INTERNSHIP REPORT

TAIKISHA ENGINEERING INDIA PVT.LTD

NAME: SHRESHTHA SRIVASTAVA

ROLE: SOFTWARE DEVELOPMENT INTERN

DEPARTMENT: IOT DEPARTMENT

DURATION: JUNE-JULY 2025

MENTOR: MR. ROHIT NAYAL

HEAD OF DEPARTMENT- MR. AJAI SRIVASTAVA

Certificate

This is to certify that **Shreshtha Srivastava** has successfully completed his Summer Internship under the guidance of **Mr. Rohit Nayal** at **Taikisha Engineering India PVT.LTD** from 5th June to 5th July 2025.

During his internship in the IoT Department, he contributed to both frontend and backend development of internal platforms and client-facing applications. He was actively involved in tasks in Python Django Framework, API integration, database management, SVG Html, and supported the team in building scalable and responsive web interfaces.

He showed strong technical skills, adaptability, and a proactive approach in completing the assigned tasks. His performance throughout the internship was professional and commendable.

We appreciate his contribution and wish him success in his future endeavours.

Date: 5th June 2025

Intern: Shreshtha Srivastava

H.O.D: Mr. Ajai Srivastava

Nexel Technical Center,
Taikisha Engineering India pvt. Ldt.
IMT Manesar, Haryana.

Table of Contents

INTRODUCTION

What is IoT?

The Internet of Things (IoT) refers to a vast network of physical devices—ranging from household appliances like refrigerators and thermostats to industrial equipment like sensors and machines—that are embedded with sensors, software, and connectivity to collect, exchange, and act on data over the internet. These smart devices communicate with each other, central systems, or cloud platforms to enable automation, real-time monitoring, and data-driven decision- making. IoT transforms everyday objects into intelligent systems, creating interconnected ecosystems that enhance efficiency and convenience across various sectors, including homes, healthcare, agriculture, manufacturing, and smart cities. For example, a smart thermostat learns a user's schedule to adjust home temperature, while IoT sensors in agriculture monitor soil moisture to optimize irrigation.

Key Components

- **Devices/Sensors**: Hardware components that capture data such as temperature, humidity, motion, or pressure. For instance, a wearable fitness tracker measures heart rate and steps.
- **Connectivity**: Devices connect via Wi-Fi, Bluetooth, Zigbee or 5G to transmit data. This allows seamless communication across local or global networks.
- **Data Processing**: Data is analyzed using cloud platforms (e.g., AWS, Azure) or edge computing for real-time insights. Edge computing processes data locally to reduce latency, such as in autonomous vehicles.
- **Applications**: User interfaces (e.g., mobile apps) or automated systems display insights or act on data, like a smart home app controlling lights remotely.

Examples

Smart Home: Thermostats like Nest adjust temperature based on user habits, saving energy.

Healthcare: Wearable devices track vital signs and alert doctors to anomalies, improving patient

Agriculture: IoT sensors monitor soil conditions, enabling precise

Transportation: Connected vehicles share traffic data to optimize routes and reduce congestion.

Benefits:

Efficiency: IoT automates tasks like turning off lights in empty rooms, saving energy.

Cost Savings: Sensors detect issues early, preventing costly breakdowns.

Real-Time Insights: Live data from devices (e.g., traffic sensors) helps in smart decision-making.

Convenience: Devices like thermostats can be controlled remotely via smartphone.

Sustainability: Smart systems like irrigation reduce water and resource waste.

- **Challenges**:

- **Security Risks:** IoT devices, if poorly secured, are vulnerable to hacking, potentially exposing sensitive data or allowing unauthorized control.
- **Privacy Concerns:** Constant data collection (e.g., smart home devices tracking user behavior) raises ethical questions about data usage.
- **Data Overload:** The massive volume of data from IoT devices requires robust systems to process and analyze effectively.
- **Interoperability:** Different IoT devices and platforms may use incompatible standards, complicating integration.
- **Reliability:** Dependence on internet connectivity can lead to failures in areas with poor network coverage.

What is SCADA?

Supervisory Control and Data Acquisition (SCADA) is a system used to monitor and control industrial processes in sectors like manufacturing, energy, water management, oil and gas, and transportation. SCADA systems collect real-time data from equipment (e.g., pumps, turbines, or conveyor belts), display it on a centralized interface, and allow operators to control processes manually or automatically. They ensure efficient, safe, and reliable operations by providing visibility into complex systems and enabling rapid responses to issues. For example, in a power plant, SCADA monitors generator performance and adjusts output to meet demand, while in a water treatment facility, it controls pumps to maintain water flow. SCADA systems are critical for managing large-scale, critical infrastructure,

integrating data from multiple sources to optimize performance and prevent failures.

Key Components

- **Sensors/Devices**: Measure parameters like temperature, pressure, flow, or voltage, installed on industrial equipment.
- **RTUs/PLCs**: Remote Terminal Units (RTUs) or Programmable Logic Controllers (PLCs) collect sensor data and send it to the SCADA system, also executing control commands.
- **Communication Network**: Connects devices to a central system using wired (e.g., Ethernet) or wireless (e.g., cellular) networks.
- **Human-Machine Interface (HMI)**: Software interface displaying data via graphs, alerts, or dashboards, allowing operators to monitor and control processes.
- **Central System**: Processes and stores data, often using servers or cloud platforms in modern setups, for analysis and decision-making.

Examples

- **Power Plants**: SCADA monitors electricity production, adjusts turbine speeds, and detects faults to prevent outages.
- **Water Treatment**: Tracks water levels and quality, controlling pumps and valves to ensure supply stability.
- **Manufacturing**: Oversees production lines, ensuring machines operate efficiently and alerting operators to issues like equipment jams.
- **Transportation**: Manages railway signaling systems, ensuring trains run safely and on schedule.

Evolution of SCADA Systems

SCADA systems have evolved through distinct generations, aligning with industrial revolutions, from standalone systems to modern, IoT-integrated solutions within Industry 4.0.

SCADA 1.0 (1960s–1970s)

SCADA 1.0 refers to the first generation of SCADA systems, developed during early industrial automation. These standalone systems used analog hardware to monitor and control processes locally. They relied on physical gauges or lights for data display, requiring on-site operators to adjust settings manually. For example, in a 1970s factory, SCADA 1.0 monitored motor speed using analog meters. These systems were robust but lacked connectivity and data storage.

- **Features**: Analog controllers, physical displays, manual control.
- **Examples**: Monitoring water levels in treatment facilities.
- **Limitations**: No remote access, costly hardware, no data storage.
- **IoT Connection**: None, as IoT did not exist.

SCADA 2.0 (1980s–1990s)

SCADA 2.0 emerged in the 1990s with the use of digital computers, Programmable Logic Controllers (PLCs), and basic Human-Machine Interfaces (HMIs). These systems enabled limited automation and allowed operators to monitor processes through computer screens. Local Area Networks (LANs) were used to connect devices within a facility, such as in oil refineries or power plants where parameters like pressure or generator performance were monitored. Although SCADA 2.0 improved data handling and local control, it was still limited by proprietary systems, lack of internet connectivity, and basic data analytics. While it did not connect to IoT, its use of local networking laid the groundwork for future IoT-enabled systems

SCADA 3.0 (2000s–2010s)

SCADA 3.0 introduced internet connectivity, enabling remote monitoring and control. Open standards like OPC improved device compatibility. Advanced HMIs provided graphical data visualization, and databases enabled trend analysis. For example, a 2000s water utility monitored pumps remotely via a web interface. However, SCADA 3.0 lacked advanced AI and had cybersecurity vulnerabilities.

- **Features**: Internet access, open standards, advanced HMIs.
- **Examples**: Remote monitoring of power substations.
- **Limitations**: Limited analytics, security risks, costly infrastructure.
- **IoT Connection**: Early internet use resembled IoT's connectivity.

SCADA in Industry 4.0

Industry 4.0, the fourth industrial revolution, integrates IoT, AI, cloud computing, and automation to create smart, connected industries. SCADA systems in Industry 4.0 are transformed to leverage these technologies, becoming more intelligent, scalable, and connected compared to earlier versions. They use IoT's wireless sensors, cloud platforms, and AI-driven analytics to enhance monitoring and control, aligning with the vision of smart factories and infrastructure.

Benefits

SCADA systems in Industry 4.0 offer significant advantages over earlier versions:

- **Proactive Maintenance**: AI analyzes IoT sensor data to predict equipment failures before they occur, reducing downtime. For example, in a manufacturing plant, SCADA predicts motor wear based on vibration data, scheduling repairs proactively.
- **Cost Efficiency**: Affordable IoT sensors and cloud platforms reduce the need for expensive proprietary hardware used in SCADA 1.0–3.0. This lowers setup and maintenance costs, enabling smaller businesses to adopt SCADA.
- **Global Access**: Cloud-based SCADA systems allow operators to monitor and control processes from anywhere via web or mobile apps. For instance, a power grid operator can adjust settings remotely during a storm, improving response times.
- **Sustainability**: IoT-driven SCADA optimizes resource use, such as reducing energy waste in smart grids or water loss in treatment systems, supporting environmental goals.
- **Enhanced Automation**: AI and IoT enable SCADA to automate complex tasks, like adjusting production speeds in a factory based on real-time demand, minimizing human intervention and errors.

