



Course Name: Software Engineering
Course Number and Section: 14:332:152
Group #10

VirtualLogicLabs

Report #3

Github: <https://github.com/SagarPhanda/VirtualLogicLabs>

Date Submitted: April 22, 2018

Group Members (6):

Sagar Phanda, Khalid Akash, Dhruvik Patel, Vikas Khan, Joe Cella, Yiwen Tao

Individual Contributions Breakdown

All group members contributed equally.

(Each member spent the same amount of time and contributed equally for each segment of the Report. This was worked on in team meetings as well as independently by each team member.)

Table of Contents

Summary of Changes	5
Summary of Changes Part 2: Feedback-based changes	6
Clarification on the Issue of Coding	7
1. Customer Statement of Requirements	8
1.1 Problem Statement	8
1.1.1. The Problem:	8
1.1.2 Proposed Solution:	8
2. Glossary of Terms	11
3. System Requirements	13
3.1 Enumerated Functional Requirements	13
3.1.1 General Requirements	13
3.1.2 Lab 1 Requirements	14
3.1.3 Lab 2 Requirements	14
3.1.4 Lab 3 Requirements	15
3.2 Enumerated Nonfunctional Requirements	15
3.3 On-Screen Appearance Requirements	16
3.3.1 Login UI	16
3.3.2 Student UI	17
3.3.3 Pre-lab Assignment UI	17
3.3.4 Lab UI	19
3.3.5 Post-lab Assignment UI	21
4. Functional Requirements Specification	22
4.1 Stakeholders	22
4.2 Actors and Goals	22
4.3 Use Cases	22
4.3.1 Casual Description	22
4.3.2 Use Case Diagram	24
4.3.2.1 Student	25
4.3.2.2 Instructor	25
4.3.2.3 Camera	26
4.3.3 Traceability Matrix	26
4.3.4 Fully Dressed Descriptions	29
4.4 System sequence diagram	34
4.4.1 UC-1 manageEquipment	34
4.4.2 UC-2 finishAndCheck	35
4.4.3 UC-5 grades	35
4.4.4 UC-14 giveQuiz	36

4.4.5 UC-15, UC-16, and UC-17 labOne, labTwo, and labThree	37
6. Domain Analysis	40
6.1 Domain Model	40
6.1.1 Concept Definitions	41
6.1.2 Association Definitions	41
6.1.3 Attribute Definitions	43
6.1.4 Traceability Matrix	45
6.2 System Operation Contracts	46
7. Interaction Diagrams	49
7.1 UC-1: manageEquipment	49
7.2 UC-2: finishAndCheck	50
7.3 UC-5: grades	51
7.4 UC-14: giveQuiz	52
7.5 UC-15, UC-16, UC-17: labOne, labTwo, labThree	53
8. Class Diagram and Interface Specification	54
8.1 Class Diagram	54
8.1.1 General Class Diagram	55
8.1.2 Equipment Manager Inner Structure	Error! Bookmark not defined.
8.2 Data Types and Operation Signatures	58
8.3 Traceability Matrix	62
9. System Architecture and System Design	64
9.1 Architectural Styles	64
9.2 Identifying Subsystems	66
9.2.1 UML Package Diagram	66
9.3 Mapping Subsystems to Hardware	66
9.4 Persistent Data Storage	67
9.5 Network Protocol	67
9.6 Global Control Flow	67
9.6.1 Execution Orderness	67
9.6.2 Time Dependency	68
9.6.3 Concurrency	68
9.7 Hardware Requirements	68
10. Algorithms and Data Structures	69
10.1 Algorithms	69
10.2 Data Structures	73
11. User Interface Design and Implementation	75
11.1 Login Page	75
11.2 Admin Subsystem	75

11.3 Changes from previous UI design	75
12. Design of Tests	78
12.1 Test Cases	78
12.2 Test Coverage of Test Cases	80
12.3 Integration Testing Strategy	80
13. History of Work	80
14. References	82

Summary of Changes

1. Functional requirements were added when we came across potential problems/needs. Specifically, we added a trash can for discarding pieces, and a sandbox mode for practice with the circuit elements (REQ38 and REQ39).
2. Requirement were renumbered, because previously the numbering was inaccurate, and we needed to account for the new requirements. The traceability matrix was also changed accordingly.
3. The chat feature was not able to be completed
4. Adjusted use case descriptions to account for new functional requirements
5. Functionality requirements were changed based on convenience and ease of use. For example, we initially intended to have a click-and-drag logic probe, but eventually determined that it would be easier to display the values of each node directly on the protoboard itself, represented by colors green and red.
6. Grading methods were changed. Initially, we were going to allow students only 1 submission, and grade based on that, also taking points off for taking a long time. Now, the timer has no impact on the grade, and students get as many attempts as it takes to get it correct, with points being deducted for each incorrect attempt.
7. Added content to section 10, Algorithms and Data Structures in order to show more effectively the complexity of the code that was used in this project.
8. Added a description of Unity, the development tool used for this project, to section 10, so that people better understand the development process.
9. Added detailed descriptions to each piece that is usable within the labs, as well as descriptions of how they were built into the program.
10. UI design evolved immensely (viewable in section 11.3 of this report), as we were able to make it easier and more convenient to use.
11. Added identifying numbers to all images and tables in the report
12. Added descriptions to most of the tables and images used in the report.
13. Updated user interface diagrams, now showing the actual lab interfaces instead of just designs.
14. Converted plan of work into history of work, reviewing how accurately we stuck to deadlines and completed what we intended to complete.
15. Alphabetized glossary of terms for a more organized presentation

Summary of Changes Part 2: Feedback-based changes

After demo 1, we received a good amount of feedback about how we could possibly improve our project and add to its functionality. We took this feedback very seriously, and made several adjustments to our original plan, and included them in our system.

They are as follows:

- Completed grading feature. Following the first demo, there seemed to be some confusion as to how exactly grading would be done. For demo 2, the grading functionality was completed, and operates in this fashion:
 - Grading is done by the system and is done instantly. There is no need for a professor to grade any of the labs him/herself.
 - Grading is based on 100 total points. For every incorrect attempt that the user makes to submit the prelab or postlab, 1 point is deducted. For every incorrect attempt to submit the actual lab, 5 points are deducted. The user will not be allowed to advance until they complete the given section.
 - As an example, if a student submits an incorrect solution to the prelab once before succeeding, an incorrect solution to the lab twice before succeeding, and an incorrect solution to the prelab three times before succeeding, his final grade would be
$$100 - 1*1 - 2*5 - 3*1 = 86$$
- Grades can be taken by the professor and output into the format (CSV) used by numerous grade-saving websites, namely sakai.
- Made protoboard expandable in order to give the user more freedom when building circuits. Now, clicking the “+” sign on the left-hand side of the board will add another section to it, while clicking the “-” sign will remove a section, provided that there is more than 1 section already there.
- Added leaderboard functionality. This allows students to see the top 10 scores for each lab so they can compare their scores with the scores of their peers. We did, however, take into account students’ right to privacy, as these leaderboards display numerical grades only, omitting any identifying factor for students, such as name, ID number, username, etc.
- Although originally planned, we were not able to implement the save-state functionality due to its incredibly complex nature, and the time constraints for this project. The system has a lot of data associated each state of the program. Data involved includes the positions of every single device, and logic node/logic pin associated with each device, and the *states* of each of those pins. Our attempt to recreate the previously saved state involved saving the positions of all of the active devices on the screen (and their nodes/pins). However, due to the fact that we are using an Event Driven framework, simply changing the positions to their previous positions on initialization was not good enough as our Digital Logic Design algorithms would not be executed. Although we did not have enough time to implement this, a possible point of attack to this problem would have been to load the device GameObjects in a random location on the screen on a separate

thread, put the thread to sleep for a few frames/a few milliseconds, and then move them to their saved locations. This would allow our framework to detect changes to the positions and produce callbacks to our algorithms to process the change in positions.

Clarification on the Issue of Coding

Feedback from demo 1 caused us to realize the we did not make clear how much of this project was coding-based, and how much of it was done using a desktop framework. This section aims to display how significant coding was to the development of our project.

Firstly, the use of the Unity Engine framework (Version 2017.3.1f1) was essential to this project, as it served as the basis upon which we built. Unity's main job is to render the 2D or 3D models on to the screen (for this case, its 2D) and facilitate a platform to detect interactions between different "GameObjects". Everything that can be interacted with in our interfaces is considered a "Game Object," and these game objects can be manipulated, to an extent, by the user, with Unity working to detect collisions among these objects. This does not mean, however, that the project was developed using simple click-and-drag setups. In fact, every game object has a script associated with it that tells it how to behave and interact with other game objects.

The game objects' reliance on scripts (in this case, written in C#) is what really makes the coding the focal point of our project. While Unity provided blank game objects for us to manipulate, we had to use the code to write scripts that define the behavior of every single element that is used. Each individual chip, along with the wires, LEDs, switches, UI buttons, magnifying glass, trash can, and every single node on the protoboard had a script associated with them that told them how to react in certain situations. For example, when designing OR gate game objects, we had to write scripts that specifically told the game object how to function in every situation. Essentially, we programmed that game object to function exactly how an OR gate would be expected to function. Overall, this took thousands of lines of code to accomplish, and could not possibly have been done just by using a desktop framework.

Further information of specific algorithms we used between our "GameObjects" can be found in the **Systems Structure and Algorithms sections (Section 9 and 10)** of our report.

1. Customer Statement of Requirements

1.1 Problem Statement

1.1.1. The Problem:

In a world where technology is so prevalent and easily accessible, many students are using it to supplement learning from lectures. These students would greatly benefit from the ability to observe the workings and functions of a laboratory settings on their own time - however, it is difficult to provide labs for large classes of students due to the sheer number of people and logistical or financial restrictions of the schools and universities. Both students and instructors would greatly benefit from a resource that would allow for a laboratory experience that would enhance what they learned in class, but would also allow for instructors to track the progress of their student easily. Specifically, our focus will be on Digital Logic Design Labs. These labs are very technical and require a lot of different parts (power source, protoboard, gates, wires, LEDs, etc.). There are, however, labs that one can use to gain a good comprehension of the material, but do not necessarily require physical interactions with the pieces to gain a full understanding of the working of the circuits. Such labs would be perfect labs to be completed in an online environment.

Even in scenarios where students do get experience in an actual lab, professors must manually grade and track each student's results and lab reports, which can be extremely tedious and time consuming. In addition to keeping data for the students, the school would need to hire lab instructors to manually help and serve hundreds of different students, which takes a great deal of time and resources. With hundreds of students in lab classes, returning grades to students in a timely manner is becoming harder and harder. The process for grading is extremely time consuming, with a procedure that includes collecting hard copies of reports, observing students' work on their labs, physically grading each report, and lastly uploading the grades. This procedure is also an inconvenience for the students as well because it is hard for them to know how they are doing in the class at critical times. One example of this is when the deadline to drop a course is approaching, and students would like to know how they are doing in the class, so they can make an important decision. Lastly, it is very difficult to grade all of the hard copies the students submit, especially when students fail to follow the specified format or make small mistakes that cause discrepancies in the entire lab.

1.1.2 Proposed Solution:

The way that we chose to approach this problem is to have virtual sessions where students can simulate labs on their own computers. These sessions can accurately depict the procedures involved in the labs and allow the students to become more familiar with the material. The main goal of this treatment is to facilitate student learning, with an added bonus being that we are also able to aid the instructors in

keeping track of their students' progress. The former is of higher priority, because we feel that if students are able to learn at their own pace and on their own time, it will assist with the overall improvement of the course. Some of the things the software system is expected to include are listed below.

- Labs accessible 24/7:
 - The student will be tasked with completing two labs that are typically given in a real digital logic design laboratory setting. The labs will allow them to apply their digital logic knowledge in various situations, such as constructing a full adder, or testing the output of certain circuit configurations. The student will be able to select any devices needed for the lab that is available through the laboratory library and mix and match them anyway they like just like they would be able to in the lab. The laboratory will be the implementation of the first three labs in this manual: http://www.ece.rutgers.edu/~marsic/Teaching/DLD/lab-man_2012.pdf
- Instant Feedback:
 - The most important aspect of any form of learning is feedback of whether the student was successful or unsuccessful in any given task. The virtual lab should check for any wrongdoings upon user request for completion and give the user instant feedback, so he/she may understand what was done incorrectly, and correct it in future use. Visual indicators will also be given throughout the lab in order to show the interaction between the circuit devices, so the student is able to see exactly how the circuit is functioning as it is being built.
- Reinforced Learning:
 - Before and after each lab, the student will be required to take a quiz, which covers the topics that a typical pre-lab report would cover. The purpose of these pre-quizzes is to ensure the student will be able to perform the lab adequately and that he or she has the proper prerequisite knowledge to perform it. Since the purpose of the pre-quiz is to aid preparation, it will be checked for completeness and correctness before the student is allowed to advance. After the lab is performed, a post quiz will be given to ask general questions about the lab to attempt to see if the student really learned anything valuable from the lab. The quizzes in general will cover topics such as truth tables, karnaugh maps, and logic equations, in addition to general questions.
- Instant Grading:

- After configuring a circuit, based on the lab the student is performing, the operation of the circuit will be checked for correctness. The student will lose points for every attempt that is made to finish a lab with the incorrect configuration. The pre-lab and post-lab quizzes are also graded and added to the final grade for that lab. Lab grades will be available to both students and instructors immediately following the completion of a lab
- Cheating Verification:
 - Unfortunately, an unintended consequence of having a virtualized lab session is the increased likelihood of some students taking advantage of the opportunity to cheat by allowing others to do these labs for them. A solution to this issue is to implement facial recognition software. Each student should be asked to take a picture that will be analyzed by a facial recognition software. The camera of the student's laptop/computer will stay on while the application is running, which will ensure that the original student is the one completing the lab experiment.

2. Glossary of Terms

Administrator	An “Actor” of this project that takes the role of viewing grades, creating and deleting users. Also referred to as “Teacher”, “Professor”, and “Instructor”.
AND gate	A boolean operator that outputs the value one if and only if all the operands are equal to one, and otherwise has a value of zero
Boolean	A binary variable, having two possible values, “true” and “false”.
Combinational system	A logical system with no memory, for which the output depends only on the current input values
Full Adder	A circuit configuration which adds three one-bit binary numbers (C, A, B) and outputs two one-bit binary numbers, a sum (S) and a carry (C1). The full-adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. bit binary numbers.
Grey Code	An ordering of the binary numeral system such that two successive values differ in only one bit (binary digit).
Karnaugh Map	A diagram consisting of a rectangular array of squares, each representing a different combination of the variables of a Boolean function
LED (light-emitting diode)	A semiconductor device that emits visible light when an electric current passes through it.
Logic Diagram	Diagrams in the field of logic, used for representation and to carry out certain types of reasoning.
Logical product	A logical term that is the Boolean AND of two or more variables
Logical sum	A logical term that is the Boolean OR of two or more variables
Minterm	A Boolean expression resulting in 1 for the output of a single cell, and 0s for all other cells in a Karnaugh map, or truth table. If a minterm has a single 1 and the remaining cells as 0s, it would appear to cover a minimum area of 1s.
NAND gate	A Boolean operator that gives the value zero if and only if all the operands have a value of one, and otherwise has a value of one (equivalent to NOT AND).
OR gate	A Boolean operator that gives the value one if at least one operand (or input) has a value of one, and otherwise has a value of zero.
Protoboard	A board used for making experimental models of an electric circuit (fig 2.1)

Student	An “Actor” of this project that takes the role of making use of the laboratory simulations that is graded.
Switch	A device for making and breaking the connection in an electric circuit.
Truth table	A tabular list of all possible input combinations for a combinational system and the corresponding outputs.
XOR gate	A digital logic gate that gives a logic one output when the number of true inputs is odd, and otherwise has a value of zero. (Fig 2.2)

Table 2: Glossary of Terms

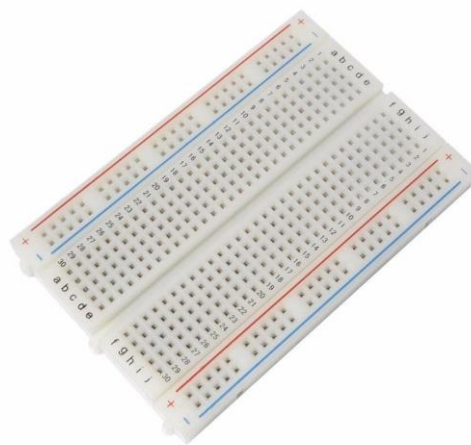


Figure 2.1: A Protoboard used for creating experimental models of electric circuits

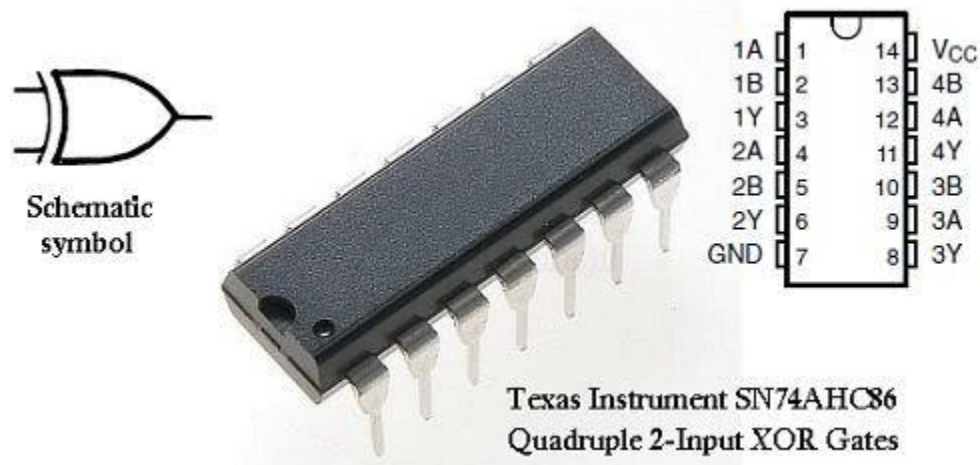


Figure 2.2: A circuit representation of an XOR gate, along with an image of the chip, and the breakdown of the functioning of the chip's pins

3. System Requirements

3.1 Enumerated Functional Requirements

Below is a comprehensive list of every requirement for the system that we have designed. It is separated into sections based on where the requirement is needed within the scope of the system.

The table below contains three sections. The first is a numbered list, giving a unique identifier to each requirement. The second is a number between 1 and 5 representing the importance, or priority of each requirement, with 5 being high priority, and 1 representing low priority. The final column is a general description of each requirement.

3.1.1 General Requirements

Number	PW	Requirement
REQ1	5	The user shall be able to place any circuit material onto a protoboard, and that element should accurately portrays the correct logic.
REQ2	2	The user shall be able to “probe” around any logic circuit to view whether certain areas are high and low (represented by green and red nodes).
REQ3	3	The user shall be able to use an power source that provides a constant 5 volts, and a ground that the user can connect to the circuit.
REQ4	5	The user should be able to place logic switches into the circuit board that can be turned on to let current through, and off to cut off current.
REQ5	4	The system should be able to auto-grade the performance of the student for both the viewing of the student and the professor.
REQ6	2	The system should let the student use LEDs that turn on and off based on the input given to it.
REQ7	2	The system should time the student, and make the finishing time viewable to the student and professor.
REQ8	5	The system should give a list of all the materials available, and allow the student to select which ones to use.
REQ9	5	The system should allow flexible wiring to be connected to the protoboard and other logic devices.
REQ10	2	The system shall allow the student to use a “magnifying glass” that allows the student to see the inner workings of any logic chips.
REQ11	1	The system should be able to display to the student their achievements for each lab they complete (grade, time to complete, etc.).

REQ12	2	The system shall take a picture of the student in a well-lit condition to serve as a “base picture”, and then a picture should be taken every minute thereafter and match the pictures by facial recognition algorithms to detect potential cheating.
REQ13	3	The system shall assign students grades based on their performance in the labs which is decided by pre-lab and post-lab performance, as well as the number of attempts it takes to submit a correct circuit during the lab
REQ14	1	The system shall connect the students to each other and to the professor in a chat-like service inside the application.
REQ15	1	The system shall produce grades in a common markup language for the professor to use and upload to his/her grading system.
REQ16	4	The system shall give pre and post quiz that is specific to each lab to ensure the student is properly prepared for the lab.

Table 3.1.1: General Requirements

3.1.2 Lab 1 Requirements

Number	PW	Requirement
REQ17	4	The system will let the user implement a simple logic function to become familiar with future labs.
REQ18	5	The system will allow the user to select circuit elements that can be used to implement the given logic function.
REQ19	5	The system will present the user with a logic function and a truth table to fill out. This will serve as the pre-lab assignment.
REQ20	3	The system will have a “Check” button that allows the users to check their answers upon completion of the lab.
REQ21	5	The system will display a protoboard, power supply, and a drop-down list of circuit elements, switches, and LEDs in order to implement the boolean function.
REQ22	1	The last screen of the pre-lab, post lab, and lab will display a congratulations message when submissions are correct and advance the user to the next interface after a brief period of time

Table 3.1.2: Lab 1 Requirements

3.1.3 Lab 2 Requirements

Number	PW	Requirement
--------	----	-------------

REQ23	3	The student should be able to mark Karnaugh maps with the logic function as a given. This shall serve as the pre-lab assignment.
REQ24	2	The student should be able to get the minimal sum-of-products expression and draw the logic diagram with a 2 input NAND, 3 input NAND gate, 2 input OR gate and a hex inverter.
REQ25	3	The student should be able to get the outcome waveform with the logic diagram drawn.
REQ26	2	The student should be able to modify the sum-of-products expression to eliminate timing hazard and make a new logic diagram.
REQ27	4	The system should allow the student to be able to construct a given logic circuit, and use it to construct a logic table of the output given each set of inputs
REQ28	5	The system should allow the student to use the given components to construct a full adder and verify that it works correctly for all input combinations.

