

MAJOR PROJECT

EMPLOYEE LEAVE
MANAGEMENT SYSTEM

TECHNICAL SYSTEM
WHITEPAPER

Contents

1.	Introduction	3
2.	Basic Overview	4
3.	Challenges	12
4.	Front-End Development Decision	17
5.	React.js Development	18
6.	Bootstrap Framework	24
7.	Back-End Development Decision	34
8.	Java Development	37
9.	RESTful API Implementation	41
10.	Front-End & Back-End Connection	44
11.	AXIOS Framework Implementation	45
12.	Database Interaction	47
13.	Concerns About Security	53
14.	A Future Sight	58
15.	Conclusion	59
16.	Bibliography	61
17.	Copyright Information	62

INTRODUCTION

This is a datasheet designed by the design, development and deployment team of the project and primarily contains the working principles and technical information about the project.

The Employee Leave Management System is a website that helps you easily manage leaves on the fly. For Employees, it gives you a very intuitive UI for logging into the website, registering if you are a new user, and easily apply for leaves. They can also see the leaves that are already applied in the past. As for the admins, they can easily see who have applied, and the other parameters of the leaves. From there, he can also easily allow or reject leaves. This helps the user and the admin to intuitively interact for leaves.

In this documentation, we will give you a good overview of the website, and also the technical part on which it is developed.

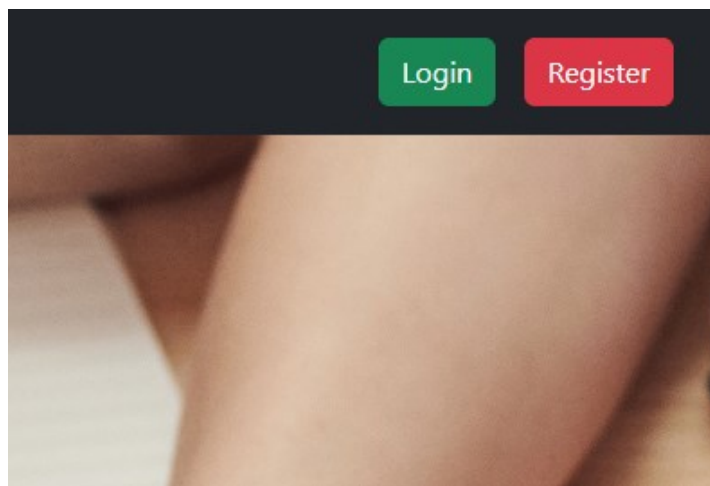
BASIC OVERVIEW

The website has a very simple and efficient UI.

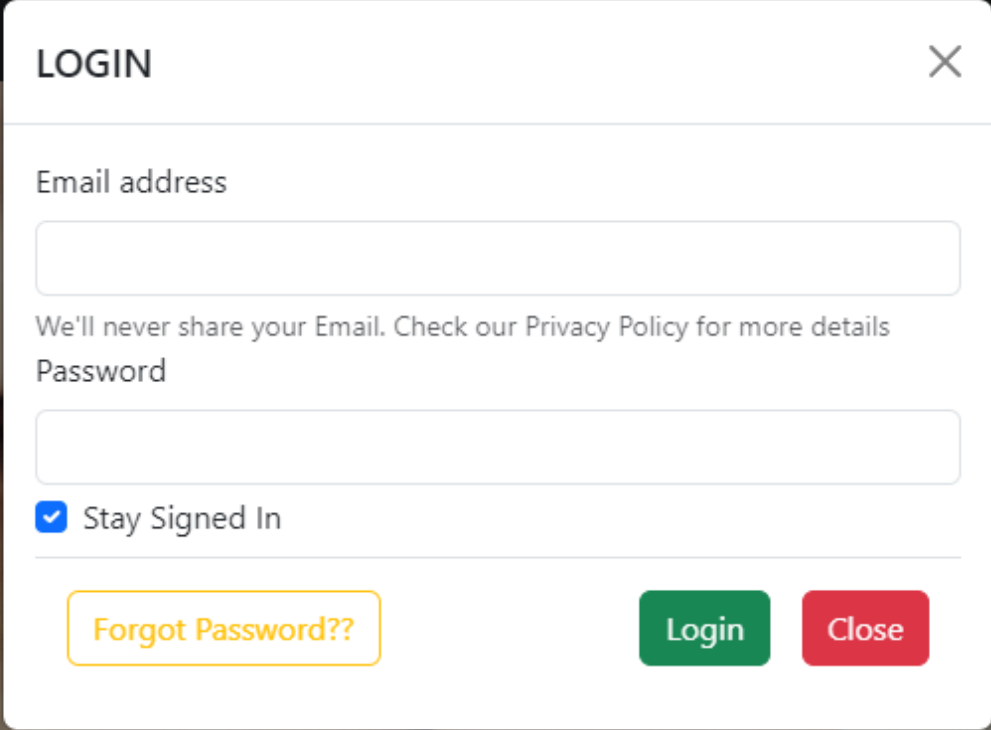
If a user has already not logged into the system, then the homepage appears.



As you can see, you can easily log-in from the top-right hand corner, Where the Login and Register Button is positioned.



From here, You can login with your given credentials. The User can login using his/her E-Mail and the password.



LOGIN

Email address

We'll never share your Email. Check our Privacy Policy for more details

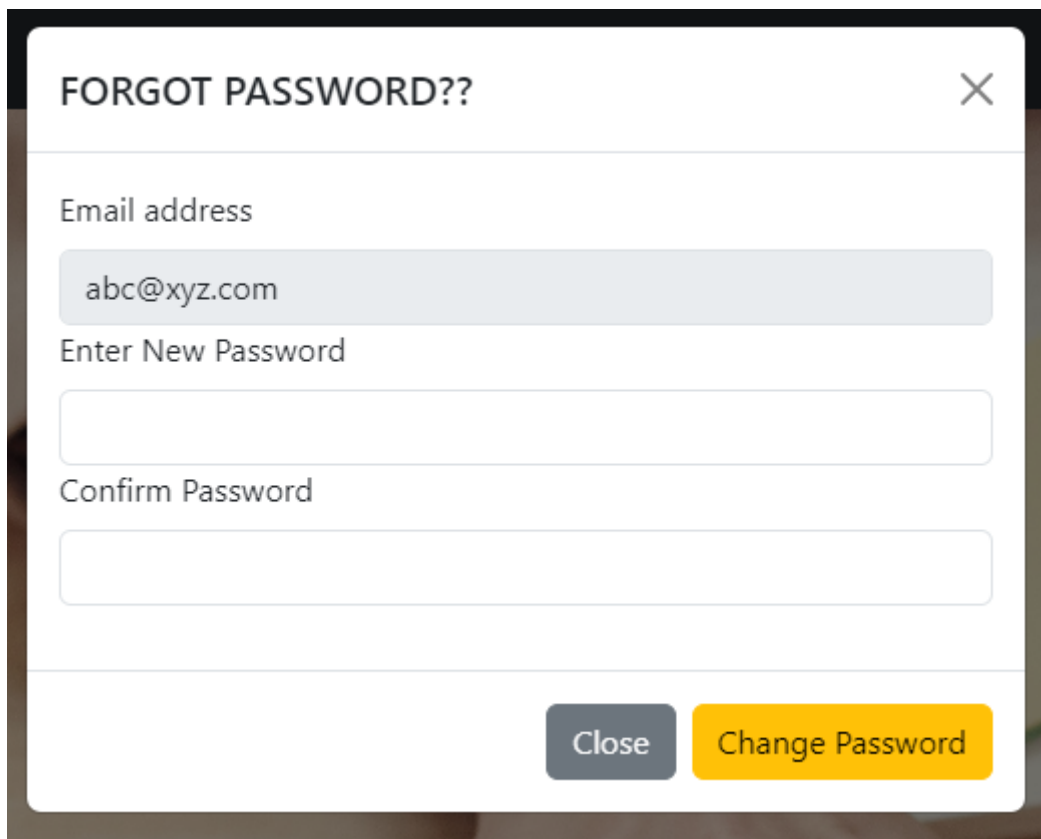
Password

☒ Stay Signed In

[Forgot Password??](#) [Login](#) [Close](#)

