



Project Pioneers

Multi-Tenant ERP website

Project by :

Vishnu Ramu Chityala (E23CSEU0049)

Vansh Singhal (E23CSEU0046)



SUBMITTED TO

SCHOOL OF COMPUTER SCIENCE ENGINEERING AND TECHNOLOGY, BENNETT
UNIVERSITY

GREATER NOIDA, 201310, UTTAR PRADESH, INDIA

April 2023

DECLARATION

I/We hereby declare that the work which is being presented in the report entitled “Project Pioneers” is an authentic record of my/our own work carried out during the period from March 2023 to April 2023 at School of Computer Science and Engineering and Technology, Bennett University Greater Noida.

The matters and the results presented in this report has not been submitted by me/us for the award of any other degree elsewhere.

Signature of Candidate

Vishnu Chityala

(Enroll. No. E23CSE0049)

Vansh Singhal

(Enroll. No. E23CSE0046)

Index

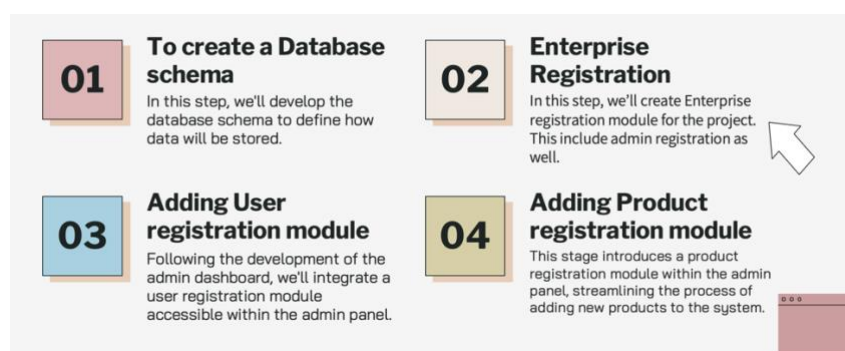
- Introduction
- Project Phases
- Conclusion
- Sources

Introduction

This report documents the development of a multi-tenant Enterprise Resource Planning (ERP) web application built with Spring Boot. Driven by a personal interest in Spring Boot's framework for building intricate applications, this project delves into the creation of a centralized platform. This platform caters to the needs of multiple businesses, allowing them to manage their inventories and resources collaboratively on a single system.

ERP software is primarily created for a particular kind of enterprise, depending on the product and product type for which it is intended. By visualising a new kind of ERP web application, in which the enterprise administrator determines the product and product type and the website operates according to the specified configuration, by this the enterprise can easily manage its various product types and do so quickly.

Project has been distributed in four phases:



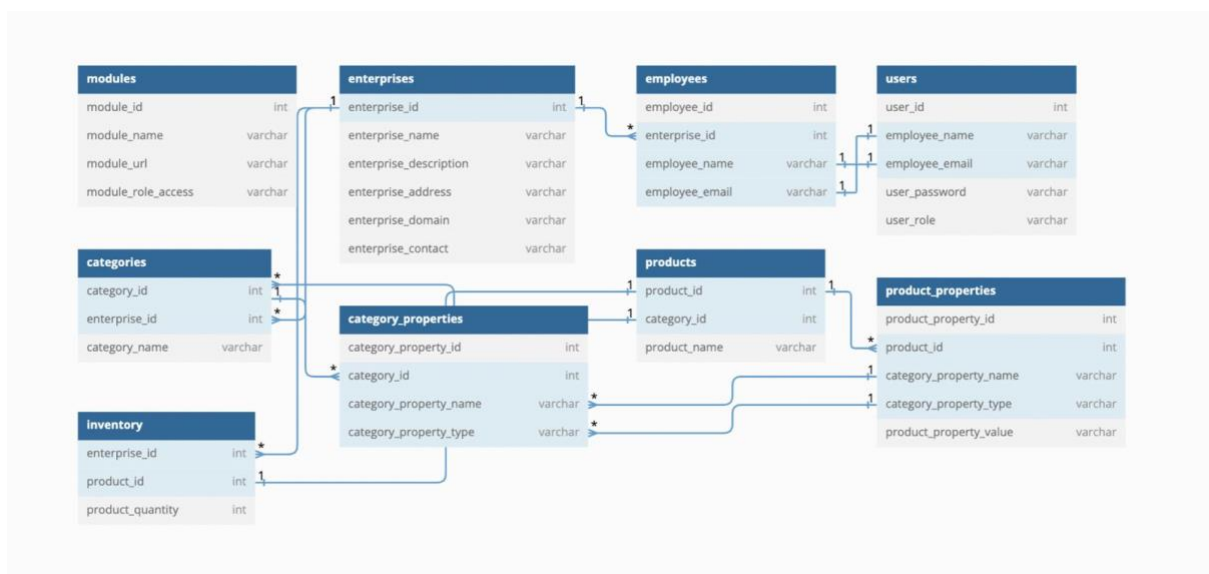
(Image used in previous presentation)

Detailed report of all phases is there in upcoming section of project report.