Table 3.1.3: Lab 2 Requirements

3.1.4 Lab 3 Requirements

Number	PW	Requirement
REQ29	5	The system should be able to let the student select from a list of available materials, however, for this lab, three multiplexers and an exclusive OR gate will be used.
REQ30	5	The system should check if the student has put the lab materials in the right configuration in the protoboard for the decoder configuration, and encoder configuration for Binary to Gray Code.
REQ31	5	The user will be asked to enter the correct response to the equation needed to solve the Binary to Gray code conversion.

Table 3.1.4: Lab 3 Requirements

3.2 Enumerated Nonfunctional Requirements

Number	PW	Requirement
REQ32	2	The system should give feedback to every right or wrong action.
REQ33	3	The system shall have login functionality for both instructors and students.

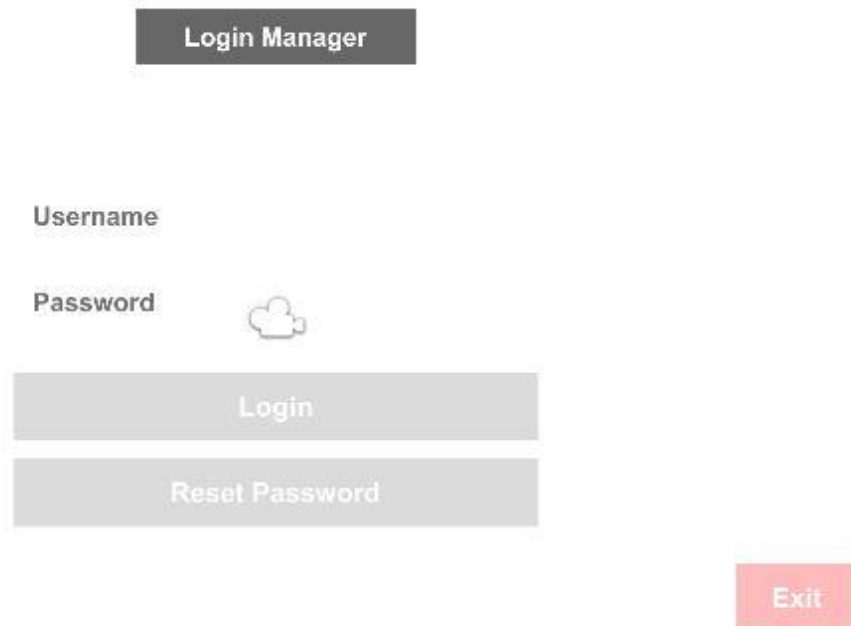
REQ34	3	The system will allow admins to create new users and put them into the system.
REQ35	1	The system will allow users to recover passwords.
REQ36	5	The system shall have an interface that allows the users to select which labs to perform.
REQ37	2	The system shall save the state of each lab so that the user is able to return and finish the lab at another time.
REQ38	2	The system shall contain a “Sandbox Mode,” which allows students to test different circuits and become familiar with the circuit elements without having to complete an actual lab.
REQ39	2	The system shall have a “trash can” icon that users can drag unwanted pieces to in order to discard them

Table 3.2: Enumerated Nonfunctional Requirements

3.3 On-Screen Appearance Requirements

Each subsection below contains diagrams of each specific interface that is included in this project. Each has a small description below it to distinguish what it is and when it will be presented to the user.

3.3.1 Login UI



The diagram illustrates the Login Manager interface. At the top, there is a dark gray header bar with the text "Login Manager" in white. Below the header, the interface is divided into two main sections. The left section contains two input fields: "Username" and "Password". The "Password" field has a small eye icon to its right, indicating a toggle for password visibility. Below these fields are two large, light gray buttons: "Login" and "Reset Password". The right section of the interface is a vertical sidebar containing a single red button labeled "Exit".

Figure 3.3.1: Login UI

Figure 3.3.1 is our login UI, and it is the first screen that the user is presented with when they start the application. They will enter their username and password in the text boxes, then click login, following which the system will determine if they are in the database. If they are, it will direct them to the next UI. If not, they will not be let in. There is also a Reset Password button in the case of a forgotten password.

3.3.2 Student UI

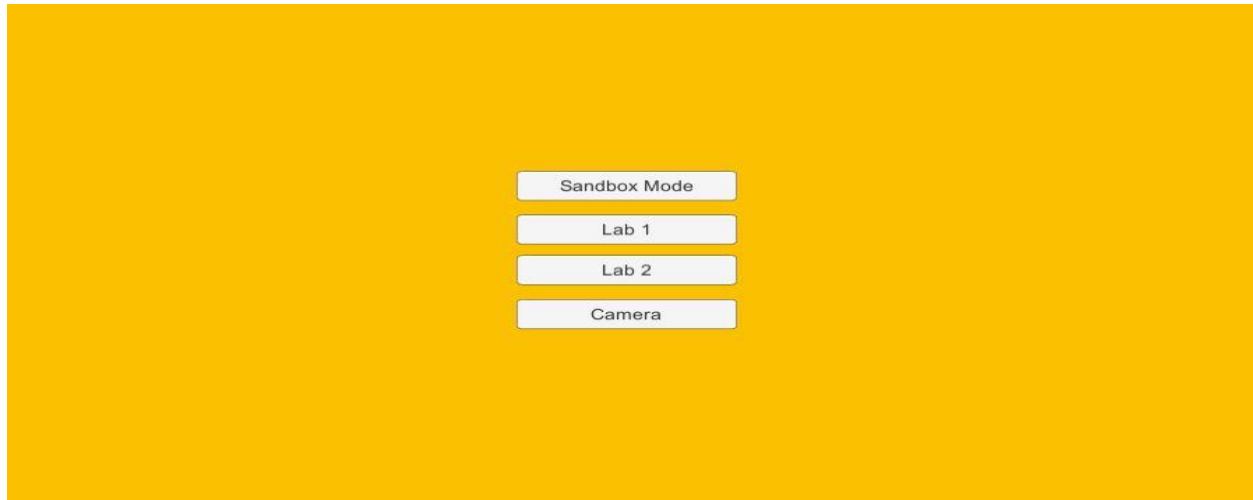


Figure 3.3.2: Student UI

Figure 3.3.2 is the UI that the system displays after it detects that a student has logged in. It allows the student to select one of four different modes. Three of them are labs, and the final one brings the user to sandbox mode, where they can experiment with different circuit designs.

3.3.3 Pre-lab Assignment UI

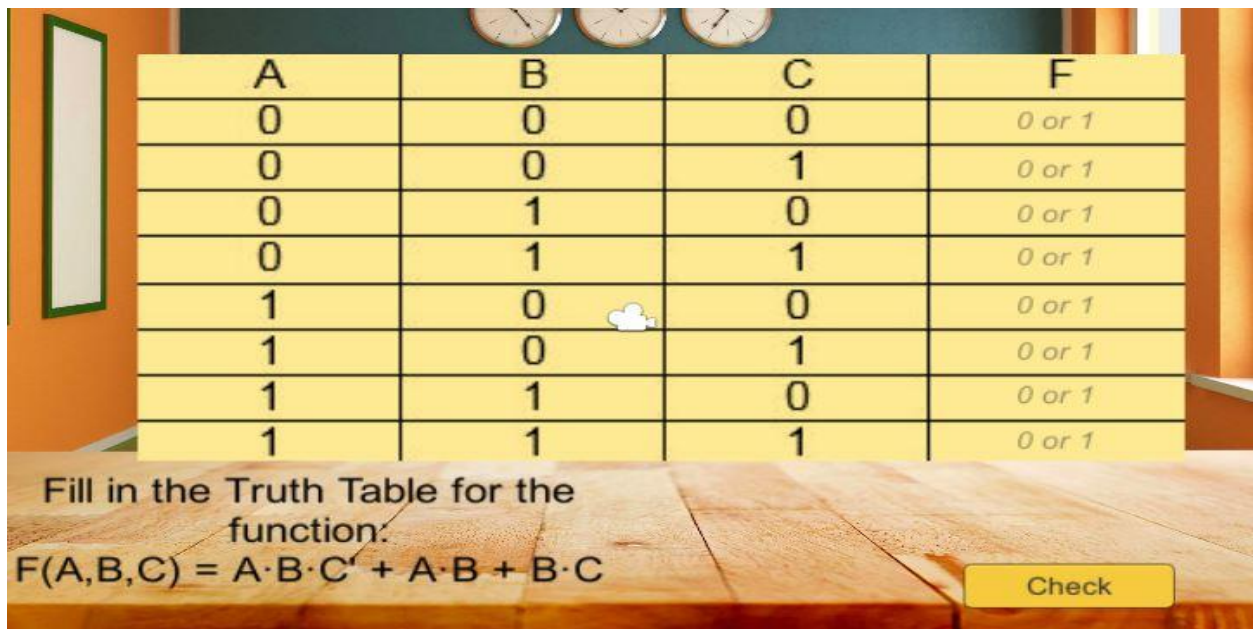


Figure 3.3.3.1: Prelab 1 UI

A	B	Cin	S	Co
0	0	0	0 or 1	0 or 1
0	0	1	0 or 1	0 or 1
0	1	0	0 or 1	0 or 1
0	1	1	0 or 1	0 or 1
1	0	0	0 or 1	0 or 1
1	0	1	0 or 1	0 or 1
1	1	0	0 or 1	0 or 1
1	1	1	0 or 1	0 or 1

Fill in the Truth Table for the Full Adder.

Check

Figure 3.3.3.2: Prelab 2 UI

Figures 3.3.3.1 and 3.3.3.2 above are the prelab UIs for labs 1 and 2 respectively. Both ask the user to fill in a truth table based on a given logic function. The boxes are clickable, and doing so will allow the user to fill in either a 0 or a 1 by typing it into their keyboard. After finishing, the user clicks a check button to submit the answers for grading.

3.3.4 Lab UI

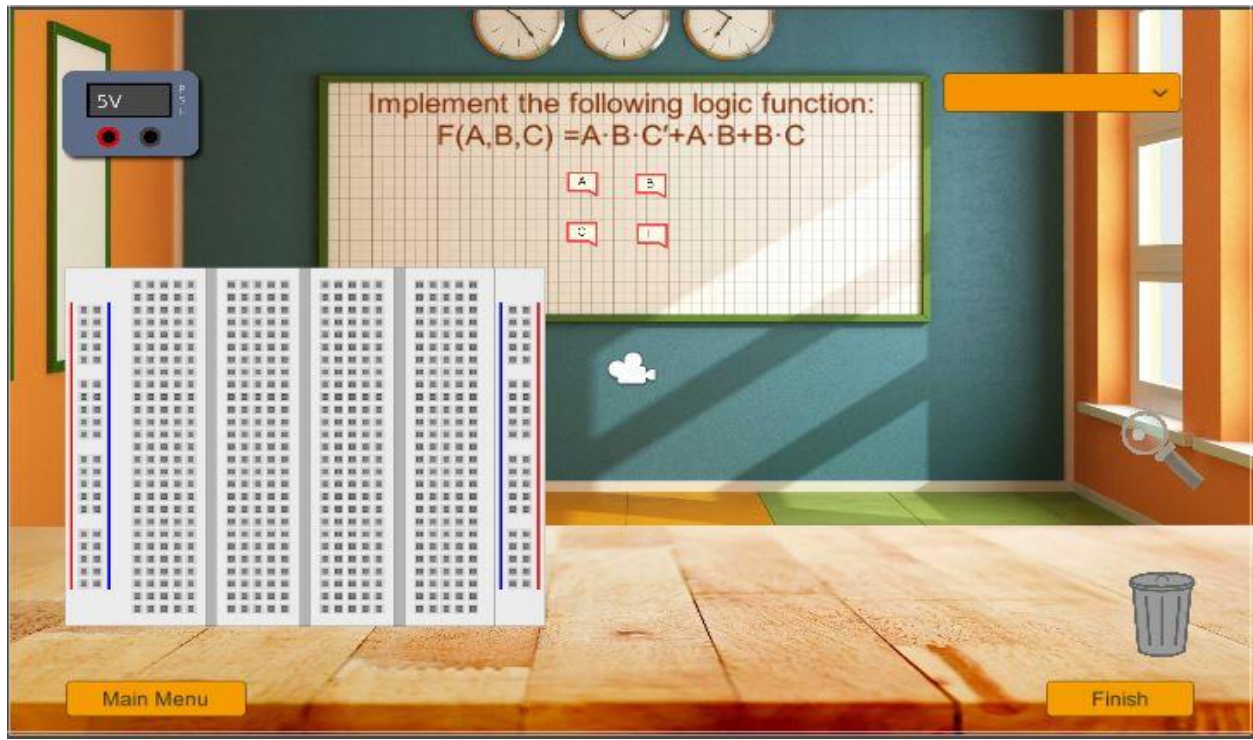


Figure 3.3.4.1: Lab 1 UI

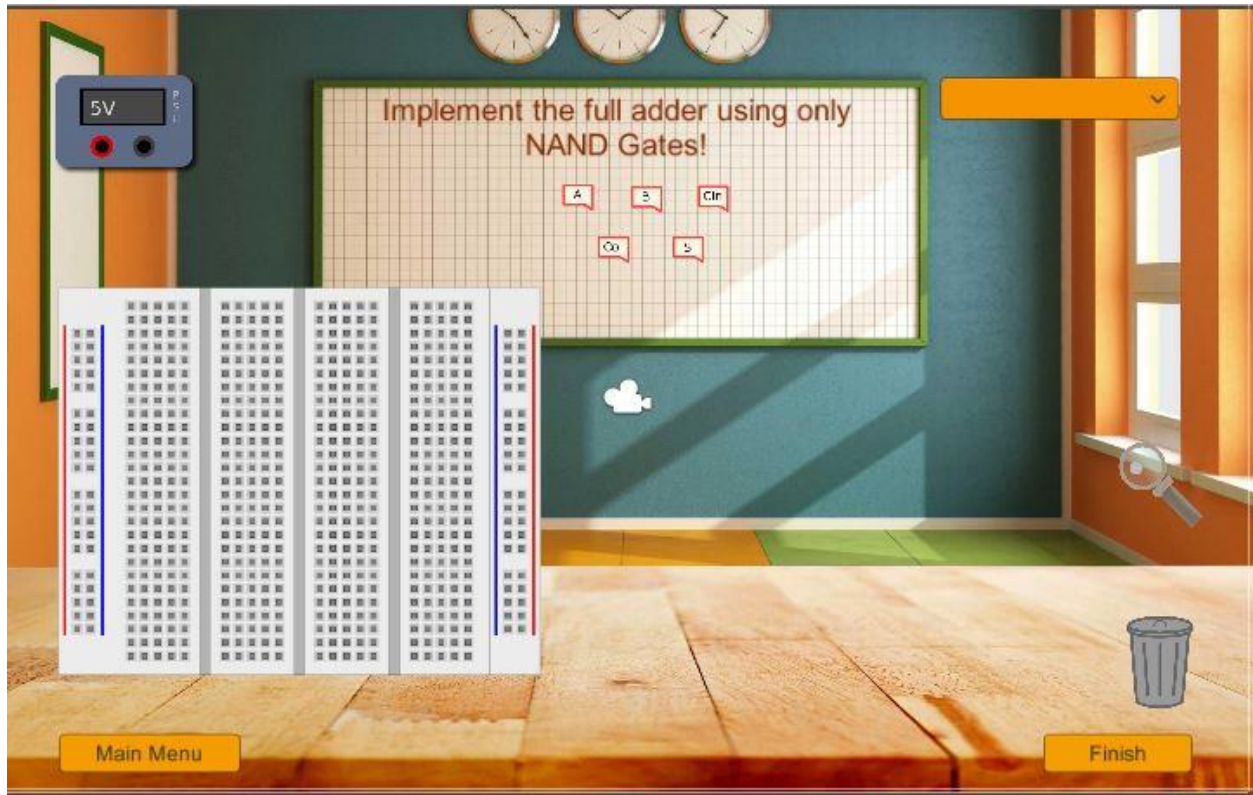


Figure 3.3.4.2: Lab 2 UI

The lab interfaces displayed above in figure 3.3.4.1 and 3.3.4.2 are very similar. Both contain a protoboard, on which one can connect chips, wires, switches, and LEDs, as they will “lock” into place if lined up with nodes. In the top left corner is a power source, which has two separate connections for wires: a red one for +5 volts, and a black one for ground. The top right corner contains the drop down menu, from where users may obtain chips, wires, switches, and LEDs. The magnifying glass may be dragged over chips to obtain a closer look at the inner structure. Unwanted pieces can be dragged over to the trash can and released to discard them. The small tags above and to the right of the protoboard can be attached to circuit elements, and are the main basis for grading. For example, in lab 1, the A, B, and C tags should be attached to the switches that they are represented by, and the F tag should be attached to an LED representing the output. These tags will also “lock” into place when put in the correct position. Finally, there exist a “Main Menu” button to bring the user back to the main menu, and a “Finish” button to submit the assignment.

3.3.5 Post-lab Assignment UI

AB

	00	01	11	10
00	0 or 1	0 or 1	0 or 1	0 or 1
01	0 or 1	0 or 1	0 or 1	0 or 1
11	0 or 1	0 or 1	0 or 1	0 or 1
10	0 or 1	0 or 1	0 or 1	0 or 1

CD

Fill in the K-map for the following logical function:
 $F(A,B,C) = A' \cdot C' \cdot D + B' \cdot C \cdot D + A \cdot C' \cdot D + B \cdot C \cdot D$

Check

Figure 3.3.5.1: Postlab 1 UI

Figure 3.3.5.1 is the postlab assignment UI for lab 1. This particular assignment is a Karnaugh Map, and the user is asked to fill in the correct values for each box. The boxes are clickable, and doing so will allow the user to type in either a 0 or a 1 from the keyboard. The check button at the bottom is used to submit answers when finished.

4. Functional Requirements Specification

4.1 Stakeholders

- Student
 - The student will need to be able to complete assigned lab reports and must be able to communicate any questions they have to their teacher. They should also be responsible for making sure the cheat detector (Camera) detects them taking the laboratory.
- Instructor (Professor, Teacher)
 - The teacher will need to be able to enhance the learning of the students by viewing statistics and grades to see what topics should be focused on. They will be able to communicate with the students, check the grades for each students that have completed the labs, and be able to manage any new or existing users.

4.2 Actors and Goals

- Student (Initiating):
 - to perform digital logic experiments and get results at the end of each labs using logic equipment such as encoders, decoders, multiplexers, etc.
- Teacher (Initiating):
 - to present assignments (labs) and obtain grades for all the students in the class. Also they should be able to have live interactions with the students.
- Database (Participating):
 - SQL relational database that keeps all the data, including usernames, passwords, chat, and grades.
- Camera (Participating):
 - Facilitates the cheat detector to insure that students are completing their own assignments.

4.3 Use Cases

4.3.1 Casual Description

Table 4.3.1 below presents a complete list of our use cases, created to resolve each of our functionality requirements. It consists of 4 columns. The first serves to list the unique numerical identifier for each use case. The second provides the common name that we will use to refer to the use case. The third is a short description of what the functionality of the use case will be. The last is a list of the requirements that the use case satisfies.

Use Case	Name	Description	Requirement
----------	------	-------------	-------------

UC-1	manageEquipment	Allows the user to get equipment from a drop down menu and place them in the lab, connect them logically, or delete them.	REQ1, REQ2, REQ3, REQ4, REQ6, REQ8 REQ9, REQ17, REQ18, REQ19, REQ20, REQ21, REQ24, REQ29, REQ39
UC-2	finishAndCheck	Checks the output for all logical input cases and compares with the correct answers	REQ5, REQ20, REQ28, REQ30
UC-3	login	Logs the user or instructor into the system to their respective views.	REQ33
UC-4	manageUsers	Allows the admin to add a user with a set password, and delete users.	REQ34, REQ35
UC-5	grades	Grading subsystem for instructor to view or delete grades. The instructor can also choose the download the grades in a markup language.	REQ5, REQ13, REQ15
UC-6	selectLab	Allows students to select a specific lab and open it up to the screen. Allows students to select lab 1, lab 2, and lab 3.	REQ36, REQ38
UC-7	magnifyEquipment	Allows student to obtain additional information on different objects	REQ10
UC-8	timer	Keeps track of how much time a user takes to complete a given lab. Times are saved so students and instructors can see the results.	REQ7
UC-9	saveState	Saves state of the lab if the student shuts the program down or changes lab.	REQ37
UC-10	switchToMainMenu	Allows the user to stop their current activity and go back to the main menu.	REQ22
UC-11	openChat	Allows anyone to open and speak in the program chat or contact instructor	REQ14

UC-12	giveFeedback	Gives students visual feedback on their circuit's performance, showing a green check mark if it obtains the correct output for a particular combination of inputs, or a red "X" if it obtains the incorrect output for a particular combination of inputs	REQ5, REQ11, REQ13, REQ32
UC-13	cheatDetector	Allows instructors to track students and identify cheating with the camera.	REQ12
UC-14	giveQuiz	Gives the student pre and post quizzes. May ask to fill out karnaugh maps and truth tables.	REQ16, REQ19, REQ23, REQ24, REQ25, REQ26, REQ27, REQ31
UC-15	labOne	Perform laboratory one which also acts as a tutorial to the students on how to perform the labs.	REQ17, REQ18, REQ19, REQ20, REQ21, REQ22
UC-16	labTwo	Perform laboratory two which lets student implement a 'full adder'	REQ23, REQ24, REQ25, REQ26, REQ27, REQ28
UC-17	labThree	Perform laboratory three which lets student implement a 'binary to gray' converter	REQ29, REQ30, REQ31

Table 4.3.1: Casual Description of Use Cases

4.3.2 Use Case Diagram

The three diagrams below are the use case diagrams as they apply to the student, instructor, and camera, respectively. From these diagrams, one can see how the actors interact directly and indirectly with each of the use cases, as well as how the use cases interact with each other within the scope of the program.

