Winter Internship 2024 Report

On

UI Improvements, Bug Fixes and LTI Auto-grading Improvement

Submitted by

Kaung Hsu San
Myanmar Institute of Information Technology
Mandalay, Myanmar

Under the guidance of

Prof. Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

Mentors:

Mr. Nagesh Karmali
Ms. Firuza Aibara

March, 2025

Acknowledgements

I, the summer intern of the **FOSSEE - Arduino On Cloud project**, am deeply humbled and overwhelmed with gratitude as I acknowledge all those who have helped me perfect my ideas and have assigned tasks that challenged me, pushing me beyond simplicity into creating something concrete and unique.

I wholeheartedly thank **Prof. Kannan M. Moudgalya** for having faith in me, selecting me to be a part of his valuable project, and for constantly motivating me to strive for excellence. I also thank **Mr. Nagesh Karmali** and **Ms. Firuza Aibara** for providing me with the opportunity to work on this project.

I am immensely grateful to my mentors for their valuable guidance and suggestions. They were always there to show me the right path when I needed help, and with their brilliant encouragement and direction, I was able to complete all tasks efficiently and meet the expectations set.

Throughout this experience, I had the opportunity to strengthen both my technical and non-technical skills and deepen my understanding of key concepts.

Last but not least, I sincerely thank all my colleagues working on different projects under **Prof. Kannan M. Moudgalya**. Their critical advice and support have played a significant role in helping me grow and evolve during this period.

Contents

List of Figures

Chapter 1

Introduction

Arduino on Cloud is an cloud-based simulator designed to provide students and researchers with a platform to test and run simulations before executing their projects in real-world scenarios. This system simplifies the process of virtual experimentation by allowing users to drag and drop various Arduino components from the left panel into the working space on the right.

The system provides users with the ability to connect the Arduino board's pins to a range of input/output devices such as LEDs, motors, and push-buttons, using virtual wires. To enhance clarity and ease of use, users can customize the color of wires, LEDs, and other components, helping to visually differentiate between various elements in the simulation.

Once the hardware setup is complete, users can write their code in the code window and simulates. For documentation purposes, users have the option to print or save their simulation in PDF format. Additionally, the platform offers an Electronic Rule Check (ERC) feature, which enables users to quickly identify and correct any errors in their circuit design, ensuring a seamless and error-free simulation experience.

1.1 Project Overview

During this internship, I worked on an existing project that had already been partially completed. My main tasks included bug fixes, addition of new features and improvement of the existing functionalities, all aimed at enhancing the overall performance and user experience of the simulator.

Chapter 2

Photoresistor's Moon Movement Sync

The task deals UI bug in the photoresistor component, where the moon was not aligning properly with the slider and the sun when simulating. When moving the photoresistor, the sun and the slider move along with the photoresistor but the moon move very little.

2.1 Root Cause Analysis

The issue stemmed from the moon SVG being much larger than the sun SVG, which led to incorrect scaling and translation. In SVG (Scalable Vector Graphics), scaling an element changes both its size and the impact of any translation (movement). When an object is larger, translating it by a fixed amount causes it to appear to move a smaller distance relative to its size, when combined with scaling. Since the moon SVG was bigger, applying the scaling and translation caused the moon to move less than expected, resulting in unsynchronized movement between the moon, sun, and slider.

2.2 Implementation

To fix this, I first scaled down the moon SVG directly in the `ArduinoFrontend/src/assets/jsons/PhotoResistor.json` file. This made the moon's dimensions match the sun's proportions. Next, in `ArduinoFrontend/src/app/Libs/inputs/PhotoResistor.ts` I adjusted the translation formula so that it is on the left side of the slider. This ensured that the moon moved correctly along with the slider and the sun.For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/558`

2.3 Result

The following figures illustrate the change before and after the fix.

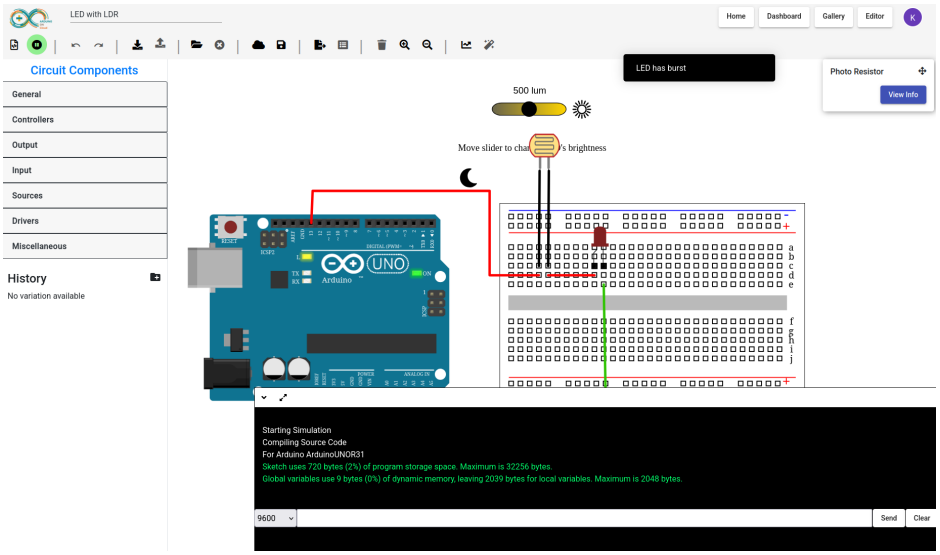

Figure 2.1: Before the fix: The moon's movement was unsynchronized with the sun and slider.

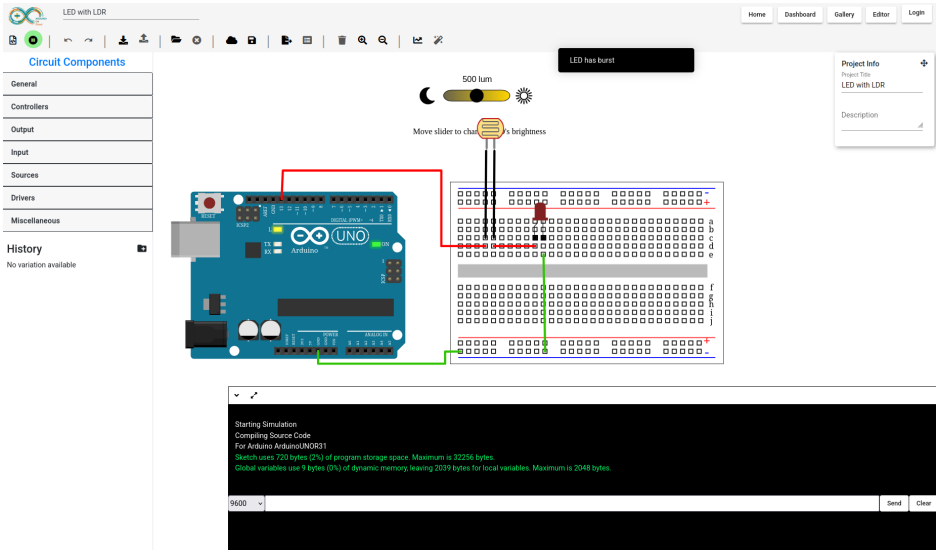

Figure 2.2: After the fix: The moon now moves in sync with the sun and slider.

Chapter 3

Potentiometer Dial Misplacement

This task addresses a UI bug in the potentiometer component, where the dial becomes misplaced when stopping the simulation.

3.1 Root Cause Analysis

Previously, the potentiometer's transformation (including its position) was applied using only the tx and ty translation values. However, the rotation of the potentiometer dial was not saved or restored correctly between simulation runs. This caused issues where the potentiometer dial either didn't reflect the last known rotation or would reset to an incorrect state upon re-initialization. This lack of proper handling of the transformation state led to inconsistencies in the potentiometer's behavior, causing it to appear misaligned when the simulation was restarted.

3.2 Implementation

To resolve this, the implementation was updated to restore the potentiometer's rotation angle (if previously saved) and ensure the potentiometer's position is correctly set when the simulation begins. The rotation angle, if available, is now retrieved and applied at the start of the simulation, ensuring that the dial reflects its last known state. Additionally, the translation values (tx, ty) are properly applied, ensuring that the potentiometer's position remains consistent after the simulation is restarted.

This fix ensures that the potentiometer's dial no longer resets to an incorrect

position, and the transformation (both position and rotation) is preserved correctly across simulation runs. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/561`

3.3 Result

The following figures illustrate the change before and after the fix.