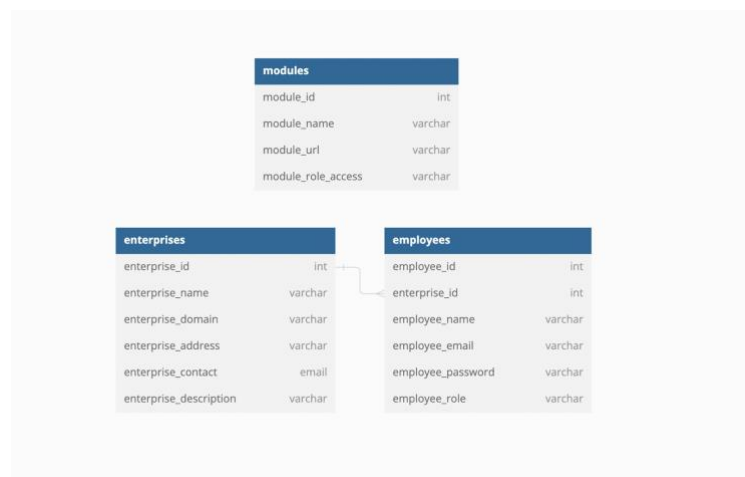
Phase One (Database schema)

The very first phase involved creating a database schema so that data from several enterprises could be stored in a single database. A database ought to be built with the ability to handle products with different properties and types of property values.

Database schema which was designed:



Because of time constraint project implements the following database schema:



Data regarding enterprises which have registered is kept by the project in a table named "Enterprises." Every company has a distinct name and, more significantly, a distinct domain name. Email addresses for users within that enterprise are planned to be created using this domain name. If an enterprise uses the domain "sivpl.com," for instance, user emails may appear as "<username>@sivpl.com". In this manner, email addresses are distinct and make it obvious to which enterprise a user is affiliated with.

Employee data is kept by the project in a table named "Employees." All employees who work for different enterprises that use the ERP system are listed in this table. Every employee has an ID, a name, a password that is securely stored, an email address that is likely created using the domain of their enterprise, and a role that they play in the company (e.g., ADMIN, EMPLOYEE). Each employee is linked to their respective enterprise in a separate table by the "enterprise_id" column. In this manner, the system can effectively manage workers from various enterprises.

The URLs for the modules in the ERP web application are kept in the modules table, which lists the modules according to the user role. The table stores the module along with its name and URL.

We must create models or JPA entities in our project, which will manage all of the project's data, in order to integrate database tables with our Spring Boot project.

```

@Entity
@Table(name = "employees", uniqueConstraints = @UniqueConstraint(columnNames = "employee_email"))
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "employee_id")
    private Long id;
    @ManyToOne
    @JoinColumn(name = "enterprise_id")
    private Enterprise enterprise;
    @Column(name = "employee_name")
    private String name;
    @Column(name = "employee_email")
    private String email;
    @Column(name = "employee_password")
    private String password;
    @Column(name = "employee_role" )
    private String role;
}

```

(Employee Entity)

```

@Entity
@Table(name = "enterprises", uniqueConstraints = @UniqueConstraint(columnNames = "domain"))
public class Enterprise {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "enterprise_id")
    private Long id;
    @Column(name = "enterprise_name")
    private String name;
    @Column(name = "enterprise_description")
    private String description;
    @Column(name = "enterprise_address")
    private String address;
    @Column(name = "enterprise_domain")
    private String domain;
    @Column(name = "enterprise_contact")
    private String contact;
}

