Murdoch University

CourseLoop API - Requirements and Analysis

ICT302 IT Professional Practice Project,
Team (10) Black Widow

Blake Williams, Enrique Getino Reboso, Jordan Chang-Yeat Lau, Patrick NKhata, Jared Eldin and Sara Hussain

23-8-2019
Version 1.0

Table of Contents

Executive Summary

Murdoch University has in place a manual process where the staff communicate information between two applications, CourseLoop to Callista. Due to the likelihood of human error, the University has assigned Black Widow to create an application that allows communication between the two systems to reduce human error and to provide efficient and timely information to students.

This project purposes the requirements for the extraction of the course structural information from the CourseLoop curriculum management system via API, which will then be transformed to the course and unit set rule strings and loaded to the Callista student management systems using the Callista standard rule strings.

This document outlines the problems with the current application communication processes as well as the benefits of implementing an automation tool to reduce human interaction. Also outlined are:

- A methodology
- A solution
- Primary and secondary Functional and Non-Functional requirements
- Diagrammatic requirements

Black Widow aims to develop the aforementioned API to ensure the students receive up to date course and unit information to increase successful and accurate enrolments.

Introduction

Project Purpose

Black Widow purposes the extraction of the course structural information from CourseLoop curriculum management system via API and transform the payload into course and unit set rule strings for use in the Callista student management systems, utilising Callista standard rules syntax. The application will aim to create job tickets for rule string creation and amendment for all changes to the course and unit sets to ensure they meet the defined business rules. The application also intends to include exception reporting, such as warning for rules unable to be translated or changed to standard syntax, for example where the API payload include queries rather than directly referencing curriculum item.

This project is needed in order to automate and facilitate the correct communication between CourseLoop and Callista. This will assist in providing a better understanding of the Handbook by students, University staff and users in general. It will also provide a better procedure in terms of managing the communication between both systems, as the API will be designed to try to avoid human errors in the processes executed. This application will be oriented towards the simplification of the communication processes between CourseLoop and Callista, avoiding human interaction as much as possible.

Background

The current process that is being used for communication between CourseLoop and Callista is not automated. It is necessary to take the data generated by the first application and handle it manually so the second application can read the payload extracted. This process is being carried out by university staff, which means that human resources are used in the communication management of both platforms. The design and development of an application that performs the communication functions between both systems would be very beneficial for better management of both, the information and the resources of the university. This API should ease the mentioned processes as well as make them faster, releasing workload from the university staff who is carrying out these tasks at the moment.

Problems and Opportunities

PROBLEMS

- The actual communication between CourseLoop and Callista is done manually by university staff.
- It is time consuming and the fact that the process is done manually can always cause human errors which could cause malfunctions in the systems.

- A malfunction or error in the system may cause that the Handbook shows wrong or incomplete information about the course structure, leading the user to be misinformed about the requisites needed to obtain a degree.

OPPORTUNITIES

- The creation and implementation of the API could mean an enhancement of the whole communication process between systems.
- The human resources used to perform the process manually could be used in other fields.
- Automation of this type of processes can also help to identify errors in the communication of the systems.
- The API could be adapted to perform other tasks if needed.

Objectives and Goals

The objectives of the API, as stated in the previous sections, will be to automate the communication processes between both platforms, CourseLoop and Callista. Our goal is, as requested by the client, to ensure that at least 80% of communications between both platforms is accurate and without failures.

Methodology

The methodology that Black Widow has decided to proceed with this project is the Agile methodology. We believe that Agile will be the most appropriate methodology for this project due to the API needing constant iterations for a perfected product. Breaking bigger tasks (stories) into smaller tasks (activities) allows the team to react to immediate feedback set by both the supervisor and the client, where as if Black Widow followed the waterfall method, we would have to retrace all of our steps in order to respond to feedback. SCRUM is also an excellent tool bundled with the Agile methodology (which we have deployed via a web application called Trello), in turn the team gets a clear indication of where activities are at in their stage, what the member is doing what task and an overall summary of how far the stories are from being completed. The Agile methodology is used in many other IT projects, and we believe it suits our needs perfectly.

