Software Design Document

April 3, 2020

Team 3 Owen, Gracie, Abby, Ben, Brad, Katie, Ana

Introduction:

This report describes Team Software Design for our project

1. **Introduction (generally similar to the introduction in your architecture document)**

1.1. Purpose

The purpose of this project is to build a gradebook for the mastery system, modeled after the system used by Davidson professor Dr. Heather Smith. This document provides detailed information about the elements we are including and how they will be constructed.

1.2. Scope

This document covers a basic version of our application where one professor can set up a course and enter grades for that course. Additionally, students can access that course and view their grades. This document does not specify details allowing a professor to create multiple courses within the same instance of the application. That is, if the professor wants to create a new course they would need to create a new instance of the application.

1.3. Definitions, Acronyms, and Abbreviations

- Mastery Grading System (MGS): A grading system in which students must achieve a complete understanding of a topic before receiving full credit for that topic. The final grade is determined by the number mastered over the total number of concepts in the class.
- Open Database Connectivity (ODBC): A standard applications programming interface (API) for accessing database management systems.
- Free TDS: set of libraries for Linux that allows programs to communicate with a Microsoft SQL Server

1.4. References

In this document, we reference the C&C diagram from our Architecture Report. We referenced our Project Management Report to obtain an estimated size of each module in LOC.

1.5. Overview

Our web application will consist of three main components. The student and professor dashboards, the database, and the servers hosting the application and database.

We need our application to display student grades and their progress in the course. The aggregate of this information will be available in the professor's gradebook. Professors can modify assignments, grades, and the grade scale throughout the course and the application will be responsive to these changes.

The database will store student, assignment, and grade information for a given course and will be hosted on a Data Cats server through Davidson. Our web application will access this information to display course data.

The application is written in R and developed using the Shiny package. Then, we can use a Shiny IO server to host our application. This Shiny IO server can communicate with our SQL database that is hosted by Davidson's Data Cats server. Together, these components communicate and exchange data, allowing professors and students to set up a course and access that course through the web.