```

(Enterprise Entity)

```

@Entity
@Table(name = "modules")
public class AppModule {

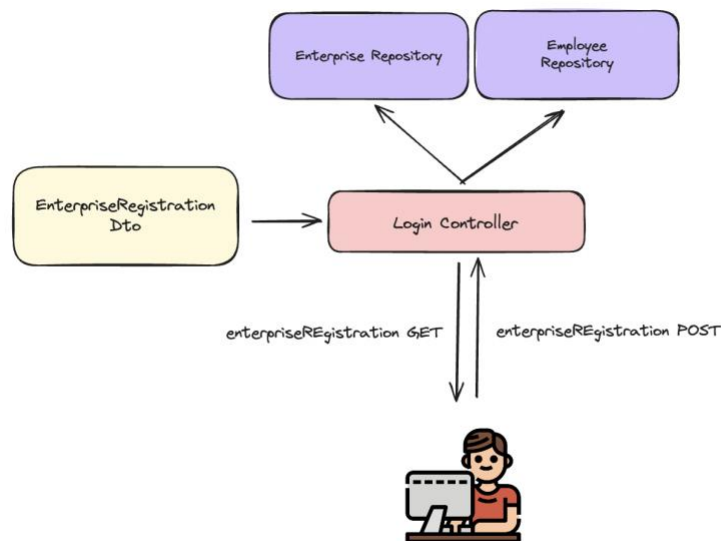
    @Id
    private Long id;
    @Column(name = "module_name")
    private String module_name;
    @Column(name = "module_url")
    private String module_url;
    @Column(name = "role_access")
    private String role_access;
}

```

(AppModule Entity)

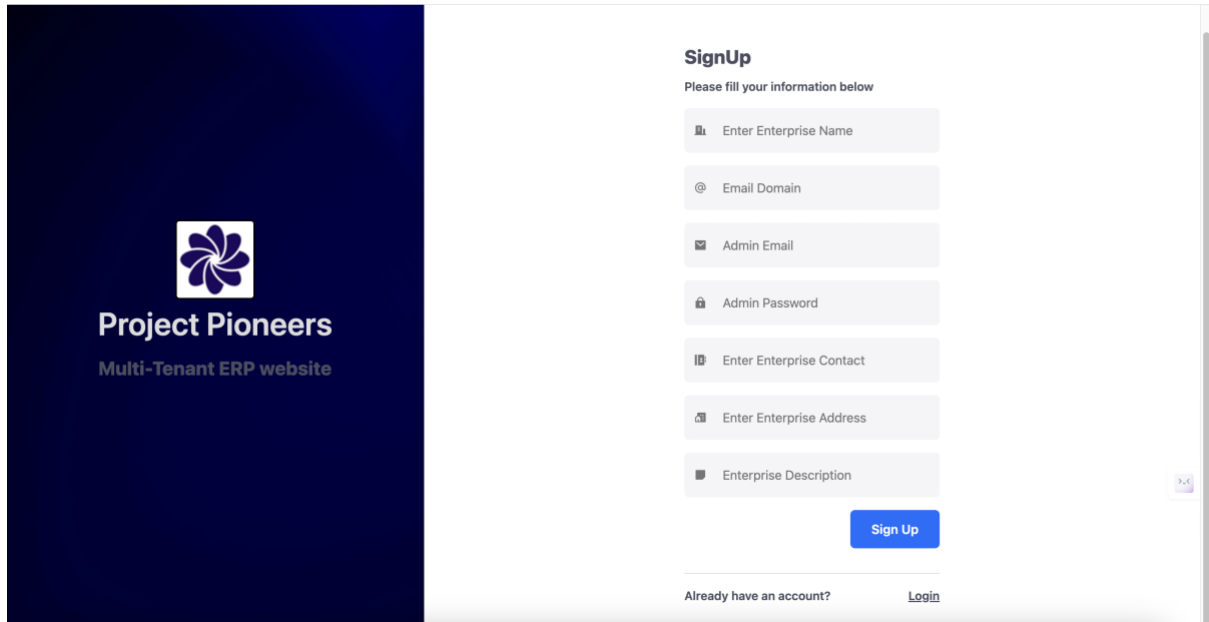
Phase Two (Enterprise Registration)

This phase was dedicated to developing the "Enterprise Registration Module." During signing up, users provide essential enterprise information. When the registration is successful, it immediately grants the new enterprise system access for resource and personnel management, makes a new entry in the "Enterprises" table, and automatically creates a default admin employee account.



Users initially see a GET request (registration form). The system saves the data and adds a new entry for their enterprise in the database when they submit the form (POST request). Additionally, a default admin user account is created for them automatically. This enables them to manage employees and resources by gaining instant access to the system.

