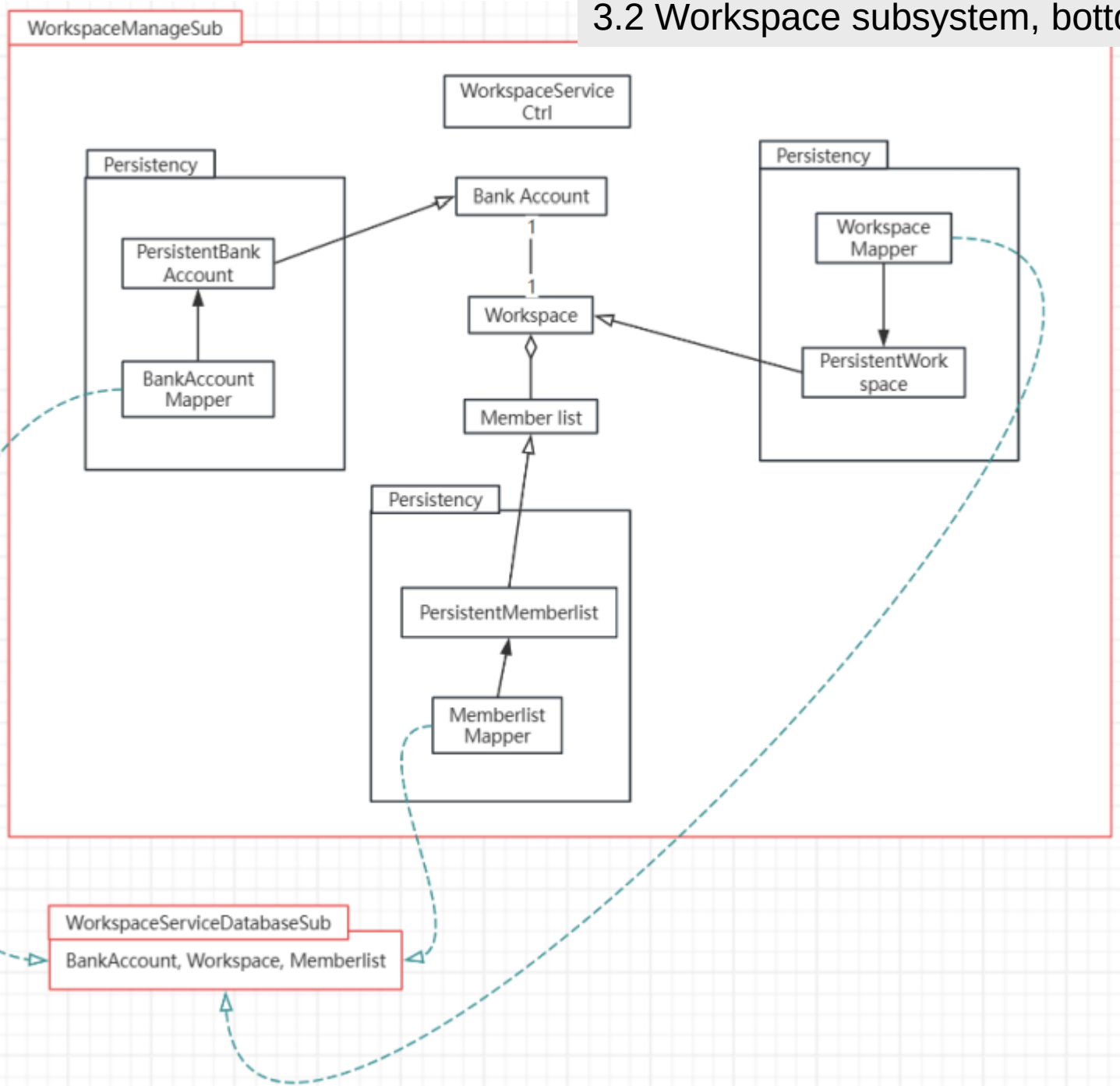
RegApp ID	O-convener UserID (for association)	E-Admin UserID (for association)	Status

