Software Design Specification

Hayden Eubanks

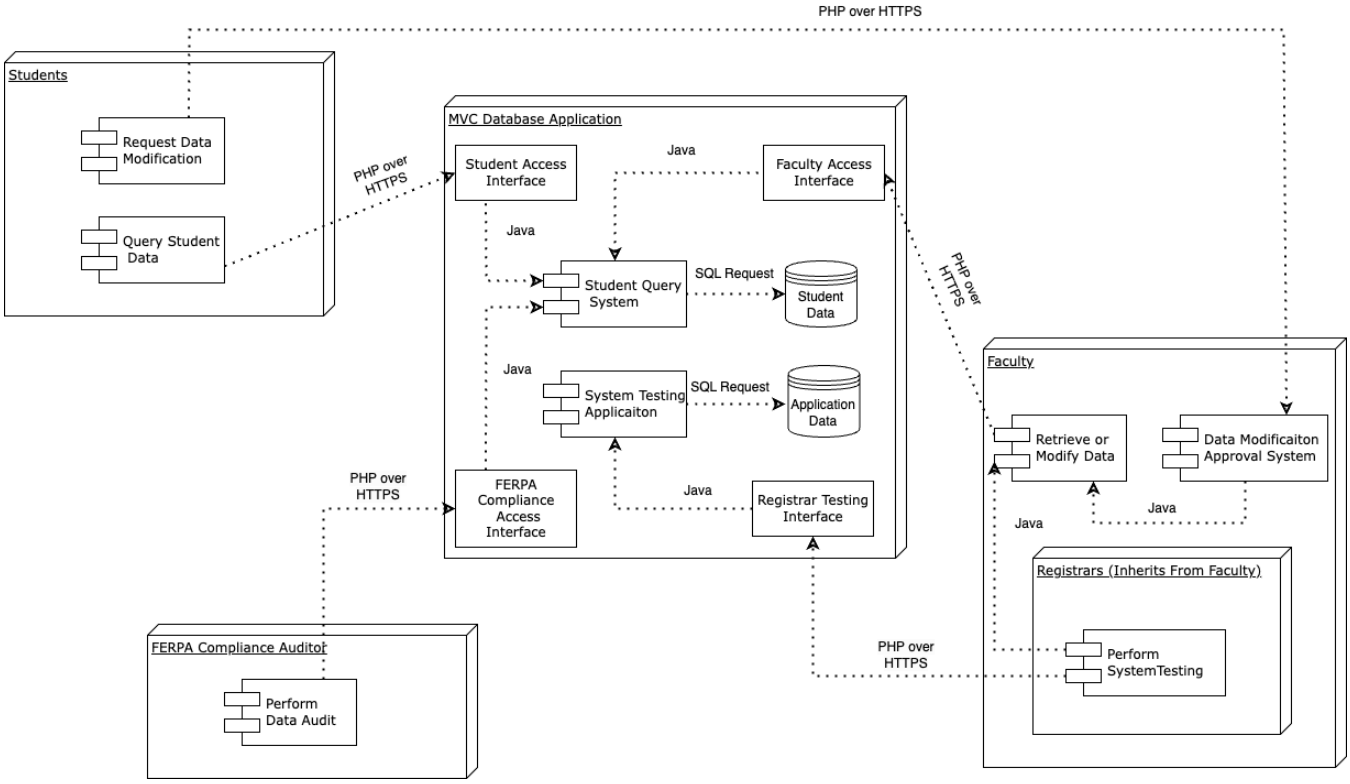
October 2, 2023

Table of Contents

1. System Architecture

1.1 Architectural Design

Overview Diagram of MVC Database Application and Users



A general overview of the components that form the Mountain Valley College (MVC) database application can be seen in the diagram above, and through studying this diagram, a greater understanding of the interconnectedness of components can be achieved. As the diagram needed to model the main system components as well as the protocols and languages used to connect them, a diagram framework similar to that of a deployment diagram was chosen (Microsoft, 2021). However, this diagram can be seen to vary from the traditional deployment diagram by adding additional component features such as class inheritance and the relationship between system function calls and external users who initiate those calls. Modifying the diagram in this way then allows a general

overview of system components to be obtained within the context of the users that initiate those commands. A general understanding of data flow can then be gleaned from this diagram model (Microsoft, 2021) to assist with the understanding of data flow within the system. As mentioned previously, the purpose of this diagram is to provide a general overview of system components and their interconnectedness, and through studying this diagram a shared understanding of these application characteristics can be realized.

One aspect of the system that is clear in the diagram above is that the central component of the application is the database system, with the external entities of students, faculty, registrars, and compliance officers issuing calls to interact with database components. The only exception to this can be seen in the student's ability to issue modification requests to a faculty member which will be sent directly to the faculty member. This is performed in this manner to uphold access control as a faculty member must then pass through authentication measures to gain access to the faculty interface with appropriate credentials (Merkow & Raghavan, 2012). As the primary hub of the system is a database, the connections made to the system are then performed using PHP over the HTTPS protocol to provide additional data protection (Razumov et al., 2023). Internally, each entity will then use the Java and SQL languages. However, this code is not executed directly on the database as an interface for each user class exists to provide the appropriate features for a given entity while protecting database data through the interface layer (Tsui, Karam, & Bernal, 2016). Interfaces are an important element of software design, and in the case of the MVC application, the interface can be used to provide a layer of abstraction between the user commands and underlying implementation to increase system security and better protect student data (Hartson &

Pyla, 2012). In addition to the interfaces, the database can then be seen to hold functionality to query the database and perform system testing gaining access to the additional database of system performance data. This simplified overview of functionality can then be seen to serve the needs of each user class, and with this, each user class contains the functionality to initiate their respective use cases from their end over HTTPS web requests. However, this diagram does not provide a comprehensive overview of the system modules related to data security such as encryption, user validation, firewalls, or access control enforcement, and as such can be generalized within this model to assume security implementations at each entry and exit point. Modeling a system is an essential aspect of software design, and through creating a generalized view of system components a shared understanding of the way components interact can be formed among a development team (Tsui, Karam, & Bernal, 2016).

1.2 Architectural Style(s) & Trade-off Analysis

Styles	Student Query System	System Testing Module	Modification Request System	System Access Interfaces
Pipes and Filters	**1-** This system was not chosen for the student query system as the pipes and filters architecture benefits from developing linear processes with few forks and branches (Tsui, Karam, & Bernal, 2016). As this system is based on a database of information, further design choices that directly address database structure may better represent the system's use case.	**1-** In a similar manner to the query system, the system testing module may be better represented through design architectures built around the relational database structure (Tsui, Karam, & Bernal, 2016). This is due to the primary component of the module being the database of system information. The linear approach provided by the Pipes	**2-** Out of all the systems, the modification request system is best suited to be represented by the pipes and filters system as it is fairly linear in structure, but despite this would provide a relatively poor implementation as a high degree of user input is required to drive the software actions forward. For this reason, event-based approaches may better describe the system	**1-** This component represents the application interface that users will interact as a layer of abstraction between the user input and the database functionality (Hartson & Pyla, 2012). As this component is based on the idea of a graphical interface, an event-driven design structure would provide the architecture that

		and Filters design architecture can then be seen to be insufficient for guiding the needed functionality.	architecture and provide a framework from which development can efficiently occur.	best serves the desired functionality.
Event-driven	**2-** When considering the database querying system, the event-driven approach may not best serve the intended functionality as minimal user input or "events" are triggered by the database querying component. As a need for dynamicity in event handling is not required, other software architecture may be better suited	**2-** As the system testing module also deals with accessing a database in a non-event-driven manner, the event-driven approach to software architecture may not be the most advantageous for this component.	**5-** The event-driven architecture is the ideal candidate for the modification request system as several events will be initiated that the software will be responsible for resolving (Tsui, Karam, & Bernal, 2016). For example, the student user will initiate an event by selecting the function to request data modification. This will then	**5-** Similar to the modification request system, the event-driven architecture can be seen as the optimal architecture model for the database system's application interfaces. These interfaces are based on graphical formats that will process actions based on user input into the graphical interface (Tsui, Karam, &

	to cater to the design of this component.		trigger the event on the faculty system for a faculty member to acknowledge this request. This then will result in the next event of the faculty either processing the transaction further to the database query system or rejecting the modification which will trigger the rejection case events. From this, the event-driven nature of this design is clear and could be seen as the best-fit architecture for the software component.	Bernal, 2016). This is the ideal use case for the event-driven architecture as different user inputs into the interface, such as clicking buttons, will trigger different events with different functionalities. The event-driven architecture can then be seen as the best-fit architecture to fulfill the requirements of this system component.

