Parson's Problems Appliance for Learning Management Systems (PPALMS)

Authors:

1. Al Yaqdhan Al Maawali
2. Fahia Tabassum
3. Mulki Yusuf
4. Ahmed Al Raisi

Group: 04

Content

Document Revision History

Date	Version	Description	Author

1 Introduction

1.1 Purpose

The purpose of this document is to show a detailed object-oriented design to the intended system. The document includes an overview of the design, the design structure, scenarios, traceability, and nonfunctional requirements.

1.2 System Overview

The PPALMS system is a system to be used by instructors in a class setting to assess students' knowledge on code writing exams. The system will exclude students from writing code from scratch and instead make them rearrange/match pieces of written code. The goal of the system is to give students problems that will prepare them better for the industry.

1.3 Design Objectives

The overall system design goal is to achieve the purpose of the system; generate and output problem questions. This is achieved through the different sections in this design document. The design covers 7 main functionalities, giving the user the ability to: 1) select source code 2) annotate lines 3) form and order line tuples 4) select question type 5) select LMS Target 6) reuse of problems 7) get statistics. Design also ensures that the non-functional requirements are met such as making sure code is thoroughly tested for safety and security, use little memory for performance, and leave code and class structure open for future extension and development. More about the non-functional requirements can be read in section (6).

1.4 References

Software Requirements Specification
use-cases

1.5 Definitions, Acronyms, and Abbreviations

Refer to the glossary in the Software Requirements Specification document.

2 Design Overview

2.1 Introduction

The design approach for the PPALMS system is an object-oriented approach. With this approach, there is an emphasis on the process of the system as a whole. Building a system like PPALMS requires a breakdown in planning, designing, building, and monitoring. A decentralized approach like this one allows for focus on each component of the system and its characteristics. This can be achieved through object-oriented design features such as objects, classes, inheritance, polymorphism, and encapsulation. These concepts provide a way for the components of the PPALMS system to function and interact with each other coherently.

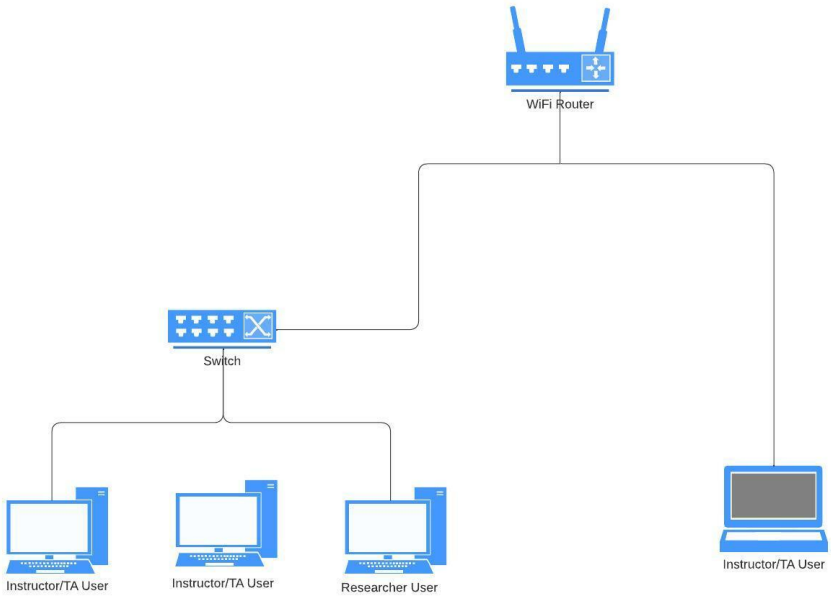
The proposed architecture for this system is a Layered method. This approach starts with the narrow Business layer at the top which consists of business rules and processes. The second Data layer contains data such as the source code that is provided to the system. The third Application layer contains external software used in this process such as the LMS targets. Another example of a custom application within this system is the log-in interface. Lastly, the wide, bottom Technology layer incorporates technology such as the user's desktop computer. This is the computer that will have PPALMS installed so it is critical to incorporate this technology in the system's architecture.

The relevant tools used to display design diagrams and models for this system are Lucidchart and Diagrams.net

2.2 Environment Overview

The PPALMS system will reside among the user's desktop applications on the latest versions of MacOs, Linux, or Windows. This system is installed in each user's computer and there are no interactions across the systems of each user. The scope of this system is that each user inputs the source code, manages questions, and exports the files to an LMS on their respective machines.

Network diagram:

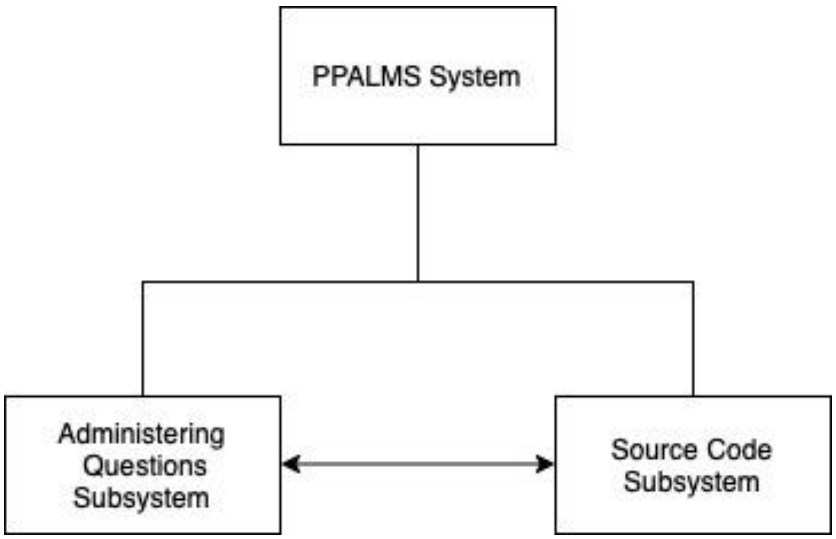


This diagram shows each computer or laptop connected to the WiFi router and working individually. The computers in which the system is running on can be connected to the same network or separate networks but the objective is that this is a system that is installed as a desktop application on each user's machine.

