GSOC 2025 - Circuitverse

Project 2: Desktop Application & Vue Frontend Updates

Personal Details

- **Name:** Harsh Rao
- **College**: BITS PILANI, GOA
- **Course/Program:** B-Tech in Electronics and Communications
- **Email:** harshraocodesup@gmail.com
- **GitHub:** ThatDeparted2061
- **Socials:** LinkedIn
- **Phone:** +91 8171763478
- **Current Country:** India
- **Resume/CV:** Link

About Me

- Hey! I am Harsh Rao, a 2nd-year Engineering student pursuing a B-Tech in BITS Pilani - Goa Campus. I found an early interest in Computer Science watching my dad code on his pc. Later, I found myself equally interested in electronic gadgets and their workings. So, I started learning to code from my dad and pursued my interest in electronics by building projects, interlinking coding with electronics. I started with C, C#, Cpp then tried out Arduino and Raspberry Pi OS, ROS. Then I moved on to web development and

came across the marvels made upon JavaScript and got infatuated with JavaScript only to later move on to typescript. I came across Circuitverse while being a nomad on the internet. After getting to know about it and realising that open-source could let me contribute to such huge platforms and work with people who are equally interested and far more knowledgeable, I decided that I would try contributing to this platform, learn from the people around me. Nevertheless, in the past 6 months, I made amazing friends through the Circuitverse dev community and met amazing mentors. Then got to know that I could make even more beneficial contributions to the organization through GSOC, and that brings us here.

After reading previous contributors' works on the vue-simulator, I feel that we are very close to completing our longtime goal of replacing the legacy simulator, and I can see the things left to work on for that goal to be finally achieved. I have felt that way for a long time and have been contributing extensively to vue-simulator.

After working with the Circuitverse community for this long I realised the impact created by Circuitverse on the people all around the world. I feel this can be my first chance to do something that can directly help so many people all around the world and make me part of the world's citizens. This is the reason why I have put in everything I can towards this proposal and my contributions to the community.

Commitment

- Project duration: 175 hours
- Dedicating 20+ hours a week
- I have an educational internship during the vacation, but that'll be online, so it won't affect the working hours.
- I'll be having some evaluative exams in August, so I'll balance out the working hours in other weeks around that time.

Contributions So Far

- **Contributions through PRs (Well-documented data)**
    - Total PRs - 50
    - Merged PRs - 15
    - Documentation PRs - 3
    - Collaborations - 8
    - Open PRs - 24
    - Bugs found - 1 + 1 (collaboration)
- **Currently Collaborating on:**

    - Rails Upgrade
    - Automated testbench for Weekly Contest (**Documented Approach**)
    - Fixing the CI failure due to Percy
- **Community Contributions**
    - Member of the issue triaging team; reviewed many bugs and PRs to
        - Improve the efficiency of Circuitverse's GitHub issue tracker.
        - Ensure equal opportunity to the contributors by marking PRs that violated the code of Conduct.
        - Reduce the number of PRs, value and examine their current value and make decisions based upon that.
    - Collaborated with other folks in managing and hosting the weekly meetings for the past 6 months.
    - Attended to the issues and concerns that were reported by the users in the community channel by helping them navigate through their shortcomings.

Abstract

I have selected **Project 2: Desktop Application & Vue Frontend Updates**. This project will finally mark the completion of vue-simulator. After the work of the past 4 gsoc contributors. After long discussions with the Niladri, Aryann and Josh, I can gauge perfectly what is left to be done and strongly believe that with this meticulous implementation that I have tailored for the project, we can finally mark the completion of Vue-simulator and move on to the next steps in Circuitverse history.

I have divided the project into 7 parts. I'll be taking them in the same serial as mentioned in the detailed implementation

Deliverables

1. **Web-Simulator (cv-frontend-vue codebase):**
    a. All the **bugs** and **failures** will be resolved
    b. The **src** folder is synced with **v0/src** and **v1/src**, with the src folder further removed
    c. Implement a **dynamic version switch** in the simulator for the user to switch between versions.
    d. Improved **Testbench implementation** with more and improved features to enhance user experience.
    e. All the **JavaScript** files will be converted to **TypeScript** to ensure type safety

f. **Vue's reactivity** implemented to UI sections previously covered by JavaScript and improving the current Vue files. Increase usage of **Vue's lifecycle hooks** and **Composition api**.
g. Integrate the Vue-simulator with the primary codebase to replace the legacy simulator.