Client-Server	**4**- As the student query system will be accessed through a web application, the client-server model could be a reasonable solution for implementation that would fulfill the objectives of the software in implementation (Tsui, Karam, & Bernal, 2016). In fact, if a single approach was needed to be chosen for all of the components, the client-server might be seen as the best fit for the application as a whole. However, the web-based nature of the application is a	**3**- The system testing model, in a similar manner to the query system, could be implemented with a client-server architecture, but as the primary functionality of this component centers around a database the database-centric design architectures may better fulfill the goals of implementation. However, the only users that will access this portion of the web application are the registrar users, and as this is the smallest system user group, the client-server	**4**- The client-server model could be seen as a strong candidate for the modification request system as a web application is used to interact with the component inferring a design that this architecture could facilitate (Tsui, Karam, & Bernal, 2016). However, as this component also revolves around a series of events, I have instead opted to implement this component through the event-driven approach.	**4**- Like the other components, the database interface component could have been implemented with the client-server architecture and achieve full functionality, but the nature of asynchronous events inherent in graphical interfaces points to the event-driven approach as the optimal candidate for this component.

	generalization of the entire system and as this individual component is primarily centered around a database, the database design architectures may better represent the desired functionality.	architecture can be further be seen as suboptimal for this component.		
Model-view-controller	**2**- This architecture is primarily for a GUI component that needs to display multiple views of data at once (Tsui, Karam, & Bernal, 2016) and as such is not relevant to this component. With the given system specifications, only a single view of data will need to	**2**- While the system testing module may require the GUI to display multiple pieces of data at a given time, the primary functionality of this component revolves around retrieving data from the system database, and as such, this architecture is not	**1**- The model-view-controller architecture is the least-best-fit architecture for the modification request system as this component functions in a relatively linear manner with never more than a single piece of data being viewed at a time. For this reason, the	**3**- The graphical user interface implemented to provide an abstraction of the underlying database could partially be implemented through this design architecture. However, the event-driven approach is better suited to handle the

	be generated at a time indicating that this design architecture would not fulfill the needs of the component, making it a bad choice for design.	the best fit for the component's implementation.	model-view-controller architecture is not an ideal candidate for implementation.	varying use cases posing this system, and as such a lower comparative rating has been given to this architecture.
Layered	**3-** The layered architecture could potentially be implemented for this component but would not be representative of the best-fit architecture as interactions with the database exclusively operate on the bottom layer of development. However, if the entirety of the system were considered, the layered	**3-** The system testing module could additionally be seen as an inadequate candidate for implementing the layered architecture as its primary operations are focused exclusively on the lowest level of implemented abstraction being the database operations themselves. However,	**1-** The layered architecture has been chosen as one of the least-fit architectures for the modification request system component as this component functions around the interaction between two user groups facilitated by the system software. This highlights a use case that does not follow the layered architecture model and as	**4-** The layered architecture could be seen as an ideal candidate for user interfaces as an interface is a layer of abstraction that performs similarly to a layer between two components (Tsui, Karam, & Bernal, 2016). However, the event-driven approach has been chosen as the best-fit

	architecture could be seen to provide the desired abstraction between the database and its users (Tsui, Karam, & Bernal, 2016).	considering implementing the entire system with a single architecture could then point to the layered architecture as a strong candidate for use.	such would be sub-optimally implemented with the layered approach.	architecture for this component as the dynamic nature of choosing actions could best be described through that approach.
Database-centric	**4-** As this function centers around a database and performs queries on the database, the database-centric design architecture is a really strong implementation choice. However, as this application is additionally a web application taking user input the three-tiered approach provides the added functionality that indicates	**4-** Like the database querying component, the accessing of the secondary database also highlights the database-centric design architecture as a strong candidate for implementation. However, the three-tiered approach provides the added event-driven functionality present within web applications	**2-** As this component does not directly interact with a database, this architecture would not be best suited for its implementation. Additionally, the event-driven interaction inherent in this application highlights the event-driven architecture as a better-suited candidate.	**3-** While this component does create an interface between users and a database, the database is not at the center of this components functionality. Because of this other solutions should be examined and implemented as better-fit design architectures.

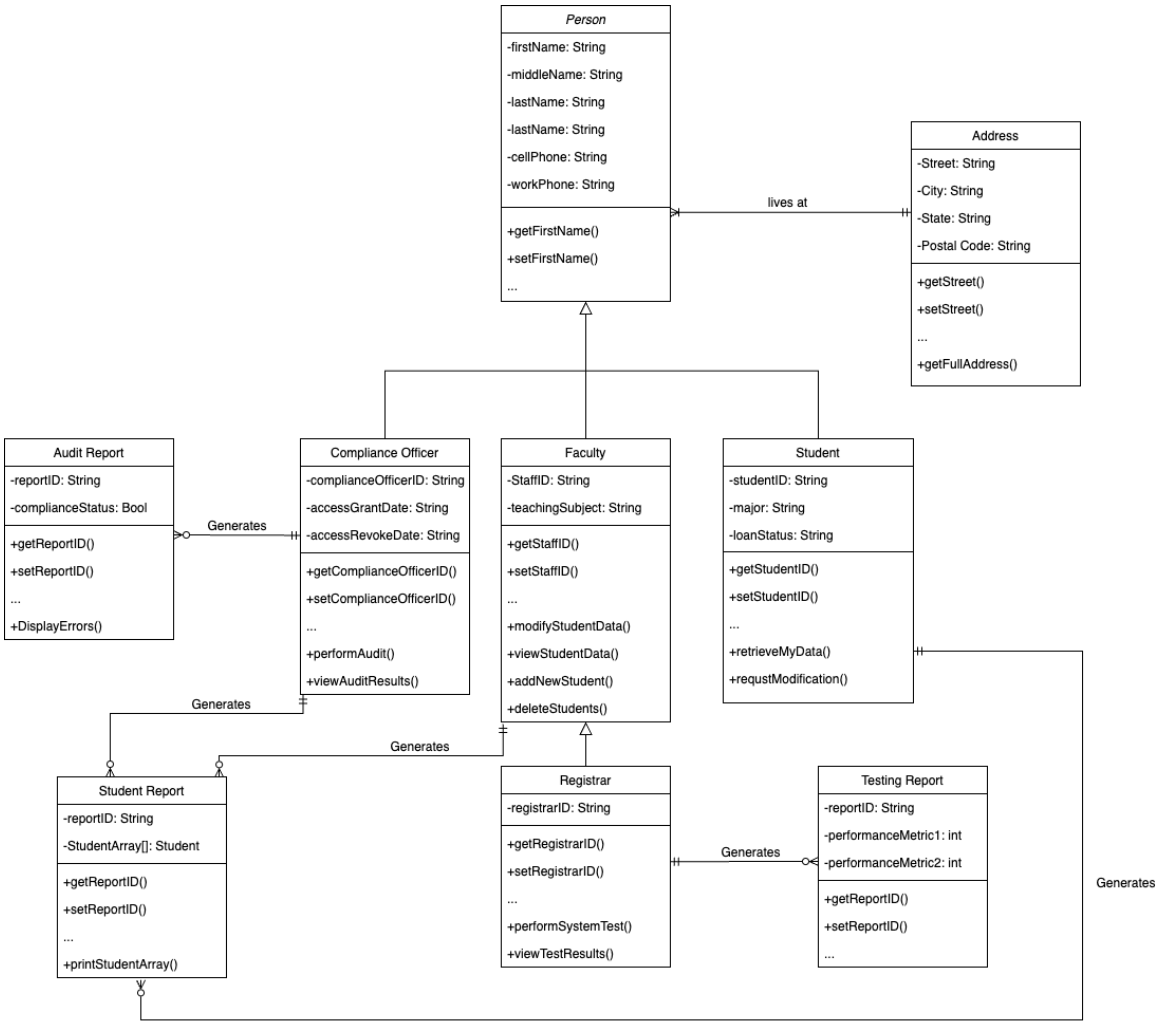
	that architecture is a better fit than this one.	highlighting that architecture as a better-fit solution.		
Three-tier	**5**- The three-tiered approach is the ideal candidate for the database querying system as this component interacts with the database through the context of a web application. This then highlights a need for relational database functionality combined with client-server interactions highlighting the optimal use case for implementing the three-tiered architecture (Tsui, Karam, & Bernal, 2016).	**5**- Additionally, the system testing module also interacts with the database in the context of client-server interactions and as such would require the added functionality provided by the three-tiered architecture. This is a marginal improvement over the database-centric architecture but can be seen to more accurately model the interactions happening within the software component.	**4**- As this component is adjacent to the database and remains present within the web application, the three-tiered approach could be used to implement it. However, other architectures such as the event-driven architecture could be better utilized to highlight this component's implementation indicating that another approach should be chosen.	**4**- Similar to the modification system, the system interfaces could be implemented with the three-tiered architecture but point to the event-driven approach as a better fit due to the dynamic nature of human interaction within a web application graphical interface.

Score from 1-5 with 1 the least fit and 5 the best fit

From the table above, it can be seen that a combination of the three-tiered and event-driven architectures was chosen to implement the design of the MVC database web application. This approach was taken as choosing a best-fit architecture for each component could better ensure the quality of the system on the component level, therefore increasing the quality of the system as a whole (Sobhy et al., 2021). The three-tiered architecture was chosen to be implemented for the components interacting with the database directly as the database is accessed through a web application indicating that it could benefit from the added client-server functionality present in the three-tiered architecture (Tsui, Karam, & Bernal, 2016). This means that the business logic will be performed in one place on an intermediary layer between the database and the users and as such can provide a layer of abstraction that hides the underlying implementation and increases system security (Merkow & Raghavan, 2012). The components chosen to implement the three-tiered architecture include the querying systems for student data and system data as these components are primarily centered around the database components. If budget constraints would require a single architecture implementation to be chosen, the three-tiered approach would be the primary candidate for the implementation of the system as a whole. In contrast to the three-tiered architecture chosen for the database adjacent components, the event-driven architecture was chosen for the modification request and system access interface systems as they require dynamic event-based interactions. The event-driven architecture is optimal for systems such as graphical interfaces that allow the flow of actions to go in many differing directions based on the user's input to the system (Sobhy et al., 2021). This dynamic choice is highly present in these two components, and as such the event-driven approach can be seen to best cater

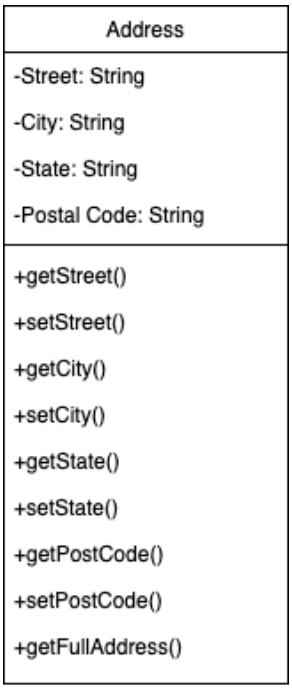
for the needs of these components. Choosing an architecture that best fits the requirements of a system can be a difficult task, but the proper application of system architectures in design can vastly increase the effectiveness of an implementation (Sobhy et al., 2021) making it a worthwhile endeavor to examine the benefits of varying architecture styles.

2. Software Design



2.1 Class: Address

2.1.1 Class Description



2.1.2 Attributes

Attributes	Descriptions
Street	The street attribute refers to the street address of the person and should include the street name and house number information. This field can accept alphanumeric data up to a length of fifty characters including capitalization.
City	The city attribute refers to the city name that the street address resides in. Only alphabetic characters are accepted for this field and the field can have a max length of fifty characters.