4.3.2.1 Student

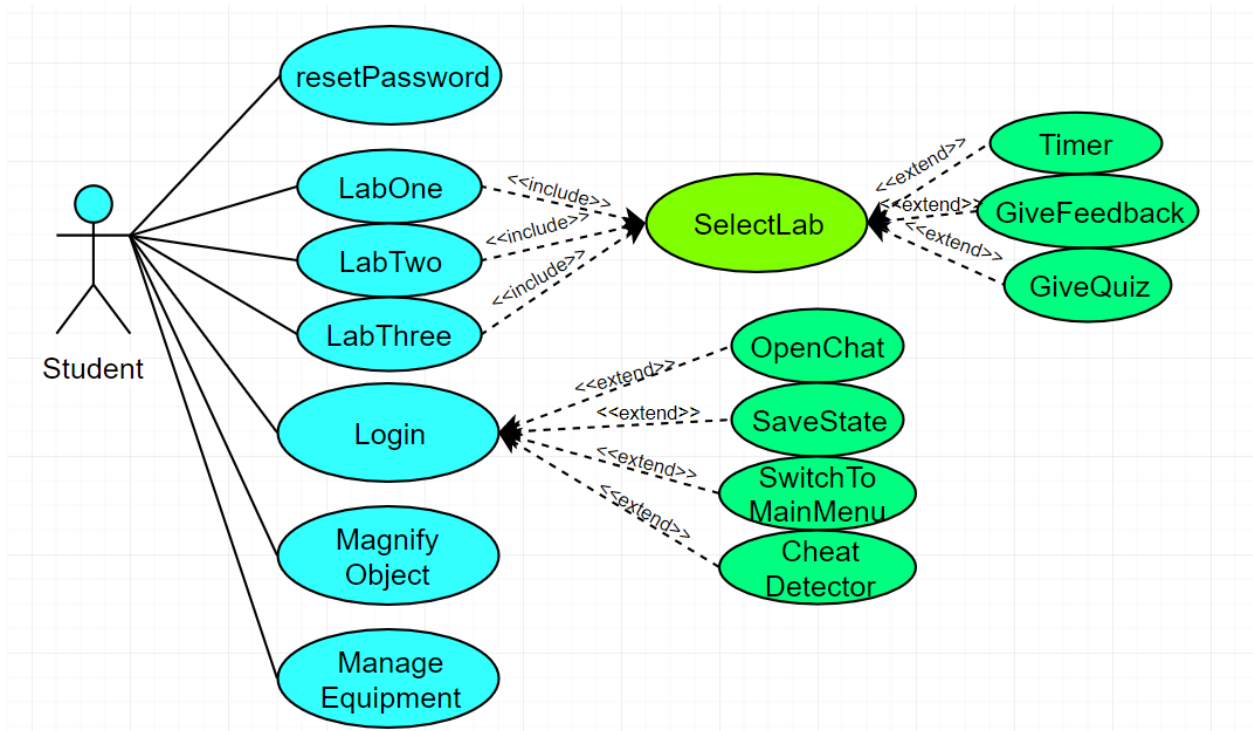


Figure 4.3.2.1: Student Use Case Diagram

4.3.2.2 Instructor

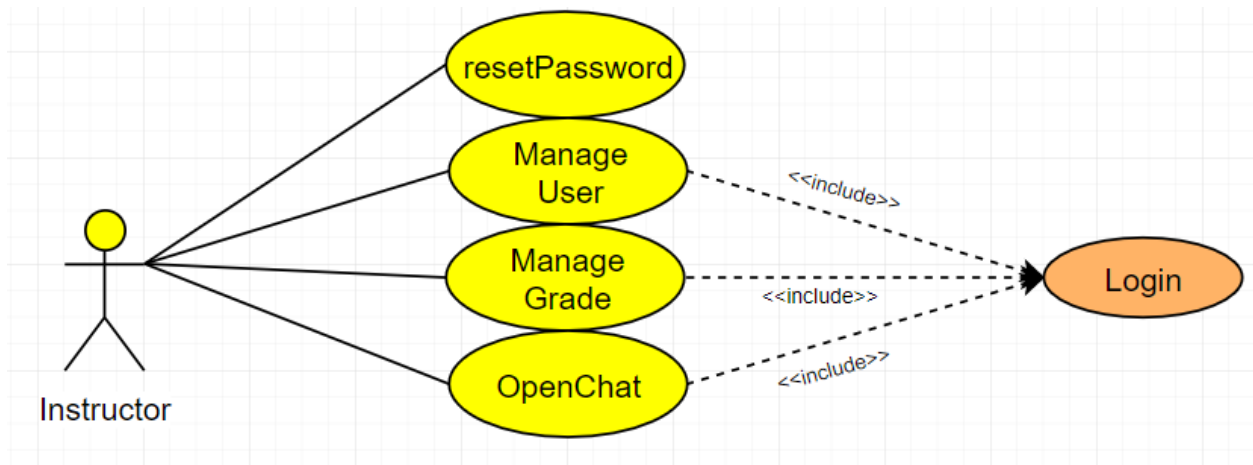


Figure 4.3.2.2: Instructor Use Case Diagram

4.3.2.3 Camera

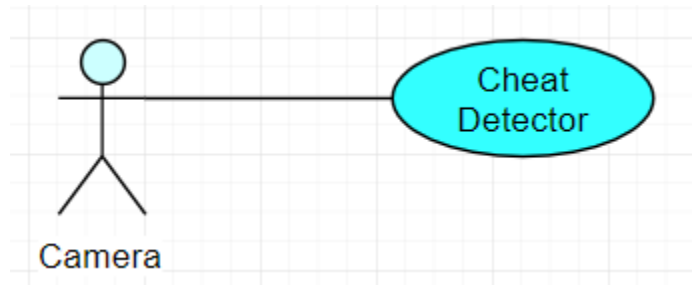


Figure 4.3.2.3: Camera Use Case Diagram

4.3.3 Traceability Matrix

Table 4.3.3 is our traceability matrix, which displays the interaction between our use cases, and our functional requirements. The rightmost column lists the requirements in numerical order, while the topmost row lists the use cases in the numerical order. The second column from the left corresponds to the priority on a scale from 1 to 5 corresponding with each requirement. An “X” in a box on the table means that the requirement in that row is satisfied by the use case in the intersecting column. The second to last row lists the maximum priority of a requirement that is covered by that particular use case. The final row lists the sum of the priority values of the requirements covered by that use case.

	PW	UC-1	UC-2	UC-3	UC-4	UC-5	UC-6	UC-7	UC-8	UC-9	UC-10	UC-11	UC-12	UC-13	UC-14	UC-15	UC-16	UC-17
REQ1	5	X																
REQ2	2	X																
REQ3	3	X																
REQ4	5	X																
REQ5	4		X			X							X					
REQ6	2	X																
REQ7	2								X									
REQ8	5	X																
REQ9	5	X																
REQ10	2							X										
REQ11	1												X					
REQ12	2													X				
REQ13	3					X							X					

REQ14	1											X						
REQ15	1					X												
REQ16	4													X				
REQ17	4	X														X		
REQ18	5	X														X		
REQ19	5	X												X	X			
REQ20	3	X	X													X		
REQ21	5	X														X		
REQ22	1										X					X		
REQ23	3														X		X	
REQ24	2	X													X		X	
REQ25	3														X		X	
REQ26	2														X		X	
REQ27	4														X		X	
REQ28	5		X														X	
REQ29	5	X																X
REQ30	5		X															X
REQ31	5														X			X
REQ32	2												X					
REQ33	3			X														
REQ34	3				X													
REQ35	1				X													
REQ36	5						X											
REQ37	2									X								
REQ38	2						X											
REQ39	2	X																
MAX PW	5	5	5	3	3	4	5	2	2	2	1	1	4	2	5	5	5	5

TOTAL PW	124	58	17	3	4	8	7	2	2	2	1	1	11	2	28	23	19	15
---------------------	------------	-----------	-----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	----------	-----------	-----------	-----------	-----------

Table 4.3.3: Traceability Matrix

4.3.4 Fully Dressed Descriptions

The boxes below contain fully dressed descriptions of certain use cases. The use cases were chosen based on the number of requirements that they satisfied, and the number of priority points that it corresponded to. Each table contains the name and number of the use case, the requirements that it covers, the initiating actor(s), the actor's goals, the preconditions, postconditions, and the flow of events.

Use Case UC-1:	manageEquipment
Requirements:	REQ1, REQ2, REQ3, REQ4, REQ6, REQ8 REQ9, REQ17, REQ18, REQ19, REQ20, REQ21, REQ24, REQ29, REQ39
Initiating Actor:	Student
Actor's Goals:	To open an items menu and select a logic device to place on the protoboard while constructing a circuit
Preconditions:	The student must be actively completing a lab or in sandbox mode, where they will be presented with the option to select/move logic devices.
Postconditions:	Once completed the student will be able to see the item in its correct place on the protoboard, and will be able to put other logic devices around it in order to complete the circuit
Flow of events:	
→The user clicks the drop-down equipment button, and selects an equipment he/she wants. ←The equipment appears next to the protoboard. →The user clicks and drags the equipment to the desired destination. ←The equipment follows the user's mouse and stops at the destination.	

Use Case UC-2:	finishAndCheck
Requirements:	REQ5, REQ20, REQ28, REQ30
Initiating Actor:	Student
Actor's Goals:	To submit their constructed circuit for grading
Preconditions:	The student must have completed the circuit for a particular lab, at which point they can click a button to submit their work for grading
Postconditions:	The student will have submitted his or her work, and the system will compare the student's answers to the correct solution, assigning grades accordingly.

Flow of events:	
→	The user moves clicks and drags pieces to the protoboard and constructs the circuit
←	The system responds, displaying the user's completed circuit
→	The user clicks the finish button
←	The system checks the students' answers for correctness, comparing it to the desired answer

Use Case UC-5:	grades
Requirements:	REQ5, REQ13, REQ15
Initiating Actor:	Student/Instructor
Actor's Goals:	To view grades from completed labs, and in the case of the instructor, to document the grades for each student
Preconditions:	Labs must be completed for a grade to be assigned.
Postconditions:	After grades are assigned, both students and instructors will be able to view them.
Flow of events:	
Student: →The student completes and clicked finish button. ←The system instantly records the corresponding grade of the student. →The student completes the post quiz and presses submit ←The system assigns the student the post-quiz grade. Instructor: →The instructor logs in to the system ←The system presents the instructor with a) Download Grade b) Display All Grades buttons →The instructor clicks a) Download Grade b) Display All Grades → a) A pdf version of students' grade is downloaded b) All students' grades are displayed	

Use Case UC-14:	giveQuiz
Requirements:	REQ16, REQ19, REQ23, REQ24, REQ25, REQ26, REQ27, REQ31
Initiating Actor:	student
Actor's Goals:	The student will be required to take a quiz after the completion of each lab that will test their understanding of the work that was just done.

Preconditions:	The student must have completed the lab, or decided to start a lab
Postconditions:	The student will be asked to complete a quiz, consisting of potentially multiple choice questions, Karnaugh maps, and truth tables. Upon completion, they will be able to submit the quiz for grading
Flow of events:	
→The student enters or completes a lab ←The system responds by directing the user to a pre-lab or post-lab quiz →The user completes the quiz by filling out necessary information →The user submits for grading by pressing submit ←The system compares the users answers to the correct answers and assigns grades accordingly	

Use Case UC-15:	labOne
Requirements:	REQ17, REQ18, REQ19, REQ20, REQ21, REQ22
Initiating Actor:	Student
Actor's Goals:	Perform the lab (Introduction to Hardware) and the quizzes included within it and receive a grade in the end.
Preconditions:	Student must have logged in and not have finished this particular lab prior to starting it.
Postconditions:	Student will have completed all the requirements to finish this lab, including the quiz, and have received a grade for it.
Flow of events:	
→The student clicks on the 'Laboratory One' button. ←The system switches the scene to the introduction to lab one and presents a pre-quiz to prepare the student →The student enters the required answers correctly ← The system starts the lab and gives the user the option to use any devices available in the system that they wish → The student can use the 'magnifying glass' object to inspect particular equipment ← The system shows the student more information about the particular device. → The student configures the devices in the protoboard as they wish, with set outputs designated in a predefined area and clicks finish after he or she is finished. ← The system checks the output with the required output to ensure that the user has completed the lab correctly. The system will require the student to use an Inverter, AND gate, and OR gate with the correct output on the designated LEDs to pass the experiment correctly. The system then presents the student with a post-quiz which is graded. → The student takes the post-quiz, answers the questions, and presses submit.	

← The system checks the answer and assigns the grade accordingly.

Use Case UC-16:	labTwo
Requirements:	REQ23, REQ24, REQ25, REQ26, REQ27, REQ28
Initiating Actor:	Student
Actor's Goals:	Perform the lab (Combinational SSI Circuits) and the quizzes included within it and receive a grade in the end.
Preconditions:	Student must have logged in and not have finished this particular lab prior to starting it.
Postconditions:	Student will have finished all the requirements to finish this lab, including the quiz, and have received a grade for it.
Flow of events:	
<p>→The student clicks on the 'Laboratory Two' button. ←The system switches the scene to the introduction to lab one and presents a pre-quiz to prepare the student →The student enters the required answers correctly ← The system starts the lab and gives the user the option to use any devices available in the system that they wish → The student can use the 'magnifying glass' object to inspect particular equipment ← The system shows the student more information about the particular device. → The student configures the devices in the protoboard as they wish, with set outputs designated in a predefined area and clicks finish after he or she is finished. ← The system checks the output with the required output to ensure that the user has completed the lab correctly. The system will require the student to use an Inverter, NAND gates, and Or gate to design a Full Adder with the correct output on the designated LEDs to pass the experiment correctly. The system then presents the student with a post-quiz which is graded. → The student takes the post-quiz, answers the questions, and presses submit. ← The system checks the answer and assigns the grade accordingly.</p>	

Use Case UC-17:	labThree
Requirements:	REQ29, REQ30, REQ31
Initiating Actor:	Student
Actor's Goals:	Perform the lab (Combinational MSI Circuits) and the quizzes included within it and receive a grade in the end.
Preconditions:	Student must have logged in and not have finished this particular lab prior to starting it.
Postconditions:	Student will have finished all the requirements to finish this lab, including the quiz, and have received a grade for it.
Flow of events:	
<p>→The student clicks on the 'Laboratory Three' button.</p> <p>←The system switches the scene to the introduction to lab one and presents a pre-quiz to prepare the student</p> <p>→The student enters the required answers correctly.</p> <p>← The system starts the lab and gives the user the option to use any devices available in the system that they wish</p> <p>→ The student configures the devices in the protoboard as they wish with set outputs designated in a predefined area and clicks finish after he or she is finished.</p> <p>← The system checks the output with the required output to ensure that the user has completed the lab correctly with XOR gates, and MUXes to implement a grey to binary decoder. The system then presents the student with a post-quiz.</p> <p>→ The student takes the post-quiz and answers the questions.</p> <p>← The system checks the answer and assigns the grade accordingly.</p>	

4.4 System sequence diagram

Figure 4.4.1 through 4.4.5 are the system sequence diagrams for each of the use cases that had a fully-dressed description associated with them. These diagrams show the events brought about by external actors, as well as the internal workings of the system.