Important: To keep everything safe, security checks should be turned back on after being momentarily disabled during registration.



(registration page preview)

```
@RequestMapping(method = RequestMethod.GET, value = "/registration")
public ModelAndView registration()
{
    EnterpriseRegistrationDto enterpriseRegistrationDto = new EnterpriseRegistrationDto();
    ModelAndView modelAndView = new ModelAndView(viewName:"enterpriseRegistration");
    modelAndView.addObject(attributeName:"registrationDto", enterpriseRegistrationDto);
    return modelAndView;
}

@RequestMapping(method = RequestMethod.POST, value = "/registration")
public String registrationEnd(@ModelAttribute("registrationDto") EnterpriseRegistrationDto enterpriseRegistrationDto)
{
    EnterpriseDto enterpriseDto = new EnterpriseDto(
        enterpriseRegistrationDto.getName(),
        enterpriseRegistrationDto.getDescription(),
        enterpriseRegistrationDto.getAddress(),
        enterpriseRegistrationDto.getDomain(),enterpriseRegistrationDto.getContact()
    );

    enterpriseService.createEnterprise(enterpriseDto);

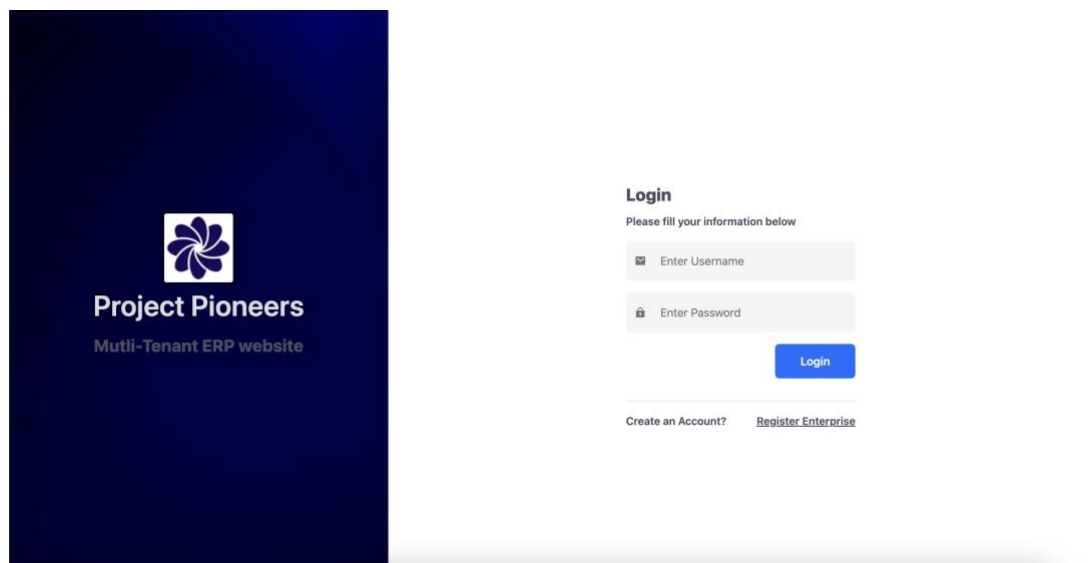
    Enterprise enterprise = enterpriseService.getEnterprise(enterpriseRegistrationDto.getDomain());
    EmployeeDto employeeDto = new EmployeeDto(
        name:"admin",
        enterpriseRegistrationDto.getEmail(),
        enterpriseRegistrationDto.getPassword(),
        role:"ADMIN"
    );

    employeeService.createEmployee(employeeDto, enterprise);

    return "redirect:/login";
}
```