3.2 Workspace subsystem, top layer for UI’s



UI’s from use cases: ManageWorkspaceUI, AddUserUI

3.2 Workspace subsystem, bottom 2 layers



3.2.2 Workspace subsystem tables

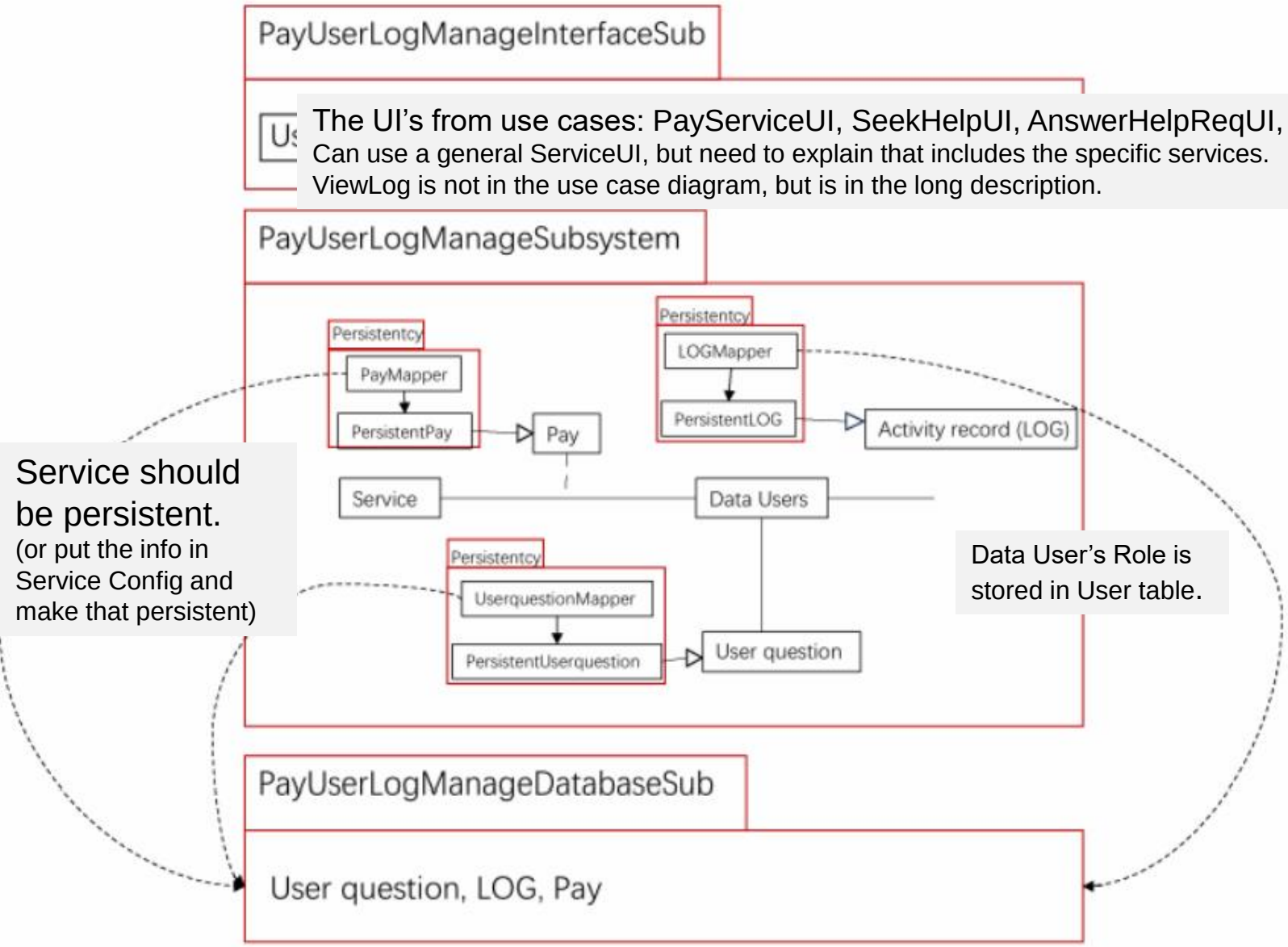
Workspace		
Organization	O-Convener UserID	

Note: key to a table is underlined.

Bank Account		
Bank Account ID	Workspace Organization (for association)	

Members List		
Members UserID	Workspace Organization (for association)	

3.3 Payment and User Activity subsystem



The UI's from use cases: PayServiceUI, SeekHelpUI, AnswerHelpReqUI, etc.
Can use a general ServiceUI, but need to explain that includes the specific services.
ViewLog is not in the use case diagram, but is in the long description.