4.4.1 UC-1 manageEquipment

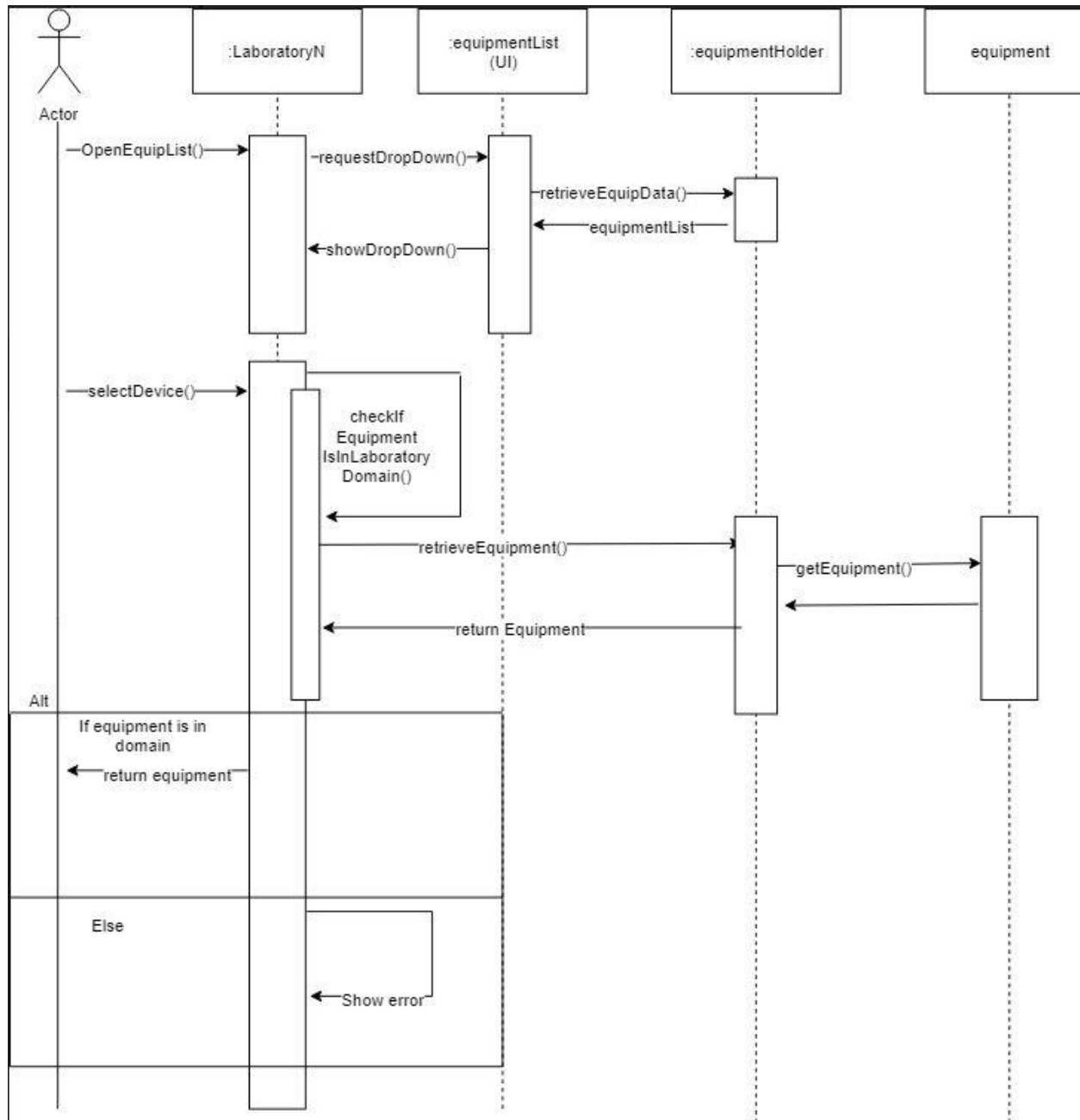


Figure 4.4.1: manageEquipment Sequence Diagram

4.4.2 UC-2 finishAndCheck

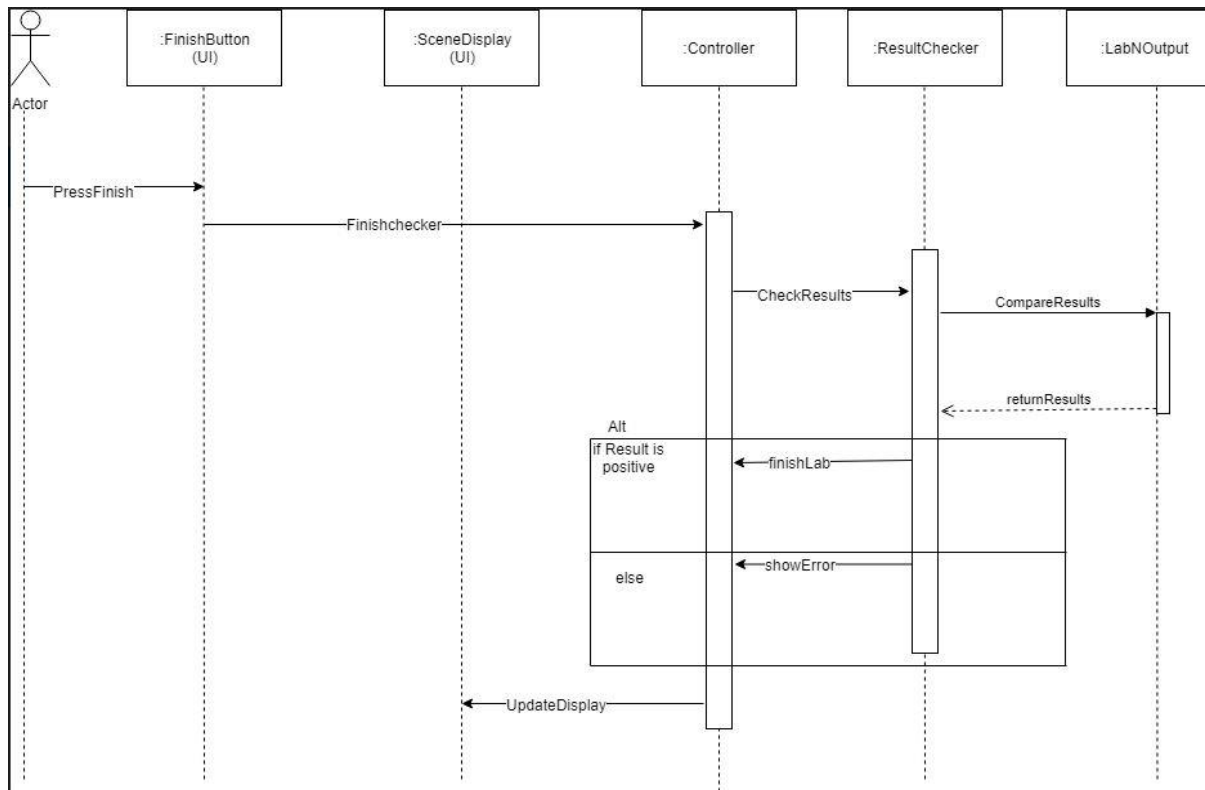


Figure 4.4.2: finishAndCheck System Sequence Diagram

4.4.3 UC-5 grades

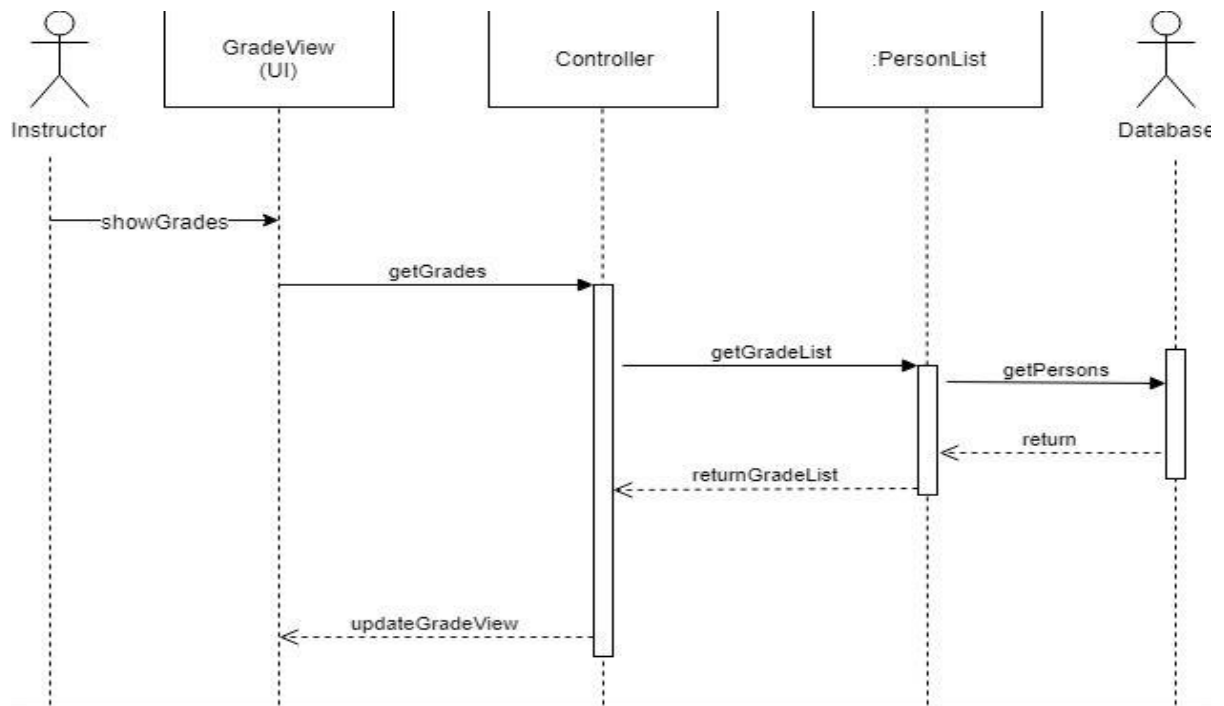


Figure 4.4.3: grades System Sequence Diagram

4.4.4 UC-14 giveQuiz

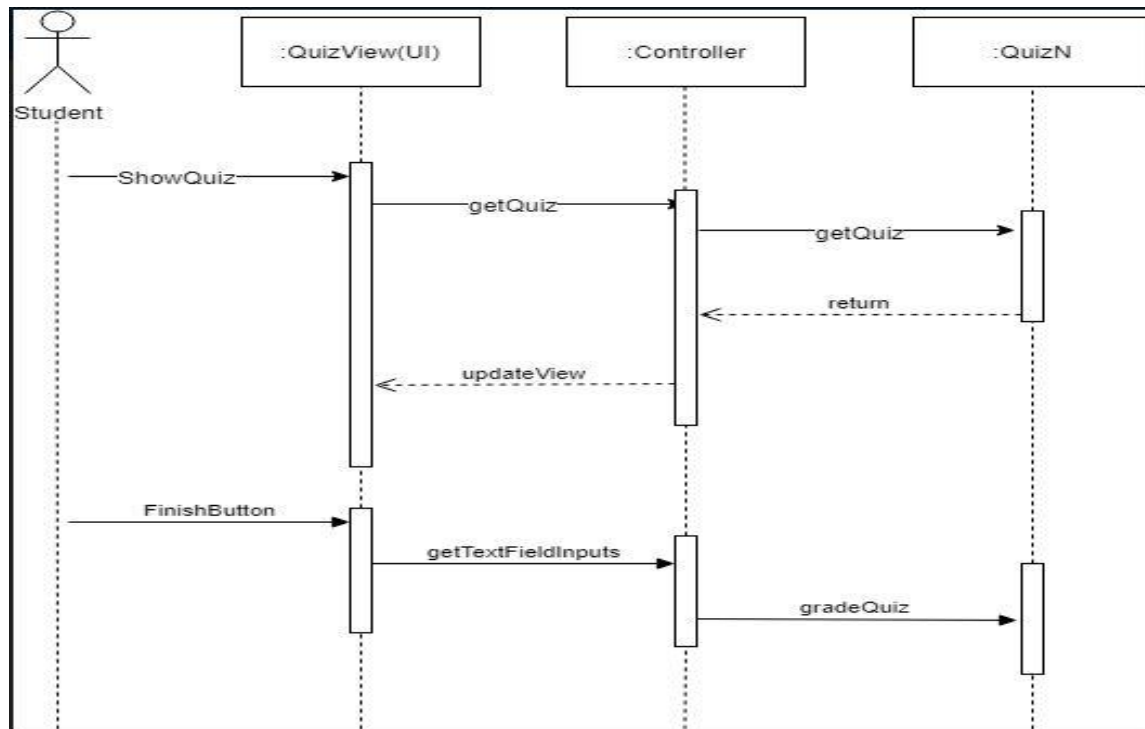


Figure 4.4.4: giveQuiz System Sequence Diagram

4.4.5 UC-15, UC-16, and UC-17 labOne, labTwo, and labThree

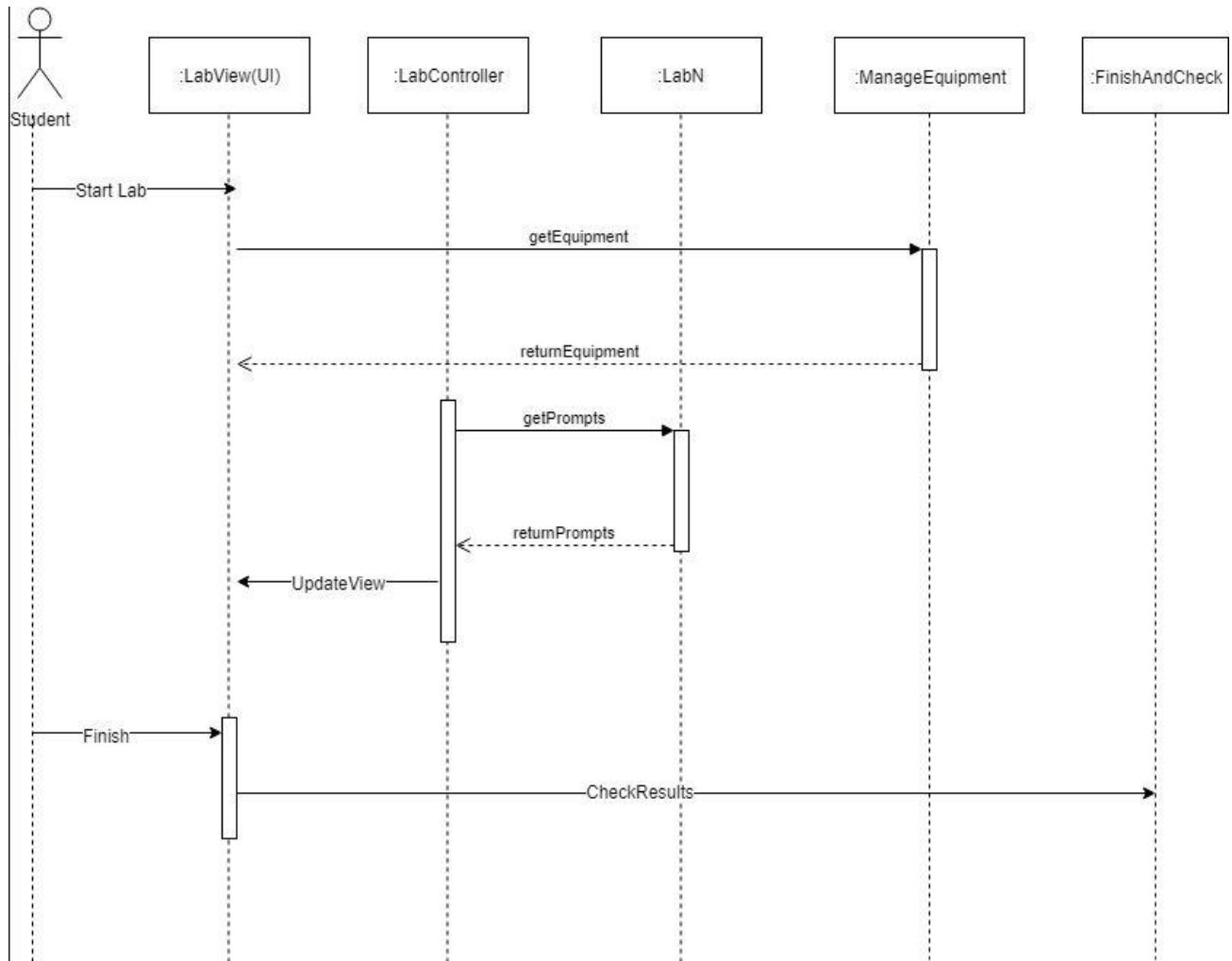


Figure 4.4.5: labOne, labTwo, and labThree System Sequence Diagrams

The system sequence diagram in Figure 4.4.5 demonstrates the general sequence for all three labs, as they follow a very similar pattern:

5. Effort Estimation Using Use Case Points

0. Log In - Total variable presses + 1/0 clicks

- a. Enter Username (variable length)
- b. Enter Password (variable length)
- c. Press enter/click log in

1. Manage Equipment - Total 3 Clicks

- a. Click on drop down menu that holds all the equipment
- b. Click on the equipment desired which places the equipment under the mouse
- c. Click anywhere in the view to place the equipment

2. Grades - Total 2 Clicks

Assumption: Instructor logs in

- a. Instructor clicks on the grades tab

3. Set Cheat Detector Camera - 2 Clicks

Assumption: Student is already logged in and camera exists

- a. Click on "ProctorTrack" (name to be changed) tab.
- b. Press calibrate and wait for software approval

4. Chat - Total 2 clicks + 1 press

Assumption: Student is already logged in

- a. Click on Chat tab
- b. Click on the text box on the bottom to enter text
- c. Press enter

5. Create User - Total 3 clicks

Assumption: Instructor is already logged in

- a. Click the User Management tab
- b. Enter username and password
- c. Click create user

6. Select Lab - Total 2 clicks

Assumption: User is already logged in

- a. Click on the Labs tab
- b. Click on the lab of choice

7. Individual labs

- Selecting new device - total 3 clicks
 - Click on dropdown menu
 - Click on device
 - Click on the lab scene
- Connecting two nodes with wire - total 3 commands/clicks
 - Press W to start the wire connection

- Click on first node
 - Click on second node
- Finish and check - Total 1 click
 - Click the finish button
- Magnifying glass - Total 1 click
 - Click on the magnifying glass and drag over an IC chip
- Trash - Total 1 Click
 - Click on the targeted device, and drag to trash
- Total Lab Simulation - Various amounts of clicks with a combination of the above operations

8. Individual quizzes

- Insert Answer - Total 1 Click + Answer Character Amount
 - Click on the input field, and enter the dynamic answer
- Finish and Check answer - Total 1 Click
 - Click on the button to finish

6. Domain Analysis

6.1 Domain Model

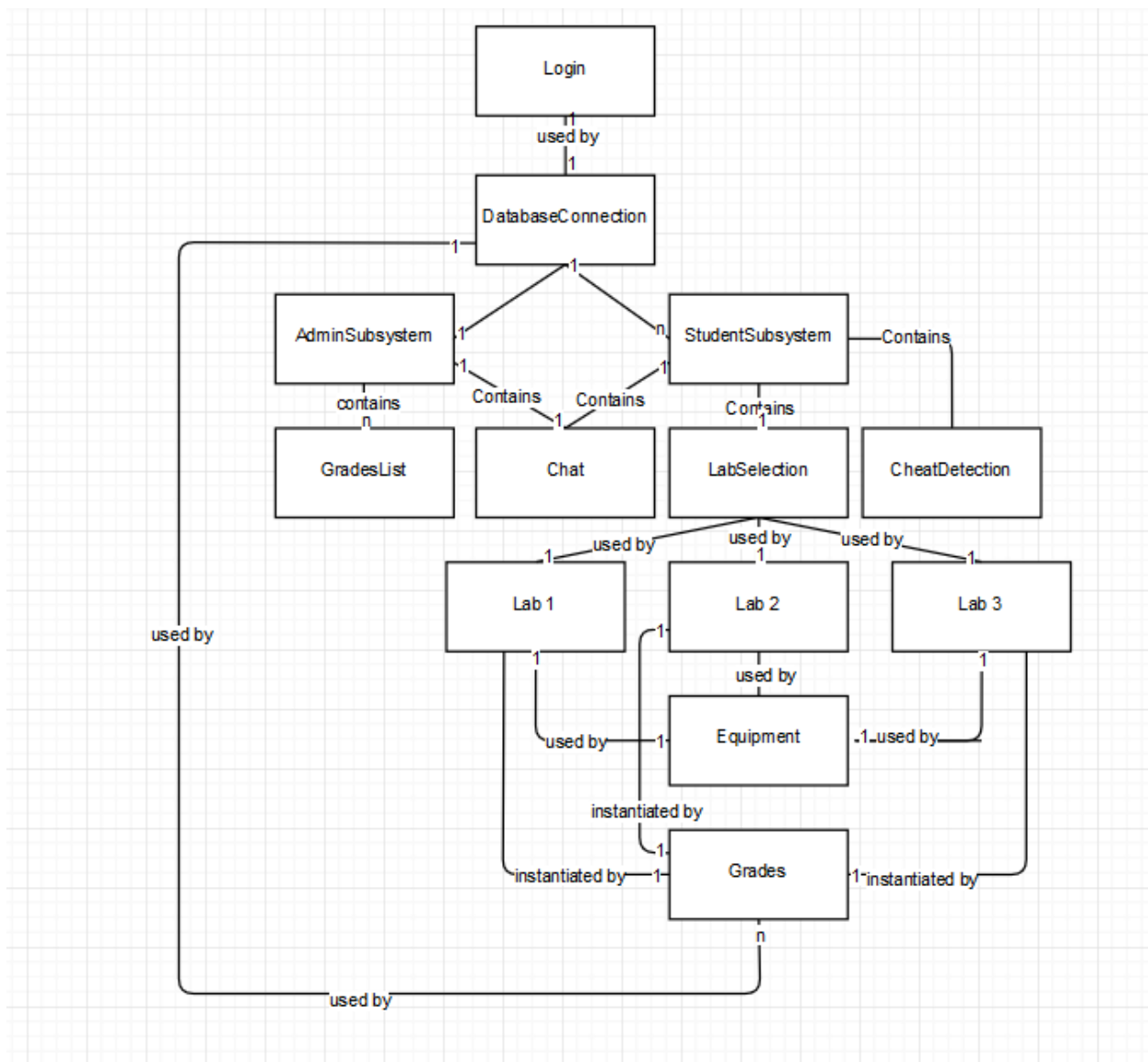


Figure 6.1: The Domain Model

6.1.1 Concept Definitions

Table 6.1.1 lists the system's concept definitions. The first column is the general responsibility description. The second denotes whether the concept is of type "D" (Doing) or type "K" (Knowing). The final column is the name of the concept.

Responsibility Description	Type	Concept Name
Provides the interface GUI for the actors to login to the application, the authentication is done through the database	K	Login
Allows the user to communicate with other users	D	Chat
Stores student account data	K	Person
Presents the user with lab 1 activity	D	Lab1
Presents the user with lab 2 activity	D	Lab2
Presents the user with lab 3 activity	D	Lab3
Provides user with tool to interact with the lab environment in order to perform the lab	D	Equipment
Generates a summary of lab performance results	D	Grade
Provides an interface for the student to interact with GUI and interacts with Lab classes to show the laboratory simulations	D	StudentSubsystem
Provides an interface for the instructor to interact with the GUI	D	StudentSubsystem
Identifies if the user is cheating by using the camera	D	CheatDetector
Holds 'Persons' in an online database, and holds chat messages	D	Database
Administers a quiz to the student at the end of lab 1	D	Lab 1
Administers a quiz to the student at the end of lab 2	D	Lab 1
Administers a quiz to the student at the end of lab 3	D	Lab 1
Student that participates in the Laboratory Simulation portion of the application	D	Student
Instructor that views the grade and oversees the account creations in the application	D	Instructor

6.1.2 Association Definitions

Table 6.1.2 displays the association definitions, showing how the concepts interact with one another. The first column shows the concepts that are interacting. The second column is a

description of how the concepts associate with each other. The last is the name of the association, which is put into broad categories based on the type of interaction.

Concept Pair	Association Description	Association Name
Student ↔ StudentGUI	Student passes request to StudentGUI to display student interface qualities	Conveys request
Instructor ↔ InstructorGUI	Instructor passes request to InstructorGUI to display instructor interface qualities	Conveys request
StudentGUI ↔ LoginGUI	Actor's interface signifies that a student is attempting to sign in.	Conveys request
InstructorGUI ↔ LoginGUI	Actor's interface signifies that an instructor is attempting to sign in.	Conveys request
LoginGUI ↔ StudentGUI	Login passes request to StudentGUI to trigger student interface from the Student class	Generates
LoginGUI ↔ InstructorGUI	Login passes request to InstructorGUI to trigger instructor interface from the Instructor class	Generates
Lab ↔ Equipment	Active Labs will allow students to interact with equipment as they build the necessary circuits	Generates
Lab1 ↔ Quiz1	The completion of a lab 1 triggers a quiz that the student must complete to receive a grade	Generates
Lab2 ↔ Quiz2	The completion of a lab 1 triggers a quiz that the student must complete to receive a grade	Generates
Lab3 ↔ Quiz3	The completion of a lab 1 triggers a quiz that the student must complete to receive a grade	Generates
Lab ↔ CheatDetector	Active labs enable CheatDetector to identify if a student is using outside materials	Requests notify
CheatDetector ↔ Camera	CheatDetector accesses camera and uses it to monitor student as labs and quizzes are in progress	Provides data
StudentGUI ↔ CheatDetector	The GUI uses the CheatDetector to allow the student to calibrate his/her face and verify.	Generates
Student ↔ Report	The student class uses the report to assign the student the respective lab's grade.	Provides data
Lab ↔ Report	The lab uses the report to generate the grade	Generates

	for the lab portion of the simulation	
Quiz↔Report	After the quiz is finished, the quiz generates a grade report for the Quiz portion of the simulation	Generates
Chat↔StudentGUI	The chat functionality is displayed in the Student section of the GUI	Generates
Chat↔InstructorGUI	The chat functionality is displayed in the Instructor section of the GUI	Generates
Chat↔Database	The chat queries the database for new messages	Requests notify
CheatDetector↔Student	The cheat detector uses the image associated with the student to verify that the user is the correct student.	Requests notify
Student↔Database	Generate student class from the database	Generates
Instructor↔Database	Generate instructor class from the database	Generates

Table 6.1.2: Association Definitions

6.1.3 Attribute Definitions

Table 6.1.3 contains the attribute definitions. The first column lists the concepts. The second contains the attributes that the concept is associated with. The final column provides a brief description of the attribute.