2.3 System Architecture

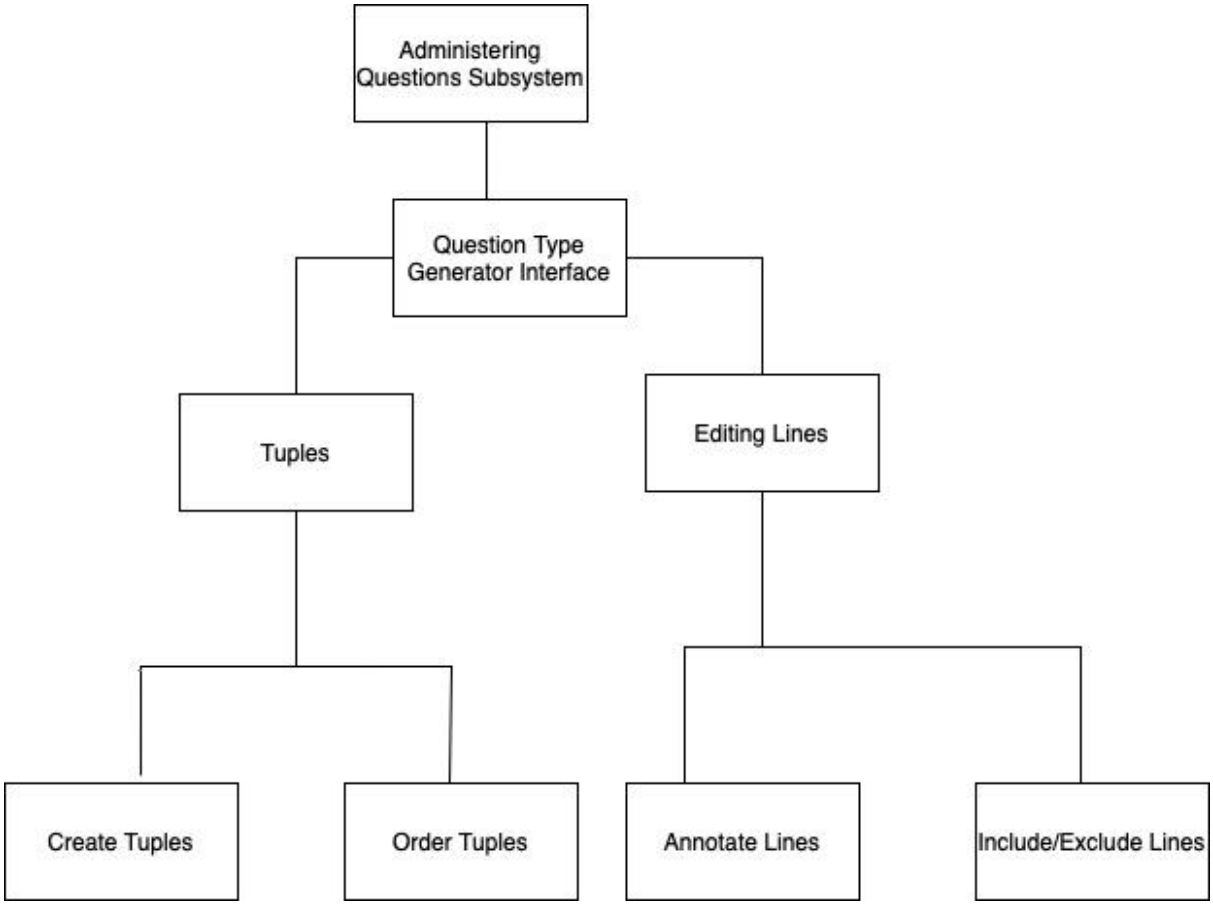
The system architecture has a central PPALMS system and is split into two major components. The two main components are the Administering Questions Subsystem and the Source Code Subsystem. The interaction between these components is bi-directional because the Source Code Subsystem deals with import of the source code. This source code is later used in the Administering Questions Subsystem to complete tasks like selecting and editing questions. Ultimately, the Source Code Subsystem exports the output file to a LMS target.

2.3.1 Top-level system structure of PPALMS



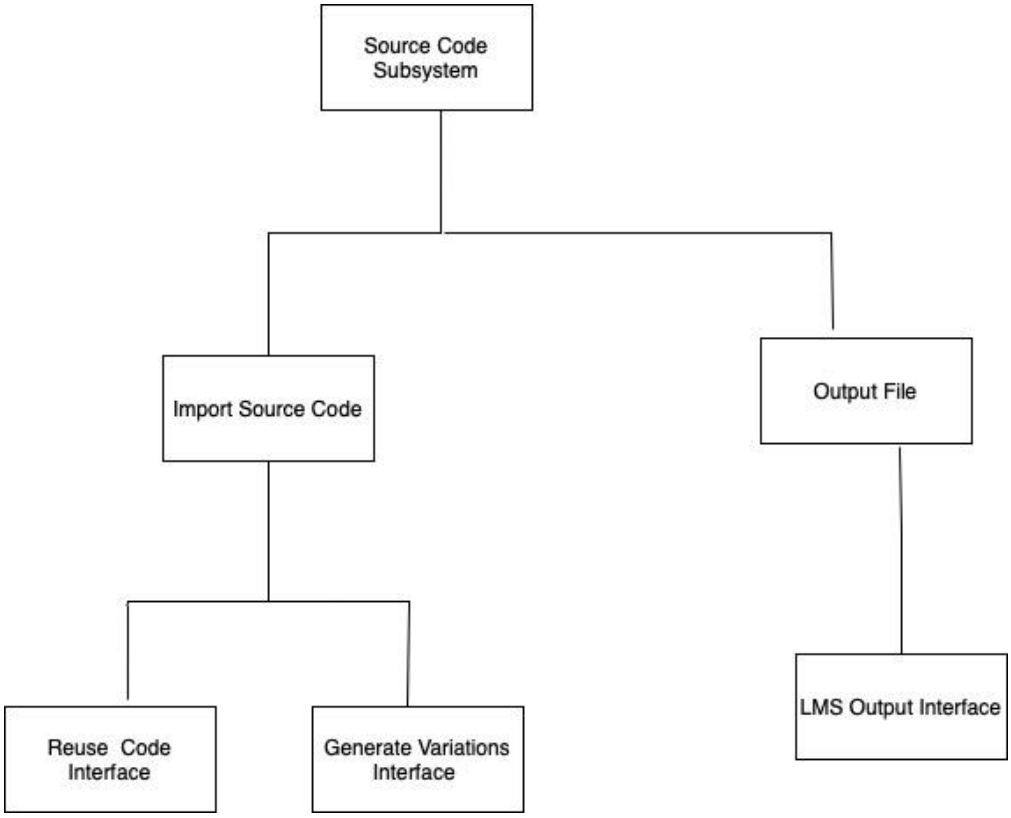
This diagram shows the top-level system as well as the two major subsystems which have a bi-directional relationship as stated in section 2.3.

2.3.2 Administering Questions Sub-system



This block diagram illustrates the Administering Questions Subsystem. This subsystem specifically manages the tasks that are related to the questions. In this subsystem, the question type is selected and then there are branches that consist of the Tuples and Editing Lines components. In Tuples, there are the tasks of creating and ordering the tuples. The Editing Lines component, however, handles annotating, including, and excluding lines. These cover the requirements that relate to creating and handling the questions from the source code. This structure shows the hierarchy of these components and how they relate to each other.

