Murdoch University

CourseLoop-Callista String API Design Document

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

26-8-2019

Version 1.0

Table of Contents

Executive Summary

Black Widow has been assigned the task of creating a communication application which will transform the extracted information from CourseLoop JSON file and translate it into a string readable by the Callista System. Black Widow's goal is to create and modify the documentation to ensure the new design works in conjunction with the pre-existing system. The design of the existing system will not be documented here.

This design document includes information that will be used to assist in the automation of an existing system. Outlined here are the data, structural and interface designs of the additional components needed to improve an existing system. The purpose of the algorithm presented is to search CourseLoop JSON file input to create several strings which will be combined to output accurate results from Callista. The architecture and infrastructure outline the requirements for each, such as additional components (architecture) and performance (infrastructure). The interface design provides a short description of the visual and practical aspects of the project after the new changes are implemented.

The changes are intended to improve the existing process of providing accurate information to staff and students and is in no way intended to replace any persons.

Introduction

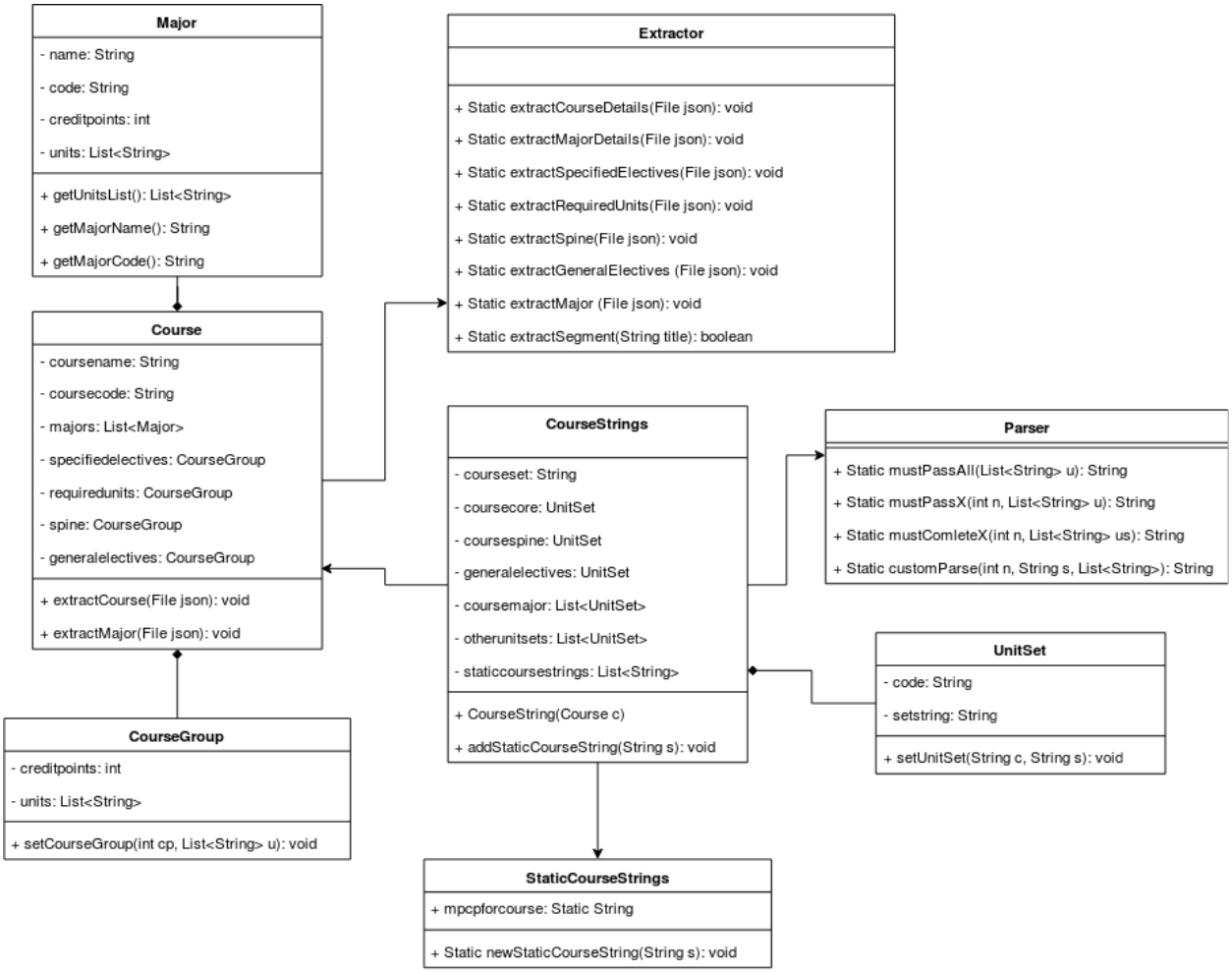
The purpose of this design document is to provide a description of the proposed design so that this project can be implemented to the existing system. The existing system currently allows the gathering of data from CourseLoop so that it can be modified in Callista before it is output. The designs presented in this document are proposed so that the extraction and modification of the data from Courseloop to Callista becomes an automated process. This document will define the data design, architecture and infrastructure requirements and the interface design. The data design outline will provide detailed descriptions of background processes whereas the interface design will describe and showcase the user interaction via a visual representation of the process.