Solution Outline

Scope

The project is to take course structural information from CourseLoop curriculum management system via API and transform the payload in to course and unit set rule strings for use in the Callista student management systems utilising calilista standard rules syntax.

The application should create job tickets for rule string creation/amendment for all course and unit set changes meeting defined business rules. The application should include exception reporting, including warning for rules unable to be translated or to standard syntax, for example where the API payload include queries rather than directly referencing curriculum item.

Narrative Description

The implementation of the system will be through the Javascript programming language, using the typescript superset to add language features that will aid in development. Most notably static type checking which should make code more robust and aid in debugging data type errors which javascript development is prone too. The API will be developed in Node.JS to run the code locally on the machine rather than in a web browser. This will make deploying the finished application intuitive and simple in most environments

Receive input from CourseLoop

Translation of the structural information from CourseLoop into strings readable by Callista.

- this task is a critical task being our main priority. It requires being able to receive an input from CourseLoop, making it it's dependency. A major risk associated with this task is that if the input is incorrectly translated, we could be sending the wrong data to the Callista system. This task will be initiated after the design document is complete.

Output translation to Callista system

Data must be in a string format following the syntax rules of Callista platform

Functional Requirements

REQ-ID	**Description**	**Criticality (1-5)**	**Priority (LOW, MEDIUM, HIGH)**	**Requires**
REQ-1	Listen for incoming job tickets	3	MEDIUM	
REQ-2	Add job ticket to queue when listener triggers	3	LOW	REQ-1
REQ-3	Get job ticket from front of queue	3	LOW	REQ-2
REQ-4	Request data from Courseloop API	3	HIGH	
REQ-5	Extract required data from Courseloop payload	2	MEDIUM	REQ-4

REQ-6	Transform payload into Callista standard rules syntax query string	5	HIGH	REQ-4, REQ-5
REQ-7	Verify output string	4	HIGH	REQ-6
REQ-8	Log details of job ticket	1	LOW	REQ-3
REQ-9	Deliver output string to Callista system	3	MEDIUM	REQ-7

Breakdown of Functional Requirements

REQ-1

1. Develop a network listener that waits on a certain port for job tickets incoming over the network and adds them to the processing queue

REQ-2

1. develop a FIFO (queue) data structure that can hold job tickets.
2. Have it push job tickets detected from the REQ-1 listener and add them to the queue.
3. Develop functionality for getting job tickets from the queue and removing the frontmost job ticket.

REQ-3

1. Call get and clear method of REQ-2 data structure to get the ticket at the front of the queue which will then undergo processing.

REQ-4

1. Based on the JobTicket Send a HTTP request to the CourseLoop API requesting course data
2. IF: API returns OK (HTTP CODE 200) save returned data into a program variable and Proceed to REQ-5
    - ELSE: Print HTTP Code to console and retry step 1. After five failed attempts log the job ticket with relevant error and go back to REQ3.

REQ-5

1. Extract the required data from the CourseLoop JSON data returned in REQ-4 and sort it into relevant data structures required to process the data into a string query as easily as possible (these data structures will be designed and detailed after we have example query strings to base them off)

REQ-6

1. Use loops and/or Regex on the data structures created in REQ-5 to create the Callista query string

REQ-7

1. Verify string meets requirements by comparing unique aspects of a standard calista string to the string generated in REQ-7. This step will likely use Regex to determine string validity.
2. Depending on the outcome of step 1-mark string VALID if passed tests or INVALID if failed. Attach status to the ticket as a new field.
3. IF INVALID handle exception, Proceed to REQ-8 but don’t continue with REQ-9

REQ-8

1. Log details of job ticket (Date-Time, Output string, Validity, Original JSON input) and save to a local file for future reference and debugging.
2. IF string is VALID proceed to step 9

REQ-9

1. Return the query string to the Murdoch system or application requesting the data.
2. If the system attempts to return back a second query string catch the data and handle as an exception (dealing with a query string that returns another query string is out of the scope of this project, it will be documented how to catch the data and ideas on how to work with it for future improvements to the API).