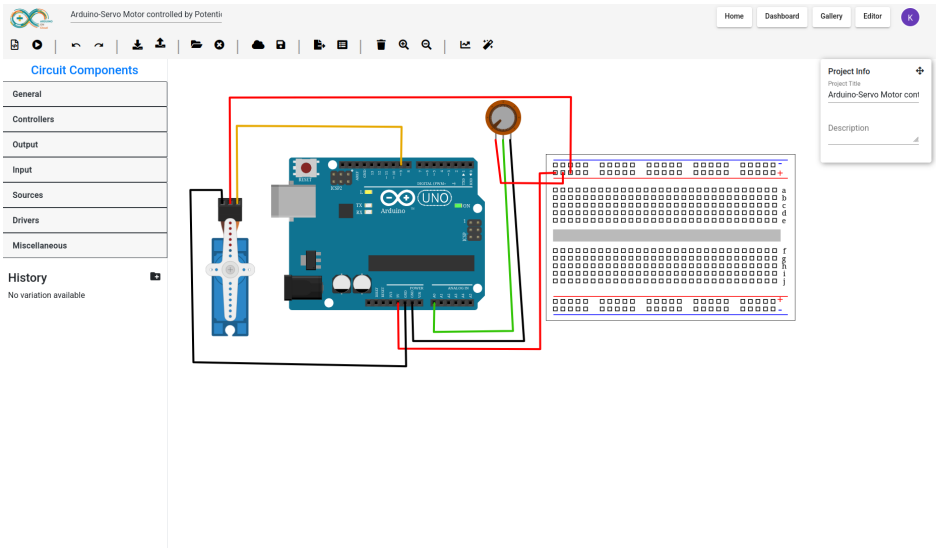

Figure 3.1: Before the fix: The potentiometer dial was reset to original after stopping the simulation.

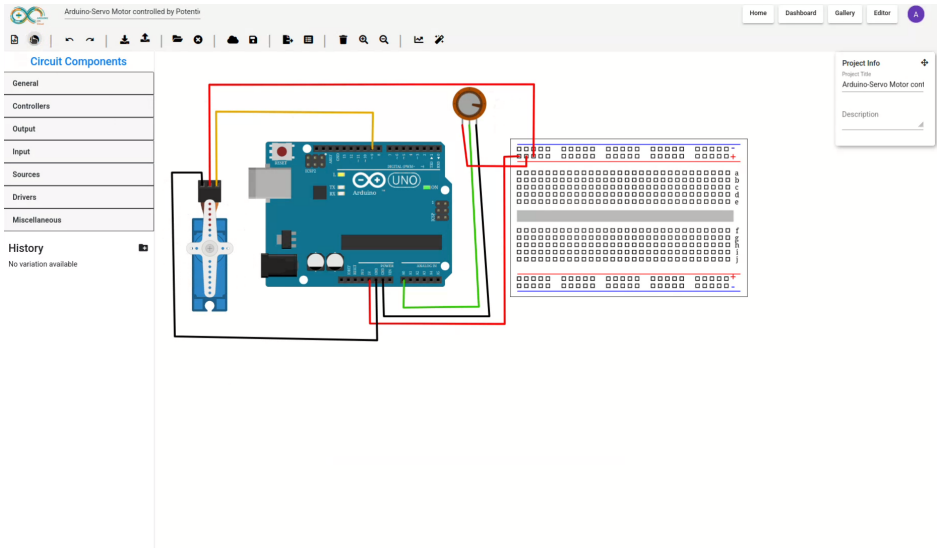

Figure 3.2: After the fix: The potentiometer dial now maintains its correct position and rotation across simulation restarts.

Chapter 4

Slide Switch Dragging Issue

This task addresses a UI bug in the Slide Switch component, where dragging the switch after closing the simulation caused the component to become misaligned or behave unexpectedly.

4.1 Root Cause Analysis

Previously, the Slide Switch's transformation (position updates during dragging) was not being applied correctly. The issue stemmed from the fact that the drag listeners were not being initialized properly for the component. This caused problems where:

- The Slide Switch did not move smoothly as expected.
- The connections between the switch and other circuit elements appeared visually detached.

The root cause was that the setDragListeners() function, which manages drag behavior, was not explicitly called in the constructor of the Slide Switch component. As a result, the necessary event listeners were missing, preventing the component from updating its position correctly.

4.2 Implementation

To resolve this issue, the setDragListeners() function was explicitly called in the constructor of the Slide Switch component. This ensures that drag event listeners are properly attached when the component is created. Now, when the switch is dragged, both its position (tx, ty) and the connections to other elements update correctly.

This fix ensures that the Slide Switch behaves as expected when moved, maintaining alignment with circuit connections and preserving its position accurately. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/562`

4.3 Result

The following figures illustrate the change before and after the fix.

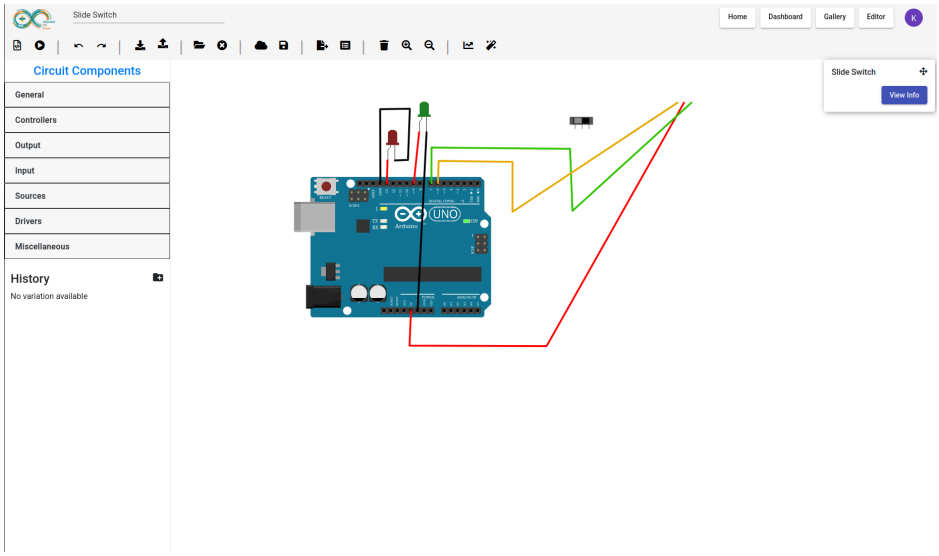

Figure 4.1: Before the fix: The Slide Switch did not move correctly, causing misalignment.

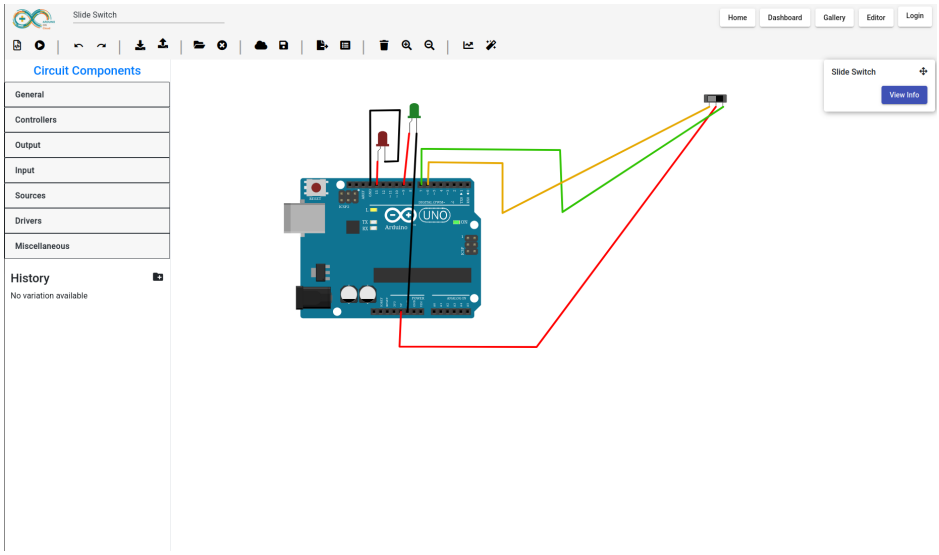

Figure 4.2: After the fix: The Slide Switch now moves correctly, maintaining alignment.

Chapter 5

Navigation and Save Prompts

Two related issues were identified regarding the save functionality when navigating between sections such as "Home," "Gallery," and "Dashboard":

1. **Unnecessary Save Prompt:** A pop-up appears asking the user to save the circuit when navigating from the editor to sections like "Home" or "Gallery," even if no changes have been made.