2.3.3 Source Code Sub-system



This block diagram illustrates the Source Code Subsystem. This subsystem specifically manages tasks related to the source code. It has a bi-directional and back and forth interaction with the Administering Questions Subsystem since the source code is used by the Administering Questions Subsystem and then returned to the Source Code Subsystem. In this subsystem, there are features to reuse the source code and to generate multiple variations of the questions imported from the other subsystem. These are two key features from the system requirements. The second branch of this subsystem handles creating the output file and an interface to select a LMS target to upload the output file to. This subsystem also incorporates hierarchical categories as well as depicting how the components are related to each other.

2.4 Constraints and Assumptions

1. Constraint: LMS Security Constraint

   This constraint is detailed in the requirements to be a security constraint for when the system should not communicate with the selected LMS due to security concerns. In this case, the system should not upload the output file to the LMS. The only thing the system should do is export the file to the user's computer. This constraint is addressed by the

design in place. “Output file” is a component under the Source Code Sub-system that is responsible for creating the output file. The only component that branches out from this is the LMS Output Interface. This set-up in the block diagram shows that this is separate from the other components in this sub-system. This security constraint should not cause problems in the system as a whole. Instead if there is a security constraint, only that component is affected and the user still gets an output file saved in their computer.

2. Constraint: Importing source code is in a separate subsystem than the selecting question type component

   This is a constraint that is imposed by the design choice for this system. This constraint arises due to the “Import Source Code” component not belonging to the same subsystem as the “Question Type Generator Interface” component. The reason for this is because Import Source Code shares the commonality of interacting with the source code like many of the other components in that subsystem. The tradeoff for this is that there is a resolution by creating a bi-directional interaction between the two top-level subsystems. This type of interaction makes it possible for the two components to cooperate even if they are in two different subsystems.

3. Constraint: Output file type must match the target LMS

   This is a constraint imposed in the requirements and arises when the system is interacting with an external software. This is an LMS target such as Canvas. This constraint is addressed in the system design since there is a LMS Output Interface under the Output File component. This interface is responsible for selecting the desired LMS target and generating an output file that is compatible with that LMS target.

4. Assumption: An assumption for this system is that users have adequate storage and memory to run the PPALMS system on their desktop as well as to store output files generated by the system. This is necessary to ensure that the PPALMS system runs smoothly without any crashes.

3 Interfaces and Data Stores

This section describes the interfaces into and out of the system as well as the data stores you will be including in your system.

3.1 System Interfaces

3.1.1 Source Code Interface

This interface allows the user to import their desired source code to be included within the questions generated by PPALMs, this interface will only allow the user to proceed if the file is within the accepted format and within the correct size limit (1mb), otherwise an error message will be displayed.The source code will not have to be used, the user will have an option to include or exclude source code based on their desires.

3.1.2 Question Type Generator Interface

This interface allows the user to select the type of questions they would like to generate questions from. There are multiple options of questions in the system that the user can choose from in the interface, these question types are multiple choice, matching, ordering, completing the code questions, and predicting the output from different pieces of code. The user is granted the option to generate questions with the source code or without the imported source code.

3.1.3 LMS Output Interface

This interface allows the user to select the desired LMS target and pick a type of output file, as a result the LMS output interface will create an output file and is synched with the output file.

3.1.4 Statistics Interface

This interface will allow users to receive statistics based on the question types, and how the students responded to them.This interface will mainly be used by researchers using the PPALMS system.

3.1.5 Login Interface

This interface acts as the first graphical interface that allows the user to input their username and password. This allows the system to understand the type of user accessing the system, and grants the user the permissions based on their account type. This also allows the user to access previous questions generated from previously uploaded source code, without the need of uploading the source code again.

3.1.6 Annotate Lines Interface

This is a key part of the question type interface, where after the user imports a source code file, they are presented with an interface that allows them to annotate lines which they would like to

include, in addition to indicating which lines will not be included in the questions that are generated. This is a key part of the question type generator subsystem.

3.1.7 Form and order line tuples Interface

This interface allows the user to pick out certain lines of code from the source code to either exclude or include in the question generation step. This interface allows the user to pick which lines they want to keep together, and which other lines can be rearranged.

