FULL STACK DEV

Introduction to Vue.js as frontend

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

INTRODUCTION TO VUE.JS

- Vue.js offers a progressive framework approach, being approachable for beginners while powerful enough for complex applications. We'll use modern Vue 3 syntax, specifically the Composition API with <script setup>, which provides a very clean and explicit way to write component logic.
- As with Angular and React, your Spring Boot backend will remain the same, serving data from http://localhost:8080/api/students. You'll just need to ensure your backend's CORS configuration allows requests from Vue's default development server port, which is typically http://localhost:5176 (since we'll use Vite).
- For this section, we will tackle the Human Resource application which is build using Vue and mySQL.

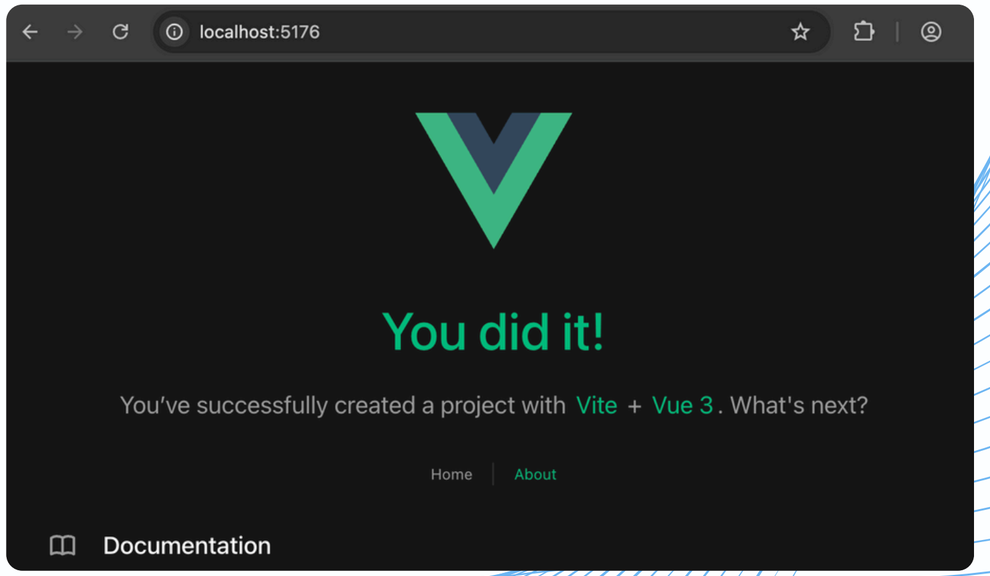
ARCHITECTURE

- **@CrossOrigin(origins = "http://localhost:5176")** → in controller !

- Create new directory inside your git repo - here it is frontend folder inside HumanResourceApp
  - Navigate inside this directory
  - Run : **>> npm create vue@latest frontend**
- Make sure you install extensions for Vue in Visual Studio Code
  - Check all options
  - Navigate into your new project directory (here it is frontend) and install dependencies
  - **>>npm install**
- Run the development server to verify everything works
  - **>>npm run dev**

INSTALLING VUE.JS

```
(base) rajeev@Rajeev-Khoodeeram git % cd session6-vue
(base) rajeev@Rajeev-Khoodeeram session6-vue % npm create vue@latest
Need to install the following packages:
create-vue@3.18.0
Ok to proceed? (y) y


> npx
> "create-vue"

┌  Vue.js - The Progressive JavaScript Framework
│
◇  Project name (target directory):
│  vue-project
│
◆  Select features to include in your project: (↑/↓ to navigate, space to
select, a to toggle all, enter to confirm)
│  ■ TypeScript
│  ■ JSX Support
│  ■ Router (SPA development)
│  ■ Pinia (state management)
│  ■ Vitest (unit testing)
│  ■ End-to-End Testing
│  ■ ESLint (error prevention)
│  ■ Prettier (code formatting)
└
```



