USER MANUAL

Overview

The student administration system has been designed for the department. All the functional requirements have been considered and implemented.

The system was designed with ASP.Net MVC framework. A file storage system in JSON format was used as the data storage.

Benefits

This new system is flexible, fast and makes room for expansion. This was also developed as a plug-and-play system. There is no need for further set up of backend/data storage system.

System Users

There are two main users of the system.

Admin: The admin of the system takes care of the Course, Module, Assessment and Student registration / management. The mapping of Module(s) to Course, Assessment to Module and Assessment score update.

Student: Students are required on the system to select Modules for their course of choice, view assessments, view modules, view profile details.

System Design and Process

The system has been designed to be loosely coupled. Every feature of the system is separate. However, relationships between features are being managed as separate entities. For instance, Course as a feature on the system is standing alone without modules. Only when a module is assigned to the course then the course can have a module. However, the relationship between the course and the module assigned to it is stored as a separate record (JSON format) in the file storage system.

Functional Features:

Course Management: New courses are being registered by the admin of the system. Courses are at first registered without any module. On the system, courses are initially independent of any module. A course can also be made inactive (status that exists on the system to manage the availability of a content).

Module Management: New modules are registered with this feature and existing modules can be made inactive (status that exists on the system to manage the availability of a content). All modules are stored as independent entities, this allows the system to easily assign a module to any course on the system. This thus, makes it flexible to easily upgrade the module(s) of a course in order to meet the required standard.

Student Management: Student's profile are being managed by both admin and the student that owns the profile. Initially, admins register students on the system. However, students are allowed to view their profile details and update some dynamic values on their record. A student can be made inactive (status that exists on the system to manage the availability of a content).

Course-Module Management: This is a function managed by the admin of the system. The process involves adding existing modules to any course. The system is designed this way in order to allow enough flexibility when upgrading any course to meet standard requirements. A module that has been assigned to a course can also be made inactive (status that exists on the system to manage the availability of a content).

Module-Assessment Management: This menu allows the admin to add assessment(s) to a module. Logically, this feature has a way of managing the total score of all assessments assigned to a module to be sure it does not exceed the module total score. An assessment can also be made inactive (status that exists on the system to manage the availability of a content) on the system.

Student-Assessment Update: An admin can supply the score of a student's assessment through this feature. This feature has a way of logically monitoring the score entered to be sure it does not exceed the assessment assigned total score.

Reporting Features:

Course View: Admin and students can view all courses registered on the system.

Module View: Admin view all modules registered on the system. Student is only allowed to view modules mapped to his / her course of study at the point of selecting modules for the course. Student also view all modules he / she has selected for the course he / she is studying.

Student View: Admin can view all students registered on the system. A student can only view his / her profile.

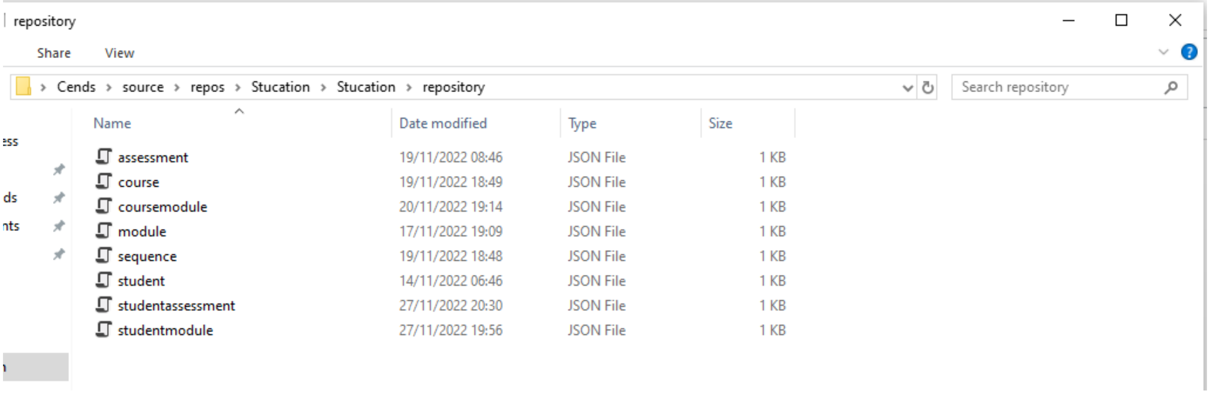
Assessment View: An admin can view all assessments stored on the system. Student can only view assessments assigned to module(s) he / she has selected to for the course he / she is studying. There is another cumulative view of assessments in form of a result. This has the summation of assessments in a module as a total of the module score.