Schedule

Reference the project management plan and Gantt charts for further scheduling details of the overall project. After the detailed design phase further details will be uncovered and task scheduling will be broken down further for each functional requirement before implementation begins. As a general guide development will begin with Functional requirements which have no other requirements. Functional requirements with the most prerequisite requirements to ensure functionality will be completed last.

Limitations of Current Functional Requirements

Development – We do not yet have concrete details of the exact query string needing creation. Once we gather more details over the coming weeks functional requirements can be further broken down into the processes/technical details required to extract Callista standard rules syntax query string from Courseloop API data.

Non- Functional Requirements

- PERFORMANCE: System must process job tickets in a timely manner to ensure systems relying on the produced query strings aren't delayed when calling the end product API (especially important for systems that produce and consume these strings in real time and are user facing). Anything over 10 seconds would be considered unacceptable even under peak load and generally in lower load load environments a significantly faster time would be expected and desired (Less than 1 seconds).

  - BOTTLENECKS: the API should be able to process as fast as the CourseLoop API can return data over the network. Disk read/writes should be much faster and the main bottleneck will be network latency and speed of transferring data over the network from CourseLoop and too Callista.

- QUALITY: The finished API should be able to automate translation from the Courseloop API to a Callista standard rules syntax query string for at least 80% of the most common Callista query strings. Uptime should target 98% and generally beat this goal. System restarts should target having the system back up and running within 5 minutes.
- PORTABILITY: The software should be able to run on common server hardware and common computers but will not be designed for low performance computer hardware (such as a raspberry pi or similar low performance hardware). As the software will be built in NodeJS the software will be largely portable between operating systems - which will simply need to install NodeJS to run the API. Therefore, this API will be suitable to run on a large variation of host machines, aiding in deployability options.

- DOCUMENTATION: required for each method, and a description of its relationship to other methods and how an IT department could implement the overall system with current systems will need to be provided with the finished project to enable using and potentially upgrading the API in the future.

- ERRORS: The API should be able to handle errors such as invalid or corrupted input, Courseloop API HTTP error codes and handle crashes by automatically restarting the API and recovering job tickets. Where The program crash is repeated and likely to a job ticket logic error the queue should be emptied after several failed restart attempts to allow an automatic restart.

- SECURITY: The application will have an API key that grants access to the murdoch CourseLoop data. As such this application should only be deployed on secure systems as people who gain access to this API key may be able to pull student information from CourseLoop if it’s available (not sure if this is the case). In the case of a breach API access should be revoked in CourseLoop and a new API key generated added to the software and deployed on secure hardware for continued secure usage of the system. Anyone with access to the system running this API will have access to the API and therefore access to the API key, this decision will be up to Murdoch staff IT and security staff as we are unaware of the exact details of the system this API would run on once built.

