# OVERVIEW

This program will implement a credentialing system (a subsystem of the scheduling system) for Davenport University. The database will store the requirements for a course regarding instructor credentials, room equipment, room software, location requirements, and other relevant constraints that pertain to the scheduling of a course.

Due to the time and resource constraints that were placed on the project and the team, the program will not be finalized. However, some features will be functional and the technical design will allow for future teams to continue the project. The MVC application will provide capabilities such as: creating, updating and searching for certain elements (software, equipment, room, courses etc.).

# SCOPE

The scope of the project is to design and implement the aforementioned credentialing system to reflect the approved use cases. The purpose of this project is to create a more efficient way of tracking instructor credentials and determining which instructors are qualified to teach specific courses. The primary stakeholder who would be benefiting from the results of this project would be the scheduling department. The credentialing system will handle credentialing for all departments, except for Occupational Therapy, Health, and Online classes. The project has various constraints and risks that will be mentioned throughout this document.

# RISKS

## Critical

1. Estimation and scheduling - The team is unfamiliar with the APIs and frameworks. This lack of experience has the potential to influence the schedule. To mitigate this risk, the team conducted research using lynda.com and Microsoft channel 9 into JQuery, Bootstrap, and MVC. Although the team has conducted thorough research on the required topics, the team is far from being proficient with the tools. This will cause the project to be delayed. This risk cannot be mitigated; however, it can be minimized. The current team will create thorough documentation to ensure a smooth transition for the next team.

2. Breakdown of specifications - Requirements can end up conflicting with one another. This risk should be avoided by having the whole team at stakeholder meetings, so that every member can clearly understand his/her role and how to implement certain requirements.

3. Compromising on Designs - Designs are often rushed to get to the actual coding phase of the project. The risk will be avoided by conducting the research to understand requirements by having the team create a thorough design plan before implementation.

4. Productivity Issues - Developers tend to underestimate longer projects early on. This should be mitigated by the teams experience with larger projects (previous internship experiences).

### Non Critical

1. Sudden Growth in Requirements - Without a thorough understanding of the requirements, issues can arise as the deadline approaches. This risk will be mitigated by interviewing stakeholders and performing a SWOT analysis to gather requirements.

2. Gold Plating - The addition of unnecessary features to show off will be avoided by adhering to the triple constraints, most notably time. The project is already going to fill the semester without adding anything extra.

3. Missing the "big picture" - This risk can cause confusion when it comes to establishing requirements and designing the system. In order to mitigate this risk, the team will communicate openly and efficiently with each other and get their teammates on the same page.

4. Developing Cross Platform - The team had to troubleshoot some issues connecting to the SQL database in a Linux environment. This was mitigated by dropping the Linux environment and only working on the Windows machines on campus.

5. Naming Conventions - Naming conventions are an important part of the design of both the database and the overall system. Establishing naming conventions early is essential to avoid having to go back and rework existing tables, stored procedures, and other data in the system. Risk has been mitigated by creating a document with naming convention standards.

6. Issues updating and retrieving information to and from GitLab - The team has had issues with pulling, cloning, and pushing the code from GitLab from home computers. Using powershell and communication with other team members mitigate this risk. Using the computers at the Davenport Lettinga campus is usually the solution, but during the holiday weekend, the computer labs are closed.

## DATABASE

### Naming Conventions

The database will utilize the following naming conventions:

1. Database Prefix - the database prefix will be ACB350
2. Connector - the underscore _ will be used as the connector
3. Prefix - the prefix will be utilized for tables and stored procedures
   - Table: TBL
   - Stored Procedure: SPR
4. Action - describes the action that is performed (used for stored procedures only)
   - Insert
   - Delete - A column named *TableName*Deleted of datatype bit will be added to every table. The default will need to be set to 0 for active. If a record gets deleted, the value will be changed to a 1 for inactive.
   - Update
   - Inquire
5. Object - describes the object that is inquired or modified (tables and stored procedures)
   - Room
   - Campus
   - Instructor
   - Etc.

   Examples
6. Table
   - ACB350_TBL_Campus
   - ACB350_TBL_Credentials
7. Stored procedure
   - ACB350_SPR_Insert_Campus
   - ACB350_SPR_Update_Instructor
8. Add an extra attribute to differentiate between stored procedures
   - ACB350_SPR_Inquire_Software_Room
   - ACB350_SPR_Inquire_Software_Name

### Accessing the Database

The database can be accessed two ways.

