FOSSEE Spring Internship Report

On

Development of 3psLCCA-web Application for Osdag

Submitted by

Ayushman Gupta

Intern,

FOSSEE Semester Long Internship 2026

3psLCCA-web Development

Mumbai

Under the Guidance of

Prof. Siddhartha Ghosh

Department of Civil Engineering

Indian Institute of Technology Bombay

Mentors:

Ajmal Babu M S

Parth Karia

Ajinkya Dahale

May 19, 2026

Acknowledgments

- Start with a general statement of thanks. Express your overall gratitude to everyone who supported you during your project or research.
- Project staff at the Osdag team, Ajmal Babu M. S., Ajinkya Dahale, and Parth Karia,
- Osdag Principal Investigator (PI) Prof. Siddhartha Ghosh, Department of Civil Engineering at IIT Bombay
- FOSSEE PI Prof. Kannan M. Moudgalya, FOSSEE Project Investigator, Department of Chemical Engineering, IIT Bombay
- FOSSEE Managers Usha Viswanathan and Vineeta Parmar and their entire team
- Acknowledge the support from the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education (MoE), Government of India, for their role in facilitating this project
- Acknowledge your colleagues who worked with you during your internship or project.
- If appropriate, thank your college, department, head, and principal for their support during your studies.

Contents

Chapter 1

Introduction

1.1 National Mission in Education through ICT

The National Mission on Education through ICT (NMEICT) is a scheme under the Department of Higher Education, Ministry of Education, Government of India. It aims to leverage the potential of ICT to enhance teaching and learning in Higher Education Institutions in an anytime-anywhere mode.

The mission aligns with the three cardinal principles of the Education Policy—**access, equity, and quality**—by:

- Providing connectivity and affordable access devices for learners and institutions.
- Generating high-quality e-content free of cost.

NMEICT seeks to bridge the digital divide by empowering learners and teachers in urban and rural areas, fostering inclusivity in the knowledge economy. Key focus areas include:

- Development of e-learning pedagogies and virtual laboratories.
- Online testing, certification, and mentorship through accessible platforms like EduSAT and DTH.
- Training and empowering teachers to adopt ICT-based teaching methods.

For further details, visit the official website: www.nmeict.ac.in.

1.1.1 ICT Initiatives of MoE

The Ministry of Education (MoE) has launched several ICT initiatives aimed at students, researchers, and institutions. The table below summarizes the key details:

No.	Resource	For Students/Researchers	For Institutions
Audio-Video e-content			
1	SWAYAM	Earn credit via online courses	Develop and host courses; accept credits
2	SWAYAMPRABHA	Access 24x7 TV programs	Enable SWAYAMPRABHA viewing facilities
Digital Content Access			
3	National Digital Library	Access e-content in multiple disciplines	List e-content; form NDL Clubs
4	e-PG Pathshala	Access free books and e-content	Host e-books
5	Shodhganga	Access Indian research theses	List institutional theses
6	e-ShodhSindhu	Access full-text e-resources	Access e-resources for institutions
Hands-on Learning			
7	e-Yantra	Hands-on embedded systems training	Create e-Yantra labs with IIT Bombay
8	FOSSEE	Volunteer for open-source software	Run labs with open-source software
9	Spoken Tutorial	Learn IT skills via tutorials	Provide self-learning IT content
10	Virtual Labs	Perform online experiments	Develop curriculum-based experiments
E-Governance			
11	SAMARTH ERP	Manage student lifecycle digitally	Enable institutional e-governance
Tracking and Research Tools			
12	VIDWAN	Register and access experts	Monitor faculty research outcomes
13	Shodh Shuddhi	Ensure plagiarism-free work	Improve research quality and reputation
14	Academic Bank of Credits	Store and transfer credits	Facilitate credit redemption

Table 1.1: Summary of ICT Initiatives by the Ministry of Education

1.2 FOSSEE Project

The FOSSEE (Free/Libre and Open Source Software for Education) project promotes the use of FLOSS tools in academia and research. It is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India.

1.2.1 Projects and Activities

The FOSSEE Project supports the use of various FLOSS tools to enhance education and research. Key activities include:

- **Textbook Companion**: Porting solved examples from textbooks using FLOSS.
- **Lab Migration**: Facilitating the migration of proprietary labs to FLOSS alternatives.
- **Niche Software Activities**: Specialized activities to promote niche software tools.
- **Forums**: Providing a collaborative space for users.
- **Workshops and Conferences**: Organizing events to train and inform users.

1.2.2 Fellowships

FOSSEE offers various internship and fellowship opportunities for students:

- Winter Internship
- Summer Fellowship
- Semester-Long Internship

Students from any degree and academic stage can apply for these internships. Selection is based on the completion of screening tasks involving programming, scientific computing, or data collection that benefit the FLOSS community. These tasks are designed to be completed within a week.

For more details, visit the official FOSSEE website.

Figure 1.1: FOSSEE Projects and Activities

1.3 Osdag Software

Osdag (Open steel design and graphics) is a cross-platform, free/libre and open-source software designed for the detailing and design of steel structures based on the Indian Standard IS 800:2007. It allows users to design steel connections, members, and systems through an interactive graphical user interface (GUI) and provides 3D visualizations of designed components. The software enables easy export of CAD models to drafting tools for construction/fabrication drawings, with optimized designs following industry best practices [1, 2, 3]. Built on Python and several Python-based FLOSS tools (e.g., PyQt and PythonOCC), Osdag is licensed under the GNU Lesser General Public License (LGPL) Version 3.

1.3.1 Osdag GUI

The Osdag GUI is designed to be user-friendly and interactive. It consists of

- **Input Dock**: Collects and validates user inputs.
- **Output Dock**: Displays design results after validation.
- **CAD Window**: Displays the 3D CAD model, where users can pan, zoom, and rotate the design.
- **Message Log**: Shows errors, warnings, and suggestions based on design checks.

