**COURSELY DEVELOPER GUIDE**

VERSION 1.0

BY: CHRISTEN FORD

# 1.0 Introduction and Background

## 1.1 Introduction

Coursely is course management software. Coursely was motivated through my frustration with modern course management software. In my experience in higher education, most of this software tends to either be overly complicated, or it fails to provide correct functionality that course management software should provide. Coursely was built with the intention to alleviate these issues. I have designed it so that there is as small a learning curve as possible. The user interface of Coursely has been designed so that it requires as little input as possible from the user.

Coursely assigns users to one of three roles: Administrator, Instructor, and Student. Coursely provides a unified platform for students to maintain their schedule (by enrolling an unenrolling from courses), instructors to manage their courses, and advisors to manage their advisees. There is also functionality built into the system to allow administrators to manage the various data sources used by the system. The user interface is setup to automatically respond to each role and present different views to each role.

The purpose of this developer guide is to provide maintainers of the Coursely software with a unified location to lookup important design information regarding Coursely. This developer guide includes information regarding the architecture of Coursely, a maintenance guide, and remarks on the implementation as well as suggestions for future improvement of the software. If at any point, this guide fails to provide information needed by the maintainer, the reader is free to utilize any of the other major deliverables of the Coursely software that provide more fine-grained details as to the design and implementation of the software.

## 1.2 Objectives

As state earlier, Coursely is course management software. That said, Coursely implements most of the expected functionality of course management software. It does not implement the functionality found in a product like Blackboard, instead it focuses more on a subset of the functionality provided by a product like Cleveland State Universities, CampusNet. Coursely explicitly provides the following functionality:

* Students can manage their schedule including:
  + Enrolling/unenrolling from course sections
  + Viewing their schedule
  + Viewing their grades
* Instructors can manage their advisees by:
  + Viewing their advisees
  + Viewing their courses
  + Overriding prerequisites for their advisees
  + Viewing advisee grades
  + Viewing advisee schedules
* Administrators can manage the system:
  + Adding users
  + Adding and modifying academic schools (i.e School of Nursing)
  + Adding and modifying departments
  + Adding and modifying courses
  + Adding and modifying course sections
  + Adding and modifying buildings
* All users can manage their account:
  + Change their email
  + Change their password

## 1.3 Risks

This section details the risks faced by the current implementation of the software. The goal is to detail potential issues with the software and potential risks it faces.

* Coursely is exception heavy. What I mean by that is that whenever an issue occurs at the Logical layer and below, that exception is passed all the way up to the presentation layer where it is then caught. This may not be the best way to handle issues under-the-hood, however it was the best method I could think of. This of course comes with some caveats:
  + Exceptions are memory intensive. Each time one is thrown the run time must allocate space to store information on that exception. The maintainer should be aware of this fact when working with the code.
  + The better model is to follow what I have implemented in the JavaScript error handling in the various Validate files in the JS folder. In this model, one would have their method return true or false depending on the status of the method. C# fortunately allows the return of multiple values form a method using the ‘out’ keyword. If we use the out keyword to signify we are passing out a string from a method alongside the Boolean value, then we have essentially replicated the current error handling system without the overhead of using Exceptions.

# 2.0 Architecture

## 2.0 Software Architecture

When designing Coursely, I explicitly chose to utilize a Layered architecture. I had briefly considered employing the MVC architecture pattern, however time constraints on the project alongside my inexperience with the pattern in ASP.NET led to me choosing a Layered architecture instead. This has led to some curious implementation details whereby the system mimics an MVC architecture (stressing that this is not an implementation of MVC).

The architecture diagram for Coursely is shown below. Proceeding the architecture diagram is a description of all the components that make up the diagram.