2. Software Design Description

2.1. Client- R Shiny Application

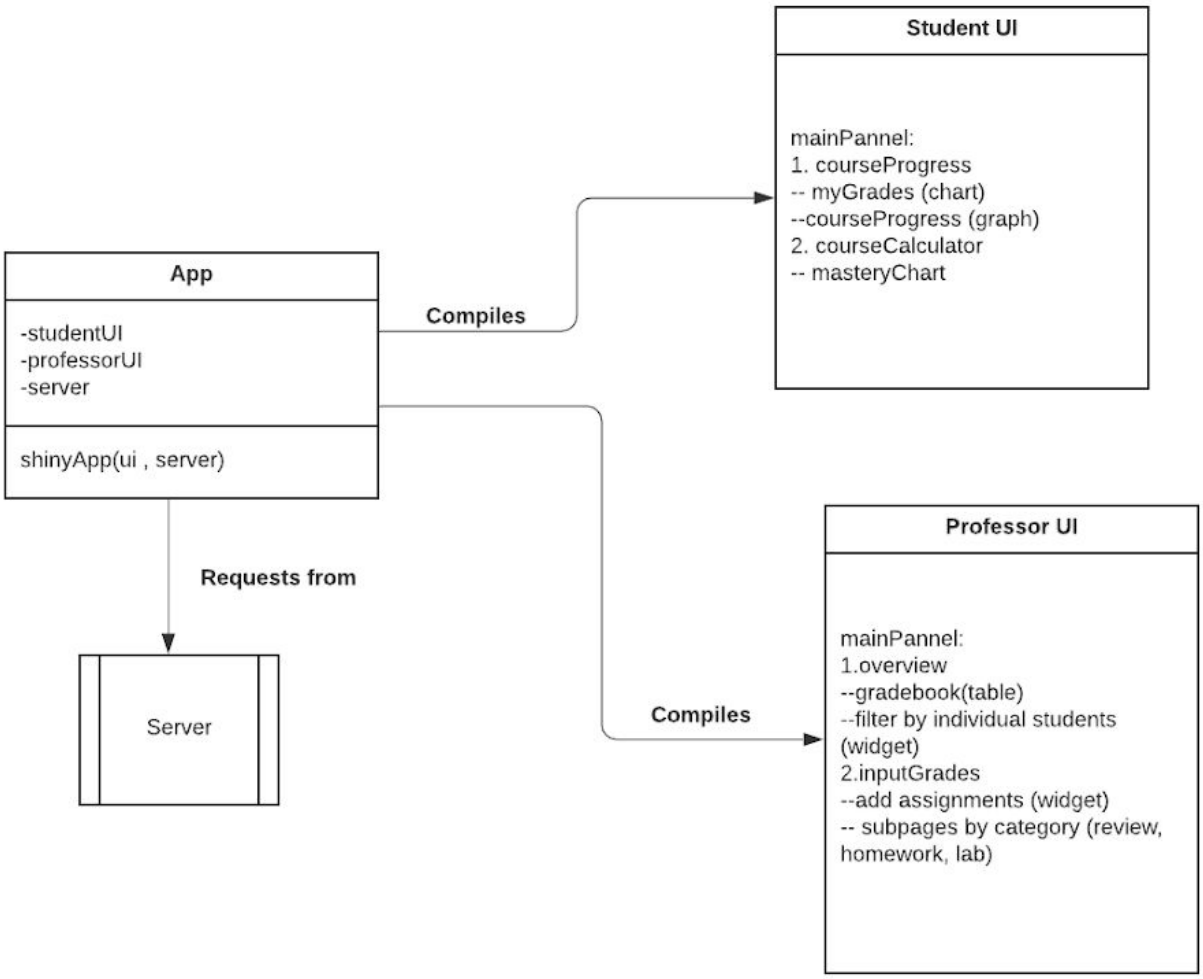
2.1.1. Design Overview

We will use an Object-Oriented approach and have classes for the client and server following the Shiny IO layout. The client information and functions will be stored in their own class and compiled in another class with the server to launch the application.