You can also check the **Stay Signed In** box. It will help you when you try to access the webpage again later, it can automatically log you in after a certain period.

If you have forgotten your password, it can be reset from the Forgot Password Section. It is very easy to use.

A modal dialog box titled "FORGOT PASSWORD??" with a close button (X) in the top right corner. The form contains three input fields: "Email address" with the value "abc@xyz.com", "Enter New Password", and "Confirm Password". At the bottom right, there are two buttons: a grey "Close" button and a yellow "Change Password" button.

FORGOT PASSWORD??

Email address

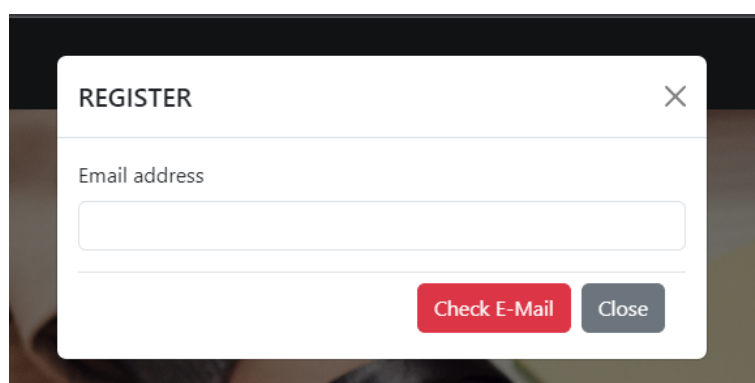
abc@xyz.com

Enter New Password

Confirm Password

Close Change Password

If you're a new user, you can register as a new user from the register section in the website. To avoid one user from creating 2 accounts, the E-Mail is verified first, if a user exists already, it will throw an error.

A modal dialog box titled "REGISTER" with a close button (X) in the top right corner. The form contains one input field labeled "Email address". At the bottom right, there are two buttons: a red "Check E-Mail" button and a grey "Close" button.

REGISTER

Email address

Check E-Mail Close

But If You're not already registered, you can Register using the data given.