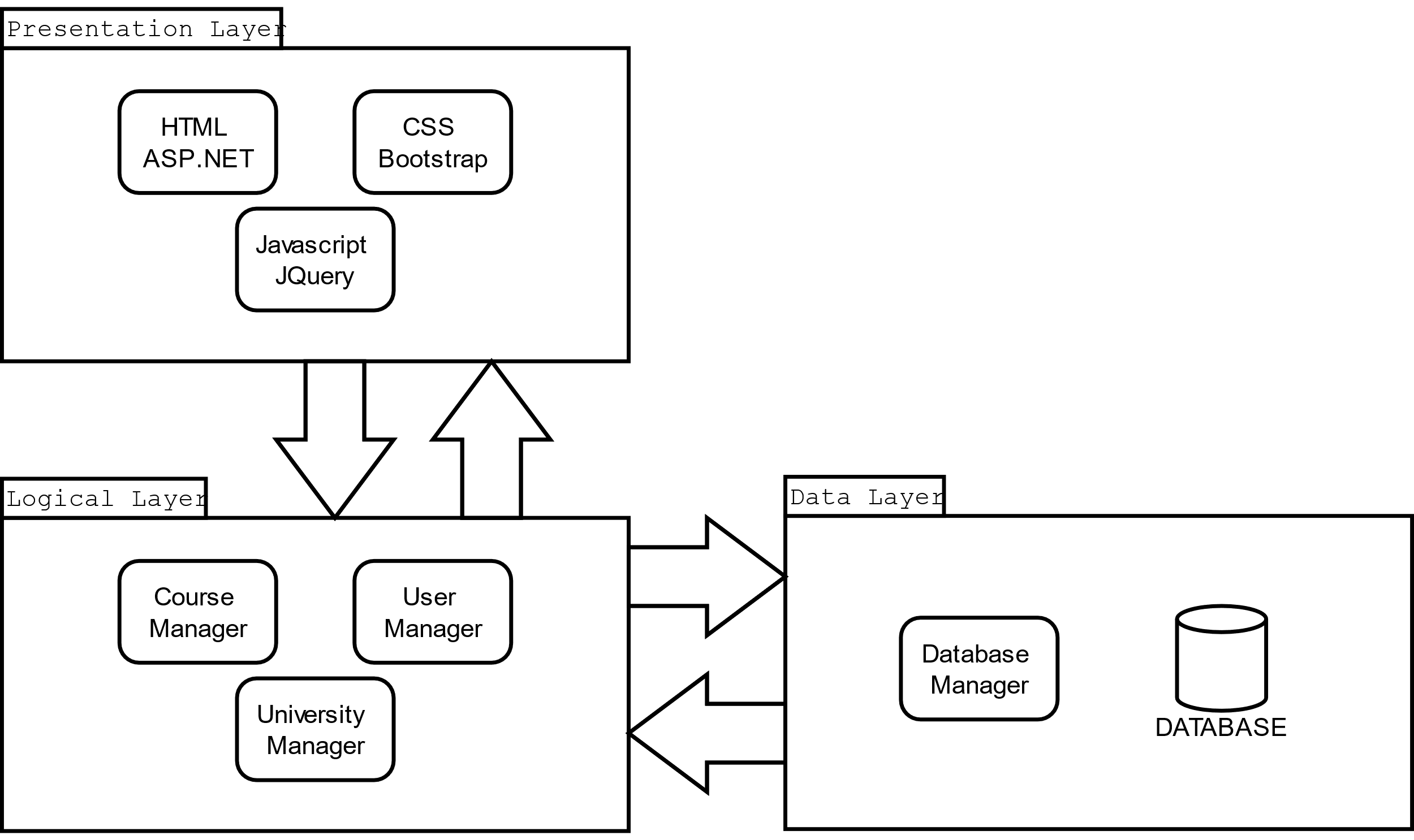