State	The state name must be a valid U.S. state and be written in the format of USPS state codes. For this reason, a maximum of two characters can be entered into the field and these characters must represent a valid state.
Postal Code	Similarly, the postal codes will represent the postal code for the address and be written in the USPS postal code format. This means that exclusively numeric values will be accepted and the max length for a data entry is five characters.

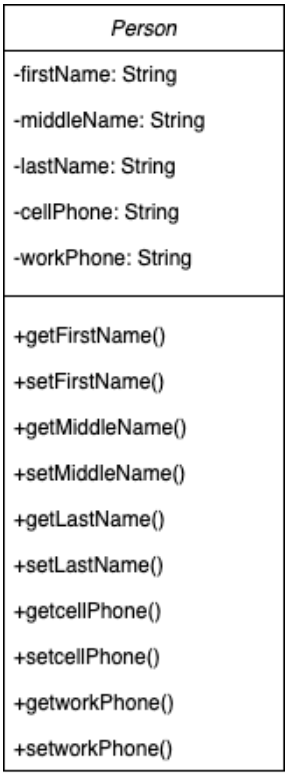
2.1.3 Methods

Methods	Descriptions
getStreet()	The getStreet() method will be used to access the value stored in the street attribute. This will provide an interface for safe access to the private street data member and this principle will remain consistent across all get methods.
setStreet()	The setStreet() method can then be used to modify the value stored in the street attribute. As such, the input data for this field must be alphanumeric and comply with the data input restrictions.
getCity()	The getCity() field can be used to retrieve the value stored in the city data member. Further, this get method will provide safe access to the private data member upholding the principle of abstraction in this class's design.
setCity()	The setCity() method can then be used to set the value of the city data member and the input value will be validated before modification occurs.

getState()	The getState() method will then return the two-character USPS state code for the state data member. Further, this public get method will provide safe access to the private state data members.
setState()	The setState() method can then be seen to allow the modification of the value stored in the state data member. The validation used upon initial data entry will be enforced within this method to ensure the proper data value is stored.
getPostalCode()	The getPostalCode() method can be used to retrieve the value of the postal code field stored within the class object. One element worth noting is that this field returns a string and not an integer value as the primary context that postal codes will be used for would highlight the advantage of a string value over an integer value.
setPostalCode()	The setPostalCode() method can then be used to validate and modify the value stored in the postal code data member.
getFullAddress()	The getFullAddress() method can then be seen to return the full address for a given person object. This method accomplishes this task by calling each of the other get methods for individual data items but allows this to be accomplished through a single function call. This will be extremely useful in coding implementations that require the full address to be returned which will align well with the context of the MVC application.

2.2 Class: Person

2.2.1 Class Description



2.2.2 Attributes

Attributes	Descriptions
firstName	The first name field will be used to represent the first name of a person and as such, only alphabetic characters will be accepted as valid input. Further, the max field length for a first name is 25 characters and this will be validated upon data entry.
middleName	Similarly, the middlName field will then be used to represent the person object's middle name. This field is allowed to be empty as not every person will have a middle name. This field only accepts alphabetic characters and has a max length of 25 characters.

lastName	The last name field is then mandatory and will be used to represent the last name of a person object. The last name will only accept alphabetic characters and must be 25 characters in length at a maximum.
cellPhone	The cellPhone data field will then be used to store the cellPhone number of a person. This field is mandatory so if a user only holds a single phone number it must be entered into this field. The field should only accept numeric input and be entered in the three-digit area code format. With this, the max length for a cell phone number value should be ten characters in length.
workPhone	The workPhone data field is then similar to the cellPhone field, but this field is allowed to be left empty. This field will only accept numeric values and should be of a max length of ten characters. Additionally, the three-digit area code format should be followed for data entered into this field.

2.2.3 Methods

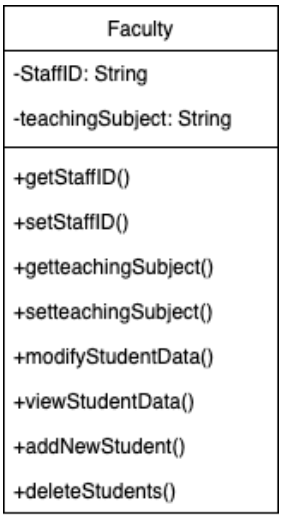
Methods	Descriptions
getFirstName()	The getFirstName() method can be used to retrieve the private data member stored in the firstName field. As this is a get method, the abstraction provided can protect access to the private data members and as such upholds data security best practices.
setFirstName()	The setFirstName() method can then be seen to allow the modification of values within the firstName field.
getMiddleName()	The getMiddleName() method can be used to access data stored within the middleName data field. The niddleName field is allowed to be

	empty, and as such this method should return a message indicating that the field is empty when this is true.
setMiddleName()	The setMiddleName field can then be used to set or modify the value stored in the middleName field. This data will be validated to ensure that all restrictions on data entry are met.
getLastName()	The getLastName() method can then be seen to return the data stored in the lastName field. This data is stored in a private member, and as such the access method must be used to safely interact with the data stored in that field.
setLastName()	The setLastName method can then be used to modify data stored within the lastName field. All data validation will be performed within this method to ensure that only valid data is updated into the lastName field.
getCellPhone()	The getCellPhoneMethod() can then be used to return the cellPhone value stored in this object's cellPhone field. This will provide safe access to the data field through the get method.
setCellPhone()	The setCellPhone method can then be used to update the value in the cellPhone field while validating the input to ensure that is of the correct type, format, and length.
getWorkPhone()	The getWorkPhone() method can be implemented to return the value stored in the workPhone field. As this field can be empty, a message

	that indicates the field is empty will be returned when no value is stored within this field.
setWorkPhone()	The setWorkPhone method can then be implemented to set the value of the workPhone data member. This function will perform data validation to ensure that all data restrictions are met upon modification.

2.3 Class: Faculty

2.3.1 Class Description



2.3.2 Attributes

Attributes	Descriptions
StaffID	The staffID field will represent a unique value through which the staff member can be identified. This value will be automatically generated upon the creation of the object but should be unique among the staff member objects. This field can be of a

	maximum of 9 characters and must follow the social security numbering format. Additionally, this field can only contain numeric characters.
teachingSubject	The teaching subject field will then store a string value of the subject that the faculty member teaches or is linked to. A list of valid subjects will be stored, and this field compared against it to ensure that only valid subjects are entered into this field.

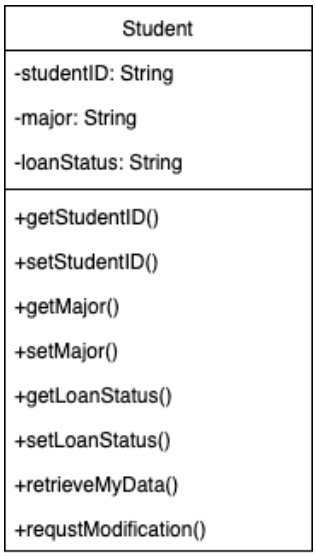
2.3.3 Methods

Methods	Descriptions
getStaffID()	The getStaffID() function can be executed to access the StaffID for this staff member object. This value will be returned to the user, and this function will be the only way to access that private data member.
setStaffID()	The setStaffID() function can then be used to modify the value of the staffID field. However, this field must be unique among the staff objects and as such, this field will be validated to ensure that this characteristic is met.
getTeachingSubject()	The getTachingSubject() method can then be used to return the value of the teachingSubject field of this object.
setTeachingSubject()	The setTeachingSubject() method can then be used to modify the value stored in the teachingSubject field and with this, the appropriate data validation will be performed.
modifyStudentData()	The modifyStudentData() method will then be used to call the set methods of student values through access to the student database. This

	method can then be seen to provide an interface through which the modification of student values can be safely performed by faculty members. This field will take the field(s) to be modified and the new value(s) as input, and then will perform the appropriate modification returning the confirmation of method success.
viewStudentData()	The viewStudentData() method can then be used to execute a query on student values using the object type of faculty as a qualifying criterion for performing the search. This limits advanced searches to faculty members maintaining the principle of access control in the system design.
addNewStudent()	The addNewStudent() method can then be used to add a new student to the student database. This takes the mandatory student fields as input and creates a new database entry based on those values.
deleteStudents()	The deleteStudent() function can then be used to delete student entries from the database. This will require a commit action that will be triggered by this function. This method will take a student's data as input and then will delete the student entry and return the confirmation of method success.

2.4 Class: Student

2.4.1 Class Description



2.4.2 Attributes

Attributes	Descriptions
studentID	The studentID field can be used to store the unique ID used to distinguish differing students. This will assist the scenario in which two students with the same name enroll in the college's program. This field will only accept numeric data and must be a maximum of nine characters long in social security numbering format.
major	The major field will be a string value of the student's major. This field will only accept alphabetic characters and must be a maximum of four characters long as the college’s major code format must be followed. This value will additionally be verified against the college’s list of majors to ensure that the value entered is a valid major value.

loanStatus	The loanStatus field can then be seen to hold a three-character long loan status code. A list of valid loan status codes will be maintained by the university and as such, validation against this list of codes will be performed. Only alphabetic characters in the three-character format will be accepted for this field.