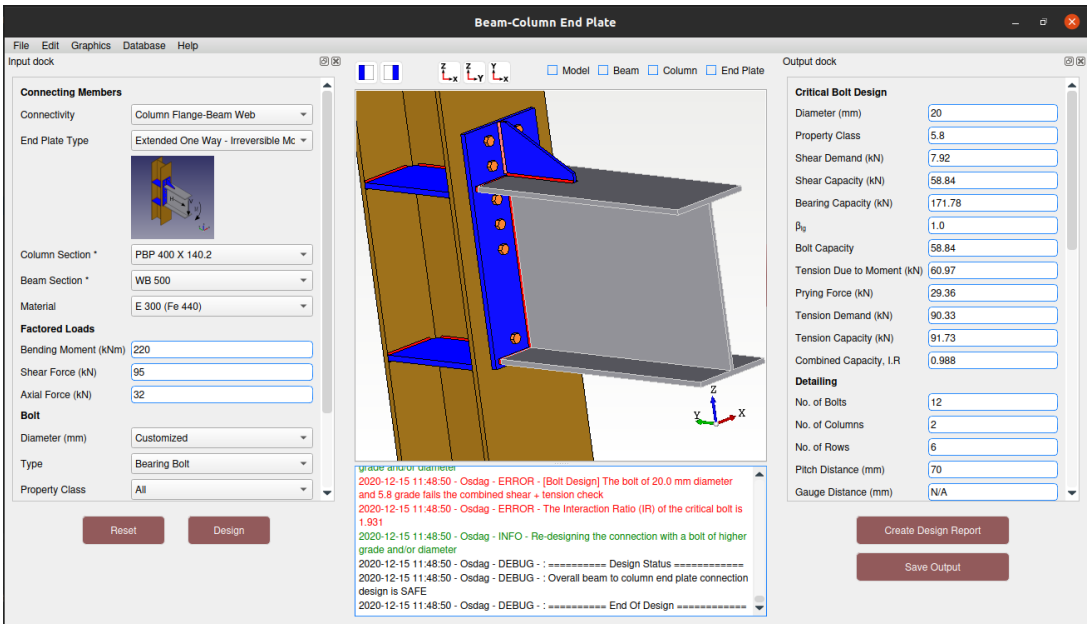

Figure 1.2: Osdag GUI

1.3.2 Features

- **CAD Model**: The 3D CAD model is color-coded and can be saved in multiple formats such as IGS, STL, and STEP.
- **Design Preferences**: Customizes the design process, with advanced users able to set preferences for bolts, welds, and detailing.
- **Design Report**: Creates a detailed report in PDF format, summarizing all checks, calculations, and design details, including any discrepancies.

For more details, visit the official Osdag website.

Chapter 2

Screening Task

2.1 Problem Statement

The objective of the screening task was to develop a desktop-based Graphical User Interface (GUI) titled **Group Design** for the Bridge Module in Osdag. The application was required to handle user inputs, perform validation checks, manage field dependencies, and load reference values from engineering data tables for temperature, wind, and seismic data.

Key requirements included:

- Designing an intuitive layout with a **Basic Inputs** tab containing fields for Structure Type, Project Location, Geometric Inputs, and Material Inputs.
- Displaying a non-interactive Bridge Cross-Section reference image.
- Managing interdependencies between geometric fields such as *Girder Spacing*, *No. of Girders*, and *Deck Overhang Width* through a Modify Additional Geometry pop-up, ensuring mathematical consistency.
- Implementing validation rules for constraints (e.g., minimum Carriageway Width, Span limits, Skew Angle checks).
- Handling dynamic project location data either by manually tabulating custom loading parameters or by creating an integrated database for automatic fetching of temperature, wind speed, and seismic data.

2.2 Tasks Done

The following tasks were successfully performed to meet the objectives of the screening assignment (source code available at GitHub Repository):

1. **UI Layout Design:** Developed a clean, responsive interface featuring input panels on the left and a persistent Bridge Cross-Section reference image on the right. Included a placeholder for *Additional Inputs*.

2. **Input Validation and Constraints:** Implemented strict validation checks for parameters such as span lengths ($20 \leq \text{Span} \leq 45$), carriageway width limits, and skew angle warnings, displaying clear error feedback to the user.

3. **Dynamic Geometry Synchronization:** Created the *Modify Additional Geometry* pop-up dialog. Programmed the logic to automatically recalculate and adjust interdependent fields (Overall Width, Girder Spacing, No. of Girders, and Deck Overhang) in real-time, enforcing the relationship: No. of Girders = (Overall Width − Overhang)/Spacing.

4. **Location Data Integration:** Designed the Project Location selection using mutually exclusive modes. Implemented the logic to retrieve basic wind speed, seismic zone factors, and shade air temperatures from external reference tables based on the selected state and district.

5. **Code Organization:** Ensured the codebase was structured for clarity, maintainability, and reusability, supplemented by a comprehensive README file detailing installation and usage instructions.