2. **Save Not Triggering on Home Button:** When the user clicks "Save" in the pop-up after navigating to "Home" from the toolbar, the circuit is not saved. It only saves when navigating to other sections like "Dashboard" or "Gallery," creating inconsistency and confusion.

5.1 Root Cause Analysis

The first issue stemmed from the lack of a flag to track unsaved changes, causing unnecessary save prompts. The second issue arose when the "Save" button was clicked in the pop-up after navigating to "Home." Despite confirming the save, the "before unload" alert still appeared, suggesting unsaved changes. However, the circuit was actually saved, indicating that the "before unload" prompt should only appear when there are unsaved changes. Both issues were traced to the system's failure to properly track saved and unsaved changes.

5.2 Implementation

To address these issues, the `hasUnsavedChanges` flag was introduced. It is used in `ArduinoFrontend/src/app/Libs/Workspace.ts` and `ArduinoFrontend/`

`src/app/simulator/simulator.component.ts` to determine when the "Save" prompt should appear and ensure the "before unload" prompt only triggers when there are actual unsaved changes.For more details, see the GitHub pull requests: `https://github.com/frg-fossee/eSim-Cloud/pull/549https://github.com/frg-fossee/eSim-Cloud/pull/550`

5.3 Result

With the changes implemented, the save prompts now behave as expected:

- The "Save" prompt only appears when there are unsaved changes.
- The "before unload" prompt triggers only when there are unsaved changes, preventing unnecessary alerts.
- The circuit is consistently saved across all sections, including when navigating to "Home."

Chapter 6

Toast Message Duration

This task addresses an issue where the toast message in the application remained visible for a fixed duration of 10 seconds, even if the conditions to hide it were met before the timer expired. Specifically, when the simulation was stopped or other actions were taken, the toast message still stayed visible until the 10-second duration had passed, which led to undesirable behavior. The goal was to allow the toast message to disappear based on specific conditions rather than being hardcoded to stay for a set time.

6.1 Root Cause Analysis

The issue stemmed from the toast message being hardcoded to stay on for a fixed duration of 10 seconds. Even when the conditions to hide the message were met, the timer for hiding the toast did not consider whether the message should disappear early. As a result, the toast message would remain visible until the 10-second timer expired, even when it no longer needed to be displayed.

6.2 Implementation

To resolve this, I implemented a flag to track the state of the toast message. I also added a `hideToast()` method to explicitly remove the toast message when required, instead of relying on a fixed timer. The flag tracks whether the toast message should be hidden based on user interactions or other conditions. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/554`

6.3 Result

The update introduces a dynamic toast message duration, allowing the message to disappear based on specific conditions rather than staying fixed for 10 seconds.

Chapter 7

Undo/Redo Stack Management

When a circuit is loaded from the gallery or imported via JSON, the undo operation should be disabled immediately without the user making any changes. The undo stack is to be cleared upon loading or importing the circuit. I implemented this behavior by introducing a resetStack() function, which is called when a circuit is loaded or imported, ensuring the undo and redo stacks are cleared.

7.1 Implementation

To implement the reset of the undo/redo stack, the `resetStack()` function was introduced in `ArduinoFrontend/src/app/Libs/UndoUtils.ts`. This function is called in `ArduinoFrontend/src/app/Libs/Workspace.ts` in the methods `static LoadWires()` and `Load()`. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/546`

7.2 Result

This update disables the undo operation after loading a circuit from the gallery or importing from a JSON file. The undo operation is only enabled once the user makes new changes. The following figure illustrates the state of the system before and after the `resetStack()` function is called.

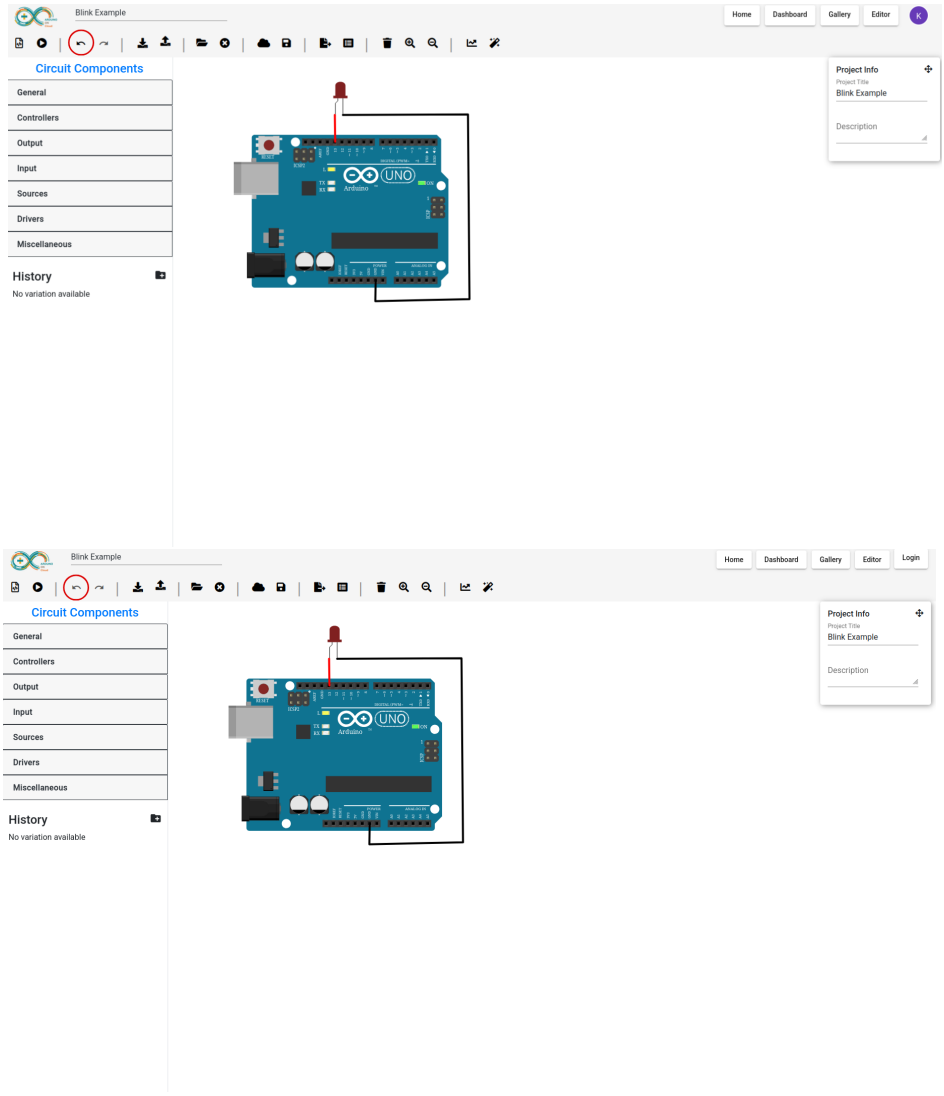

Figure 7.1: Before and after adding the `resetStack()` function.

Chapter 8

Resizable Code Window

The task was to make the code editor window resizable horizontally. Previously, the size was fixed.

8.1 Implementation

To begin, I introduced a resize handle within the editor's container and implemented event listeners (mousedown, mousemove, mouseup) to track mouse movements for resizing. To ensure the Monaco editor inside the container resizes correctly, I added an `ngOnChanges` lifecycle method in the CodeEditor component. This method monitors changes to the width input property and dynamically adjusts the editor's height and layout using Monaco's `layout()` method. Additionally, I implemented limits on the maximum and minimum size of the code editor window. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/547`

8.2 Result

The update introduces a resizable code editor, allowing users to resize the editor container using the resize handle on the right side of the editor window. The following figures illustrate the behavior of the editor before and after the resizing functionality was added:

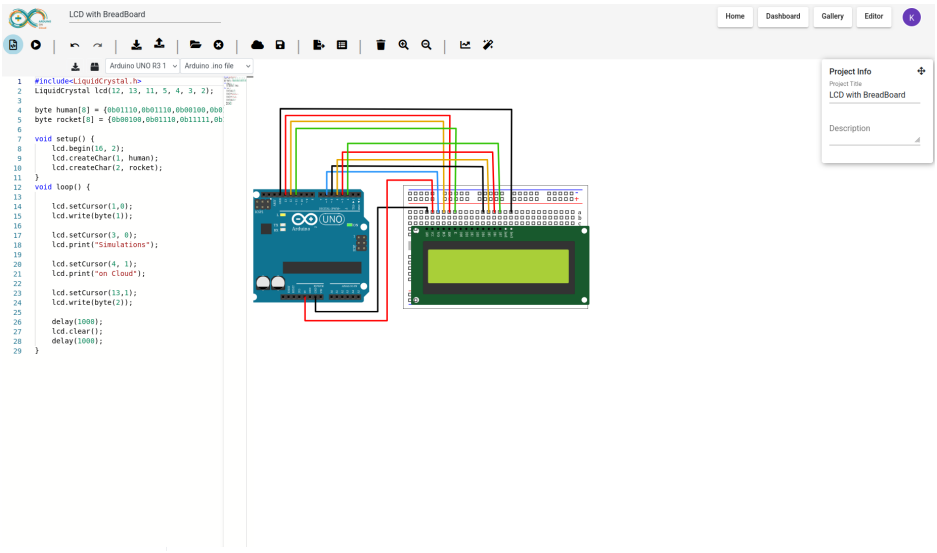

Figure 8.1: Before adding the resize functionality.