Data Design

The Data structure that is being implemented here is already in place. It takes its form as a JSON file that is provided by CourseLoop. The JSON file transmits an object to our API, this can take form as a whole course, spine units or a major. JSON files use XML to structure the data, it structures it into a dictionary that can be used in programming languages by using keys or indexes to pull the required information out of the JSON file. The Callista system on the other hand uses a one-line string syntax in order to interact with that system. Our API will be taking the current data dictionary to form a relationship between the CourseLoop system, and the Callista system, by parsing the current data structure that is inserted into it and parsing the data into a one line string syntax to the Callista system.

JSON MAIN KEY VALUE: Academic_item_code.

UML Class Diagram



UML Key
u = units, us = unitsets, mpcpforcourse = “must pass credit points for course”

UML Notes
Constructors, getters and setters will be implemented for all classes and private member variables. Only substantially relevant or cases with unique implementation requirements are included here as to avoid bloating the UML diagram.

Overview
The Course class stores relevant blocks of unit codes as CourseGroup objects. It also stores multiple majors related to a Course in a List of Major objects. An object of the Course class can take in a CourseLoop JSON file for a course or major in the extractCourse() or extractMajor() methods respectively. These methods will open the file, feed it to the Extractor class which extracts and returns units lists, which are then stored in the calling Course objects member variables. Once a Course object has unit information loaded it can be used as input to a CourseStrings object.

The CourseStrings class uses the Course member variables passes them to the Parser function which returns and stores the output callista string in the CourseStrings object member variables. Any extra course strings needed that aren’t based on the CourseLoop input can be retrieved and stored in a CourseStrings object from the StaticCrouseStrings Class and stored in staticcoursestringsList<String>. Once the CourseStrings file contains all the required data it is ready to output a full callista rulestrings to the user on the webpage. An example full string which served as the basis for this design and will be the final output can be found below at *Appendix D: Example rule sets*.

UML Diagram Class Breakdown

Major:

Variables

- name - contains the name of the major
- code - contains the major code of the major
- creditpoints - contains the credit points required to qualify for graduation in a Major
- units - contains a list of units in the major

Methods

- getUnitsList() - returns units
- getMajorName() - returns name
- getMajorCode() - returns code

CourseGroup:

Variables

- creditpoints - contains the credit points need to qualify for graduation for a CourseGroup logical unit. These are extracted from the CourseLoop file and needed to create parts of the Callista string
    - E.g 6 credit points required out of 10 units in the list indicates 2 of the 10 (one unit is 3 credit points) units need to be completed. This field is required for the mustPassX() method
        - E.g Parser.MustPassX(acourse.getSpecifiedElectives().getCreditPoints(), acourse);
- units - contains a list of units for a course group

Methods

- setCourseGroup() - allows setting of the course group member variables similar to a constructor

Course:

Variables

- coursename - contains the name of the course
- coursecode - contains the code of the course
- Majors - contains a List of Major objects which contain information about a major and it's units
- specifiedelectives - a CourseGroup object containing details on a courses specified electives
- requiredunits - a CourseGroup object containing details on a courses required units which together with the specified electives make up the coursecore unit set
- spine - a CourseGroup object containing details on a courses spine units.
- generalelectives - a pre-filled CourseGroup with a List<string> of university wide general elective units.

Methods

- extractCourse(File json) - takes in a json file as an argument and calls methods from the Extractor class to extract the required details into the Course object member variables
- extractMajor(File json) - takes in a json file as an argument and calls methods from the Extractor class to extract the required details into a Course object major List variable

Extractor:

Methods

- extractCourseDetails() - implements extraction of course details (course name, course code) from the JSON
- extractMajorDetails() - Implements extraction of major details (major name, major code)
- extractSpecifiedElectives() - Implements the unit extraction algorithm and stores the unit codes in a CourseGroup object
- extractRequiredUnits() - Implements the unit extraction algorithm and stores the unit codes in a CourseGroup object

- extractSpine() - Implements the unit extraction algorithm and stores the unit codes in a CourseGroup object
- extractMajor() - Implements the unit extraction algorithm and stores the unit codes in a CourseGroup object
- extractSegment() - a helper function that can be called by other extractor functions to search for a segment in the JSON by the key “title”. This function is experimental, previously tested extraction algorithms will be used before attempts are made to implement this method.
    - e.g extractSpine() calls extractSegment("spine"), which traverses the JSON to find the title = “spine”. IF no section is found return == true and error should be reported.
    - This way of traversing the JSON could be versatile if the course sections are updated in the future as all that would need to be changed is the string argument to search for.
    - Another potential option may be to traverse a whole CourseLoop file and extract the info from each title section without knowing what to search for to begin with and sorting it based on the title and credit point requirements to generate the Callista string.