2.3 Screenshots

The following screenshots illustrate the developed application interface:

Figure 2.1: Welcome Screen of OSDAG Group Design Module

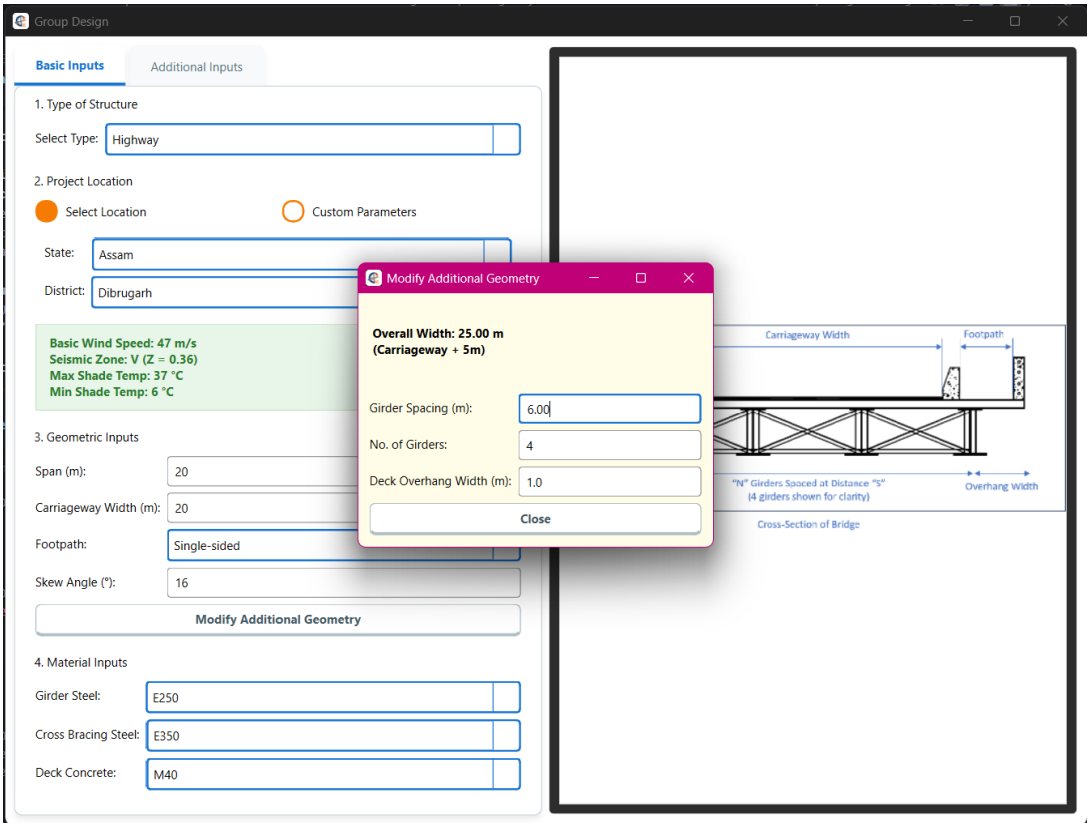

Figure 2.2: Main Input UI with Modify Additional Geometry Pop-Up

Chapter 3

Internship Task: 3psLCCA-web Development

3.1 Task 1: Problem Statement

The primary objective of the internship was to develop the frontend of the **3psLCCA-web** application — a browser-based Life Cycle Cost Analysis (LCCA) tool for steel bridges, integrated within the Osdag project ecosystem. The goal was to design and implement a multi-panel, data-driven React application capable of collecting engineering input data from across the bridge's entire life cycle, then performing and presenting results of financial analysis.

The application needed to support:

- Multi-section data collection across distinct **life-cycle phases**: Construction, Maintenance, Traffic, Demolition, Recycling, and Financial Parameters.
- A robust **global state management** system using React Context API, with **browser cache (localStorage)** persistence so project data is not lost on page refresh.
- A **dynamic theming system** supporting multiple named light and dark colour palettes, responding both to manual user selection and to the operating system's dark-mode preference.
- Component-level **input validation** with actionable error feedback and field-level

error highlighting.

- A consistent and accessible UI design using **Bootstrap 5** and CSS custom properties (CSS variables).

3.2 Task 2: Architecture and General Information

3.2.1 Technology Stack

The application is built using the following technologies:

Technology	**Purpose**
React 19 (Vite)	Component-based UI rendering and state management
Bootstrap 5 + react-bootstrap	Layout, forms, modals, and responsive design
CSS Custom Properties	Dynamic theming (light/dark/named palettes)
localStorage (Browser Cache)	Persistence of project state across page reloads
React Context API	Global state sharing between components
Recharts	Data visualisation for output charts
jsPDF + html2canvas	PDF export of analysis results

Table 3.1: Technology Stack of 3psLCCA-web

3.2.2 Directory Structure

The source code is organised as follows:

- src
  - App.jsx (Root: routing, theming, browser cache)
  - contexts/
    - ProjectDataContext.jsx (Global state + localStorage)
  - gui/components/
    - bridgedata/ (Bridge identification & specs)
    - constructionworkdata/ (Material tables by structure type)
    - maintenance_and_repair/ (Maintenance cost parameters)
    - trafficdata/ (Traffic mode & road user costs)
    - recycling/ (Material recyclability tables)
    - demolition/ (End-of-life cost parameters)
    - financialdata/ (Discount/inflation/interest rates)
    - carbon_emission/ (Carbon emission inputs)
    - outputs/ (Results, charts, PDF export)

3.2.3 Application Flow

The application follows a linear navigation model. On launch, the user logs in, then opens or creates a project. The `App.jsx` component maintains a `CONTENT_MAP` that maps each navigation node name to its corresponding React component. The active panel is determined by an `activeNode` string. All routing is in-memory — there are no URL changes; navigation state itself is stored in `localStorage`.

Listing 3.1: Navigation state persisted via localStorage (App.jsx)

```
const [activeNode, setActiveNode] = useState(
    () => localStorage.getItem('activeNode') || 'General Information'
);
useEffect(() => {
    localStorage.setItem('activeNode', activeNode);
}, [activeNode]);
```

3.3 Task 3: Global State and Browser Cache Implementation

3.3.1 Problem Statement

Without persistent state, every time the user navigates between panels or refreshes the browser, all entered data is lost. The application required a mechanism to retain project data across navigation events and browser reloads.