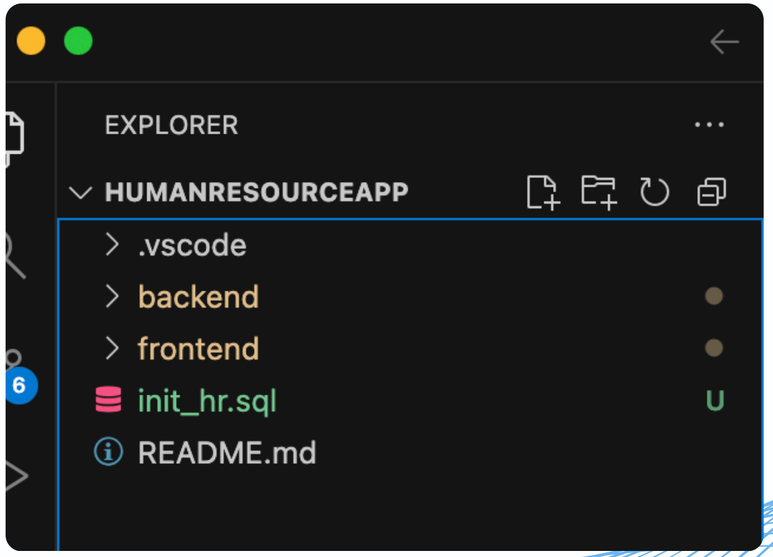
USING A TEMPLATE

- But For this section, I will show you how to work with a downloaded template :shadcn-vue-landing-page

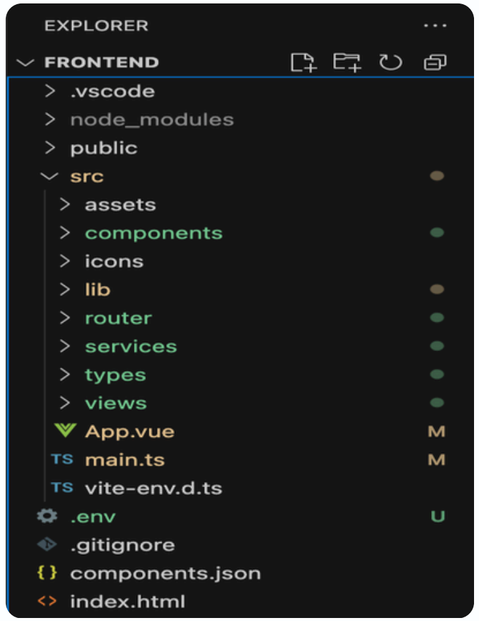
- Rename it as frontend and put inside HumanResourceApp

- So your folder should look like this :

- HumanResourceApp
  - frontend
  - backend



DON'T FORGET THE CSS FILE



- Open frontend folder in VS Code
  - >>npm install

- Remember
  - index.html calls —> main.ts which calls —> App.vue

- Run npm install vue-router (will create folder router and add index.js (see github))
  - This provides all the routes for your app - if not generated then you create it manually !!

- Important to note here - We will have two interfaces :
  - one is the website the public views
  - the other one is the admin view for managing the database

MAIN.TS

We will modify the main.ts to cater for routing
main.ts

```
import { createApp } from "vue";
import App from "./App.vue";
import "./assets/index.css";
import router from "./router/index.js";

const app = createApp(App);
app.use(router);
app.mount("#app");
```

main.ts has been modified to take routing into consideration

INDEX.JS FOR ROUTING

```
const routes = [
  {
    path: "/",
    component: PublicLayout,
    children: [
      { path: "", component: Home },
      { path: "about", component: About },
    ],
  },
  {
    path: "/admin",
    component: AdminLayout,
    children: [
    { path: "employees", component: () => import("@/views/admin/Employees.vue") },
    { path: "department", component: () => import("@/views/admin/Department.vue") },
    { path: "employees/create", component: () => import("@/components/employee/EmployeeNew.vue") },
    { path: "employees/edit/:id", component: () => import("@/components/employee/EmployeeNew.vue"), props:
  ],
  }
];
```

MAIN.TS

We will modify the main.ts to cater for routing
main.ts

```
import { createApp } from "vue";
import App from "./App.vue";
import "./assets/index.css";
import router from "./router/index.js";

const app = createApp(App);
app.use(router);
app.mount("#app");
```

main.ts has been modified to take routing into consideration

UTILITIES FILE

- Centralized ts or utils files (it is in the lib folder)
  - Create a utils folder in src
  - Create your file (ex here formatDate.ts) or add your function in the same utils.ts

- Formatting date from database in Due
  - >>npm install dayjs

- import dayjs from "dayjs";

- export function formatDate(dateStr: string): string {
- return dayjs(dateStr).format("DD-MM-YYYY");
- }