StaticCourseStrings:

Variables

- mpcpforcourse - static variable to hold “Must pass credit points for course

Methods

- newStaticCourseString(String s) - adds a new static member variable to the Class.

CourseStrings:

Variables

- courseset - contains the Callista string representing course completion rule
- coursecore - contains the UnitSet with specified electives and required units
- coursespine - contains a UnitSet with the course spine
- generalelectives - contains a UnitSet with the university wide general elective units
- coursemajor - contains a List of UnitSets with major set completion rules
- otherunitsets - variable for additional unit sets if needed for future use cases
- staticcoursestrings - contains required static course strings needed to complete the course completion rule

Methods

- addStaticCourseString(String s): void - adds a new static course string to the staticcoursestring variable

Parser:

Methods

- mustPassAll(List<String> u): String - creates and returns callista must pass all unit set rule filled with String List contents

    - Return = ”must pass all units in {u[0], u[1]}”
- mustPassX(int n, List<String> u): String - creates and returns callista must pass x of group unit set rule filled with String List contents
    - E.g Return = “must pass 2 units in {u[0], u[1], u[2], u[3]}”
- mustCompleteX(int n, List<String> us): String - creates and returns callista must complete x of group unit set rule filled with String List contents
    - E.g Return = “must complete 1 unit sets in {us[0], us[1]}”
- customParse(int n, String s, List<String>): String - creates and returns a custom callista string based on the method arguments passed, the string s contains a placeholder to indicate where to place the number in the string. This is mainly for extremely complicated custom Callista output and probably won’t be used for this project, but may be useful for future use cases.

UnitSet:

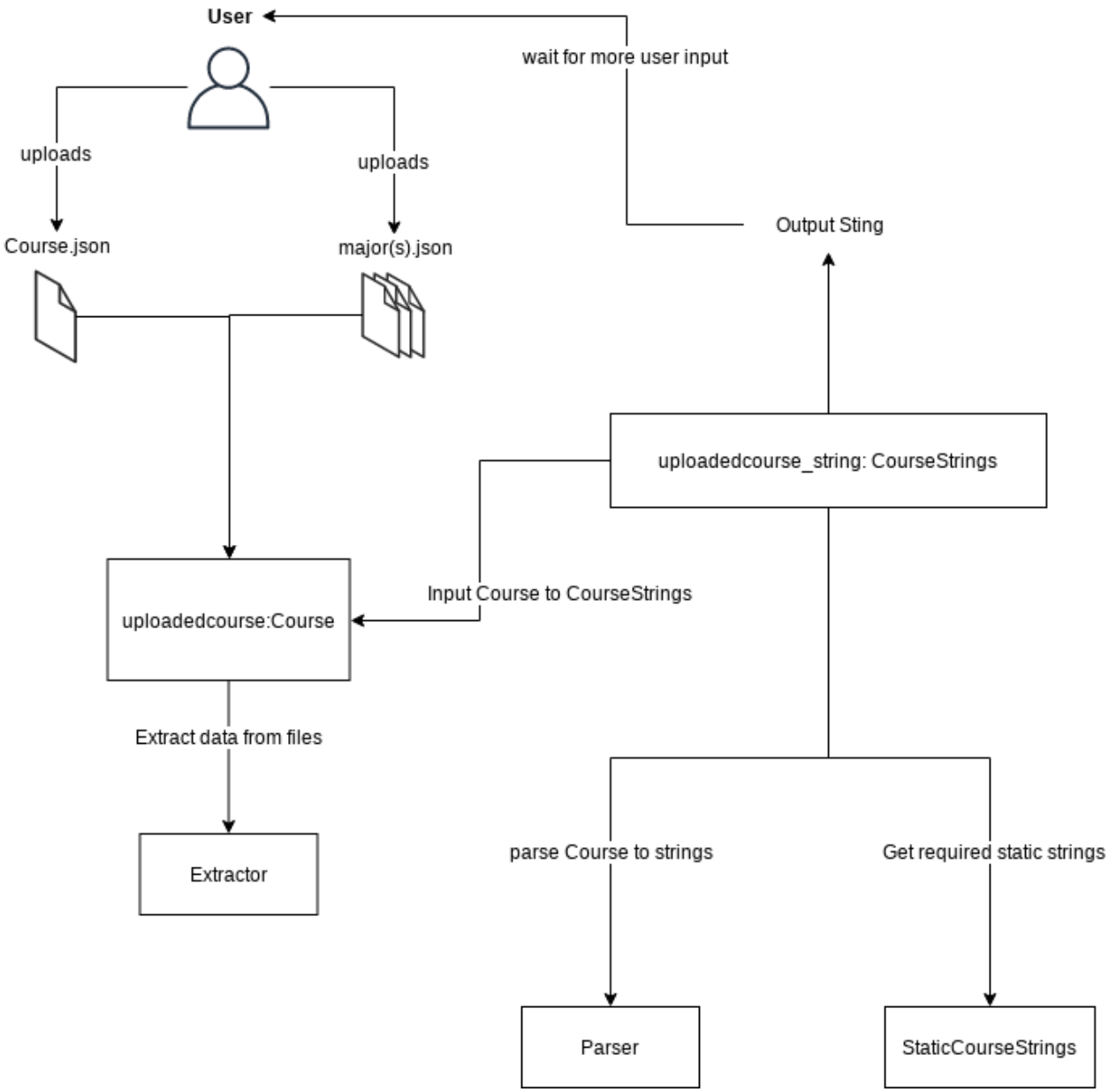
Variables

- code - contains a code for a unitSet String
    - E.g “MJ-CRIB”, “C-B1345-1” etc
- setstring - contains a unit set string
    - E.g “must pass all in {ICT101, ICT105, ICT203}”

Methods

- setUnitSet(String c, String s): void - sets the member variables for a UnitSet object with the arguments given

Data Flow Relationship



Data Dictionary

Display name	Description	Data type	Character length	Acceptable values	Required	Example	Null ?
Academic_item_code	The academic item code uniquely identifies a unit	Varchar	6	[A-Z]*3+[0-9]*3	Y	ICT302, BSC100	N
Major	The Major uniquely identifies a major belonging to a course	varchar	255	*	Y		N
Output	The output variable outputs the callista string from courseloop api	varchar	255	*	Y		N
Course	The course the overseas the major and units	varchar	255	*	No, a course is not required for a major json.		N
String	A string(s) is required to store the data from the JSON files and output to the user	varchar	255	*	Y		Y

Process Design

Process descriptions

The process of the CourseLoop API is to turn JSON CourseLoop files, into a single Callista string. This is achieved by sending the CourseLoop JSON file to our API, which will then parse the information into a format the Callista system can read.

Our algorithms search the CourseLoop JSON to find out what units are inside a major and a course by looping through the maximum size of where the units are stored, it is then concatenated into a string variable which is then displayed via the web browser.

Major CourseLoop To Callista structured English

```
For Each Unit in Major
        IF Unit is the last one
                1)don't concatenate ","
        else
                1)Get Unit code
                2) concatenate onto a string
                3) concatenate ","
End for
Output "Must pass all units in {" + string + "}"
```

Course CourseLoop to Callista structured English

```
For each Major in Majors
                If majors last
                        1)Don't concatenate ","
                Else
                        1) get unit code
                        2) concatenate onto string1
                        3)concatenate ","

For each unit in Specified Electives
                If Unit is last
                        1)Don't concatenate ","
                Else
                        1) get unit code
                        2) concatenate onto string2
                        3)concatenate ","
```

End for

For each unit in Spine

If Unit is last

1)Don’t concatenate “,”

Else

1) get unit code

2) concatenate onto string3

3)concatenate “,”

End for

For each unit in Required units

If Unit is last

1)Don’t concatenate “,”

Else

1) get unit code

2) concatenate onto string4

3)concatenate “,”

End for

Output “Must have completed 1 unit sets in {“ + String1 + “) &”