3.3.2 Approach: ProjectDataContext with localStorage

The `ProjectDataContext.jsx` provides a React Context that holds the entire project's data model as a single JSON object. On mount, it loads from `localStorage`; on every state update, it writes back to `localStorage`.

Listing 3.2: ProjectDataContext — load from and sync to localStorage

```
const ProjectDataContext = createContext(null);

export const ProjectDataProvider = ({ projectId, children }) => {
  const storageKey = `projectData_${projectId}`;

  const [projectData, setProjectData] = useState(() => {
    const saved = localStorage.getItem(storageKey);
    try { return saved ? JSON.parse(saved) : {}; }
    catch (e) { return {}; }
  });

  // Write-through: keep cache in sync with state
  useEffect(() => {
    localStorage.setItem(storageKey, JSON.stringify(projectData));
  }, [projectData, storageKey]);

  const updateProjectData = useCallback((section, data) => {
    setProjectData(prev => ({ ...prev, [section]: data }));
  }, []);

  return (
```

```
22      <ProjectDataContext.Provider value={{ projectData,
            updateProjectData }}>
23        {children}
24      </ProjectDataContext.Provider>
25    );
26  };
```

Each component reads its section from context on mount and writes back on every form change. This ensures:

- Data survives panel navigation (no re-fetch needed).
- Data survives browser refresh.
- Different projects are isolated by their `projectId` key.

3.4 Task 4: Theming System (Light / Dark Mode)

3.4.1 Problem Statement

The application needed to support multiple named colour palettes in both light and dark modes, respond to the operating system's appearance preference, and let users override this manually in the Settings panel.

3.4.2 Design Approach

All components reference CSS custom properties (variables) such as `--app-bg-main`, `--app-primary-accent`, `--app-text-primary`, etc. The actual values of these variables are injected at runtime on `document.documentElement` by `App.jsx`, based on the active theme.

Listing 3.3: Runtime CSS variable injection for theming (App.jsx excerpt)

```
1  useEffect(() => {
2    const root = document.documentElement;
3    root.setAttribute('data-bs-theme', isDarkMode ? 'dark' : 'light');
4
5    if (isDarkMode) {
6      if (userSettings.darkTheme === 'Dracula') {
```

```
      root.style.setProperty('--app-bg-main', '#282A36');
      root.style.setProperty('--app-primary-accent', '#BD93F9');
      // ...additional colour tokens
    }
  } else {
    if (userSettings.lightTheme === 'standard light') {
      root.style.setProperty('--app-bg-main', '#F8FAFC');
      root.style.setProperty('--app-primary-accent', '#2563EB');
    }
  }
}, [isDarkMode, userSettings]);
```

Mode	Theme Name	Accent / Background
Dark	Dracula	Purple accent (`#BD93F9`) on deep dark background
Dark	Neon City	Magenta accent (`#E93CFF`) on near-black background
Dark	Standard Dark	Blue accent (`#3B82F6`) on slate background
Light	Standard Light	Blue accent (`#2563EB`) on white/grey background
Light	Soft Pink	Pink accent (`#D94680`) on pink-tinted background
Light	Soft Light	Green accent (`#86A022`) on off-white background

Table 3.2: Available Themes in 3psLCCA-web

The OS-level dark mode preference is detected via the `window.matchMedia` API and kept in sync with a `MediaQueryList` change listener. When the user sets *Appearance Mode* to `Auto(follow os)`, the theme responds automatically.

3.5 Task 5: Bridge Data Component

3.5.1 Problem Statement

The first data-entry panel required collecting general bridge identification and technical specification data, providing inline hints, validating required fields, and syncing all values

into the global project context.

3.5.2 Design and Implementation

File: `src/gui/components/bridgedata/BridgeData.jsx`

The component is built from four atomic sub-components: `TextField`, `SelectField`, `NumberField`, and `SectionHeader`. All sub-components receive their props declaratively, making them reusable. The `SelectField` implements a custom dropdown (not a native `<select>`), built with a `useRef` click-outside listener to close the menu when the user clicks away.

Field	Type	Notes
Bridge Name	Text	Required
Owner / Agency	Text	Required
Location Country	Select	Derived from `countriesdata` utility
Address, From, Via, To	Text	Optional location descriptors
Type of Bridge	Select	Girder, Arch, Cable-Stayed, etc.
Span	Number	Required, unit: m
Number of Lanes	Number	Required, 0–20
Footpath	Select	Yes / No
Wind Speed	Number	Required, unit: m/s
Carriageway Width	Number	Required, unit: m
Year of Construction	Number	Required, 1900–2200
Design Life / Service Life	Number	Required, unit: years

Table 3.3: Bridge Data Form Fields

Validation is triggered by the parent controller (or can be called externally). Missing required fields are highlighted using Bootstrap's `is-invalid` class and listed in a consolidated `alert-danger` banner.

Listing 3.4: Validation logic in BridgeData.jsx

```
1 const validate = () => {
2     const newErrors = new Set();
3     const missing = [];
4     REQUIRED_KEYS.forEach((key) => {
```

```
        const val = form[key];
        if (val === '' || val === null || val === undefined) {
            newErrors.add(key);
            missing.push(key.replace(/_/g, ' '));
        }
    });
    setErrors(newErrors);
    if (newErrors.size > 0) {
        setValidationMsg(`Missing required bridge data: ${missing.join(
            ', ')}`);
        return { valid: false };
    }
    return { valid: true };
};
```

3.6 Task 6: Construction Work Data Component

3.6.1 Problem Statement

Construction cost data is spread across four distinct structural categories. Each category contains a list of materials with quantities, unit rates, and computed costs. The component needed to support switching between categories without losing data, and to allow adding, editing, and removing material line items.