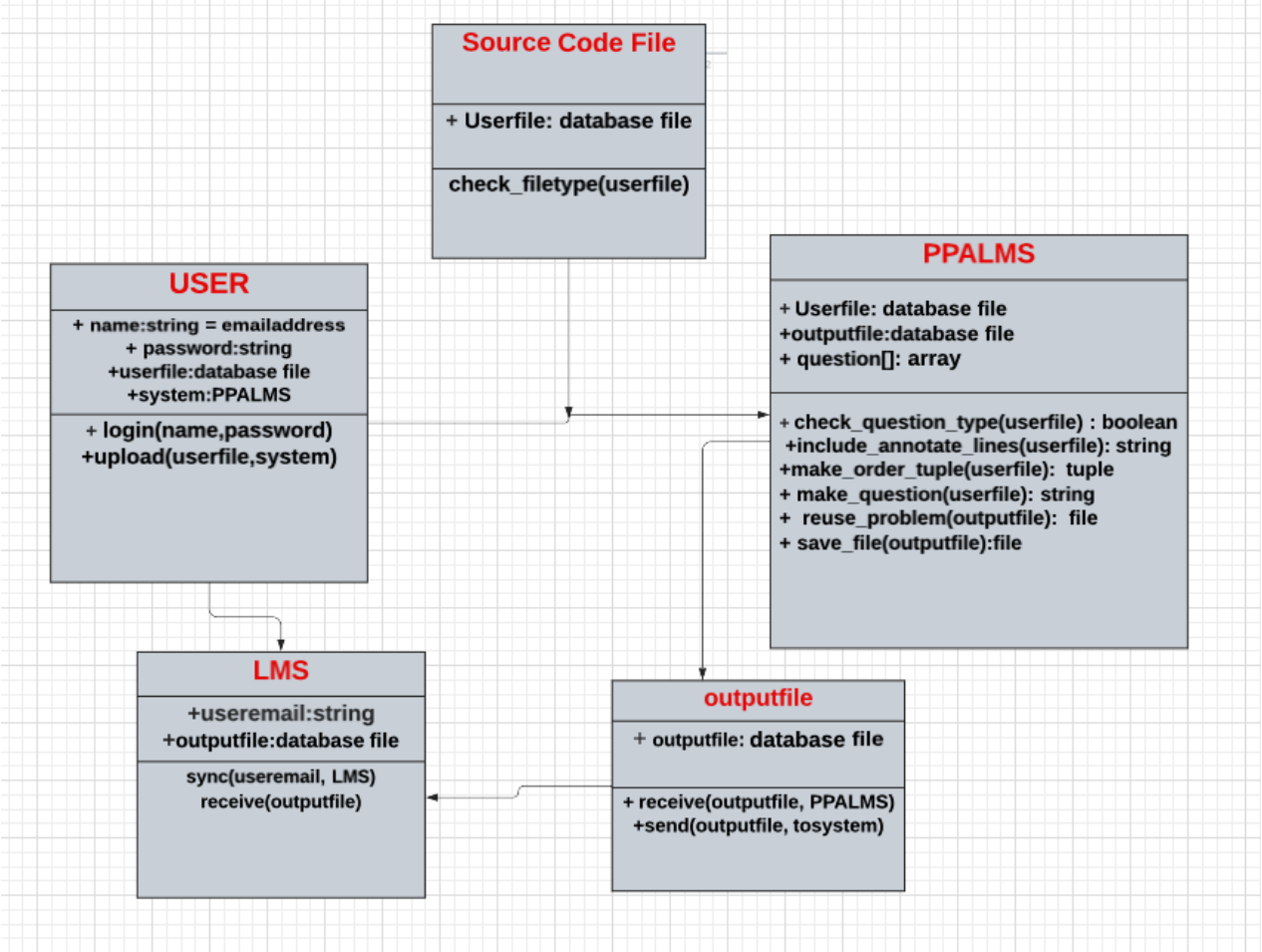
3.2 Data Stores

There are two contexts in which the PPALMS system will use data stores, the main one is saving the problems that have been generated to allow for reuse in future problem generations. This was a main requirement for the client in the elicitation session. This eliminates the need to import the same source code for generating questions that have been previously generated.
In addition to that there also will be data stores to store the statistics of the questions generated, this feature will mainly be used by researchers, to determine how different questions types do in certain contexts.

4 Structural Design

4.1 Class Diagram

The classes with the main methods and attributes have been provided here in the diagram below:



4.2 **Class Descriptions**

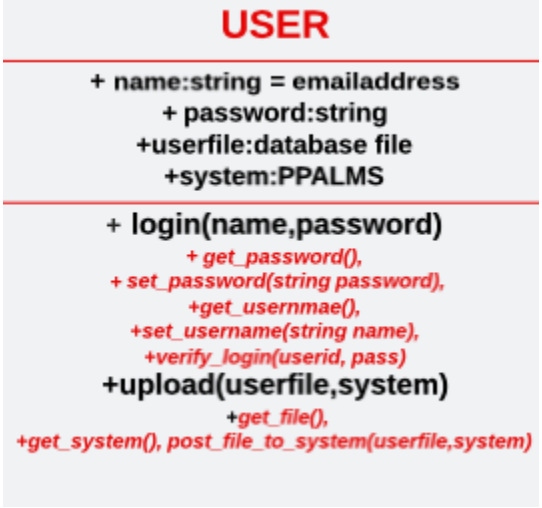
The main classes in this system will be the following classes:

User class 2. LMS class 3. PPALMS class 4. Output File class 5. Source code file class

Every class has its specific attributes and methods which will be used to process different functionalities of the PPALMS system. More details of different classes and their functionalities have been provided in the section 4.3 of this document. Briefly, a short description of each class has been provided below:

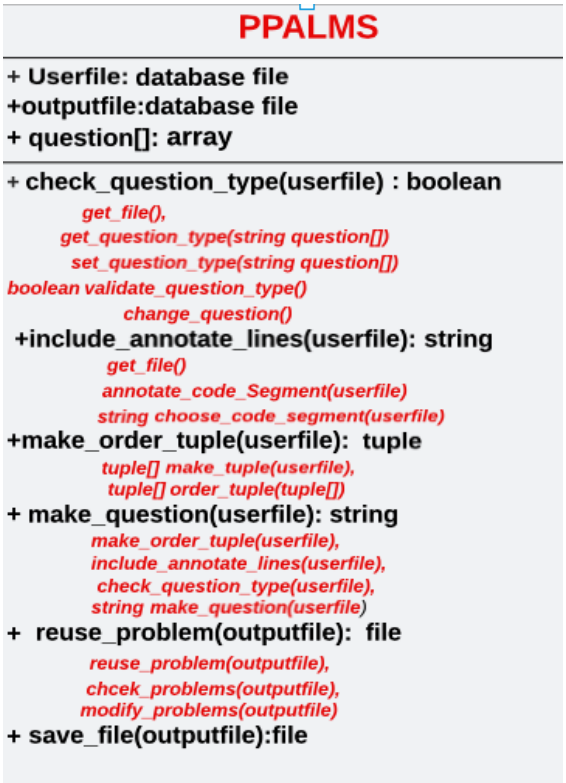
The user class:

The additional methods have been marked red and bold in the third bar of the class.



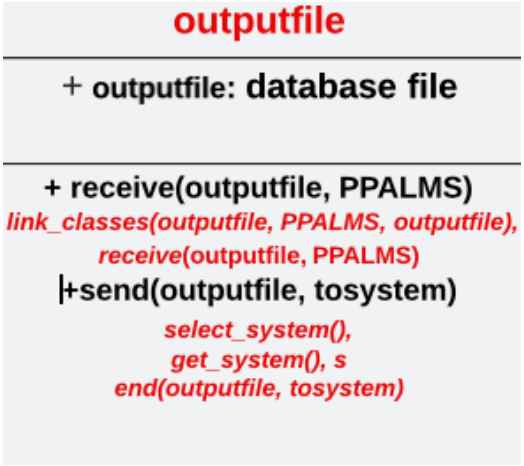
The PPALMS class:

The additional methods have been marked red and bold in the third bar of the class.



The outputfile class:

The additional methods have been marked red and bold in the third bar of the class.



4.3 Classes in the PPALMS

4.3.1 Class: User

- **Purpose**: the user class uploads the source code file in the system. The user has to login using his username and password and he has to synchronize his school LMS to the system
- **Constraints**: 1. Should have a valid user who should be an official school researcher/TA/professor 2. Should have a school credential or office email to verify the authenticity of the user's information.
- **Persistent**: No(created at system initialization from other available data)

4.3.1.1 Attribute description:

1. **Attribute**: *Password*

   Type: an array of strings and integers

   Description: This attribute is a combination of strings and integers which will be inserted by the user. This attribute will help the user to login to the system. The function login() will use this attribute so that the user can login with his/her username and password.

   Constraints: **1**.The length of the password has to be 6 characters long, containing lower case, uppercase, numbers and symbol characters 2. The Password Attempts will be limited to 3 times and the Lock-Out Policy will be implemented. 3. Change Passwords Every 90 Days 4. Password History will be enforced 5. Email should be linked for future password recovery

2. **Attribute**: *Name*

   Type: string

Description: This is the username which will be used in the login function to login to the system. Every user will have an unique username which will be used for their individual login into the system.