Concept	Attributes	Attribute Description
Login	checkType()	Checks if the user is a professor or a student
	authenticate()	Checks if the username and password match the record in the database
Lab	compareOutput()	Compares the output of the logic to the answer
	saveProgress()	Saves student's current work to database
Equipment	getEquipment()	Gets the equipment that the user wants ready to be placed
Report	EvaluateLab()	Evaluates the lab to generate a grade for the specified lab.

	EvaluateQuiz()	Evaluates the quiz to generate a grade for the specified quiz
	GetGradeFromLab()	Returns grade from the specified lab for the student class
StudentGUI	processCommand()	Processes any GUI commands given (generic name for future implementation)
InstructorGUI	processCommand()	Processes any GUI commands given (generic name for future implementation)
CheatDetector	retrieveData()	Retrieves camera footage
	verifyPerson()	Uses facial recognition to determine if student is cheating
Quiz	giveQuiz()	Administers quiz to student
	recordResults()	Records results of quiz
	checkAnswers()	Checks the users answers with the recorded actual answers.
Database	authenticatePerson()	Connects to the online database and authenticates the entered Username and Password with the data in the database.
LoginGUI	checkType()	Checks the user's type (student/instructor)
	launchGUI()	Launches the respective GUI for the student or the instructor
	authenticate()	Calls on the database to authenticate the entered username and password
Student	resetGrade()	Method to reset the specified lab grade from the Instructor
Instructor	getGrade()	Method to get a list of students' grade
Chat	enterMessage()	Allows user to enter a custom message to the chat

Table 6.1.3: Attribute Definitions

6.1.4 Traceability Matrix

Table 6.1.4 is the traceability matrix, showing how the use cases interact with the domain concepts.

	Domain Concepts																	
Use Case	PW	LoginGUI	Chat	Person	Lab1	Lab2	Lab3	EquipmentManager	Report	StudentGUI	InstructorGUI	CheatDetector	Database	Quiz1	Quiz2	Quiz3	Student	Instructor
UC-1	55							X										
UC-2	17									X								
UC-3	3	X		X									X				X	X
UC-4	4												X					
UC-5	6										X							X
UC-6	5										X							
UC-7	2							X		X								
UC-8	2									X								
UC-9	2									X								
UC-10	1									X	X							
UC-11	1		X															
UC-12	8								X									
UC-13	2											X						
UC-14	27													X	X	X		
UC-15	25				X													
UC-16	19					X												
UC-17	15						X											

Table 6.1.4: Traceability Matrix

6.2 System Operation Contracts

Each of the boxes below contains a system operation contract. They display the classes and functions that will be utilized by the system, as well as any variables that certain functions may be dependent on.

Name: manageEquipment(partType, partLocation)

Responsibility: Control the movement of the pieces by responding to the user's click/drag

Cross-References: UC-1

Output: The system can introduce new object to the board and have them interact with each other

Pre-Conditions: Student clicks on and drags a piece to place on the protoboard

Post-Conditions:

- An Equipment object is created with the ability to interact with the protoboard
- Different parts interact with each other as they are moved around the board
- Location values for each object constantly change as they are moved around

Name: check();

Responsibility: To check to see if the solution that a student has submitted is correct

Cross-References: UC-2

Output: The system will either output a message saying that the user is correct, or display a grade and a description of the errors that were made

Pre-Conditions: The student finishes a task and presses the finish button

Post-Conditions:

- The student's answers are assigned to variables and compared to the values of the correct answers to test for equivalence
- Variable grade is assigned a value based on percentage of correct responses
- Test if grade equals 100 (perfect score), and will display congratulations message accordingly

Name: grade(labScore);

Responsibility: Receives the scores from each lab that the user completes and copies the grade into the gradebook for both students and teachers to view

Cross-References: UC-5

Output: The output to this is the grade that the student received, and it is available to both students and teachers

Pre-Conditions: The student has finished a lab and submitted his/her work for grading

Post-Conditions:

- Each student object has their grade variable assigned to them
- Grade becomes viewable by students and instructors

Name: quiz()

Responsibility: Give the student s a quiz upon the completion of each lab

Cross-References: UC-14

Output: A quiz is given to the user, which they must complete

Pre-Conditions: The user completes and submits a lab

Post-Conditions:

- The user is presented with a quiz, typically the image of a Karnaugh Map or Truth table that must be solved
- Empty spaces must be filled with values, which will be compared to the correct values after submission

Name: lab1()

Responsibility: To allow the user to perform tasks given to them in lab 1

Cross-References: UC-15

Output: Brings the user to a screen where they are able to complete lab 1

Pre-Conditions: The user must select “Lab 1” from the labs menu

Post-Conditions:

- The user is given the ability to complete lab 1
- System displays image of a truth table, which user must fill out correctly
- There is no circuit design for lab 1
- quiz() is called to give the user a quiz afterwords

Name: lab2()

Responsibility: To allow the user to perform tasks given to them in lab 2

Cross-References: UC-16

Output: Brings the user to a screen where they are able to complete lab 2

Pre-Conditions: The user must select “Lab 2” from the labs menu

Post-Conditions:

- The user is given the ability to use lab 2
- manageEquipment(partType, partLocation) is called often as the user moves pieces around the board
- quiz() is called to give the user a quiz afterwords

Name: lab3()

Responsibility: To allow the user to perform tasks given to them in lab 3

Cross-References: UC-17

Output: Brings the user to a screen where they are able to complete lab 1

Pre-Conditions: The user must select “Lab 3” from the labs menu

Post-Conditions:

The user is given the ability to use lab 2
manageEquipment(partType, partLocation) is called often as the user moves pieces around the board
quiz() is called to give the user a quiz afterwords

7. Interaction Diagrams

The figures below are the interaction diagrams for several of our use cases. Each one has a description below it, explaining in detail how the system is set up and how the interactions are carried out.

7.1 UC-1: manageEquipment

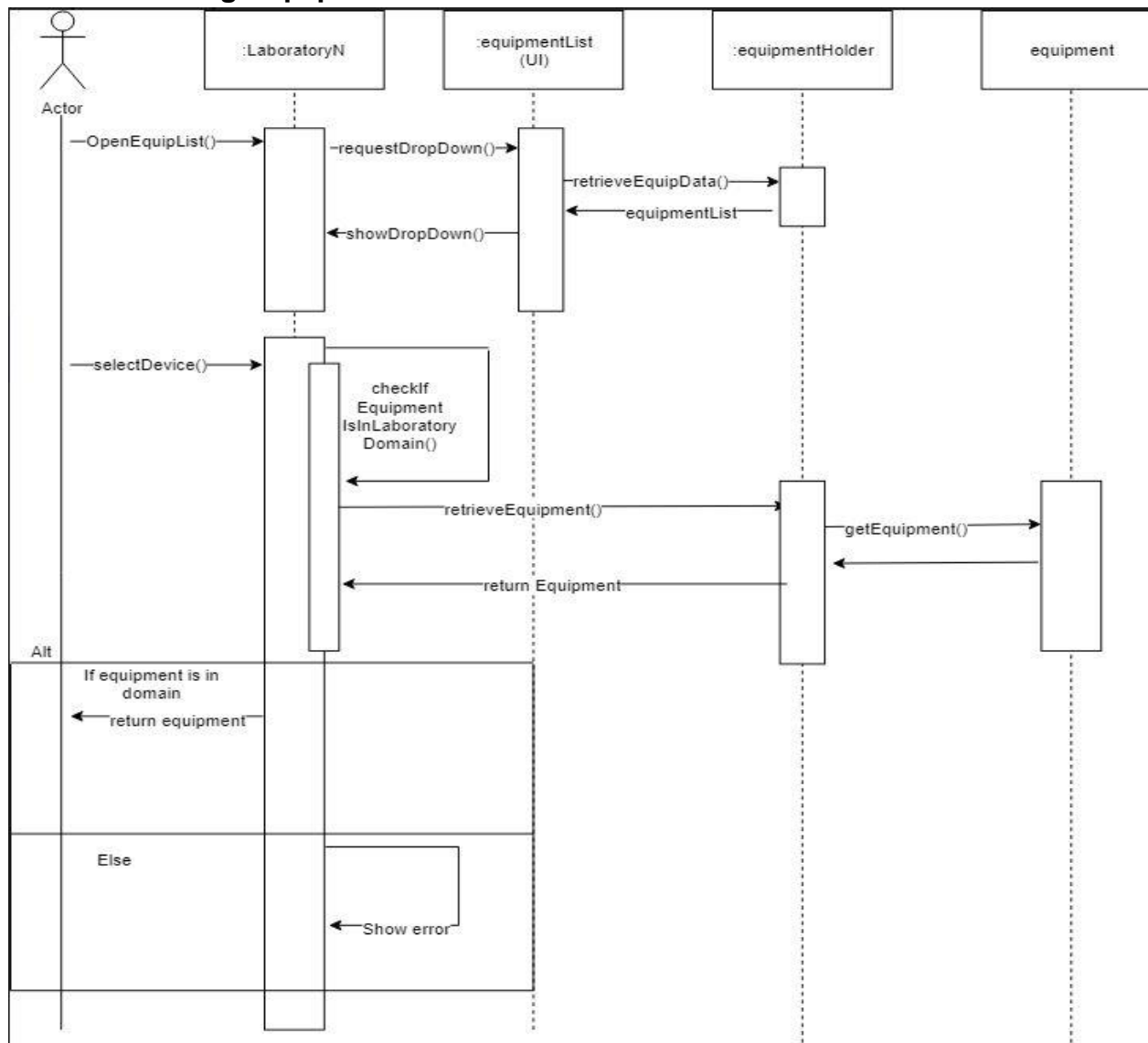


Figure 7.1: manageEquipment Interaction Diagram

The core of this particular sequence diagram is the equipmentList, equipmentHolder, and equipment objects. EquipmentList is the graphical user interface component that shows the list of equipments available to the user. This is coupled with the equipmentHolder class which holds the list of equipment objects. This allows us to easily change the type and number of equipments without worrying about breaking the functionality of the equipmentList component. LaboratoryN is the 'controller' which takes the responsibility of communicating with the equipment related classes. The inner workings of this structure are displayed above in Figure 7.1.

7.2 UC-2: finishAndCheck

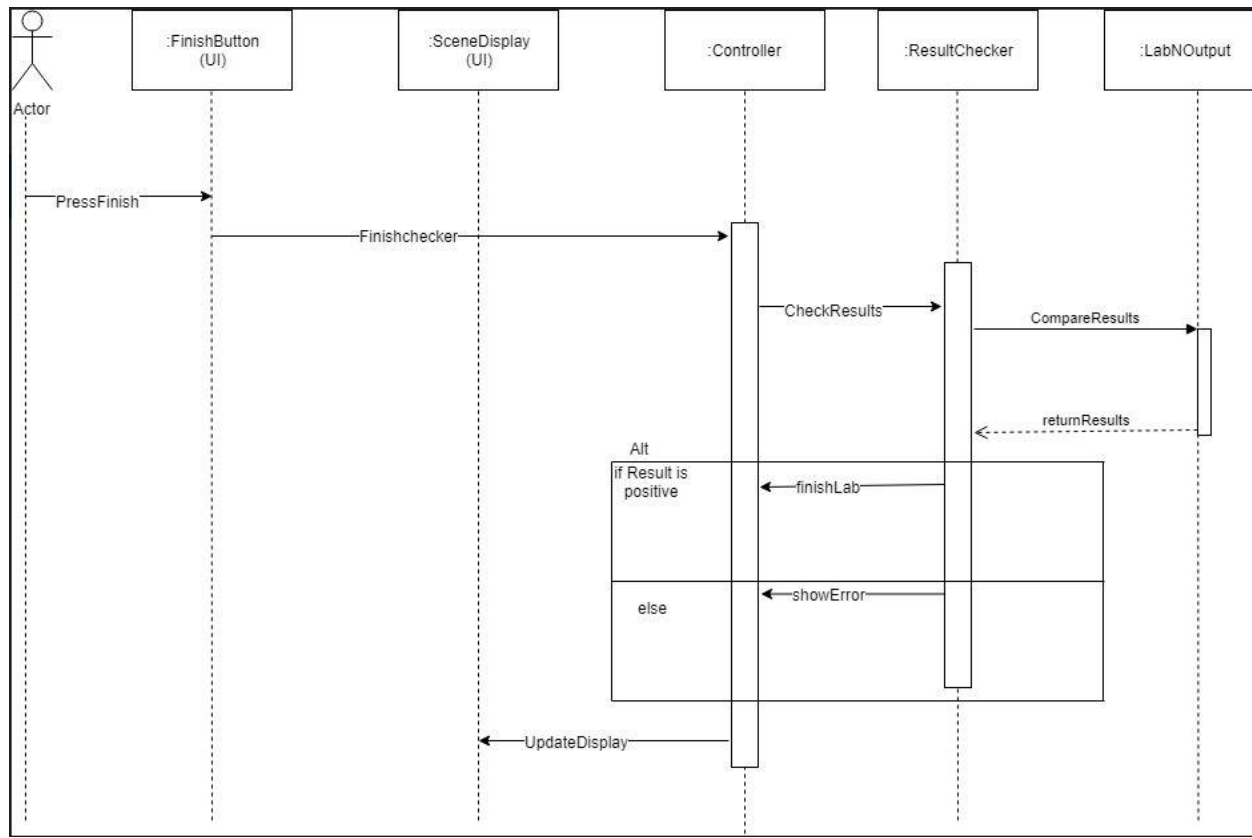


Figure 7.2: finishAndCheck

In the scenario depicted in Figure 7.2, the user (student) needs a method of notifying the system that he/she is done with a particular section of the lab. This is done through a button tied to the user interface (FinishButton). After this button is clicked, the system transfers control to the controller. The controller signals the ResultChecker entity to check whether the user's answers to the particular section of the lab are correct. The ResultChecker then signals the lab output (Lab3Output, Lab2Output, or Lab1Output) to compare the student's inputs with the correct inputs, and then return the results. If the student's answers are correct, the ResultChecker notifies the controller that the sequence to finish the lab can be initiated. Otherwise, an error is returned to the controller. Depending on whether the student was successfully able to answer the lab section correctly or incorrectly, and the number of attempts remaining, the controller will then instruct the SceneDisplay user interface to update the GUI accordingly.

7.3 UC-5: grades

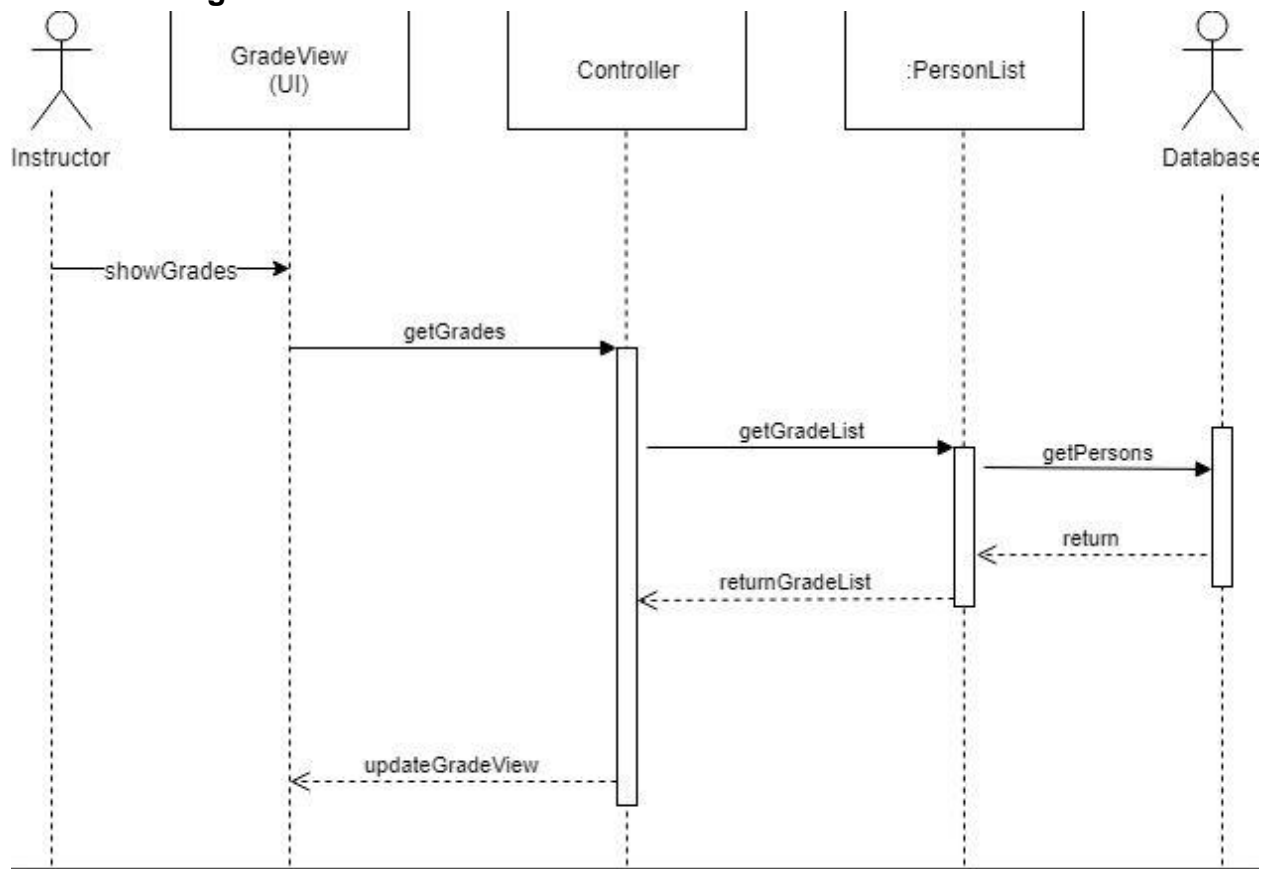


Figure 7.3: grades Interaction Diagram

The scenario depicted in figure 7.3 begins with the instructor attempting to look at the grades for a particular student or group of students. The instructor interacts directly with the GradeView user interface, and when the instructor requests grades, the controller is signalled from the interface. The controller then contacts PersonList, which goes through the database containing the names and grades for each student. The database then returns these values to PersonList, which then passes it back to the controller, and in turn to GradeView, which displays the information for the user to see.

7.4 UC-14: giveQuiz

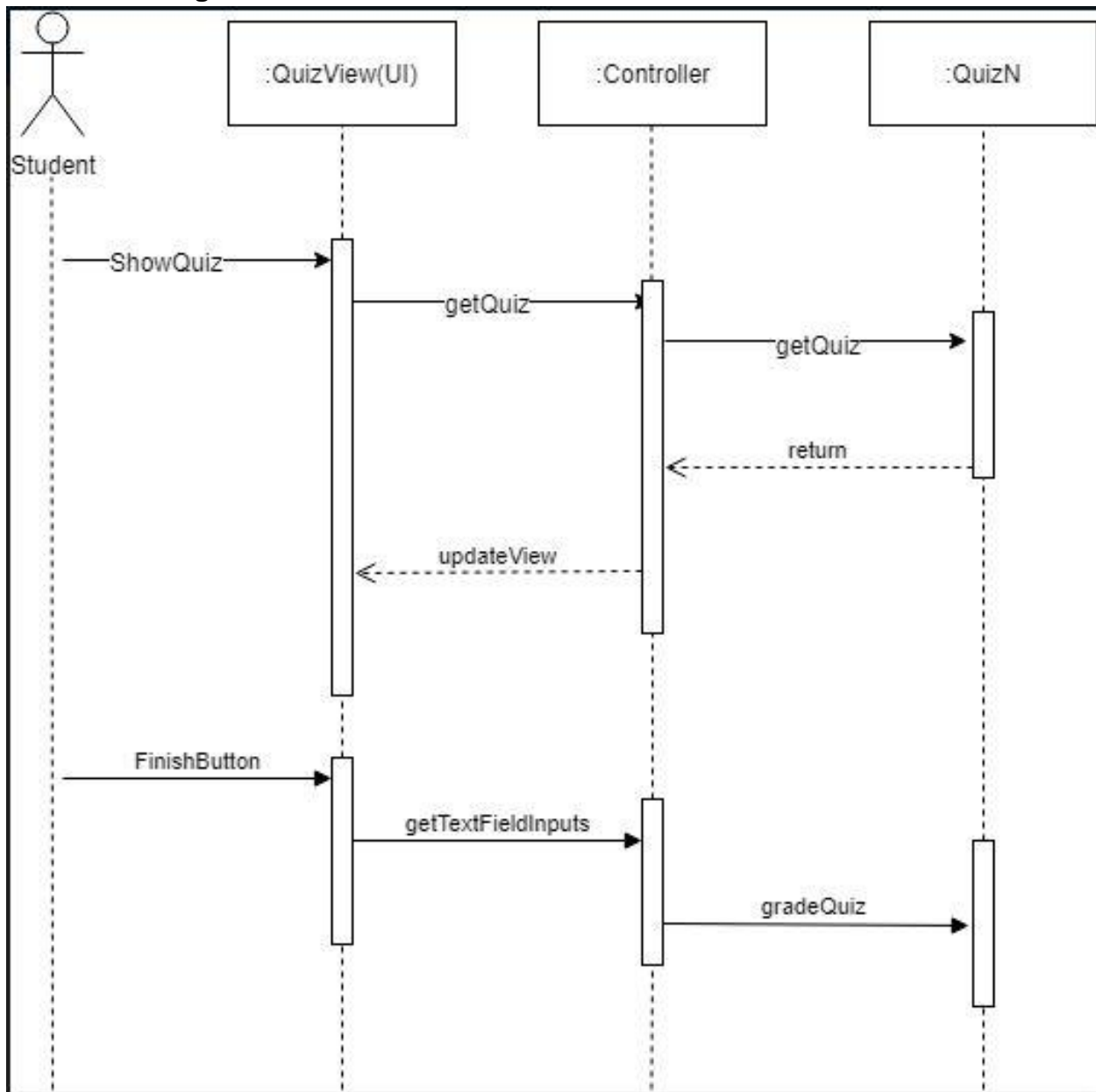


Figure 7.4: giveQuiz Interaction Diagram

For the scenario in Figure 7.4, the Single Responsibility Principle was what we attempted to implement. The QuizView is responsible for showing the graphical representation of the quiz, the controller receive information from the specific quiz to control the QuizView's graphical capabilities. The QuizN class has the responsibility of holding the questions and checking the answers entered into the QuizView. The controller delivers the answers the the QuizN class.