2. **Desktop Simulator (Tauri)**
   a. All the bugs and failures will be resolved using **Tauri Plugins** (Recommended by Tauri)
   b. Refactored JS and Vue files to use **Rust backend** for handling **APIs** to improve resource usage
   c. Refactored **User Authentication** for the Tauri App to be more robust and user-friendly.
   d. Using plugins like **Window state** and **Persisted scope** to enhance user experience.
   e. Refactor code to be suitable for **multi-platform build**.
3. **CI for cv-frontend vue**
   a. Create workflows for the Simulator and Desktop Application.
   b. Add additional **vitest testing** for the **IPC requests** we'd be creating.
   c. **Mocking the tauri APIs** while testing for **quicker CI** and reduced load on GitHub actions.
4. **CD for cv-frontend vue:**
   a. Establishing an **automated release pipeline** for streamlined deployment using Tauri actions.
   b. Building artifacts for all 3 platforms (**Multi-platform build**) with each release and enlisting them under each release.
   c. Enhancing the **security of the builds** through **Code signing** for future deployment in App Store, Microsoft Store and Debian packages
   d. **Optimizing the build artifacts** by adding Cargo profile, rust flags and other such measures under Tauri App-size
   e. Implementing a reliable **update mechanism** to deliver future updates to the Application seamlessly
5. **Documentation**
   a. Refactored the **official documentation** as per the latest features of the simulator.
   b. Updated **developer documentation** for the Tauri application and version implementation.

Programming Languages/tools used - VueJS, Tauri, TypeScript, JavaScript, GitHub Actions, Rust, Ruby on Rails, SCSS, Bootstrap, Vite.

Detailed Implementation

1. Fixing the Simulator and Desktop Application

- Fixing the current issues in the Web and Desktop Simulator
    - Current Failures link
        - Vue-simulator: Documented issues link
        - Desktop Simulator: Documented issues link
    - Most of these are due to irregularities in type definitions, imports, and exports. I'd fix these first to use the resources properly.
- Implementing Features, **Origin of major issues:** Vue methods that require opening anything over the simulator canvas fail because the Tauri frontend is not directly allowed to open external links or more windows from the Webview. Due to this, we'd require manipulations in Webview components under the DESKTOP_MODE environment variable using Turi-plugins. Some examples -
    - Clipboard failures - Clipboard plugin
    - Improve user interaction while importing, exporting, and saving files and images - Dialog plugin and Fs plugin
    - Keyboard Shortcut failures - Tauri Keyboard shortcut API or Tauri Global Shortcut plugin
    - Documentation link failures - Opener plugin or shell plugin(kind of overkill)
    - Window state for better UX - Window state plugin
    - Save filesystems and asset scopes - Persisted scope plugin

- **User Authentication for Desktop Application**
  (Procedure Flowchart) – Don't use dark mode

The current user authentication doesn't work for the desktop Application. Currently, we change the URL to **`/users/sign_in`** (Link to code), which is blocked by Tauri due to strict CORS policies. I intend to change it and deliver the following -

- Have a built-in login form that sends the credentials directly to the backend for authentication.
- The user stays entirely within the desktop application during the login process, improving the UX and maintaining the session persistence locally
- The user stays logged in until the user logs out intentionally, clears the cache or deletes the application

Implementation steps for the above -

- **Render Native Login UI:** Create a Vue component within the simulator for email/password input. Add “Remember me” functionality using a checkbox and Captcha for human verification.
- **Send Credentials to Backend:** As specified in the CircuitVerse API, submit credentials via a **POST** request to `https://api.circuitverse.org/users/sign_in.json`. Ensuring the request body includes the user’s email and password under a **`user`** key. Using **`Axios`** or **`fetch`** to send the request with the appropriate **`Content-Type: application/json`** header.
- **Handle Auth Response:** On success, receive a session cookie or token (depending on Devise setup). Extract and store this for future use.
- **Securely Store Session Data (Procedure):** We’ll use **`@tauri-apps/api/fs`**, **`tauri-plugin-store`,** or **`tauri-plugin-secure-storage`** to save login state locally (e.g., user email and session cookie).
- **Maintain Persistent Session:** On app launch, check for the existing valid session. If found, auto-login the user without showing the login screen.
- **Perform Authenticated API Actions:** Now use the stored session cookie in requests to authenticated endpoints like **`/api/v1/circuits`** for saving circuits online.