Constraints: 1. Restriction if the username is not unique 2 the first letter of the username should be alphabet in lowercase

3. **Attribute**: Userfile

Type: database file

Description: This is the file which will be uploaded by the user to the system.

Constraints: 1. The file should have a specific limit of length so that the user can easily upload the file. Large files will be restricted to get uploaded in the system.

2. The file should be in .pdf or .txt format

4. **Attribute**: System

Type: PPALMS system

Description: This is the system which will be used by the users to make questions from their uploaded file.

Constraints: The PPALMS system should be properly linked so that the users can upload the file to the correct system.

4.2.1.2 Method Description:

1. **Method** : *login(name,password)*

Return Type: will give the user an access to the system

Parameters: name and password

Return Value: system login

Pre-condition: The user must come up with an unique and valid username and should have a password following all the constraints.

Post-condition: The user can save the password and username. Additionally, to user in the future, the user must remember the username and email for recovery.

Attributes read/used: name and password

Methods called: *get_password(), set_password(string password), get_usernmae(), set_username(string name), verify_login(userid, pass)*

Processing logic: The getter and setter methods will set up the password and username for the user which will be used in this method to login the system. The verification of the login will be done by the verify_login function in this class.

Test Case 1: Call login() with the name and password. The expected output: will let the user login to the system

2. **Method** : *upload(userfile,system)*
   Return Type:will let the user upload his file to PPALMS
   Parameters: userfile and system
   Return Value: uploaded file in the system
   Pre-condition: The user should have a valid userfile and should be able to access the system. The userfile
   Post-condition: The valid file will be uploaded to the PPALMS system
   Attributes read/used: userfile and system
   Methods called: *get_file(), get_system(), post_file_to_system(userfile,system)*

Processing logic: The correct file will be chosen by the get_file() function in this class, and the correct system will be linked with the get_system() function. post_file_to_system(userfile,system) function will help the user to upload the file to the system where the userfile and system will be accessed by the user from the get_file() and get_system() functions.

Test Case 2: Select the file by calling get_file() from the local device 2. Link the system by calling the get_system() function
Test Case 3: Post the file in the system by calling post_file_to_system(userfile,system) function

4.3.2 Class: PPALMS

- **Purpose:** To modify the source code and make an output file
- **Constraints**: None
- **Persistent**:No (created at system initialization from other available data)

4.2.2.1 Attribute Description:

1. **Attribute**: Userfile
   Type: database file
   Description: This is the file which will be uploaded by the user to the system.
   Constraints: 1. The file should have a specific limit of length so that the user can easily upload the file. Large files will be restricted to get uploaded in the system.
   2. The file should be in .pdf or .txt format

2. **Attribute**: outputfile

   Type: database file

   Description: This is the file which will be uploaded by the user to the system.

   Constraints: 1. The file should have a specific limit of length so that the user can easily upload the file. Large files will be restricted to get uploaded in the system. 2. The file should be in .pdf or .txt format

3. **Attribute**: question[]

   Type: array of questions which will contain different types of questions

   Description: This attribute is to classify different types of questions in the system

   Constraints: 1. The question should be made from the types of questions mentioned in Question array

4.2.2.2 Method Description:

1. **Method** : check_question_type(userfile)
   Return Type: checks the type of the question user wants to make
   Parameters: userfile
   Return Value: boolean
   Pre-condition: 1. The user should have chosen the correct type of the file 2. The the user should know what type of question he/she wants to ask.
   Post-condition: The user can validate/change the question type.
   Attributes read/used: userfile
   Methods called: *get_file(),*
   get_question_type(string question[])
   set_question_type(string question[])
   boolean validate_question_type()
   change_question()

Processing logic: The user will be able to upload the file. Then the user will be able to choose the correct type of the question which he/she wants to make for the students. The user will be given the option to change/check/validate the question type before posting to his school LMS.

Test Case 4: The user will upload the file calling get_file() function
Test Case 5: The user will choose the question type from calling the get_question_type() function which will choose the type of question the user wants to make from the question[] array
Test Case 6: The set_question_type(question) will set the question type user wants to make

Test Case 7: The validate_question_type wil validate the valid question types by returning the boolean value(True if that's the correct type of question; otherwise false)
5. If the user wants to change the question type then he/she can call the change_question() function which will use the getter and setter to set the new question type

2. **Method** : include_annotate_lines(userfile)
   Return Type: string
   Parameters: userfile
   Return Value: code segment from the userfile
   Pre-condition: 1. Valid userfile 2. User's choice on the length of the code segment
   Post-condition: N/A
   Attributes read/used: userfile
   Methods called: *get_file()*
   annotate_code_Segment(userfile)
   string choose_code_segment(userfile)

Processing logic: The user will be able to annotate different code segments from the userfile which he/she can choose in future in the make_question(userfile) function of this class. The user will be able to choose the code segment from the file he/she has uploaded from his/her local device
Test Case: 8. The user will upload the file calling get_file() function
Test Case 9: The user will annotate the code segment from the file by calling the annotate_code_segment(userfile) method.
Test Case 10:.The user will choose the code segment he/she has annotated before by calling the choose_code_segment(userfile) function.

3. **Method :** make_order_tuple(userfile)
   Return Type: tuple
   Parameters: userfile
   Return Value: makes and order tuple in the code segments from the userfile
   Pre-condition: 1. The user has to choose the valid form of the file. 2.The user should know which parts of the file he/she can want to make a tuple of.
   Post-condition: The expected output from this function is a valid tuple which can be used further to make the questions.
   Attributes read/used: userfile
   Methods called: *tuple[] make_tuple(userfile), tuple[] order_tuple(tuple[])*

Processing logic: The user will be able to make and order tuples from the userfile which he/she can use in future in the make_question(userfile) function.

Test Case 11: Call the make_tuple(userfile) function which will return an array of tuples from the userfile
Test Case 1**2**: Call the order_tuple(tuples[]) to order the tuples in the correct order.

4. **Method :** make_question(userfile)
   Return Type: string
   Parameters: userfile
   Return Value: makes the question from the code segments
   Pre-condition: 1. The user should have used the make_order_tuple(userfile), include_annotate_lines(userfile) functions beforehand. **2.** The user must have used the check_question_type(userfile) function so that the correct type of question will be made.

   Post-condition: Valid question should be made and it should meet the user's satisfaction.
   Attributes read/used: userfile()
   Methods called: *make_order_tuple(userfile), include_annotate_lines(userfile), check_question_type(userfile), string make_question(userfile)*

Processing logic: The user will be able to make questions by using the other functions of the class.
Test Case: 13. Call the make_question(userfile) function which will return a string as a question.