REGISTER ✕

Email address

abc@xyz.com

Name

Gender

☐ Male ☐ Female

Role

☐ User ☐ Admin

Phone

Date of Birth

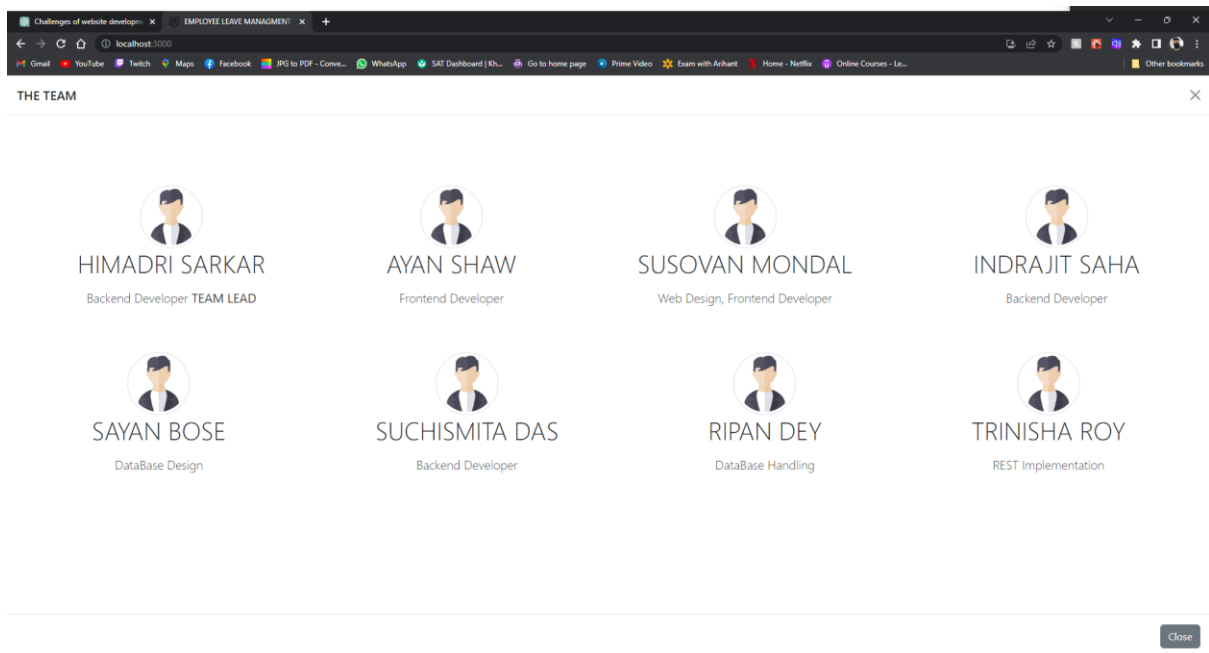
dd-mm-yyyy 📅

Password

Confirm Password

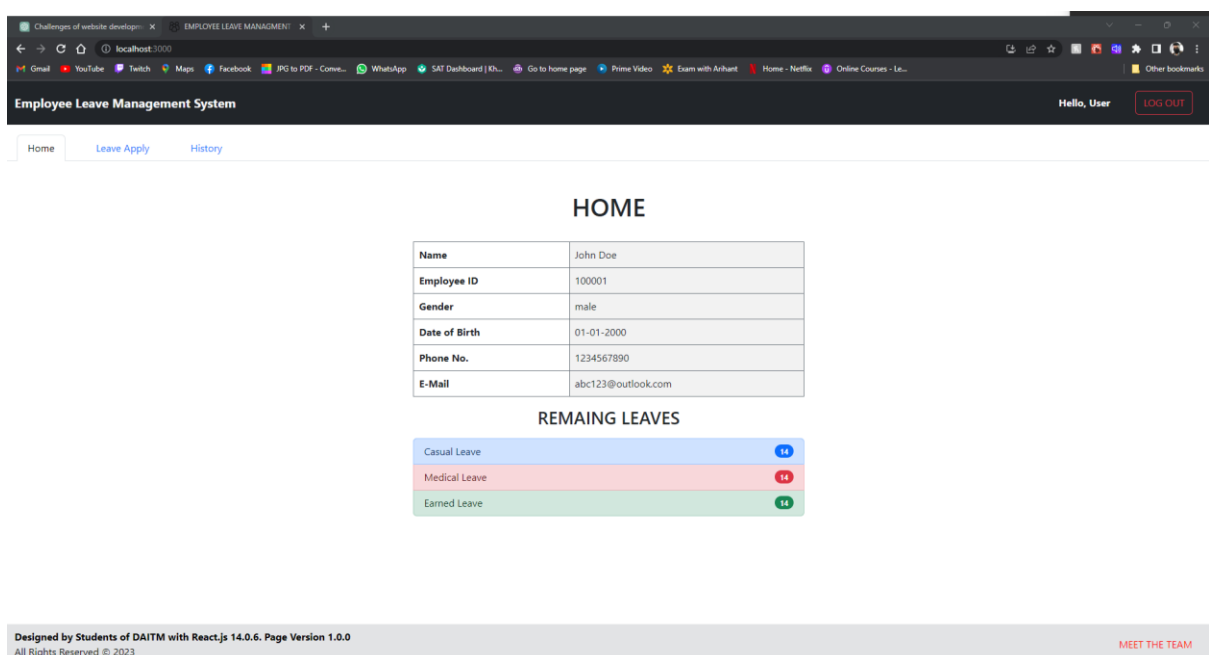
Register **Close**

There's also the meet the team button, it will give you preliminary information about our team.

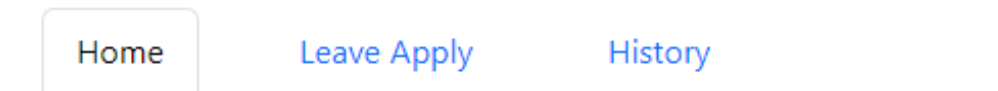


If you are already logged in, there are two parts of the page.

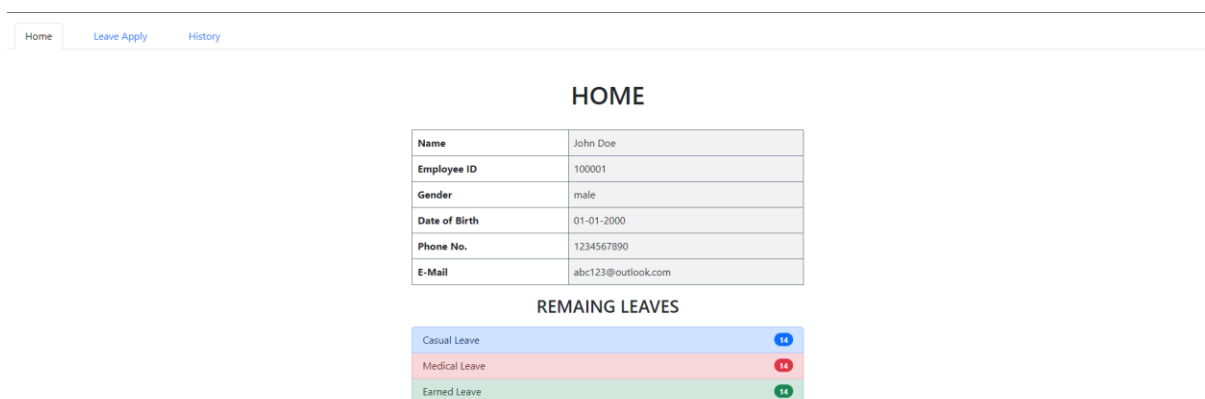
If You're a user, you will see the user specific page.



This Page has 3 areas, denoted in the navbar, The Home Page, Leave Apply Page and The History Page.



The Home Page primarily consists of user information, Here You can see the user Information and the remaining leaves.



And then there is the Leave Apply Page, from here, you can apply for leaves. There you can show the remaining and type of leave you want to apply, the start and end date, and the reason for application, on submission, The data is sent to admin for review.

[Home](#) [Leave Apply](#) [History](#)

APPLY FOR LEAVE

Casual Leave 14

Medical Leave 14

Earned Leave 14

Select Type of Leave you want

Start Date
dd-mm-yyyy

End Date
dd-mm-yyyy

Reason

Submit

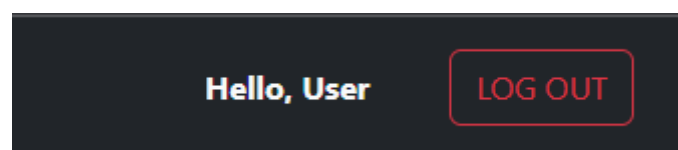
And at last, there is the history page, where you can see the previously applied leaves, The rows also change color based on if the leave was rejected, granted or pending.

[Home](#) [Leave Apply](#) [History](#)

HISTORY

Serial No.	Leave Type	Reason	Date Applied	No. of Days	Status
1	Casual Leave	Going to hometown lol ...	2021-10-10	2	Rejected

And at last, there's the logout button placed in the top right corner at the most convenience.



As for the Admin Page, it also has the home page. It shows the admin's data and pending Leave Grants.

HOME	
Name	John Doe
Employee ID	100001
Gender	male
Date of Birth	01-01-2023
Phone No.	1234567890
E-Mail	ayanshw@outlook.com

Pending Leave Grants

Casual Leave	14
Medical Leave	14
Earned Leave	14
Password Change Requests	2

And the other page is the leave grant page, from there the admin can easily grant and reject leaves in a click of a button.

GRANT LEAVES							
Employee ID	Name	Leave Type	Reason	Date Applied	No. of Days	Grant or Reject the Leave	
100001	John Doe	Casual Leave	Going to hometown...	2021-10-10	2	Grant	Reject
100002	John Doe2	Casual Leave	Going to hometown...	2021-10-10	2	Grant	Reject
100003	John Doe3	Casual Leave	Going to hometown...	2021-10-10	2	Grant	Reject

This page also has the logout button.

This is the whole page, explained.

CHALLENGES

Creating a webpage comes with its challenges. And creating a webpage that manages employee's leaves makes for another challenge.

If we want to make an employee leave management system, we must think about these:

1. **Integration:** An employee leave management system must integrate with other HR systems, such as payroll and employee data management. Ensuring that the system works seamlessly with other systems can be a complex task.
2. **Data Accuracy:** The system must accurately capture employee leave requests, approvals, and balances. Inaccurate data can lead to problems with payroll, employee benefits, and scheduling.
3. **Compliance:** Leave management systems must comply with labor laws and company policies. Ensuring that the system is up-to-date with all relevant regulations can be a complex task.

4. **Accessibility:** The system must be accessible to all employees, including those with disabilities. Ensuring that the system meets accessibility guidelines can be a challenge.
5. **Security:** The system must be secure and protect employee information. This includes ensuring that the system is only accessible by authorized personnel and that the data is stored securely.
6. **Reporting:** The system should provide reports on employee leave requests, approvals, and balances. The reports should be easy to generate and provide valuable insights for HR managers.
7. **Scalability:** The system should be scalable and able to handle the needs of a growing organization. This includes being able to handle an increasing number of leave requests and users.