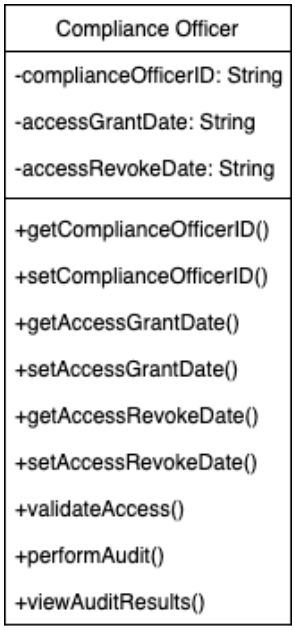
2.4.3 Methods

Methods	Descriptions
getStudentID()	The getStudentID() function can be used to return the value of the studentID and in doing so will uphold the principle of abstraction in object-oriented design.
setStudentID()	The setStudentID() method can then be seen to allow for the modification of the data held in the studentID field. This method will validate all data entry to ensure it is compliant with the field's data restrictions.
getMajor()	The getMajor() method can be used to return the value of the major code stored within the major field of this object.
setMajor()	The setMajor() method can be utilized to modify the major code stored in the major field of the student object. This field will then be validated and verified against the list of valid major codes before updating the data field.
getLoanStatus()	The getLoanStatus() method can then be used to return the value of the loan status field. This will return the loan status code stored in the field.

setLoanStatus()	The setLoanStatus() function can be used to modify the value stored in the loan status field of the student object. The data entered for modification will then be validated to ensure that only the correct data is inserted into the data field.
retrieveMyData()	The retrieveMyData() function can then be used to return the student's data from the student database. This field is different from that of the faculty counterpart as only a single student's data will be returned matching the student that initiated the request. No other student's data must be accessible through function calls to this method.
requestModification()	The requestModification() function can then be used to request modification of the student's data values stored in the database. This request will be sent to the faculty verification system where a faculty member is then able to review the request.

2.5 Class: Compliance Officer

2.5.1 Class Description



2.5.2 Attributes

Attributes	Descriptions
complianceOfficerID	The complianceOfficerID data field will be used to store a unique ID value through which a compliance officer can be uniquely identified. This value will be automatically generated upon the creation of a class object. This field will only accept numeric data and a maximum length of nine characters in social security number formatting will be accepted for validation.
accessGrantDate	The accessGrantDate field will note the date that the compliance officer was granted access to the system. This field will take a string value of the date and will be used for validating compliance officer access to the system.

accessRevokeDate	The access revoke date can then be represented in the accessRevokeDate field and will be used to validate if the compliance officer maintains access to the system. Like the grant date, the revoke date will be stored as a string value of the date that access rights will be revoked.

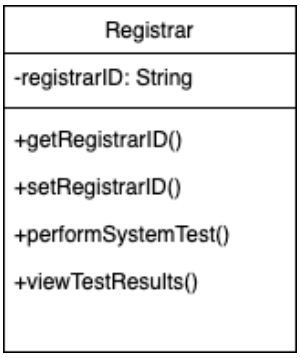
2.5.3 Methods

Methods	Descriptions
getComplianceOfficerID()	The getComplianceOfficerID() method can be used to return the compliance officer ID. The get method is used to provide a layer of abstraction between the private data members and user interface and as such will provide increased security over data values.
setComplianceOfficerID()	The setComplianceOfficerID() can then be used to modify the value of the compliance officer ID. This value will be validated according to the restrictions for data entry in this field.
getAccessGrantDate()	The getAccessGrantDate() method will return the value of the access grant date to the user. This will return the string value stored in the data field for the user to see. This function is also used in the validation function seen below.
setAccessGrantDate()	This method can be used to modify the access grant date field in situations where the grant date has not already passed. This can allow the college to grant access to a compliance officer earlier than intended. This value must be validated to ensure the formatting is compliant with data restrictions for entry within this field.

getAccessRevokeDate()	The getAccessRevokeDate() method can then be seen to return the revocation date value stored in the accessRevokeDate field. This get function will allow for data protection as direct access to the private data member is not given to the system user.
setAccessRevokeDate()	The setAccessRevokeDate() function can then be used to modify the value stored in the accessRevokeDate field. This can allow MVC to grant a compliance officer more time to perform their compliance audit if so required. Further, it can be used to immediately remove access by setting the revocation date to a date in the past.
validateAccess()	The validateAccess() function will then call the get functions to view the dates that access was granted and will be revoked to ensure that the current date is within that range. If the date is not within that range, the compliance officer will not be granted access to system resources.
performAudit()	The performAudit() function can then be executed to generate a report of compliance-officer-related metrics. This report can then be used to validate the university's compliance with FERPA regulations.
viewAuditResults()	The viewAuditResults() method can then be used to view a report generated by the compliance officer. This method can accept reportIDs as input and will then display the appropriate reports as output.

2.6 Class: Registrar

2.6.1 Class Description



2.6.2 Attributes

Attributes	Descriptions
registrarID	The registrarID field will be used to store a unique ID for the registrar faculty member through which they can be uniquely identified. As the registrar's inherit from the faculty class, this is the only additional attribute that must be added. This field will only accept numeric values and additionally will only accept up to nine characters in social security numbering format.

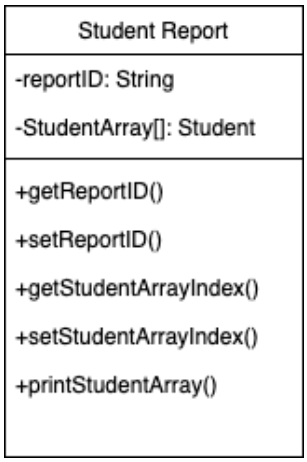
2.6.3 Methods

Methods	Descriptions
getRegistrarID()	The getRegistrarID() function can then be used to return the ID of the registrar class object. This function will provide protection to the underlying data by enforcing a controlled interface through which users can access the data.

setRegistrarID()	The setRegistrarID() function can then be used to modify the value of the registrarID. The proper validation associated with this field will then be enforced to ensure that only valid data values are accepted into this field.
performSystemTest()	The performSystemTest() method can then be used to initiate a test of system components and resources and then generate a report of the findings. This function can take specific component names as input for performing the test, but if no values are given a full system test will be performed.
viewTestResults()	The viewTestResults() function will then be used to review the reports of test results generated by the performSystemTest() function. This function can take reportIDs as input and then will display the reports for the IDs entered into the function.

2.7 Class: Student Report

2.7.1 Class Description



2.7.2 Attributes

Attributes	Descriptions
reportID	The reportID attribute will create and store a unique ID through which each student report can be uniquely identified and referenced. This field will only accept numeric data up to nine characters in length and will be generated upon an instance of the class being created.
studentArray[]	The studentArray can then be seen as an array that will store student objects held within the report. This can then allow a report to be inspected as well as the students that make up that report. As roughly 500 students currently attend MVC a maximum report length of 500 students can be generated to allow a report size sufficient to display all students.

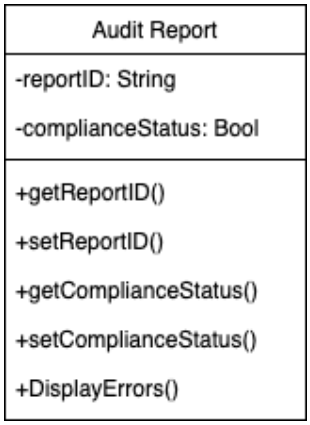
2.7.3 Methods

Methods	Descriptions
getReportID()	The getReportID() method can then be used to return the reportID of the given class object. The get method is used to provide a layer of data protection as the private class member cannot be accessed directly.
setReportID()	The setReportID() method can then be used to modify the value of the reportID. This updated value must maintain the feature of being unique among student reports and as such this will be checked in validation before the update occurs.

getStudentArrayIndex()	The getStudentArrayIndex() method can be used to return the student object to a specified index within the array of student objects. This function can then be seen to take an index value as input and as output will return the student object at that index.
setStudentArrayIndex()	The setStudentArrayIndex() can then be seen to modify the value of a student object at a given index. To accomplish this, an index value will be passed to the function as input in addition to a new student object, and then the value at that index will be updated.
printStudentArray()	The printStudentArray() function will then be used to display the report and will print to the screen every student in the student array.

2.8 Class: Audit Report

2.8.1 Class Description



2.8.2 Attributes

Attributes	Descriptions
reportID	The reportID field for the audit reports will represent a unique report ID through which audit reports can be distinguished and identified. This field must be unique among report IDs and as such validation will enforce this characteristic. Further, this field can only accept data up to nine characters in length of exclusively numeric data values.
complianceStatus	The complianceStatus is a Boolean value that will be set to true to represent compliance and false to represent noncompliance. As the report must prove that the system is compliant, this value will be initialized to false and then only set to true after the test proves the system is sufficiently compliant.

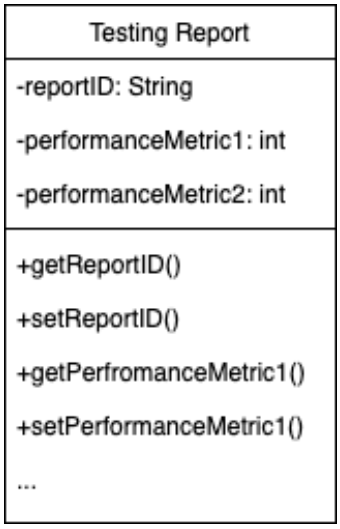
2.8.3 **Methods**

Methods	Descriptions
getReportID()	The getReportID() method can then be seen to return the value of the report ID. As this is a get method, the private reportID value can only be accessed by first going through a public function such as this one.
setReportID()	The setReportID() method can be used to modify the value of the report ID. This method will also provide sufficient validation to ensure that the new value for the ID is compliant with the restrictions set by the data field.
getComplianceStatus()	The getComplianceStatus() function can then be used to return the Boolean compliance value. This function is essential to viewing the

	report so that data privacy best practices can be upheld within the system.
setComplianceStatus()	The setComplainceStatus method can then be used to modify the value stored in the compliance status data field. This field will ensure that a Boolean value is entered and that the value of the field is updated appropriately.
DisplayErrors()	The displayErrors() method can then be used to highlight areas of non-compliance identified by the report. This will print all of the non-compliance errors allowing the user to understand what is needed to achieve compliance.

2.9 Class: Testing Report

2.9.1 Class Description



2.9.2 Attributes

Attributes	Descriptions
reportID	The reportID method can be seen to represent a unique ID value through which the testing report can be uniquely identified. This field only accepts numeric values, and the max character length of these values is nine characters.
performanceMetric1	Several performance metrics will exist within the system testing and to demonstrate this, two performance metrics have been listed here. These will most likely be numeric values and the length of these values will be dependent upon the type of data being collected. These metrics will serve as the basis for the report and allow system testers to understand how a system is performing.

performanceMetric2	Several performance metrics will exist within the system testing and to demonstrate this, two performance metrics have been listed here. These will most likely be numeric values and the length of these values will be dependent upon the type of data being collected. These metrics will serve as the basis for the report and allow system testers to understand how a system is performing.