5. **Method** : reuse_problem(outputfile)
   Return Type: file
   Parameters: outputfile
   Return Value: reuses the problems
   Pre-condition: The user should already have existing problems
   Post-condition: Should return the output file which will be using the existing problems
   Attributes read/used: outputfile
   Method called:
   reuse_problem(outputfile), chcek_problems(outputfile),modify_problems(outputfile)

Processing logic: The user should be able to reuse the existing questions to make a different new problem. The user should be able to modify the existing problems.
Test Case 14: Call reuse_problem(outputfile) to reuse the existing questions.

Test Case 15: Call check_problems(outputfile) to check if the questions already exist or not

Test Case 16: Call modify_problmes(outputfile) to modify the existing questions

6. **Method** : save_file(outputfile)
   Return Type: file
   Parameters: outputfile
   Return Value: saves the file in the user's local device directory
   Pre-condition: The user should have an output file, The user should select the directory in his/her local device.
   Post-condition: The user should save the file as a new file and in .pdf format
   Attributes read/used: outputfile
   Methods called: *save_file(outputfile)*

Processing logic: The user will be able to save the output file in his local directory
Test Case 17: Call the save_file(outputfile) to save the file in local device

4.3.3. Class:LMS

- Purpose:
- Constraints:
- Persistent: No(created at system initialization from other available data)

4.3.3.1: Attribute description:

1. **Attribute**: useremail
   Type: string
   Description: This is the email address which will be used in the login function to login to the PPALMS. Every user will have an unique email which will be used for their individual login into the system. Additionally, this same email will be used to synchronize the LMS and PPALMS.
   Constraints: 1. Restriction if the email address doesn't end with .edu or school address.

2. **Attribute**: outputfile
   Type: database file
   Description: This is the file which will be uploaded by the user to the system.

Constraints: 1. The file should have a specific limit of length so that the user can easily upload the file. Large files will be restricted to get uploaded in the system. **2.** The file should be in .pdf or .txt format

4.3.3.2. Method Description:

1. **Method :** sync(useremail, LMS)
Return Type: will synch user's LMS to the PPALMS
Parameters: useremail
Return Value: will synch user's LMS to the PPALMS
Pre-condition: 1.User should have an official school email address. 1.User should have a valid LMS which can be synced with PPALMS through the same email address.
Post-condition: The user should remember the email address for future use
Attributes read/used: useremail, LMS
Methods called: sync(useremail,LMS)

Processing logic: The user can easily sync their LMS to PPALMS by their user email.
Test Case 18: Call the sync(useremail,LMS) method to sync the LMS to PPALMS

2. **Method** : receive(outputfile)
Return Type: file
Parameters: outputfile
Return Value: receive the file from the PPALMS
Pre-condition: The user has to select the correct file from the PPALMS
Post-condition: N/A
Attributes read/used: outputfile
Methods called: *receive(outputfile)*

Processing logic: This class will receive the output file from the PPALMS which can be linked to the LMS. There is a save_file function in the PPALMS. So, the user can already save the output file or the user can use this class to receive the output file from the PPALMS system so that he can upload it to LMS in future.

Test Case: 19: Call the receive(outputfile, PPALMS) to receive the output file from the PPALMS.

4.3.4 Class: outputfile

- Purpose: To receive file from PPALMS and to send that file to LMS
- Constraints: None

- Persistent: No(created at system initialization from other available data)

4.3.4.1 Attribute description:

1. **Attribute**: outputfile
   Type: database file
   Description: This is the file which will be uploaded by the user to the system.
   Constraints: 1. The file should have a specific limit of length so that the user can easily upload the file. Large files will be restricted to get uploaded in the system. 2. The file should be in .pdf or .txt format

4.3.4.2 Method Description:

1. **Method** : receive(outputfile, PPALMS)
   Return Type: file
   Parameters: userfile and PPALMS
   Return Value: receive the file from the PPALMS
   Pre-condition: The user has to select the correct file from the PPALMS
   Post-condition: N/A
   Attributes read/used: outputfile, PPALMS
   Methods called: *link_classes(outputfile, PPALMS, outputfile),*
   receive(outputfile, PPALMS)

Processing logic: This class will receive the output file from the PPALMS which can be saved or linked to the future LMS. There is a save_file function in the PPALMS. So, the user can already save the output file or the user can use this class to receive the output file from the PPALMS system so that he can upload it to LMS in future.

Test Case 19: Call the receive(outputfile, PPALMS) to receive the output file from the PPALMS.

2. **Method** : send(outputfile, tosystem)
   Return Type: send that file to the LMS
   Parameters: outputfile and tosystem
   Return Value: send that file to the LMS
   Pre-condition: 1. The user has to create the output file. 2. the user must know in which system he/she wants to send the file.
   Post-condition: Successful sending to the LMS of the user.
   Attributes read/used: outputfile and tosystem
   Methods called: *select_system(), get_system(), send(outputfile, tosystem)*

Processing logic: The user will send the file from PPALMS to his/her preferred LMS by choosing the valid LMS, file and by linking the PPALMS and LMS

Test Case 20: Call the *select_system()* to select the system user wants to send the file to. 2. Call *get_system()* to verify the system 3. Call *send(outputfile, tosystem)* to send the file from the PPALMS to the user's LMS

4.3.5.Class: Source Code File

- Purpose: provide the user with specific type of file
- Constraints: 1. The source code file should be of either .pdf or .txt format
  2. The size of the file should be 100kb -1000kb
- Persistent: No(created at system initialization from other available data)

4.3.5.1 Attribute description:

Attribute : userfile

Type: database file

Description: This is the file which will be uploaded by the user to the system.

Constraints: 1. The source code file should be of either .pdf or .txt format
2. The size of the file should be 100kb -1000kb

4.3.5.2 Method Description

Method : check_filetype(userfile)

Return Type: check the type of the file. If it's not .txt/.pdf then will print error message

Parameters: userfile

Return Value: check the type of the file. If it's not .txt/.pdf then will print error message.

Pre-condition: The user should know about the restriction on the type of the file.

Post-condition: Valid type of file should be uploaded

Attributes read/used: userfile

Methods called: *boolean check_filetype(userfile)*

Processing logic: The main function of this class is to check the correct format of the file. The system only takes the .pdf/.txt formats of the file in it. This class will work in between

the user class and PPALMS class where it will help the user to select the correct format of file in the PPALMS system.

Test Case 21: Call check_filetype(userfile) to check the format of the file.