Challenges

Despite its advancements, SCADA in Industry 4.0 faces significant challenges:

- **Cybersecurity Risks**: Increased IoT connectivity exposes SCADA systems to cyberattacks, such as hackers tampering with power grids or pipelines. Robust measures like encryption, secure protocols (e.g., MQTT), and real time threat detection are critical to protect sensitive infrastructure.
- **Integration Complexity**: Upgrading from SCADA 3.0 to Industry 4.0 systems requires integrating legacy hardware with modern IoT devices, which can be costly and technically challenging. For example, older PLCs may not support new wireless protocols, necessitating replacements.
- **Data Overload**: IoT generates massive data volumes, requiring advanced filtering and processing to avoid overwhelming SCADA systems. For instance, a smart city's SCADA system must manage data from thousands of sensors without delays.
- **Reliability Concerns**: Dependence on the internet and cloud connectivity can lead to failures in areas with poor network coverage, such as remote oil fields, unlike the standalone SCADA 1.0.
- **Skill Gaps**: Operating and maintaining Industry 4.0 SCADA systems re- quires expertise in IoT, AI, and cybersecurity, which may be scarce in traditional industries, necessitating extensive training.

What is a PLC?

PLC Definition: A programmable logic controller is a specialized computer designed to operate in industrial settings, managing and automating the mechanical processes.

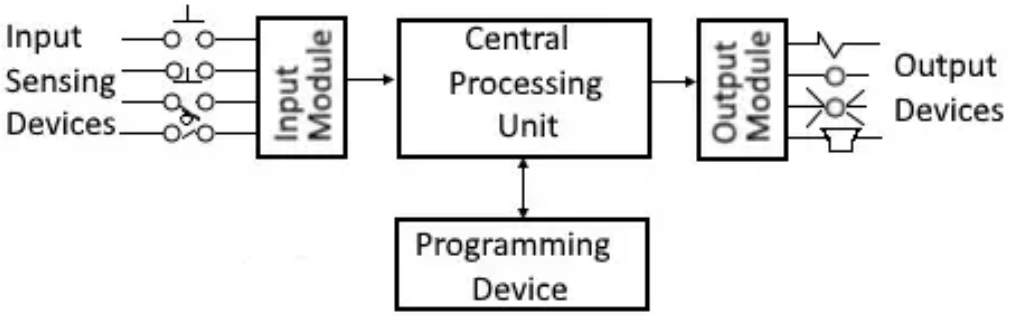
Key features:

- **Rugged design:** Built to withstand extreme temperatures, vibrations, and electrical noise**.**
- **I/O modules:** Connect to sensors and actuators for input/output control.
- **Programming:** Uses languages like Ladder Logic, Structured Text, or Function Block Diagrams, often following the IEC 61131-3 standard.
- **Applications:** Found in industries like manufacturing, oil and gas, and water treatment for tasks like conveyor control, robotic automation, or process monitoring**.**

PLC Device

Physical Structure of PLC

The structure of a PLC is almost like a computer's architecture.



PLC Block Diagram

Technologies Used Before PLCs

1. **Relay-Based Control Systems**:
   - **Description**: Hardwired electrical relays (electromechanical switches) were the primary method for automation. Circuits were designed using relays, timers, and counters to control machinery.
   - **How It Worked**: Relays opened or closed circuits based on electrical signals, creating logic sequences (e.g., turning a motor on when a switch was activated).
   - **Limitations**:
     1. Complex wiring for large systems, making installation and maintenance difficult.
     2. Hard to modify (required rewiring for changes).
     3. Prone to wear and failure due to mechanical components.

2. **Hardwired Logic Controllers**:
   - **Description**: Custom-built electronic panels using fixed logic circuits (e.g., transistors, diodes) to perform specific control tasks.

- **Use Cases**:
    1. **Material Handling**: Controlled sorting or conveyor systems in warehouses.
    2. **Chemical Plants**: Managed basic batch processes, like mixing or heating.
- **Limitations**:
    - No flexibility, changes required new hardware.
    - Expensive to design and build for each application.

3. **Pneumatic and Hydraulic Control Systems**:
- **Description**: Used air or fluid pressure to control mechanical devices, often with pneumatic timers or valves.
- **How It Worked**: Pneumatic/hydraulic signals triggered actuators (e.g., cylinders) to control machinery.
- **Limitations**:
    - Slow response times and limited precision.
    - Complex tubing/piping systems, hard to maintain.

4. **Analog Controllers**:
- **Description**: Early electronic devices using analog signals (e.g., voltage or current) to control processes, often with vacuum tubes or early transistors.
- **How It Worked**: Adjusted outputs (e.g., motor speed) based on continuous input signals (e.g., temperature sensors).
- **Limitations**:
    - Limited to simple proportional control (no complex logic).
    - Sensitive to noise and drift, requiring frequent calibration.

5. **Manual Control Systems**:
- **Description**: Human operators manually operated switches, valves, or levers to control machinery.
- **How It Worked**: Operators monitored gauges or indicators and adjusted controls based on observations.
- **Limitations**:

    Labor-intensive and prone to human error.

    Inefficient for large or complex systems.

6. **Drum Controllers**:
   - **Description**: Mechanical devices with rotating drums or cams that activated switches in a fixed sequence.
   - **How It Worked**: The drum's rotation triggered contacts at set intervals, controlling machine sequences.
   - **Limitations**:
     - Fixed sequences, hard to modify without rebuilding the drum.
     - Limited to simple, repetitive tasks.

Human Machine Interface (HMI)

The term HMI stands for Human-Machine Interface, a system that allows humans to interact with and control machines or automated processes, typically in industrial settings. It serves as the user-friendly interface between operators and equipment, such as Programmable Logic Controllers (PLCs), which you mentioned earlier.

There are three basic types of HMIs:

HMIs can be categorized by physical form, functionality, and Interface Type:

1. (Based on Physical Form/Application)

- **Dedicated HMI Panels**: Rugged, touchscreen-based devices mounted near machinery (e.g., Siemens SIMATIC HMI, Allen-Bradley PanelView). Used for local control in factories or plants.

- **PC-Based HMIs**: Run HMI software on industrial PCs or standard computers, offering flexibility for complex applications, typically in control rooms.
- **Mobile HMIs**: Use tablets, smartphones, or web-based interfaces for remote monitoring/control, common in IoT-integrated plants.
- **Embedded HMIs**: Integrated into machinery or compact PLCs with small displays, used for simple, standalone systems.

2. (Based on Functionality)

- **Push-Button HMIs**: Replace physical pushbuttons and LEDs, offering a digital interface for basic control tasks.
- **Data Handling HMIs**: Focus on processing and displaying data, such as sensor readings or production metrics.
- **Overseer HMIs**: Provide supervisory control and monitoring, often integrated with SCADA (Supervisory Control and Data Acquisition) systems for high-level oversight.

3. (Based on Interface Type)

- **Graphical User Interfaces (GUIs)**: Computer-based interfaces using software (e.g., Microsoft Windows, Wonderware InTouch) to display process data graphically.
- **Touchscreens**: HMIs with touchscreen displays for direct interaction, used in space-constrained or rugged environments.
- **Physical Control Panels**: Use buttons, switches, and displays, suited for harsh environments with vibration or dust.
- **Web-Based HMIs**: Accessed via web browsers for remote monitoring/control from any device.

Windows-Based SCADA vs. Web-Based SCADA

Here's a side-by-side comparison of **Windows-based SCADA** (traditional, running on Windows OS) and **web-based SCADA** (accessible via web browsers), focusing on key differences:

Aspect	Windows-Based SCADA	Web-Based SCADA
Definition	SCADA software installed on a Windows PC or server, using a "fat client" model.	SCADA software hosted on a server, accessed via web browsers on any device.
Platform	Runs on Windows OS (e.g., Windows 10, Server 2016). Requires local installation.	Runs on servers (cloud or on-premises) and accessed via browsers (e.g., Chrome).
Accessibility	Limited to devices with the SCADA software installed, typically on-site.	Accessible from any device (PC, tablet, smartphone) with internet and a browser.
HMI Interface	Uses desktop-based HMIs (e.g., graphical dashboards) on Windows, often tied to PLCs.	Uses web-based HMIs (HTML5, JavaScript) for remote monitoring/control.
Installation	Requires installation on each client PC, often complex and OS-specific.	No client installation browser-based access simplifies deployment.
Scalability	Scaling requires additional hardware (servers/PCs) and software licenses.	Easily scalable via cloud or server resources, no extra client hardware

		needed.
Maintenance	Requires local maintenance (OS updates, patches, antivirus). Prone to Windows issues.	Maintained by server/cloud provider, reducing local IT burden.
Performance	Fast for local control, but limited by PC hardware and network.	May have latency due to internet reliance but optimized for remote access.
Security	Vulnerable to Windows-specific attacks (e.g., malware). Needs VPN for remote access.	Requires strong cloud security (e.g., SSL/TLS, firewalls) but exposed to web attacks.
Cost	High upfront costs (hardware, licenses); lower ongoing costs.	Lower upfront costs; higher ongoing costs for cloud subscriptions.
Examples	Siemens WinCC, AVEVA Wonderware, GE CIMPLICITY (Windows versions).	Ignition by Inductive Automation, Rapid SCADA, IntegraXor (web versions).
Use with PLCs	Connects to PLCs (e.g., Siemens S7-1200) via local networks (e.g., Modbus, OPC UA).	Connects to PLCs via IP protocols (e.g., TCP/IP, MQTT) for remote data access.

