1. Team Information
    - Group Name:
        1. Group 4 - Online MIPS Emulator
    - Team Members:
        1. Alec Brookens, Andrew Olsen, Anthony Velez, Javier Golpe, Bleon Muqaj, Arman Bunic-Duratovic
    - Preferred Communication Medium:
        1. Email: bmuqaj@uwm.edu (Questions will be relayed to team)
2. System Overview

    The project our group worked on for a web-based MIPS Simulator was designed for students learning MIPS assembly. It allows users to write, assemble, and execute basic MIPS programs within their browsers, eliminating the need for a local simulator to be installed. The project also has useful features for learning MIPS assembly, including syntax highlighting, register viewing, memory outputs, and step-through execution using breakpoints.

    The web-based MIPS Simulator tool is meant for education. It lifts students from the barrier of complicated installation and allows them to focus on learning MIPS assembly. Students can easily run and debug MIPS code that is consistent with the current Computer Science Architecture course at UW-Milwaukee.

3. Access Instructions
    - **Prerequisites:**
        1. **Node.js must be installed on your system.** Follow the instructions for your operating system at: https://nodejs.org/en/download
        2. An IDE of your choice (We recommend IntelliJ Idea (Educational License) or VSCode)
    - **Step 1:** Clone code repository **directly** within the IDE (GitHub):
        1. **Paste this when IDE asks for GitHub Repo Source:** https://bleon-muqaj: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@github.com/bleon-muqaj/group4capstoneproject.git
        2. This link is a private repository with **read-only privileges**. Consult the documentation for your IDE to clone the repository.
            1. **Special Circumstance:** If token is not working, contact email for access to code repository via https://github.com/bleon-muqaj/group4capstoneproject.
                - You do NOT have current access through this link without contacting for access.
    - **Step 2:** Install necessary packages
        1. From the root directory of the cloned project, run the following command in your system's terminal: **"npm i --legacy-peer-deps"**
        2. This will install all the necessary packages and avoid compatibility issues.
    - **Step 3:** Start the project (No login Credentials needed)

1. From the root directory of the cloned project, run the following command in your system's terminal: **"npm start"**
2. This will run the app and allow you to access it in your browser from the provided localhost address.

4. Required Test Scenarios (Perform in order)
    - **Create & assemble program**
        1. **Using Premade Example Steps (recommended)**
            1. "Select Example" in the top bar and choose "Hello World." A new tab opens with a pre-filled MIPS assembly with the program.
            2. Click "Assemble"
            3. Click "Run"
        2. **Expected Results**
            1. After "Assemble," the UI will switch to the execute view and the Text and Data segment tables populate.
            2. Terminal shows: "Assembly of Hello_World.asm was successful. You can now run your code."
            3. After "Run," the console prints "Hello, World!"
    - **Error detection (four quick sub tests)**
      (Start each sub-test with a clear Hello World file in the edit tab (either using ctrl-z or create a new Hello World file from "Select Example")
        1. **Sub-test 1 - Invalid label**
            1. Inject in a blank line "li $v0, notalabel"
            2. Observe the red underline and tooltip when hovered over.
            3. Click "Assemble"
                - Expected Assemble output: "...notalabel" is not a valid MIPS Register or Label."
        2. **Sub-test 2 - Invalid Instruction**
            1. Inject in a blank line "l $v0, 10"
            2. Observe the red underline and tooltip when hovered over.
            3. Click "Assemble"
                - Expected Assemble output: "...\"l\" is not a valid MIPS instruction."
        3. **Sub-test 3 - Invalid register**
            1. Inject in a blank line "li $v22, 10"
            2. Observe the red underline and tooltip when hovered over.
            3. Click "Assemble"
                - Expected Assemble output "...\"$v22\" is not a valid MIPS Register or Label."
        4. **Sub-test 4 - Wrong argument count**
            1. Inject in a blank line "li $v0"
            2. Observe the red underline and tooltip when hovered over.
            3. Click "Assemble"
                - Expected Assemble output "...\"li\" expects 2 argument(s), but got 1."

(In all four cases, the tab stays in the Edit tab and does not automatically switch to the Execute view).

- **Execute with Breakpoint and Controls**
    1. Revert to a clean Hello World file (reload example or undo errors)
    2. Click “Assemble” to enter “Execute View”
    3. If the “Code” column is hex, click “Show as Instructions” (bottom of text-segment panel).
    4. Tick the checkbox next to the first “syscall” instruction (prints out the string during execution).
    5. Set Speed slider (top-right) to 1.
    6. Click “Run to BP,” and execution should stop on the breakpoint line.
    7. Click “Step” once, and the program prints “Hello, World!”
    8. Click “Run to BP” again (nothing to stop at, program keeps running).
    9. Before reaching the end of completion, press “Pause” resulting in the execution halting.
    10. Click “Resume” and the execution finishes
    11. Click “Set BPs” (all breakpoints on), then immediately click again “Set BPs” (clears all breakpoints).
    - **Expected Results**
        - Execution halts at the checked instruction
        - Registers update live as execution is occurring
        - Console shows the system output in console immediately after the first syscall instruction.
        - Pause causes an instant halt and Resume continues from the current instruction line.
        - Set BPs toggles on and off the checkbox in the text segment.

5. Optional Test Scenarios
    - **PDF Manual Viewer**
        1. Click the “View Manual” button in the header.
        2. On the bottom right of the panel, drag to resize the pop-out.
        3. On the side-bar contents select “addiu”
        4. Drag the pop-out across the page.
        5. Click the “X” button on the top right of the pop-out to close.
        - **Expected Results**
            1. Manual appears in overlay.
            2. Page jumps directly to the “addiu” section.
            3. Sidebar and page adjust in size when the pop-out size is adjusted.
    - **File System workflow**
        1. With a current file open (pre-made code example) and Assembled Successfully with the “Assemble” button, click the “File” drop down (top-left)
        1. Select “Download .asm”
        2. Select “Download .data”

3. Select “Download .text”
4. In your download, drag the .asm file recently downloaded and in the top menu area when a green highlighted section appears and drop the file.

- **Expected Results**
  - The files are downloaded
  - The files contain content in multiple rows (if applicable to the program itself).
  - The .asm file is able to be loaded into the program with a drag and drop functionality.

- Theme and UI User Customizability
  1. Click “Settings” (top-right)
  2. Toggle “Dark Mode”
  3. Move “Font Size” slider
  4. Click “Hide Line Numbers”
  5. Then click “Close”
  - Expected Results
    - The entire UI switches to a dark theme
    - The font size in the editor changes
    - The editor line numbers disappear.
    - The close button exits the settings.