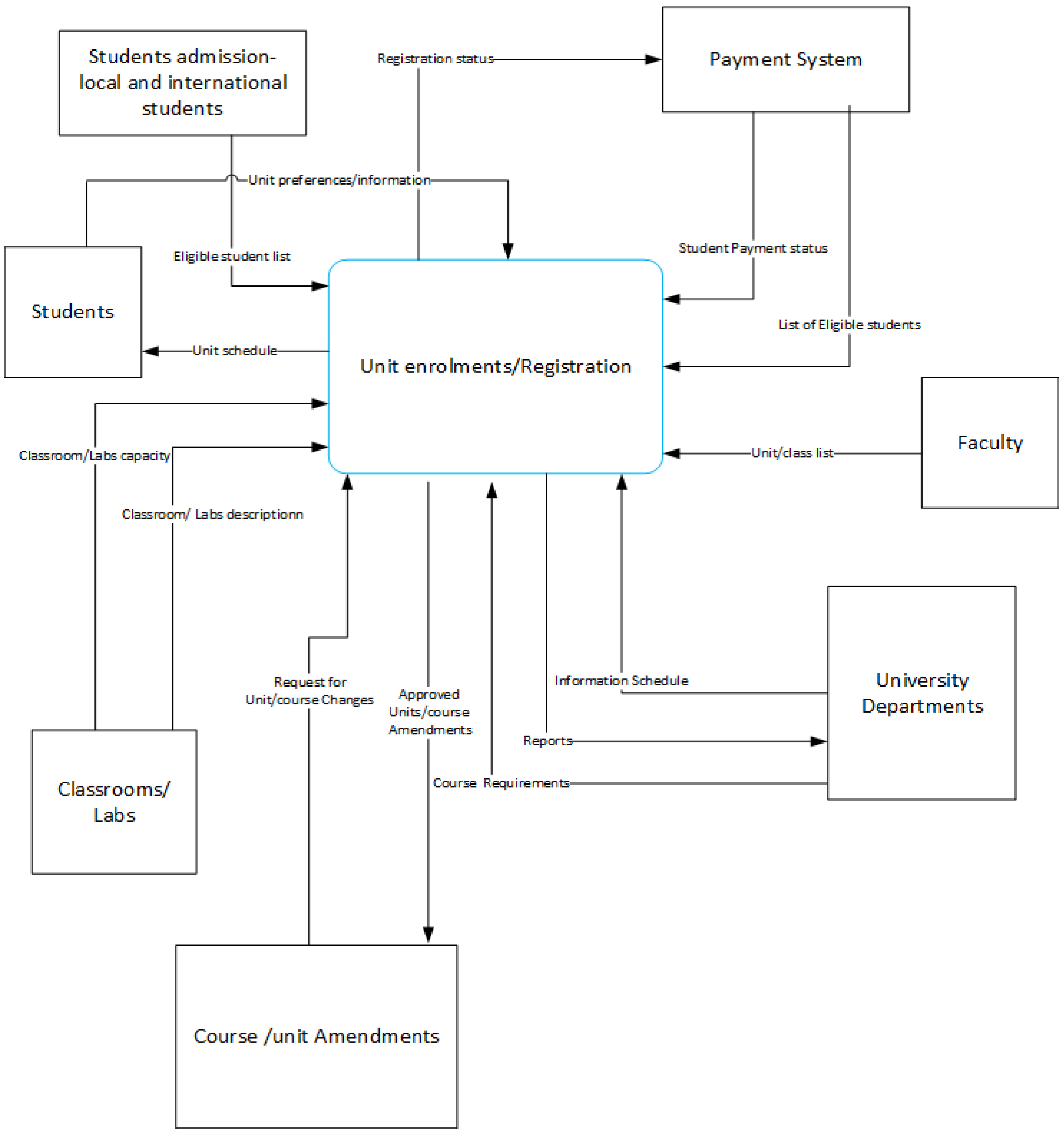
- BACKUPS: A copy of the software should be kept offline and in the cloud to ensure it can be recovered if the host system fails and data is lost. a version control system should be used so old versions can be archived and recovered if issues with an upgraded API are discovered. Modifications should also use Version control techniques so old versions are available to rollback in case of issues.

- MODIFICATIONS: In the future Murdoch devs may require modifying or upgrading the software to suit evolving needs. Documentation should be referenced first to ensure the functionality isn’t already available. From there old software should be backed up and then new changes can be implemented. If the form of the string Callista consumes or variations of strings are required upgrades would be expected.