3.6.2 Design and Implementation

Files: `ConstructionWorkData.jsx`, `MaterialTable.jsx`, `MaterialAddModal.jsx`, and sub-folders `foundation/`, `substructure/`, `superstructure/`, `miscellaneous/`

The `ConstructionWorkData` component acts as a **tab controller**. The `TABS` array maps keys to their respective sub-components:

Listing 3.5: Tab configuration in ConstructionWorkData.jsx

```
const TABS = [
    { key: 'Foundation',     label: 'Foundation',     component:
        Foundation },
    { key: 'SuperStructure', label: 'Super Structure', component:
        SuperStructure },
```

```
    { key: 'SubStructure',    label: 'Sub Structure',   component:
        SubStructure },
    { key: 'Miscellaneous',   label: 'Miscellaneous',   component:
        Miscellaneous },
];
```

The active tab is controlled via `initialTab` prop from `App.jsx`, meaning the sidebar navigation can deep-link directly into any structural sub-tab. Each sub-component renders a `MaterialTable` which lists materials with columns for name, quantity, unit rate, and total cost. A `MaterialAddModal` (Bootstrap modal) is used to add or edit items.

3.7 Task 7: Maintenance and Repair Component

3.7.1 Problem Statement

Maintenance parameters consist of multiple sections (Routine, Periodic, Major Works, Bearings & Expansion Joints), each containing cost, frequency, and duration fields. The component also needed to provide pre-filled *suggested values* typical for Indian highway bridges.

3.7.2 Design and Implementation

File: `src/gui/components/maintenance_and_repair/MaintenanceAndRepair.jsx`

All field definitions are stored declaratively in the `MAINTENANCE_SECTIONS` constant — an array of section objects, each containing a `fields` array. This data-driven design means adding a new field requires only updating the constant, not adding new JSX.

Listing 3.6: Data-driven field definition pattern (MaintenanceAndRepair.jsx)

```
const MAINTENANCE_SECTIONS = [
  {
    title: "Routine Maintenance",
    fields: [
      { key: 'routine_inspection_cost', label: 'Routine Inspection Cost
          ',
        type: 'float', min: 0.0, max: 100.0, step: 0.001,
        unit: '(% of initial construction cost)', required: true },
```

```
      { key: 'routine_inspection_freq', label: 'Routine Inspection
         Frequency',
        type: 'int', min: 0, max: 50, unit: '(yr)', required: true }
    ]
  },
  // ... additional sections
];

const SUGGESTED_VALUES = {
    routine_inspection_cost: 0.1,
    routine_inspection_freq: 1,
    periodic_maintenance_cost: 0.55,
    // ...
};
```

The *Load Suggested Values* button merges the `SUGGESTED_VALUES` into the form state without overwriting non-overlapping fields:

Listing 3.7: Applying suggested values without full reset

```
const handleLoadSuggested = () => {
    setForm(prev => ({
        ...prev,
        ...Object.fromEntries(
            Object.entries(SUGGESTED_VALUES).map(([k, v]) => [k, String
               (v)])
        )
    }));
};
```

3.8 Task 8: Traffic Data Component

3.8.1 Problem Statement

Traffic data required a calculation mode selector, a road user cost input, and a rich text notes field. The rich text editor needed to support bold, italic, lists, tables, and table manipulation (add row/column), without introducing a heavy third-party library.

3.8.2 Design and Implementation

File: src/gui/components/trafficdata/TrafficData.jsx

The **rich text editor** is implemented using the browser's `contentEditable` API combined with `document.execCommand` for text formatting and custom DOM manipulation for table row/column insertion. The toolbar is defined as a static `TOOLBAR` array, where `null` entries are rendered as visual separators.

Listing 3.8: Rich text editor toolbar definition (TrafficData.jsx)

```
const TOOLBAR = [
    { label: 'B', title: 'Bold', action: () => exec('bold') },
    { label: 'I', title: 'Italic', action: () => exec('italic') },
    null, // separator
    { label: '    List', action: () => exec('insertUnorderedList') },
    { label: '+ Table', action: insertTable },
    { label: '+ Row',   action: insertRow },
    { label: '+ Col',   action: insertCol },
];
```

The custom `Dropdown` sub-component (same click-outside pattern as in `BridgeData`) is used for the *Calculation Mode* selector with two options: `GLOBAL` and `LOCAL`.

3.9 Task 9: Demolition Component

3.9.1 Problem Statement

The end-of-life demolition panel needed to capture demolition costs (as a percentage of initial construction cost), the associated carbon cost, duration, and demolition method.

3.9.2 Design and Implementation

File: src/gui/components/demolition/Demolition.jsx

The `Demolition` component follows the same data-driven pattern as `MaintenanceAndRepair`. A `DEMOLITION_SECTIONS` array defines all fields. The `InputField` sub-component handles both numeric inputs and `select` dropdowns in a single generic component by branching on the `field.type` property.

Field	Type	Notes
Demolition & Disposal Cost	Float %	Percentage of initial construction cost
Demolition Carbon Cost	Float %	Percentage of initial carbon cost
Demolition Duration	Integer	In months, 0–120
Demolition Method	Select	Implosion, Mechanical, Deconstruction, Wrecking Ball

Table 3.4: Demolition Data Fields

Suggested values: Cost = 10%, Carbon Cost = 10%, Duration = 1 month, Method = Implosion.

3.10 Task 10: Financial Data Component

3.10.1 Problem Statement

Financial parameters drive the entire LCCA computation. These include time-value-of-money rates (discount, inflation, interest), the investment ratio, and the analysis time horizon.

3.10.2 Design and Implementation

File: `src/gui/components/financialdata/FinancialData.jsx`

Like `BridgeData`, this component synchronises directly with `ProjectDataContext`. On mount, it reads the saved state from the context (which was loaded from `localStorage`); on every change, it writes back.