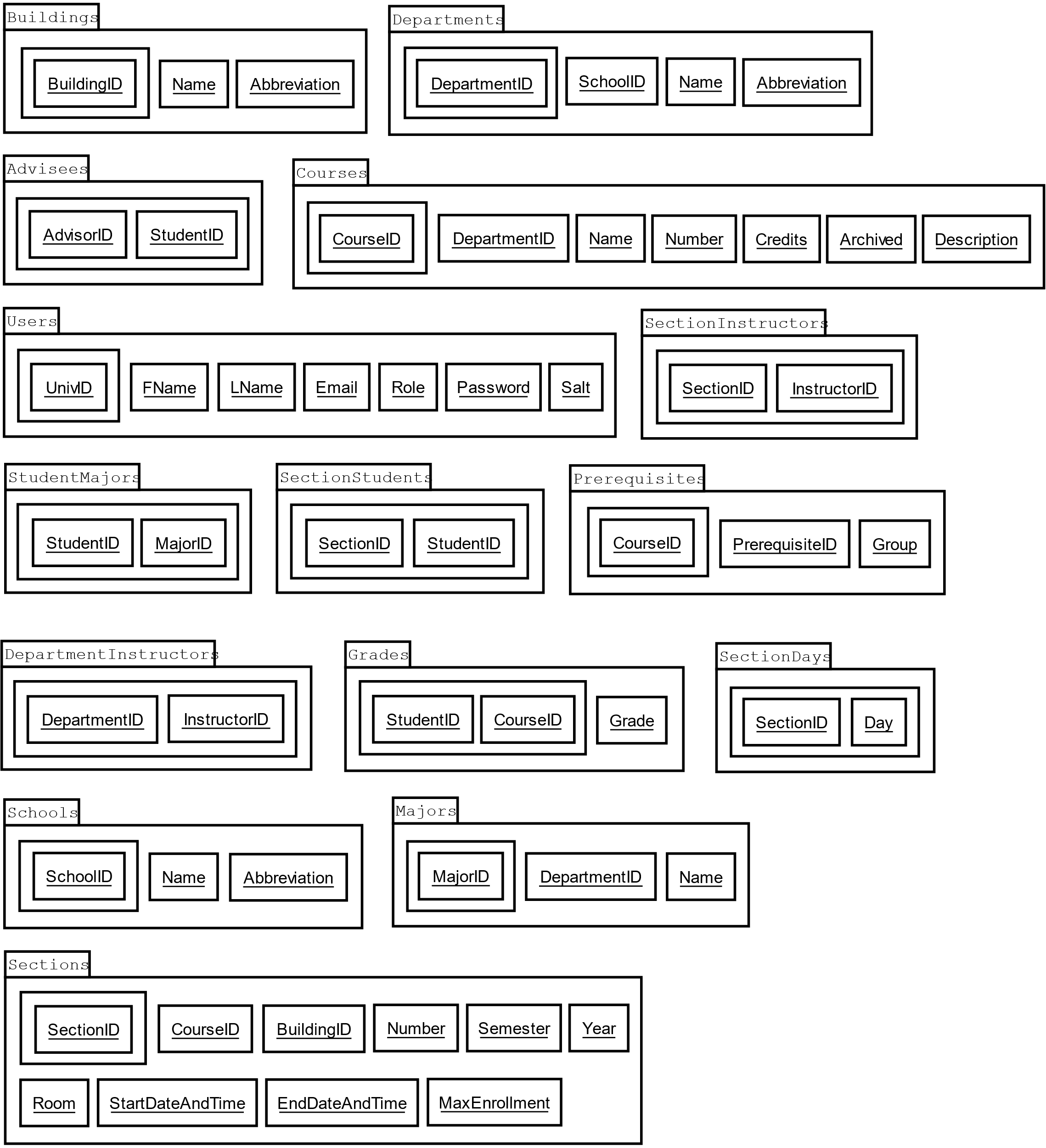