RECAP

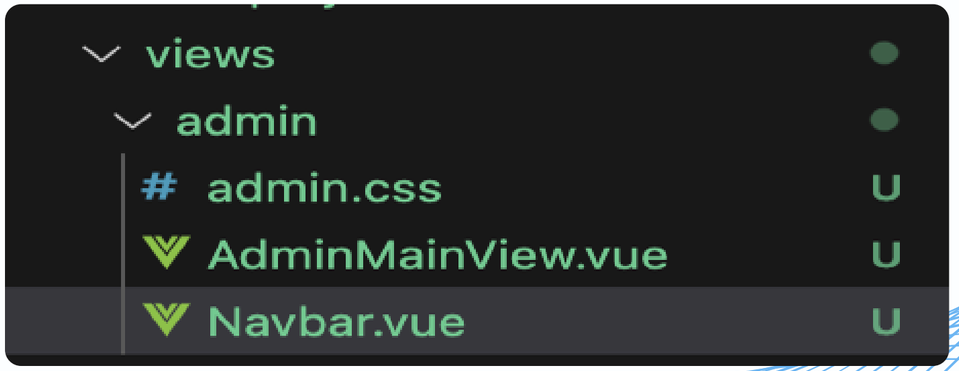
- All views (like pages) are stored in the views folder
- All models are stored in the types folder
- All endpoints(under entity name for example employee will have EmployeeAPI, etc) are stored in services folder
- All associated functionalities (list, new, etc) are stored in the components folder
    - (under entity name for example employee will contain EmployeeListComponent, EmployeeNewComponent, etc)

ADMIN SITE

- The admin site will be accessible using :
  - http://localhost:5173/admin/employees

- So we have two folders admin and client inside views - for this section we will focus on the admin site

- Step 1: Create admin.css
- Step 2: Import Navbar.vue from existing components
- Step 3: Create AdminMainView.vue



FULL STACK DEV

spring boot

HumanResourceApp backend : the Employee entity

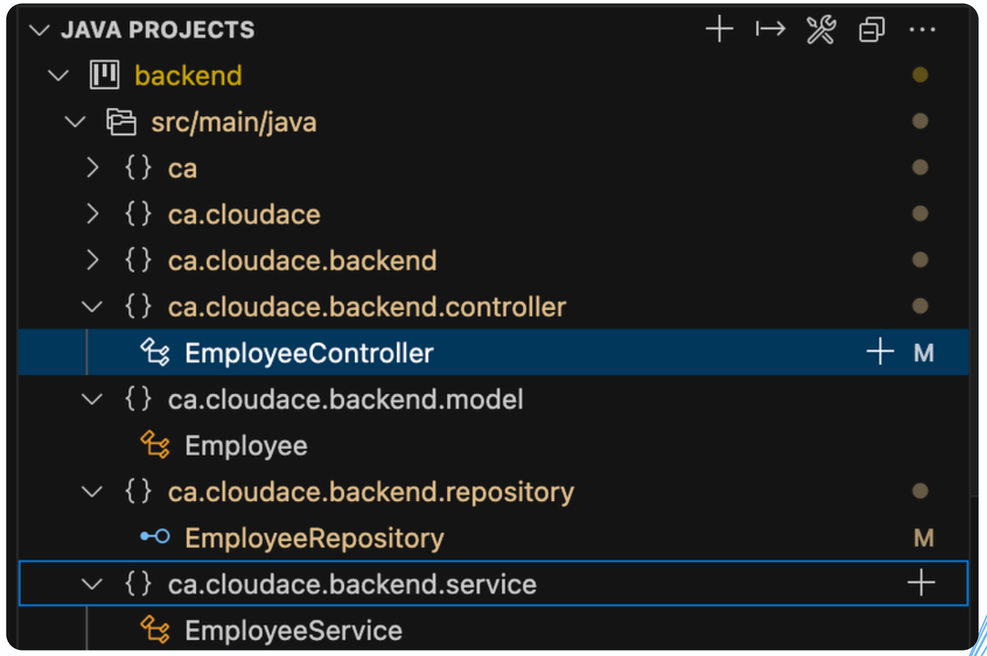
Presented by:

Rajeev Khoodeeram

OCTOBER 2025

HUMANRESOURCE APP - BACKEND

- Create the following :
  - Employee model
  - EmployeeRepository
  - EmployeeService
  - EmployeeController



- Employee endpoint testing with postman

EMPLOYEE MODEL

```java
@Entity
@Table(name = "employee")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long employeeId;

    @Column(name = "employee_firstname")
    private String employeeFirstName;

    @Column(name = "employee_lastname")
    private String employeeLastName;

    @Column(name = "employee_email")
    private String employeeEmail;

    @Column(name = "employee_phone")
    private String employeePhone;
```

Column Name	#	Data Type
123 employee_id	1	int
A-Z employee_firstname	2	varchar(255)
A-Z employee_lastname	3	varchar(255)
A-Z employee_email	4	varchar(255)
A-Z employee_phone	5	varchar(255)
employee_hiredate	6	datetime(6)
A-Z employee_title	7	varchar(255)
A-Z employee_salary	8	varchar(255)
123 department_id	9	int

EMPLOYEE REPO.