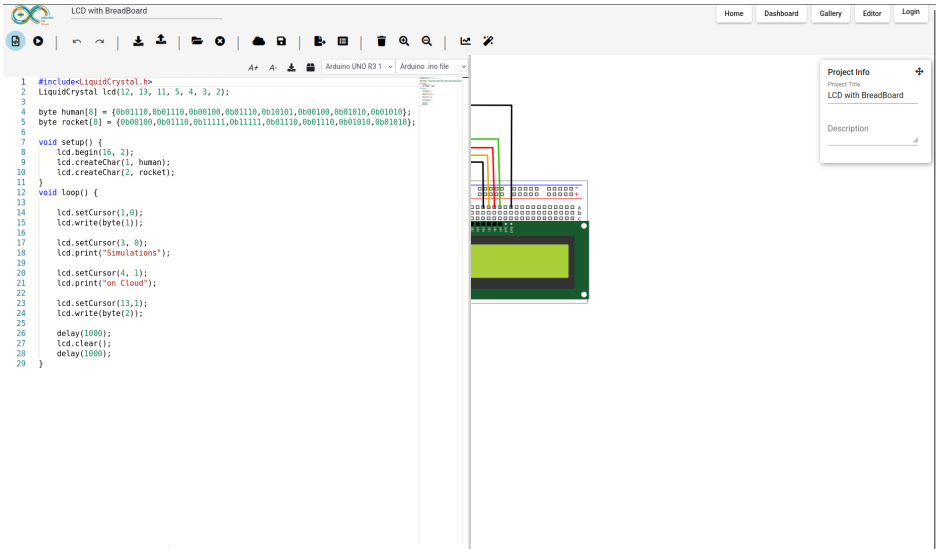

Figure 8.2: After adding the resize functionality

Chapter 9

LTI Autograding Improvement

I addressed an issue where the system was grading preconnected pins provided by the teacher, which we wanted to exclude from the grading process. The goal was to improve the LTI autograding system to grade only the pins connected by the student. Originally, the system simply compared the teacher's pins and the student's pins, regardless of any preconnected pins assigned to the student.

9.1 Implementation

To achieve this, I retrieved the pins originally assigned to the student and identified those that were not part of the teacher's circuit. By comparing the student's connected pins with the teacher's circuit, the system now calculates how many pins are correctly connected. This was implemented in the `arduino_eval` function of the `esim-cloud-backend/ltiAPI/process_submission.py` file. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/555`

9.2 Result

The update ensures that the LTI autograding system now grades only the pins connected by the student, excluding any preconnected pins provided by the teacher. This change enables a more accurate grading process by focusing on the student's work and eliminating unnecessary grading of teacher-supplied connections.

9.3 Further Extension

Currently, there is no function for saving the circuit in the LTI system. As a result, if the circuit is accidentally reset or if the student loses their progress, they would need to start from scratch. Adding a save functionality would ensure that the student's progress is preserved even if the circuit is reset or if they need to return to the circuit later.

Chapter 10

Simulation with No Code Written

When the user attempts to simulate a circuit without writing any code in the code window, a popup should appear stating that no code has been written. Additionally, no call should be made to the backend for compilation in this case. This functionality is handled entirely in the frontend.

10.1 Implementation

To implement this behavior, in the texttttStartSimulation() function of `ArduinoFrontend/src/app/simulator/simulator.component.ts`, a check is added to verify if code has been written. If no code is found, a popup appears, the simulation is halted, and the button is re-enabled. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/551`

10.2 Result

The update ensures that when the simulation button is clicked, it first checks for code in the code window. If no code is found, a popup notifies the user that no code has been written, the simulation is halted, and the simulation button is re-enabled. The following figures illustrates the behavior when no code is present and the popup appears.

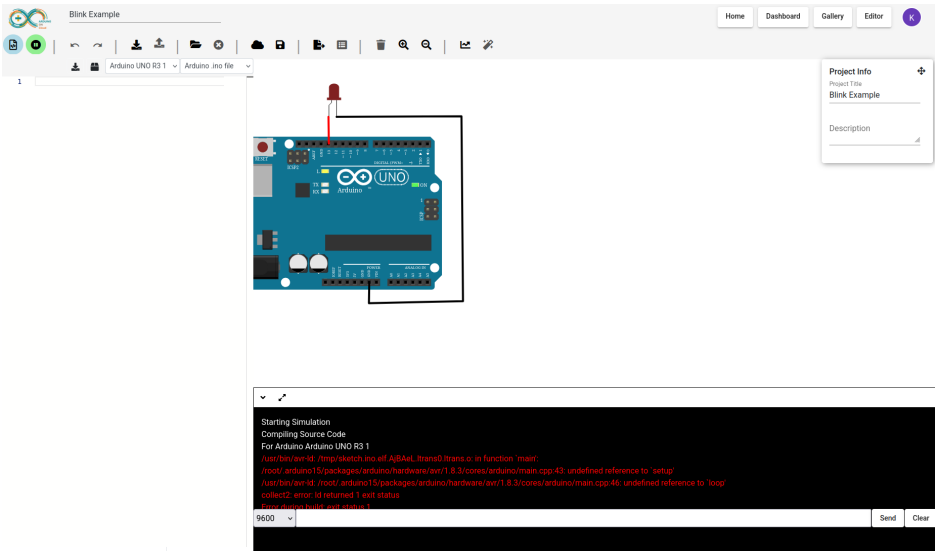

Figure 10.1: Before adding the check in `StartSimulation()` function

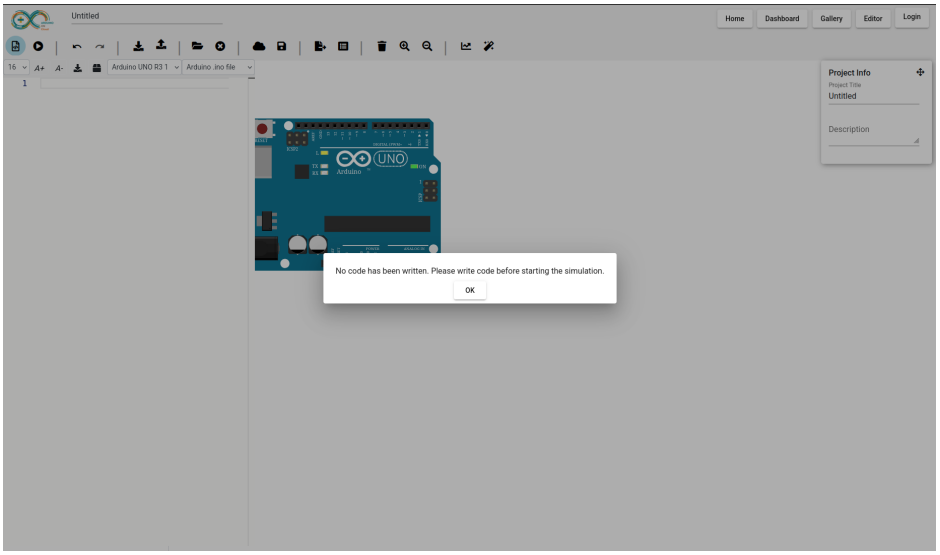

Figure 10.2: After adding the check in `StartSimulation()` function

Chapter 11

Font Size Dropdown List for Code Editor

Originally, there was a feature for changing the font size in the code editor, but it was button-based, requiring multiple clicks to reach the desired size. This approach was inconvenient for users as it was time-consuming and not user-friendly.

11.1 Implementation

To improve usability, I added a dropdown list to complement the existing button-based system for changing font sizes. In the `ArduinoFrontend/src/app/code-editor/code-editor.component.ts` file, I created a `fontSizes` array and implemented the `updateFontSize()` method to update the editor's font size when the user selects a new value. In the `ArduinoFrontend/src/app/code-editor/code-editor.component.html` file, I integrated the dropdown list with two-way data binding using `[(ngModel)]` to automatically update the `editorFontSize` variable. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/552`.