Fixing the Broken Verilog Module in the Desktop Simulator

The current Desktop application has a broken Verilog module that limits its functionality. This issue arises because the API endpoint **`fetch('API/v1/simulator/verilogcv')`** in the **`Verilog2CV.js`** module fails to retrieve data in the Tauri environment. This is due to the backend environment not supporting standard **`fetch`** requests, unlike the Web version, which uses a Node.js/Express backend to handle such requests. Additionally, Tauri's WebView enforces strict CORS policies by default, which contributes to the problem. We’ll use the Tauri HTTP plugin because Tauri allows this plugin to bypass CORS policies (Github Discussion).

- **Implementation plan**
  - We’ll be using the plugin Tauri HTTP client as per the **DESKTOP_MODE** environment variable to create the API requests.
  - Enable the Tauri HTTP client plugin in the tauri.config.json with domain whitelisting for Circuitverse API endpoints.
  - Set required **CORS** permissions for the Webview
  - Implement a dual-mode fetch service that -
    - Uses **@tauri-apps/api/http** in **DESKTOP_MODE**

        - Falls back to native Fetch API in web mode
    - Normalise response formats between both environments.
    - Now, to integrate the Verilog module, replace all **fetch()** calls with the environment-aware service
    - Maintain identical request/response interfaces for backwards compatibility
    - Add error boundary handling for platform-specific features.
    - For system adaptation :
        - Configure Vite to tree-shake Tauri-specific code during Web builds.
        - Set environment variables for API endpoint Routing.
    - Web: Relative paths (`/api/v1`)

- **Testbench Improvements:** The current testbench is great, but improvements can be made. Here are the improvements I am gonna implement -
    - Giving users 2 options to enter the inputs, either through the manual method they use by adding inputs and name them, or they can choose the inputs by clicking on them one by one; here, the inputs will be automatically labelled. This step will improve the user experience
    - Fix the current issues in the testbench, the occasional broken tests, and improve the efficiency of switching between different groups.
    - Basic Figma link for the testbench idea.

---

2. Integration of TypeScript and Vue Frontend Updates

TypeScript Integration

1. **Modular Conversion**: Convert core circuit logic files to TypeScript while separating UI logic into Vue components for better structure and maintainability.
2. **Improved Type Safety**: Refine type definitions, enable strict mode, and leverage TypeScript's inference to catch errors early and reduce bugs.
3. **Optimised Codebase**: Consolidate shared interfaces, optimize global variables, and refactor function signatures for improved performance and clarity.

Vue Optimization

1. **API Modernization and JQuery removal**: Migrate components from Options API to Composition API, remove jQuery, and replace legacy lifecycle methods with modern Vue hooks.
2. **State & Structure Cleanup**: Standardize state management, clean up unused logic, and eliminate outdated mixins/directives for leaner components.

3. **Typed Vue Components**: Ensure full TypeScript support by defining props, emits, and refs correctly while converting the remaining UI JavaScript into dynamic Vue components.