Service should be persistent.
(or put the info in Service Config and make that persistent)

Data User's Role is stored in User table.

3.3.2 Payment and User Activities subsystem tables

Pay		
DataUser UserID (for association)	ServiceID (for association)	

Shows which data user pays for which service.

Service			
ServiceID	Service Type	Price	

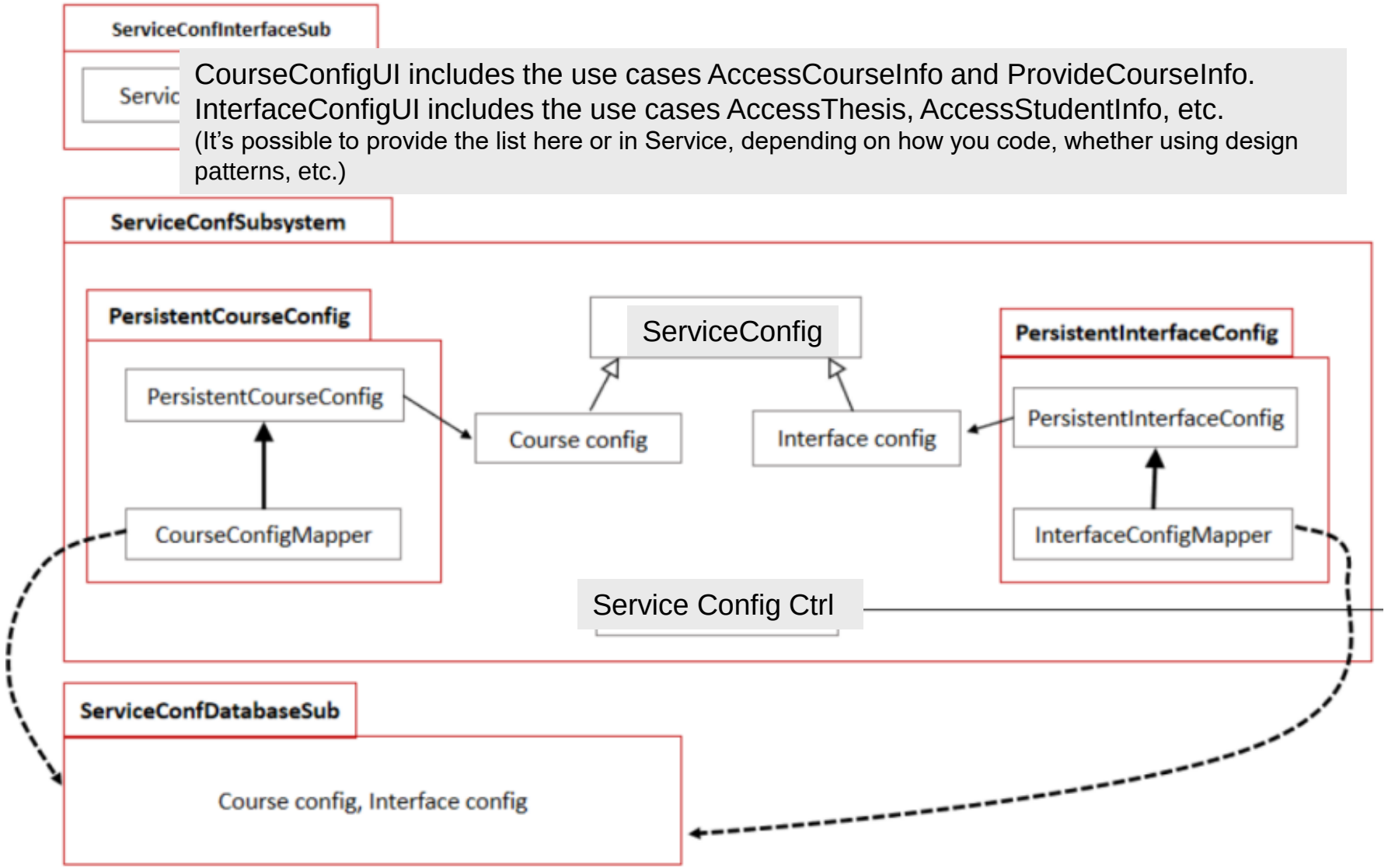
Service Type includes: AccessThesis, AccessStudentInfo, etc.