11.2 Result

Users can now directly select a font size from the dropdown, which dynamically updates the editor while staying synchronized with the existing buttons. This improvement offers a more efficient and flexible way to adjust font size, eliminating unnecessary clicks. Users can still adjust the font size gradually using the buttons, offering greater flexibility.The following images show the code editor before and after the addition of the font size dropdown feature:

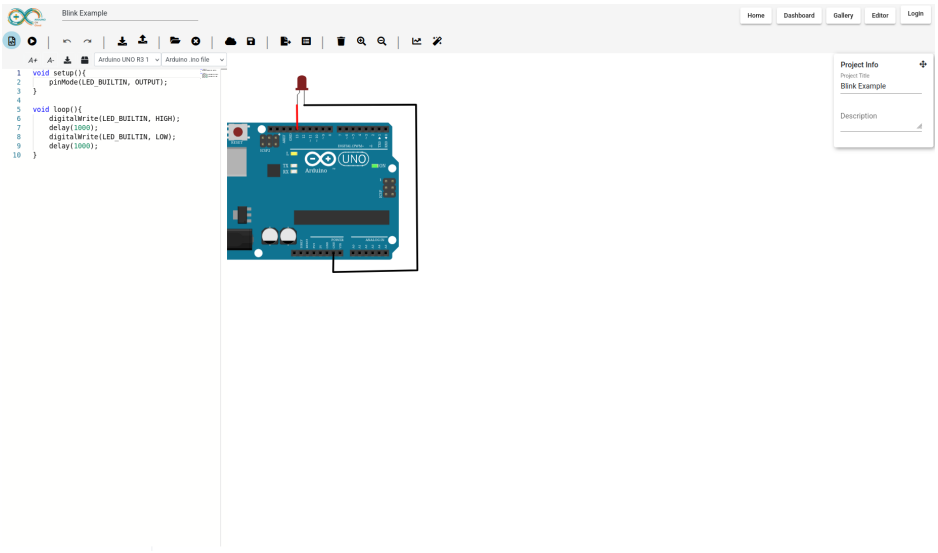

Figure 11.1: Before adding the font size dropdown feature

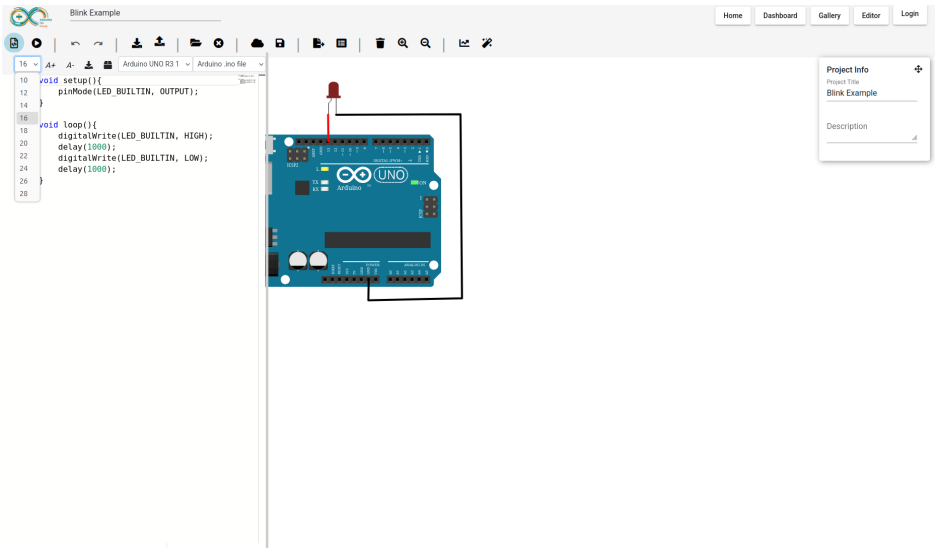

Figure 11.2: After adding the font size dropdown list feature

Chapter 12

Pin Mismatch Detection

The task is to introduce a feature that detects pin mismatches between the schematic and the code. A popup will alert users to any discrepancies before they begin the simulation, helping them catch errors early. Previously, users could connect pins to the Arduino without validation, leading to situations where the simulation ran but did not function as expected due to incorrect pin mappings. While this reflects real-world conditions, the platform is designed for students. Adding this validation will help catch errors early, reducing confusion and improving the learning experience.

12.1 Implementation

To detect pin mismatches, I implemented a validation process in the `Arduino Frontend/src/app/Libs/outputs/Arduino.ts` file that runs on the frontend without requiring a backend call. The process is divided into the following steps:

- **Extract Pins from Arduino Circuit:** The declared pins in the schematic are extracted and stored in an array. This allows us to track all pins that are connected to the Arduino.
- **Extract Pins Used in Code:** A regular expression (regex) is used to extract pin numbers from the Arduino code. This regex searches for variable assignments and function calls that involve pin numbers.
- **Check Pin Mismatch:** After extracting the pins used in the code, the system compares them to the array of declared pins from the schematic. If any pin referenced in the code is not found in the Arduino circuit (or if there are pins declared in the circuit but not used in the code, a popup

is triggered to notify the user about the mismatch. This validation occurs before initiating the simulation, ensuring that users are aware of any issues with the pins in their code.

For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/553`.

12.2 Result

The pin mismatch detection feature provides immediate feedback before running the simulation, preventing errors caused by incorrect pin mappings and reducing troubleshooting time. It enhances the user experience by identifying common mistakes early, allowing users to ensure that the pins in the schematic match those in the code, minimizing confusion and frustration. The following images show the state of the system before and after the implementation of the pin mismatch detection feature:

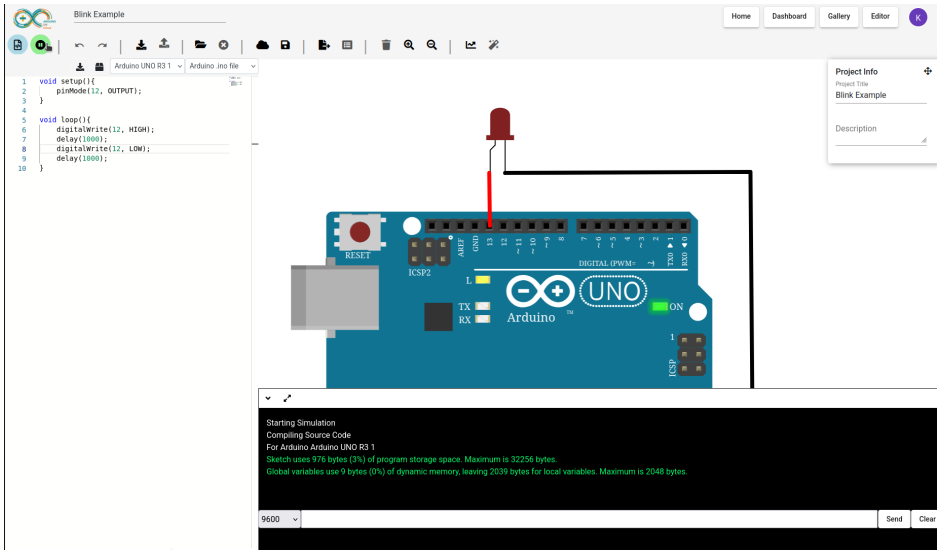

Figure 12.1: Before implementing the pin mismatch detection feature

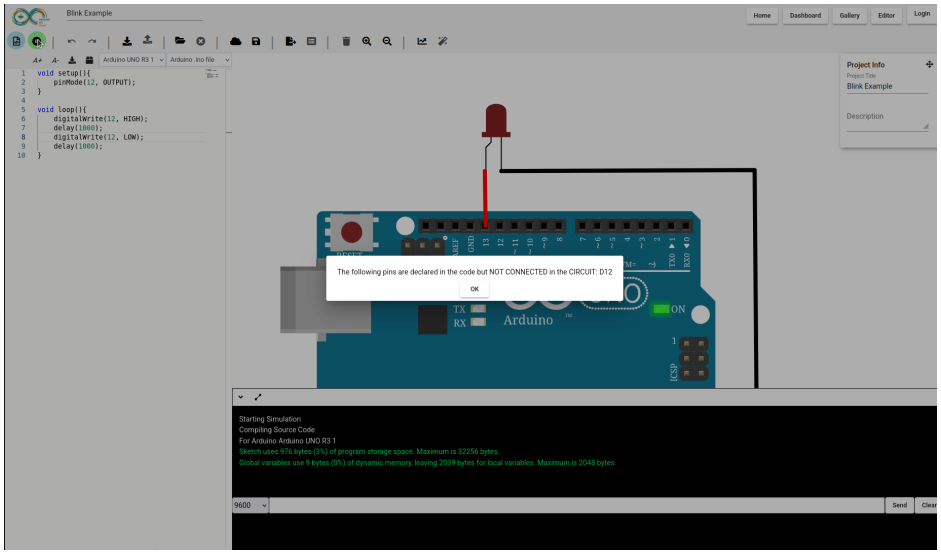

Figure 12.2: After implementing the pin mismatch detection feature

Chapter 13

Short Circuit Detection on Breadboard

To address the issue of components being placed on the first two or last two lines(power rails) of the breadboard, which can cause a short circuit on the physical breadboard, we need to implement a detection feature in the system. Currently, this problem is not flagged, and the circuit continues to simulate without any warnings.