7.5 UC-15, UC-16, UC-17: labOne, labTwo, labThree

The system sequence diagram below demonstrates the general sequence for all three labs as they follow a very similar pattern:

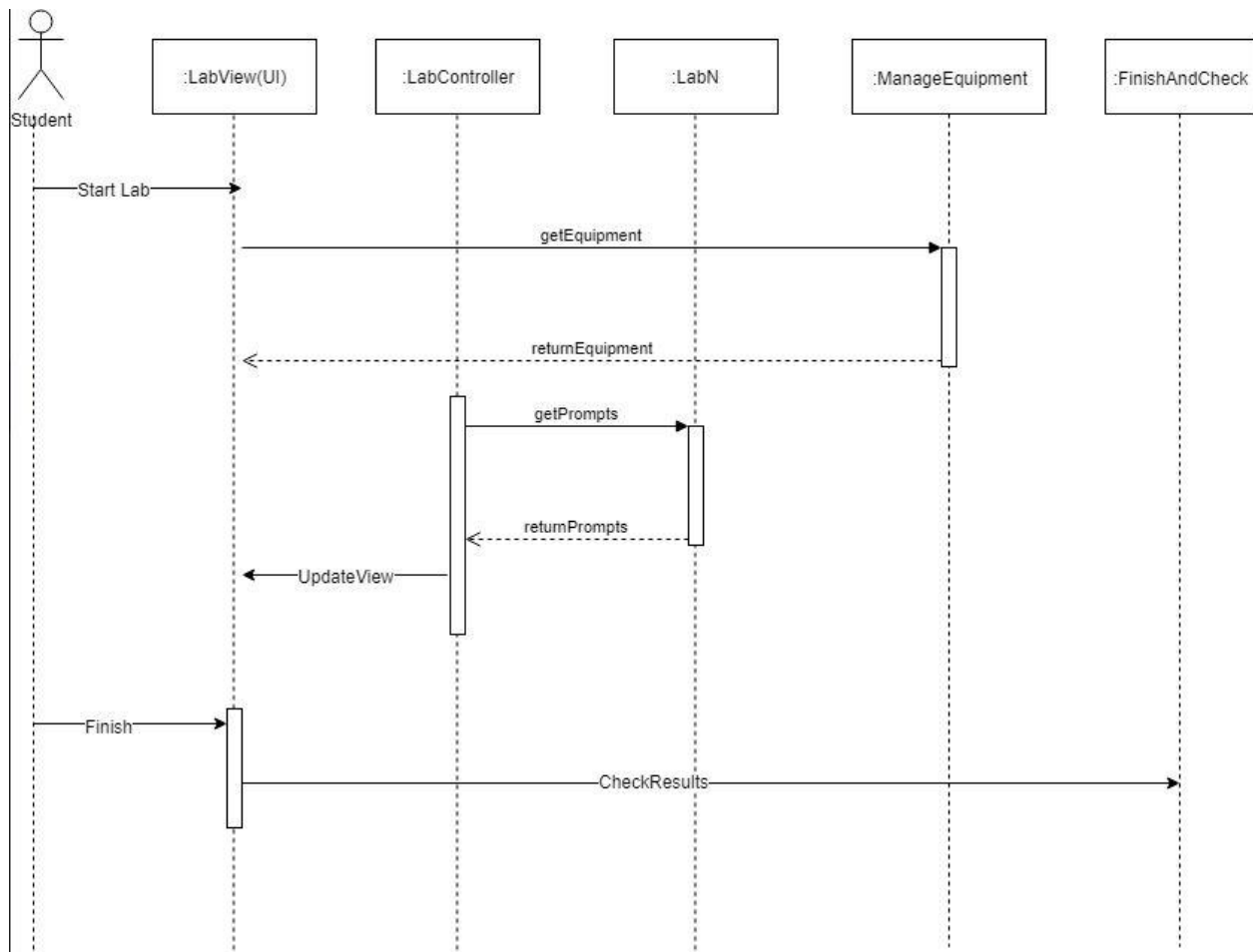


Figure 7.5: labOne, labTwo, labThree Interaction Diagram

This sequence diagram tries to implement is little coupling as possible between the involved classes.

Note: The ManageEquipment class is a representation of the UC-1 Sequence Diagram shown above. We attempted to separate as many of the functions as possible to ensure there is opportunity to dynamically change either the actions taken in when selecting equipments, checking the results of the laboratory, and the prompts presented by the particular lab. We tried to ensure that the Single Responsibility Principle is kept, where the LabView is only responsible for rendering the graphics to the screen, the LabController is responsible for controlling the view, and LabN is in charge of the lab specific instructions needed for the user.

8. Class Diagram and Interface Specification

8.1 Class Diagram

Figures 8.1.1 and 8.1.2 are the class diagrams for our system. They are separated to make each one more visible and easier to understand. Equipment manager received its own class diagram because of the complexity of its inner structure.

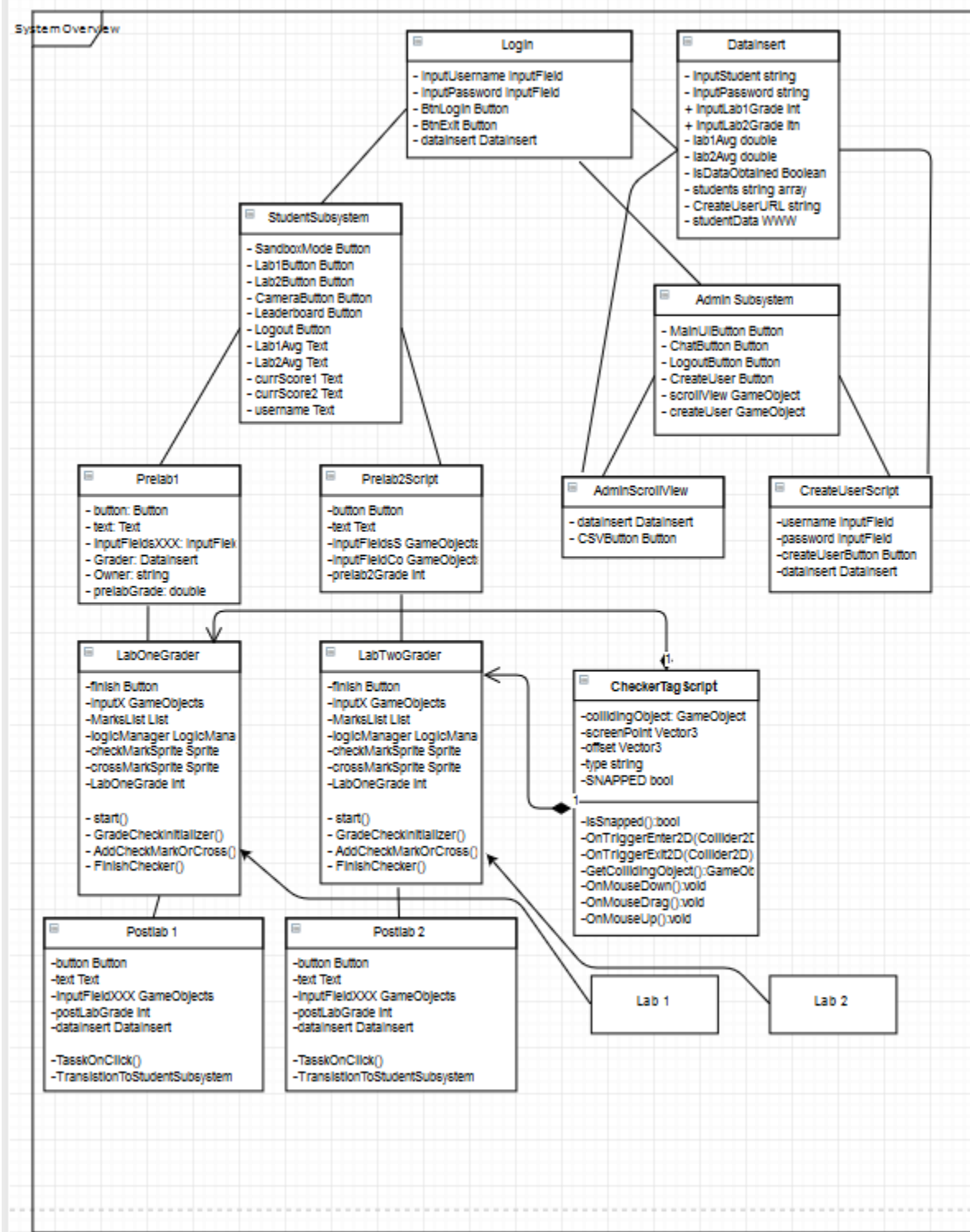
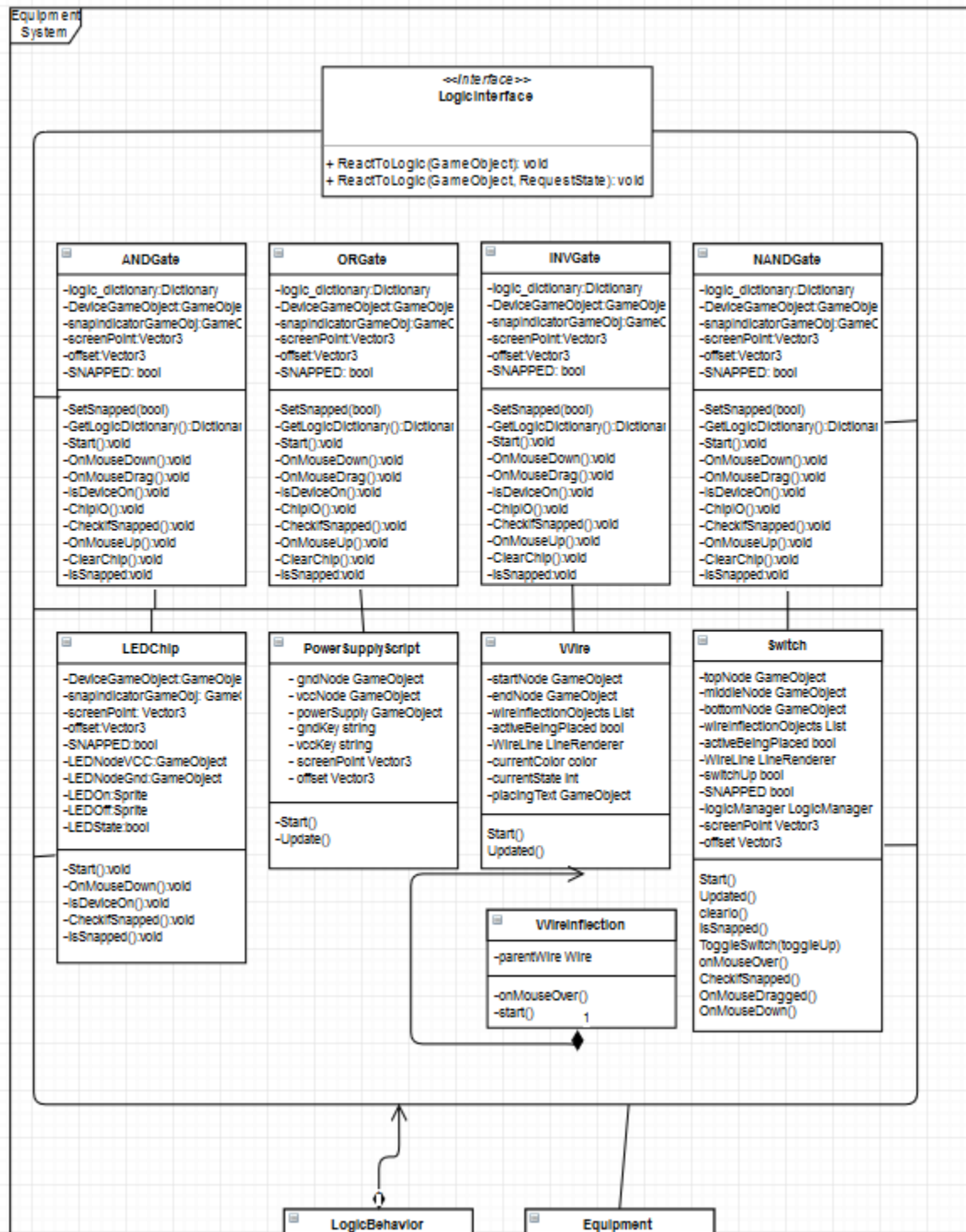


Figure 8.1.1: General Class Diagram (Updated)

8.1.2 Equipment Manager Inner Structure



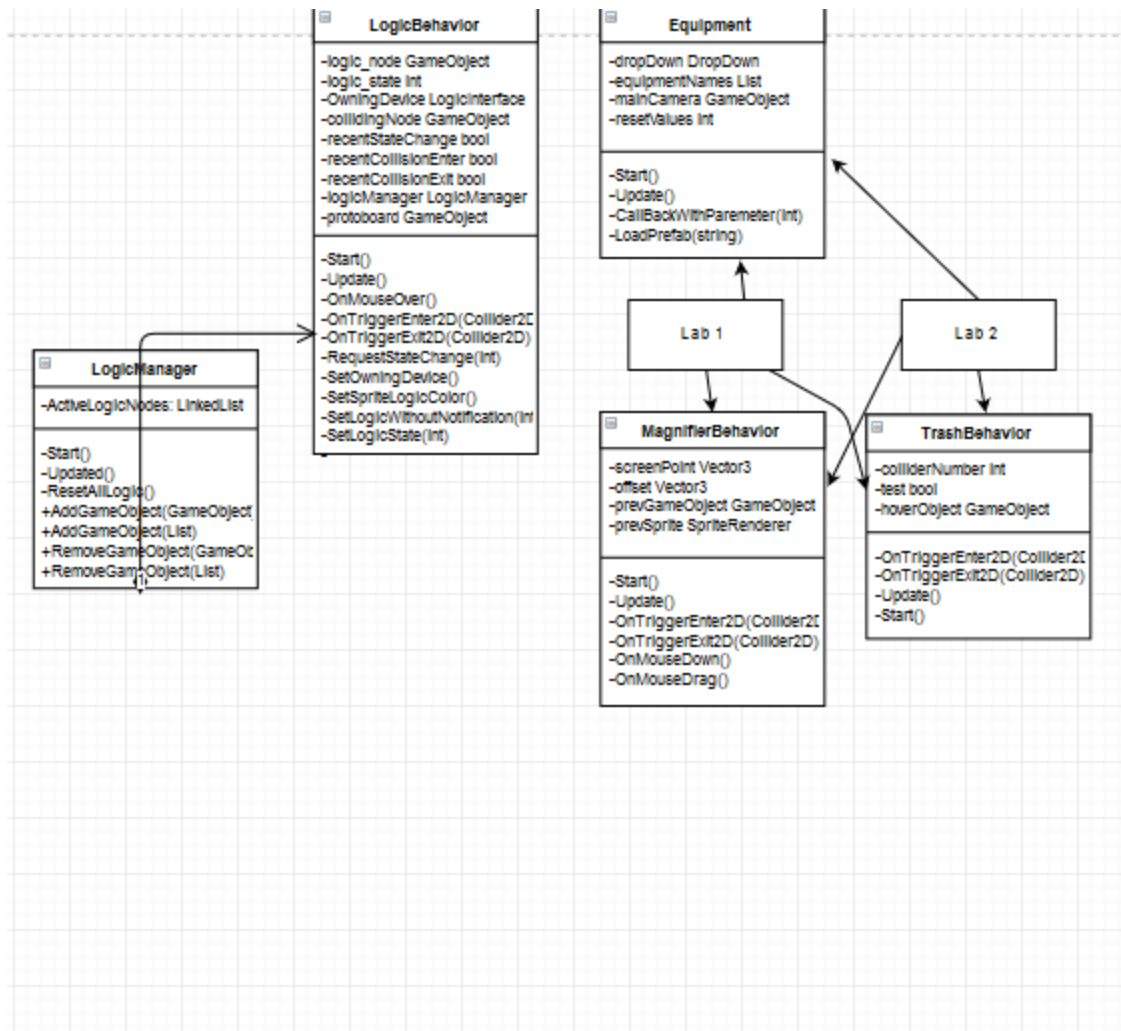


Figure 8.1.2: Equipment Manager Inner Structure Class Diagram (Updated)

8.2 Data Types and Operation Signatures

Each of the boxes below is representative of a class in our system. The tables have 2 distinct sections: an attribute section containing any important values or variables to that class, and an operations section that lists the functions available for that class to use. Both of these are aided by descriptions of their functionality.

Class Name:	LoginGUI - Provides interface for login	
Attributes:	-username:string -password:string -accountType:type	Stores username Stores password Stores account type (student/instructor)
Operations:	+checkType() +launchGUI():String +authenticate()	Checks the type (student/instructor) Initiates login GUI for users Tests if username and password are correct

Class Name:	Database - stores users, passwords, grades	
Attributes:	-DatabaseKey:String	Stores key for database
Operations:	+authenticatePerson():Person	Tests if user is in database

Class Name:	Chat - Allows students to chat with each other and instructors	
Attributes:	-Messages:List	Stores the chat messages
Operations:	+enterMessage():String	Allows student/instructor to input a new chat message

Class Name:	Person - Responsible for keeping track of individuals	
Attributes:	AccountType:String Name:String	Holds type of account (student/instructor) Stores name of user
Operations:	-	-

Class Name:	Student - Keeps track of a particular student's grades and other info	
Attributes:	-Lab1Grade:float -Lab2Grade:float -Lab3Grade:float	Holds lab 1 grade Holds lab 2 grade Holds lab 3 grade

	-cheatDetectorVerification:img	Stores image of user to identify cheating
Operations:	+resetGrade(int labNumber)	Resets a grade for a specific lab (designated by input labNumber)

Class Name:	StudentGUI - interface for students	
Attributes:	-GUI Elements: GUIType	Contains the buttons, textfields, and other GUI attributes that are used in the StudentGUI.
Operations:	+processCommand()	Processes user commands

Class Name:	Instructor - keeps track of instructor information	
Attributes:	-personListGUI Elements: -GUIType:List	Contains the buttons, textfields, and other GUI attributes that are used in the personListGUI and other Instructor GUI elements.
Operations:	+getGrades(): List	Gets student grades for instructor

Class Name:	InstructorGUI - interface for instructors	
Attributes:	-GUI Elements: GUIType	Contains type of GUI (instructor)
Operations:	+processCommand()	Processes user commands

Class Name:	Report - delivers grade reports to students and instructors	
Attributes:	-LabGrade: string	Contains grades for lab
Operations:	+evaluateLab() +evaluateQuiz() +getGradeFromLab(): string	Compares lab performance to correct solutions Compare quiz solutions to correct solutions Obtains lab grades

Class Name:	Lab1, Lab2, Lab3 - control the operations of the individual labs	
Attributes:	-LabName: string -LabInstructions: List -ExpectedOutput: GameObj	Contains name of lab Contains lab instructions Contains correct results
Operations:	+compareOutput():output	Compares student answers to correct output

	+saveProgress()	Saves student lab progress
--	-----------------	----------------------------

Class Name:	Quiz1, Quiz2, Quiz3 - controls operations of the quizzes	
Attributes:	-questions_answers:Map	Holds answers to questions
Operations:	+giveQuiz() +checkAnswers(): List +recordResults()	Gives student quiz Checks for correct answers Records student grade on quiz

Class Name:	EquipmentManager - handles all equipment for the labs	
Attributes:	-equipmentList: List	Contains pieces of lab equipment
Operations:	+getEquipment(): string	Gets selected equipment piece

Class Name:	CheatDetector - identifies if students are cheating	
Attributes:	-camera: object	Computer camera is used to monitor students
Operations:	+verifyPerson(): img +retrieveData()	Verifies that the student matches his/her stored image Retrieves stored images of students to compare to camera image

Class Name:	LogicBehavior - is a component (A subclass of a GameObject) of a GameObject called LogicNode which sets the voltage states of a particular node in the Electrical Device.	
Attributes:	+LogicID: String +LogicState: int -LogicNode: GameObject -OwningDevice: GameObject	The unique ID for each LogicNode gameobject. The state that emulates the circuit voltage of the particular node (Low, High, Invalid). The device that owns the particular logicNode.
Operations:	-Start() -Update()	Initializes the conditions for the component script. Allows the logicBehavior script to react to any changing input from the user every frame.