1. From Davenport University
   - Open Visual Studio – "Data Connections" – "Add Connection"
   - Under "Server Name", type
   - Under "Select or enter a database name", select
   - Click "OK"
2. From a remote location
   - Windows Environment - Use a windows script.
   - Linux Environment - Issues have been encountered when using a Linux machine; therefore, it is possible but not recommended.

# COMPONENTS

## ERD

The revised ERD can be found at the end of this document.

Important ERD definitions & notes:

1. Definitions
   - Credentials
     (a) Credentials can be degrees, certifications, and other relevant attachments (see below).
     (b) An instructor will need various credentials to teach specific courses. For the scope of this project, the credentials will be identified by codes (TBD).
     (c) A relationship between Courses and Credentials will be identified in order to fulfill queries.
     (d) Credential expiration dates will be handled by allowing NULL data. Queries will be refined to handle NULL types.
     (e) CredentialTypeID, CredentialLevelID, and CredentialName in Credentials all define the credential - all three together need to have a UNIQUE constraint
     (f) CredentialLevelCode in CredentialLevel would be defined as: Masters, PhD, Bachelors, Certificate, etc.
   - Attachments
     (a) Attachments can be resumes, CVs, certifications, degrees, and other relevant documents.

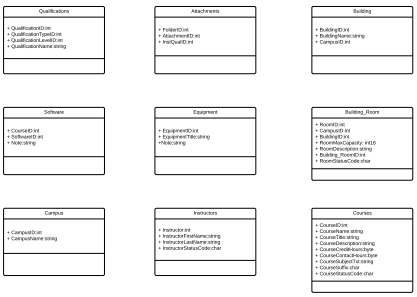
(b) An instructor can have various attachments. A limit will have to be set on the number of attachments.

(c) The attachments will be stored on the file system rather than in the database.

(d) An Attachment Folder can hold many attachments, but an attachment can only be in one folder. The folder holds the path on the disk to the file and the attachment holds the name of the file. Putting those together would get the full path on the disk.

- Software
  (a) Software includes any program or tool needed in a classroom.
  (b) The regular computer lab requirements include Adobe, MS Office, Minitab, etc.
  (c) Specialty labs include software such as Visual Studio, Construct 2, Game Maker Studio, Unity, Android Studio, MATLAB, etc.
- Equipment
  (a) Equipment includes any hardware or other items needed in a classroom.
  (b) The classroom requirements include computer labs, specialty labs, and other equipment.
  (c) Specialty labs include CISCO/NETW, CSCI, Biometrics, Physics, Health, Chem, etc.
- Building_Room
  (a) The building room will define the max capacity of a room as well as the room description.
  (b) The RoomID will reflect the room number (101, 236, 320); however, the Building_RoomID will be a UNIQUE identifier to locate a specific room, in a specific building, on a specific campus.
- Courses
  (a) CourseName in Courses must have a UNIQUE constraint.
  (b) The Course table will hold information about a course such as contact hours, credit hours, title, description, suffix, and subject.

### Class UML




The main tables that will be implemented as classes are Software, Equipment, Building_Room, Building, Instructor, Attachments, Campus, and Courses. A more in-depth UML diagram is attached at the end of this document.

## MVC

### Naming Conventions

The program will utilize the following naming conventions:

1. Models - the models will take the name of the object. Example: Campus, Building, Equipment, etc. The exceptions are Building_Room - this class is simply called Room - and ModelInquire - this class is implementing error handling for inquire queries.

2. Views - each view (index, create, edit, etc.) will be classified by the respective model folder. Example: the view for the Campus model will be under the folder Campus.

3. Controller - the controller will take the name of the object and add the word controller at the end (example: SoftwareController, CampusController, etc.).

4. Data Access - each model will have a data access layer class titled name of the object and DB at the end of it (example: CampusDB, BuildingDB, etc.).

### GitLab Updates

To access the GitLab Repository, go to Login with Davenport credentials. The following updates have been pushed to GitLab:

1. Update 12/10: This commit has some updates to CourseDB for search parameters. Stored procedure was added to work with the search.

2. Update 12/09: Made slight changes to the Layout for Room, Building, and Campus. Room Controller was found to have issues when creating a new room. The bugs were fixed and it works properly now.

3. Update 12/08: Error handling has been implemented for Course class. Search and inquire for Course have updates

4. Update 12/07: Campus, Building, Room, Equipment, and Software data access layers now return a Model Inquire model of specific type, along with a status code and a status description. The status description is displayed by the shared view "SharedError" when an error occurs (e.g. looking for details for a nonexistent record will display " Error, no records found ").TODO: fix layout and minor details (labels need to be changed in views).