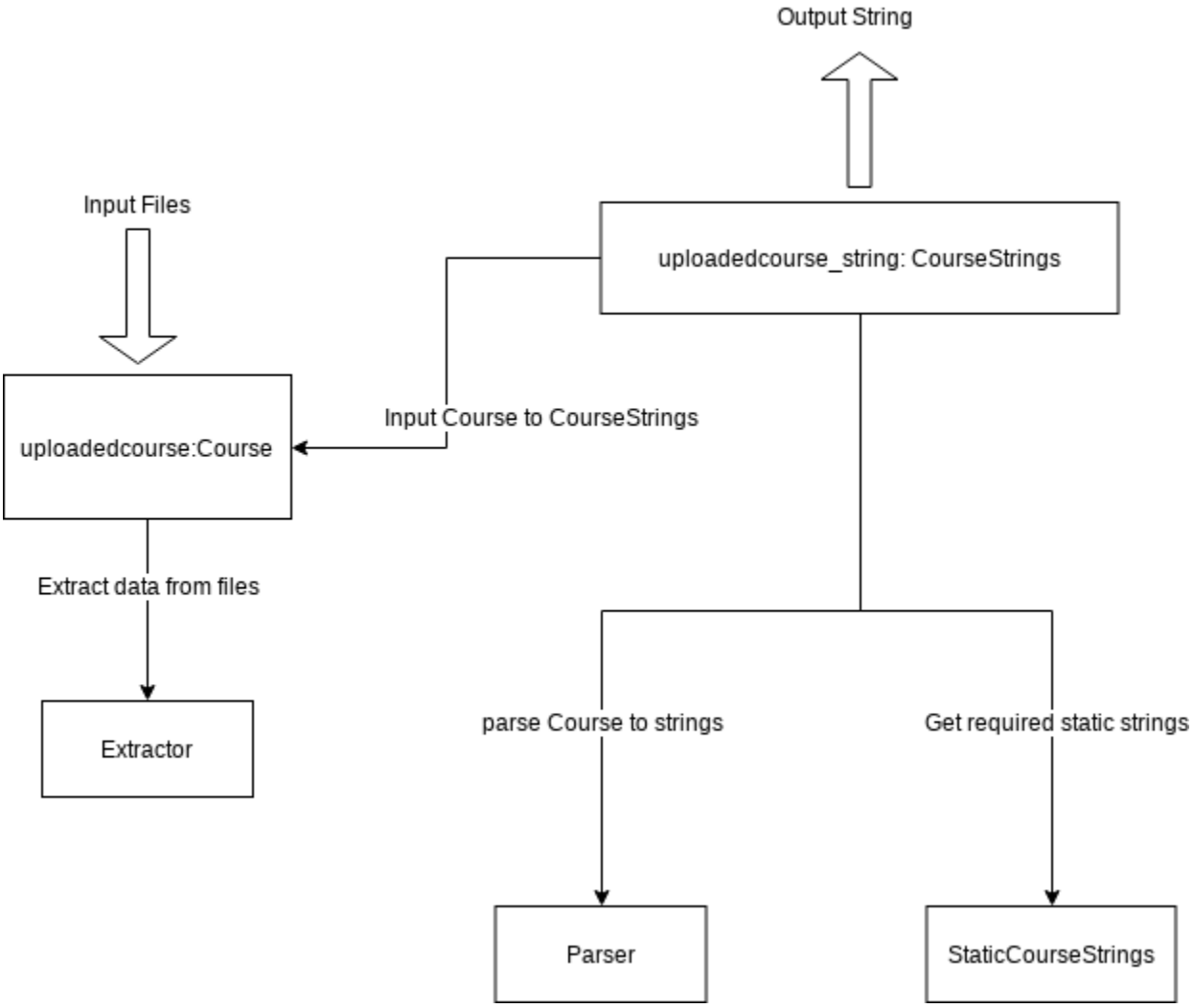
Output “Must pass credit points for course”

Output “Must pass all units in {“ + String4 + “} & ”

Output “Must pass 2 unit(s) in {“ + String2 + “}”

Output “Must pass all units in {“ + String3 + “}”

