Software Requirements Specification

for

PPALMS

Version 1.2 approved

Prepared by Sam Williams, Daniel Swarts, Jack Sellner, Farhan Idris

University of Minnesota: CSCI 5801

10/17/2022

Table of Contents

Introduction	**3**
Purpose	**3**
Intended Audience and Reading Suggestions	**3**
Product Scope	**3**
References	**3**
Overall Description	**3**
Product Perspective	**3**
User classes and Characteristics	**4**
Product Functions	**4**
Design and Implementation Constraints	**4**
User Documentation	**4**
Assumptions and Dependencies	**5**
External Interface Requirements	**5**
User Interfaces	**5**
Hardware Interfaces	**5**
Software Interfaces	**5**
Communications Interfaces	**5**
System Features	**6**
Upload Source Code	**6**
Create Quizzes	**6**
Upload Quiz to LMS	**7**
Access Generated Problems	**8**
Other Requirements	**10**
Appendix	**11**
Appendix A: Glossary	**11**
Appendix B: Analysis & Conceptual Models	**11**
Use Case Diagram	**11**
Context Diagram	**12**
Interaction Diagram	**13**
Structure Diagram	**15**
Appendix C: To Be Determined List	**16**

Revision History

Name	Date	Reason For Changes	Version
Jack Sellner, Sam Wiliams	10/17/22	Requirements Review Revisions	1.1
Sam Williams, Daniel Lifson, Farhan Idris	10/17/22	Add Structure Diagram, Interaction Diagram, Context Diagram (respectively)	1.2

1. Introduction

1.1 Purpose

The purpose of this software is to automatically generate fresh Pearson's problems. Those problems will then be used to populate a Learning Management System with an effectively infinite number of variations to train new CS students with.

1.2 Intended Audience and Reading Suggestions

The intended audience will include Instructors, TAs, Programmers and Researchers. Each should get a full understanding of the features included in the current iteration of the system best through getting and understanding via the overview and the detailed description of the features below.

1.3 Product Scope

The scope of this project is in creating the problems themselves, not with iterating with the LMS's. To expand, outside parties are working on creating a system for uploading quizzes to the LMS, so this system's main scope is from a user (primarily an instructor) gathering source code and preparing a quiz document outlining their quiz needs, to the quiz being generated and passed to the LMS upload system (if complete), or exported to a readable format.

1.4 References

The sources referenced in this document are as follows:
- https://en.wikipedia.org/wiki/Parsons_problems

2. Overall Description

2.1 Product Perspective

The product is a new standalone system that is being developed in order to provide a persistent, machine-based application. It is designed to mimic similar products which have been developed for research purposes, and distributed and used over the cloud. Since the research would eventually end, this leaves those products out of service for smaller teachers. This product is to be designed as a relatively permanent solution which can remain on the machine of a teacher without expiration. Essentially, the product's goal is to take source code and generate Parson's Problems. Using the problems provided, an instructor can upload the output file to the corresponding LMS which administers the quizzes. The diagram shown in Appendix B, 6.2.1, shows the system flow with the instructor, the main user, and the LMS.

2.2 User classes and Characteristics

The main scope for this project includes Instructors and TAs, who essentially utilize the system in the same way. This software is explicitly designed for instructors (and by extension, their TAs) to be able to make quizzes for their students, so the instructor should be informed (via the README and this specification) how to fully use the PPALMS for quiz production. Programmers are expected to be interested in expanding or utilizing the usage of PPALMS, so will need a more in depth understanding of the architecture of the system, as outlined in the diagrams in the Appendix. Researchers are individuals intending to gather data from the usage of PPALMS, and although at this time no express data collection methods are intended to be implemented, expanding this in future is definitely a possibility.

2.3 Product Functions

Users should be able to:

- Upload source code
- Generate a new quiz
- Upload quiz to LMS
- Access previously generated problems using problem handler
- Access previously generated quizzes within quiz directory

2.4 Operating Environment

The programming language is currently TBD. No matter which operating system you use, you must be able to compile and run executable programs in the decided programming language.

2.5 Design and Implementation Constraints

The main constraints are going to involve limitations on time and space. A large chunk of source code, for example, could potentially produce a near limitless number of problems which would leave the product generating for extended periods of time. Additionally, an output file of such a scenario would be a massive file which would be difficult to transport throughout a system and to an LMS by normal means in addition to the storage space it occupies. Adhering to this, a design must have a way to limit or provide tools for limitations on such edge cases.

2.6 User Documentation

Instructions for operating the systems will be included in the README. More TBD.