Data Repository

All data storage are in file storage system in the repository folder under the application main domain path. All file content are in JSON Object format. The following are the obtainable storage files that will be created automatically upon new storage creation. For instance, there should be a course.json file stored in the repository folder under the application main domain path. However, on fresh deployment of the system, only sequence.json file would come with the deployment.

On any further deployment or update to the system, the repository folder should always be backed up and then deployed back to the new system build.

See sample figure below of how the repository folder looks like:



Name	Date modified	Type	Size
assessment	19/11/2022 08:46	JSON File	1 KB
course	19/11/2022 18:49	JSON File	1 KB
coursemodule	20/11/2022 19:14	JSON File	1 KB
module	17/11/2022 19:09	JSON File	1 KB
sequence	19/11/2022 18:48	JSON File	1 KB
student	14/11/2022 06:46	JSON File	1 KB
studentassessment	27/11/2022 20:30	JSON File	1 KB
studentmodule	27/11/2022 19:56	JSON File	1 KB

Repository Files Description

Sequence.json: This stores counter and sequence ID for the following:

a. Course
b. Module
c. Student
d. Assessment

At every time when a new course, module, student or assessment is added to the system. A call is made to the sequence.json to retrieve the format and the last count of ID stored for that particular system component (course, module, student or assessment). Once the new course, module, student or assessment is added, the counter: "recentSequence" field is increased by 1.

The sequence.json holds the following fields for each component:

a. SequenceType: holds the sequence type name. e.g. STUDENT
b. InitialSequence: holds the sequence initial value and length
c. recentSequence: holds the last utilized ID for that sequence type

Course.json: Once a new course is added at start, the course.json file would be created (if not existing) and value is stored in it. The fields in the course.json file are a representation of what CourseDto.cs file holds in the Model folder. This is because a list of the CourseDto is created within the system and serialized to json. The value is then saved in the course.json file in repository folder. This means that once there is any change in the Model data transfer object (DTO) CourseDto fields (addition of fields), this will automatically reflect in the course.json file. The current fields of the course.json file are:

a. courseId
b. courseName
c. courseDetails
d. coursePeriod
e. maximumStudents
f. status
g. dateCreated

Module.json: Once a new module is added after deployment, the module.json file would be created (if not existing) and value is stored in it. The fields in the module.json file are a representation of what ModuleDto.cs file holds in the Model folder. This is because a list of the ModuleDto is created within the system and serialized to json. The value is then saved in the module.json file in repository folder. This means that once there is any change in the Model data transfer object (DTO) CourseDto fields (addition of fields), this will automatically reflect in the course.json file. The current fields of the module.json file are:

a. moduleId
b. moduleName

c. moduleDetails
d. term
e. status
f. dateCreated

student.json: Once a new module is added after deployment, the student.json file would be created (if not existing) and value is stored in it. The fields in the student.json file are a representation of what StudentDto.cs file holds in the Model folder. This is because a list of the StudentDto is created within the system and serialized to json. The value is then saved in the student.json file in repository folder. This means that once there is any change in the Model data transfer object (DTO) StudentDto fields (addition of fields), this will automatically reflect in the student.json file. The current fields of the student.json file are:

a. studentId
b. firstName
c. middleName
d. surname
e. dateCreated
f. courseId
g. gender
h. mobileNumber
i. email
j. dob
k. address
l. status

assessment.json: Once a new module is added after deployment, the assessment.json file would be created (if not existing) and value is stored in it. The fields in the assessment.json file are a representation of what AssessmentDto.cs file holds in the Model folder. This is because a list of the AssessmentDto is created within the system and serialized to json. The value is then saved in the assessment.json file in repository folder. This means that once there is any change in the Model data transfer object (DTO) AssessmentDto fields (addition of fields), this will automatically reflect in the assessment.json file. The current fields of the assessment.json file are:

a. assessmentId
b. assessmentTitle

c. assessmentDetail
d. assessmentTotalScore
e. moduleId
f. status
g. dateCreated

<u>coursemodule.json:</u> This is a relationship storage for course and it’s modules. At any point a new module is assigned to a course by the department, the coursemodule.json is updated to have a record of that relationship. The course and module must already be existing on the system before establishing this relationship. This makes it flexible enough to upgrade a course by adjusting it’s modules rather than creating the course afresh. The fields in the coursemodule.json file are a representation of what CourseModuleDto.cs file holds in the Model folder. This is because a list of the CourseModuleDto is created within the system and serialized to json. The value is then saved in the coursemodule.json file in repository folder. This means that once there is any change in the Model data transfer object (DTO) CourseModuleDto fields (addition of fields), this will automatically reflect in the coursemodule.json file. The current fields of the coursemodule.json file are:

a. courseModuleId: This is generated by combining course id and module id
b. courseId
c. moduleId
d. priority
e. status
f. dateCreated