Question	
QuestionID	Question

Log Activity	
ActivityID	Activity

3.4 Service Config subsystem

CourseConfigUI includes the use cases AccessCourseInfo and ProvideCourseInfo.
InterfaceConfigUI includes the use cases AccessThesis, AccessStudentInfo, etc.
(It's possible to provide the list here or in Service, depending on how you code, whether using design patterns, etc.)



3.4.2 Service Config tables

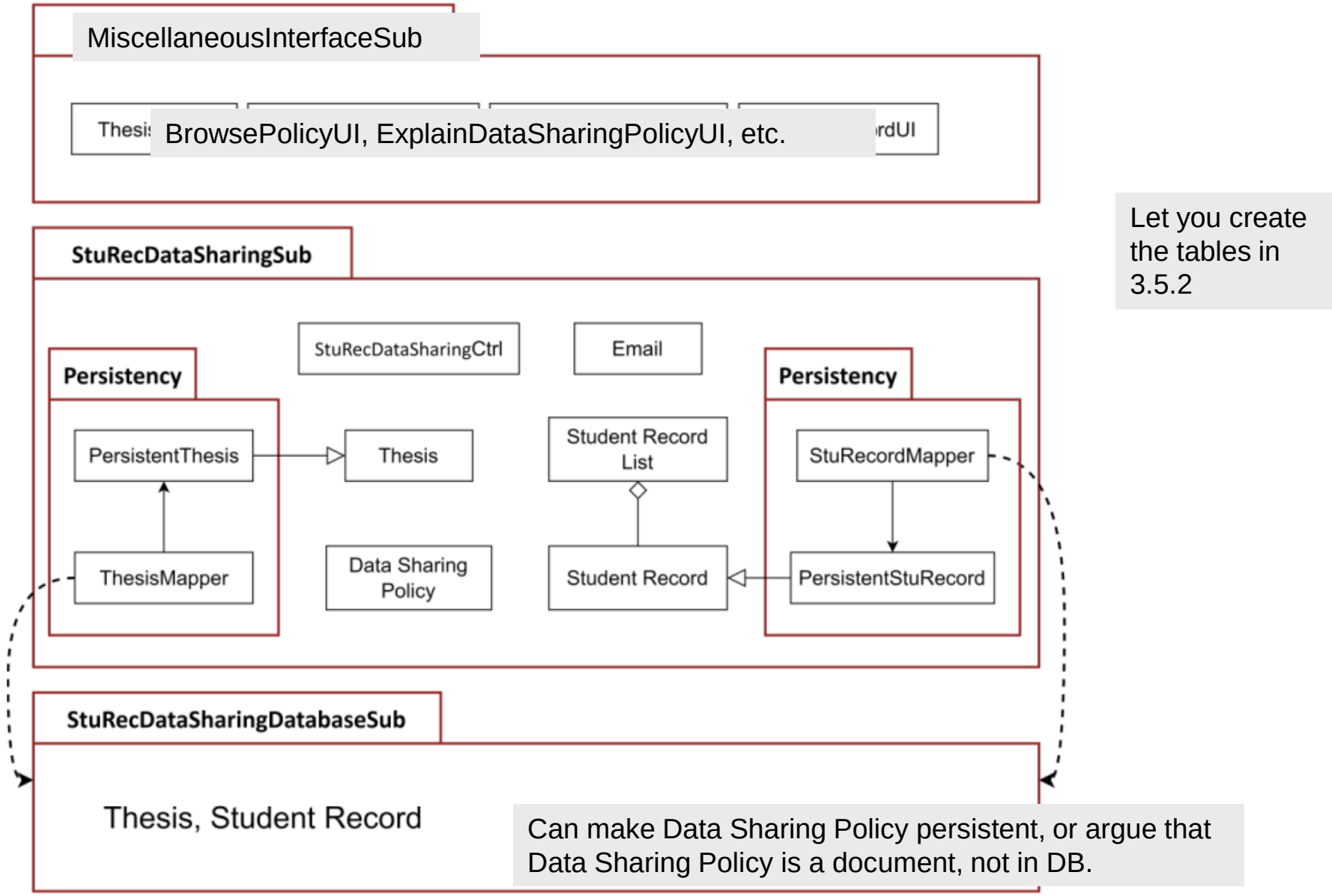
Course Config		
CConfigID	CName	

Interface Config		
IConfigID	IName	

For a disjoint inheritance, don't need a Service Config table.