While these are some pretty hard challenges, when designing a webpage around these, we must also think about:

- 1.Planning and Design:** Before any coding can begin, a solid plan and design need to be created. This requires careful consideration of the website's purpose, audience, content, and functionality. Without proper planning, the website may lack direction, usability, and appeal.
- 2.Technical Expertise:** Developing a website requires technical knowledge of programming languages, web design principles, and database management. It may be necessary to hire a web developer or a team of developers to ensure the website is built with clean, efficient, and effective code.
- 3.Compatibility:** Websites need to be compatible with various browsers, devices, and screen sizes. Ensuring that the website works on all platforms requires thorough testing and debugging.

4. **Content Management:** A website's content needs to be organized, structured, and easy to update. The content management system (CMS) used should allow for easy editing and publishing of content without compromising the website's design or functionality.
5. **Security:** Websites are vulnerable to hacking and cyber-attacks, and sensitive information must be protected. A secure website requires regular software updates, secure passwords, SSL certificates, and other security measures.
6. **Search Engine Optimization (SEO):** To attract visitors, a website must be optimized for search engines. This involves careful keyword research, on-page optimization, and off-page optimization.
7. **User Experience (UX):** The website's usability and user experience are critical to its success. User experience design involves creating an intuitive and enjoyable experience for the user, from navigation and search to functionality and responsiveness.

8. Budget: Developing a website can be expensive, and the costs can vary depending on the scope and complexity of the project. It's important to establish a realistic budget and timeline before starting the development process.

FRONT-END DEVELOPMENT DECISION

While developing in core HTML, CSS and JavaScript is easy and can be done, it is not efficient and could be problematic due to rendering issues, that's why we wanted to make the frontend with framework, which makes it easy and more dynamic.

But Choosing a Front-End framework is not easy as it seems, there are some challenges which include:

1. Learning curve
2. Compatibility issues
3. Maintenance requirements
4. Customization limitations
5. Performance impact
6. Licensing restrictions
7. Security vulnerabilities
8. Lack of support resources

That's why we chose React.js, which is easy to learn, dynamic and has some basic security.

REACT.JS FRONT-END DEVELOPMENT

Developing with React.js has its benefits. React.js is popular because it offers a component-based architecture, virtual DOM, large community and ecosystem, easy learning curve, flexibility, SEO-friendly features, scalability, and is backed by Facebook.

There are some features like components in React.js which are used extensively in our website.

In React, every webpage is built up of multiple parts/chunks, called components.

Components are one of the core concepts of React. They are the foundation upon which you build user interfaces (UI).

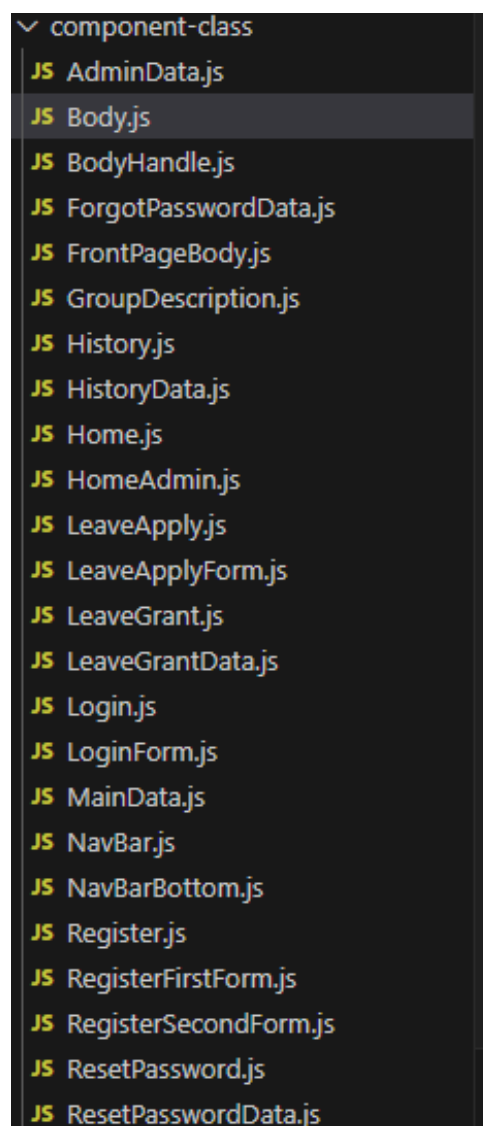
Your React application begins at a “root” component. Usually, it is created automatically when you start a new project. For example, if you use Create React App, the root component is defined in `src/App.js`.

Most React apps use components all the way down. This means that you won't only use components for reusable pieces like buttons, but

also for larger pieces like sidebars, lists, and ultimately, complete pages! Components are a handy way to organize UI code and markup, even if some of them are only used once.

For more information about components visit: <https://react.dev/learn/your-first-component#components-ui-building-blocks>

In our case we have made chunks of our pages a component, which are shown below:



As we've used function-based components instead of classes, every component is a function instead of being part of a class.

The main "root" is shown as below:

```
function App() {
  // const isLoggedIn = () => {
  //   const token = localStorage.getItem("token");
  //   if (token !== null) {
  //     return true;
  //   } else {
  //     return false;
  //   }
  // }
  const isLoggedIn = false;
  if (isLoggedIn) {
    const datain = {
      name: "User",
      accessSpecifier: "admin"
    }
    return (
      <>
        <NavBar name={datain.name} />
        <BodyHandle accessSpecifier={datain.accessSpecifier} />
        <NavBarBottom />
      </>
    );
  }
  else {
    return (
      <>
        /* DONT MESS WITH THIS, if something doesn't work, it's your fault */
        <NavBar name="" />
        <Login />
        <Register />
        <FrontPageBody />
        <NavBarBottom />
      </>
    );
  }
}
export default App;
```

As we can see there are multiple components used in the core page, so it's not crowded at all. As for the main front page it consists of 5 components.

```
return (  
  <>  
    { /* DONT MESS WITH THIS, if something doesn't work, it's your fault */ }  
    <NavBar name="" />  
    <Login />  
    <Register />  
    <FrontPageBody />  
    <NavBarBottom />  
  
  </>  
);
```

All these components are JavaScript Functions, which are separated from each other.

JavaScripts are typically vulnerable but due to React's security features and server-side rendering, it's pretty harmless in this context.

React also has a function called Hooks, with them core React functions are easily implemented. An example will be a `{useState}` hook, which is heavily used in our site.

An example would be whenever we take an input it stores the value as key-value pair.

```
const loginChangeHandler = (event) => {
  const name = event.target.name;
  const value = event.target.value;
  setInputs(values => ({...values, [name]: value}))
}
```

The `useEffect` Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

`useEffect` accepts two arguments. The second argument is optional.

`useEffect(<function>, <dependency>)`

```
useEffect(() => {
  axios.post(`http://localhost:8086/api/emp/empById/${pdleaves.id}`)
    .then(response => setEmployeeName(response.data.employeeName))
}, [pdleaves.id])
```

We have also used `Toastify` which is a node.js package.