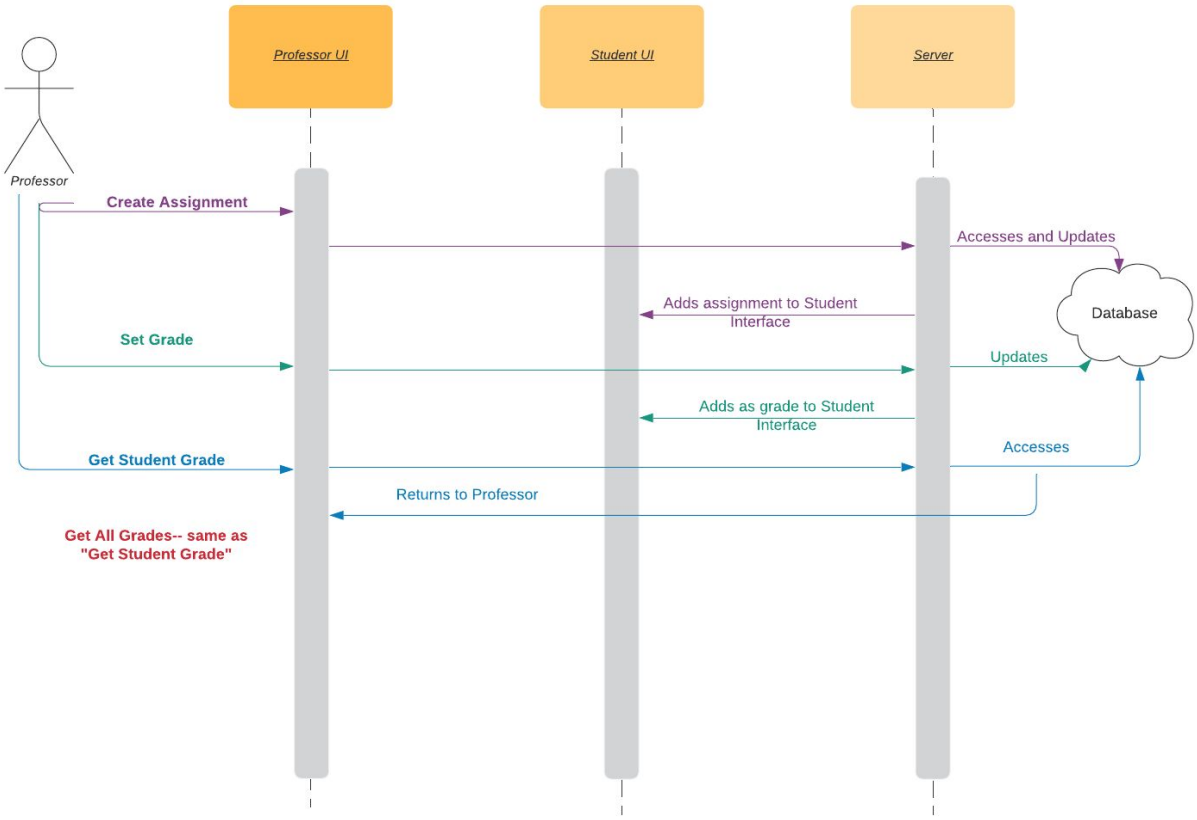
2.1.2. Language and Infrastructure

We will program using the R language and the Shiny package.

2.1.3. Class diagram or data structure diagram


Student UI
mainPannel:
1. courseProgress
-- myGrades (chart)
--courseProgress (graph)
2. courseCalculator
-- masteryChart
App
-studentUI
-professorUI
-server
shinyApp(ui , server)
Compiles
Requests from
Server
Compiles
Professor UI
mainPannel:
1.overview
--gradebook(table)
--filter by individual students (widget)
2.inputGrades
--add assignments (widget)
-- subpages by category (review, homework, lab)

2.1.4. Sequence diagram for key use cases



2.1.5. Detailed Design

2.1.5.1. Logic/Algorithm Design

Professor Interface Function Analysis:

setGrade() - professor inputs student grade for associated assignment

getStudentGrade() - professor filters grades by student

getAllGrades() - professor filters grades by assignment

createAssignment() - professor creates new assignment

Student Interface Function Analysis:

getMyGrade() - student views grade on a particular assignment