---

3. CI for Simulator and Desktop application (Abstract Implementation) (Template)

- **Step 0:** The primary step will be fixing the current test failures. Once that is done, we move on to the next steps.
- **The final workflows would include: Codeql**, **Dependabot Updates**, **ESLint**, **Web Simulator tests**, **Desktop Simulator tests** and **Desktop Simulator Release**: (Resource for writing the workflows)
- The primary testing for the simulator and the desktop app will be from the spec files.
- **Vitest Desktop Testing environment:** We will be mocking Tauri APIs (Resource with reason) (Comparing options) using **`@tauri-apps/api/mocks`**. This plugin primarily intercepts IPC requests, which are not required in most of our spec files. However, as we integrate additional Tauri modules like **`fs`** (currently in use), we will incorporate this mocking into the CI pipeline and new test cases for IPC request testing. Mocking APIs helps us prevent building a whole backend for each PR, which makes CI/CD quicker and lighter.
  The following steps ensure that the Tauri-specific mocking applies only in Desktop Mode:
  - After installing tauri's mocks api package, we'll create setup.ts, which will be mocking the API's when the **`DESKTOP_MODE`** is true.
  - Modify the vite.config.ts to use setup.ts
  - Simply check for **`DESKTOP_MODE`** and write **Tauri-specific test cases**
  - Update the Desktop Simulator tests workflow file
- **Additional Vitest testing (Desktop + Web):**
  - Adding Tests for the new IPC requests we'd be creating for the plugins. This would help us achieve platform-agnostic code that is reusable for both the Web Simulator and the Desktop Simulator.
  - Adding tests for the additional implementation in the testbench and Verilog support

---

5. Integrating the Vue simulator with the primary code base

The vue-simulator was previously integrated for dev server only through flag toggle using flipper in this PR. Now, when we complete the vue-simulator, the procedure to integrate the vue-simulator is as follows (Detailed Implementation) -

- Modify the conditional renderings with flipper to automatic rendering of the vue simulator without the flags like in this code (here we just need to remove this line). This step makes the Vue simulator the default simulator for the dev server.
- Now for the production, after minor refactors, we'd be fine because the major work is already done, like the multi-stage build in Dockerfile.Production (code), etc, from this PR.
- Now comes the most tedious step, which would be done most meticulously- Refactoring the codebase to remove the legacy simulator, its rendering, removing its assets from public and src, like changing the render embed and render edit etc.

---

4. Versioning of the App and release for the Desktop Application and the Update mechanism for the Tauri App (Abstract Implementation Proof of Work)

Deliverable: A robust, secure, and automated CI/CD pipeline for cross-platform desktop application distribution, leveraging **Tauri's `tauri-build` and `tauri-distribute`** modules. The pipeline will streamline release management with the following features:

- Automated builds will be triggered for **Windows, macOS, and Linux on each GitHub Release event**.
- Built artifacts will be automatically attached to the **GitHub Release** as downloadable assets.
- Each release will include a **changelog** (Like this) documenting breaking changes (if any) and enhancements over the previous version.

Implementation

1. **Version Extraction:** Parse the application version directly from **`tauri.conf.json`** to maintain consistency.
2. **Code Signing (Procedure):** Implement platform-specific **code signing** to ensure authenticity and security.
    - **Windows:** Sign MSI installers with a Code Signing Certificate.
    - **macOS:** Notarize and sign DMG builds for Gatekeeper compliance.
    - **Linux:** Sign AppImage executables for distribution security.
3. **Platform-Specific Runners:** Due to cross-compilation limitations in Tauri's experimental phase, native builds will be executed on **OS-specific runners**. This ensures stability and avoids binary compatibility issues.

4. **Future Compatibility:** This workflow lays the foundation for **listing on the Microsoft Store and Mac App Store** by adhering to platform-specific distribution requirements.

Build Artifacts

- **Windows: `.msi`** (Microsoft Installer)
- **Linux: `.AppImage`** (Portable Executable)
- **macos: `.dmg`** (Disk Image Format)

Workflow Steps

1. **Version Increment** in **`tauri.conf.json`**
2. **Trigger GitHub Actions Workflow** on release event
3. **Parallel Platform-Specific Builds**
4. **Code Signing & Notarization**
5. **Artifact Generation**
6. **GitHub Release Publication**

Build-Optimisation: The build method stated above creates the artefacts, but we should further reduce their size through the optimisation of the build to keep the builds more robust and lightweight. This shall be done through the following ways

- Adding Cargo profile for frontend agnostic size improvements (Resource)
- Enabling remove unused command etc in the tauri-config.json, because why pay for what you don't use. **(feat: add a new option to remove unused commands #12890)**
- Other such commands will be added as per the then state of the codebase. I'll be referring to **The Cargo Book / Profiles** for the optimisations.