5. Update 12/06: Error handling was implemented for Building, Campus, and Room. Shared Error was created - it allows all controllers to use this view to display error messages.

6. Update 12/05: Updates to Layout and color scheme were made.

7. Update 12/04: Changes from yesterday were changed again. There was some issue pushing out the changes. Header was changed to red - DU's color.

8. Update 12/03: Some error handling was implemented for Campus and Building. Some bugs were fixed after implementing the error handling. TODO: Implement more error handling.

9. Update 11/29: Course Create is functioning properly now. No other major changes.

10. Update 11/28: Start of the MVC for Courses has been created. Inquire all is the only functioning View.

11. Update 11/27: The Room controller has been modified and a Search view has been implemented. The Search view allows for a user to search for a room that contains a specific software and/or equipment based on a specific campus. TODO: Need to handle errors. Certain issues do arise and they will need further attention; however, the program does not have any bugs or physical issues.

12. Update 11/19: Room Controller - "Create" data aceess layer/ model/ view was created. You can now successfully create a new room by choosing the building (and campus) the room is in and by choosing which software and equipment the room currently offers.When the stored procedure is run, the program also creates entries in the bridge tables for Building_Room_Software and Building_Room_Equipment. TODO: Need

to implement the "Update" functionality for Room, modify stored procedures (to handle exceptions) and to globally handle errors in the MVC app.

13. Update 11/18: Edit views were created for the Building and Campus models. The Index and Details views were created for the Room model. TODO: create a builder for the Room in order to make the Create and Edit views for the Room model.

14. Update 11/13: Controllers, models, and data access were created for Campus, Building, Software, and Equipment. Add, list, and detail views were created for the aforementioned items.

# PROCESSING LOGIC

The process requires that the database is operational and that it is populated with testing data.

## Data Access Layer

1. Campus

   - Inquire Campus By ID
     (a) Create SQL connection
     (b) Create SQL command (ACB350_SPR_Inquire_Campus_BY_ID)
         – Set the @ID parameter
     (c) Execute command
     (d) If there is a record to read, then create a new building object
     (e) Get the return value (constant)
     (f) Set the status code to equal the return value (constant)
     (g) Return a model inquire of type Campus
   - Inquire All Campuses
     (a) Create SQL connection
     (b) Create SQL command (ACB350_SPR_Inquire_All_Campuses)
     (c) Execute command
     (d) While there are records to read, then add them to a list of Buildings
     (e) Get the return value (constant)
     (f) Set the status code to equal the return value (constant)
     (g) Return a model inquire list of type Campus
   - Inquire Campus By Name
     (a) Create SQL connection
     (b) Create SQL command (ACB350_SPR_Inquire_Campus_BY_Name)

        - Set the @Name parameter
    (c) Execute command
    (d) If there is a record to read, then create a new Campus object
    (e) Get the return value (constant)
    (f) Set the status code to equal the return value (constant)
    (g) Return a model inquire of type Campus
- Create Campus
    (a) Create SQL connection
    (b) Create SQL command (ACB350_SPR_Insert_Campus)
        - Send the @Name parameter
    (c) Execute the non query command
    (d) Get the return value (constant)
    (e) Return the Campus object
- Edit Campus
    (a) Create SQL connection
    (b) Create SQL command (ACB350_SPR_Update_Campus)
        - Send the @Name parameter
        - Send the @ID parameter
    (c) Execute the non query command
    (d) Get the return value (constant)
    (e) Return the Campus object

2. Building

- Inquire Building By ID
    (a) Create SQL connection
    (b) Create SQL command (ACB350_SPR_Inquire_Building_BY_ID)
        - Set the @ID parameter
    (c) Execute command
    (d) If there is a record to read, then create a new building object
    (e) Get the return value (constant)
    (f) Set the status code to equal the return value (constant)
    (g) Return a model inquire of type Building
- Inquire All Buildings
    (a) Create SQL connection
    (b) Create SQL command (ACB350_SPR_Inquire_All_Buildings)
    (c) Execute command
    (d) While there are records to read, then add them to a list of Buildings
    (e) Get the return value (constant)

(f) Set the status code to equal the return value (constant)
(g) Return a model inquire list of type Building