getAllAssignments() - student views grades received for all assignments

2.2. Shiny IO Server

2.2.1. Design Overview

We will be utilizing an Object-Oriented design. The server will allow the app to be deployed and allow the professor to update the data by accessing it from the Data Cats server, making sure that the app always contains the most recent data.

2.2.2. Language and Infrastructure

R with the Shiny framework

2.2.3. Class diagram or data structure diagram: we are using the Shiny IO server that comes with the R-Shiny package and will not be designing the structures/classes of this server.

2.2.4. Sequence diagram for key use cases

See the sequence diagram under 2.3.4

2.2.5. Detailed Design

2.2.5.1. Logic/Algorithm Design: Again, we are using the Shiny IO server and so we will not be designing any algorithms for this existing server.

2.3. Data Cats Server

2.3.1. Design Overview

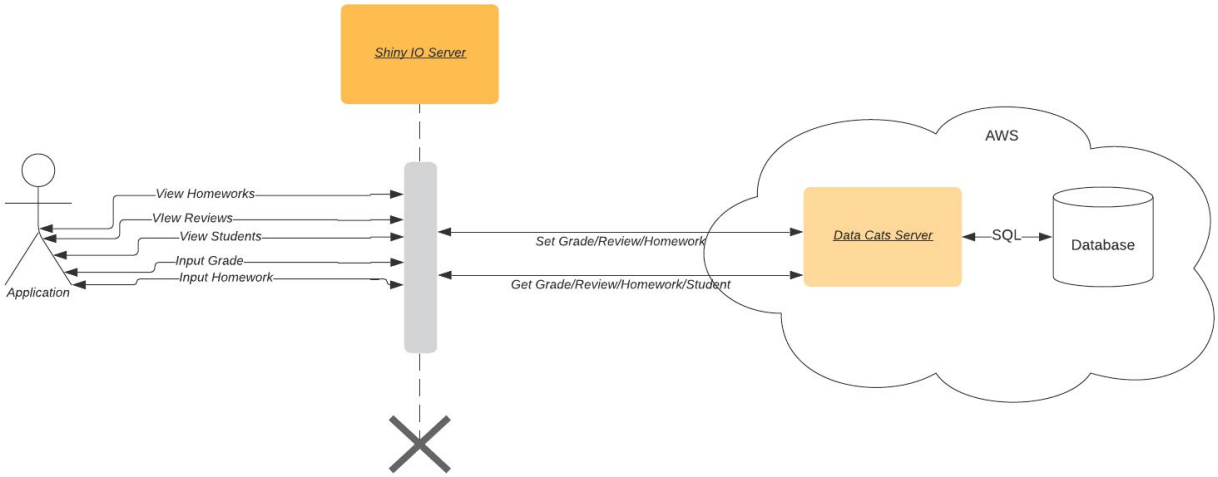
Functional design. Allows data from the database to be stored and updated remotely.