Figure 1: Coursely Architecture Diagram

## 2.1 Database Design

Coursely follows a relational database model that approximates third-normal form. There were some decisions regarding certain tables where I though it would be more prudent to just repeat data in a column rather than create a small (3-7 row) table for it (such as with days, and years). This data is checked for correctness before it is entered into the database anyway, so no invalid values are can exist in the affected tables.

Coursely’s database design schema is listed below:



Following is a description of the tables in presented in the ERD as well as their relations:

|  |  |  |
| --- | --- | --- |
| Description of the Tables Present in the Coursely ERD | | |
| Table Name | Related To | Description |
| Advisees | Users | Holds the relationships between an advisee and advisor. |
| Buildings | N/A | This table holds records related to buildings on campus. It is used by the Sections table to indicate where the section is held. |
| Courses | Departments | This table holds records regarding a course, it is one of the primary tables in the database model. |
| DepartmentInstructors | Departments, Users | This table relates instructors to the departments they belong to. |
| Departments | Schools | This table holds records regarding a department within a school. |
| Grades | Courses, Users | This table holds a student’s academic record. |
| Majors | Departments | This table holds records regarding academic majors. |
| Prerequisites | Courses | This table holds records relating courses to their prerequisites. |
| Schools | N/A | This table holds records regarding academic schools. |
| SectionDays | Sections | This table holds records regarding the days on which a section is held. |
| SectionInstructors | Sections, Users | This table holds records regarding the instructors that teach a section. |
| Sections | Buildings, Courses | This table holds records regarding a course section. |
| SectionStudents | Sections, Users | This table holds records regarding the students enrolled in a section. |
| StudentMajors | Majors, Users | This table holds records relating students to their majors. |
| Users | N/A | This table holds records regarding users in the system. |

Following is a presentation of Coursely’s data dictionary:

|  |  |  |
| --- | --- | --- |
| **Data Dictionary for Coursely** | | |
| **Name** | **Data Type** | **Description** |
| Abbreviation | String | The abbreviation of a name. Constrained to 4 characters in length. |
| AdvisorID­ | Integer | Used to pair an instructor with a student. Foreign key of UnivID. |
| Archived | Boolean | Whether a course is archived or not. |
| BuildingID | Integer | Uniquely identifies a building in the system. |
| CourseID | Integer | Uniquely identifies a course in the system. |
| Credits | Integer | The amount of credit hours a course is worth. |
| Day | String | One of: {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday} |
| DepartmentID | Integer | Uniquely identifies a department in the system. |
| Description | String | A description for a course. Constrained to 1,024 characters in length. |
| Email | String | A users email address, constrained to 128 characters in length. |
| EndDateAndTime | DateTime | An end date and end time. Used to indicate the last day of a section as well as the time the section ends each day. |
| FName | String | A users first name. Constrained to 32 characters in length. |
| Grade | String | A grade that a student has received for a course. |
| GroupID | Integer | Used to group prerequisites together. |
| InstructorID | Integer | Used to relate instructors to sections. |
| LName | String | A users last name. Constrained to 32 characters in length. |
| MajorID | Integer | Uniquely identifies a major in the system. |
| MaxEnrollment | Integer | The maximum amount of student that can enroll in a section. |
| Name | String | The name of aa building, course, department, major, or school in the database. Constrained to 64 characters in length. |
| Number | String | A course number. Constrained to 4 characters in length. Stored as a string for easier formatting. |
| Password | String | A hashed and salted password. Constrained to 64 characters in length. |
| PrerequisiteID | Integer | A prerequisite for a course. References a CourseID. |
| Role | String | A role of a user, one of: {Administrator, Instructor, Student} |
| Room | Integer | A room number. Used to identify where a course section is held. |
| Salt | String | The salt used to salt the password. Must be stored to validate the password prior to login. Constrained to 32 characters in length. |
| SchoolID | Integer | Uniquely identifies a school in the system. |
| SectionID | Integer | Uniquely identifies a section in the system. |
| Semester | String | A semester at the university. One of: {Fall, Spring, Summer} |
| StartDateAndTime | DateTime | A start date and start time. Used to indicate the first day of a section as well as the time the section begins each day. |
| StudentID | String | Used to link a student to a section. References a UnivID. |
| UnivID | String | A university identifier. Uniquely identifies a user in the system. |
| Year | Integer | A year. |

## 2.2 Class Level Design

This section explains every class present in Coursely, as well as the interactions between them. I will also provide rationale behind the design of several of the more complicated classes in the system. The classes listed below are organized as they appear in the Visual Studio solution: First by folder, then by class.

### 2.2.1 Cache

This folder contains classes related to an in-memory cache.

#### 2.2.1.1 CacheItem

Implements a key-value pair cache item. Both the key and value are generic for increased flexibility. A cache item implements methods for accessing and mutating both the key and value. It is the primary class utilized by the MemoryCache class.

#### 2.2.1.2 MemoryCache

Implements an in-memory cache that stores key-value pair CacheItems. The MemoryCache class implements an aging mechanism for paging. It also utilizes the C# delegate design model to dynamically invoke a method handler for page events. This provides a direct benefit for classes that use the MemoryCache class in that they can catch objects as they are paged out of cache and handle them appropriately. The MemoryCache class employs all of the standard operations one would normally expect from a software-based cache.