EmployeeRepository.java M

src > main > java > ca > cloudace > backend > repository > EmployeeRepository.java > ...

```java
package ca.cloudace.backend.repository;
import org.springframework.data.jpa.repository.JpaRepository;

import ca.cloudace.backend.model.Employee;

public interface EmployeeRepository
extends JpaRepository<Employee, Long> {
 // necessary to implement custom methods
}

```

EMPLOYEE SERVICE

```java
@Service
public class EmployeeService  {
    private final EmployeeRepository employeeRepository;

    public EmployeeService(EmployeeRepository employeeRepository) {
        this.employeeRepository = employeeRepository;
    }

    // Business logic methods that use employeeRepository
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    public Employee getEmployeeById(Long id) {
        return employeeRepository.findById(id).orElse(other:null);
    }


    public Employee saveEmployee(Employee employee) {
        return employeeRepository.save(employee);
    }
```

EMPLOYEE CONTROLLER

```java
@RestController
@RequestMapping("/api/employees")
@CrossOrigin(origins = "http://localhost:5173") // WE WILL SET THIS LATER
public class EmployeeController {
    private final EmployeeService employeeService;

    public EmployeeController(EmployeeService employeeService) {
        this.employeeService = employeeService;
    }

    // Define your endpoint methods here
    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeService.getAllEmployees();
    }

    @GetMapping("/{id}")
    public Employee getEmployeeById(@PathVariable Long id) {
        return employeeService.getEmployeeById(id);
    }
```

TESTING API WITH POSTMAN

HumanResource Application / **Get all employees** Save Share

GET http://localhost:8080/api/employees Send

Params Auth Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body 200 OK • 41 ms • 1.53 KB Save Response

JSON Preview Visualize

	employeeId	employeeFirstName	employeeLastName	employeeEmail	employeePh
0	4	John	Smith	jsmith@gmail.com	9053756999
1	28	xcvvd	vcvcvcxvcx	vcxvxcvc@yahoo.co	342
2	29	xcvvd	vcvcvcxvcx	vcxvxcvc@yahoo.co	342

employee

Properties Data Diagram

Show SQL Enter a SQL expression to filter results (use Ctrl+Space)

Grid Text

	e_id	employee_firstname	employee_lastname	employee_email
1		John	Smith	jsmith@gmail.com
2		xcvvd	vcvcvcxvcx	vcxvxcvc@yahoo.co
3		xcvvd	vcvcvcxvcx	vcxvxcvc@yahoo.co
4		sdfdf	sdfsdf	dsfsdfd@yahoo.com
5		John	Smith	jsmith@gmail.com

FULL STACK DEV

Employee model and Service - Vue

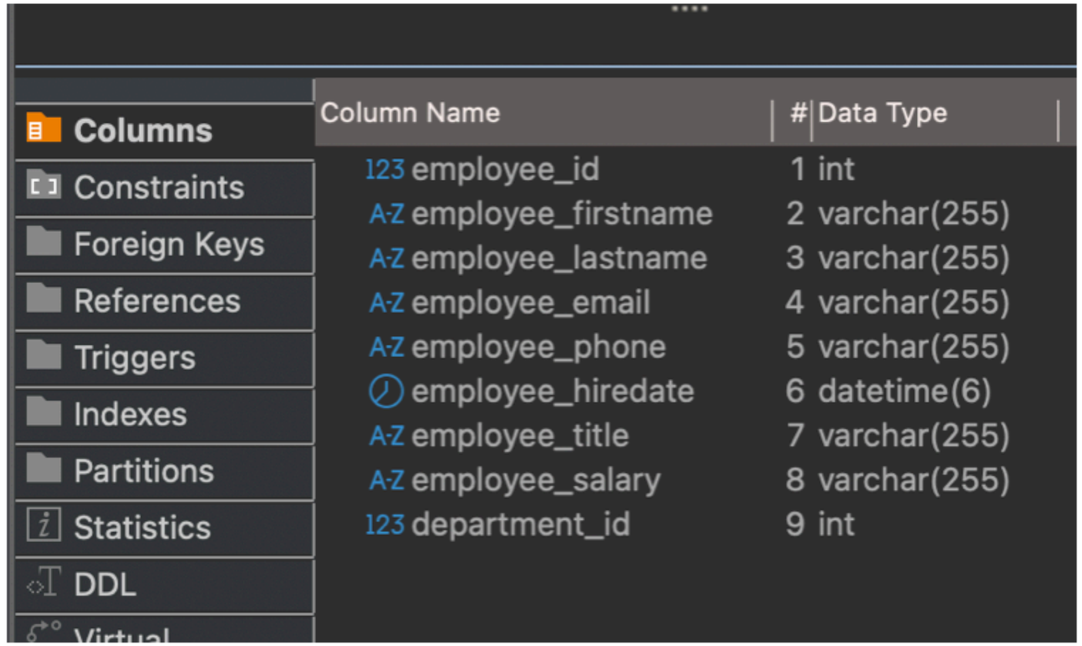
Presented by:

Rajeev Khoodeeram

OCTOBER 2025

EMPLOYEE MODEL

```typescript
export interface Employee {
  employeeId: number;
  employeeFirstName: string;
  employeeLastName: string;
  employeeEmail: string;
  employeePhone: string;
  employeeHireDate: string;
  employeeTitle: string;
  employeeSalary: number;
  departmentId: number;
}
```



EMPLOYEE SERVICES OR API

- **STEP 1**
  - Create .env file to store API_URL
  - VITE_API_URL=http://localhost:8080/api

- **STEP 2**
- Modify the EmployeeServices to get all employees from Spring backend.
- export const getAllEmployees = async () => {
- console.log("API URL:", API_URL);
- const response = await fetch(`${API_URL}/employees`);
- if (!response.ok) {
- throw new Error('Failed to fetch employees');
- }
- const employees: Employee[] = await response.json();
- console.log("Fetched employees:", employees);
- return employees;
- };

EMPLOYEELIST.VUE

```
<p v-if="employees.length === 0">No employees found.</p>
  <table v-else class="employees-table">
    <thead>
      <tr>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Email</th>
        <th>Phone</th>
        <th>Hire Date</th>
        <th>Title</th>
        <th>Salary</th>
        <th>Actions</th>
      </tr>
    </thead>
    <tbody>
      <tr v-for="employee in employees" :key="employee.employeeId">
        <td>{{ employee.employeeFirstName }}</td>
        <td>{{ employee.employeeLastName }}</td>
        <td>{{ employee.employeeEmail }}</td>
```

```
    <tbody>
      <tr v-for="employee in employees" :key="employee.employeeId">
        <td>{{ employee.employeeFirstName }}</td>
        <td>{{ employee.employeeLastName }}</td>
        <td>{{ employee.employeeEmail }}</td>
```

LOADING THE DATA FROM BACKEND

```
onMounted(async () => {
  try {
    // const response = await fetch("http://localhost:8080/api/employees");
    employees.value = await getEmployees();
    console.log("Fetched employees:", employees.value);
  } catch (error) {
    console.error("Error fetching employees:", error);
  } finally {
  }
});
```

LISTING EMPLOYEES (ADMIN)

http://localhost:5173/admin/employees

ShadcnVue | Features | Department | Employees | Leave

Create Employee

Employee List

First Name	Last Name	Email	Phone	Hire Date
John	Smith	jsmith@gmail.com	9053756999	02-04-2025
xcvvd	vcvcvcxvcx	vcxvxcvc@yahoo.co	3423434324	26-08-2025
xcvvd	vcvcvcxvcx	vcxvxcvc@yahoo.co	3423434324	23-08-2025

HTTP HumanResource Application / **Get all employees** Save Share

GET http://localhost:8080/api/employees Send

Params Auth Headers (6) Body Scripts Settings Cookies

Query Params

	Key	Value	Description	Bulk Edit
	Key	Value	Description	

Body 200 OK 41 ms 1.53 KB Save Response

{ } JSON Preview Visualize

	employeeId	employeeFirstName	employeeLastName	employeeEmail	employeePh
0	4	John	Smith	jsmith@gmail.com	9053756999
1	28	xcvvd	vcvcvcxvcx	vcxvxcvc@yahoo.co	3423434324