JavaScript Concepts Learned

JavaScript Basics & Output Methods

- I learned how to print outputs in the browser using various JavaScript **Console APIs** like:
    - console.log() – used to log output
    - console.error() – to show error messages
    - console.warn() – to display warnings

These helped me understand how developers debug JavaScript code.

Variables, Data Types & Objects

- I learned about the basic **data types** such as strings, numbers, booleans, arrays, and objects.
- Also learned how to declare variables using let, const, and var.
- Objects were especially interesting because they allow storing data in **key-value** pairs. Example:

```
let student = {
  name: "Shreshtha",
  age: 21,
  course: "Web Development"
```

};

Arrays & Functions

- Arrays allowed me to store multiple values in a single variable.
- I explored array methods like push(), pop(), shift(), unshift(), forEach(), etc.
- I also studied **functions** – reusable blocks of code. I understood:
    - How to declare regular functions
    - The concept of **arrow functions**, which are shorter versions of normal functions.

Example:

```
const greet = (name) => {
  console.log("Hello " + name);
};
```

DOM Manipulation

DOM (Document Object Model) was one of the most exciting parts. I learned how JavaScript can interact with HTML elements using methods such as:

- document.getElementById() – to select elements by ID.
- document.querySelector() – to select elements using CSS selectors.
- appendChild() – to add a new element to the DOM.
- replaceChild() – to replace an existing DOM element with another.

This gave me the ability to make interactive web pages.

Event Handling

Understanding **JavaScript Events** was a major milestone. I learned about:

- addEventListener() – to attach events like click, hover, etc.
- How to execute functions when a user interacts with a page.

Example:

```
document.getElementById("btn").addEventListener("click", function() {
  alert("Button Clicked!");
});
```

This helped me build dynamic user interactions.

Timing Functions

- **setTimeout()** – to delay the execution of a function.
- **setInterval()** – to execute code repeatedly at a set time interval.

These functions are helpful in creating animations or periodic updates on web pages.
Example:

```
setTimeout(() => {
  console.log("Executed after 2 seconds");
}, 2000);
```

ColorBlockChanger: A Responsive Web Development Project

I was given a task to create a simple color block changing project, which led to the development of ColorBlockChanger. This beginner-level web development project demonstrates the core concepts of HTML, CSS, and JavaScript through an interactive webpage featuring a square block that changes color based on user input. Its straightforward design, modular code structure, and responsive layout make it an ideal learning tool for students new to web technologies. The webpage is fully functional across devices, including desktops, tablets, and smartphones, ensuring accessibility and usability.

Objective

The objective of this project is to build a webpage where a square block changes color to green, orange, or red when the corresponding buttons are clicked and reverts to its original gray color upon clicking a reset button. The project aims to teach fundamental web development skills, including webpage structure, styling, interactivity, and responsive design using rem units. Additionally, it emphasizes clean code organization and user-friendly design for educational purposes.

Project Description

The ColorBlockChanger webpage includes the following components:

- A square block (200px × 200px, or 12.5rem) centered on the page, initially gray.
- Three buttons labeled “Green,” “Orange,” and “Red” arranged horizontally below the block.
- A “Reset” button positioned below the color buttons.
- Clicking a color button changes the block’s color to match (e.g., “Green” changes it to green).
- Clicking the “Reset” button restores the block’s gray color.
- The design is responsive, using rem units and media queries to adapt to various screen sizes, ensuring a consistent user experience on mobile phones, tablets, and desktops.

Technical Details

The project is organized into three separate files to promote modularity and maintainability:

1. **index.html**:
    - Defines the webpage’s structure using HTML.
    - Contains a div element with ID colorBlock for the square block, a div with class button-container for the color buttons, and a reset button.
    - Links to styles.css for styling and script.js for functionality.
    - Uses semantic HTML to ensure clarity and accessibility.
2. **styles.css**:
    - Manages the webpage’s appearance using CSS.
    - Centers content using CSS Flexbox (display: flex) and applies a light gray background (#f0f0f0) for a clean look.
    - Styles the block with a gray background (#ccc), a 2px black border, and a size of 12.5rem.
    - Styles buttons with solid colors (green, orange, red for color buttons; gray for reset) and a hover effect that reduces opacity to 0.8 for visual feedback.
    - Uses rem units (1rem = 16px) for all measurements to ensure scalability.
    - Implements media queries for responsiveness:
        - For screens ≤ 400px: Reduces font size to 14px, block size to 10rem, and button sizes.
        - For screens ≤ 300px: Further reduces font size to 12px and block size to 8rem.

3. **script.js**:
    - Handles interactivity using JavaScript.
    - Uses document.getElementById to select the block and buttons by their IDs.
    - Adds addEventListener to each button to change the block's backgroundColor property to green, orange, red, or #ccc (gray for reset).
    - Employs arrow functions for concise event handling, aligning with modern JavaScript practices.
    - Includes detailed comments to make the code accessible to beginners.

How It Works

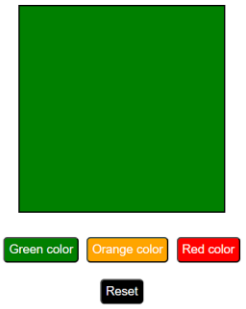
The user interacts with the webpage as follows:

- On loading, the webpage displays a gray square block, three color buttons (Green, Orange, Red), and a reset button, all centered.
- Clicking a color button triggers a JavaScript event listener that changes the block's backgroundColor to match the button's color.
- Clicking the "Reset" button reverts the block to its default gray color (#ccc).
- The layout remains consistent across devices, with smaller elements on mobile screens for better usability.

Learning Outcomes

This project provided significant learning opportunities:

- **HTML**: Gained expertise in creating a structured webpage with semantic elements like div and button.
- **CSS**: Learned to style elements using flexbox, apply responsive design with rem units, and use media queries to adapt to different screen sizes.
- **JavaScript**: Mastered DOM manipulation using document.getElementById and event handling with addEventListener to create interactive features.
- **Responsive Design**: Understood how to design webpages that work seamlessly on desktops, tablets, and smartphones.
- **Code Organization**: Learned the importance of separating structure (HTML), style (CSS), and behavior (JavaScript) into distinct files.
- **Debugging**: Gained experience testing and ensuring cross-browser compatibility and responsive behavior.



Screenshot of the ColorBlockChanger webpage showing the block and buttons

Github link- Shreshtha03/Color_Block_Changer

Check working on-**https://shreshtha03.github.io/Color_Block_Changer/**

Traffic Light : A traffic Ligth working using HTML CSS & JS

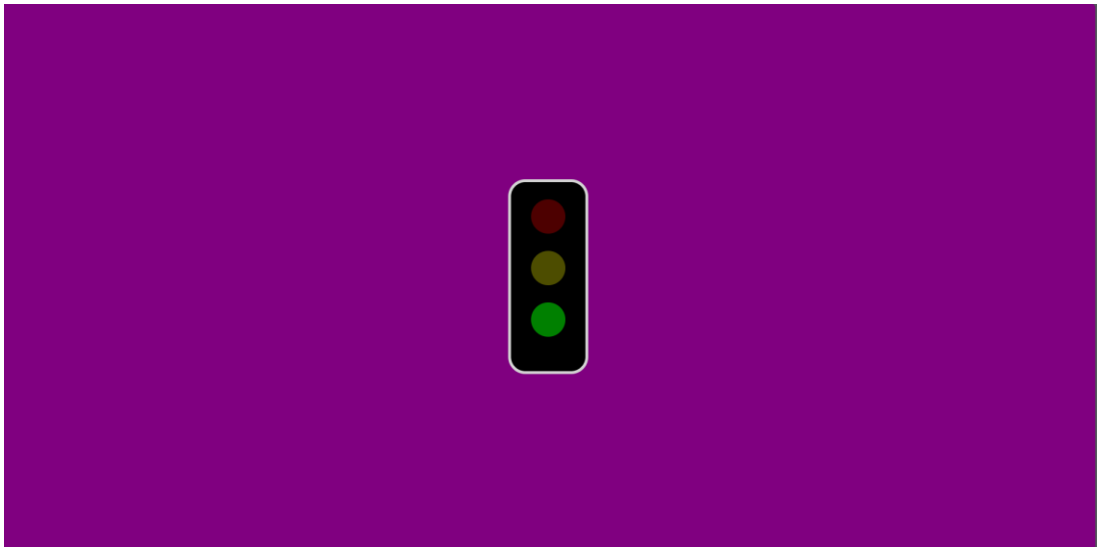
A simple and fun **Traffic Light Simulation** built using **HTML**, **CSS**, and **JavaScript**. This project mimics a real-world traffic light where red, yellow, and green lights cycle every few seconds, just like on the road.