3.5 Miscellaneous subsystem

For classes that are decoupled from other subsystems.



Let you create the tables in 3.5.2

Can make Data Sharing Policy persistent, or argue that Data Sharing Policy is a document, not in DB.

Issue 2: Detailed Design Template Explanation

- Teacher explains the detailed design template. The template is available on iSpace.
- Main points
  - Section 2.1: Whole class diagram. Same as the one you put in Section 2.1 in the Architecture Design Document
  - Section 3.1: Put the class diagram restructured from Section 2.1
    - The architecture diagram may subject to changes due to the restructuring; we will notify you if so.
    - Follow the restructuring guidelines in Lecture 11, around slide 19.

Issue 2: Detailed Design Template Explanation

- Section 3.1 (continue)
  - The restructured class diagram (with lots of details) occupies 2 pages. Put the classes from the miscellaneous subsystem in page 2 because it is decoupled from other subsystems.
  - Try to put the classes (from the other 4 subsystems) into page 1.
  - ~~If that doesn't fit, you can put the ServiceConf subsystem in page 2, because that only has 1 connection to the other 3 subsystems.~~
    - ~~See the next 2 slides.~~

Issue 2: Detailed Design Template Explanation (Cont.)

- Starting from 3.2, describe each class in the class diagram
  - Replace 3.#.1 section title with real class name to be described in this section .
  - Put the class notation with all its attributes and operations there before giving any explanation. The class should contain the finished interface design.
    - Following the first two items in Lecture 11, around slide 7.
    - Details of these two items are in the subsequent slides Lecture 11.
  - Explanation is optional, only needed when some new attributes or operations are added from the architecture design.

Issue 2: Detailed Design Template Explanation (Cont.)

- Pre-, post- conditions are usually required for each operation which has pre- and post- conditions in every class.
- However as a practice in this submission, you only need to **choose one** operation from one class to give pre and post condition.
  - I.e., give pre and post condition to **only ONE operation** in the whole document.

– Remember to update the Table of Content after finishing the description for all classes,

Example from last year

- Next 2 slides contain examples from last year, 3.1 and 3.2.1

- In this year's 3.1 restructured class diagram, classes (in Miscellaneous subsystem) that are decoupled from the other subsystems can be put into page 2 of 3.1.

3.1 Class diagram

The Figure 2 Updated Class Diagram is below:

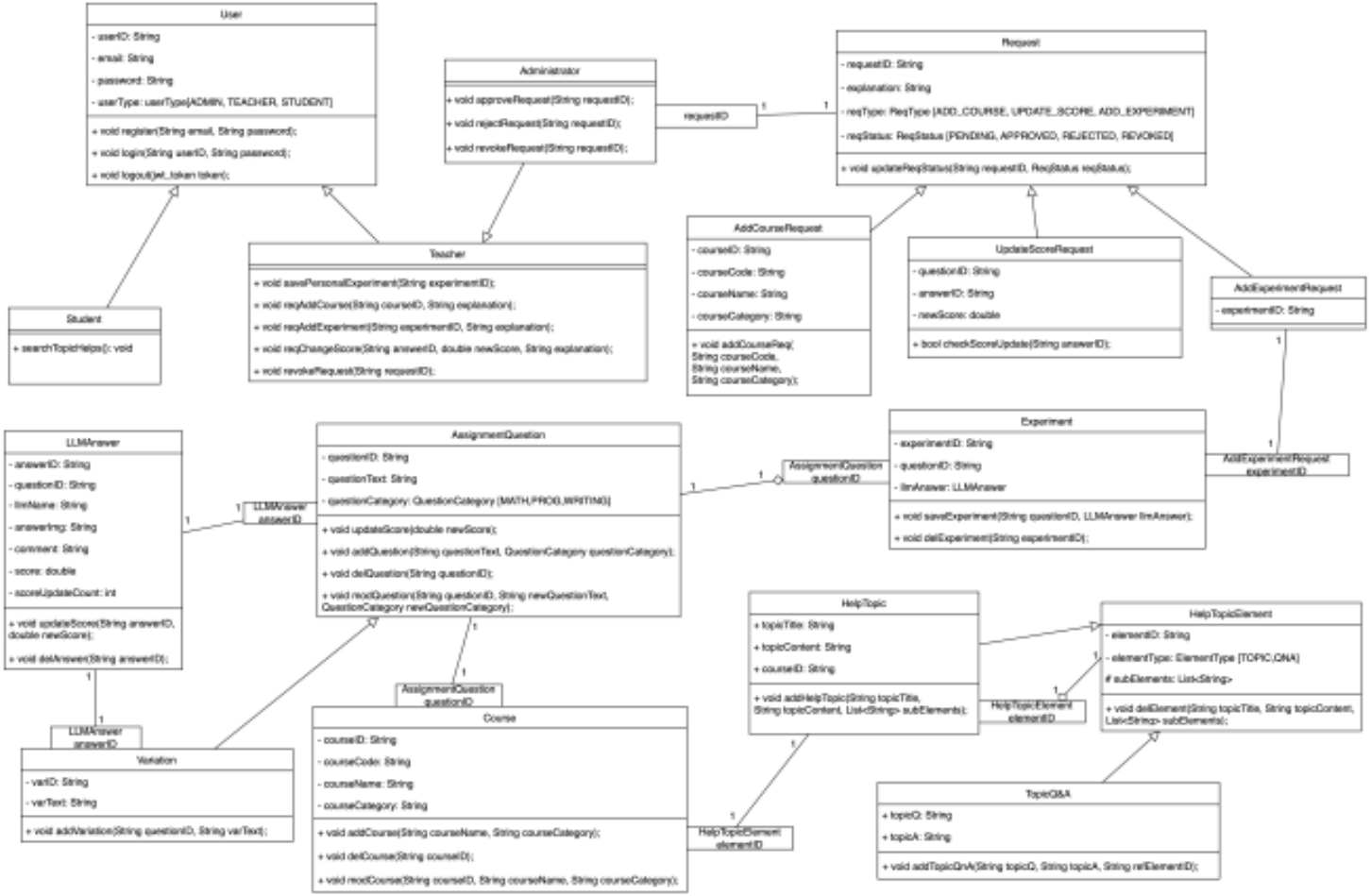
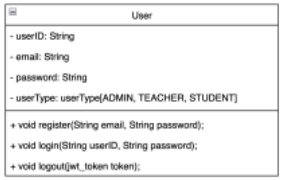


Figure 2 Updated Class Diagram for LLM-Homework



Explanations

- The **User** class has an additional attribute **`userType: UserType`** which presumably indicates the role of the user `(ADMIN, TEACHER, STUDENT)`.
- The operations **`register()`**, **`login()`**, and **`logout()`** have been modified to include token-based authentication as suggested by the parameter **`token`** added to these methods.

Constraints

1. For the **`register`** method:

   Pre-condition:

   - **`email`** and **`password`** must not be null.
   - **`email`** must be in a valid email format.
   - **`password`** must meet the security requirements (e.g., minimum length).

   Post-condition:

   - if the **`email`** is already registered, return `EMAIL_ALREADY_REGISTERED`.
   - if the **`password`** does not meet the security requirements, return `INVALID_PASSWORD`.
   - otherwise, a new User is created and return `REGISTRATION_SUCCESSFUL`.

2. For the **`login`** method:

   Pre-condition:

   - **`userID`** and **`password`** must not be null.
   - **`userID`** must correspond to an existing user.
   - **`password`** must correspond to the password of the user with the given **`userID`**.

   Post-condition:

   - if the **`userID`** does not exist, return `USER_NOT_FOUND`.
   - if the **`password`** is incorrect, return `INCORRECT_PASSWORD`.
   - otherwise, the user is logged in and return `LOGIN_SUCCESSFUL`.

3. For the **`logout`** method:

   Pre-condition:

   - **`jwt_token`** must not be null.

Student Task 1: Prepare Detailed Design Document

- Students finish the Detailed Design Document according to the template.
  - Follow guidelines in “Issue 2: Detailed Design Template Explanation” in previous slides.
  - Deadline: 2 pm Wed 9 April 2025
- Each team submits the Detailed Design document, ONE copy into iSpace and GitLab.
  - On the cover of the documents next to the student names, leaders (with the support of most of the team) indicate how much each member contributed: Full, Fair, Little or None.