2.7 Assumptions and Dependencies

It is currently TBD if we will lean on prebuilt software/libraries. Each use case/process will specify assumptions in respective sections.

3. External Interface Requirements

3.1 User Interfaces

The system will be programmed to run on a Linux bash terminal, and with a general file reading interface for complex information transfer (such as quiz generation) Other interfaces, such as a GUI, are not planned but will be considered. This decision was reached due to time constraints on the project, but care will be made to minimize the amount of interaction so a future GUI would not be difficult to implement.

3.2 Hardware Interfaces

The system will be programmed to run on Intel processor running machines, running Ubuntu Linux, but allowing other hardware interfaces is still TBD.

3.3 Software Interfaces

An API for interacting with different LMS systems is in the works, and will be how the system uploads quizzes to there. The system is expected to allow direct upload to the API if possible, but if not it is expected that the user will need to take the exported quiz file and upload using the LMS upload system directly. We have no knowledge of the time table for the LMS upload system being released. No other software interfaces are planned at this time.

3.4 Communications Interfaces

No communication interfaces are planned at this time.

4. System Features

4.1 Upload Source Code

4.1.1 Description and Priority

The system should allow instructors to upload source code for the system to generate problems out of. This is of the highest priority, as without it there will not be questions for the quizzes to be generated from.

4.1.2 User Process

- Step 1; identify source code file.
- Step 2; upload file to the "input" directory within the system.
- Step 3; upon quiz generation start, source code handler reads from input directory and will utilize a randomly chosen piece of source code.
- Step 4; upon use, source code will be moved to an "archive" directory.

Alternative Paths: If no source code directory or archive exists, when Step 3 is reached, a directory will be made and the program will exit gracefully, notifying the user that a directory was not found and is being generated.
Exception Paths: If no uploaded source code includes the file extension outlined in the quiz document, but source code exists in the folder at Step 3, the program will exit and inform the user that no source code matches the type outlined in the quiz document. If no source code exists, but the folder does upon reaching Step 3, the program will exit and inform the user that no source code is present at all.
Extension Points: None.
Trigger: User intends to upload source code to PPALMS.
Assumptions: User has read this document and the README and is familiar with what types are source code are allowed, and where to put them.
Precondition: User has gathered various pieces of source code.
Postcondition: Source code has been handled for quiz generation and archived to avoid repeat questions.

4.1.3 Functional Requirements

- REQ-01: System should accept code of many different supported languages (.py, .java, .c, and .cc files).
- REQ-02: System should be able to identify files based on file extension.

4.2 Create Quizzes

4.2.1 Description and Priority

The system should allow instructors to request quizzes to be created from the problems generated from previously uploaded source code, the quantity and makeup of which will be defined by them. This is the highest priority, as generating quizzes is the primary function for the PPALMS system.

4.2.2 User Process

- Step 1; create a quiz specification .txt document.
- Step 2; upload this document to the "document" directory within the system.
- Step 3; compile and run the executable program that will generate the quiz.
- Step 4; quiz will be generated, and output to a folder, and uploaded to the LMS using the appropriate API (in the works), depending on how the config is set up.

Alternative Paths: If step 3 is reached, and no quiz documents folder or config file exists, the program will instead generate the folder and file (whichever is missing), inform the user, and exit gracefully. This check is done before any exception paths.
Exception Paths: If an invalid file name is entered for the quiz specification document, when Step 3 is reached the program will exit and inform the user that the file name does not match the correct formatting. If no quiz specification document is present in the appropriate folder, when Step 3 is reached the program will exit and inform the user that a quiz document is needed. If the quiz specification document is incorrectly written (improper syntax, or using types not yet implemented in the system), when Step 3 is reached the program will exit and inform the user that the quiz document is not properly formatted, and explain why. If the config is not properly set up, when Step 3 is reached, the program will exit and inform the user that the config file is formatted incorrectly, and explain why.
Extension Points: None.
Trigger: User intends to build a quiz for a class.
Assumptions: User has read this document and README and understands the quiz document format requirements.
Precondition: Use case 4.1 has been completed up to Step 2.
Postcondition: A quiz text file has been generated, the quiz has been uploaded (assuming automatic LMS upload is implemented and enabled), and problems are stored for future use.

4.2.3 Functional Requirements

- REQ-03: Create many different types of pearson's problem.
- REQ-04: Create problems in many different programming languages.
- REQ-05: Compile problems into a human readable "quiz" .txt file.