In Node.js, `Toastify` is a library that provides an easy way to display toast notifications. Toast notifications are small, non-intrusive pop-up messages that provide brief information or feedback to the user.

The Toastify library in Node.js allows you to create and display toast notifications in your command-line or terminal-based applications. It provides a simple API for configuring and customizing the appearance and behaviour of the toast notifications.

```
var a = window.sessionStorage.getItem('toastflag');
if (a) {
  var ttype = window.sessionStorage.getItem('toasttype');
  var tmsg = window.sessionStorage.getItem('toastmsg');
  switch (ttype) {
    case 'success':
      toast.success(tmsg, {
        position: toast.POSITION.TOP_CENTER,
      });
      break;
    case 'error':
      toast.error(tmsg, {
        position: toast.POSITION.TOP_CENTER,
      });
      break;
    case 'warn':
      toast.warn(tmsg, {
        position: toast.POSITION.TOP_CENTER,
      });
      break;
    case 'info':
      toast.info(tmsg, {
        position: toast.POSITION.TOP_CENTER,
      });
      break;
    default:
      toast.error("Error in toast", {
        position: toast.POSITION.TOP_CENTER,
      });
      break;
  }
}
```

BOOTSTRAP FRAMEWORKS

While using plain CSS is easy and accessible, we want to use a CSS framework to make it easy and better for rendering and writing ease.

CSS frameworks offer pre-written CSS styles and rules that can save time, ensure consistency, enable responsive design, ensure cross-browser compatibility, provide community support, allow customization, and aid in rapid prototyping. However, they may not be suitable for every project and can introduce dependencies and code bloat if not used wisely.

We can choose a lot of frameworks, Bootstrap and Tailwind CSS are the most famous, but we used bootstrap due to its ease of use and compatibility with everything.

Some of the elements of Bootstrap we used are:

Layout

For layout we used Containers which are the most basic layout element in Bootstrap and are required when using our default grid system. Containers are used to contain, pad, and (sometimes) center the

content within them. While containers *can* be nested, most layouts do not require a nested container.

Bootstrap comes with three different containers:

- `.container`, which sets a max-width at each responsive breakpoint
- `.container-{breakpoint}`, which is width: 100% until the specified breakpoint
- `.container-fluid`, which is width: 100% at all breakpoints

Forms

Bootstrap's form controls expand on our Rebooted form styles with classes. Use these classes to opt into their customized displays for a more consistent rendering across browsers and devices.

Be sure to use an appropriate type attribute on all inputs (e.g., `email` for email address or `number` for numerical information) to take advantage of newer input controls like email verification, number selection, and more.

Add the `disabled` attribute to a `<fieldset>` to disable all the controls within. Browsers treat all native form controls (`<input>`, `<select>`, and `<button>` elements) inside a `<fieldset disabled>` as disabled, preventing both keyboard and mouse interactions on them.

However, if your form also includes custom button-like elements such as `...`, these will only be given a style of pointer-events: none, meaning they are still focusable and operable using the keyboard. In this case, you must manually modify these controls by adding `tabindex="-1"` to prevent them from receiving focus and `aria-disabled="disabled"` to signal their state to assistive technologies.

Ensure that all form controls have an appropriate accessible name so that their purpose can be conveyed to users of assistive technologies. The simplest way to achieve this is to use a `<label>` element, or—in the case of buttons—to include sufficiently descriptive text as part of the `<button>...</button>` content.

For situations where it's not possible to include a visible `<label>` or appropriate text content, there are alternative ways of still providing an accessible name, such as:

- `<label>` elements hidden using the `.visually-hidden` class
- Pointing to an existing element that can act as a label using `aria-labelledby`
- Providing a title attribute
- Explicitly setting the accessible name on an element using `aria-label`

If none of these are present, assistive technologies may resort to using the placeholder attribute as a fallback for the accessible name on `<input>` and `<textarea>` elements.

Also browser default checkboxes and radios are replaced with the help of `.form-check`, a series of classes for both input types that improves the layout and behaviour of their HTML elements, that provide greater customization and cross browser consistency. Checkboxes are for selecting one or

several options in a list, while radios are for selecting one option from many.

Structurally, our `<input>`s and `<label>`s are sibling elements as opposed to an `<input>` within a `<label>`. This is slightly more verbose as you must specify id and for attributes to relate the `<input>` and `<label>`. We use the sibling selector (`~`) for all our `<input>` states, like `:checked` or `:disabled`. When combined with the `.form-check-label` class, we can easily style the text for each item based on the `<input>`'s state.

Our checks use custom Bootstrap icons to indicate checked or indeterminate states.

Components

Bootstrap has a base `.btn` class that sets up basic styles such as padding and content alignment. By default, `.btn` controls have a transparent border and background color, and lack any explicit focus and hover styles.

The `.btn` classes are designed to be used with the `<button>` element. However, you can also use these classes on `<a>` or `<input>` elements (though

some browsers may apply a slightly different rendering).

When using button classes on `<a>` elements that are used to trigger in-page functionality (like collapsing content), rather than linking to new pages or sections within the current page, these links should be given a `role="button"` to appropriately convey their purpose to assistive technologies such as screen readers.

Before getting started with Bootstrap's modal component, be sure to read the following as our menu options have recently changed.

- Modals are built with HTML, CSS, and JavaScript. They're positioned over everything else in the document and remove scroll from the `<body>` so that modal content scrolls instead.
- Clicking on the modal "backdrop" will automatically close the modal.
- Bootstrap only supports one modal window at a time. Nested modals aren't supported as we believe them to be poor user experiences.
- Modals use `position: fixed`, which can sometimes be a bit particular about its

rendering. Whenever possible, place your modal HTML in a top-level position to avoid potential interference from other elements. You'll likely run into issues when nesting a `.modal` within another fixed element.

- Once again, due to `position: fixed`, there are some caveats with using modals on mobile devices. See our browser support docs for details.
- Due to how HTML5 defines its semantics, the `autofocus` HTML attribute has no effect in Bootstrap modals.

Documentation and examples for Bootstrap's powerful, responsive navigation header, the navbar. Includes support for branding, navigation, and more, including support for our collapse plugin.

Here's what you need to know before getting started with the navbar:

- Navbars require a wrapping `.navbar` with `.navbar-expand{-sm|-md|-lg|-xl|-xxl}` for responsive collapsing and color scheme classes.

- Navbars and their contents are fluid by default. Change the container to limit their horizontal width in different ways.
- Use our spacing and flex utility classes for controlling spacing and alignment within navbars.
- Navbars are responsive by default, but you can easily modify them to change that. Responsive behaviour depends on our Collapse JavaScript plugin.
- Ensure accessibility by using a `<nav>` element or, if using a more generic element such as a `<div>`, add a `role="navigation"` to every navbar to explicitly identify it as a landmark region for users of assistive technologies.
- Indicate the current item by using `aria-current="page"` for the current page or `aria-current="true"` for the current item in a set.
- **New in v5.2.0:** Navbars can be themed with CSS variables that are scoped to the `.navbar` base class. `.navbar-light` has been deprecated and `.navbar-dark` has been rewritten to override CSS variables instead of adding additional styles.

Navbars come with built-in support for a handful of sub-components. Choose from the following as needed:

`.navbar-brand` for your company, product, or project name.