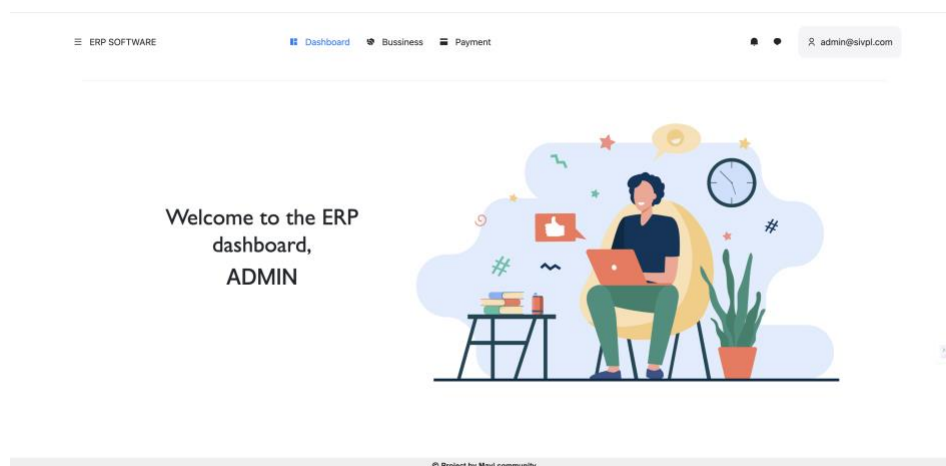
(registration URLs mapping)

Phase Three (Employee Registration)



(login page preview)

After successfully logging in, users are taken to a dashboard that is specific to them. Modules are dynamically displayed on this dashboard according to the user's role. This makes sure that the user experience is optimised by showing them only the features that are necessary for their job.



Employees Registration:

The system uses POST and GET requests to manage current employees and create new ones. An "employeeRegistrationDTO" object is added to the ModelAndView object during a GET request. The admin-submitted form data is temporarily housed in this DTO. The system uses the data in the DTO to create a new employee entry in the database upon form submission (POST request), creating a connection between the newly created employee and the admin's enterprise.

Employee Editing:

Editing current employee information also uses a combination of GET and POST requests, just like registering a new employee. An exclusive URL mapping, "/admin/editEmployee/{id}", records the employee's distinct ID that has to be modified. In response to a GET request made with a particular ID, the system populates a "employeeDTO" object with the relevant data. The employee registration form is pre-populated with the current data by this object, making it editable. The system uses the updated information in the DTO to update the current employee record in the database after receiving the amended form (POST request).

Employee Deletion:

For the purpose of eliminating employee records, a different URL mapping, "/admin/deleteEmployee/{id}", is defined. Upon utilising a unique employee ID (POST request), the system finds the relevant record instantly and deletes it, thereby eliminating the user from the system.

With the use of distinct employee ID referencing, this method preserves data security, guarantees uniformity, and streamlines the employee management experience.

```
@RequestMapping(method = RequestMethod.GET, value = "/admin/employeeRegistration")
public ModelAndView userRegistration(@AuthenticationPrincipal UserDetails userDetails, EmployeeDto employeeDto, EditEmployeeDto editEmployeeDto)
{
    ModelAndView modelAndView = new ModelAndView(viewName:"employeeRegistration");
    String userName = userDetails.getUsername();
    modelAndView.addObject(attributeName:"userEmail", userName);
    modelAndView.addObject(attributeName:"employees", employeeService.getAllEmployeesByDomain(employeeService.getDomainByEmail(userDetails.getUsername())));
    modelAndView.addObject(attributeName:"newUser", employeeDto);
    modelAndView.addObject(attributeName:"modules", moduleService.getAllModules());
    modelAndView.addObject(attributeName:"userRole", userDetails.getAuthorities().iterator().next().getAuthority());
    modelAndView.addObject(attributeName:"editEmployee", editEmployeeDto);
    return modelAndView;
}

@RequestMapping(method = RequestMethod.POST, value = "/admin/employeeRegistration")
public String registerEmployee(@AuthenticationPrincipal UserDetails userDetails, @ModelAttribute EmployeeDto employeeDto)
{
    Enterprise enterprise = enterpriseService.getEnterprise(
        employeeService.getDomainByEmail(userDetails.getUsername())
    );
    employeeService.createEmployee(employeeDto, enterprise);
    return "redirect:/admin/employeeRegistration";
}

@RequestMapping(method = RequestMethod.POST, value = "/admin/editEmployee/{id}")
public String editEmployee(@AuthenticationPrincipal UserDetails userDetails, @ModelAttribute("editEmployee") EditEmployeeDto employeeDto, @PathVariable String id)
{
    Employee edEmployee = employeeService.getEmployeeById(Long.valueOf(id));
    edEmployee.setEmail(employeeDto.getEmail() == "" ? edEmployee.getEmail() : employeeDto.getEmail());
    edEmployee.setName(employeeDto.getName() == "" ? edEmployee.getName() : employeeDto.getName());
    edEmployee.setPassword(employeeDto.getPassword() == "" ? edEmployee.getPassword() : employeeDto.getPassword());
    employeeService.editEmployee(edEmployee);
    return "redirect:/admin/employeeRegistration";
}

@RequestMapping(method = RequestMethod.POST, value = "/admin/deleteEmployee/{id}")
public String deleteEmployee(@PathVariable String id)
{
    employeeService.deleteEmployeeById(Long.valueOf(id));
    return "redirect:/admin/employeeRegistration";
}
```