4.3 Upload Quiz to LMS

4.3.1 Description and Priority

The system should allow instructors to upload a generated quiz file directly to their preferred LMS system, via the API being produced by the backend operators. This is a medium priority, as that functionality has not been implemented yet, but eliminating an instructor's need to take the quiz file to a separate system greatly increases the ease of use of the system.

4.3.2 User Process

- Step 1; run the appropriate executable file (TBD).
- Step 2; input the name of the quiz .txt file when prompted.
- Step 3; input the name of the preferred LMS when prompted.

Alternative Paths: If the automatic upload feature is completed and selected in the config, these steps are skipped, and the file will be uploaded using the executable automatically.
Exception Paths: If the Step 2 is reached and the file is formatted incorrectly or does not exist, the executable will exit and inform the user of the cause. If Step 3 is reached and the input name of the LMS has not been implemented, the executable will exit and inform the user that the LMS they entered is not valid.
Extension Points: None.
Trigger: A quiz is exported (if automatic upload is implemented and enabled) or the user intends to upload a previously generated quiz txt file to their desired LMS.
Assumptions: Some system for LMS upload has been implemented by the party involved in its development, and the file format for quiz .txt files is such that it can be read by their system. Further, the user has read this document and the README and understands how to proceed.
Precondition: A quiz .txt file has been previously generated and acquired by the user, or a generated quiz is exported (if automatic upload is implemented and enabled).
Postcondition: The quiz is converted to a format readable by the user's desired LMS and uploaded to their system.

4.3.3 Functional Requirements

- REQ-6: Quiz .txt file must be in a format such that it can be properly and easily handled by the LMS upload system (system outside scope of our project).
- REQ-7: Must support all common LMS systems.

4.4 Access Generated Problems

4.4.1 Description and Priority

The system should allow the instructor to access examples of problems generated from uploaded source code during quiz generation, to view, extract, and remove them as necessary. This is a low priority, it would allow an instructor to pull problems for outside quiz use (lectures), and verify questions are formatted as intended, but the system can run without this.

4.4.2 User Processes

- Step 1; User prompt system to display pre-generated problems.
- Step 2; System returns numbered list of pre-generated problems and awaits instructions.
- Step 3; User chooses a problem to open.
- Step 4; System prints details of problem and awaits instruction to do with problem.

- Step 5; Users may choose to extract or delete the problem.
- Step 6; System follows directive to extract or delete the problem.
- Step 7; System prints summary and exits.

Alternative Paths: Users may exit at any time in steps 3 and 5.
Exception Paths: If no problems exist, step 1 skips to step 7. If the problems directory doesn't exist at step 1, the system generates a new problem directory then exits and informs the instructor of the reason for exiting.
Extension Points: None.
Trigger: User intends to view and/or pull problems from problem directory.
Assumptions: User has read this document and the README and is aware of how to proceed.
Precondition: Problems have been previously generated during quiz generation.
Postcondition: The user has a text based version of a problem (if desired) and have removed any problems they wished removed.

4.4.3 Functional Requirements

- REQ-08: When prompted, list stored problems in readable format.
- REQ-09: All extraction of stored problems.
- REQ-10: Allow removal of stored problems.

5. Other Requirements

5.1 List of Parsen's problems:

- Fill in the blank
- spot the bug
- ordering
- matching
- multiple choice
- etc.

5.2 List of supported languages:

- c
- java
- python
- pseudo-code
- etc.

5.3 Performance requirements:

- The primary performance metric, aside from if the software simply can accomplish the desired tasks, is time related. The software must be able to generate quizzes and upload to LMS within a reasonable amount of time. This should be configurable by the user, but it should likely default to approximately 30 seconds time limit on time generation. This also helps limit creation of massive files with huge numbers of problems and quizzes created which consumes storage resources. If instructors wish to expand any of the performance requirements, the values they are limited to will be customizable within the config file.

5.4 Security requirements:

- There is minimal security risk involved with this software. Without connection to the cloud, such security threats are a much lower priority. Additionally, the number of malevolent users significantly decreases. The primary concern in security is to ensure the protection of quizzes, problems and their answers from students looking to cheat. This can be done simply through common sense computer security practices such as locking one's computer when not in use and having password protected login. Now, if these practices are not followed there should be an option to encrypt the quiz files or lock storage folders containing such problems to ensure security. Due to the minor risk of this scenario, these security requirements are of low priority. However, once the system is fully functional, additional security measures are an early area for improvement, especially at the interface between the PPALMS and the LMS uploading system being produced externally.