2.9.3 Methods

Methods	Descriptions
getReportID()	The getReportID() method will be used to return the value of the report ID so that the user can safely view this value. The security principle of abstraction is upheld through this implementation as get functions form an interface to access the private data member values.
setReportID()	The setReportID() can then be seen to allow for the modification of the reportID field. This report ID must uphold the property of uniqueness among the report IDs and as such thorough input validation will be performed within this method.
getPerformanceMetric1()	This represents the get methods instantiated for each performance metric value implemented in this class. The number and value of metrics will be dependent on the specific metrics desired to be measured in the report and as such an abstracted representation of the get function for these variables has been created here.
setPerformanceMetric1()	The set methods in a similar manner then represent the abstract instantiation for all of the performance metrics that will be generated

	by the report. This function will perform strong data validation for all inputs and will serve to update performance metric variable values.

3. User Interface Design

3.1 Use Case(s) Interface

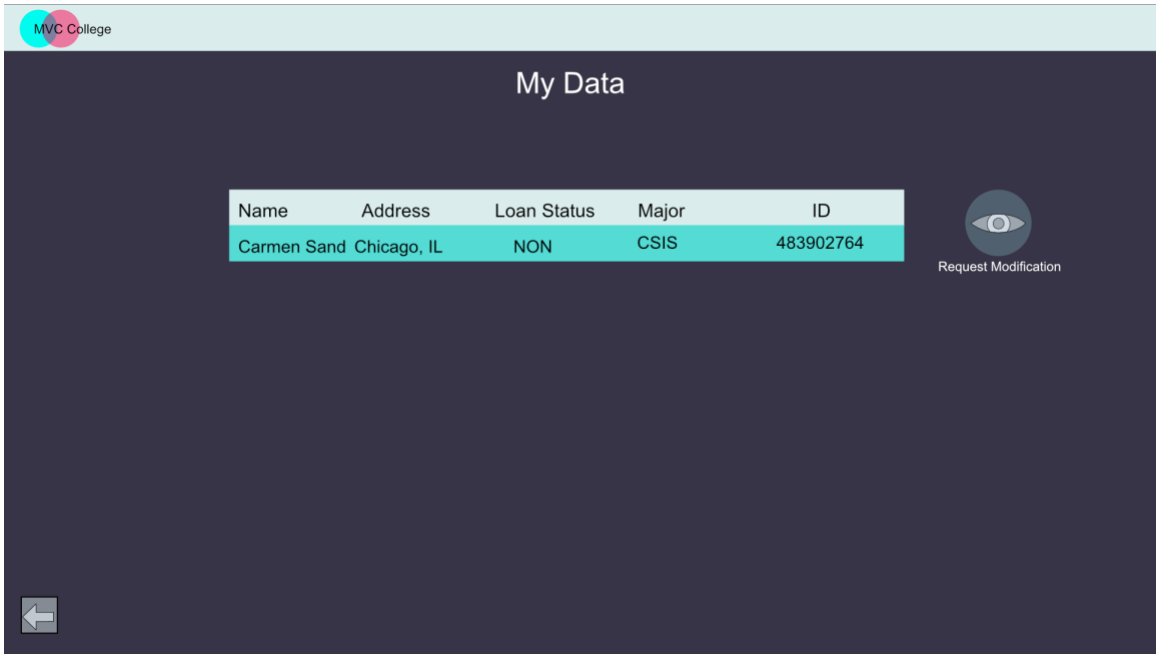
3.1.1 Student

3.1.1.1 First user interface for Use Case 3.1.1

Description:

As the student use case is the simplest in terms of functionality, I have opted to display this interface first and explain its features. When the user enters the system, the only functionality options that they will have include the ability to view their own data, and then request modification of those data fields. The interface below will be displayed after the student logs in (See section 3.1.5 for login interface) and will then display the student's data being held by the system. The student user is then able to select a field they wish to edit and then the "request modification" button to initiate the action that will begin the modification request function. When the user logs in, their user group student will be detected, and as such only the details for that particular student will be displayed maintaining the principle of confidentiality for the data of other students. The use of the login system tied to the user ID also stops the incorrect chowing of data from students with shared names, further promoting data confidentiality within the system.

Image:



Fields Table:

Fields	Descriptions
Selected Field	This first field refers to the user's ability to select one of the given fields on the screen by clicking on the text representative of that field. For example, if the student user wished to modify the details of their address, they would first click on the address ("Chicago, IL" in the example above) before clicking on the request modification button to continue the modification request.
Request Modification	This field is a button that can be selected after the user has selected a field they wish to be modified. After selecting this button, a text box will appear asking the student to enter the new value for this field. Depending on the field selected, this could then prompt the user to key in a value or select the value from a dropdown box of options.

Error Messages Table:

Error Messages	Descriptions
"No modification field selected. Please select a field for modification before selecting the request button."	This error message will appear when the user presses the button to request modification before selecting a field to modify. The message then clearly directs the user to select a field before modification can occur.
"Invalid modification value. Please ensure the data you entered is of the correct type and try again."	This is the generalized message that will appear as the catch-all to explain improper input while maintaining the security principle of not being overly specific with error messages (Merkow & Raghavan, 2012). This error message will then be displayed when a non-validated value is entered into a field that accepts alphanumeric data as the next two error messages catch simpler type errors.
"This field only accepts alphabetic characters. Please try again"	This message will appear when a user attempts to enter non-alphabetic data into a field that only accepts data of that type. For example, if numbers were entered into a name field then this error would be displayed.
"This field only accepts numeric characters. Please try again"	This then can be seen as the inverse of the previous error message as this message will be displayed if the user attempts to enter alphabetic characters into a field that only accepts numeric values. For example, attempting to enter letters into the student ID field would generate this error message.

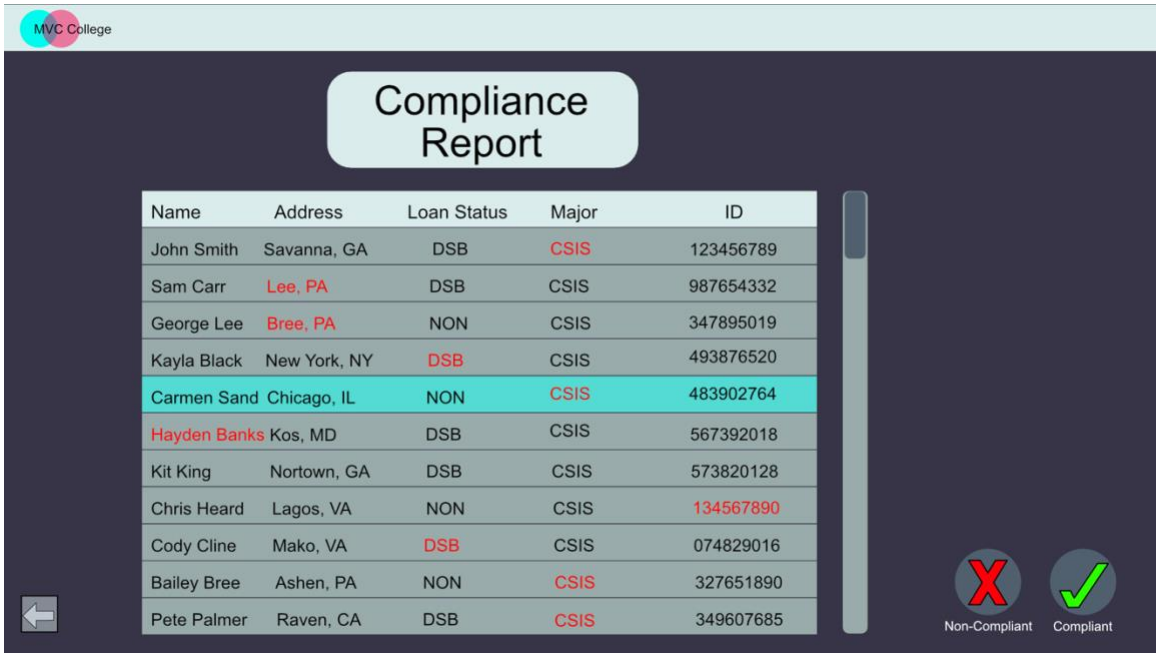
3.1.2 FERPA Compliance Officer

3.1.2.1 First user interface for Use Case 3.1.2

Description:

This use case would then be the interface through which a compliance officer generates compliance reports. An example is generated below where the compliance officer would gain access to the database with the compliance audit component of the software highlighting any areas that may be of concern and require further auditor inspection. The auditor then has the ability to choose the “compliant” or “non-compliant” options to process the audit summary. Upon selecting one of these buttons, the compliance officer will be asked to verify that their choice is correct and submit their findings. The report generated as an artifact of this step can then be used in improving the system to ensure compliance is maintained.

Image:



Fields Table:

Fields	Descriptions
Compliant	This button can be selected if after reviewing the output data the compliance officer is content that the system is compliant with FERPA regulations. Other than the compliant and non-compliant buttons, there are no fields that can be manually entered through the compliance officer's interface. This practice upholds the principle of least privilege as the compliance officer is only given access to the portions of the system he requires.
Non-Compliant	Likewise, the non-compliant button can be selected if compliance has been found to be broken and the compliance officer can then use this report as an artifact to justify their decision.

Error Messages Table:

Error Messages	Descriptions
"Are you sure you want to mark this system as Compliant?"	While not an error message, the commit message will appear to prompt the compliance officer to confirm their choice of compliance results. This is to stop the compliance officer from submitting an incorrect report due to a mis-click or other mistake.