### 2.2.2 Classes

This folder contains classes related to data bearing classes as well as some miscellaneous classes.

#### 2.2.2.1 Building

Represents a building on campus. This class is utilized by sections to indicate where the section is held on campus. It contains information such as the building name and name abbreviation.

#### 2.2.2.2 Course

Represents a course on campus. This class is one of the primary classes used by the system. It includes information such as the name, number, department, description, credits, and whether the course is archived or not. This class is liberally utilized by almost every component of the system. If something is wrong with the system, it can likely be traced back to the utilization of this class in some component.

#### 2.2.2.3 DataTypeExtensions

This class contains several C# style extension methods that provide additionally functionality for certain data types. For instance, there is a method defined in this class that determines whether a generic array contains a certain element.

#### 2.2.2.4 Department

This class is utilized in several places within the system. It holds information such as the school the department belongs to, the department name, and the departments name abbreviation.

#### 2.2.2.5 Major

This class holds information related to academic majors. Every student user in the system must have at least one major. There is not a maximum limit to the number of majors a student may have. It holds information pertaining to the major’s name, and name abbreviation.

#### 2.2.2.6 School

This class represents an academic school. It contains information such as the school name, and name abbreviation.

#### 2.2.2.7 Section

This class is one of the other primary class utilized by the system (the other being the Course class). This class contains information such as the section number, start and end date, start and end time, building, room number, semester, year, and max enrollment.

#### 2.2.2.8 User

This class is the most important class utilized by Coursely. It provides information regarding users such as their name, email, username, role, and various other pieces of information. A user is registered with the system whenever they login and deregistered whenever they log off the system. This prevents a user from having more than one active session. It also prevents a user from manipulating the web controls client-side to view pages their role does not permit them to view.

### 2.2.3 Managers

This folder contains classes related to managing the interactions between the front-end graphical user interface and the back-end database. These classes are also used to cache the classes they are responsible for maintaining.

#### 2.2.3.1 CourseManager

This class is responsible for providing access to and maintaining synchronicity across courses and sections. It provides middleware versions of the course and section methods found in the DatabaseManager class. The GUI interacts with this class to request information regarding courses and sections.

#### 2.2.3.2 DatabaseManager

This class does all the heavy lifting involved with interacting with the database. This is the DatabaseManagers sole responsibility. Most exceptions regarding the data classes can usually be traced back to this class. The CourseManager, UniversityManager, and UserManager classes make use of this class to indirectly interact with the database. Many of the methods in this class are wrappers around SQL code.

#### 2.2.3.3 UniversityManager

This class is responsible for providing access to as well as synchronicity across the Building, Department, Major, and School data classes. The GUI interacts with this class to request information regarding the aforementioned classes.

#### 2.2.3.4 UserManager

This class is responsible for providing access to as well as synchronicity across the User class. There was at one point a Schedule class as well that this class would have been responsible for maintaining. However, I made the decision to retrieve it from the UserManager in an ad-hoc manner. That said, there is no Schedule class. Schedules are generated by the UserManager but not cached.

## 2.3 Software Interfaces

Coursely utilizes several outside software packages to accomplish its objectives. When installing Coursely, the maintainer must be certain (in most cases) that the following software packages are installed on the hosting system:

* Visual Studio: Official Microsoft IDE for .NET. Coursely was implemented inside Visual Studio Community 2017. It is recommended that the maintainer utilize at least this version of the software,
* .NET Framework 4.6.1: Coursely targets .NET Framework version 4.6.1. The maintainer must have at least this version of the .NET Framework installed on their machine if they want to work with the Coursely software.
* SQL Server Express: This software ships with Visual Studio (if the proper packages are installed with it). SQL Server Express was used during development as the database provider for Coursely. It is highly suggested that the maintainer transfer the database schema to a full-fledged SQL Server or Azure instance on dedicated hardware. It is not recommended to run Coursely on the same physical hardware that the database exists on.
* JQuery\*: Provides advanced JavaScript functionality to the system and supports the bootstrap theming system.
* Bootstrap\*: Provides basic theming capabilities to the system.
* NUnit 3: Coursely utilizes NUnit for automated unit testing. The maintainer must have NUnit version three installed if they wish to perform automated unit testing on the Coursely software. This software can be installed through the Nuget package manager built into Visual Studio.
* DocFX: Used to automatically generate developer documentation from triple‑slash developer comments. The documentation generated by this tool may be accessed by opening the ‘\_site’ folder in the root directory of the Coursely project solution and then opening ‘index.html’ in their browser. This package can be installed through the Nuget package manager built into Visual Studio.
* Microsoft IIS: This specific software can be installed and configured when installing Visual Studio (if testing/developing the software) or installed separately on a server (if hosting the software). In the former case, it is as easy as opening the Visual Studio installer and selecting the ASP.NET option under manager components. In the latter case, you would follow the steps provided by Microsoft (I am of no help here, as I do not have any experience setting up a IIS backed web server myself).
* DIA diagramming tool: All of the models for the system were built using DIA. DIA is free software distribute by the GNU project. It is available on both Windows and Linux and is a great alternative to Microsoft Visio. In order to work with the models and physically alter them, you will need DIA installed on your system. I have also taken care
* Github Version Control: Available at https://github.com, this web service hosts the project. The maintainer must be familiar with its operation.

\*Note: JQuery and Bootstrap are currently setup to pull from official Content Distribution Networks for the respective software systems. It is not necessary to do anything during installation of the Coursely software to ensure either system will function.

# 3.0 Conclusion

## 3.1 Remarks on Implementation

This section contains my remarks regarding the current implementation of Coursely available at: https://github.com/gollum18/CourseRegistrationPOC. This implementation is nearly complete (I estimate that ~95% of the functionality outlined in the Requirements Spec. is met). For instance, advisors cannot currently override prerequisites for their advisees. Likewise, I ran out of time to complete the theming of the system. It has partially met the layout specified in the Design Specification, however the arrangement of the controls within the system are not as I had specified.

Also, due to the time constraints and limited time I personally had to work on this project, I made some implementation decisions that are not optimal from a performance standpoint. I have tried to outline these areas within the source code of the system. However, I may have missed some areas. I have tried my best to provide developer comments for every method but there are some methods that I have missed (the stubs are there, they but they are not filled in).

## 3.2 Possible Future Improvements

Coursely was hastily implemented by one person in a ~2 and ½ month timeframe. As I have emphasized throughout this document, there are some minor issues with the current implementation of Coursely. Fortunately, for the maintainer, this presents some opportunities to improve the software:

* Complete the developer comments for the system. This should be easy, as the methods are all self-explanatory. Likewise, they have common sense naming that clearly specify what the method should do.
* Complete (or redesign) the layout of the GUI. I would prefer just completing the layout of the user interface. Redesigning it is not necessary as there is complex user interface logic in several of the pages within the system. However, I am also not a graphical designer, therefore it is highly likely that someone else with that particular skillset could design it better than I.
* Refactor the DatabaseManager class. I always meant to do this, but never got around to it. The DatabaseManager class in its current form is ~4,500 SLOC (including a few hundred lines of comments). Much of this is repeated code due to the setup necessary to interact with the database. A keen developer should be able to extract this code into a method (or small series of methods) that should make it so that all you have to do is pass in the information needed by each SQL query as well as what it will return. Many of the methods should be able to be replaced my stubs that call the setup method and pass in the required SQL information.
* Refactor some of the methods within the manager classes so that they are simpler. There are several methods in most of the manager classes that are complex that should be able to be refactored to a simpler, shorter form.
* Update the sequence diagrams to reflect the current state of the system. This may be an arduous task as there are around ~18 sequence diagrams for the various use cases in the system. While I have followed the layered architecture, design specified at the beginning of this document, many methods intermingle with each other. This would make updating the system diagrams more difficult as the maintainer has to trace their way through the entire system, often times from the GUI to the database.
* Move the database to a full-fledged SQL Server instance of Azure instance. This would centralize the database and allow maintainers to work independently of each other (as currently the database is hosted inside the GitHub repo for the project.