6. Appendix

6.1 Appendix A: Glossary

PPALMS - Parsons Problem Application for Learning Management Systems

Parsons Problem - Parson's problem is a program in which the lines of code containing the solution have been mixed up and the person solving the problem must rearrange the blocks of code into the right order.

LMS - Learning Management System

Learning Management Systems are essentially web tools designed to help instructors run courses. This includes giving announcements, posting information or lectures, the distributing, collecting, and grading of homework, quizzes, tests, and projects; displaying grades, and other general course activities. Examples include Canvas, Moodle, and Blackboard.

6.2 Appendix B: Analysis & Conceptual Models

6.2.1 Use Case Diagram

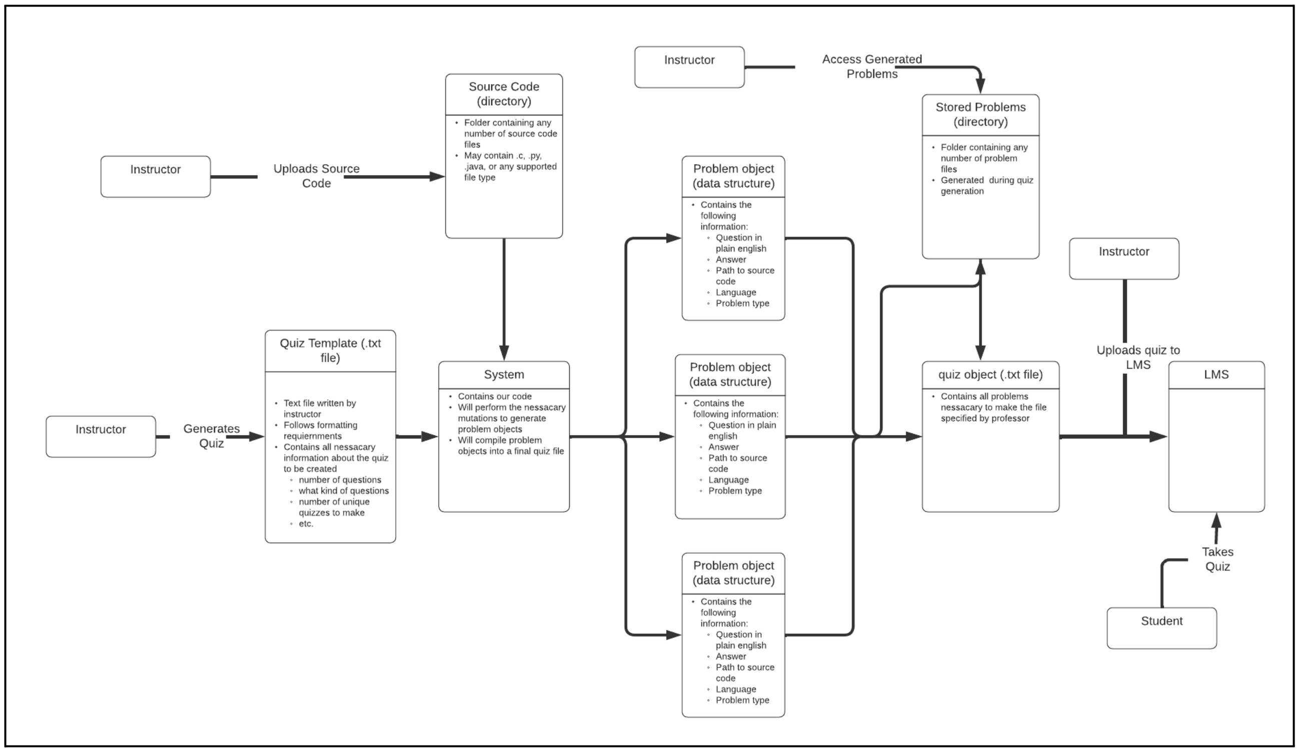

Instructor
Access Generated Problems
Source Code (directory)
Folder containing any number of source code files
May contain .c, .py, .java, or any supported file type
Stored Problems (directory)
Folder containing any number of problem files
Generated during quiz generation
Instructor
Uploads Source Code
Problem object (data structure)
Contains the following information:
Question in plain english
Answer
Path to source code
Language
Problem type
Instructor
Quiz Template (.txt file)
Text file written by instructor
Follows formatting requiernments
Contains all nessacary information about the quiz to be created
number of questions
what kind of questions
number of unique quizzes to make
etc.
System
Contains our code
Will perform the nessacary mutations to generate problem objects
Will compile problem objects into a final quiz file
Instructor
Generates Quiz
Problem object (data structure)
Contains the following information:
Question in plain english
Answer
Path to source code
Language
Problem type
quiz object (.txt file)
Contains all problems nessacary to make the file specified by professor
Uploads quiz to LMS
LMS
Takes Quiz
Student
Problem object (data structure)
Contains the following information:
Question in plain english
Answer
Path to source code
Language
Problem type