Class Name:	ProtoboardObject - Object that holds the logic nodes for a protoboard gameobject and changes the state of its nodes accordingly.	
Attributes:	-NodeDictionary	Dictionary (Hash Table) of LogicID to Node GameObject mappings.
Operations:	-start() -update() -setNodeProperties()	Initializes the conditions for the component script. Allows the device script to react to any changing input from the user every frame. A helper function to help the protoboard initialize the logic nodes inside it.

Class Name:	PowerSupply - Object that holds the logic nodes for a PowerSupply gameobject and changes the state of its nodes accordingly.	
Attributes:	-NodeHashMap	Dictionary (Hash Table) of LogicID to Node GameObject mappings.
Operations:	-start() -update() -setNodeProperties()	Initializes the conditions for the component script. Allows the device script to react to any changing input from the user every frame. A helper function to help the protoboard initialize the logic nodes inside it.

Class Name*:	AndGate/OrGate/XORGate/MuxChip/NandGate/WireBehavior- Object that holds the logic nodes for a that particular gameobject and changes the state of its nodes accordingly.	
Attributes:	-NodeHashMap	Dictionary (Hash Table) of LogicID to Node GameObject mappings.
Operations:	-start() -update() -setNodeProperties()	Initializes the conditions for the component script. Allows the device script to react to any changing input from the user every frame. A helper function to help the protoboard initialize the logic nodes inside it.

*Please note that the above table combined the classes for all of the chips and gates in order to avoid repetition. Each one contains its own NodeHashMap, and has the same three operations: start(), update(), and setNodeProperties().

8.3 Traceability Matrix

Below is our traceability matrix containing classes and domain concepts. It enables one to see how the individual classes satisfy each of the domain concepts. It is broken into three separate tables to make it easier to read, as we have a large number of classes and fitting them all onto a single table would make it difficult to read. Below the table is a description of the mappings.

Domain Concepts	Classes							
	LoginGUI	Database	Person	Student	StudentGUI	Chat	InstructorGUI	Report
Login	X							
Database Connection		X						
Admin Subsystem							X	
Student Subsystem					X			
GradesList							X	X
Chat						X		
Grades								X

Table 8.3.1: Traceability Matrix Part 1

Domain Concepts	Classes							
	Lab 1	Lab 2	Lab 3	Equipment Manager	Quiz1	Quiz2	Quiz3	CheatDetector
Equipment	X	X	X	X				
LabSelection	X	X	X					
Lab1	X				X			
Lab2		X				X		
Lab3			X				X	

CheatDetector								X
----------------------	--	--	--	--	--	--	--	----------

Table 8.3.2: Traceability Matrix Part 2

Domain Concepts	Classes								
	Logic Interface	Logic Behavior	Protoboard Object	Wire Behavior	Power Supply	AND gate	OR gate	NAND gate	XOR gate
Equipment	X	X	X	X	X	X	X	X	X

Table 8.3.3: Traceability Matrix Part 3

Most of our domain concepts mapped very similarly over to our class diagrams with a few exceptions. The Login, StudentSubsystem, and AdminSubsystem domain concepts were changed by adding a postfix 'GUI' at the end of them as much of their functionality are GUI operations. We changed the name DatabaseConnection to Database as we expect that class to handle everything from the initial connection to database management methods. We changed the name of the concept Grades to Report as we felt we wanted more information within that class than just the grades, such as the specific areas where the students failed in the Virtual Lab to diagnose design issues. We changed the Lab domain concepts to contain both a Lab class and a Quiz class as they are designed to be decoupled different systems. The equipment underwent the biggest change as during development of the equipments, we found uses for many classes that specifying different equipments.

9. System Architecture and System Design

9.1 Architectural Styles

Event-driven architecture - This architectural style relies heavily on detection of, and reaction to events, or changes in state. As our system will be an interactive one, much of it will rely on the actions of the users, and must respond accordingly. Our system must be able to respond appropriately to each button click, as well as the click-and-drag features installed in each lab. The systems developments and procedures are almost exclusively reactions to the changes in state brought about by the user. The Unity Game Engine (Framework) is based on an Event driven architecture where each script is initialized in a Start method and the event loop executes the Update method. There are special cases where a callback is necessary, such as when the user clicks the mouse, a mouse operation callback is executed which can be handled in the “scripts” or “classes”.

Database-centric architecture - This type of architecture typically relies on a standard relational database management system rather than in-memory or file-based data structures. We will be relying on databases for a good portion of this project, as a means to store information about each student and instructor, lab grades, quiz grades, usernames, and passwords

What is Unity and why was it used for this project?

This project is built on top of the Unity Engine framework (Version 2017.3.1f1). The Unity Engine is a graphics/game library that can be interfaced with the .NET C# language, or Javascript. For this particular project, we have solely focused on implementing our program in C#. Unity's main job is to render the 2D or 3D models on to the screen (for this case, its 2D) and facilitate a platform to detect interactions between different “GameObjects”.

Everything in Unity is based around the concept of ‘GameObjects’ which have properties or components that describe the GameObject, such as the Transform component, which contains information about the GameObjects position, rotation, and scale on the screen. Each GameObject can take in user made “Scripts” implemented as classes from the Classical Object Model. They all ‘extend’ or ‘inherit’ from Unity's own GameObject class called ‘MonoBehaviour’ which allows the programmer to immediately access information about the GameObject, such as all the components belonging to the GameObject, and allows Unity to add it to its callback systems. Unity's callback system allows us to do powerful, things, such as detect collisions between two GameObjects, detect user inputs such as mouse click. In addition to the advantage of using these GameObject constructs, we can also utilize Unity's multithreading system, called Coroutines, that allowed us to do computationally heavy tasks/or animations without freezing up the main graphical thread of the application.

Unity is an event-driven system, what this means is that there is an inner loop, called the main Update loop which changes the components that each GameObject has during each frame that the loop is called. This allows programmers to dynamically take user inputs, react to changes to the overall system, and update the system based on predefined programmed instructions. What this also means is that it is the programmer's responsibility to ensure that any algorithm taking advantage of this system is efficient so that it doesn't affect the user experience as all computation must be done before moving on to the next frame/loop.

In addition to Unity's powerful libraries, we also heavily utilized the powerful .NET framework libraries that come in with C#. We heavily utilized built in data structures, such as: Dictionaries (Hash Table), Linked Lists, Array Lists, and Arrays in general.

There were several reasons we decided to use Unity over other engines. Our project is a visual, simulation based project. Many visual libraries are built on C++ due to its speed and optimization capabilities, however, due to the scope and experience of our members on this project, we wanted to really focus on our mission itself, to create an accurate simulation of Digital Logic laboratories. We wanted a language that was powerful, but also was easily buildable/testable to different target operating systems and/or platforms. After a significant amount of research, we decided that Unity would be the best framework to work under. It made use of several different languages, including C# which is easily compilable to different platforms due to its Virtual Machine system and JIT compiler (Similar to Java). It also frees the programmers worry about garbage collection, which was important to us. In addition to the language of choice, Unity came out on top due to its extensive documentation, support, popularity amongst independent developers, and most importantly, the ease of building and testing for different platforms.

9.2 Identifying Subsystems

9.2.1 UML Package Diagram

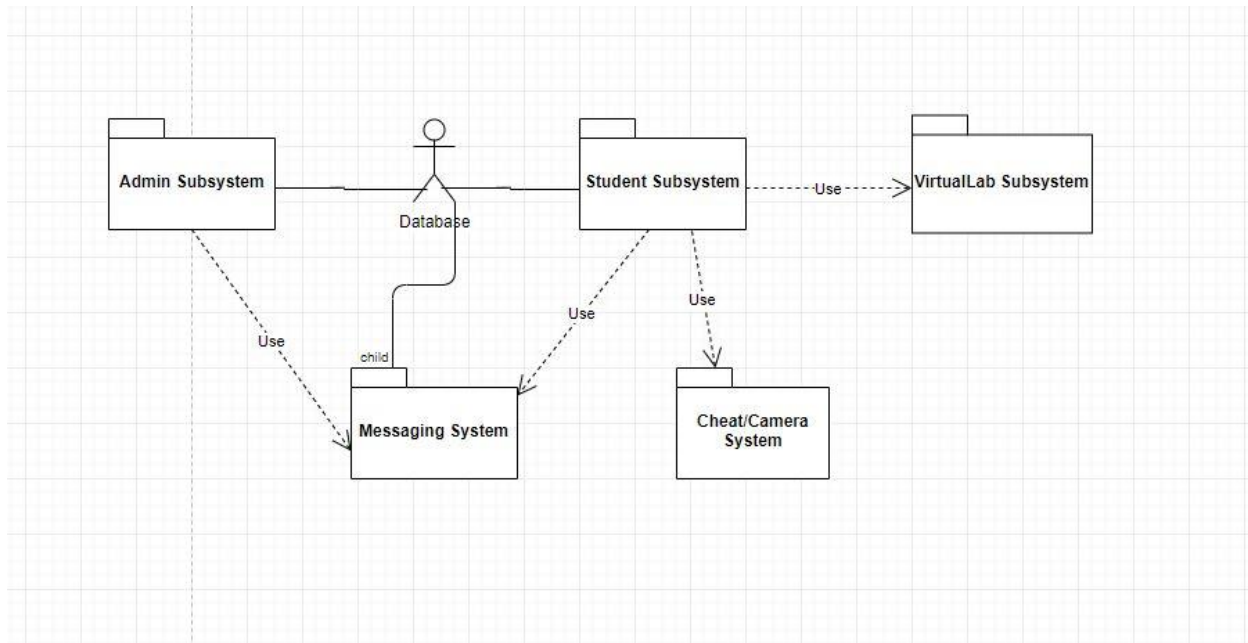


Figure 9.2.1: UML Package Diagram

The reason the packages are assembled the way that they are in Figure 9.2.1 is that they are being developed independently. We can develop each group of classes independently, and test them in such a way, such as developing the Virtual Lab subsystem, and Student Subsystem independently without affecting the other. Each package however contains its own Model-View-Controller structure that allow each subsystem to work. For example, the VirtualLab Subsystem has its own model, which represent the equipment available to the lab, how the equipments react to each other, and what the correct output is for a particular lab. The view is run by the Unity Engine Framework that shows the sprite components of a particular equipment. The controller controls how user input causes the equipments to react and passes the appropriate actions to the model.

9.3 Mapping Subsystems to Hardware

Our system does run on multiple hardwares. We have a client - server interaction between our application and a cloud based relational database. Our database keeps track of the student's data (Student Subsystem) for the administrator to manage (Admin Subsystem). Our server also facilitates the messaging system. When a user, be it the student or the administrator (Instructor/Professor), types in the messaging system, the message is sent to a database and relayed along all other open messaging systems that query it.

9.4 Persistent Data Storage

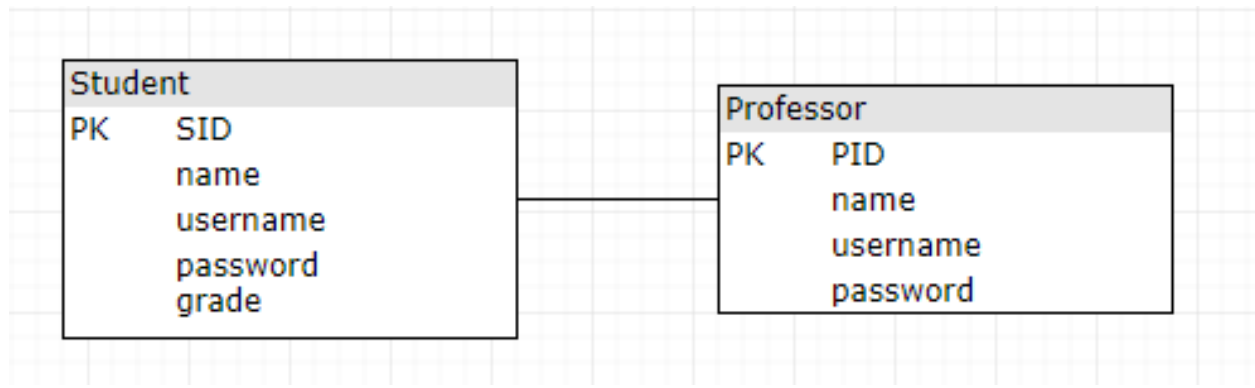


Figure 9.4: Persistent Data Storage Model

Our lab system as an event-driven system so users can do whatever they want in the real lab. Each event will change the associated value in the database. The system tracks how far each user goes in each lab so that their progress is saved on cloud. We use amazon web service to store the data, which is reliable and efficient. Here we have two roles: student and professor. They have their own id, name, username and password. Student has a extra grade item to keep their scores.

9.5 Network Protocol

We will be using the Firebase SDK to help us set up a connection to the firebase server. The SDK will facilitate setting up the connection in C#, which is the programming language of choice for the Unity Engine Framework.

9.6 Global Control Flow

9.6.1 Execution Orderness

Our system is event-driven, as once users enter each individual lab, they are responsible for the circuits that are created, and they can use any combination of materials that they choose. It is likely that many students will have circuits that consist of different pieces, or are presented in a different layout. Every action that the user makes while constructing the circuit affects the outcome and the functioning of the circuit, which can impact the grade that they receive. In the end, multiple users may end up with the same results, but they will not have necessarily taken the exact same steps, as their circuits can differ in chip placement, or the way that they went about constructing the circuit. Along with the virtual laboratory section, the user interface is also loop based - event driven. It waits for a particular action from the user, and updates the UI accordingly.

9.6.2 Time Dependency

There is a timer in a system, but it is only used as a reference for students and teachers to see how long it takes to complete a lab. A longer completion time may be an indicator that the students do not understand the material well as they should. However, the timer does not force the user to exit the program or any type of execution at any time. In fact, we allow the user to save their progress to come back to it later.

9.6.3 Concurrency

We do not have a concurrency model for this project.

9.7 Hardware Requirements

Listed below are specific requirements for our system:

- OS: Windows XP SP2+, Mac OS X 10.9+, Ubuntu 12.04+, SteamOS+
- Graphics card: DX9 (shader model 3.0) or DX11 with feature level 9.3 capabilities
- CPU: SSE2 instruction set support
- Screen: minimum resolution of 720p
- Minimum network bandwidth: 56k
- Minimum hard drive space: 100 megabytes

10. Algorithms and Data Structures

10.1 Algorithms

Many of our complex algorithms reside in getting the Virtual Laboratory section of the application working. What this means is that the logic between the circuit equipments, such as the Protoboard, Chips, Wires, and other components are computed in conjunction with each other. Each digital logic component is filled with objects that we call “Logic Nodes” positioned in critical locations in the chip, such as the output pins in the following 74LS00 Double NAND chip:

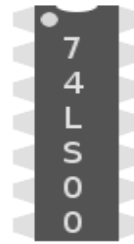


Figure 10.1: A 74LS00 NAND gate chip

The interaction between the Logic Nodes of this NAND gate (figure 10.1) and other digital circuit equipment is facilitated by the Unity Engine’s collision detection system (when one Logic Node object is positionally overlapping another Logic Node object). When the device is ‘set’, meaning the user has stopped moving the device via the mouse, the collided nodes are detected and the device is forced to evaluate the ‘Logic States’ (Logic LOW, Logic HIGH, or INVALID) of any external Logic Nodes. Based on the particular device specification (a NAND device object would follow the logic of the NAND gate configurations), the device will set its own Logic Node in reaction to the inputs. Some devices may have outputs, which will force the external collided node’s Logic State to change.

To facilitate this algorithm, we force the equipment objects to implement an interface that has the method `ReactToLogic()`. When a state in the Logic Node changes, the `ReactToLogic()` method is called for the respective devices. What this means is that we don’t have to continuously check the states of all Logic Nodes in the system, but can efficiently compute any changes for any devices that may be affected by any new collisions, or state changes.

The following is a basic breakdown of the individual algorithmic operations going on with our `GameObjects`:

Logic Nodes:

Digital Logic Design, in the most basic form, is based on two states for every logic circuit input and output, a logic high (usually 5V), and a logic low (ground). Every Digital Logic device has this “State” property on it’s input and output nodes. It was important to make a system to easily detect these, and create a reliable interaction between different nodes within the system. Using this idea, we created a `GameObject` called conceptually “Logic Node” that have several properties. We represent a Logic Node graphically with a small circle that has three different

colors: green for logic high, red for logic low, and white for neutral. These states are kept by each Logic Node as an integer that is predefined statically. Outside of testing scenarios, every single Logic Node is a child of a GameObject that implements an interface called Logic Device. We will expand upon this further down in the document.

For each Logic Node, it is incredibly important to determine if it is positionally overlapping with another Logic Node, analogous to a digital logic component connecting to another digital logic component via physical contact. This is only detected when a collider component in the shape of the GameObject's collision perimeter is added to the Logic Node GameObject on object instantiation. Whenever this overlap happens between two Logic Nodes, a collision is detected by the Unity engine, and a callback function called `OnTriggerEnter()` is called, which notifies the programmer to react to this collision. For Logic Nodes, we notify the object that a collision has been detected recently and keep note of the Logic Node that collided. The Logic Node object does not immediately react to any collisions as the user may be actively moving the Logic Node's position. Upon reaching the next Update loop, responsibility of how to change the Logic State is given to the owning device of the Logic Node.

Devices:

Every device in this system is implemented as a Device interface that implements a few functions. The most relevant one right now is the `ReactToLogic()` function. What this allows us to do is it lets every logic node access it's owning device's specific logic configuration without code duplication and let's every device handle it's logic data structures in anyway that it wants. Once the `ReactToLogic()` function is called, all the Logic Nodes that the device owns, and have two important things checked: their Logic States, and the states for any colliding Logic Nodes.

Usually, most devices handle inputs and outputs during Logic computation in the following way: The device's input logic are never set to a specific state, but rather they keep their state's neutral to ensure that their colliding nodes aren't influenced by their states. This is particularly important when a device is colliding with the Protoboard device as the logic calculation on a set of rows/columns in the Protoboard's Logic Nodes are based on a priority system. The priority system prioritizes a logic low, then a logic high, and finally a logic neutral. If an input Logic Node on a colliding device is set to low, and the Protoboard's set of Logic Nodes are requested to change to a logic of high, it will refuse to change as it detects that a colliding node has a state of low.

The device's output are always set to the state that it needs to be, to let the output Logic Node communicate to it's colliding Logic Node to request the owning device to change states. This is the pattern followed for all devices outside of special cases such as the Protoboard.

Typically, devices are movable (with exceptions), and the device's position based on the mouse position are controlled by the `OnMouseClicked()` and `OnMouseDown()` callback functions from Unity. Here, the current position of the device, and the offset from the device and the mouse position is calculated to move the device to the correct mouse position.

Protoboard -

The Protoboard acts as both an input and output device on all of its Logic Nodes. A crucial data structure for the Protoboard is the hash table, due to the way the data is structured, and the speed of the retrieval of data. As specific rows, and specific columns of Logic Nodes have the relationship of representing one Logic Nodes, they need to be represented in a way where a list of Logic Nodes is retrieved for a specific column/row request. A Hash Table is the perfect data structure for this as a key can be assigned to every set of related nodes, and a List (Array) data structure of Logic Node GameObjects can be assigned as the value for the key value pair. During the `ReactToLogic()` function, the relevant list of Logic Nodes can be received by knowing the calling Logic Node's key in a time complexity of $O(1)$. As mentioned earlier, a priority system is used to update the list of Logic Node's state as a set must all have the same state. All colliding Logic Node's with the set are checked for their Logic States, and based on a priority system, the set as a whole is assigned one logic state. The priority system assigns the logic low first, logic high second, and assigns logic neutral last. The protoboard is an immovable device as for the protoboard to be clickable, it would need to have a Box Collider component for the mouse input callbacks to be registered. However, since the Logic Nodes contained inside it also have Colliders, the Unity engine has a difficult time distinguishing which GameObject is colliding with which other GameObject. We decided to remove the movable functionality from the protoboard due to this.

Chips (74LS00 (NAND), 74LS04 (INVERTER), 74LS08 (AND), 74LS32 (OR)) -

The chips are all movable devices that contain 14 Logic Nodes. These Logic Nodes are stored in Hash Tables, but the implementation can easily be changed to Lists as well. To function correctly, they must be 'snapped' to 14 other nodes, typically this means a collision between all of the chip's Logic Nodes and the Protoboard's Logic Nodes are detected simultaneously. Once the chip detects that all 14 nodes are collided with, and the user lifts the mouse, the `OnMouseUp()` callback is recorded, and the position of the chip is snapped to the top left Logic Node's collided Logic Node's position (arbitrarily chosen), a green indicator is shown to show potential snappings. Once the device is snapped, before any logic calculation is done, the chip must detect a collided node on both the 7th pin, and the 14th pin, with a logic low and a logic high going to the respective nodes. After that, based on the datasheet, the collided input Logic Node's states are taken, and the output is set.