- Inquire Building By Name
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Building_BY_Name)
    – Set the @Name parameter
  (c) Execute command
  (d) If there is a record to read, then create a new building object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire of type Building
- Create Building
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Insert_Building)
    – Send the @Name parameter
  (c) Execute the non query command
  (d) Get the return value (constant)
  (e) Return the Building object
- Edit Building
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Update_Building)
    – Send the @Name parameter
    – Send the @ID parameter
  (c) Execute the non query command
  (d) Get the return value (constant)
  (e) Return the Building object

3. Room

- Inquire Room By ID
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Room_BY_ID)
    – Set the @ID parameter
  (c) Execute command
  (d) If there is a record to read, then create a new room object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire of type Room
- Inquire All Rooms
  (a) Create SQL connection

(b) Create SQL command (ACB350_SPR_Inquire_All_Rooms)
(c) Execute command
(d) While there are records to read, then add them to a list of Rooms
(e) Get the return value (constant)
(f) Set the status code to equal the return value (constant)
(g) Return a model inquire list of type Room

- Inquire Building By Software, Equipment and Building
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Software_AND_Equipment)
    – Set the @Equipment parameter
    – Set the @Software parameter
    – Set the @CampusID parameter
  (c) Execute command
  (d) If there is a record to read, then create a new building object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire of type Room
- Create Room
  **Insert Room**
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Insert_Room)
    – Send the @RoomID parameter
    – Send the @CampusID parameter
    – Send the @BuildingID parameter
    – Send the @RoomMaxCapacity parameter
    – Send the @RoomDescription parameter
  (c) Open connection
  (d) Execute the non query command
  (e) Get the return value (constant)
  (f) Close the connection

  **Insert Software**
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Insert_Room_Software)
    – Send the @RoomID parameter
    – Send the @SoftwareID parameter
  (c) Open connection
  (d) Execute the non query command
  (e) Get the return value (constant)
  (f) Close the connection

**Insert Equipment**

(a) Create SQL connection
(b) Create SQL command (ACB350_SPR_Insert_Room_Equipment)
  – Send the @RoomID parameter
  – Send the @EquipmentID parameter
(c) Open connection
(d) Execute the non query command
(e) Get the return value (constant)
(f) Close the connection

- Edit Room
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Update_Room)
    – Send the @RoomID parameter
    – Send the @CampusID parameter
    – Send the @BuildingID parameter
    – Send the @RoomMaxCapacity parameter
    – Send the @Description parameter
    – Send the @ID parameter
  (c) Execute the non query command
  (d) Get the return value (constant)

4. Software

- Inquire Software By Name
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Software_BY_Name)
    – Set the @Name parameter
  (c) Execute command
  (d) If there is a record to read, then create a new software object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire of type Software
- Inquire Software By ID
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Software_BY_ID)
    – Set the @ID parameter
  (c) Execute command
  (d) If there is a record to read, then create a new software object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)

(g) Return a model inquire of type Software

- Inquire All Software
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_All_Software)
  (c) Execute command
  (d) While there are records to read, then add them to a list of Software
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire list of type Software
- Inquire Software By Room
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Software_BY_Room)
    – Set the @ID parameter
  (c) Execute command
  (d) If there is a record to read, then create a new software object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire of type Software
- Create Software
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Insert_Software)
    – Send the @SoftwareTitle parameter
  (c) Execute the non query command
  (d) Get the return value (constant)
  (e) Return the Software object
- Edit Software
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Update_Software)
    – Send the @SoftwareTitle parameter
    – Send the @ID parameter
  (c) Execute the non query command
  (d) Get the return value (constant)
  (e) Return the Software object

5. Equipment

- Inquire Equipment By Name
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Equipment_BY_Name)

    - Set the @Name parameter
  (c) Execute command
  (d) If there is a record to read, then create a new Equipment object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire of type Equipment
- Inquire Equipment By ID
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Equipment_BY_ID)
    - Set the @ID parameter
  (c) Execute command
  (d) If there is a record to read, then create a new Equipment object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire of type Equipment
- Inquire All Equipment
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_All_Equipment)
  (c) Execute command
  (d) While there are records to read, then add them to a list of Equipment
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire list of type Equipment
- Inquire Equipment By Room
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Inquire_Equipment_BY_Room)
    - Set the @ID parameter
  (c) Execute command
  (d) If there is a record to read, then create a new Equipment object
  (e) Get the return value (constant)
  (f) Set the status code to equal the return value (constant)
  (g) Return a model inquire of type Equipment
- Create Equipment
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Insert_Equipment)
    - Send the @EquipmentTitle parameter
  (c) Execute the non query command
  (d) Get the return value (constant)