Figure 01: PPALMS Use Case Diagram

6.2.2 Context Diagram

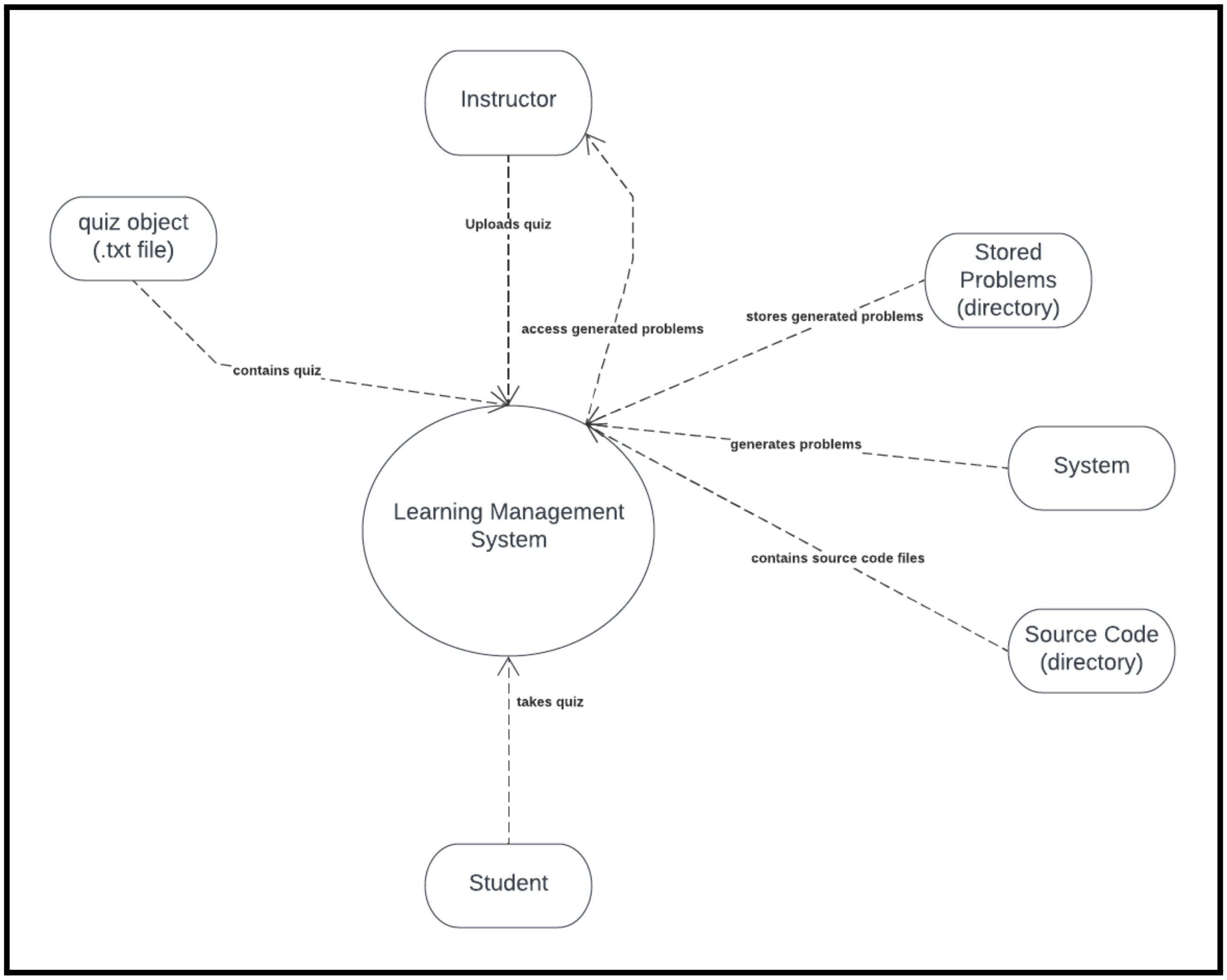
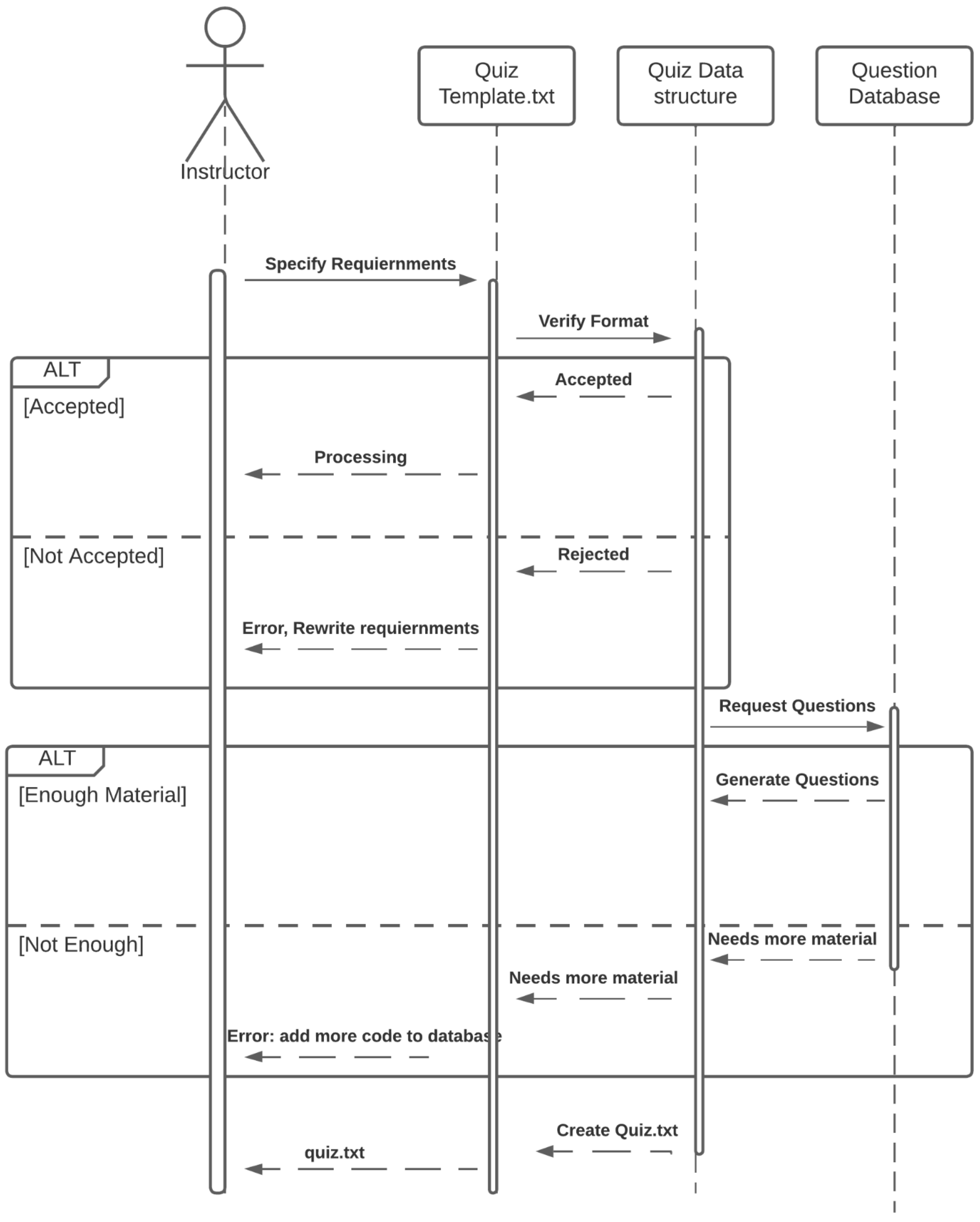