13.1 Implementation

I implemented three functions: `checkBreadboardPowerConnections()`, `checkAllPinsConnectedToOneRail()`, and `checkShortCircuit()`. The `checkShortCircuit()` function calls both first two functions to perform the necessary checks for detecting short circuits. This function is invoked when user clicked the simulate button.the system will first check if the power rail is connected to power, then examine the component placements to see if all of their pins are connected to these powered power rails. If a component is incorrectly connected to a power rail, a popup will alert the user about the potential short circuit and stop the simulation. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/556`.

13.2 Result

The feature successfully detects short circuits caused by improper component placement on the power rails. When a short circuit is detected, the system notifies the user with a popup, preventing further simulation errors.

The following images show the state of the system before and after the implementation of the short circuit detection feature:

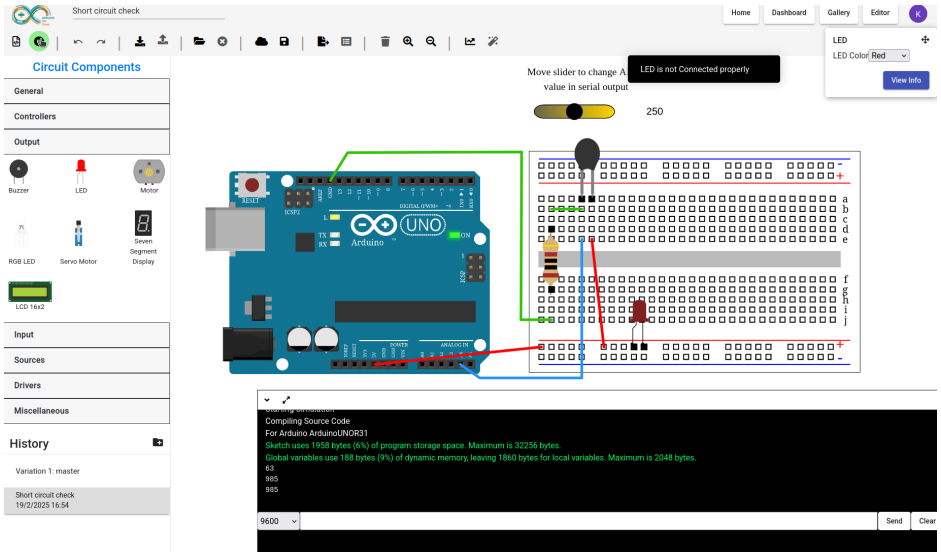

Figure 13.1: Before implementing the short circuit detection feature

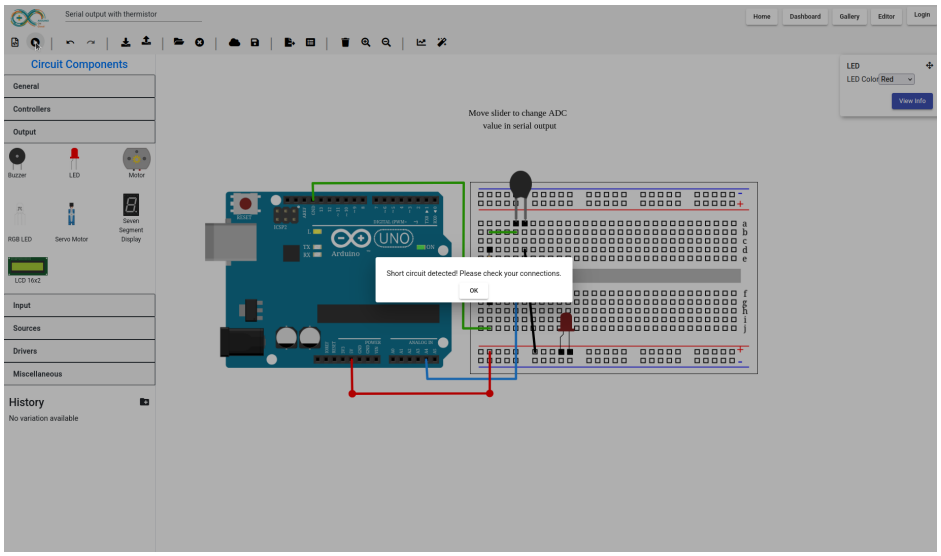

Figure 13.2: After implementing the short circuit detection feature

Chapter 14

Upload feature for .ino code into the editor

Previously, users could manually enter or copy-paste their Arduino code into the editor, which was time-consuming and prone to errors. To improve user experience and streamline the process, a feature was introduced that allows users to upload their Arduino code files (`.ino`) directly into the code editor.

14.1 Implementation

To enable file uploads, I modified the `ArduinoFrontend/src/app/code-editor/code-editor.component.html` file by adding a hidden file input element with a button for triggering the file selection. When the user selects a file, the `UploadCode()` method in the `code-editor.component.ts` file is invoked. This method uses the `FileReader` API to read the contents of the selected `.ino` file as text and load it into the code editor. Additionally, I ensured the file input is reset after the file is successfully loaded, preventing the user from uploading the same file multiple times without selecting it again. For more details, see the GitHub pull request: `https://github.com/frg-fossee/eSim-Cloud/pull/559`.

14.2 Result

With the new upload feature, users can now easily upload their Arduino code files by clicking the "Upload" button and selecting a `.ino` file from their local system. The editor will automatically load the content of the file, allowing users to start editing immediately. This eliminates the need for manual copy-pasting or typing, improving the speed and accuracy of code

entry. The file upload is fully synchronized with the existing code editor, and users can continue to modify their code as needed after the upload. The following figures illustrate the process of the user uploading the code:

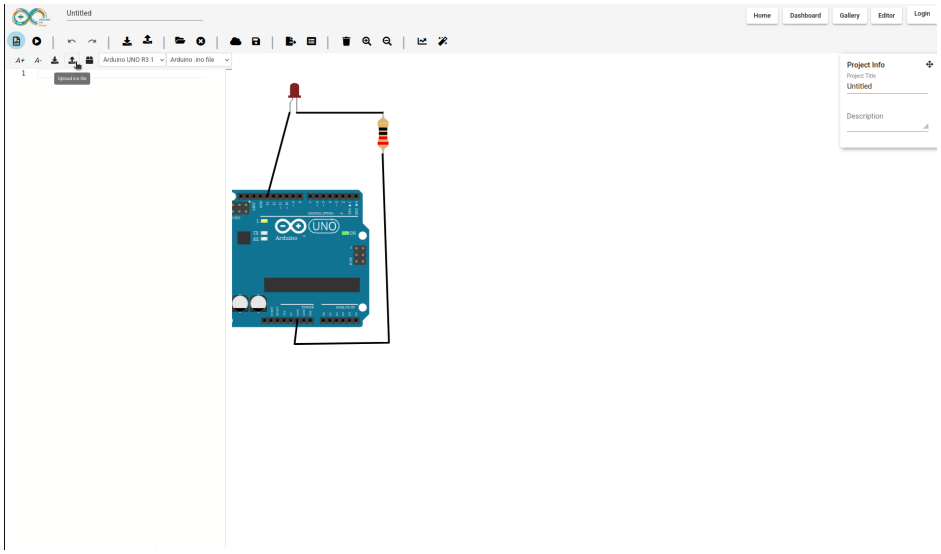

Figure 14.1: The upload code functionality process (part 1).