(e) Return the Equipment object

- Edit Equipment
  (a) Create SQL connection
  (b) Create SQL command (ACB350_SPR_Update_Equipment)
    – Send the @EquipmentTitle parameter
    – Send the @ID parameter
  (c) Execute the non query command
  (d) Get the return value (constant)
  (e) Return the Equipment object

## Controllers

1. Campus
   - Index
     (a) Create a model inquire list of type campus and inquire all campuses
     (b) If the status code is greater than or equal to 0, return the list to the view
     (c) Otherwise, redirect to Error page (display the status description)
   - Details (given a specific ID)
     (a) Create a model inquire list of type campus and inquire the campus based on the given id
     (b) If the status code is equal to 1, return the object to the view
     (c) Else, redirect to Error page (display the status description)
   - Create
     (a) Create a new campus object
     (b) Add the new campus object name to the collection of campuses
     (c) Run the Create method (stored procedure)
     (d) Redirect to the Index
     (e) If exception is caught, redirect to error page and display message
   - Edit
     (a) Create a new campus object
     (b) Add the new campus object name to the collection of campuses and set the campus id
     (c) Run the Edit method (stored procedure)
     (d) Redirect to the Index
     (e) If exception is caught, redirect to error page and display message
2. Building
   - Index

(a) Create a model inquire list of type building and inquire all buildings
(b) If the status code is greater than or equal to 0, return the list to the view
(c) Otherwise, redirect to Error page (display the status description)

- Details (given a specific ID)
  (a) Create a model inquire list of type building and inquire the building based on the given id
  (b) If the status code is equal to 1, return the object to the view
  (c) Else, redirect to Error page (display the status description)
- Create
  (a) Get the campus name from the collection of campuses
  (b) Create a new campus model and inquire based on the campus name
  (c) If the status code is not equal to 1, redirect to error page and display message
  (d) Add the new building object name to the collection of buildings
  (e) Add the new campus model
  (f) Run the Create method (stored procedure)
  (g) Redirect to the Index
  (h) If exception is caught, redirect to error page and display message
- Edit
  (a) Get the campus name from the collection of campuses
  (b) Get the building name from the collection of buildings
  (c) Create a new campus model and inquire based on the campus name
  (d) If the status code is not equal to 1, redirect to error page and display message
  (e) Add the new building object name to the collection of buildings
  (f) Add the new campus model
  (g) Set the building ID
  (h) Run the Edit method (stored procedure)
  (i) Redirect to the Index
  (j) If exception is caught, redirect to error page and display message

3. Room

- Index
  (a) Create a model inquire list of type room and inquire all rooms
  (b) If the status code is greater than or equal to 0, return the list to the view

(c) Otherwise, redirect to Error page (display the status description)

- Details (given a specific ID)
  (a) Create a model inquire list of type room and inquire the room based on the given id
  (b) If the status code is equal to 1
    – Inquire the room's equipment
    – Inquire the room's software
    – Return the object to the view
  (c) Else, redirect to Error page (display the status description)
- Create
  (a) Get the building from the collection of buildings
  (b) The building name and campus name are in the same field in the drop down list, therefore a split is required on the delimiter (-)
  (c) Get the software from the collection of software
  (d) Get the equipment from the collection of equipment
  (e) Create a list of equipment
  (f) For each equipment in the room record, inquire the equipment
  (g) Create a list of software
  (h) For each software in the room record, inquire the software
  (i) Create a new room and set the room ID, description, max capacity, and inquire the building
  (j) Run the Create method (stored procedure)
  (k) Redirect to the Index
  (l) If exception is caught, redirect to error page and display message
- Edit
  (a) Create a new room model and inquire based on the room ID
  (b) If the status code is not equal to 1
    – Set the room max capacity
    – Set the room number
    – set the room description
    – Run the Edit method (stored procedure)
    – Redirect to the Index
  (c) Else, redirect to error page and display message
  (d) If exception is caught, redirect to error page and display message
- Search for specific equipment and software
  (a) Get the equipment from the collection of equipment
  (b) Get the software from the collection of software
  (c) Get the list of campuses

(d) Create a new model inquire list of type Room and inquire based on given software, equipment and/or campus
(e) If the status code is greater than or equal to zero, return the View with the search results
(f) Otherwise, redirect to error page and display status description
(g) If exception is caught, redirect to error page and display message