Instructor
quiz object (.txt file)
Uploads quiz
access generated problems
stores generated problems
Stored Problems (directory)
contains quiz
generates problems
System
Learning Management System
contains source code files
Source Code (directory)
takes quiz
Student

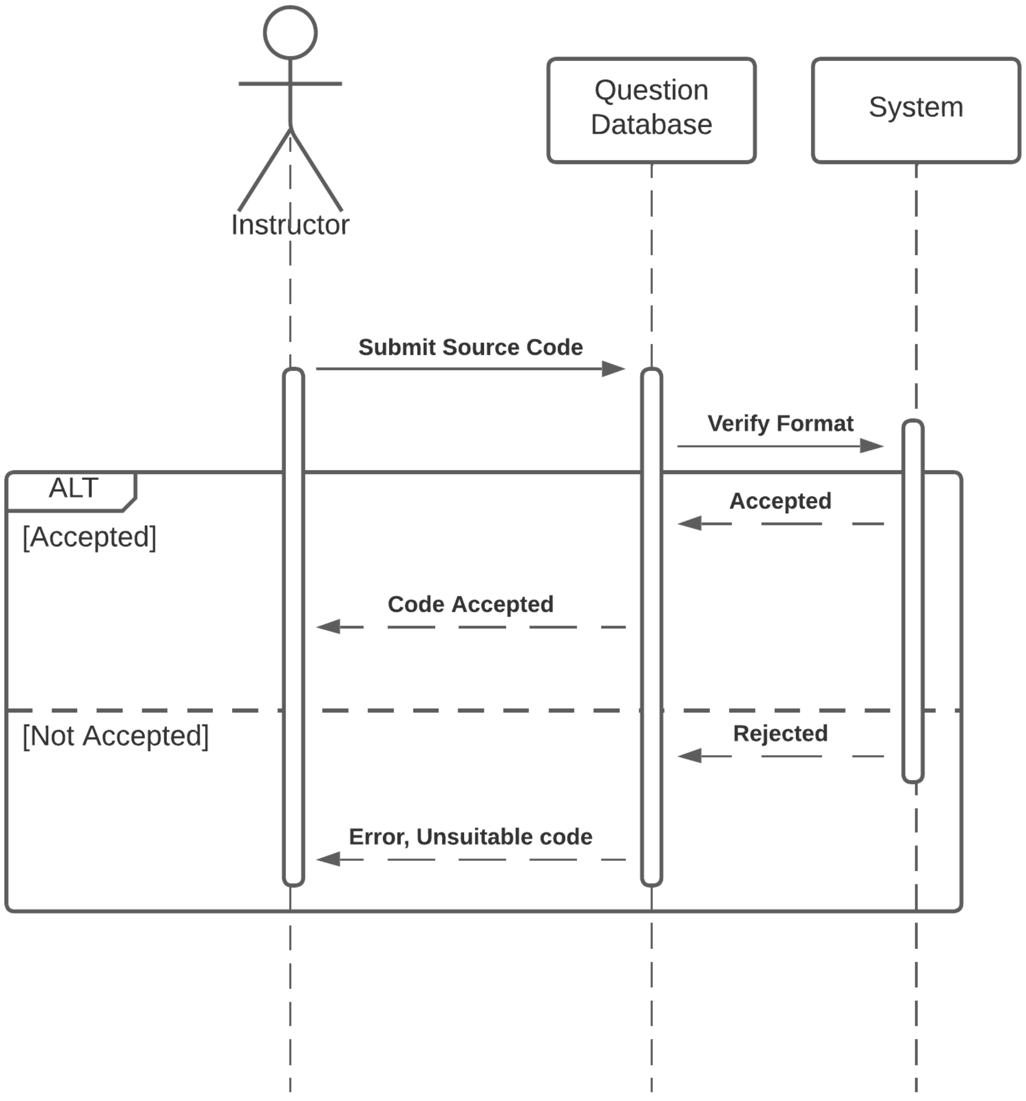
Figure 02: PPALMS Context Diagram

Structure Description: As shown in the figure above, the instructor accesses the generated problems (created by the system that contains the source code directory), from the stored problems directory and uses those files to create a quiz object and upload the quiz object to the LMS. From there the student takes the quiz that's being provided from the LMS.

6.2.3 Interaction Diagram



Sequence diagram for use case 1: generating a quiz. In order to streamline the input process, all of the user input will be saved to one text file that that system can open and reference. This document must contain the blueprints for the quiz the instructor wants. The instructor will write this document, following syntax guidelines specified by this document, so that our system can understand what is being specified. Once the quiz template text document has been finished, the system will check the syntax to ensure its integrity. Then the system will attempt to generate each individual question for the quiz from real source code contained in its database. If the database is too sparse to make the number of questions required for the quiz, the system will return an error.



Sequence diagram for use case 2: Adding source code to the database. The instructor will add a source code document to the system's question directory. The code's syntax will be tested to ensure integrity. If the source code is accepted, it is added without error. If rejected, the source code will not be added.