Implementing update mechanism - Procedure Flowchart (don't use dark mode)

***Deliverable*:** This step aims to complete the following:

1. Once a new artefact is signed and released, the users of the Desktop Application will get a prompt asking for permission for the update. They'll have the option to cancel it or postpone the request for the next simulator session.
2. The prompt delivered will have a briefing on the recent update and a link for the new changes (Github release section).

Procedure: (Link for detailed procedure) For this, we'll be using the official Tauri updater plugin (Strict Reference). The following highlights the steps:

1. **Check for Updates:** On application startup, check if a new signed release is available.
2. **Prompt User:** Show a notification asking for update permission.
3. **User Decision:** Allow users to accept, postpone, or cancel the update.
4. **Proceed with the Update:** If accepted, download and apply the update.
5. **Postpone Handling:** If postponed, remind users in the next simulator session.
6. **Cancel Handling:** If canceled, do not prompt again until the next release.
7. **Ensure Security:** Verify the update's signature before applying.

---

6. Implementation of Version Switch

(Procedure Flowchart) - Do not use dark mode

1. Syncing src to v0/src and v1/src
    a. Once all of the current PRs concerning ./src/ have been reviewed, we'll sync them to v0/src and v1/src and eliminate the separate src folder. Once the typescript and vue integration is complete, the completion of the v0 version for the simulator will be marked.
2. Implementing version switch (backend)
    a. The implementation of version switching in the simulator can be done simply by declaring an environment variable in a named version file. Still, I wish to build upon the previous work by Aryann under PR #5030, wherein we'll be rendering dynamic file paths based on the simver parameter. Then, I'll check the consistency of flipping versions using this method.
    b. The prior implementation for this will be for the dev environment.

---

7. Documentation of the Vue-simulator and the Desktop Application

Objective - Add well-tailored documentation of the Web and App simulators as per the new changes to the simulator. These changes shall be performed in 2 steps

1. Updating the developer documentation

   a. Version Control Documentation: Properly document the implementation of version control in the simulator.
   b. Desktop Application Documentation: Create developer docs for information regarding its working setup and troubleshooting
   c. Vue-Simulator Documentation: Document the changes due to the shift from legacy to the Vue simulator
2. Updating the Official Documentation of Circuitverse:
   a. The current documentation contains documentation on the legacy simulator. After it's replacement, we'd need to upgrade the resources used in the documentation with the vue, simulator.
   b. Once version control is implemented we'd need to add its use cases and how users can use circuits for different versions.
   c. After the Upgrade option is implemented, users should be able to access well-tailored documentation on how to upgrade their Desktop Application, troubleshooting etc.

Additional Objectives (if time permits, else I wish to complete these after the GSOC period is over) :

- Improve the security of the desktop builds using Tauri Security, for only then will we be safe to deploy the builds on multiple online stores like App Store, Microsoft Store.
- Fix internationalization with the replacement of intlify plugins and extend it to more languages for Desktop applications.
- Implement end-to-end testing through WebDriver using the Tauri-driver plugin. This is an amazing idea for the CI of desktop simulator since this will enable us to verify DOM changes and web interactions (like Percy), simulate button clicks and keyboard inputs (unlike Percy), access native APIs dialogs, and full E2E test.
- Implement hot version switching from the Primary codebase (frontend). - Implement a toggle switch in the simulator where in user can control the version of the simulator. The user should be able to choose to stick to v0 or try out new or beta features in v1 and furthermore.

Project Plan

Project Size - Medium Project Timeline - 12 Weeks

During the whole timeline, I will continue the **Typescript and Vue integration each week since it is the biggest task, so spreading it over the whole tenure is the best approach.