Field	Suggested Value	Description
Discount Rate	6.70%	Converts future costs to present value
Inflation Rate	5.15%	Annual price increase
Interest Rate	7.75%	Cost of borrowing capital
Investment Ratio	0.5	Fraction of cost financed by investment
Design Life	50 years	Structural design lifetime
Analysis Period	50 years	Time horizon for LCCA

Table 3.5: Financial Data Fields and Suggested Values

3.11 Task 11: Recycling Component

3.11.1 Problem Statement

At end-of-life, certain materials (primarily structural steel) can be sold as scrap, recovering economic value. The Recycling panel displays all construction materials, split into two groups: those *included* in recyclability calculations and those *excluded*. Users must be able to move items between groups and edit recyclability percentages and scrap rates.

3.11.2 Design and Implementation

Files: `Recycling.jsx`, `EditRecyclabilityModal.jsx`

Two separate `useState` arrays maintain `included` and `excluded` items. Clicking the *Exclude* (down-chevron) action removes the item from `included` and prepends it to `excluded`, and vice-versa for *Include* (up-chevron). The `EditRecyclabilityModal` allows editing the recyclability percentage and scrap rate of any individual line item.

Listing 3.9: Include/Exclude logic in Recycling.jsx

```
1 const handleExclude = (item) => {
2     setIncluded(prev => prev.filter(i => i.id !== item.id));
3     setExcluded(prev => [...prev, item]);
4 };
5
6 const handleInclude = (item) => {
```

```
7       setExcluded(prev => prev.filter(i => i.id !== item.id));
8       setIncluded(prev => [...prev, item]);
9   };
```

3.12 Task 12: Component Design Patterns and Documentation

3.12.1 Shared Design Patterns

All form-based components follow a consistent pattern summarised below. This makes onboarding new contributors significantly easier.

Pattern	Description
`INITIAL_STATE` constant	A frozen object with all field keys set to empty string `''`. Used to reset the form to a clean state.
`REQUIRED_KEYS` Set	A `Set` of field key strings. Used by `validate()` to find missing required fields.
`useCallback` on handlers	Prevents `handleChange` from being re-created on every render. Safe to pass as a dependency.
Immutable error updates	Uses `prev => const next = new Set(prev); next.delete(key); return next;` to create a new Set without mutating the previous one.
`Load Suggested Values`	Merges preset engineering values into the form using object spread. Does not reset fields absent from the preset.
`Clear All`	Resets form to `INITIAL_STATE` and clears errors and the validation message banner.
CSS variable styling	All colour references use `var(--app-*)` tokens so the UI automatically adapts to any theme without component-level changes.

Table 3.6: Shared Component Design Patterns

3.12.2 Adding a New Section (Guide for the Next Intern)

To add a completely new data-entry section to the application:

1. Create a new folder under `src/gui/components/mysection/` with `MySection.jsx`.

2. Define `MY_FIELDS`, `INITIAL_STATE`, and `REQUIRED_KEYS` following the pattern from `FinancialData.jsx`.

3. Use `useProjectData()` from context to read/write `projectData.my_section`.

4. Import and add the component to `CONTENT_MAP` in `App.jsx` with an appropriate key.

5. Add the new key to the sidebar navigation tree in `ProjectLayout.jsx`.

3.12.3 Key Exported Symbols

Each component exports the following symbols for use by the parent controller (validation orchestration):

Listing 3.10: Named exports pattern

```
// Example from FinancialData.jsx
export { FinancialData as default };
export { REQUIRED_KEYS, INITIAL_STATE, SUGGESTED_VALUES };
```

These allow a top-level controller to call `validate()` on each panel and aggregate errors before proceeding to calculation.

Chapter 4

Conclusions

4.1 Tasks Accomplished

The FOSSEE Spring 2026 Internship involved designing and developing significant portions of the **3psLCCA-web** frontend application — a browser-based Life Cycle Cost Analysis (LCCA) tool for steel bridges within the Osdag ecosystem. The following tasks were successfully completed during the internship period:

1. **Screening Task — OSDAG Bridge Module GUI:** Developed a fully functional desktop GUI titled *Group Design* for the Osdag Bridge Module. This included implementing two mutually exclusive project location modes (state/district dropdown with database-backed automatic retrieval of wind, seismic, and temperature data, and a manual tabulation spreadsheet-style pop-up), geometric input validation, material grade selectors, and the *Modify Additional Geometry* dialogue with live interdependency constraints between girder spacing, number of girders, and deck overhang width.

2. **Bridge Data Component:** Implemented the `BridgeData.jsx` panel with structured sections for bridge identification, location, technical specifications, timeline, and life cycle parameters. Included a custom accessible dropdown component with click-outside dismissal and per-field inline validation.

3. **Construction Work Data:** Developed a tabbed data entry system across four structural categories (Foundation, Sub Structure, Super Structure, Miscellaneous).

Each tab hosts a material table with add/edit/delete capability via `MaterialAddModal.jsx`.

4. **Maintenance and Repair Component:** Built a data-driven, section-based form covering Routine Maintenance, Periodic Maintenance, Major Works, and Bearings & Expansion Joints. Implemented *Load Suggested Values* functionality pre-populated with standard Indian highway bridge engineering defaults.

5. **Traffic Data Component:** Developed the traffic panel featuring a calculation mode selector and a custom browser-native rich text editor (bold, italic, lists, table insertion/row/column manipulation) implemented without any third-party library, using the `contentEditable` and `execCommand` APIs.

6. **Demolition Component:** Created the end-of-life panel capturing demolition and disposal costs, carbon costs (as percentages of initial values), duration, and demolition method, following the generic `InputField` pattern that handles both numeric and select input types.