<u>studentmodule.json:</u> This is a relationship storage for students and the modules in the course they are offering. At any point a student selects new module from the list of modules under the course he/she is offering, the module is added for the student in the studentmodule.json. The student and module must already be existing on the system before establishing this relationship. This makes it flexible enough for a student to change module(s). The fields in the studentmodule.json file are a representation of what StudentModuleDto.cs file holds in the Model folder. This is because a list of the StudentModuleDto is created within the system and serialized to json. The value is then saved in the studentmodule.json file in repository folder. This means that once there is any change in the Model data transfer object (DTO)

StudentModuleDto fields (addition of fields), this will automatically reflect in the studentmodule.json file. The current fields of the studentmodule.json file are:

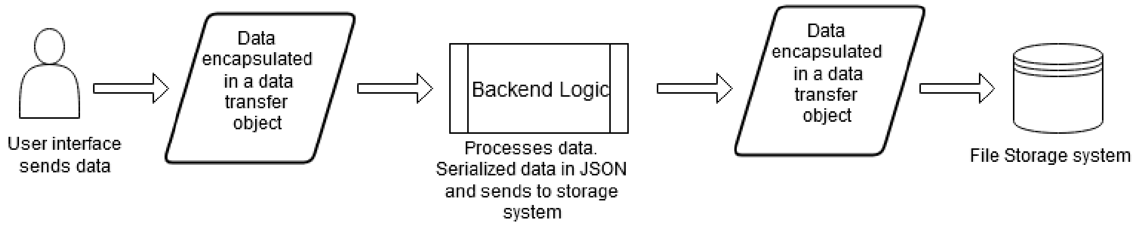
a. studentModuleId: This is generated by concatenating student id and module id
b. studentId
c. moduleId
d. status
e. dateCreated

<u>studentassessment.json:</u> This is a relationship storage for students and the assessments in the module they are offering. The student must obtain a score for this assessment before a record can be found in studentassessment.json. The student and assessment must already be existing on the system before establishing this relationship. The fields in the studentassessment.json file are a representation of what StudentAssessmentDto.cs file holds in the Model folder. This is because a list of the StudentAssessmentDto is created within the system and serialized to json. The value is then saved in the studentassessment.json file in repository folder. This means that once there is any change in the Model data transfer object (DTO) StudentAssessmentDto fields (addition of fields), this will automatically reflect in the studentassessment.json file. The current fields of the studentassessment.json file are:

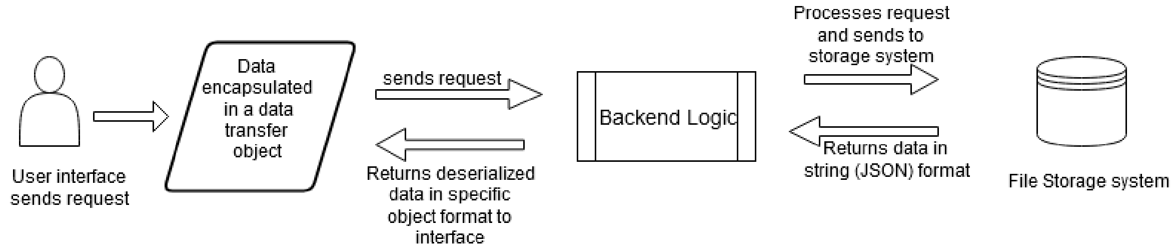
a. studentAssessmentId: This is generated by concatenating student id and assessment id
b. studentId
c. AssessmentId
d. score
e. status
f. dateCreated