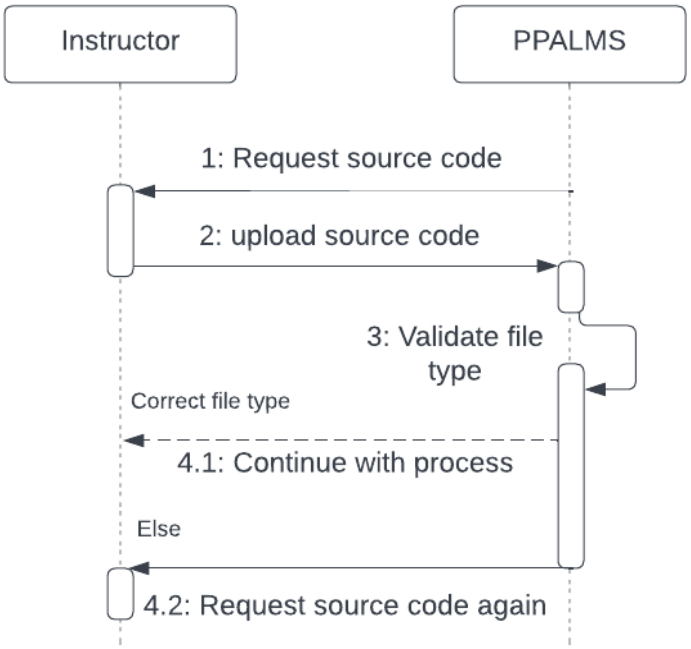
5 Dynamic Model

This section shows PPALMS behavior in different events. This will be done by showing the various scenarios the system goes through and its relative sequence diagrams.

5.1 Scenarios

5.1.1 Select source code

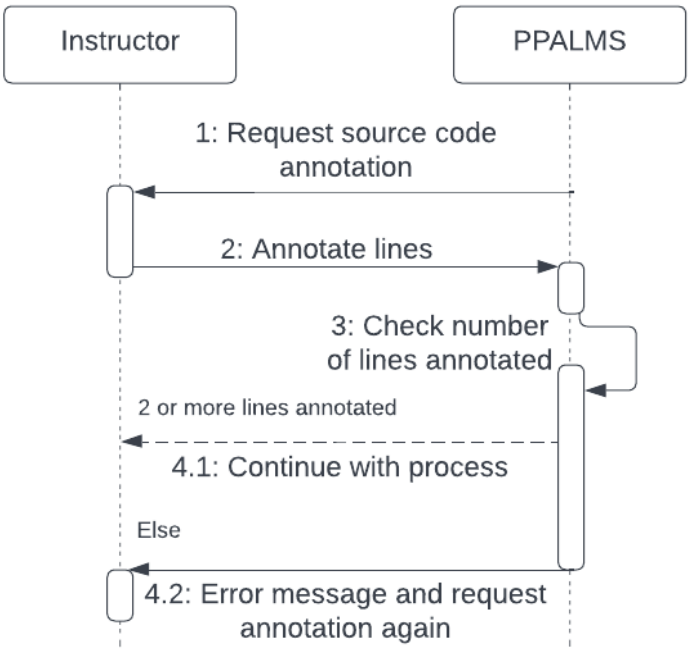
This is the first scenario that occurs in the system. The user (instructor) selects the source code to be uploaded to the system by uploading a source file from their device to the system. If the user didn't upload a .txt file then an error message is shown and is given the option to upload another file.



5.1.2 Annotate lines

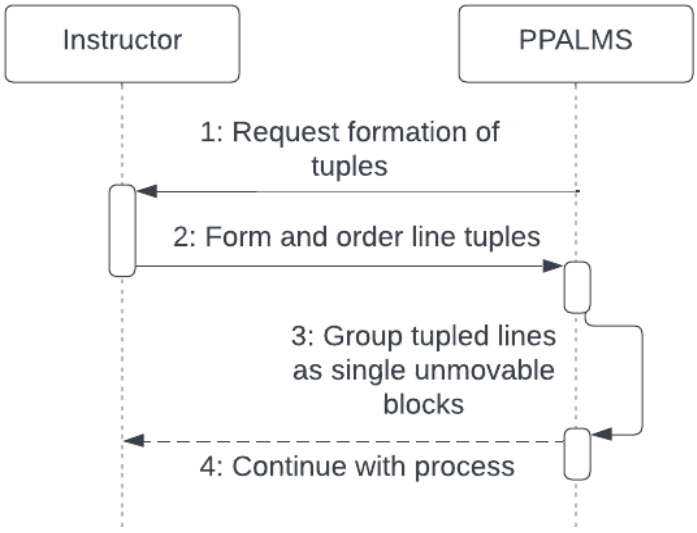
After the user uploads a source file to the system, they need to annotate lines for inclusion and leave lines out for exclusion. If the user annotates at least two lines, then the

system will proceed with the process. If the user annotated one line or none, then an error message is displayed and the user is asked to annotate lines again.



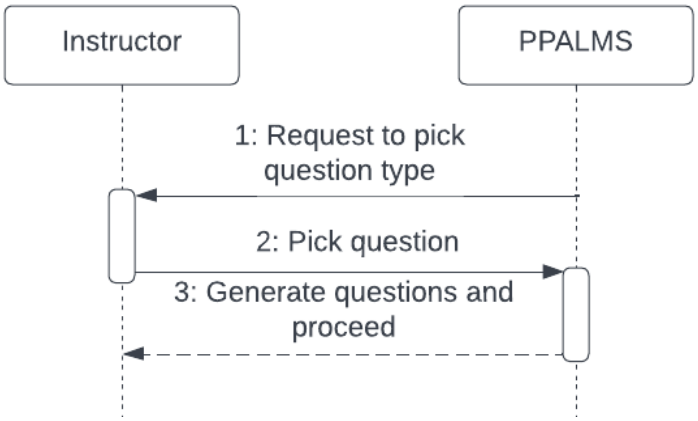
5.1.3 Form and order line tuples

The user will form and order line tuples. User should be able to group sets of lines together so that lines that are grouped together cannot be rearranged. If user didn't group lines, then the system will proceed without any line tuples.



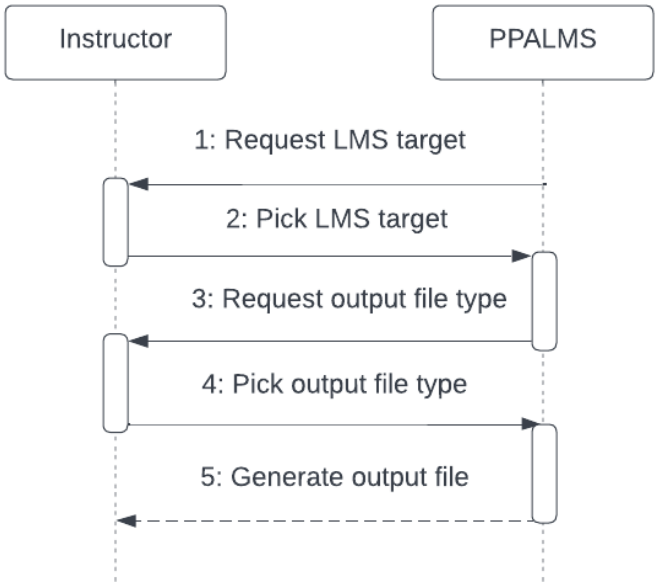
5.1.4 Select question type

This is the simplest scenario that would occur. The system will request for the user to pick a question type and wait for the user to choose. Once the user picks, the system will generate questions and then proceed with the next step in the system. This scenario cannot be ignored or bypassed.