`.navbar-nav` for a full-height and lightweight navigation (including support for dropdowns).

`.navbar-toggler` for use with our collapse plugin and other navigation toggling behaviours.

Flex and spacing utilities for any form controls and actions.

`.navbar-text` for adding vertically centred strings of text.

`.collapse.navbar-collapse` for grouping and hiding navbar contents by a parent breakpoint.

Add an optional `.navbar-scroll` to set a max-height and scroll expanded navbar content.

Documentation and examples for how to use Bootstrap's included navigation components.

Navigation available in Bootstrap share general markup and styles, from the base `.nav` class to the active and disabled states. Swap modifier classes to switch between each style.

The base `.nav` component is built with flexbox and provide a strong foundation for building all types of navigation components. It includes some style overrides (for working with lists), some link padding for larger hit areas, and basic disabled styling.

Classes are used throughout, so your markup can be super flexible. Use ``s like above, `` if the order of your items is important, or roll your own with a `<nav>` element. Because the `.nav` uses `display: flex`, the nav links behave the same as nav items would, but without the extra markup.

BACK-END DEVELOPMENT DECISION

Java is a popular choice for backend development due to several key reasons:

1. Platform Independence: Java is a cross-platform language, which means that code written in Java can run on any operating system that has a Java Virtual Machine (JVM) installed. This makes it highly versatile and allows developers to write code once and run it on multiple platforms without needing to make significant changes.

2. Large Community and Ecosystem: Java has been around for decades and has a vast and active community of developers. This results in a rich ecosystem of libraries, frameworks, and tools that make development faster and more efficient. The availability of resources and community support is beneficial for troubleshooting, learning, and collaborating with other developers.

3. Scalability and Performance: Java is designed to handle large-scale enterprise applications. It provides excellent performance and scalability, making it suitable for building robust and high-

performance backend systems. Java's Just-In-Time (JIT) compilation and advanced memory management contribute to its efficiency.

4. Object-Oriented Programming (OOP): Java is an object-oriented language, which promotes modular, reusable, and maintainable code. This approach allows developers to structure their code in a way that closely reflects real-world entities, making it easier to design and maintain complex systems.

5. Security: Java has built-in security features that help developers build secure applications. It provides a robust security model that includes features like class loaders, bytecode verification, and a strong security manager. Additionally, the Java ecosystem offers numerous security libraries and tools to further enhance application security.

6. Industry Adoption: Java has been widely adopted by many large organizations and enterprises, making it a reliable and trusted choice for backend development. Its maturity, stability, and compatibility with legacy systems make it an attractive option for businesses.

7. Integration Capabilities: Java provides extensive support for integration with other systems and technologies. It has robust APIs for working with databases, web services, messaging systems, and more. This enables developers to build backend solutions that seamlessly interact with other components of the software architecture.

Overall, the combination of platform independence, a strong ecosystem, scalability, performance, security, and industry adoption makes Java a popular choice for backend development. However, it's important to note that the choice of programming language ultimately depends on the specific requirements of the project and the expertise of the development team.

JAVA DEVELOPMENT

When it comes to backend development in Java, there are several important decisions to consider. Here are some key aspects to keep in mind:

1. Introduction:

- Provide a brief introduction to the project and its purpose.
- Mention the technologies involved: Java, MySQL, and Spring Boot.

2. Architecture Overview:

- Describe the overall architecture of the application.
- Explain the layered approach, such as presentation layer, service layer, and data access layer.
- Discuss the separation of concerns and the benefits it offers.

3. MySQL Database:

- Explain the role of MySQL in the project.
- Describe the database schema and its tables.

- List the key entities and their relationships.
- Document any specific constraints or business rules enforced by the database.

4. Spring Boot:

- Describe the role of Spring Boot in the project.
- Explain the use of Spring Boot's dependency management and auto-configuration features.
- Mention the main Spring Boot components used, such as the application class and configuration files.
- Discuss the use of Spring Data JPA for interacting with the database.

5. Key Components:

- Describe the main components of the application.
- Discuss the purpose and responsibilities of each component.
- Highlight any custom logic or functionality implemented in each component.
- Explain how the components interact with each other.

6. Data Access Layer:

- Discuss the data access layer, responsible for interacting with the database.
- Explain the use of Spring Data JPA and its annotations for defining repositories and querying the database.
- Document any custom queries or complex database operations.

7. Service Layer:

- Describe the service layer, responsible for implementing business logic.
- Document the main services and their responsibilities.
- Explain how the services utilize the data access layer to perform database operations.

8. Presentation Layer:

- Discuss the presentation layer, responsible for handling user requests and generating responses.
- Explain the use of controllers and their mappings to handle REST API endpoints.

- Document any input validation or data transformation performed by the controllers.

9. API Documentation:

- Provide details on the REST API endpoints exposed by the application.
- Document each endpoint's URL, HTTP methods, request/response structures, and any required headers or parameters.
- Include example requests and responses to illustrate the API usage.

10. Deployment and Configuration:

- Explain how the application is deployed and configured.
- Document any required environment variables or configuration files.
- Mention any external services or resources the application depends on.

RESTFUL API IMPLEMENTATION

To implement a REST API in Java without providing actual code, I can provide you with an outline of the steps involved in creating a REST API and some high-level explanations of the concepts and components involved. This will serve as a documentation guide for creating a REST API in Java:

Define API Endpoints:

Determine the resources and operations your API will expose.

For example, you might have endpoints like `/users` to retrieve all users or `/users/{id}` to retrieve a specific user.

HTTP Methods:

Assign appropriate HTTP methods (GET, POST, PUT, DELETE) to each endpoint.

GET: Retrieve a resource or a collection of resources.

POST: Create a new resource.

PUT: Update an existing resource.

DELETE: Remove a resource.

Request and Response Formats:

Decide on the data format for request payloads and response bodies.

Common formats include JSON (JavaScript Object Notation) and XML.

Specify the structure and required fields for each endpoint's request and response payloads.

RESTful URLs:

Create RESTful URLs that reflect the resources and operations.

Use path parameters to handle dynamic parts of the URLs (e.g., [/delete/{employeeid}](#)).

```

1 package com.RL.RegLog.controller;
2
3 import java.util.List;
4
5 @RestController
6 @CrossOrigin
7 @RequestMapping("/api/emp")
8 public class EmployeeController {
9
10     @Autowired
11     private EmployeeService employeeService;
12
13     @PostMapping(path = "/save")
14     public String createEmp(@RequestBody EmployeeDTO employeeDTO)
15     {
16         String id = employeeService.addEmployee(employeeDTO);
17         return id;
18     }
19
20     @PostMapping(path = "/login")
21     public ResponseEntity<?> loginEmp(@RequestBody LoginDTO loginDTO)
22     {
23         LoginMessage loginMessage = employeeService.loginEmployee(loginDTO);
24         return ResponseEntity.ok(loginMessage);
25     }
26
27     @PutMapping(path = "/update")
28     public String updateById(@RequestBody Employee emp)
29     {
30         return employeeService.updateById(emp);
31     }
32
33     @DeleteMapping(path = "/delete/{employeeid}")
34     public void delById(@PathVariable int employeeid)
35     {
36         employeeService.delById(employeeid);
37         System.out.print("Data Removed");
38     }
39
40 }

```

FRONT-END AND BACK-END CONNECTION

As the Front-End and Back-End are separate we must use some API to connect both servers. The most common way to do it is Representational State Transfer or the RESTful API and HTTP requests. REST APIs offer simplicity, scalability, compatibility, and a resource-based approach. They promote stateless communication, caching for performance, and adhere to a uniform interface. As an industry standard, REST facilitates easy integration and has a wealth of resources and tools available. However, consider specific use case requirements and alternative architectures for optimal solutions.