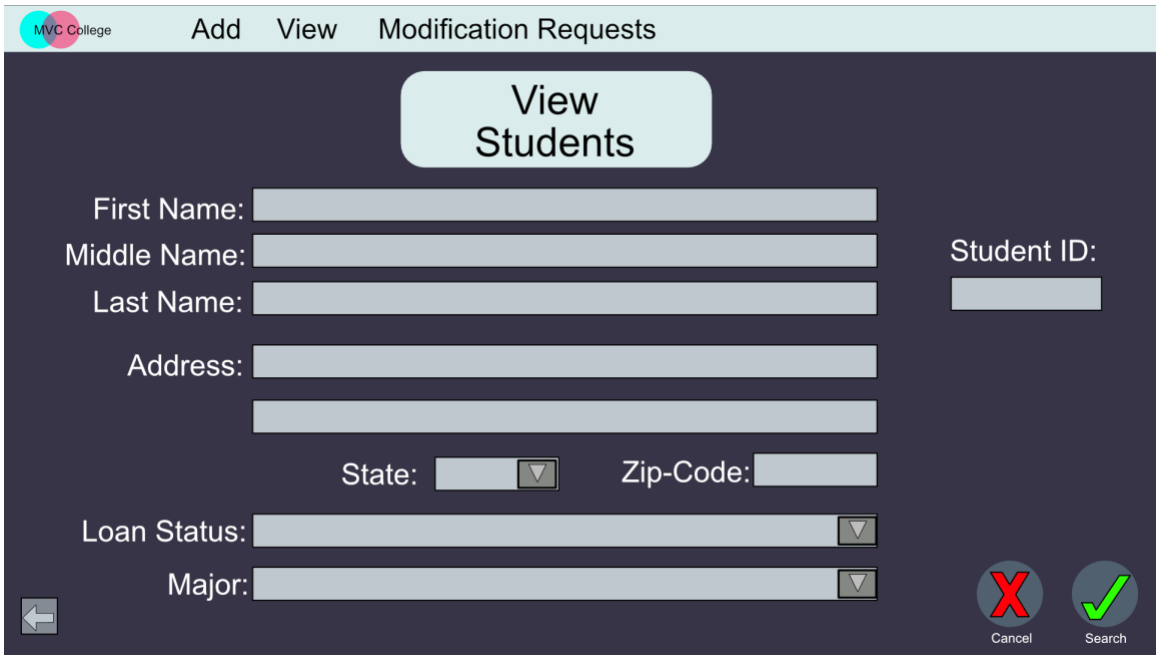
3.1.3 Faculty

3.1.3.1 First user interface for Use Case 3.1.3

Description:

The first aspect of the faculty interface is the section used for searching for students within the database. Any number of the fields presented can be filled in by the faculty member to narrow down their search and display a subsection of students relative to their input. For example, to view all of the students studying computer science, the CSIS option could be chosen from the drop-down major menu before selecting the search confirmation button. Likewise, if a faculty member wished to search for a specific student, they could fill in all of the details or the student's student ID. As can be seen in the interface design, the options also exist in the top menu bar for faculty members to switch between the action menus for different actions faculty members are able to perform such as the addition of new students or the modification request system. Further, the interface for student searches contains various forms of input such as drop-down boxes and keyed-in text. This is to reduce the number of errors generated through mistakes in user input (Hartson & Pyla, 2012) but when incorrect data is entered, error messages will be displayed to assist in correction. If no fields are selected when the search button is pressed, the search results will return all current student entries in the database.

Image:



Fields Table:

Fields	Descriptions
First Name	This field accepts up to 25 alphanumeric characters for a first name search. If this length is exceeded, an error message will be displayed to prompt for proper input. As with all the fields in this form, this field is optional and may be left blank if desired. This field does not accept spaces but does accept a limited number of special characters commonly used in names.
Middle Name	Similar to the first name field, the middle name field accepts alphanumeric characters up to a length of 25 characters. The same

	validation performed on the first name field will be performed on the middle name field through a call to the setMiddleName() function.
Last Name	The last name field will then result in a call to the setLastName() method which will attempt to insert the value, performing input validation in the process. This field must contain alphanumeric characters and be of length less than 25 characters to be accepted as input for this field.
Student ID	The student ID field will accept a numeric string of up to 9 characters in length. The numeric value entered must be within range of the university's distributed ID values and additionally must consist of only numeric characters.
Address	The address field can be up to fifty characters long and can contain alphanumeric characters not labeled as high risk for injection attacks. To further protect from injection attacks, the encoding of special characters can be performed protecting input from executing malicious code on the server (Razumov et al., 2023).
State	The state field contains a dropdown box of all of the two-character state codes. The user is then able to select the state code they desire from this dropdown menu. However, input validation will still be performed to ensure a malicious actor has not passed their own value as input for this field.

Zip Code	The zip code field will then accept numeric values of up to five digits in length as this should be sufficient to accept every postcode variation.
Loan Status	The loan status will present a dropdown menu of the loan status codes from which a user can select the loan status they desire. However, input validation will still be performed to ensure that the malicious injection of values is not accepted.
Major	Likewise, the major field will also accept input from the dropdown menu displaying a pre-selection of major values. No value outside of this list will be accepted as the setMajor() function will perform input validation on the field value.

Error Messages Table:

Error Messages	Descriptions
"The value entered for this field is too long. Please try again."	This error message will be displayed if the data entered into the field exceeds the maximum allowed character length for that field. The field length for entries is limited to mitigate the possible damages of injection attacks (Razumov et al., 2023) and as such should be strongly enforced within the system.
"This value cannot be accepted. Please ensure the data you entered is of the correct type and try again."	This is the generalized message that will appear as the catch-all to explain improper input while maintaining the security principle of not being overly specific with error messages (Merkow & Raghavan, 2012). This error message will then

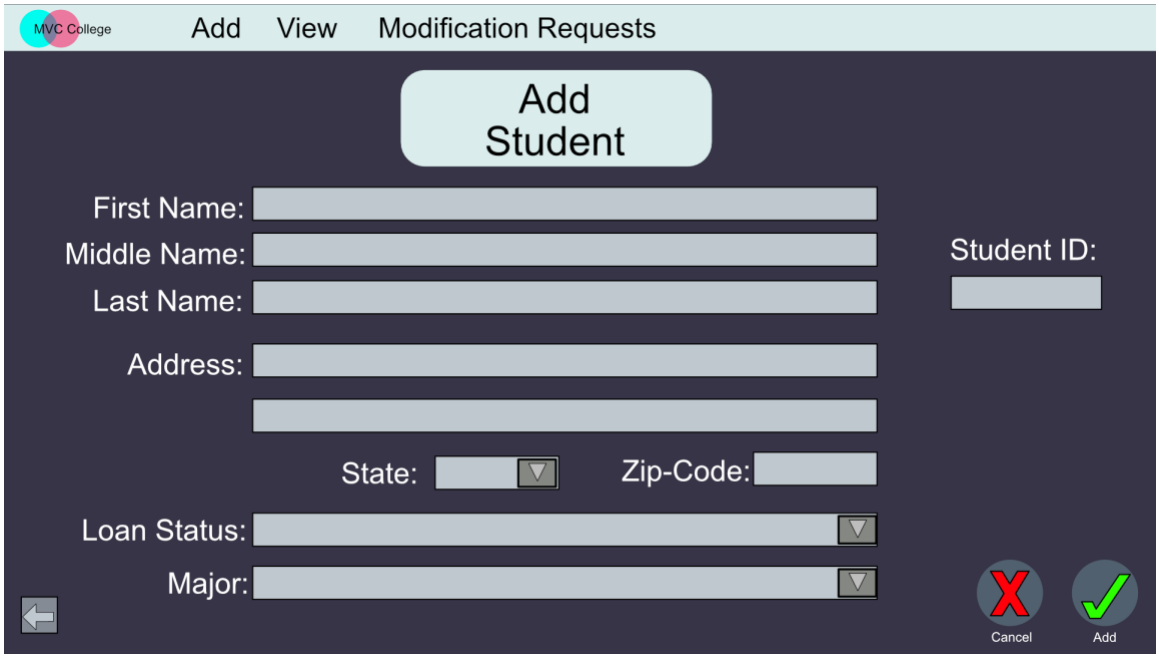
	be displayed when a non-validated value is entered into a field that accepts alphanumeric data as the next two error messages catch simpler type errors.
"This field only accepts alphabetic characters. Please try again"	This message will appear when a user attempts to enter non-alphabetic data into a field that only accepts data of that type. For example, if numbers were entered into a name field then this error would be displayed.
"This field only accepts numeric characters. Please try again"	This then can be seen as the inverse of the previous error message as this message will be displayed if the user attempts to enter alphabetic characters into a field that only accepts numeric values. For example, attempting to enter letters into the student ID field would generate this error message.

3.1.3.2 Second user interface for Use Case 3.1.3

Description:

The add student interface can then be observed as very similar to the interface used in searching for student entries. However, one of the most important distinctions between these two interfaces is that no fields within this interface are allowed to be left empty. This is to ensure complete data is kept within the system and avoids the problems associated with incomplete database entries. Additionally, the commit buttons have different labels and will generate a committal message to ensure that the user does not accidentally commit a value they do not intend to. With this in mind, the interface can then be seen to effectively add student entries to the system database.

Image:



Fields Table:

Fields	Descriptions
First Name	This field accepts up to 25 alphanumeric characters for a first name search. If this length is exceeded, an error message will be displayed to prompt for proper input. As with all the fields in this form, this field is optional and may be left blank if desired. This field does not accept spaces but does accept a limited number of special characters commonly used in names.
Middle Name	Similar to the first name field, the middle name field accepts alphanumeric characters up to a length of 25 characters. The same validation performed on the first name field will be performed on the middle name field through a call to the setMiddleName() function.

Last Name	The last name field will then result in a call to the setLastName() method which will attempt to insert the value, performing input validation in the process. This field must contain alphanumeric characters and be of length less than 25 characters to be accepted as input for this field.
Student ID	The student ID field will accept a numeric string of up to 9 characters in length. The numeric value entered must be within range of the university's distributed ID values and additionally must consist of only numeric characters.
Address	The address field can be up to fifty characters long and can contain alphanumeric characters not labeled as high risk for injection attacks. To further protect from injection attacks, the encoding of special characters can be performed protecting input from executing malicious code on the server (Razumov et al., 2023).
State	The state field contains a dropdown box of all of the two-character state codes. The user is then able to select the state code they desire from this dropdown menu. However, input validation will still be performed to ensure a malicious actor has not passed their own value as input for this field.
Zip Code	The zip code field will then accept numeric values of up to five digits in length as this should be sufficient to accept every postcode variation.