4. Software

- Index
  (a) Create a model inquire list of type software and inquire all software
  (b) If the status code is greater than or equal to 0, return the list to the view
  (c) Otherwise, redirect to Error page (display the status description)
- Details (given a specific ID)
  (a) Create a model inquire list of type software and inquire the software based on the given id
  (b) If the status code is equal to 1, return the object to the view
  (c) Else, redirect to Error page (display the status description)
- Create
  (a) Create a new software object
  (b) Add the new software object name to the collection of software
  (c) Run the Create method (stored procedure)
  (d) Redirect to the Index
  (e) If exception is caught, redirect to error page and display message
- Edit
  (a) Create a new software object
  (b) Add the new software object name to the collection of software and set the software id
  (c) Run the Edit method (stored procedure)
  (d) Redirect to the Index
  (e) If exception is caught, redirect to error page and display message

5. Equipment

- Index
  (a) Create a model inquire list of type Equipment and inquire all Equipment
  (b) If the status code is greater than or equal to 0, return the list to the view
  (c) Otherwise, redirect to Error page (display the status description)

- Details (given a specific ID)
  (a) Create a model inquire list of type Equipment and inquire the Equipment based on the given id
  (b) If the status code is equal to 1, return the object to the view
  (c) Else, redirect to Error page (display the status description)
- Create
  (a) Create a new Equipment object
  (b) Add the new Equipment object name to the collection of Equipment
  (c) Run the Create method (stored procedure)
  (d) Redirect to the Index
  (e) If exception is caught, redirect to error page and display message
- Edit
  (a) Create a new Equipment object
  (b) Add the new Equipment object name to the collection of Equipment and set the Equipment id
  (c) Run the Edit method (stored procedure)
  (d) Redirect to the Index
  (e) If exception is caught, redirect to error page and display message

### Error Handling

The *ModelInquire* class is built to facilitate error handling. Data access layers will return this, containing the model of a specific type (single object, list, etc.) along with a status code and status description.

- For models that return a list, a negative status code will return specific error messages while positive status code will signify the number of records found.
- For models that return a single object, negative numbers will return a specific error message, 0 will signify that no record was found, and greater than 1 will signify that multiple records were found for the search criteria.

## Current Issues and Future Implementations

### Current Database Issues and Changes for ERD

1. LKUPStatusCode table is not required anymore because all status codes for deletion are either 1(deleted) or 0(active)
2. FolderPath table needs to be connected to to attachments
3. Qualification Approval process needs to be implemented and the table needs to be connected to the instructor qualifications table
4. Qualifications needs an *isdeleted* field

## Future MVC Implementations

1. Course search functionality was not implemented and it will need additional search queries (stored procedures)

2. Layout was tweaked but not finalized and it will need revision. Bootstrap x4 is currently used but the team was not able to utilize the resource at their full capacity

3. Models/Controllers/Views pertaining to Qualifications, Instructors, and other use cases. A list of complete and incomplete use cases can be found at the end of this document

Table 1: Error Log

| Date | Logged By | Error # | Status |
|---|---|---|---|
| 11/28/18 | Stefana Rusu | #1 | Open |
| 12/09/18 | Stefana Rusu | #2 | Closed |
| 12/09/18 | Devon Fairman | #3 | Open |
| 12/09/18 | Devon Fairman | #4 | Closed |