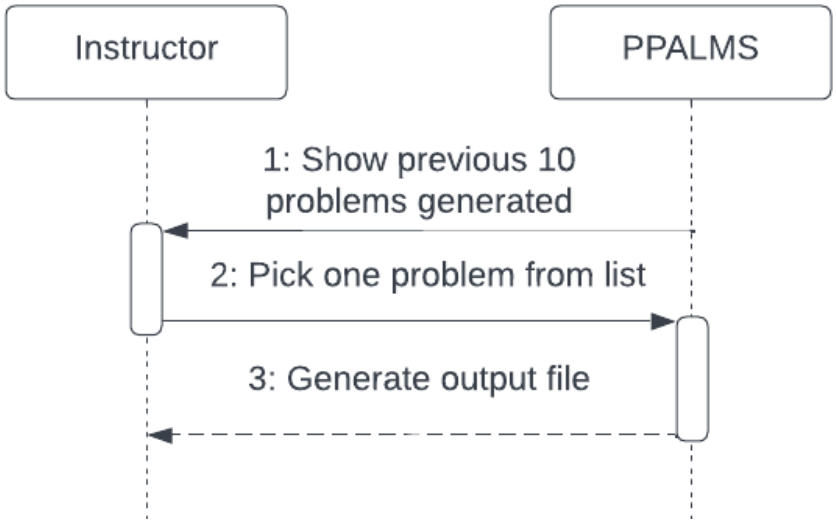
5.1.5 Select Intended LMS Target

This is the last scenario that would occur when a user uses PPALMS. The whole purpose of this system is to output a question problem, and this scenario achieves this. The user will select the intended LMS (Learning Management Systems) target and select the type of output file to export and then receive the output file.



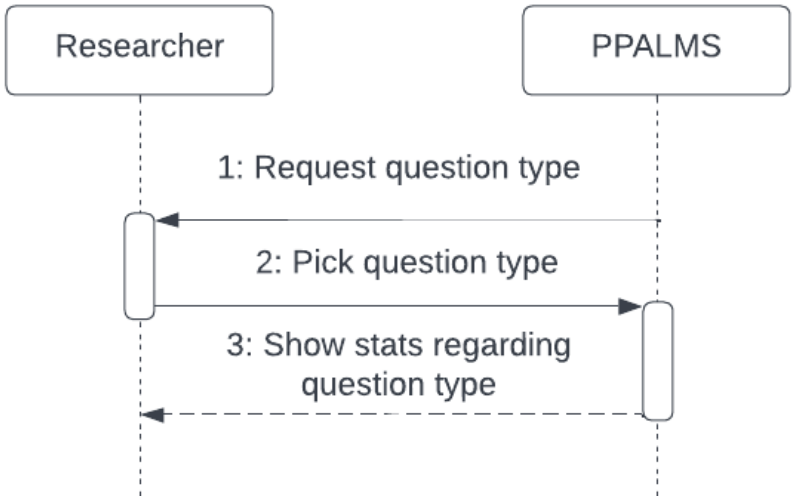
5.1.6 Reuse of problems

This scenario could occur after the user has already used the system before. In this scenario, the user would want to reuse a problem that has already been generated in the past. The system will request for the user to select one of the previous 10 problem sets they want to reuse and output that problem question.



5.1.7 Get statistics

This scenario occurs when a researcher wants to get statistics on the system. A statistic will be the percentage of the time an instructor used a specific question type. The system will request the user to select a question type then shows statistics regarding that question type.



6 Non-functional requirements

6.1 Performance Requirements

The system shall support each user utilizing the desktop application on the latest versions of MacOS, Windows, or Linux. The system's response time shall depend on the internet connection of the user. The desktop computer and the memory size shall also depend on the user. This performance requirement will be met by design by making sure the program application works on each operating system and making sure that the program doesn't use a lot of memory.

6.2 Safety Requirements

The source code for the system will be revised regularly by the research and development team. This will be achieved by design by including a bug and error reporter so the research and development team can keep track of bugs and safeguard the system from potential data loss. Test cases written in this design document should also minimize this issue.

6.3 Security Requirements

The system requires that only instructors and teacher assistants are able to assign problems and view solutions. There is no distinction between the security privileges of instructors and teacher assistants in this system. This is achieved in the design by making sure that users need to log in and their credentials are checked before giving them access to the system.

6.4 Software Quality Attributes

- **Availability:** Design of overall system makes the system available for question selection, formation, annotation, and importation into a LMS at all times.
- **Adaptability:** Design shall accommodate users with user-friendly software features that do not require additional training for using the system.
- **Reusability:** Design allows the potential re-use of problems.
- **Usability:** Design of the system is to be a desktop application that is able to download the output files to the computer.
- **Correctness:** Design has enough test cases to make sure the system shall contain the correct number of selected questions, types of questions, and any annotations made by the user.
- **Interoperability:** The system shall save the changes made by the user. Changes made by other users will not be saved across systems. Users can only change the information on their desktop.

6.5 Business Rules

- Users: Instructors and teacher assistants have the same roles and responsibilities in the system. These users have access to all of the features within the system. Design assures this by treating instructors and TAs as the same user.
- Research and development team: This team has access to the system's features as well as the code base. Only the research and development can make system changes such as bug improvements and updates to the system's interface. Design makes sure that code is documented well to make the research and development team work smoother.
- Admin: The system has no designated system administrator. Default user is all that is necessary. There is no administrator in the design.

6.6 Internalization Issues

The design is abstract enough that it allows the system to be implemented in any language to be used within the system, this eliminates any internalization issues at this stage.

6.7 Future upgrades

The class and code structure and design of the system makes sure that the system is closed for change and open for extension. Future developers can add to the code with ease because of our class structural design and documented code.

7. Requirements Traceability Matrix

ID	Requirement	Design Element	Test case
1	Import source code	Scenario 5.1.1, System Interface 3.1.1	Test Case 21
2	Annotate lines	Scenario 5.1.2, System Interface 3.1.6	Test Case 9,10
3	Form and order line tuples	Scenario 5.1.3, System Interface 3.1.7	Test Cases 11, 12
4	Select question type	Scenario 5.1.4, System Interface 3.1.2	Test Case 5, 6, 7
5	Select LMS	Scenario 5.1.5, System Interface 3.1.3	Test Case 18, 19
6	Reuse problems	Scenario 5.1.6	Test Cases 13, 14, 15, 16
7	Get stats	Scenario 5.1.7, System Interface 3.1.4	N/A