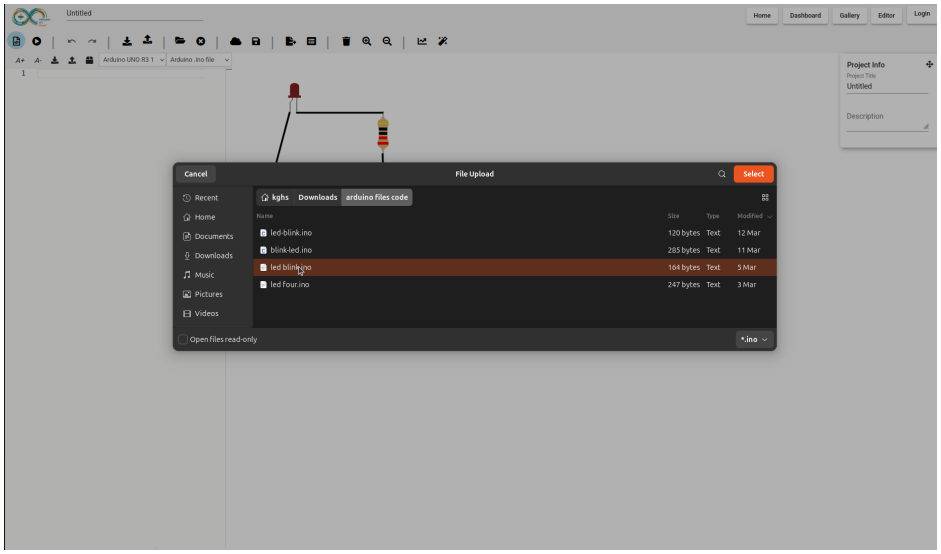

Figure 14.2: The upload code functionality process (part 2).

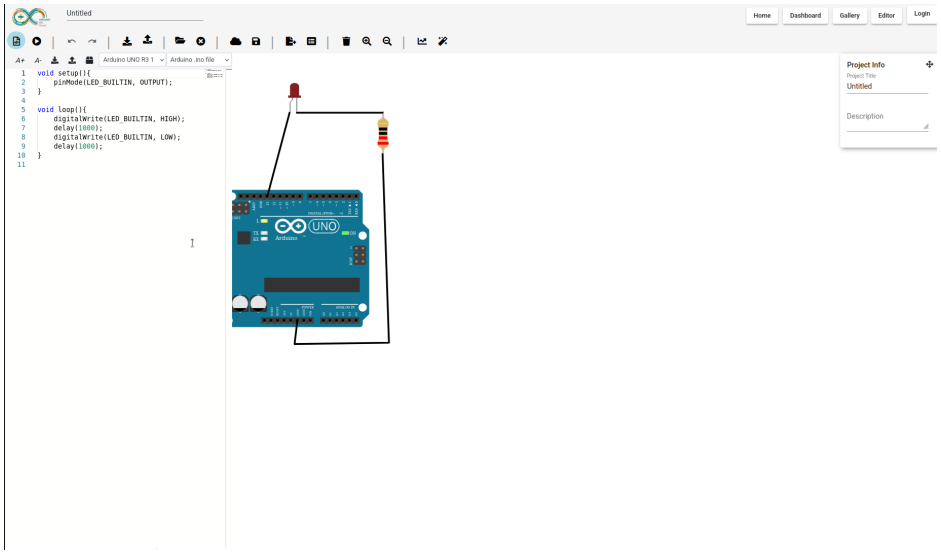

Figure 14.3: The upload code functionality process (part 3).

14.3 Further Extension

To improve the upload feature for the code editor, it would be beneficial to allow users to upload not only their Arduino .ino code files but also library files and component files (such as .h, .cpp, or other library files) that are typically required in Arduino projects. This would ensure that all necessary dependencies are included and that the code can be compiled and simulated correctly without missing external libraries.

Additionally, we can add support for uploading and importing custom libraries along with their associated components. This would allow users to import files that are specific to custom sensors, actuators, or other components they are using in their projects. By importing the library files, users can link these files to their project to ensure the correct components are available for simulation and interaction with the code.

Chapter 15

Removing Arduino Dependency in Code Editor:

Previously, the ability to write and edit `.ino` code was tied to the presence of an Arduino board on the schematic. This created a limitation where users could not write or edit Arduino code unless the board was included in the schematic. Additionally, if the Arduino board was deleted, the associated code would also be removed, which was problematic for users who wanted to retain their code independently of the board. The goal was to remove this dependency so that users can write and edit Arduino code regardless of whether an Arduino board is present or not.

15.1 Implementation

To achieve this, I made several key changes to the code editor's functionality. Firstly, I removed the dependency on the Arduino board in `ArduinoFrontend/src/app/code-editor/code-editor.component.html` by allowing the code editor to initialize independently of the board's presence. Additionally, the editor's state is no longer tied to the Arduino board. Previously, deleting the Arduino board would also delete the associated code. Now, the code remains intact in the editor even if the board is removed, ensuring users retain their work independently of the Arduino board. Furthermore, to streamline the process, I added a mechanism to automatically trigger the reinitialization of the code editor when an Arduino Uno is added to the schematic. When the user adds an Arduino Uno, a global flag is set, and the code editor is refreshed to apply the existing code to the new board. This ensures that the editor always reflects the correct state without requiring to open and closing the code editor. For more details, see the GitHub pull request:

https://github.com/frg-fossee/eSim-Cloud/pull/560.

15.2 Result

With this update, users can now write and edit `.ino` code without needing an Arduino board in the schematic. The editor no longer depends on the presence of the board, and the code remains persistent even if the board is deleted from the schematic. This change improves the flexibility and usability of the code editor, enabling users to focus solely on writing their code without worrying about the schematic layout. With this update, users can now write and edit `.ino` code without needing an Arduino board in the schematic. The editor is no longer dependent on the presence of the board, and the code remains intact even if the board is deleted. This change significantly improves the flexibility and usability of the code editor, allowing users to focus on writing their code without worrying about the schematic layout.

The following images illustrate the state of the system before and after implementing this feature:

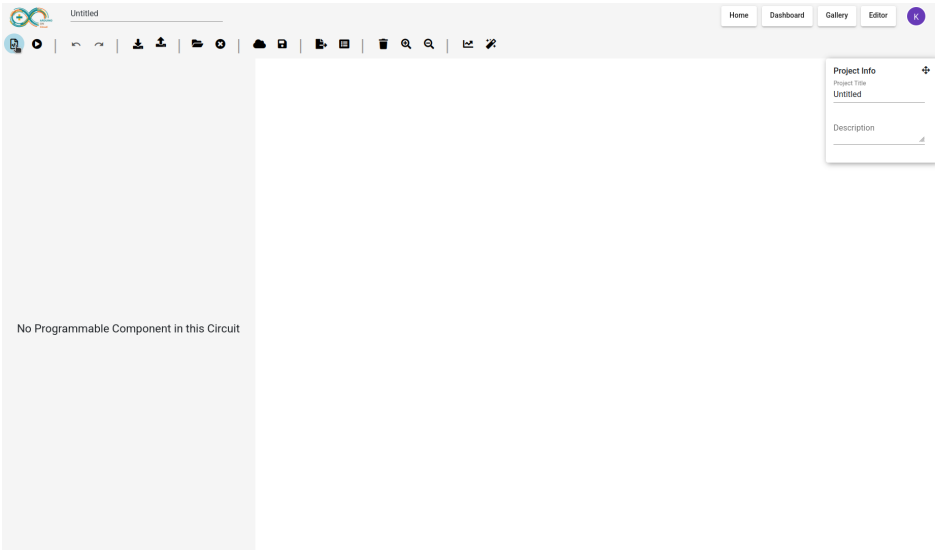

Figure 15.1: Before removing the Arduino board dependency (Part 1): Without the Arduino board, code cannot be written.

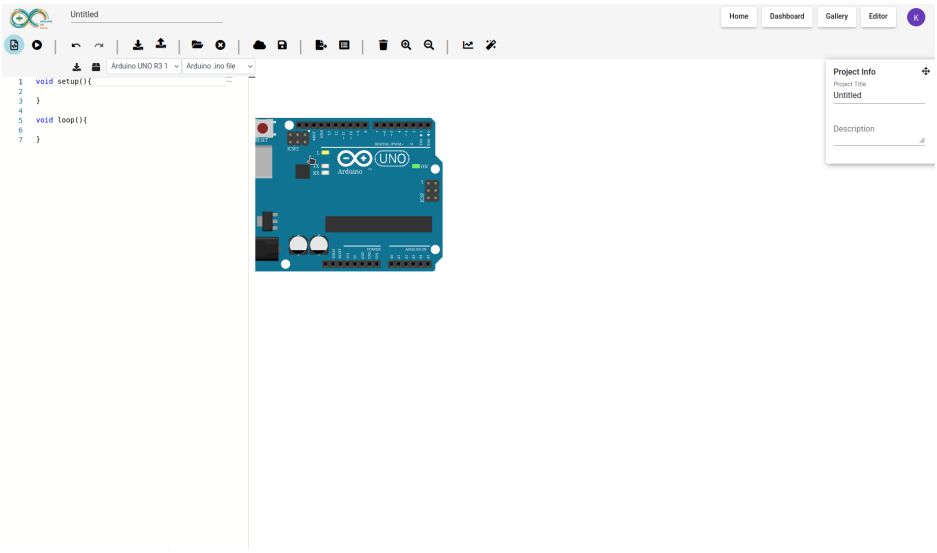

Figure 15.2: Before removing the Arduino board dependency (Part 2): Code can only be written when the Arduino board is present.