Features

- **Timed Cycle**: Each light stays on for 3 seconds before switching to the next.
- **Responsive Design**: Looks great on any screen size, centered with flexbox.
- **Smooth Transitions**: Lights fade in and out with a cool opacity effect.

How It Works

- Three circular divs styled to look like red, yellow, and green lights.
- JavaScript uses setTimeout to switch between lights every 3 seconds: Red → Yellow → Green → Red (repeats forever).
- CSS handles the styling, making the lights circular, colorful, and centered on the screen.
- The active class controls which light is "on" (fully opaque) and which are "off" (semi-transparent).



Screenshot of the Traffic Light webpage showing a Traffic Light system

Traffic Light Simulation:Updated Traffic page with Login page

A simple and fun Traffic Light Simulation built using HTML, CSS, and JavaScript. This project mimics a real-world traffic light where red, yellow, and green lights cycle every few seconds, just like on the road. Now, it includes a Login Page for added interactivity! Perfect for beginners to learn basic web development concepts like DOM manipulation, CSS styling, JavaScript timing, and user authentication.

Updates

June 11, 2025: Added a Login Page! Now users need to enter a username (admin) and password (123) to access the traffic light simulation. This makes the project more interactive and secure.

Features

- Login Page: Enter username (admin) and password (123) to access the traffic light simulation.
- Realistic Traffic Light: Red, yellow, and green lights turn on and off in a cycle.
- Timed Cycle: Each light stays on for 3 seconds before switching to the next.
- Responsive Design: Looks great on any screen size, centered with flexbox.
- Smooth Transitions: Lights fade in and out with a cool opacity effect.
- Error Handling: Shows an error message if the username or password is incorrect.

How It Works

- Login Page:
  - A simple login form with username and password fields.
  - Default credentials: Username = admin, Password = 123.
  - On successful login, redirects to the traffic light simulation page.
  - Shows an error message for incorrect credentials.
- Traffic Light:
  - Three circular divs styled to look like red, yellow, and green lights.
  - JavaScript uses setTimeout to switch between lights every 3 seconds: Red → Yellow → Green → Red (repeats forever).
  - CSS handles the styling, making the lights circular, colorful, and centered on the screen.
  - The active class controls which light is "on" (fully opaque) and which are "off" (semi-transparent).

Installation and Setup

Follow these steps to run the project locally:

1. Clone the Repository:

git clone https://github.com/Shreshtha03/Traffic-Light-Simulation.git

2. Navigate to the Project Folder:

cd Traffic-Light-Simulation

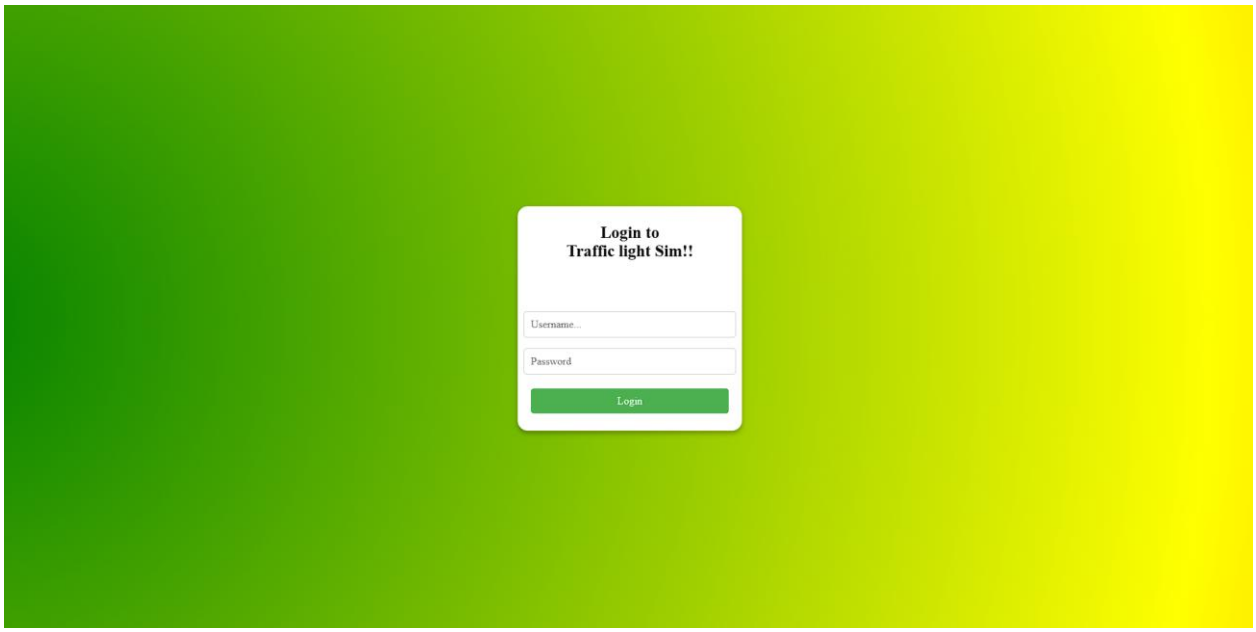
3. Open the Project:
   - Open index.html in a web browser (e.g., Chrome, Firefox).
   - You can use a local server like Live Server in VS Code for a better experience.
4. Login Credentials:
   - Username: admin
   - Password: 123

Learning Outcomes

This project helped me learn:

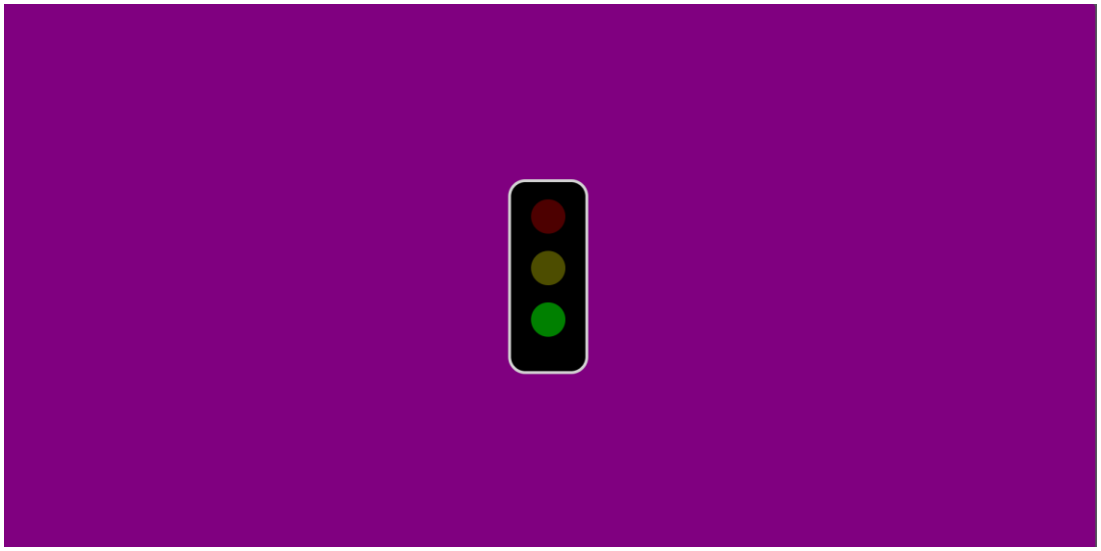
- How to create a login page with user authentication in JavaScript.
- DOM manipulation with JavaScript (getElementById, classList).
- Using setTimeout for timed events.
- CSS flexbox for centering elements.
- Creating smooth transitions with CSS.

Login Page-



Screenshot of the Login Page for Traffic Light

Traffic page-



Screenshot of the Traffic Light webpage showing a Traffic Light system

Github link- Shreshtha03/Traffic-Light-Simulation

Check working on- **https://shreshtha03.github.io/Traffic-Light-Simulation/**

Django Framework – Key Learnings from Internship

During my internship, I explored the basics of the Django framework, which is a high-level, open-source web framework written in Python. Below are the key concepts and skills I learned:

1. MVT Architecture (Model-View-Template)

- Understood Django's core structure, which follows the MVT design pattern.
- Model: Defines the structure of the database using Python classes.
- View: Handles business logic and processes user requests.
- Template: Manages the front-end and renders dynamic data using Django's template language.