2.3.2. Language and Infrastructure

AWS server.

2.3.3. Class diagram or data structure diagram: we are using Davidson's Data Cats server and will not be designing the data structures/classes of this server

2.3.4. Sequence diagram for key use cases



2.3.5. Detailed Design

2.3.5.1. Logic/Algorithm Design: Again, we are using the Data Cats server and so we will not be designing any algorithms for this existing server

2.4. Database

2.4.1. Design Overview

Functional design that allows data to be stored remotely.

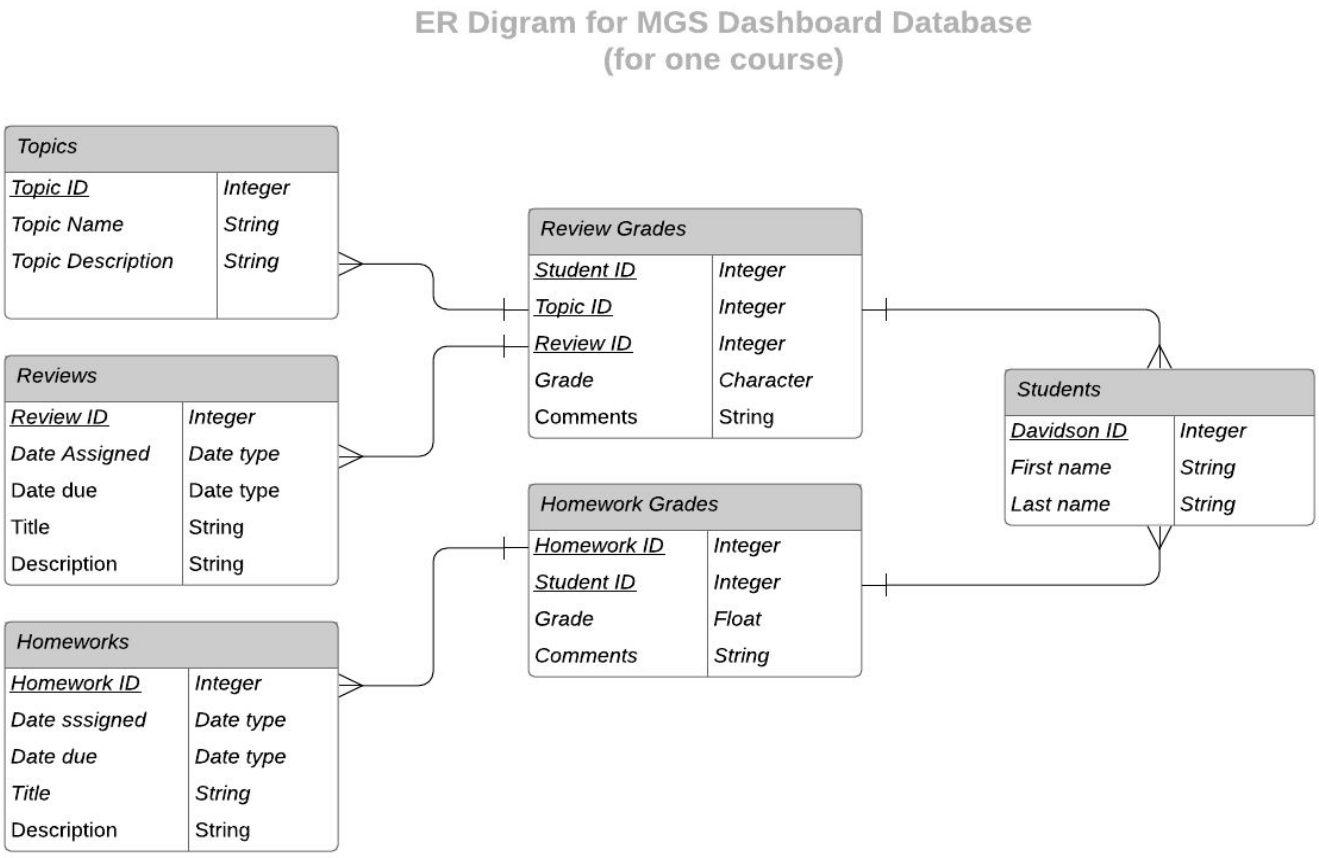
2.4.2. Language and Infrastructure

SQL - relational database schema.

2.4.3. Class diagram or data structure diagram

2.4.4. Sequence diagram for key use cases

We have combined sections 2.4.3 and 2.4.4 by creating an Entity Relationship Diagram for our Database. We believe this diagram provides all of the information we need to design our database. In this ER diagram, we have designed the schema for our SQL database.



2.4.5. Detailed Design

2.4.5.1. Logic/Algorithm Design: In the ER diagram above, we identified the columns that make up the primary key of each table with an underline. Additionally, we can define the relationship between each table in the following way:

1. Topics and Review Grades: many to one
2. Reviews and Review Grades: many to one
3. Students to Review Grades: many to one
4. Homeworks to Homework Grades: many to one
5. Students to Homework Grades: many to one

Overall, we can identify each Review grade by the topic covered, particular review on which this topic was questioned, and the particular student that was tested on this

topic. These grades represent the mastery level for each topic on each review for each student. On the other hand, we can identify homework grades by student and homework. For each homework, a student will be given a typical number grade. These are not part of the mastery system, but are still a part of the grade for each student in the course. As previously stated, this diagram shows the database schema for a single course.

3. Inter-component or inter-subsystem communications

3.1. Client and Shiny IO server

3.1.1. The client will use the Shiny IO server to deploy the app and display the data to the student side of the application. The interfaces will only pass and receive assignment data. The client will send a message to the Shiny IO server during the initial deployment and during each update. If the client makes a request to the server and the server does not respond, send a notification to the client's desktop stating the server is down and to try again later.

3.2. Shiny IO Server and Data Cats Server

3.2.1. The Shiny IO server retrieves and sends client data to and from the Data Cats server. This data is shown to the user by the Shiny IO server which stores the application. Only assignment data will be passed and received between these interfaces. Most errors will be handled by the Shiny IO server limiting the options for data to be sent to the Data Cats server. For example, when the client is selecting grades to give for an assignment, the only data types available for selection will match what is pre-set in the Data Cats server.

3.3. Data Cats Server and Database

3.3.1. Interface description.

The Data Cats Server reads and writes from the Database.

4. Metrics

4.1. Size information

A rough estimate of row counts for each table:

- Topics: ~20 (estimated from Dr. Smith's course syllabi)
- Reviews: ~10 (estimated from Dr. Smith's course syllabi)
- Homeworks: ~10 (estimated from Dr. Smith's course syllabi)
- Review Grades: ~3000 (At most, (number of topics) x (number of reviews) x (number of students))
- Homework Grades: ~300 (number of homeworks x number of students)