Data-Flow Model

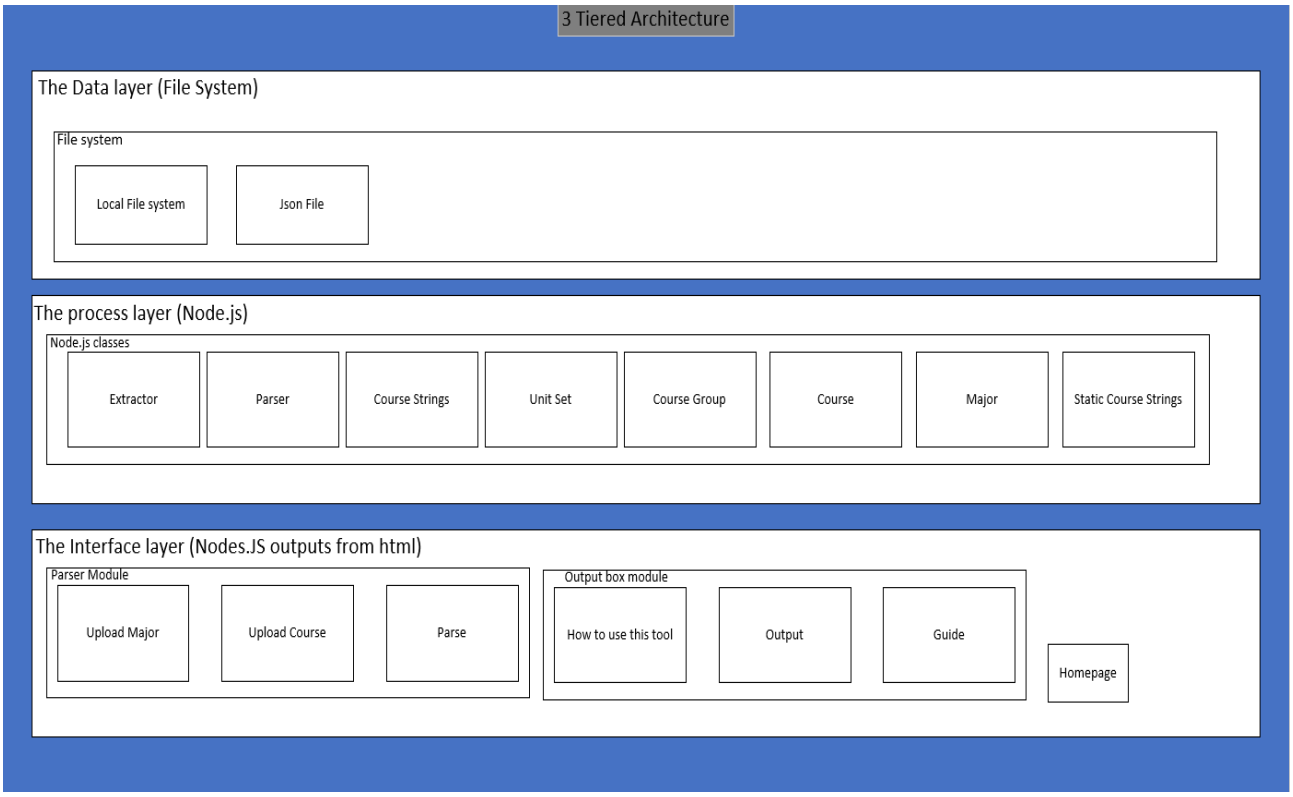


Architecture and Infrastructure Design

Architecture requirements:

The Architecture will follow a 3-tiered design, where the local file system will be uploaded to the course loop API, The API will then parse the data into a Callista string depending on what file has been uploaded via the process layer, using node.js. Once the data has been parsed, it is then sent to the interface layer for the user to copy paste, additional help dialogs are also available on the interface layer.

Architecture Diagram

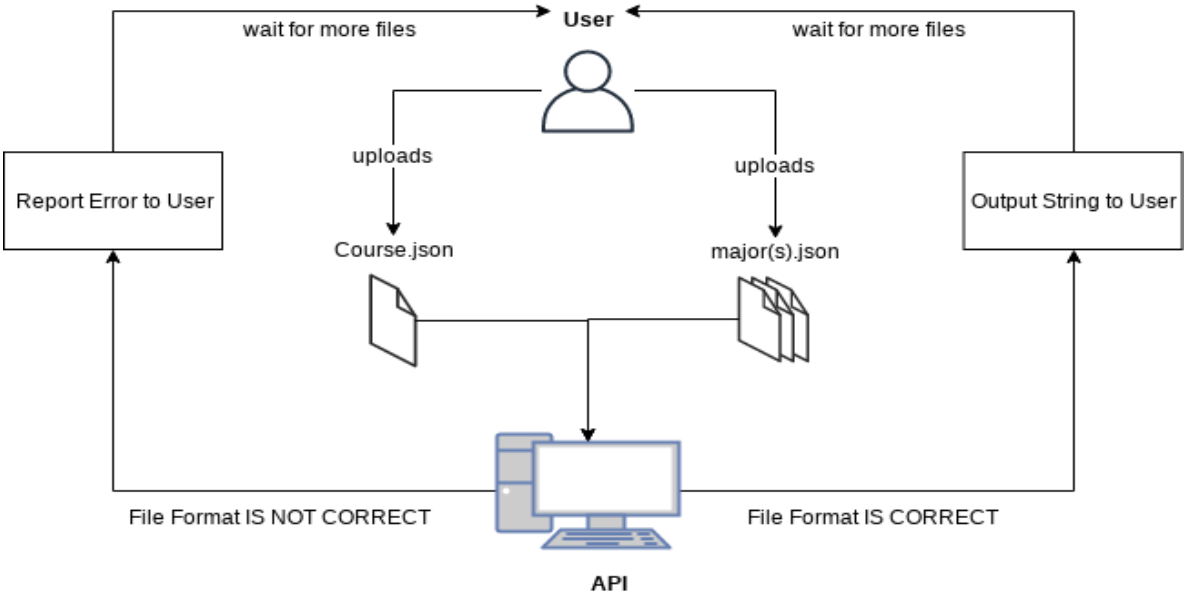


Infrastructure requirements:

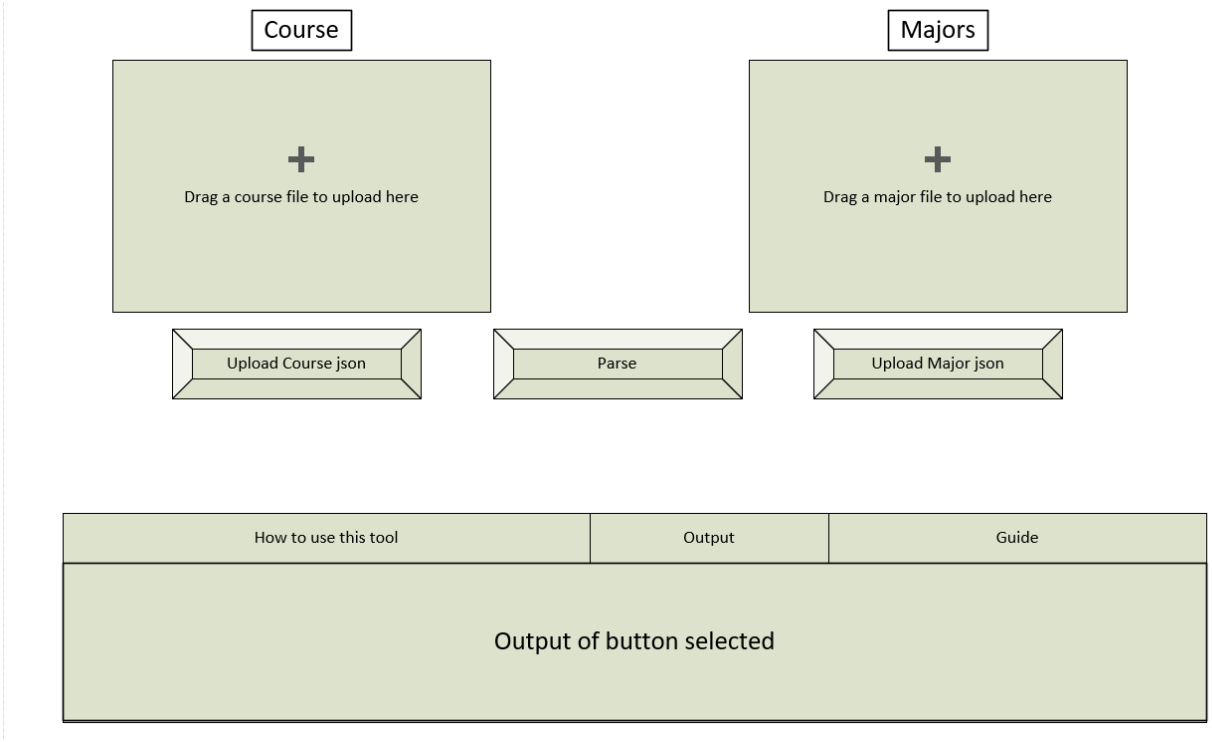
- **Capacity**: The capacity of the system will allow the user to upload the course JSON file and, depending if it is only one major or several ones, the major or majors JSON file(s).
- **Performance**: Performance will be conditioned by the resources used to run this API. As it is coded in JavaScript, it should not consume excessive amounts of resources in order to work.
- **Integration and compatibility**: As pointed above, the API has been designed in JavaScript so the integration and compatibility should not be an issue to any system used nowadays. Nevertheless, the Integration Team would need to check if it is suitable to be used with the corporate software of the university. This also includes the platform to be used, the security policies and the back-up and recovery systems.
- **Scalability**: At the moment, this API has been designed with a single purpose so the scalability is not contemplated.
- **Future proofing**: In the scenario of migration of any of the tools related to this API, it would become not usable for this purpose. However, if the course structure names are updated, or more are added in the future, the algorithm could still extract the data from the JSON file and translate it into Callista syntax.