2. Project and App Structure

- Learned to create Django projects and divide them into smaller apps for modular development.
- Used commands like startproject and startapp for setting up the project.

3. URL Routing (urls.py)

- Gained knowledge of mapping URLs to specific views using Django's URL dispatcher.
- Worked with dynamic and static URL paths.

4. Admin Interface

- Explored Django's built-in admin panel for managing models through a graphical interface.
- Registered models in admin.py to enable admin dashboard features.

5. Django ORM (Object Relational Mapping)

- Interacted with the database using Python code instead of raw SQL.
- Performed CRUD operations and used migration commands like makemigrations and migrate.

6. Templates and Template Tags

- Created HTML templates and used Django's template tags ({{ }}, {% %}) for dynamic

content.

- Implemented loops, conditions, and template inheritance for reusability.

7. Forms and Input Handling

- Built Django forms for collecting and validating user data.
- Handled GET and POST requests securely.

8. Static and Media File Management

- Configured static files (CSS, JS) and media files (user uploads).
- Learned to serve and link these files properly in the template.

9. Development Server

- Used python manage.py runserver to run the development server locally.
- Tested the application at http://127.0.0.1:8000/.

Employee Management System Database

A Django-based managing employee and department records. Administrators can create, update, view, and delete data through the Django admin panel. The Department model features a dropdown for selecting employees, simplifying department assignments.

Features

- **Department Management:**
  - Fields: Employee (dropdown with employee names), Department Name.
  - Admin panel: List view, filter by department name, search by employee name/department name, sort by employee.

- **Validations:**
  - Email format validation.
  - 10-digit phone number validation.
- **User-Friendly Admin Interface:**
  - Built-in Django admin panel for data management.

Technologies Used

- Framework: Django 5.2.2
- Language: Python 3.13
- Database: SQLite
- Validators: Django's EmailValidator and RegexValidator

Project Installation and Setup Process

To successfully run the Django-based Employee Management System project, I followed the steps below during the installation process:

- **Cloning the Repository:**
  The project source code was cloned from the GitHub repository to my local system.
- **Creating a Virtual Environment:**
  A Python virtual environment was set up to manage project-specific dependencies and avoid conflicts.
- **Activating the Environment:**
  The virtual environment was activated to ensure all packages were installed within the project scope.
- **Installing Dependencies:**
  Django was installed as the primary framework for the project, along with any other required packages.

- **Applying Migrations:**
  The necessary migrations were created and applied to set up the initial database schema.
- Creating a Superuser:
  An admin (superuser) account was created to access Django's built-in admin interface for managing the system.
- Running the Server:
  The development server was started, and the application was accessed via the local browser .

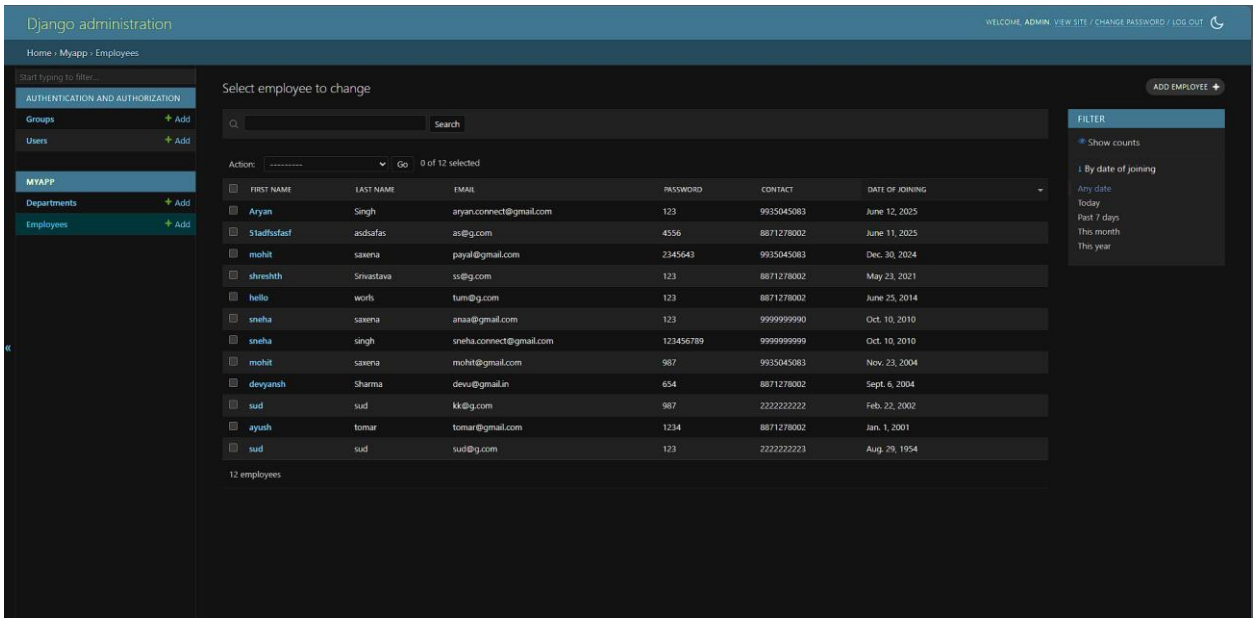
Project Structure

- myapp/models.py: Defines Employee and Department models.
- myapp/admin.py: Configures admin panel for models.

- employee_management/settings.py: Project settings.
- db.sqlite3: SQLite database.

Usage

1. **Access Admin Panel:**
    - Go to http://localhost:<port_number>/admin/ and log in with superuser credentials.
2. **Manage Employees:**
    - Add/edit employee records with validated fields.
    - Filter by date of joining, search by name/email.
3. **Manage Departments:**
    - Add/edit departments, selecting employees from the dropdown.
    - Add employees first to populate the dropdown.
    - Filter by department name, search by employee/department.



This screenshot displays the Employee section in the Django Admin Panel where user data is managed and stored.

Employee Management System Using CRUD Operation:

Employee CRUD Application A simple Employee Management System built with Django and Django REST Framework for the backend, and HTML, CSS, and JavaScript for the frontend. This application allows users to perform CRUD operations (Create, Read, Update, Delete) on employee records via a web interface. Features

List Employees: View all employees in a table with details like ID, first name, last name, email, contact, and date of joining. Delete Employee: Remove an employee record by entering their ID. API-Driven: Uses Django REST Framework to provide RESTful API endpoints. (Note: Create and Update features may be available in the backend but are not fully implemented in the provided frontend code.)

Technologies Used

Backend: Python 3.8+ Django 4.0+ Django REST Framework SQLite (default database)
Frontend: HTML CSS JavaScript (for API calls and DOM manipulation)
Tools: Git Virtualenv

Installation

Clone the Repository: git clone https://github.com/Shreshtha03/Employee-CRUD-using-Django.git cd Employee-CRUD-using-Django

- Set Up Virtual Environment: python -m venv venv source venv/bin/activate # Linux/Mac venv\Scripts\activate # Windows
- Install Dependencies: pip install -r requirements.txt
- Apply Migrations: python manage.py migrate
- Run the Server: python manage.py runserver
- Open http://localhost:8000 in your browser.
- API Endpoints
- Method Endpoint Description
- GET /api/list/ Fetch all employees
- DELETE /api/delete// Delete an employee by ID

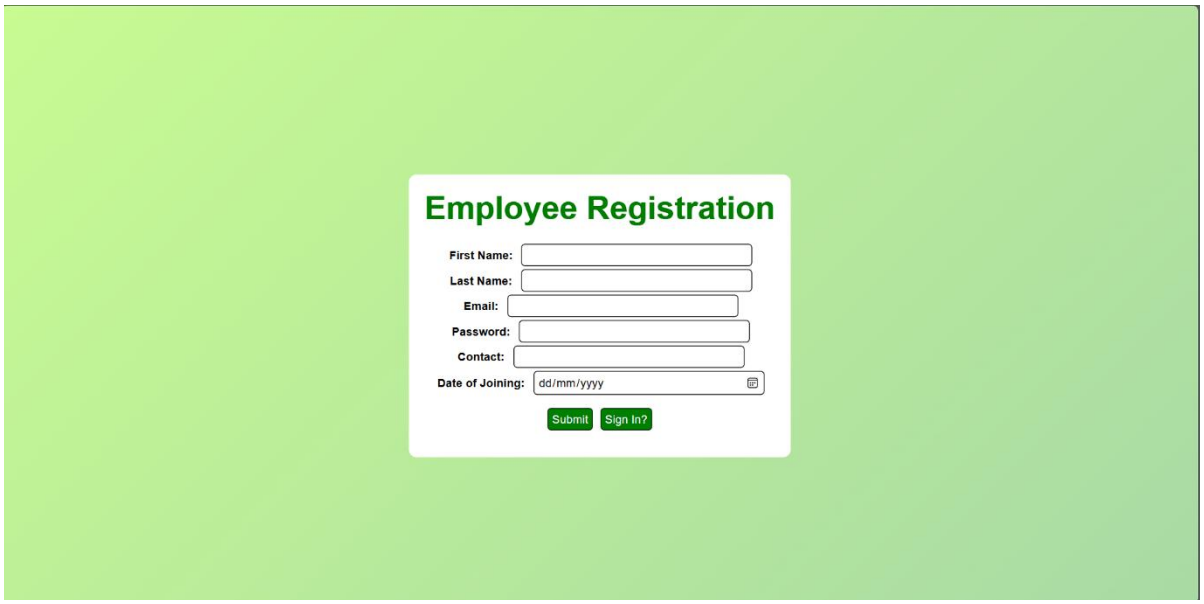
Usage

View Employees: On page load, the table displays employee data fetched from /api/list/. Delete Employee: Enter an employee ID and click "Delete" to remove the record. The table updates automatically. Error Handling: Displays error messages for invalid IDs or failed API requests.