System Architecture

Data Storage Process

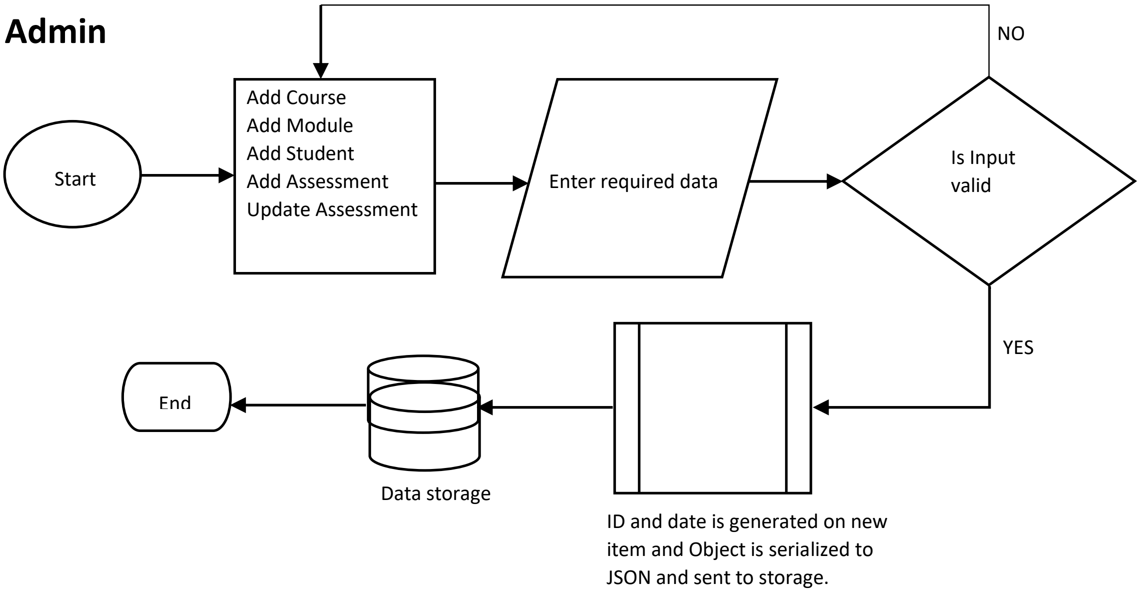

User interface sends data
Data encapsulated in a data transfer object
Backend Logic
Processes data. Serialized data in JSON and sends to storage system
Data encapsulated in a data transfer object
File Storage system

Data Retrieval Process

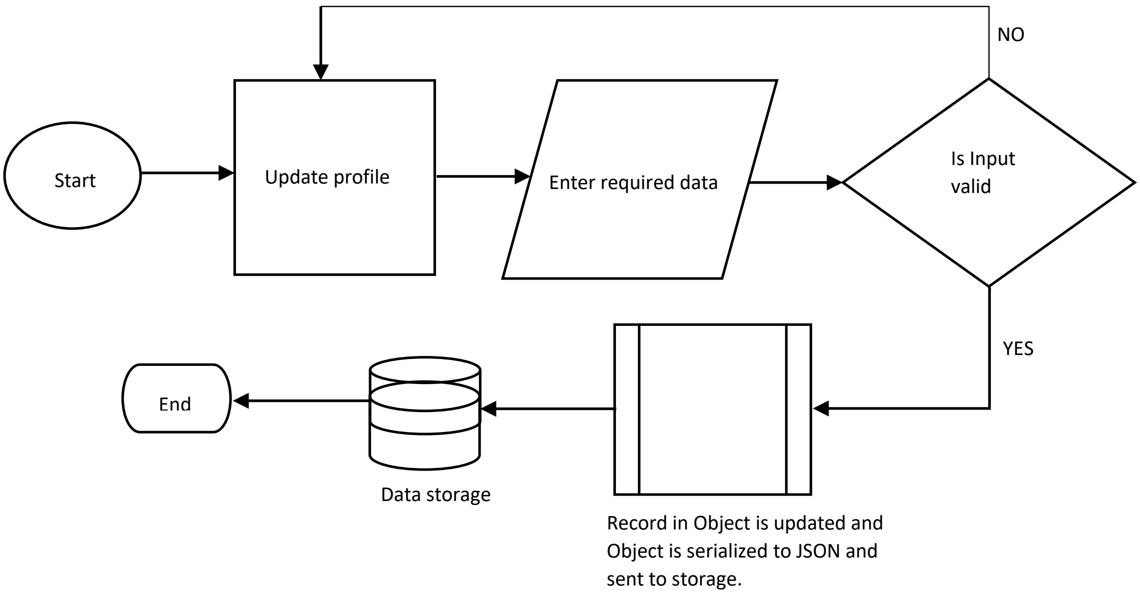

User interface sends request
Data encapsulated in a data transfer object
sends request
Returns deserialized data in specific object format to interface
Backend Logic
Processes request and sends to storage system
Returns data in string (JSON) format
File Storage system

Data Flow Diagram

Admin



Student


NO
Start
Update profile
Enter required data
Is Input valid
YES
End
Data storage
Record in Object is updated and Object is serialized to JSON and sent to storage.