AXIOS FRAMEWORK IMPLEMENTATION

Axios is a promise-based HTTP library that lets developers make requests to either their own or a third-party server to fetch data. It offers different ways of making requests such as GET, POST, PUT/PATCH, and DELETE.

It's simple to use, works across browsers, and supports various request types. It offers interceptors for customization, built-in error handling, and works well with server-side rendering. With a strong community and extensive documentation, Axios is a reliable choice for handling HTTP requests in JavaScript applications.

Usage of Axios in this web page is spread across multiple components. For Example, in the history page we used POST function of Axios to get data from the server.

```

useEffect(() => {
  axios.post("http://localhost:8086/api/emp/leave_history/alllvh")
    .then(response => {
      setdataset(response.data);
    })
    .catch(error => console.log(error));
}, []);
return (
  <>
  <div className="header mt-5 d-flex justify-content-center">

```

And in the leave grant page Axios uses it's PUT function to update the database.

```

await axios.put("http://localhost:8086/api/emp/leave_type/update", updatedLeaves)
  .then(async response => {
    if (response.status === 200) {
      const updatedLeaveHistory = {
        lvh_id: pdleaves.leave_id,
        leavetype: pdleaves.leavetype,
        leavereason: pdleaves.reason,
        leaveapdate: pdleaves.dateapplied,
        leaved: pdleaves.noofdays,
        lvt_lvh_fk: pdleaves.id,
        status: false,
        acrejflag: true
      };
      await axios.put("http://localhost:8086/api/emp/leave_history/update", updatedLeaveHistory)
        .then(res => {
          if (res.status === 200) {
            // alert('Leave Granted to ${employeeName} Successful');
            window.sessionStorage.setItem('toastflag', true);
            window.sessionStorage.setItem('toasttype', 'info');
            window.sessionStorage.setItem('toastmsg', 'Leave Grant to ${employeeName} Successfully');
            window.sessionStorage.setItem('redirect', 'leavegrant');
          }
          else{
            window.sessionStorage.setItem('toastflag', true);
            window.sessionStorage.setItem('toasttype', 'error');
            window.sessionStorage.setItem('toastmsg', "Leave Application failed due to Error: "+res.status);
            window.sessionStorage.setItem('redirect', 'leavegrant');
          }
        })
    }
  })

```

DATABASE INTERACTION

Java Spring Boot is a popular framework for building Java applications, and it provides excellent support for interacting with databases. In this response, we need to keep in mind the following points of how Spring Boot facilitates database interaction:

- 1.Database Setup:** Configure and connect to a relational database management system (such as MySQL, PostgreSQL) in the Spring Boot application.
- 2.Entity Mapping:** Create entity classes representing database tables, such as [Employee](#), [LeaveRequest](#), [LeaveType](#), etc. Use JPA annotations ([@Entity](#), [@Table](#), [@Column](#)) to define the mappings between entities and database tables/columns.
- 3.Repository Interface:** Define repository interfaces that extend the [JpaRepository](#) interface provided by Spring Data JPA. These interfaces will provide CRUD (Create, Read, Update, Delete) operations and additional

query methods for interacting with the database.

4. **Create Operation:** To create a new leave request, invoke the appropriate method on the repository interface, passing in the required data. The entity will be persisted in the database.
5. **Read Operations:** Retrieve leave requests based on various criteria like employee ID, leave status, date range, etc. Use repository methods like `findById`, `findAll`, `findByEmployeeId`, etc., to fetch the required data from the database.
6. **Update Operation:** Modify the properties of a leave request entity and save it back to the database. Use repository methods like `saveOrUpdate` to update the existing leave request.
7. **Delete Operation:** Remove a leave request entity from the database using repository methods like `delete`.

8. Querying: Define custom query methods using the Spring Data JPA query creation mechanism or write native SQL queries using `@Query` annotation to perform complex database queries.

9. Database Constraints: Define constraints at the entity level to enforce data integrity rules in the database.

```

1 package com.RL.RegLog.controller;
2
3 import java.util.List;
4
5
6
7
8
9
10
11 @RestController
12 @CrossOrigin
13 @RequestMapping("api/emp/leave_history")
14 public class LeavehistoryController {
15
16
17     @Autowired
18     private LeavehistoryService leavehistoryService;
19
20
21     @PostMapping(path = "/save")
22     public Leave_history createLvh(@RequestBody Leave_history lvh)
23     {
24         return this.leavehistoryService.createleaveh(lvh);
25     }
26
27
28     @PostMapping(path = "/lvhById/{lvh_id}")
29     public Leave_history lvhById(@PathVariable int lvh_id)
30     {
31         return this.leavehistoryService.lvhById(lvh_id);
32     }
33
34
35     @PostMapping(path = "/lvhByFk/{lvt_lvh_fk}")
36     public Leave_history lvhByFk(@PathVariable int lvt_lvh_fk)
37     {
38         return this.leavehistoryService.lvhByFk(lvt_lvh_fk);
39     }
40
41
42     @PutMapping(path = "/update")
43     public Leave_history uplhbyId(@RequestBody Leave_history lvh)
44     {
45         return this.leavehistoryService.uplhbyId(lvh);
46     }
47
48
49     @DeleteMapping(path = "/delete/{lvh_id}")
50     public void dellhById(@PathVariable int lvh_id)
51     {
52         leavehistoryService.dellhById(lvh_id);
53     }
54
55
56
57
58
59

```

```

1 package com.RL.RegLog.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5
6
7
8
9
10
11
12
13
14
15
16
17 @RestController
18 @CrossOrigin
19 @RequestMapping("api/emp/leave_type")
20 public class LeavetypeController {
21
22     @Autowired
23     private LeavetypeService leavetypeService;
24
25     @PostMapping("/save")
26     public Leave_type createlvt(@RequestBody Leave_type lvt)
27     {
28         return this.leavetypeService.createLeave(lvt);
29     }
30
31     @PostMapping(path = "/lvtById/{lvt_id}")
32     public Leave_type lvtById(@PathVariable int lvt_id)
33     {
34         return this.leavetypeService.lvtById(lvt_id);
35     }
36
37     @PutMapping(path = "/update")
38     public Leave_type uplvtById(@RequestBody Leave_type lvt)
39     {
40         return this.leavetypeService.uplvtById(lvt);
41     }
42
43     @DeleteMapping(path = "/delete/{lvt_id}")
44     public void delById(@PathVariable int lvt_id)
45     {
46         leavetypeService.delleaveById(lvt_id);
47     }
48
49 }
50