Diagrammatic representation of the requirements

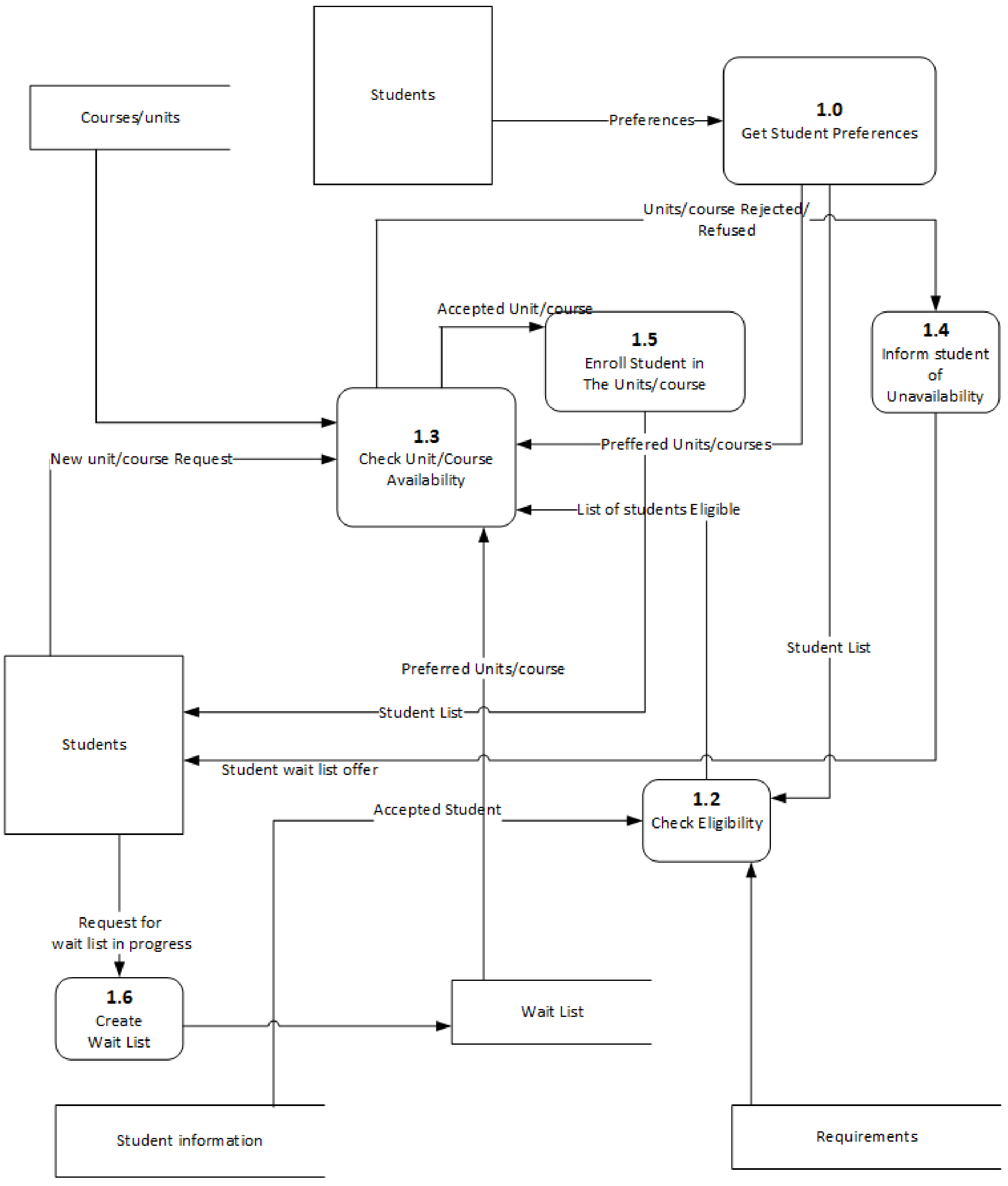
CourseLoopAPI System

Context Diagram

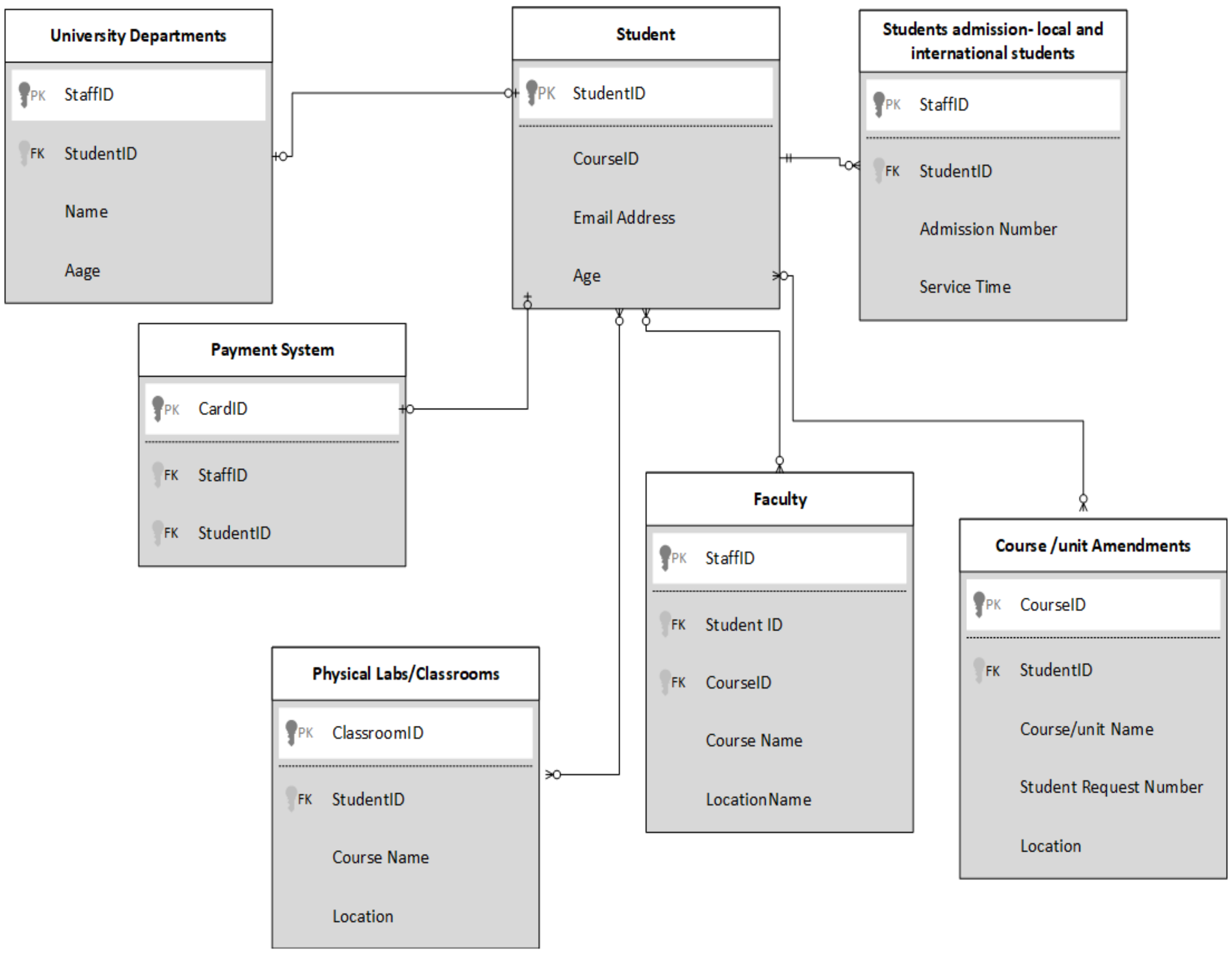


CourseLoopAPI System

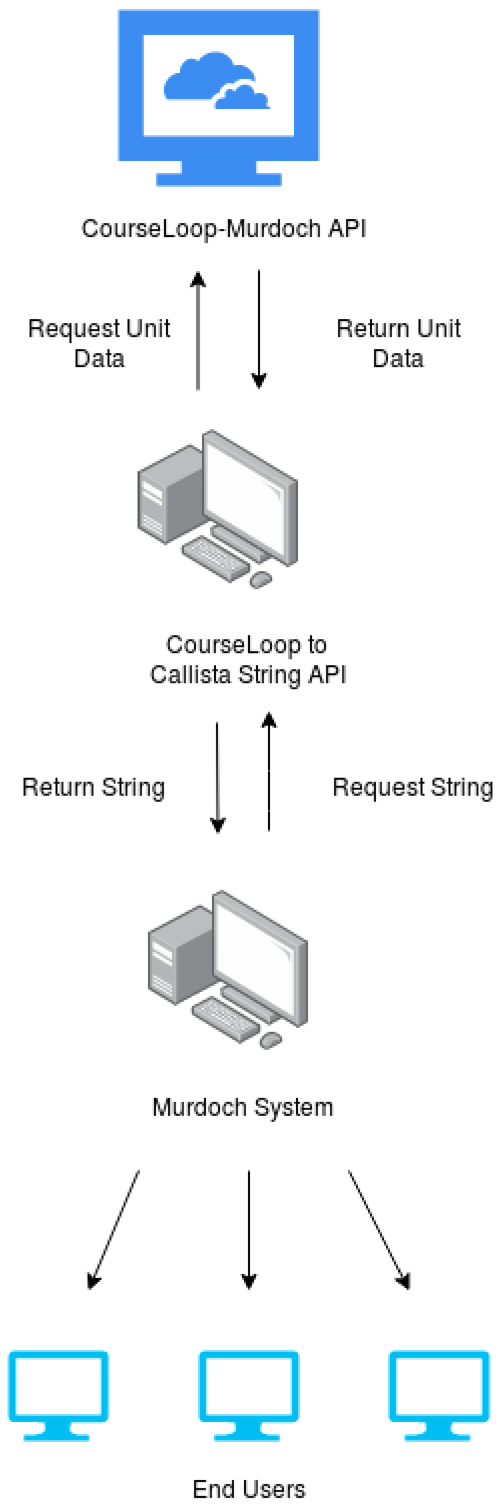
Level 0 Diagram



ERD Diagram


University Departments
PK StaffID
FK StudentID
Name
Aage
Student
PK StudentID
CourseID
Email Address
Age
Students admission- local and international students
PK StaffID
FK StudentID
Admission Number
Service Time
Payment System
PK CardID
FK StaffID
FK StudentID
Faculty
PK StaffID
FK Student ID
FK CourseID
Course Name
LocationName
Course /unit Amendments
PK CourseID
FK StudentID
Course/unit Name
Student Request Number
Location
Physical Labs/Classrooms
PK ClassroomID
FK StudentID
Course Name
Location

Network Topology



Conclusion