7. **Financial Data Component:** Implemented the financial parameters panel (discount rate, inflation rate, interest rate, investment ratio, design life, analysis period) with context synchronisation and pre-filled suggested values aligned with Indian financial parameters.

8. **Recycling Component:** Built a dual-table interface displaying materials included and excluded from recyclability calculations. Implemented move-between-groups actions and an `EditRecyclabilityModal` for modifying recyclability percentages and scrap rates.

9. **Global State and Browser Cache:** Implemented `ProjectDataContext.jsx` using React Context API with full `localStorage` persistence. Project data (per unique project ID) is loaded from cache on mount and written back on every state change, enabling seamless session recovery after browser refresh.

10. **Dynamic Theming System:** Developed a runtime CSS custom property injection system in `App.jsx` supporting three dark themes (Dracula, Neon City, Standard Dark) and three light themes (Standard Light, Soft Pink, Soft Light), with

automatic OS dark-mode detection via `window.matchMedia` and user-overridable settings persisted in `localStorage`.

11. **Component Architecture Standardisation:** Established and documented a consistent design pattern across all form components — covering `INITIAL_STATE`, `REQUIRED_KEYS`, `useCallback`-memoised handlers, immutable error state updates, and CSS variable theming — enabling rapid onboarding of future contributors.

4.2 Skills Developed

The internship provided comprehensive exposure to both technical and professional competencies in modern web application development:

4.2.1 Technical Skills

- **React.js (v19):** Deepened understanding of functional components, hooks (`useState`, `useEffect`, `useCallback`, `useRef`, `useContext`), and the Rules of Hooks. Gained practical experience resolving common pitfalls such as stale closures, illegal state updates during render, and infinite re-render loops caused by missing or incorrect `useEffect` dependency arrays.

- **React Context API:** Learned to design scalable global state with the Context + `useReducer`/`useState` pattern, and to export stable, memoised update functions to avoid unnecessary re-renders in consumer components.

- **Browser Cache (localStorage):** Gained hands-on experience with `localStorage` as an application-level data persistence layer, including safe JSON serialisation/deserialisation with error handling, per-project data isolation using dynamic storage keys, and write-through caching patterns.

- **CSS Custom Properties and Theming:** Learned to build a fully dynamic theming system using CSS variables injected at the `:root` level, enabling all components to respond to theme switches without any JSX changes. Gained experience integrating this with Bootstrap 5's `data-bs-theme` attribute.

- **Bootstrap 5 and react-bootstrap:** Applied utility-class-driven layout, responsive grid, modals, form controls, and input groups. Learned the Zero Visual Change refactoring practice of migrating inline styles to utility classes without introducing regressions.

- **Data-Driven UI Design:** Practiced defining UI structure as data (arrays of field descriptors) rather than hard-coded JSX, producing components that require no markup changes when fields are added or removed.

- **Input Validation Patterns:** Implemented multi-field form validation using a `Set`-based error tracking approach that supports per-field error highlighting (`is-invalid`) and a consolidated validation message banner, with clear-on-edit behaviour.

- **DOM APIs:** Gained practical experience with `contentEditable`, `execCommand`, `window.getSelection`, and `window.matchMedia` — browser-native APIs that are rarely covered in standard tutorials.

- **Git and Version Control:** Practised disciplined commit history management, resolving non-fast-forward push errors, handling merge conflicts, and reverting to previous commits when necessary.

- **LaTeX Report Writing:** Developed the skill of producing a structured technical report in LaTeX, including tables, code listings with the `listings` package, directory trees with the `forest` package, and cross-references.

4.2.2 Professional Skills

- **Reading and Understanding Existing Codebases:** Developed the ability to navigate and understand a large, multi-file React codebase without prior knowledge, identifying architectural patterns and component responsibilities quickly.

- **Technical Documentation:** Gained experience writing developer-facing documentation that explains design decisions, patterns, and onboarding guides for future contributors — a critical skill for open-source collaborative projects.

- **Problem Decomposition:** Practised breaking down complex UI requirements (e.g., the rich text editor, the geometry auto-adjust dialogue) into independent, testable sub-problems before implementation.

- **Open-Source Collaboration:** Gained familiarity with the FOSSEE and Osdag project ecosystems, their goals, workflows, and contribution standards. Understood the importance of code readability and maintainability in a long-lived open-source project.

Chapter A

Daily Work Report

The following is a day-by-day record of tasks and progress during the internship period from February 16, 2025, to May 15, 2025.

Week	**Date**	**Tasks Completed**
Week 1	16/02/2025	Attended Osdag Spring SLI Onboarding Meeting 2026.
Week 1	17/02/2025	Joined the LCCA team.
Week 1	18/02/2025	Attended LCCA team intro; discussed project details, Standard Operating Rates (SOR), editing features, project structure, git policies (rebasing), and assigned role for desktop/web application.
Week 1	19/02/2025	Went through the LCCA Report documentation.
Week 1	20/02/2025	Continued reviewing project documentation and existing codebase structure.
Week 2	23/02/2025	Set up local environment for contribution.
Week 2	24/02/2025	Attended meeting; researched local storage, session storage, auto-save limits, and PWA integration for local data retention.
Week 2	25/02/2025	Prepared preliminary research report on storage solutions.