FULL STACK DEV

Adding an employee

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

ADDING LINK AND ROUTE

- Step 1 : Add link to the list for navigating to the form : New / Add Employee and modify the index.js for routing

- Above the list :
- Add Employee

- Modify / check the routing in index.js
-
- {
- path: "employees/new",
- component: () => import("@/components/employee/EmployeeNewComponent.vue")
- }

THE FORM - EMPLOYEENEW

```
<CardContent>
  <form
    @submit.prevent="createEmployee"
    class="grid gap-4"
  >
```

```
<div class="flex flex-col w-full gap-1.5">
  <Label for="first-name">First Name</Label>
  <Input
    id="first-name"
    type="text"
    placeholder="Leopoldo"
    v-model="contactForm.employeeFirstName"
  />
</div>

<div class="flex flex-col w-full gap-1.5">
  <Label for="last-name">Last Name</Label>
  <Input
    id="last-name"
    type="text"
    placeholder="Miranda"
    v-model="contactForm.employeeLastName"
  />
</div>
```

```
const contactForm = ref({
  employeeFirstName: "",
  employeeLastName: "",
  employeeEmail: "",
  employeePhone: "",
  employeeHireDate: "",
  employeeTitle: "",
  employeeSalary: 0,
  departmentId: 0,
});
```

```
<Button class="mt-4">{{ isEditing ? "Update Employee" : "Create Employee" }}</Button>
```

FORM PROCESSING

```
const createEmployee = async () => {
  if (contactForm.value.employeeFirstName.trim() === "" ||
      contactForm.value.employeeLastName.trim() === "" ||
      contactForm.value.employeeEmail.trim() === "" ||
      contactForm.value.employeePhone.trim() === "" ||
      contactForm.value.employeeHireDate.trim() === "" ||
      contactForm.value.employeeTitle.trim() === "" ||
      contactForm.value.employeeSalary <= 0 ||
      contactForm.value.departmentId <= 0
  ) {
```

```
try {
  const response = await fetch("http://localhost:8080/api/employees", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify(contactForm),
  });
  if (response.ok) {
    // Employee created successfully
    console.log("Employee created:", contactForm);
    invalidInputForm.value = false;
    // Redirect to the employees list page
    window.location.href = "/admin/employees";

  } else {
    console.error("Error creating employee (response):", response.statusText);
  }
} catch (error) {
  console.error("Error creating employee - catch:", error);
}
```

IMPORTANT

- Keep the same name as in Java entity class for consistency

- The **contactForm** object is your form model that Vue binds to input fields with v-model.

- Changes in the form update contactForm, and changes in contactForm update the form automatically.

KEY VUE CONCEPTS (1)

- Single File Component (**.vue**): All HTML, JS/TS, and CSS for the component are in one file.

- **<script setup>**: Variables and functions declared directly in <script setup> are automatically exposed to the template using the Composition API.

- **lang="ts"**: Specifies that the script section is written in TypeScript.

- **ref()**: A Vue Reactivity API function which takes an inner value and returns a reactive and mutable **ref** object. When you access or mutate its .value, Vue automatically tracks changes and updates the DOM.
  - *const students = ref<Student[]>([]);*
  - *students.value = data; (You must use .value in the <script setup> block, but not in the <template>.)*

KEY VUE CONCEPTS (2)

- **onMounted()**: A Vue lifecycle hook. The code inside onMounted() runs after the component has been mounted to the DOM.

- **v-if, v-else-if, v-else**: They allow you to conditionally render blocks of content based on expressions.

- **v-for**: It's used to iterate over an array (e.g., students) and render a block of elements for each item.

- **:key="student.id"**: It helps Vue identify individual nodes in the list, allowing it to efficiently update and reorder elements. It should be unique for each item.

KEY VUE CONCEPTS (3)

- @click: A shorthand for v-on:click, Vue's event listener directive. It attaches a click event handler to the button.

- {{ }} (Mustache Syntax): Used for text interpolation in the template to display reactive data.

- <style scoped>: The scoped attribute automatically adds a unique attribute to your component's HTML elements.

    - This prevents styles from "leaking" out and affecting other parts of your application.