```

```
mysql> select * from employee;
```

employee_id	email	employee_name	password	role
1	friendsgd55@gmail.com	HIMADRI SARKAR	\$2a\$10\$.wuMRC5HGx/980cZg09oH.fXh1N9d6Sx2K33MbUz4zvFoDbLL2h.	admin
2	friendsgp200@gmail.com	Sayan Bose	\$2a\$10\$KbTsnIGVjkp0ngY1QpyW3eUK2/83zBTzX5aJkxjR5LW8KaJ2pH5uG	user
3	ayanshw@outlook.com	Ayan Shaw	\$2a\$10\$gMBs0KjNX8k0dBGc.iJLKuIK.bKNwNszsxbx.Xg84H3kleIuz4ziZW	user
4	friendsgd555@gmail.com	Suchi Das	\$2a\$10\$jk3YGlvAcoBdZ0ouqzWEuJ9oVghdeLurqGOV0fZ0Y4vPB8Qh6kVi	user

4 rows in set (0.00 sec)

```
mysql> select * from leave_history;
```

lvh_id	acrejflag	leaveapdate	leaved	leavereason	leavetype	lvt_lvh_fk	status
3	0x01	2023-05-09	3	sick	casual	2	0x00
4	0x00	2023-05-09	3	sick	medical	2	0x00
5	0x01	2023-05-10	2	sick	earned	2	0x00
6	0x01	2023-05-10	3	sick	casual	4	0x00
9	0x01	2023-05-12	6	sick	medical	3	0x00

```
5 rows in set (0.00 sec)
```

```
mysql> select * from leave_type;
```

lvt_id	casual_leave	earned_leave	medical_leave
1	14	14	14
2	11	12	-5
3	14	14	8
4	11	14	14

```
4 rows in set (0.00 sec)
```

CONCERNS ABOUT SECURITY

Even if you think a system is bullet-proof, most of the times it isn't. There are multiple security challenges when designing a system and integrating it into the website.

There are several security challenges that can arise in an employee leave management system, including:

1. **Data privacy:** The system may contain sensitive information about employees, such as their personal details, leave balances, and reasons for taking leave. This information must be protected from unauthorized access and misuse.
2. **Access control:** The system must ensure that only authorized personnel can access and modify employee leave data. This can be done by implementing role-based access control and multi-factor authentication.
3. **System vulnerabilities:** The system may be vulnerable to various types of cyber-attacks, such as phishing, malware, and denial-of-

service attacks. Regular security audits and updates can help prevent these types of attacks.

4.Data loss or corruption: Employee leave data must be backed up regularly to prevent data loss or corruption due to hardware failure, software bugs, or other issues.

5.Compliance: The system must comply with relevant laws and regulations related to data privacy and security, such as the General Data Protection Regulation (GDPR) and the Health Insurance Portability and Accountability Act (HIPAA).

However, those can be easily solved. It is important to implement strong security measures, such as encryption, firewalls, and intrusion detection systems. Regular security audits and employee training on security best practices can also help prevent security breaches.

Now Designing a system into website goes to another deep rabbit-hole. There are lots of hacker's

just waiting for a vulnerability, so we must stay secure.

here are several security challenges that can arise in a website, including:

1. **SQL injection:** Hackers can exploit vulnerabilities in the website's database to inject malicious code and gain access to sensitive data.
2. **Cross-site scripting (XSS):** Attackers can inject malicious code into a website's HTML code to steal user information or take control of the website.
3. **Broken authentication and session management:** Weak authentication and session management can allow attackers to gain unauthorized access to user accounts.
4. **Malware and viruses:** Websites can be infected with malware or viruses that can compromise user data or damage the website.
5. **Distributed denial of service (DDoS) attacks:** Hackers can launch DDoS attacks to overwhelm

a website's servers, making the website unavailable to users.

6. Vulnerable plugins and third-party software:

Third-party plugins and software used on a website can introduce security vulnerabilities if they are not regularly updated and patched.

To mitigate these security challenges, it is important to implement strong security measures such as:

1. Implementing web application firewalls and intrusion detection/prevention systems.
2. Regularly updating and patching all software, plugins, and third-party services used on the website.
3. Implementing strong authentication and session management practices, such as two-factor authentication and session timeouts.
4. Conducting regular security audits and penetration testing to identify vulnerabilities.
5. Backing up the website's data regularly to ensure that data can be restored in the event of a security breach.

6. Educating website users and staff about security best practices, such as not reusing passwords and being cautious when clicking on links or downloading attachments.

A FUTURE SIGHT

Though looking modern and stylish, a lot more features can be implemented or can be planned in our system.

A Forgot Password Section, could be added if we want to expand our project beyond our current development capacity.

We could also add a dark mode feature, which could be better if the user accesses the webpage at night.

We could also add a dynamic update and notification system, but doing that will require an always-on server which is not feasible for development at this moment.

We could have also developed a native-react app for using in phones and other handhelds, but that's for the future.

Even though there are many things we have, the sky is limit to the imagination.

CONCLUSION

In conclusion, an effective employee leave management system is critical for ensuring that an organization's workforce is managed efficiently and effectively. This whitepaper has explored the key features and benefits of such a system, including the ability to automate leave requests, manage employee leave balances, and track leave patterns and trends.

By implementing a robust leave management system, organizations can streamline their HR processes, reduce administrative workload, and improve overall workforce management. Additionally, by providing employees with greater control and visibility over their leave schedules, organizations can improve employee satisfaction and engagement, ultimately leading to improved productivity and performance.

However, it is important to note that implementing an effective leave management system requires careful planning and consideration of factors such as security, compliance, and integration with other HR systems. By taking these factors into account

and selecting a system that meets the specific needs of their organization, HR leaders can ensure that their workforce is managed in the most efficient and effective way possible.

BIBLIOGRAPHY

1. <https://react.dev/learn>
2. <https://www.w3schools.com/>
3. <https://docs.spring.io/spring-boot/docs/current/reference/html/getting-started.html#getting-started>
4. <https://hibernate.org/orm/documentation/getting-started/>
5. <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
6. <https://javaee.github.io/javaee-spec/javadocs/>
7. <https://axios-http.com/docs/intro>
8. <https://restfulapi.net/>

All information found on this webpage are open-source or easily available on the internet.

COPYRIGHT INFORMATION

All the resources used in this website are either open-source or used with proper permission from its owner and can be implemented into any system. However, implementation on our website is proprietary and designed by us, where “us” refers to us students of DAITM with identities mention in the first-page and others, and cannot be replicated without sole permission from us. Deferring to which is by definition illegal and could cause legal issues.

The picture is provided by FreePik, with proper permission and legal rights.

React, React.js and Its components is owned and developed my Facebook/Meta Inc. and used with open-source MIT License with terms defined at:
<https://opensource.fb.com/legal/terms/>.

Node.js and its components are owned by Microsoft Corp and OpenJS Foundation and used with open-source MIT License with terms defined at <https://terms-of-use.openjsf.org/>.

Java, Spring Boot and HIBERNATE are all owned by Oracle, VM-Ware and Red Hat respectively and open source.

Bootstrap CSS is open source with MIT License.

All Rights Reserved.