Continued on next page

Week	Date	Tasks Completed
Week 2	26/02/2025	Reviewed report flaws with Swastik Sir; started extensive research on Django/FastAPI, Local/Session storage vs Cookies, Firebase/Supabase, Appwrite, and CORS policies.
Week 2	27/02/2025	Evaluated Cloudflare and NoSQL alternatives; designed end-to-end storage architecture for the application.
Week 2	01/03/2025	Reported UI changes in submission group; addressed padding and margin issues.
Week 3	02/03/2025	Presented the updated storage architecture and data flow plans to the team.
Week 3	03/03/2025	Initialized basic UI component structure for the React web application.
Week 3	04/03/2025	Set up application routing and page navigation layouts.
Week 3	05/03/2025	Developed the `ProjectNavbar` component and responsive header.
Week 3	06/03/2025	Implemented initial responsive grid layouts for the main dashboard.
Week 4	09/03/2025	Began development of the `FinancialData` UI component.
Week 4	10/03/2025	Structured data input forms for standard financial parameters.
Week 4	11/03/2025	Integrated React state management for handling financial form inputs.
Week 4	12/03/2025	Added validation logic and error handling for numeric data entry.
Week 4	13/03/2025	Improved UI consistency and implemented informative tooltips for financial terms.
Week 5	16/03/2025	Started work on the Carbon Emission module (`MaterialEmissions.jsx`).

Continued on next page

Week	Date	Tasks Completed
Week 5	17/03/2025	Designed and drafted the Material Add Modal interface.
Week 5	18/03/2025	Implemented dynamic calculation logic for tracking material emissions.
Week 5	19/03/2025	Formatted the emissions results table for better readability.
Week 5	20/03/2025	Refactored overlapping input styles into shared reusable components.
Week 6	23/03/2025	Developed UI for the `TrafficEmissions` component.
Week 6	24/03/2025	Handled complex user input states for diverse traffic scenarios.
Week 6	25/03/2025	Integrated local storage APIs to save draft project data securely.
Week 6	26/03/2025	Tested local storage capacity limits, edge cases, and cache clearing behaviors.
Week 6	27/03/2025	Optimized auto-save functionality to run smoothly without affecting UI performance.
Week 7	30/03/2025	Integrated global `ProjectDataContext` across the application.
Week 7	31/03/2025	Debugged data propagation delays between `FinancialData` and `MaterialEmissions`.
Week 7	01/04/2025	Created utility functions to compute complex LCCA formulas efficiently.
Week 7	02/04/2025	Collaborated with the team to review API integration strategies.
Week 7	03/04/2025	Enhanced visual design (typography, colors) based on peer feedback.
Week 8	06/04/2025	Migrated legacy inline CSS to utility classes.
Week 8	07/04/2025	Standardized layout properties and spacing across all modals.

Continued on next page

Week	Date	Tasks Completed
Week 8	08/04/2025	Fixed alignment anomalies in detailed data rendering tables.
Week 8	09/04/2025	Conducted cross-browser compatibility testing for major UI components.
Week 8	10/04/2025	Refined the material selection workflow based on testing feedback.
Week 9	13/04/2025	Implemented session restoration logic to recover data after page reloads.
Week 9	14/04/2025	Refined PWA (Progressive Web App) manifest and service worker configs.
Week 9	15/04/2025	Tested offline capabilities and evaluated cached asset performance.
Week 9	16/04/2025	Optimized application rendering using React lazy loading and memoization.
Week 9	17/04/2025	Documented the frontend architecture and state management flow.
Week 10	20/04/2025	Designed the data export and report generation feature interface.
Week 10	21/04/2025	Formatted dynamic project summaries optimized for print/PDF export.
Week 10	22/04/2025	Integrated charting libraries to visualize cost and emission breakdowns.
Week 10	23/04/2025	Added interactive tooltips and legends to data visualizations.
Week 10	24/04/2025	Fine-tuned responsive behavior for charts across different screen sizes.
Week 11	27/04/2025	Addressed technical debt, cleaning up unused variables and console logs.
Week 11	28/04/2025	Developed unit tests covering core calculation utilities.

Continued on next page

Week	Date	Tasks Completed
Week 11	29/04/2025	Improved error boundaries to ensure graceful degradation during failures.
Week 11	30/04/2025	Conducted a comprehensive UI/UX audit to ensure premium aesthetics.
Week 11	01/05/2025	Patched minor visual glitches discovered during the UI audit.
Week 12	04/05/2025	Stabilized React Component Architecture to prevent application crashes.
Week 12	05/05/2025	Resolved critical infinite re-render loops and stale variable references in `ProjectDataContext`.
Week 12	06/05/2025	Refactored form handling to decouple state synchronization from render cycles.
Week 12	07/05/2025	Standardized UI with Bootstrap 5 utilities for `NewProject`, `TrafficEmissions`, and `MaterialAddModal`.
Week 12	08/05/2025	Prepared codebase for backend service integration testing.
Week 12	09/05/2025	Resolved non-fast-forward Git push errors and merge conflicts; fixed file encoding issues.
Week 12	10/05/2025	Sunday - Conducted comprehensive codebase stability tests post-merge.
Week 13	11/05/2025	Finalized frontend documentation and drafted the user manual.
Week 13	12/05/2025	Performed end-to-end application workflow testing.
Week 13	13/05/2025	Polished final UI elements strictly adhering to visual design standards.
Week 13	14/05/2025	Prepared presentation slides and demonstration environment for review.

Continued on next page

Week	Date	Tasks Completed
Week 13	15/05/2025	Concluded project tasks and completed formal internship submission handover.

Bibliography

[1] Siddhartha Ghosh, Danish Ansari, Ajmal Babu Mahasrankintakam, Dharma Teja Nuli, Reshma Konjari, M. Swathi, and Subhrajit Dutta. Osdag: A Software for Structural Steel Design Using IS 800:2007. In Sondipon Adhikari, Anjan Dutta, and Satyabrata Choudhury, editors, *Advances in Structural Technologies*, volume 81 of *Lecture Notes in Civil Engineering*, pages 219–231, Singapore, 2021. Springer Singapore.

[2] FOSSEE Project. FOSSEE News - January 2018, vol 1 issue 3. Accessed: 2024-12-05.

[3] FOSSEE Project. Osdag website. Accessed: 2024-12-05.