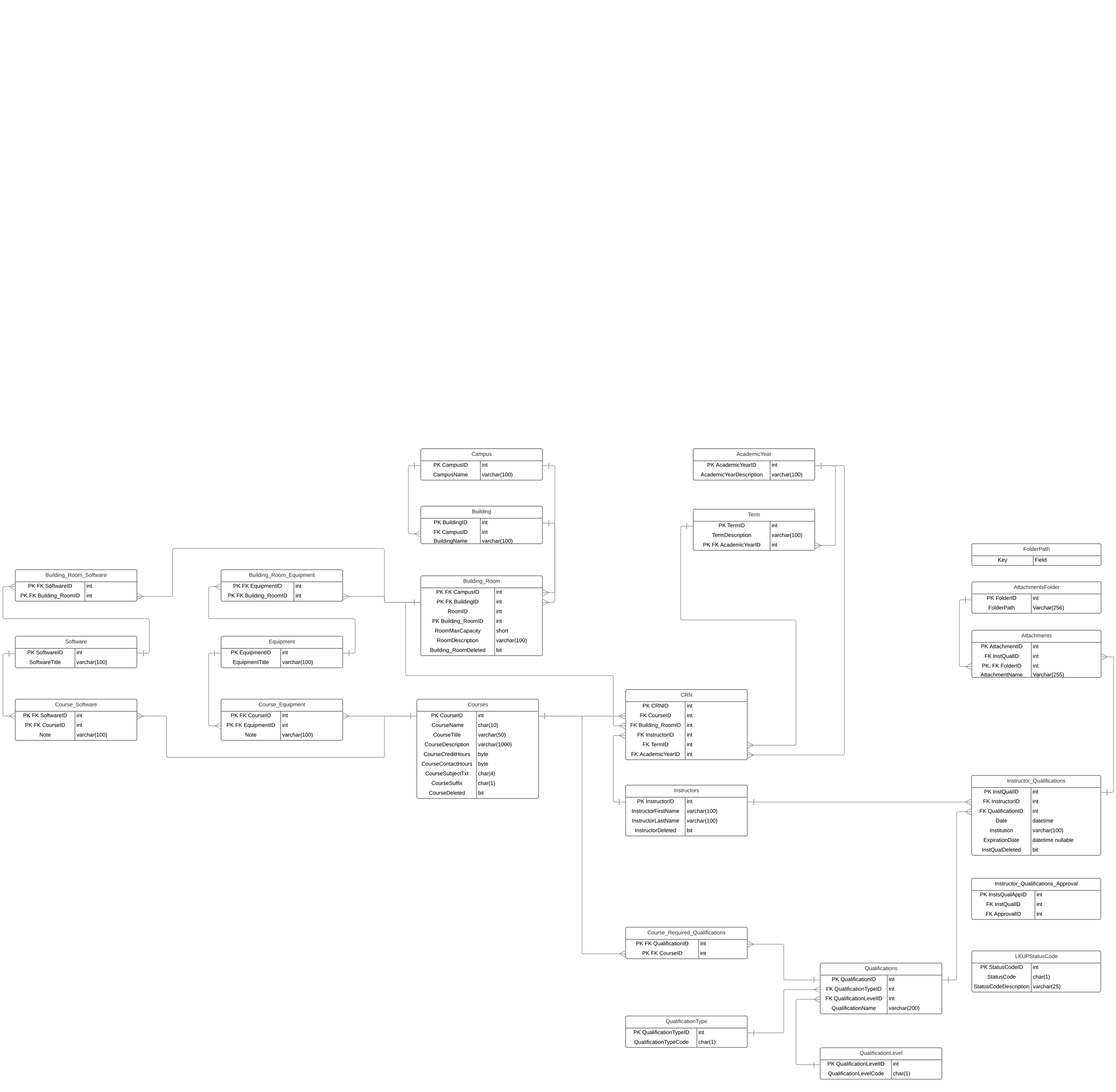
## MVC Error Log

1. **Error #1**: There seems to be a logic error in the stored procedure for searching for software and equipment. It may be the joining of the tables. For example, when searching for the word "printer", the records that contain "Printer (Regular)" show up, but the records that contain "3D Printer" are not displayed. **This error is open.**

2. **Error #2**: Upon implementing the error handling (model inquire class of type room), the view was not changed to reflect the new implementation. The system was built with no errors/ flags; however, the Create view crashed when tested. **This error is closed.**

3. **Error #3**: The Search view for the Course Controller is not functioning properly. There may be problem with the view or with the stored procedure. The current fix is to just comment out the view. **This error is open.**

4. **Error #4**: There are issues with the NuGet package when trying build or push or pull the project. Make sure to add the Nuget folder to the packages folder under repos if this issue arises. **This error is closed.**