Loan Status	The loan status will present a dropdown menu of the loan status codes from which a user can select the loan status they desire. However, input validation will still be performed to ensure that the malicious injection of values is not accepted.
Major	Likewise, the major field will also accept input from the dropdown menu displaying a pre-selection of major values. No value outside of this list will be accepted as the setMajor() function will perform input validation on the field value.

Error Messages Table:

Error Messages	Descriptions
"Student ID is already taken. Please enter a unique student ID value."	This error message will be displayed when a duplicate user ID is attempted to be added to the system. The system will recognize duplicate IDs and force the faculty member to select an appropriate ID value. This is to ensure data is not mistakenly revealed to a student to whom the data does not belong through shared user IDs.
"Are you sure you wish to commit these values to the database?"	Although not an error message, the committal message will be displayed to ensure that the user means to submit the data submission they are attempting to make. This can stop false entries and increase the quality of database values.
"The value entered for this field is too long. Please try again."	This error message will be displayed if the data entered into the field exceeds the maximum allowed character length for

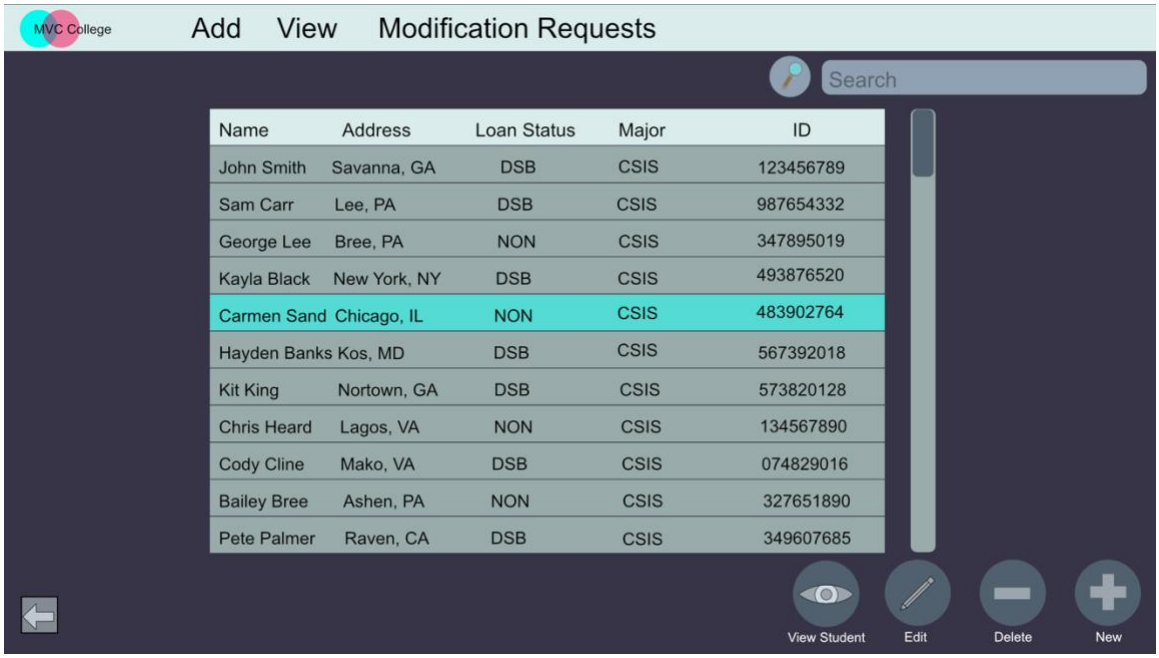
	that field. The field length for entries is limited to mitigate the possible damages of injection attacks (Razumov et al., 2023) and as such should be strongly enforced within the system.
"This value cannot be accepted. Please ensure the data you entered is of the correct type and try again."	This is the generalized message that will appear as the catch-all to explain improper input while maintaining the security principle of not being overly specific with error messages (Merkow & Raghavan, 2012). This error message will then be displayed when a non-validated value is entered into a field that accepts alphanumeric data as the next two error messages catch simpler type errors.
"This field only accepts alphabetic characters. Please try again"	This message will appear when a user attempts to enter non-alphabetic data into a field that only accepts data of that type. For example, if numbers were entered into a name field then this error would be displayed.
"This field only accepts numeric characters. Please try again"	This then can be seen as the inverse of the previous error message as this message will be displayed if the user attempts to enter alphabetic characters into a field that only accepts numeric values. For example, attempting to enter letters into the student ID field would generate this error message.

3.1.3.3 Third user interface for Use Case 3.1.3

Description:

The next interface to be viewed can then be seen as the student view interface. This page will appear upon a student search being performed and will display a list of students relevant to the search criteria. In the example below, a search has been performed for all students in the computer science program. The user is then able to scroll through this list and select students for modification or deletion from the database. Additionally, faculty members maintain the ability to further view records for that student or to add new students to the database which will launch the student addition interface page. A search bar also exists in the top right corner to further refine the search by searching for key values. This interface will have minimal input fields and therefore error messages but is an important artifact in the system's use so has been included as an interface image below.

Image:



Fields Table:

Fields	Descriptions
View Student	This field refers to the view student button which will expand a student entry for a more detailed view of student statistics. This could include student tuition rates, academic standings, extracurricular activities, etc., and could form a valuable view for faculty to access.
Edit	The edit function will then launch the modification page and previously identified and allow faculty to modify the entries for student fields.
Delete	The delete option can be used to remove a student entry from the database. This can be accomplished by first selecting the subset of students desired to be deleted and then selecting the delete button at

	the bottom of the screen. This action will need to be confirmed through the pop-up message before the action is fulfilled.
New	The new button will then launch the web page used for adding new students to the database previously outlined above.

Error Messages Table:

Error Messages	Descriptions
"No students match these input values. Please perform another search with different values."	This error message will be displayed when the values used for searching for students present no relevant database entries. This can result from either the initial search or the use of the search bar and will then return the user to the main search interface page to attempt another search.
"Are you sure you wish to delete this student record?"	This committal message will appear to ensure that a faculty member intends to delete the student record(s) from the database. This is an essential aspect of the system as the accidental deletion of student records could have serious implications on a student's academic experience.
"No students selected for deletion. Please select students and try again."	This error message will appear when the user attempts to delete student entries with no student values being selected for deletion.

3.1.3.4 Fourth user interface for Use Case 3.1.3

Description:

The final faculty interface page can then be seen as the page used in verifying student modification requests. This page appears very similar in design to the student search page, but the entries on this page consist of all students who have submitted a data change request. The data fields wishing to be modified are colored in red so that they are easier to identify and as such, a faculty member will be easily able to identify which information they must validate. Upon validation, the faculty member is then able to use the accept or reject buttons in the bottom right corner to process the requested change. Processing the change will then generate a message on the student's application page to inform them of the acceptance or denial of their request.

Image:

Fields Table:

Fields	Descriptions
Accept	This button can be used to process the student's data request change as submitted. This must first be confirmed, but upon confirmation will process the changes on the database system.
Reject	The reject button will then also need to be committed but will reject the changes and return back to the user with the rejection message. The rejection message will additionally allow the faculty to input comments as to why the request has been rejected, allowing the student to make modifications and resubmit their request.

Error Messages Table:

Error Messages	Descriptions
"Are you sure you wish to accept this change? This action will be final."	This message will be displayed in order for the faculty member to be able to confirm the choice they have made and reduce the number of risks made during this step of processing.

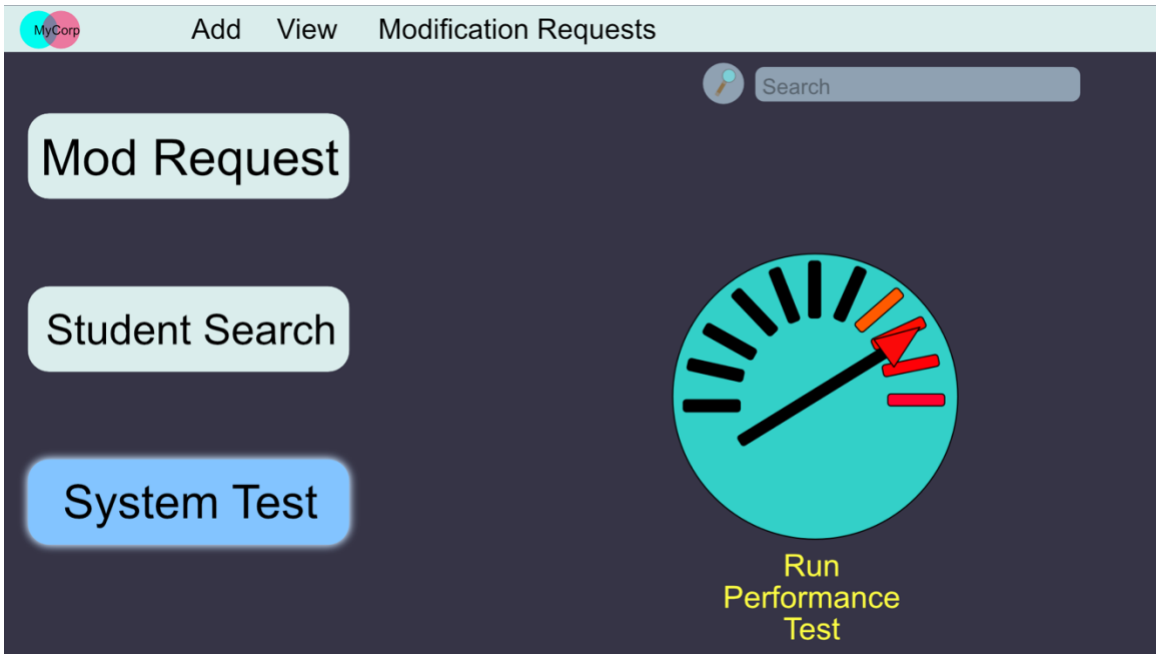
3.1.4 Registrar

3.1.4.1 First user interface for Use Case 3.1.4

Description:

The registrar use case can then be seen as an extension of the faculty use case as the registrar class extends the faculty class. However, an additional interface element can be seen on the left side of the screen further identifying the main subsystems that registrar faculty are able to interact with. Additionally, the system test function can be seen which is an exclusive system for registrars to interact with. This page features a button to run system tests and generate a system test report. This report can then be viewed to gain insights into system performance and track performance metrics throughout the system. Registrars are the users with the greatest level of system privileges and as such have been given the most involved user interface to perform those actions.