Interface Design

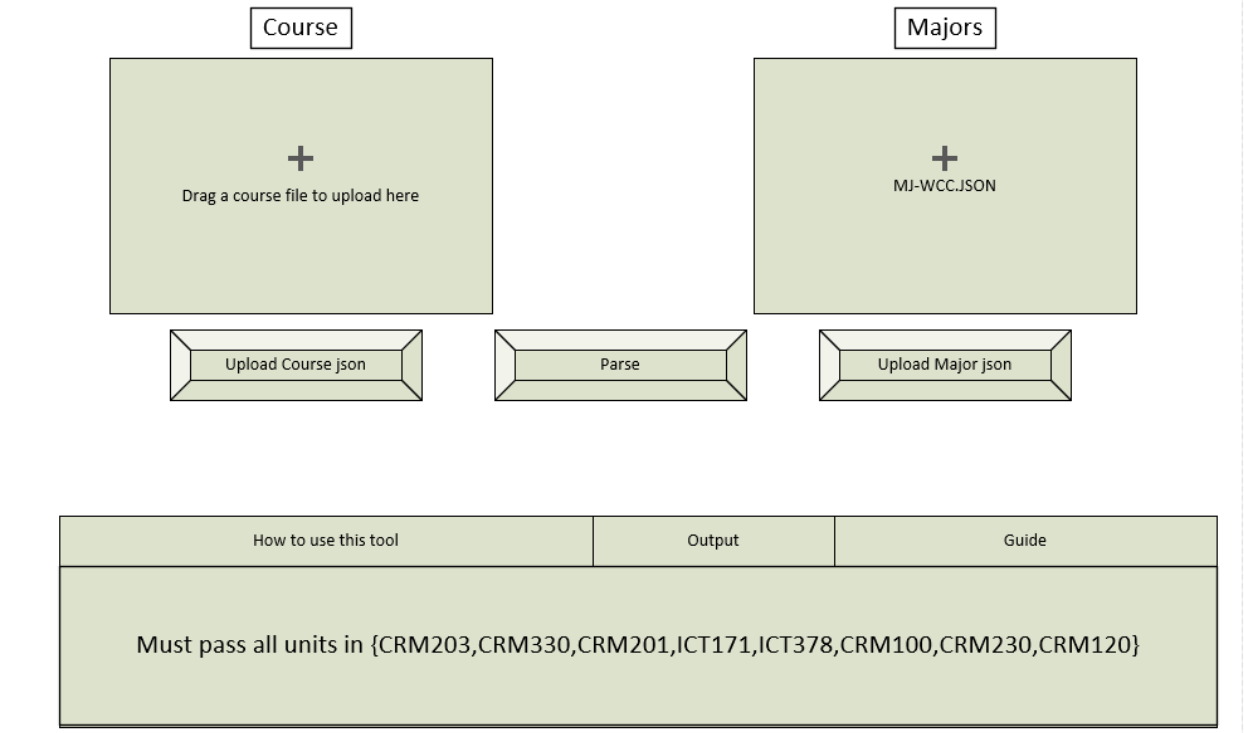
The way a user will interact with the API, will be through a webpage, through a web browser that is provided by Murdoch University (assumption is Google Chrome) . A user will drag and drop a JSON file into the correct drop box to create the corresponding Callista string. There will be 2 drop boxes ranging from: Major, Course. The web page will be easy to use, and will be created as a single page application, this will make it easier and less confusing for a user, as there is no navigation required to create a pending Callista string. The user will then copy the Callista string from the website and paste it into the Callista system. This functionality is easy, simple and quick.



Interface Diagram, resting state



Interface, Major input example



Appendices

Appendix A: Glossary

Glossary	
API	Application Program Interface
String	Sequence of characters
CourseLoop	Curriculum Management System
Callista	Student Management System
JSON	JavaScript Oriented Notation
XML	Extensible Markup Language
Javascript	Main Scripting language used
UML	Unified Modelling Language

Appendix B:
Example rule sets

B1345 Bachelor of Criminology Course Version Completion rule

Must have completed 1 unit sets in {MJ-CRIB, MJ-CRIS, MJ-LEGS, MJ-WCCC} &

Must have completed 1 unit sets in {C-B1345-1} &

Must have completed 1 unit sets in {S-B1345-1} &

Must pass credit points for course

C-B1345-1 (B1345 Bachelor of Criminology Coursecore)

Must pass all units in {CRM306, CRM307} &

Must pass 2 unit(s) in {BMS213, BRD205, CRM202, CRM219, CRM302, CRM310, CRM388, CRM389, LEG100, LEG203, LEG323, LEG391}

S-B1345-1 (B1345 Spine)

Must pass all units in {MSP100, MSP200, MSP201}

MJ-WCCC (White Collar and Corporate Crime Major) Unit Set Completion rule

Must pass all units in {CRM100, CRM120, CRM201, CRM203, CRM230, CRM330, ICT171, ICT378}

MJ-LEGS (Legal Studies Major) Unit Set Completion rule

Must pass all units in {BSL165, CRM100, CRM120, CRM201, CRM203, CRM219, LEG323, LEG391}

Appendix C: Work Breakdown Statement

WB Appendix - Design Document.d