LED -

The LED is an important movable device, and similar to the 'chips', they can be snapped. The LED takes in two inputs by detecting collision on both of its 'legs'. If the shorter leg's collided Logic Node has a state of logic low and the longer leg's collided Logic Node has a state of logic high, then the LED has a state of being "On". This also means that the sprite of the LED is modified to show that it is emitting a light source. In every other situation, the LED is in the state of being "Off" and has a sprite that reflects that. The importance of the LED doesn't only come from being a good debugging device for the user, but also is important in a technical aspect as it is used to check the input and output states of the overall circuit of the lab. This will be further expanded on.

Switch -

The switch is another important movable device, similar to the LED. It contains three Logic Nodes, two of which are inputs, and one of which is an output. It can be toggled up or down, and will prioritize the output to reflect the top, or bottom Logic Node input. Similar to the LED, they play an important role when analyzing the built in circuit as they can be designated as an overall input to the system.

Wires -

Wires are generated using the 'w' key on the keyboard, or can be accessed via the dropdown menu. After initializing them, a click on a Logic Node is listened for on a callback, after the first click, the Wire 'Line' is rendered to follow the mouse from the specified Logic Node. Every click after ward creates an inflection point for the wire, if the clicked point is not a Logic Node, in which another 'Line' is rendered from that point to the mouse. Only when a Logic Node is clicked is the wire sequence is done executing within the Wire object's Update function, and two new Logic Nodes are created at each end of the wire. The wire follows a similar priority system to that of the Protoboard by analyzing the colliding Logic Nodes of both ends of the wire.

Magnifying Glass-

The magnifying glass allows the user to access information about components that may be useful while conducting the lab, such as datasheets, instructions, and animations. This information can be uncovered by hovering the magnifying glass over the object of interest (Logic Chips, LED, Switch, Wires, etc). When this occurs, a collision is detected between the magnifying glass and the object of interest, which triggers a function in the *MagnifierBehavior* script. This function passes a collision parameter which provides information on the collision, allowing the corresponding information to be displayed. When the magnifying glass exits a collision with an object, a different function in the *MagnifierBehavior* script executes, which in turn hides the corresponding information.

Trash -

Since an unlimited number of components can be generated by the user, there must be a system of deletion in order to prevent unnecessary pile up of unwanted components. The trash feature allows the user to delete components by dragging and dropping the component to be deleted over the trash icon. Once a component is hovering over the trash icon, the resulting collision triggers a function that alerts the user by allowing the trash icon to glow. As the trash icon is glowing, an update loop checks if the user has let go of the left mouse button. If so, then the component is destroyed and memory is freed.

Power Supply -

The power supply device is unique as it does nothing with the ReactToLogic() interface even though it implements it. This device's only job is to continually set it's Logic Node's to logic high, and logic low through the Update loop. This way, even if there is a mechanism that changes it's states (it will be discussed further down), the Power Supply will force it's logic state to its proper value and permeate the value through the collision system through the rest of the circuit.

Prelab / Postlab -

For the prelabs and postlabs, the user is asked to enter inputs in the respective fields of the k-map. Once those inputs are entered, the user hits the “Check” button. For the text fields, the program restricts inputs to single digit integers. This minimizes any issues that could be caused by inappropriate spacing or text input in the fields. When the user hits the “Check” button, the system checks the user’s inputs, which are all stored in variables, compared to the answers that should have been inputted. If the answers inputted and the correct answers are the same, a message appears saying and it goes onto the next lab within 5 seconds. If the user inputs an incorrect answer, a message will display to the user informing them to try again.

10.2 Data Structures

A lot of the discussion on the specific data structures we used are stated in the above discussion (section 10.1).

Our application makes use of many data structures, ranging from Hash Tables (called Dictionaries in C#), and Lists. Many of our data structures are implemented in the Device objects that hold Logic Nodes. The data structures are chosen based on the quantity of information needed to store, and efficient method of getting such objects. Digital Logic devices such as the Protoboard requires hundreds Logic Nodes to serve its purpose whilst devices such as the 74LS00 (NAND Chip) require exactly 14.

For cases such as the protoboard, we decided to store all the nodes in a Hash Table with a designated key. The rows of the Protoboard all represent the same logic state, as do the columns of the far left and far right of the protoboard. We can store key-value pair based on Lists of Logic Nodes that have the same Logic ID (manually assigned based on the row and or column). In addition to the numerous number of Logic Nodes, the Get operation on the Hash Table usually has a complexity of $O(1)$ which allowed us to quickly retrieve any set of Logic Nodes incredibly fast.

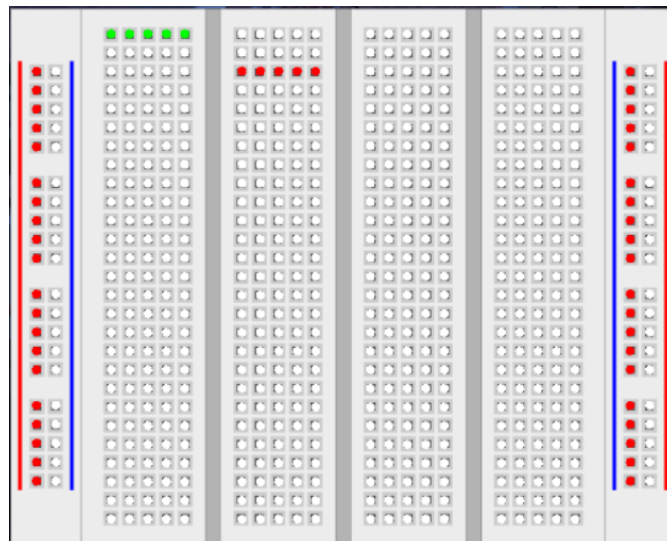


Figure 10.2: Our protoboard image with visible nodes. Green represents logic high. Red represents logic low. White indicates that it is currently neither high nor low.

11. User Interface Design and Implementation

11.1 Login Page

Our project uses Amazon AWS service as an database that can communicate with our clients. PHPMyAdmin was used for the database setup. A table which can handle all the information of our users was made. It has the following attributes:

uid: int (as the primary key)
username: varchar(20)
Password: varchar(20)

Since all we need is a simple DLD lab system. We assume professors have a maximum number of 10 (uid 1 to uid 10). Their credentials are previously set into the database. We also have students whose uids go from 11 to 999. Students are not previously built in our database but it can be created by the professor. The student's uid increments as new user is created

Enter the wrong credentials will prompt a wrong credentials error. Enter the right credentials of professors, the system will redirect the user to admin subsystem. Enter the right credentials of students, the system will redirect the user to student subsystem.

11.2 Admin Subsystem

After entering the admin subsystem, the client make a inquiry to the database server for all users' uids and usernames. All information will be shown as a table, with a delete icon on the right hand side. The interface has several buttons: create user, delete user. Click create user will prompt a new window to let the user enter the credentials of the new user. After that, a new student user is created and added to the end of the table. Click delete user icon will prompt a new window to let the professor confirm if he/she really wants to do that. Please note that Professors cannot delete other professors.

11.3 Changes from previous UI design

Our completed user interfaces may be viewed in section 3.3 of this report, beginning on page 14. There, we show pictures of each of the interfaces, and we break down exactly what each button does and how the user can interact with them. Below, we will show the evolution of these UIs, beginning from the initial drawings, then progressing from the initial drawing (Figure 11.3.1), to the computer-aided design (figure 11.3.2) to the final product (figure 11.3.3).

Lab 2 Combinational SSI Circuits

Given: $F(A, B, C, D)$
 $= \sum_{ABCD} (1, 3, 5, 6, 7, 14)$

Please complete the K-map given:

AB \ CD	00	01	11	10
00				
01				
11				
10				

RESET SUBMIT

Lab 2

AB \ CD	00	01	11	10
00				
01				
11				
10				

Your K-map is correct!

Please generate the minimal sum-of-product expression:

$F(A, B, C, D) =$ _____

SUBMIT

Lab 2

Given the logic diagram, put corresponding components on the board

```

graph LR
    D --- AND1[AND]
    A -- NOT --> AND1
    B -- NOT --> AND1
    AND1 -- OR --> AND2[AND]
    C -- NOT --> AND2
    AND2 -- OR --> F[F(A, B, C, D)]
  
```

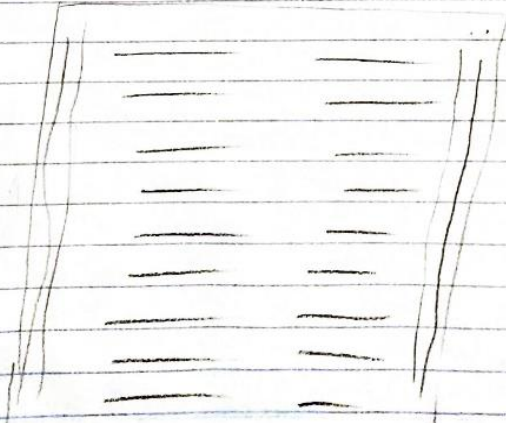
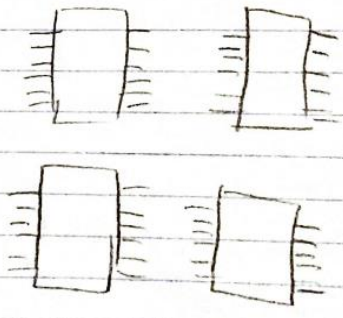



Figure 11.3.1: Initial UI drawing for Lab 2, along with prelab and postlab

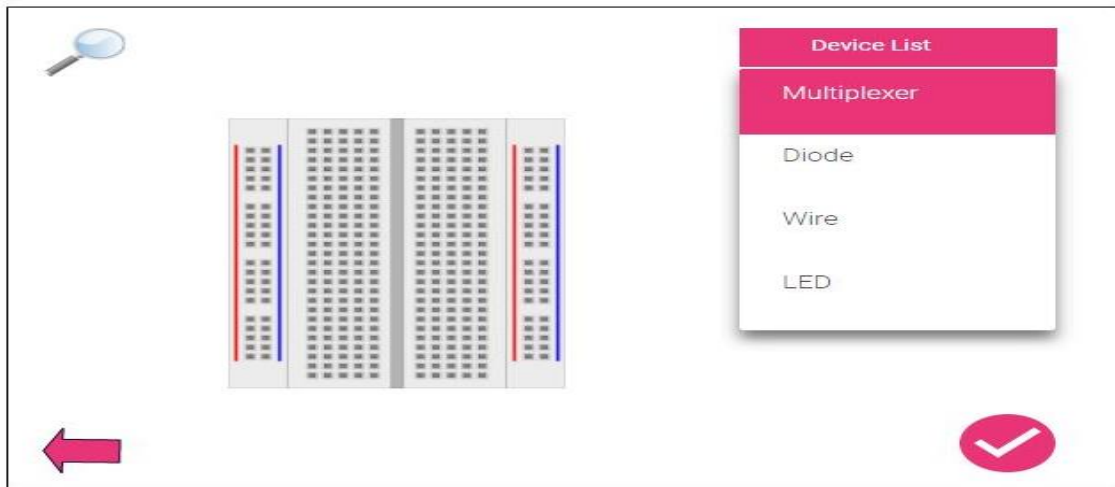


Figure 11.3.2: Computer-aided Design of Lab UI

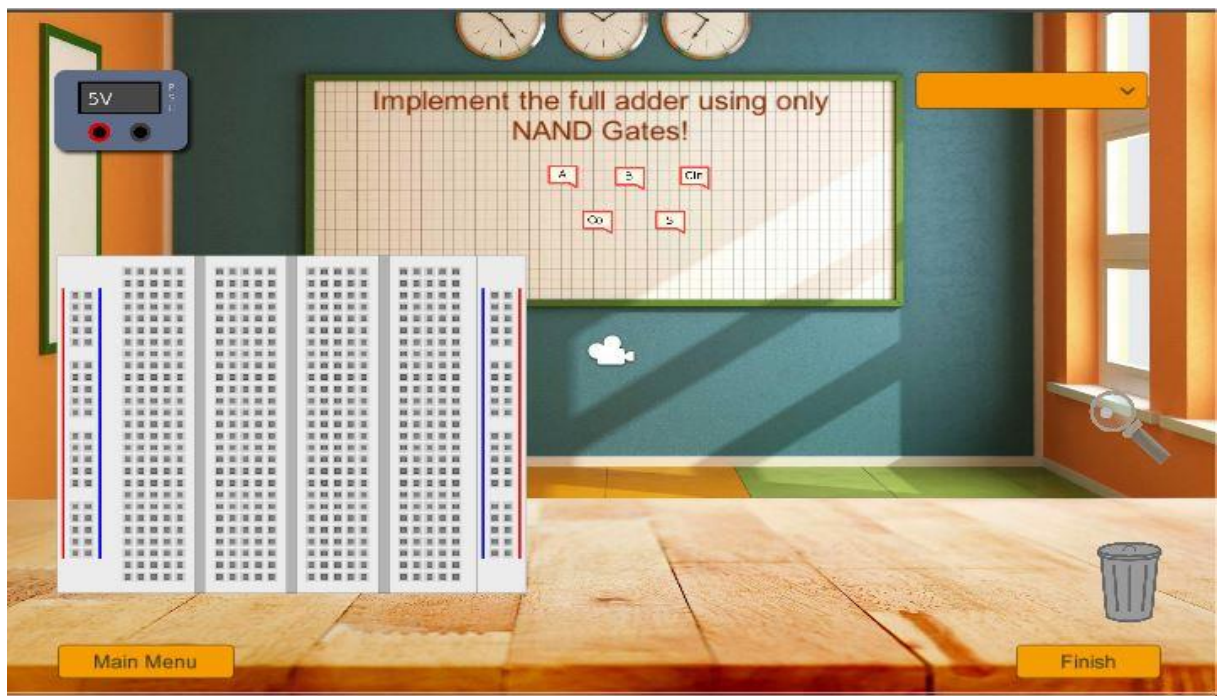


Figure 11.3.3: Finalized Lab UI

As is evident from the development of the three figures above, the UI design changed a lot during the project, as new needs arose, and ease-of-use became the forefront of our design. We feel that this development kept the interface simple, but also allowed a wide range of tools that the user may need at any point.

12. Design of Tests

12.1 Test Cases

Because our system is an interactive one, many of our tests will involve working with the circuits and GUIs, making sure that everything executes as expected. The most basic unit tests that we will complete are tests of the equipment, making sure that each piece looks like it should, reacts to clicking and dragging properly, connects to the protoboard, and functions the way that it should. We will also need to complete tests to see that the system correctly creates and deletes users, and allows them to log in to their accounts. These are the most basic functions of our system, so these are the things that we must test rigorously in order to ensure that there are no faults with them. Once our basic components are properly implemented and tested, we can begin to construct and test the more complex functions of our system. The labs will require a large amount of testing as well, as we will need to make sure that the system reacts properly to every circuit that is submitted.

Element testing:

The boxes below contain the element testing to be done with our lab. Each box has an action, then the successful and unsuccessful outputs that may result from completing that action. All of these tests will be used when debugging our code and making sure everything run properly.

Action	Output
Clicks and drags circuit element	<u>SUCCESS:</u> Circuit element moves along with user's click <u>FAILURE:</u> Circuit element does not move or moves independently of user's click
Places circuit element on protoboard	<u>SUCCESS:</u> Element connects with protoboard, and collision of nodes is detected <u>FAILURE:</u> Element will not attach to protoboard, or collision of nodes is not detected, impairing functioning of the circuit
Test circuit interactions by connecting power source, chips, and wires	<u>SUCCESS:</u> Logic "HI" and "LOW" appear where expected, based on circuit configuration <u>FAILURE:</u> Logic "HI" and "LOW" do not appear where expected, based on circuit configuration

Lab Testing

Action	Output
User completes lab according to the desired requirements and clicks the "Finish" button	<p><u>SUCCESS:</u> The system analyzes the user's circuit and determines if the correct output is received</p> <p><u>FAILURE:</u> The system fails to, or incorrectly analyzes the user's circuit</p>

Quiz Testing

Action	Output
User completes given quiz and clicks the "Finish" button	<p><u>SUCCESS:</u> The system correctly grades the user's quiz by comparing his/her answers to the correct version</p> <p><u>FAILURE:</u> The system incorrectly grades the user's quiz, mistaking wrong answers for correct ones, or vice versa</p>

UI Testing

Action	Output
The user clicks on the button to take them to the desired lab	<p><u>SUCCESS:</u> The user is taken to the lab, where they are given instructions and allowed to complete the tasks</p> <p><u>FAILURE:</u> The system does not respond to the click, or takes the user to the incorrect place</p>
The user clicks on a button to view grades	<p><u>SUCCESS:</u> The user is given access to view grades received on previous labs and quizzes</p> <p><u>FAILURE:</u> The user is unable to view previous grades, or the grades that they are able to view are incorrect</p>

User Management/Login Testing

Action	Output
User is created	<p><u>SUCCESS:</u> User is created and entered into the database, along with password</p> <p><u>FAILURE:</u> User is not created successfully, or is not associated with the correct password</p>
User enters username/password to log	<p><u>SUCCESS:</u> The user is admitted into the system after entering correct credentials, or denied access if</p>

in	incorrect <u>FAILURE:</u> The user is denied access after entering the correct credentials, or admitted access after entering incorrectly
----	--

12.2 Test Coverage of Test Cases

The system testing is a very crucial part of our project, as it serves the purpose of letting us know when a particular piece of code is not acting as expected, or where we may have made faults in design or implementation. Through the use of a wide range of testing, we can ensure that each and every fault is met early on and corrected, and that every invalid input given by a user is handled efficiently by the system, rather than having it react in an unpredicted manner. Our test coverage must not just include inputting the correct answers, but also requires that we input every combination of invalid and incorrect inputs possible, to ensure that the system can handle them and react appropriately.

12.3 Integration Testing Strategy

Our integration strategy will be top-down approach. Because our project is so large and consists of so many different sections, there will often be pieces that are ready for integration, while others are still being modified. Using a top-down approach will allow us to test the interactions of the specific sections that have been integrated, and this will allow us to find faults in the communication between classes, which is heavily relied upon. Errors in communication between classes would not be as simple to see if we were taking an approach that focused more on smaller, individual sections of the the system.

An example of the top-down testing method that we use is when we test the equipment. The protoboard is an example of an upper integrated module, and once that is integrated, we can test the branches (in this case, the wires and chips) step by step, until they too are integrated. After we verify that the protoboard is functioning properly, we test the other objects (the branches in this case) to see if they interact correctly with the protoboard (the main module). Once this has been deemed correct and properly integrated, we can test the functioning of even smaller modules that branch from the chips, such as whether they output the correct logic states in certain scenarios.

13. History of Work

January 25-28: Finalized project focus, developed project proposal

February 1-4: Developed statement of work and requirements

February 5-11: Developed functional requirement specification, and developed desired UI layouts

February 12-18: Compiled report 1; decided on a platform to develop the application; began collecting images to be used to represent objects in the application

February 19-25: Developed interaction diagrams; began coding; began development of interaction between objects and nodes in the Unity interface

February 26 - March 4: Designed class diagrams and system architecture; began development of user interfaces; wrote scripts for circuit elements (chips, wires, power source) that cause them to behave as they would in a normal lab setting

March 5-11: Compiled report 2; began development of each individual lab; added magnifying glass and trash bin features, as well as drop-down parts menu

March 12-19: Further development of labs and UI; setup of database; login functionality and ability to create users are added; began unit testing

March 19-23: Further development of labs and UI; added sandbox mode; began integration testing; completed user documentation; completed technical documentation

March 23-28: Preparation for demo 1; decided which parts to present and who would present them; designed brochure; designed presentation slides

March 29: Demo 1

March 30-April 8: began work on user chat; began development of CheatDetector; completed implementation of the labs; began working on save state functionality

April 8-22: Continued development of above features; completed required sections of report 3

April 22-30: Completion of report 3; finalized work with CheatDetector and Chat system

May 2: Demo 2

14. References

1. Professor Marsic's website, detailing the requirements for report 2
<http://www.ece.rutgers.edu/~marsic/Teaching/SE/report2.html>
2. A past project of the same topic, that was used as a guide
<http://www.ece.rutgers.edu/~marsic/Teaching/SE/report2.html>
3. Used to detail the interaction diagrams:
<https://medium.com/omarelgabrys-blog/object-oriented-analysis-and-design-design-principles-part-6-b78e2b9da023>
4. A good overview of top-down testing
<http://extremesoftwaretesting.com/Techniques/TopDownTesting.html>
5. A site that was used to develop an understanding of Unity
<https://unity3d.com/learn>
6. Used to gather information about the SN74LS00 chip that we will be simulating
<http://www.ti.com/product/SN74LS00/technicaldocuments>
7. Provided information about several gates that will be used in our simulations
<http://www.futurlec.com/74LS/74LS04.shtml>
<http://www.futurlec.com/74LS/74LS08.shtml>
<http://www.futurlec.com/74LS/74LS32.shtml>
8. Provided information about data structures that will be used in this project
<https://www.geeksforgeeks.org/data-structures/>