FULL STACK DEV

Deleting an employee

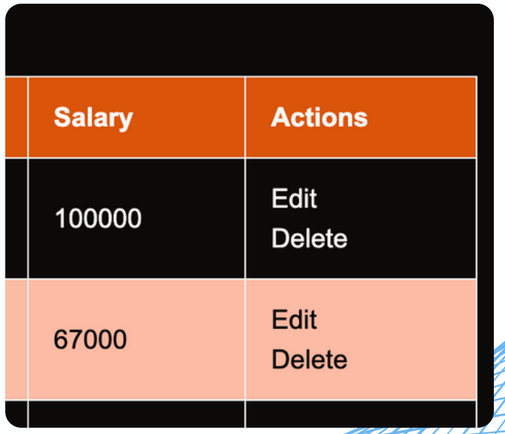
Presented by:

Rajeev Khoodeeram

OCTOBER 2025

STEP1 : ADD THE LINK

- Add link or button on the list (EmployeeListComponent.vue)
- <button @click="deleteEmployee(employee.employeeId)">Delete</button>
- Or you can do Step 2 first and Step 1 second...as you wish



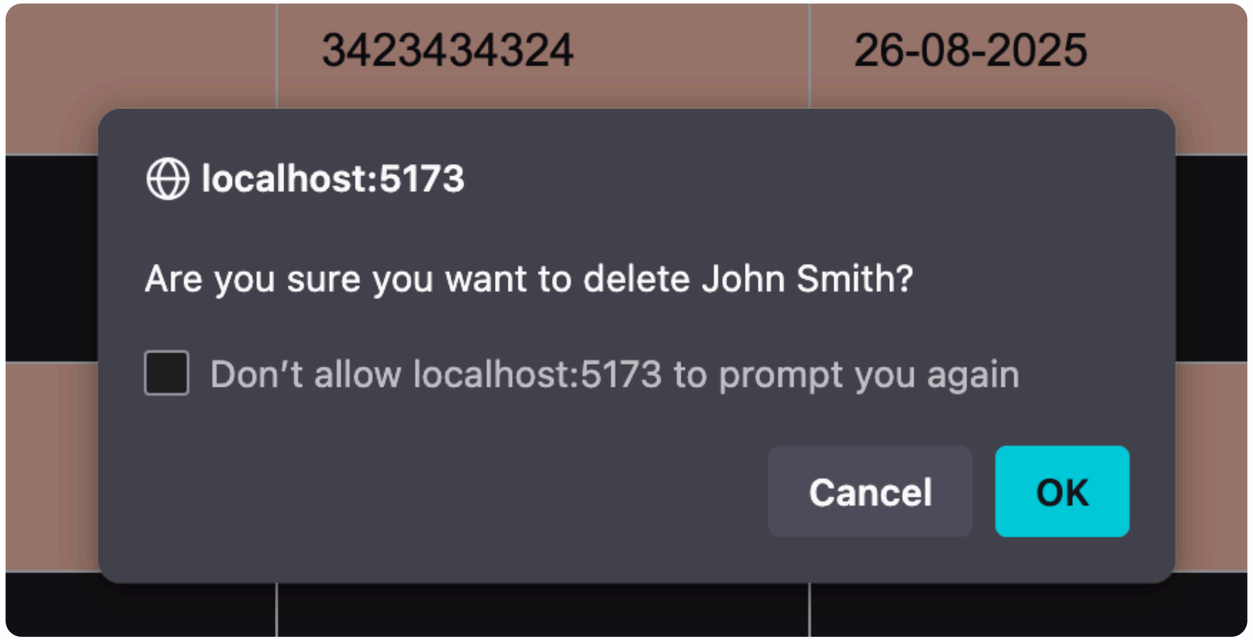
IMPLEMENT THE DELETEEMPLOYEE FUNCTION

```
/**
 * Delete an employee by ID
 * @param id The ID of the employee to delete
 * @returns The ID of the deleted employee
 */
export const deleteEmployee = async (id: number) => {

  if (confirm("Are you sure you want to delete this employee?"))
{

  console.log("Deleting employee with ID:", id);

  const response = await fetch(`${import.meta.env.VITE_API_URL}/
employees/${id}`, {
    method: 'DELETE',
  });
  if (!response.ok) {
    throw new Error('Failed to delete employee');
  }
  return id;
}
};
```

LISTING DOCTORS

- We have added a confirmation popup; if yes then delete is executed.

- Please take note that if the user cancel the confirm message the deleteEmployee is not executed.