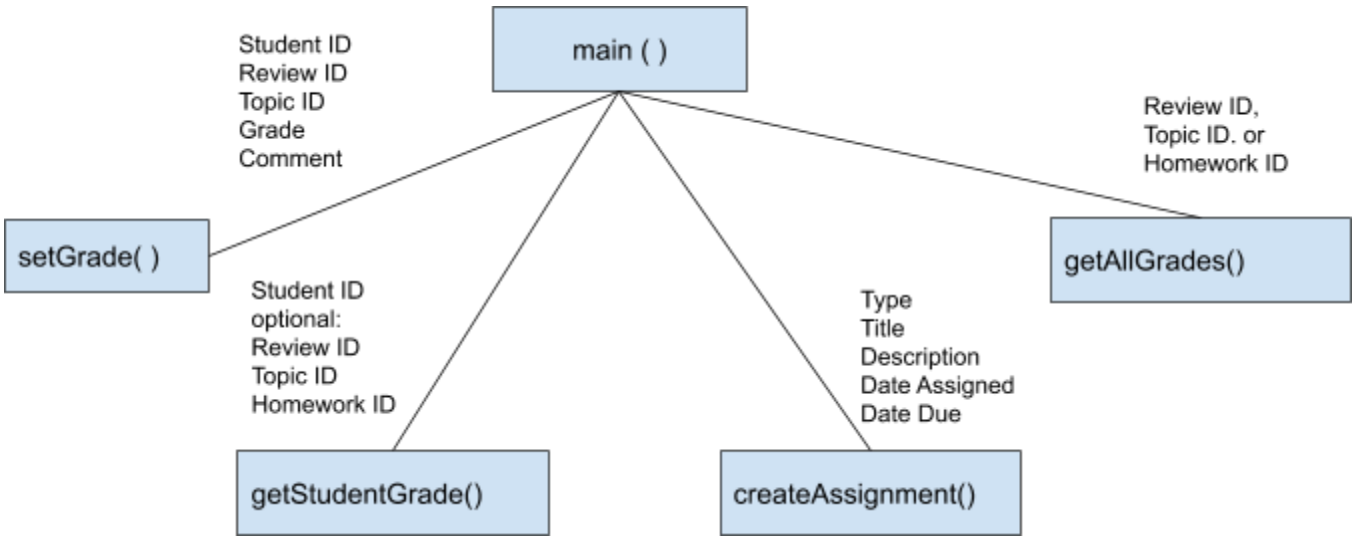
- Students: ~30 (avg size of a Davidson class ~15 x 2 sections of the class)

These estimates can vary depending on the number of students, homeworks, topics and reviews. Additionally, the number of rows in the review grades sections will vary based on student performance. That is, a student who masters each topic early will never be retested on that topic. Therefore, they would not have any more review grades for that particular topic throughout the course.

4.2. Complexity

4.2.1. Network Metrics (using the formula from the textbook/powerpoint)

Professor Side



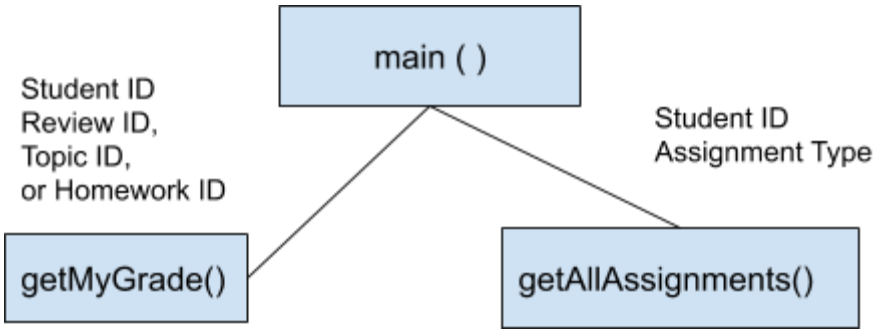
In this Structure chart above,

n = 5

e = 4

Graph Impurity = 4 - 5 + 1 = 0

Student Side



In this Structure chart above,

n = 3

e = 2

Graph Impurity = 2 - 3 + 1 = 0

4.2.2. Information Flow Metrics (using the formula from the textbook/powerpoint)

Student Side

Module size s = ~500 LOC

D_{in}= 2 (review grades, homework grades)

D_{out} = 1 (student information)

Module Design Complexity (D_C) = 500 * $(1 * 2)^2$ = 2,000

Professor Side

Module size s =~500 LOC

D_{in}= 6 (topics, reviews, homeworks, review grades, homework grades, students)

D_{out} = 6 (topics, reviews, homeworks, review grades, homework grades, students)

Module Design Complexity (D_C) = 500 * $(6*6)^2$ = 648,000

mean = 325,000

standard deviation = 323,000

Analysis-

Professor Side: mean < D_C < mean + standard deviation = COMPLEX

Student side: D_c < mean and standard deviation = NORMAL