Campus
PK CampusID int
CampusName varchar(100)
Building
PK BuildingID int
FK CampusID int
BuildingName varchar(100)
AcademicYear
PK AcademicYearID int
AcademicYearDescription varchar(100)
Term
PK TermID int
TermDescription varchar(100)
PK FK AcademicYearID int
FolderPath
Key Field
AttachmentsFolder
PK FolderID int
FolderPath Varchar(256)
Attachments
PK AttachmentID int
FK InstQualID int
PK, FK FolderID int
AttachmentName Varchar(255)
Building_Room_Software
PK FK SoftwareID int
PK FK Building_RoomID int
Building_Room_Equipment
PK FK EquipmentID int
PK FK Building_RoomID int
Building_Room
PK FK CampusID int
PK FK BuildingID int
RoomID int
PK Building_RoomID int
RoomMaxCapacity short
RoomDescription varchar(100)
Building_RoomDeleted bit
Software
PK SoftwareID int
SoftwareTitle varchar(100)
Equipment
PK EquipmentID int
EquipmentTitle varchar(100)
Course_Software
PK FK SoftwareID int
PK FK CourseID int
Note varchar(100)
Course_Equipment
PK FK CourseID int
PK FK EquipmentID int
Note varchar(100)
Courses
PK CourseID int
CourseName char(10)
CourseTitle varchar(50)
CourseDescription varchar(1000)
CourseCreditHours byte
CourseContactHours byte
CourseSubjectTxt char(4)
CourseSuffix char(1)
CourseDeleted bit
CRN
PK CRNID int
FK CourseID int
FK Building_RoomID int
FK InstructorID int
FK TermID int
FK AcademicYearID int
Instructors
PK InstructorID int
InstructorFirstName varchar(100)
InstructorLastName varchar(100)
InstructorDeleted bit
Instructor_Qualifications
PK InstQualID int
FK InstructorID int
FK QualificationID int
Date datetime
Institution varchar(100)
ExpirationDate datetime nullable
InstQualDeleted bit
Instructor_Qualifications_Approval
PK InstsQualAppID int
FK InstQualID int
FK ApprovalID int
Course_Required_Qualifications
PK FK QualificationID int
PK FK CourseID int
Qualifications
PK QualificationID int
FK QualificationTypeID int
FK QualificationLevelID int
QualificationName varchar(200)
LKUPStatusCode
PK StatusCodeID int
StatusCode char(1)
StatusCodeDescription varchar(25)
QualificationType
PK QualificationTypeID int
QualificationTypeCode char(1)
QualificationLevel
PK QualificationLevelID int
QualificationLevelCode char(1)

| USE CASE/ TASK | Expected Result | Actual Result | Notes |
|---|---|---|---|
| What Faculty can teach specific courses? | Complete | Incomplete | |
| What certifications are required to teach a course? | Complete | Incomplete | |
| What software is required for a course? | Complete | Partial | Complete in database/ NOT MVC |
| What hardware is required for a course? | Complete | Partial | Complete in database/ NOT MVC |
| What certifications does an instructor have? | Complete | Incomplete | |
| What degree does an instructor have? | Complete | Incomplete | |
| What faculty members have reached their max credits per term? | Complete | Incomplete | |
| What faculty have the required credentials to teach a course? | Complete | Incomplete | |
| What faculty have expired credentials? | Complete | Incomplete | |
| What faculty have expiring credentials? | Complete | Incomplete | |
| What faculty have a specific status (adjunct, associate, department chair, etc.)? | Complete | Incomplete | |
| What are the available classrooms for a specific course? | Complete | Partial | Complete in database/ NOT MVC |
| What classrooms have specific software? | Complete | Complete | Complete in database and MVC |
| What classrooms have specific equipment? | Complete | Complete | Complete in database and MVC |
| Does an instructor meet all requirements? | Complete | Incomplete | |
| Create Room | Complete | Complete | Complete in database and MVC |
| Create Instructor Record | Complete | Partial | Complete in database/ NOT MVC |
| Create Credential Record | Complete | Partial | Complete in database/ NOT MVC |
| Create Course Record | Complete | Complete | Complete in database and MVC |
| Create Building | Complete | Complete | Complete in database and MVC |
| Create Campus | Complete | Complete | Complete in database and MVC |
| Create Software | Complete | Complete | Complete in database and MVC |
| Create Equipment | Complete | Complete | Complete in database and MVC |
| Delete Instructor Record | Complete | Partial | Complete in database/ NOT MVC |
| Delete Credential Record | Complete | Incomplete | |
| Delete Course Record | Complete | Complete | Complete in database and MVC |
| Delete Room | Complete | Complete | Complete in database and MVC |
| Delete Building | Complete | Complete | Complete in database and MVC |
| Delete Campus | Complete | Complete | Complete in database and MVC |
| Update Employee Record | Complete | Incomplete | |
| Update Credential Record | Complete | Incomplete | |
| Update Course Record | Complete | Complete | Complete in database and MVC |
| Update Room | Complete | Complete | Complete in database and MVC |
| Update Building | Complete | Complete | Complete in database and MVC |
| Update Campus | Complete | Complete | Complete in database and MVC |
| Update Software | Complete | Complete | Complete in database and MVC |
| Update Equipment | Complete | Complete | Complete in database and MVC |