SEQUENCE OF ACTIONS

```
fetch(`http://localhost:8080/api/employees/${employee.employeeId}`, {
  method: "DELETE",
})
```

```
@RestController
@RequestMapping("/api/employees")
@CrossOrigin(origins = "http://localhost:5173")
public class EmployeeController {
```

EmployeeController.java M

src > main > java > ca > cloudace > backend > controller > EmployeeController.java > ...

```
19  @RestController
45      @PutMapping("/{id}")
48      }
49
50      @DeleteMapping("/{id}")
51      public void deleteEmployee(@PathVariable Long id) {
52          employeeService.deleteEmployee(id);
53      }
54
```

EmployeeService.java

src > main > java > ca > cloudace > backend > service > EmployeeService.java > Languag

```
9   @Service
30
31
32      public void deleteEmployee(Long id) {
33          employeeRepository.deleteById(id);
34      }
```

FULL STACK DEV

Updating an employee

Presented by:

Rajeev Khoodeeram

OCTOBER 2025

ADD LINK TO UPDATE AN EMPLOYEE

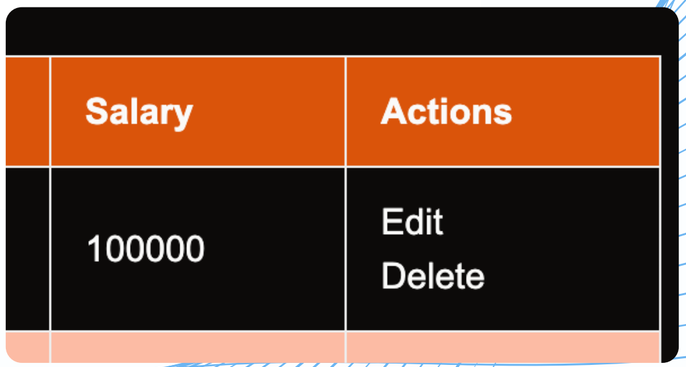
This is going to be our last functionality for the employee table. We will perform the following steps :

Step 1 : Add edit link as Actions which will navigate to the edit form

```
<button @click="router.push({ path: `/admin/employees/edit/${employee.employeeId}` })">Edit</button>
```

We must import useRouter to allow navigation to work

```
import { useRouter } from "vue-router";

const router = useRouter();
```



UPDATE INDEX.JS

Step 2 : Update the index.js for routing

```
{
  path: "employees/edit/:id",
  component: () => import("@/components/employee/
EmployeeNewComponent.vue"),
  props: true
}
```

NOTICE, IT POINTS TO THE SAME FORM FOR ADDING AN EMPLOYEE !

ADD THE UPDATEEMPLOYEE FUNCTIONALITY

```
/**
 * Update an employee by ID
 * @param id The ID of the employee to update
 * @param employeeData The updated employee data
 * @returns The updated employee
 */
export const updateEmployee = async (id: number,
employeeData: Partial<Employee>) => {
  const response = await fetch(`$
{import.meta.env.VITE_API_URL}/employees/${id}`, {
    method: 'PUT',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(employeeData),
  });
  if (!response.ok) {
    throw new Error('Failed to update employee');
  }
  const employee: Employee = await response.json();
  return employee;
};
```

CALLING THE BACKEND

Notice here : the function takes the employee id and the full employee object; it uses the **PUT** method which corresponds to the **PUT** method in the EmployeeComtroller in the Java Spring Boot backend !!

```
 @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id,
@RequestBody Employee employee) {
        return employeeService.updateEmployee(id, employee);
    }
```

MODIFY THE EMPLOYEENEW.VUE

```
const isEditing = ref(false);
const id  = route.params.id;


if (isEditing.value) {
     // Here you would typically send the updated form data to
your server
  // For demonstration, we'll just log it to the console
    console.log("Updating employee with ID:", id);
    updateEmployee(Number(id), {
        employeeFirstName,
        employeeLastName,
        employeeEmail,
        employeePhone,
        employeeHireDate,
        employeeTitle,
        employeeSalary: Number(employeeSalary),
        departmentId: Number(departmentId),
    })
        .then((employee) => {
        console.log("Updated Employee:", employee);
        })
        .catch((error) => {
        console.error("Error updating employee:", error);
        });
  console.log("Employee Updated:", contactForm);
```

```
  // Reset the form after submission
  contactForm.employeeFirstName = "";
  contactForm.employeeLastName = "";
  contactForm.employeeEmail = "";
  contactForm.employeePhone = "";
  contactForm.employeeHireDate = "";
  contactForm.employeeTitle = "";
  contactForm.employeeSalary = 0;
  contactForm.departmentId = 0;

  invalidInputForm.value = false;
  }
```