Image:



Fields Table:

Fields	Descriptions
Run Performance Test	This field will run the performance test to generate a report of system performance metrics. These metrics can then be observed by the registrar staff to glean information about the system's performance and identify areas for potential improvement.

Error Messages Table:

Error Messages	Descriptions
"System test failed. Please try again in a moment."	This message will be displayed if the system test fails, and the search is unable to be performed. The user will then be returned to the test launch page where they can reinitiate the test or select another system function to interact with.

3.1.5 General Login System

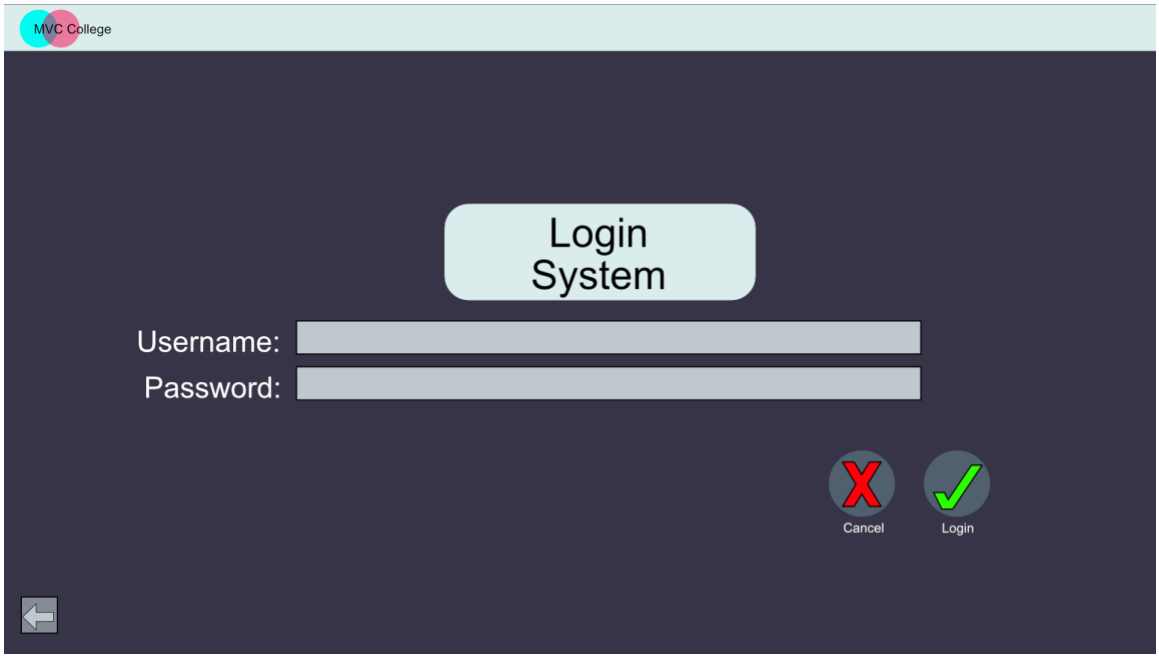
3.1.5.1 First user interface for Use Case 3.1.5

Description:

Upon first entering the system, all users will be presented with the login page to verify user credentials. This page can be used to verify users but will then further attach the associated privileges for the user group that the user is a part of. This means that students will be given student privileges and then sent to the student interface and so on with the remaining user group types. Login systems are an essential aspect of access control, and as such the login system will be strongly enforced without the possibility of navigating around this system. All authentication credentials will then be hashed and the

hashed values sent to the database for comparison with the stored hashed values (Basta, 2018). This can further ensure the security of the login system and allow for a zero-knowledge-proof approach to user authentication. The user login system presented here requires the user to enter their unique username in addition to a password of their choice. Through this, users can be effectively authenticated within the system. Strong password practices will further be practiced within the system to increase system security and ensure user data is kept safe from harm.

Image:



Fields Table:

Fields	Descriptions
Username	The username field represents a field where users can enter their unique username for system authentication. This field must be unique and can consist of alphanumeric characters. This field has a minimum length of 6 characters as well as a maximum length of 25. Enforcing

	this can help ensure that user credentials cannot be guessed, improving system security.
Password	The password field similarly contains alphanumeric characters and must be of a length between 6 and 25 characters. Additional features of the password field are that the password must contain at least one number and that the password cannot consist of a single English dictionary word with a number appended to the end. Enforcing these rules will make passwords more difficult to guess and the system therefore more secure.

Error Messages Table:

Error Messages	Descriptions
"The details you entered are not correct. Please try again"	This error message will be displayed when either the username or password is incorrect. It is worth noting that this message is intentionally vague to decrease the ability of a malicious actor to guess part of a user's login credentials.

Appendix – Updated Requirements Traceability Matrix

*<**Update** the matrix from the SRS version and include the new column that maps the requirements to the SDS sections.*

If any requirements are **not part of the design**, describe why not. >

Priority	Requirement # by Category	Description	SRS Section	SDS Section
	1	**Logging into the system**	3.3	3.1.5
1	1.1	**Validation of user Passwords**	3.3	3.1.5
1	1.1.1	**Minimum Password Requirements Met**	3.3	3.1.5
1	1.1.2	**Password Hashed and Hash Value Stored for Authentication**	3.3	3.1.5
1	1.1.3	**Passwords Must Regularly Be Updated**	3.3	3.1.5

2	1.2	**User Sessions will Timeout After a Period of Inactivity**	3.3	2.6
2	1.3	**Only Connections from Within the U.S. Should Be Accepted**	3.3	2.1
	2	**System Performance**	3.1	3.1.4
2	2.1	**Efficient Querying of Database**	3.1	1.1
1	2.2	**Sufficient Resources for College User Base**	3.1	1.1
2	2.3	**Sufficient Data Throughput for Connections**	3.1	1.1
1	2.3.1	**TCP Implemented for User Connections**	3.1	1.1
		Critical Sections of Data Modification Protected		

1	2.4		3.1	Not directly addressed in this report as the database's inner workings were not examined here.
	3	**Data Protection**	3.3	3.1
1	3.1	**Encryption of Data at Rest**	3.3	3.1
1	3.2	**Encryption of Data in Transit**	3.3	1.1
1	3.3	**Principle of Least Privilege Applied Throughout Design**	3.3	3.1.2
1	3.4	**Audit Log Maintained of all Data Transformations**	3.3	1.1
	4	**Data Quality**	4	3.1

1	4.1	**All Mandatory Fields Contain a Value**	3.2	3.1
1	4.2	**Every Student Has a Data Entry**	4	3.1
1	4.3	**Data Validation is Performed on All Fields**	4	2
1	4.4	**User Records Deleted After Allotted Time**	4	3.1
	5	**Legal Compliance**	4	3.1.2
1	5.1	**FERPA Regulation Compliance Throughout Design**	4	3.1.2
1	5.2	**All U.S. PII Best Practices and Legislation Must Be Followed**	4	3.1.2
	6	**Scalability**	4	Not directly addressed in this report, but choosing a good design in itself increases the

				ability for the system to be scaled down the line.
3	6.1	**System Must Be Easily Scalable to Accommodate Student Growth**	4	1.1
2	6.2	**The system Must Be of Adequate Size to Hold All Student Records Plus Growth Buffer**	4	2.7
3	6.3	**System Brought Up to Compliance for International Use**	4	3.1.2
	7	**System Protection**	3.2	3.1
1	7.1	**Students Must Be Able to Request Corrections to Their Data**	3.2	3.1.1
1	7.1.1	**Only the Data Owner Can Request Data Changes**		

			3.2	3.1.1
1	7.2	**Acceptable Use Policy Implemented**	3.2	Not directly addressed in this report, but security policies and best practices are highlighted throughout this design document that lends to the basis of an acceptable use policy.
1	7.3	**DOS Protection**	3.2	1.1
1	7.3.1	**TCP Connections Will Time Out After Inactivity**	3.2	1.1
1	7.4	**User Action Verification**	3.2	3.1.5

References

Basta, A. (2018). *Oriyano, cryptography: Infosec pro guide.* McGraw-Hill Education. https://bookshelf.vitalsource.com/reader/books/9781307297003/pageid/14

Hartson, H. R., & Pyla, P. S. (2012). *The UX book: Process and guidelines for ensuring a quality user experience*. Morgan Kaufmann. https://doi.org/10.1016/C2010-0-66326-7

Merkow, M. S., & Raghavan, L. (2012). *Secure and resilient software: Requirements, test cases, and testing methods (1st ed.)*. CRC Press. https://doi.org/10.1201/b11317

Microsoft. (2021). Create a UML deployment diagram. Microsoft Support. https://support.microsoft.com/en-au/office/create-a-uml-deployment-diagram-ef282f3e-49a5-48f5-a6ae-69a6982a4543

Razumov, P., Cherckesova, L., Revyakina, E., Morozov, S., Medvedev, D., & Lobodenko, A. (2023). Ensuring the security of web applications operating on the basis of the SSL/TLS protocol. *E3S Web of Conferences, 402*, 3028. https://doi.org/10.1051/e3sconf/202340203028

Sobhy, D., Bahsoon, R., Minku, L., & Kazman, R. (2021). Evaluation of software architectures under uncertainty: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, *30*(4), 1-50. https://doi.org/10.1145/3464305

Tsui, F, Karam, O., Bernal, B. (2016). *Essentials of Software Engineering* (4th ed.). Jones & Bartlett Learning. https://libertyonline.vitalsource.com/books/9781284129755