Future Improvements

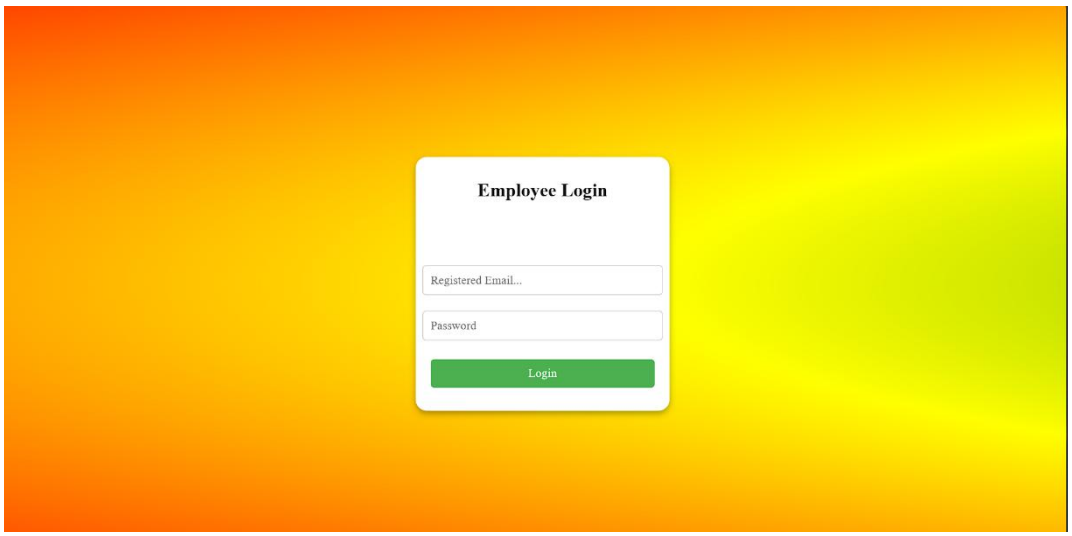
Add frontend support for creating and updating employee records. Implement user authentication for secure access. Add a loading indicator during API calls. Enhance UI with a CSS framework.

Registration Page:



Form for new employees to register by entering their basic details.

Login Page if already registered:



Shows the login form where users enter email and password to access the system.

Registered employees :

Lists of employees

ID	First Name	Last Name	Email	Contact	Joining Date
4	ayush	tomar	tomar@gmail.com	8871278002	2001-01-01
5	mohit	saxena	mohit@gmail.com	9935045083	2004-11-23
6	sneha	singh	sneha.connect@gmail.com	9999999999	2010-10-10
8	sneha	saxena	anaa@gmail.com	9999999990	2010-10-10
9	mohit	saxena	payal@gmail.com	9935045083	2024-12-30
12	devyansh	Sharma	devu@gmail.in	8871278002	2004-09-06
16	sud	sud	kk@g.com	2222222222	2002-02-22
17	sud	sud	sud@g.com	2222222223	1954-08-29
18	hello	worls	tum@g.com	8871278002	2014-06-25
20	shreshth	Srivastava	ss@g.com	8871278002	2021-05-23
22	Aryan	Singh	aryan.connect@gmail.com	9935045083	2025-06-12
23	51adfssfasf	asdsafas	as@g.com	8871278002	2025-06-11

Enter ID Delete

Register | Login

Displays a list of all registered employees with options to delete records.

Github link: [Shreshtha03/Employee-CRUD-using-Django]

Smart Dashboard using Django

A dynamic and interactive dashboard application designed to monitor and visualize real-time data including temperature, humidity, voltage, and current. Built with HTML, CSS, JavaScript, and Django, this project features customizable cards, line charts for trends, and dynamic pie charts for data distribution.

Features

- Real-Time Data Visualization: Line charts (myChart1 for Temperature/Humidity, myChart2 for Voltage/Current) to track trends over time.
- Dynamic Pie Chart: Displays the proportional distribution of the latest card values (e.g., temperature, humidity, voltage, current).
- Customizable Cards: Editable labels and values with local storage synchronization for persistence.
- Responsive Design: Fully functional across desktops, tablets, and mobile devices.
- API Integration: Fetches and updates data via RESTful endpoints.
- Data Sampling: Limits chart data points (max 50) for optimal performance and readability.

Prerequisites

- Python 3.x
- Django (with djangorestframework for API)
- Node.js (optional, for frontend dependencies if expanded)
- Git (for version control)

3. Set Up Frontent

Ensure all static files are in the static folder:

- css/style.css for styling
- js/first.js for JavaScript logic

No additional Node.js setup is required unless you add more libraries.

4. Run the Application
Open your browser and navigate to http://127.0.0.1:8000/ to access the dashboard.

Usage

- **Cards:** Click the "Edit" button on any card to modify its equipment and value. Changes are saved in the browser's local storage.

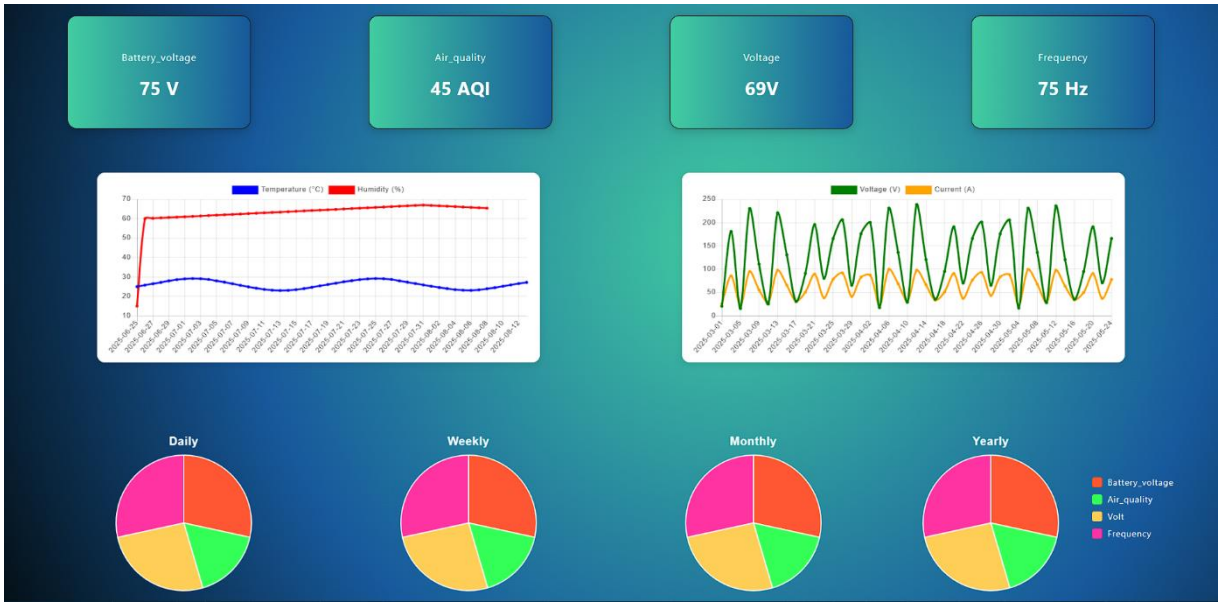
- **Charts:**
  - Line Charts: myChart1 shows Temperature (°C) and Humidity (%), while myChart2 displays Voltage (V) and Current (A).
  - Pie Chart: Visualizes the latest values of all cards as a percentage distribution.