Week	Dates	Tasks to be Completed
Week 1	June 2 - June 8, 2025	Refactor the vue and js files to fix the issues in the vue simulator and fix the test failures
Week 2	June 9 - June 15, 2025	
Week 3	June 16 - June 22, 2025	Initiating the IPC requests using Tauri plugins to fix the issues and improve User experience with the features mentioned in Point 1 of Implementation
Week 4	June 23 - June 29, 2025	
Week 5	June 30 - July 6, 2025	Implement User Authentication for the Desktop Application
Week 6	July 7 - July 13, 2025	Implement Tauri plugins and update the API endpoints to fix the broken Verilog module in the Tauri-app
Midterm Evaluation	**July 14 - July 18, 2025**	**Midterm Evaluation Milestone**
Week 7	July 14 - July 20, 2025	Implement Testbench enhancements
Week 8	July 21 - July 27, 2025	Add tests for the IPC intercepts for Tauri, Verilog integration and Testbench enhancements
Week 9	July 28 - August 3, 2025	Integrating vue-simulator with the primary codebase and deploying versioned release pipeline
Week 10	August 4 - August 10, 2025	Implement update mechanism for the Desktop Application
Week 11	August 11 - August 17, 2025	Implement the backend for the version switch from the primary codebase
Week 12	August 18 - August 24, 2025	Updating the developer docs and the official documentation of Circuitverse
Final Week	August 25 - September 1, 2025	Final testing, documentation, and code cleanup

Final Submission	**September 1, 2025**	**Project submission deadline**

** The date for **integration of the Vue simulator** with the primary codebase is currently tentative and will depend more on the time when we feel the Vue simulator is truly ready to replace the legacy simulator. This means the date might be shifted up but not down.

Major Milestones

Week 4 - All the issues have been fixed, and the Desktop Application's features have been added with the help of Tauri plugins.

Week 7 - Testbench enhancements have been implemented, and broken Verilog has been fixed

Week 9 - Vue simulator has been completely integrated, and we have completed additional Vitest Testing for Desktop Application

Week 11 - Update mechanism and and the backend for Version switch has been implemented all that is left is Documentation now.

Additional Information

I have only drafted 1 proposal, and that is for Project 2 under Circuitverse.

Ps - My username is ThatDeparted2061, where 2061 shows the time **Halley's Comet** next appears. The username is in the hope that when Halley shows up again the first of the humans by then would have departed for Mars.

Coming back to the proposal, the time I started working with the community, I have learned a lot and gained a lot of real-world experience of how things are done. Learning from people who know way more than me, discussing with them and making the plan of action have been the most joyful parts for me. I feel an urge to keep contributing and learning from this community. Further are the reasons why I think I am the best fit for this project -

- I have made extensive contributions to the vue-simulator and in total I have the **most commits, active PRs and collaborations** than any other current contributor - total 50 and merged 15. I am currently **ranked top 6** in the vue simulator repository
- I have been responsible about my work and managed to help fellow contributors on every occasion I could, which led me to have so many great friends from all across the Globe.

- I became a member of the community (link) under issue triaging team, and ever since then have reviewed countless bugs and many PRs. I have helped many contributors get their first issues assigned and raise their first PRs.
- I have successfully managed to collaborate in hosting the weekly meetings for the last 6 months
- I have learned a lot of tech skills from my time here, like **Ruby on Rails**, **Tauri**, got infatuated with **Vue, Vite and Evan You**. Got to learn tons about CI/CD and experimented with all the possible options to understand the best approach for the proposal
- Other than coding skills, I also got to learn a lot about **Documentation** while researching for my documentation PRs. Learned how documentation helps you to talk to the past contributors and know their thought processes.
- I am a **Team Player** and have always taken responsibility for my work, and from everything I have learned up to now, I would be grateful to take more responsibility and contribute more towards the community.
- For that reason, I researched extensively for the proposal and tried to mention every relevant link for the implementation. I have tried to provide as much information I could provide while being as discrete as possible.

Past work:

- Developed a comprehensive school website enabling seamless online fee payment, enhancing accessibility and administrative efficiency. Integrated secure payment workflows with a user-friendly interface to streamline financial transactions for students and parents. (Link). More such works are mentioned in my CV.

Currently Using

OS - Ubuntu, Mac **Code-editor** - Vim(Pre-intermediate), VS code

Project Size and Timeline Selection

- **Medium (175 hours)**

I believe the project timeline is valid and justifiable to accomplish the project goals.

By setting the project length to 175 hours, we now get the space to work on the targeted project goals along with spending some time writing tests, adding additional features to the Tauri App, working on documenting the progress so far, and consistently working to improve the code quality and readability. I feel this timeline is the exact time that would be required to complete the project.