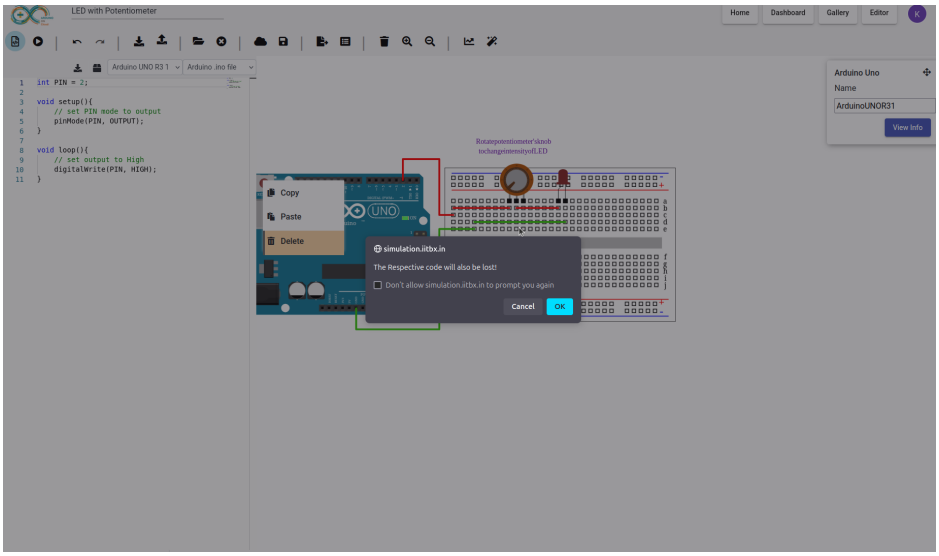

Figure 15.3: Before removing the Arduino board dependency (Part 3): A popup appears when deleting the Arduino board.

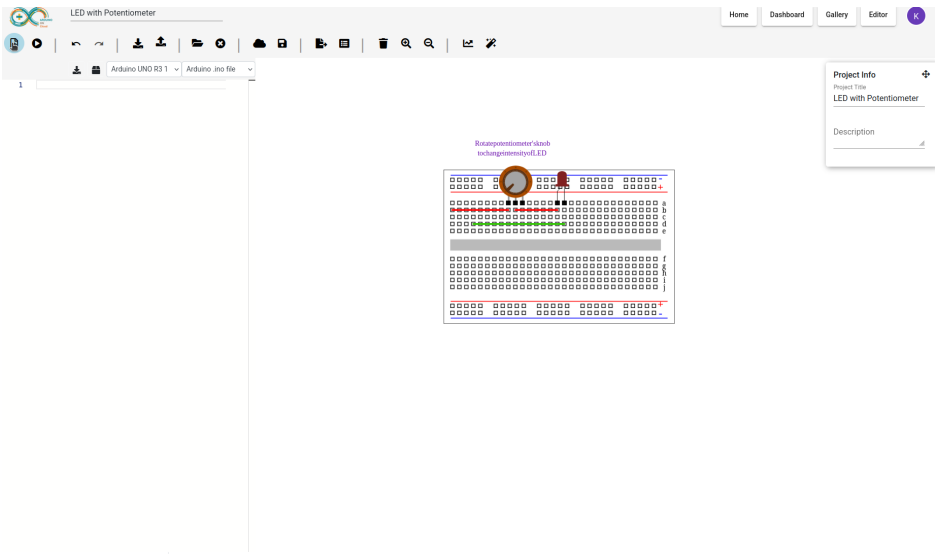

Figure 15.4: Before removing the Arduino board dependency (Part 4):Code editor after deleting the Arduino board.

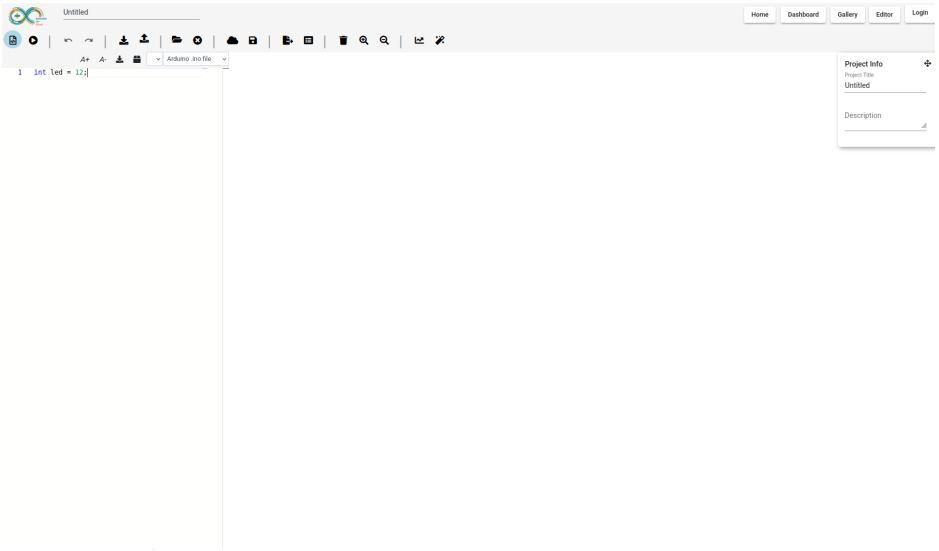

Figure 15.5: After removing the Arduino board dependency (Part 1): Code editor without Arduino board.

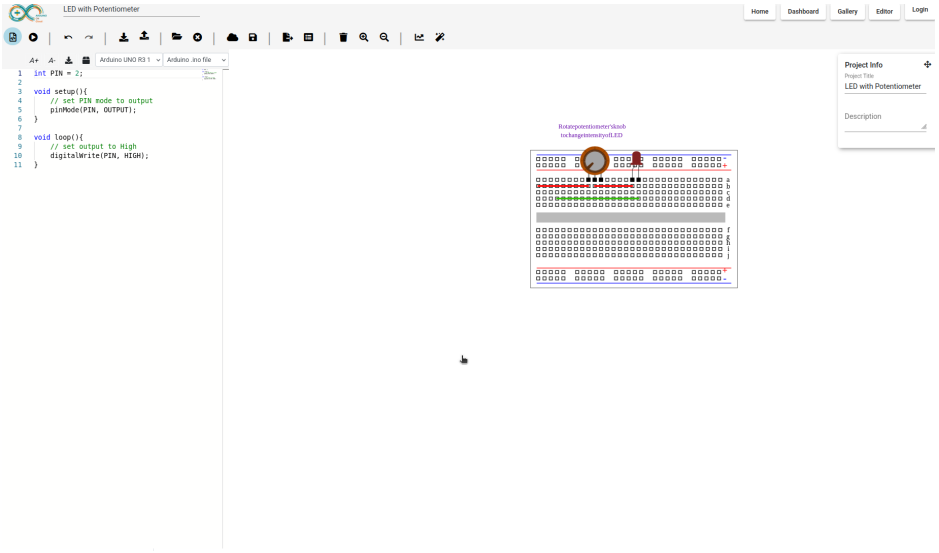

Figure 15.6: After removing the Arduino board dependency (Part 2): Code doesn't get deleted after deleting the Arduino board.

15.3 Further Extension

As the code editor is now independent of the Arduino board, a possible next step could be enabling the editor to support multiple board types (like ESP32, Raspberry Pi, etc.). This would allow users to work with a variety of platforms without requiring an associated schematic. Additionally, introducing board-specific code snippets or libraries within the editor would help guide users when writing code for different platforms, thus making the tool more versatile and appealing to a wider range of users.

References

1. https://esim-cloud.readthedocs.io/en/latest/overview/index.html#arduino-on-cloud
2. https://www.1edtech.org/standards/lti
3. https://github.com/rohitjose/django-lti-auth
4. https://github.com/Harvard-ATG/django-app-lti