- **API Endpoints:**
  - /api/Temperature/: Fetches temperature data.
  - /api/Humidity/: Fetches humidity data.
  - /api/Voltage/: Fetches voltage data.
  - /api/Current/: Fetches current data.
  - /api/Card/: Manages card configurations and values.

Project Structure

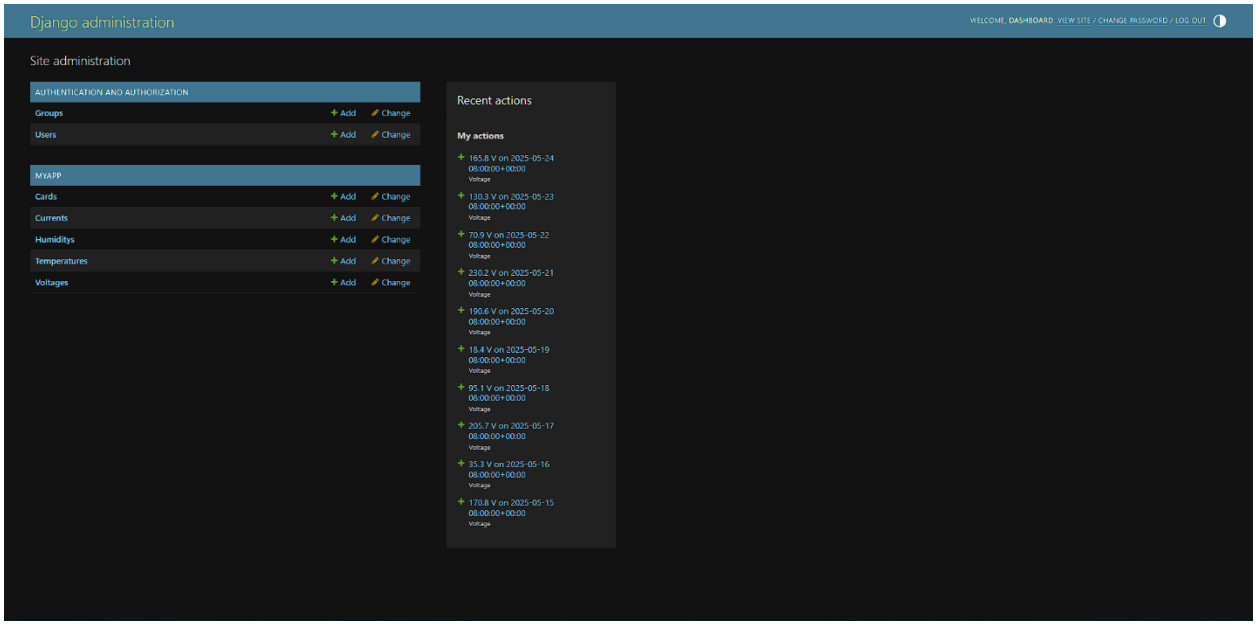
```
SmartDash/
├── dashboard/
│   ├── static/
│   │   ├── css/
│   │   │   └── style.css    # CSS for dashboard styling
│   │   └── js/
│   │       └── first.js     # JavaScript for charts and card logic
│   ├── templates/
│   │   └── index.html       # Main HTML template
│   └── views.py             # Django views for API and pages
├── manage.py                # Django management script
└── README.md                # This file
```

Screenshots
Dashboard
Overview:



Displays real-time data with line and pie charts for sensor analytics.

Django
admin:



Shows Django admin interface for managing sensor data models.

Github link: [Shreshtha03/Smart-Dash-using-Django]

Introduction to SVG and Inkscape

As part of my web development learning, I studied SVG (Scalable Vector Graphics) in HTML. SVG is a powerful XML-based markup language used to create high-quality vector images that remain sharp and clear regardless of screen size or resolution. It is especially useful for web-based graphics like icons, logos, charts, and interactive visuals.

Alongside learning SVG code manually in HTML, I also explored Inkscape, a widely used open-source vector graphics editor. Inkscape provides an easy-to-use interface for creating and designing SVG files visually, without needing to write code. I used it to draw shapes, curves, and text, then exported them as .svg files for use in websites. I also learned how to manipulate SVG using CSS and JavaScript, which helped in creating interactive web components.

This learning improved my understanding of how vector graphics work on the web and how design and development can be combined using tools like Inkscape and HTML.

Webpage Using SVG HTML

This is a responsive landing page for a web dashboard project named Smart Dash, built using HTML, CSS, SVG, and Django. It features a modern user interface with vibrant vector patterns and a welcoming home screen. The design includes custom shapes created using Inkscape, a free vector graphics editor, and embedded using SVG in HTML for scalability and clarity.

Technologies Used

- HTML5 & CSS3
- SVG (Vector Graphics via Inkscape)
- Django (for backend routing)
- Static files and templates system

Features

- Clean and modern homepage layout
- Responsive SVG patterns and visuals
- “Welcome” button with navigation intent
- Fully responsive design across devices
- Ready to integrate with other Django apps or dashboards

How to Run Locally

1**. Clone the repository**

git clone https://github.com/your-username/smartdash-landing-page.git

cd smartdash-landing-page

 # 2. Create virtual environment

python -m venv venv

Activate:

Windows: venv\Scripts\activate

Mac/Linux: source venv/bin/activate

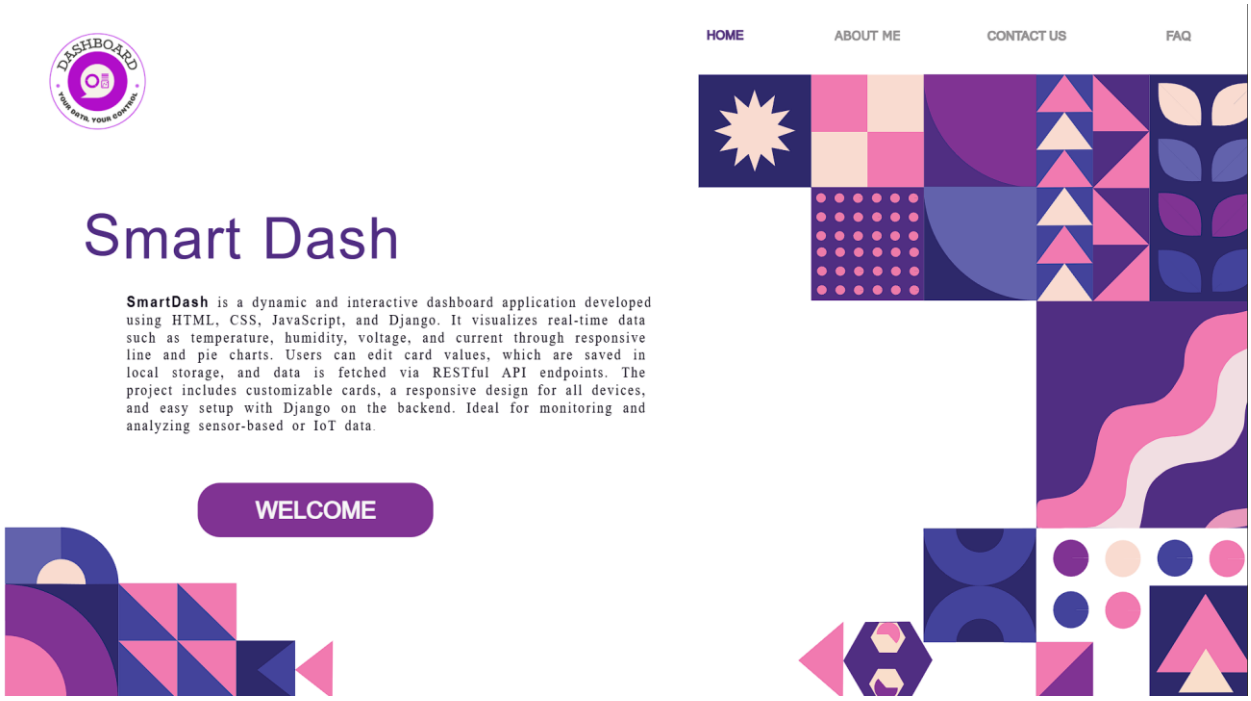
 # 3. Install Django

pip install django

 # 4. Run the development server

python manage.py runserver

Screenshot



Screenshot of the landing SVG HTML page

Github link: [Shreshtha03/Smart-dash-html-SVG]

Smart Dash – Db + SVG + SMTP + Django

Smart Dash is a Django-based web application that includes a real-time dashboard, an email contact form, and a personal portfolio page with a downloadable CV option. Built using HTML, CSS, JavaScript (Chart.js), and Django, this project demonstrates both backend logic and frontend interactivity in one integrated web app.