6.2.4 Structure Diagram

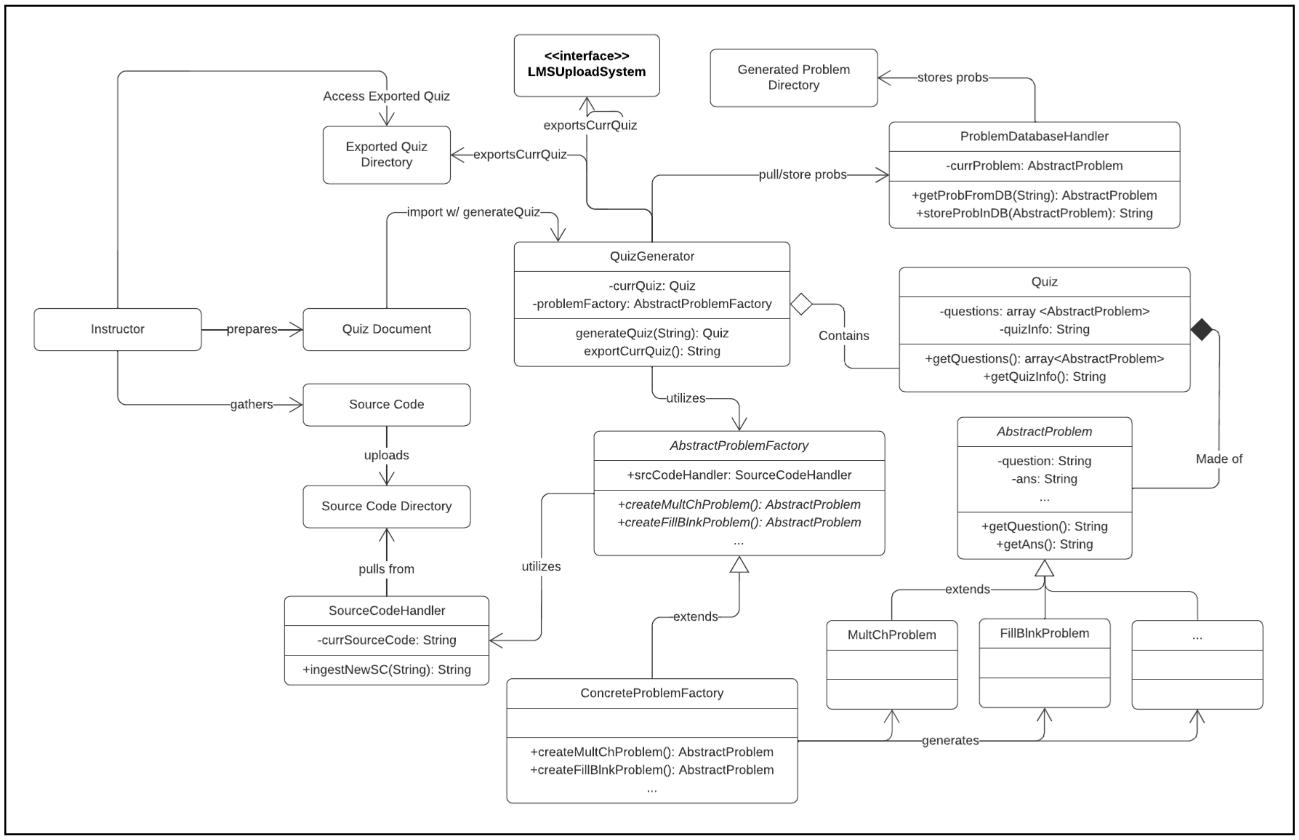


Figure 4: PPALMS Structure Diagram

Structure Description: As shown in Figure 4, the instructor (or other user) must gather source code and upload them to a source code directory, which a source code handler will later pull from. Next, the instructor must prepare a quiz document with the desired quantity and type of problems, as well as the language of source code desired. This is imported with the main method of the QuizGenerator class, passing the file extension using some form of generateQuiz method. This will utilize a problem factory to generate problems for a new quiz as outlined in the document, storing them in a quiz data structure containing an array of the abstract problem files. The factory shall utilize the source code handler mentioned earlier to pull random source code from the directory to utilize in the problem generation. Some percentage of problems will also be pulled from the existing problems directory, as outlined in the config and as available, which is handled during quiz generation by the quiz generator, not the factory, which only generates new problems. After the quiz is fully generated, the quiz is exported, which involves creating a final quiz file for the instructor, handing off the quiz file to the LMS uploading API currently in the works (assuming the feature is enabled and implemented), and storing any problems flagged as newly generated within the problem database, which at capacity will replace older problems with the newly generated ones (capacity defined in the config). This capacity method is also utilized for eliminating excess stored quizzes, and export of the quiz to either the LMS directly (once implemented) or the directory will be optional via the config.

6.3 Appendix C: To Be Determined List

1. 2.4 - Operating Environment
2. 2.6 - User Documentation
3. 2.7 - Assumptions and Dependencies
4. 3.2 - Hardware Interfaces