Murdoch's current method of manually sharing information between two systems, CourseLoop and Callista, via human input leaves room for unintentional errors. Therefore, Black Widow has been assigned the task of automating the CourseLoop to Callista interaction to reduce human input to provide efficient and accurate information to all users. The team has purposed API as a middle-man application to cancel out all human interaction amongst the two systems. This document outlines problems with the current systems communications as well as SCRUM as an agile methodology that will be used to complete the end-product. Also included is an outline of the solution and the functional and non-functional requirements of API in addition to a diagrammatic representation of these requirements.

Appendices

Appendix A – Deliverable Task Breakdown Statement (signed, scanned and inserted into document)

Unfortunately, due to having external students in the group, the document could not be signed and scanned. Nevertheless, all team members participated in the creation of the R&A document.

Requirements and Analysis

Action to be taken	Person/s responsible	Action to be completed by
Executive Summary	Enrique Sara	26/08/19
Introduction	Enrique	26/08/19
Solution outline	Enrique Jordan	26/08/19
Functional requirements	Blake Jared	26/08/19
Non-functional requirements	Jared	26/08/19
Diagrammatic representation of the requirements	Patrick	26/08/19

Appendix B: - Copies of client documents on which you based the analysis.

We haven't received any client documents on which we are basing the analysis. At this stage of the project, we still need to gather information in order to proceed with the completion of the tasks.

Appendix C: Glossary of Terms

Glossary	
API	Application Programing Interface
String	A sequence of characters
CourseLoop	Curriculum management system
Callista	Student management system