🔧 Features

📊 Dashboard (/)

- Displays real-time data: **Temperature**, **Humidity**, **Voltage**, and **Current**
- Interactive **line charts** and **pie charts** using Chart.js
- SVG logo used in the navbar for design consistency

📩 Contact Us (/sendmail/)

- Email form with:
  - Recipient email input
  - Message field
  - Image file upload (JPG, PNG, JPEG) with preview
- Emails sent through **Gmail SMTP**

👤 About Me (/aboutme/)

- This is **your personal portfolio page**
- Showcases your projects (e.g., Mental Health Platform, Shopify Clone)
- Includes a **"Download Resume/CV"** button at the bottom of the page

🗂️ Project Structure

```
Smart-dash-html-SVG-Django/
├── dashboard/
│   ├── __init__.py
│   ├── asgi.py
│   ├── settings.py
│   ├── urls.py
│   ├── views.py
│   └── wsgi.py
├── myapp/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations/
│   ├── models.py
│   ├── serializers.py
│   ├── tests.py
│   ├── urls.py
│   └── views.py
├── sendmail/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
```

```
    ├── migrations/
    ├── models.py
    ├── tests.py
    ├── urls.py
    └── views.py
├── static/
    ├── images/
    ├── aboutme.css
    ├── chart1.js
    ├── chart2.js
    ├── contact.css
    ├── contact.js
    ├── edit.css
    ├── edit.js
    ├── piechart.js
    └── style.css
├── templates/
    ├── aboutme.html
    ├── contact.html
    ├── edit.html
    ├── index.html
    └── svg.html
├── db.sqlite3
├── manage.py
└── README.md
```

⚙️ Installation

✅ Prerequisites

- Python 3.8+
- Django 4.2+
- Git
- Gmail account with **App Password** enabled

🚀 Setup Instructions

1. **Clone the Repository**

```
git clone https://github.com/Shreshtha03/Smart-dash-html-SVG-Django.git
cd Smart-dash-html-SVG-Django
```

2. **Create a Virtual Environment**

```
python -m venv venv

# Activate:
source venv/bin/activate  # On Windows: venv\Scripts\activate
```

3. **Install Dependencies**

```
pip install Django
```

4. **Configure Email Settings** Edit dashboard/settings.py:

```
EMAIL_HOST_USER = 'your_email@gmail.com'
```

EMAIL_HOST_PASSWORD = 'your_app_password'

5. **Run Migrations**

```
python manage.py makemigrations
python manage.py migrate
```

6. **Start the Server**

```
python manage.py runserver
```

Then open:

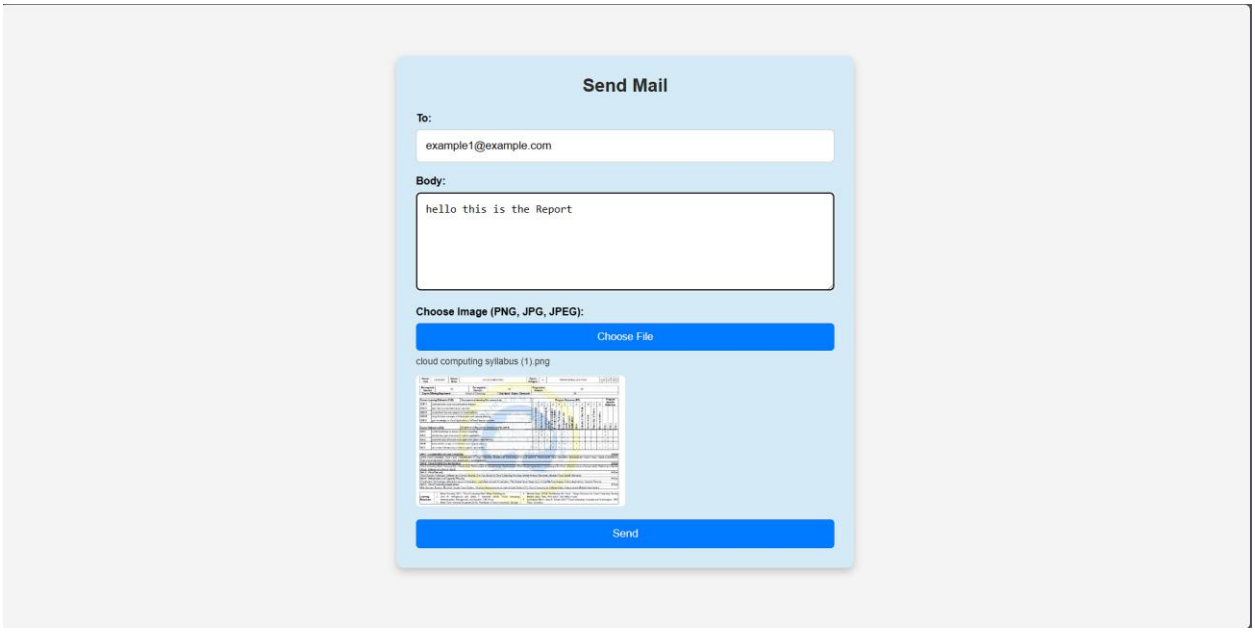
📍 http://localhost:8000/

🌐 Usage Guide

Page	URL	Description
Home	/	Dashboard with charts and real-time data
Contact Us	/sendmail/	Email form with message + image upload
Portfolio	/aboutme/	Your portfolio with **Resume Download** button

🧰 Tech Stack

Layer	Tools / Libraries
Backend	Django 4.2, Python 3.x
Frontend	HTML, CSS, JavaScript, Chart.js
Charts	Chart.js
Email	Gmail SMTP with App Password
Static	Django static files
Version Control	Git + GitHub



Send Mail Form with Image Upload and Preview

Github- link: [Shreshtha03/Smart-Dash-Db-SVG-SMTP-Django-]

BUS Booth Layout

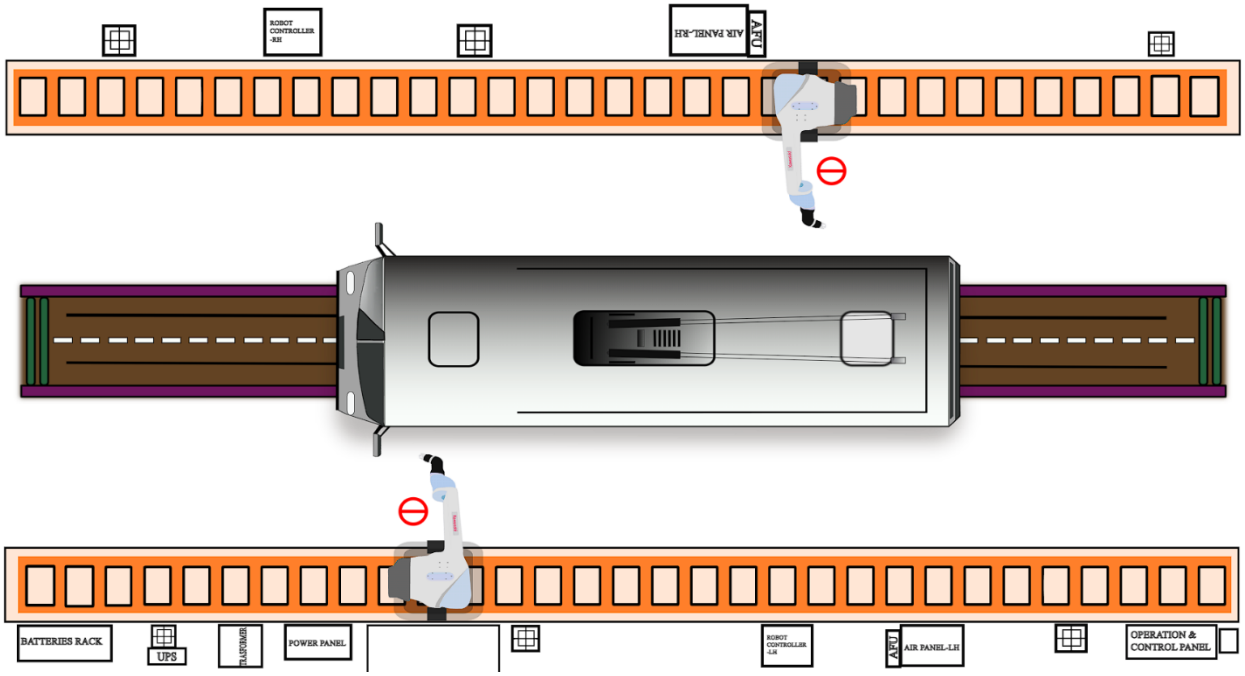
Introduction
This booth showcases JSW advanced automated manufacturing process, highlighting the use of robotics and conveyor systems to enhance efficiency in steel production.

Layout Description
The booth features a central conveyor track where robots handle materials. The top and bottom sections include workstations and power/control units, ensuring smooth operation.

Purpose
The display demonstrates JSW cutting-edge technology and automation capabilities, optimizing steel production processes.

Key Components

- Robots for material handling
- Central conveyor track
- Power supply (UPS, Transformer)
- Control panels (Robot Controller, Operation & Control Panel)



Github-link : [Shreshtha03/Paintning-Booth-SVG]