(Code snippet for URL mappings)

ERP SOFTWARE
Dashboard
Business
Payment
admin@shivpi.com

Employee Registration

Employee Name
Email Address
Password
Employee Role

Add Employee

Registered Users

ID	Username	Email Id	Password	Actions
2	Vishnu Chitkala	vishnu@shivpi.com	vishnu1234	Delete Edit
3	Aakash Chitkala	aakash@shivpi.com	aakash123	Delete Edit
4	Harshi Uttaradhi	harshi@shivpi.com	harshi123	Delete Edit
5	Arihant Gupta	arihant@shivpi.com	arihant123	Delete Edit
12	Kaveri Chitkala	kaveri@shivpi.com	kaveri123	Delete Edit
13	Shobhit Sharma	shobhit@shivpi.com	shobhit123	Delete Edit
19	John Doe	john@shivpi.com	john123	Delete Edit
20	Paras Chaudhary	paras@shivpi.com	paras123	Delete Edit
21	Lakshay Garg	lakshay@shivpi.com	lakshay123	Delete Edit

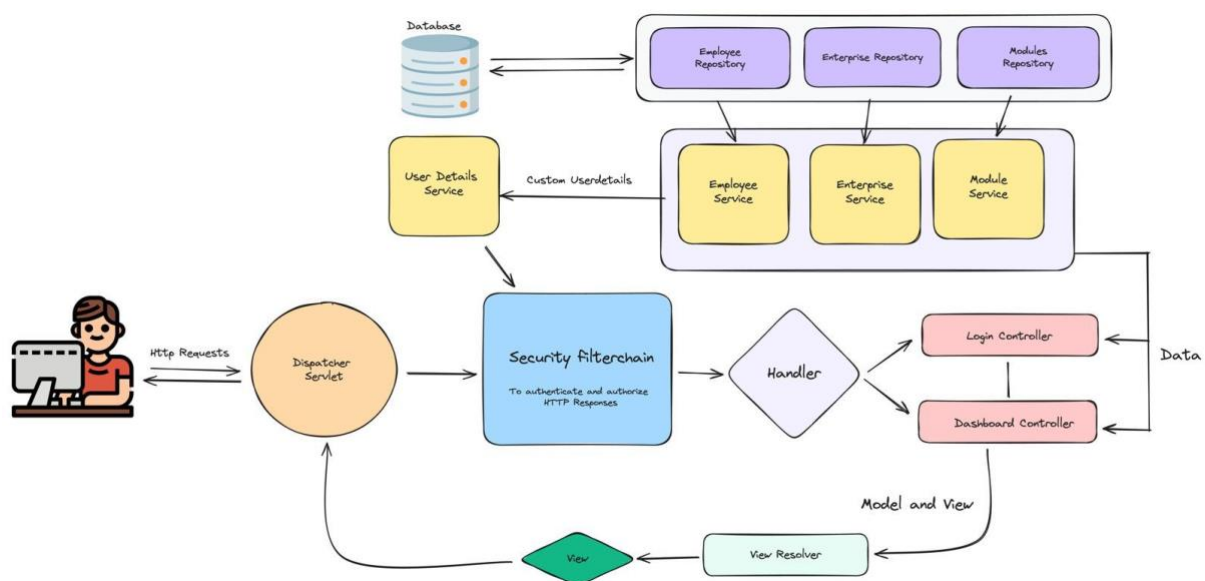
(Employee registration module)

Conclusion

Due to difficulties in using JavaScript and other web development technologies to create an interactive frontend, the project's implementation of category and product registration came to a standstill.

In conclusion, Spring Boot, Spring JPA, Spring Web MVC, Spring Security, Thymeleaf, HTML, CSS, and MySQL were used to create a functional web application.

Up until now, the web application has handled employee data from various enterprises in a seamless manner and seeks to manage all of an enterprise's resources.



(Basic layout of spring boot application functioning)

Sources

- <https://docs.spring.io/spring-security/reference/features/index.html>
- <https://docs.spring.io/spring-framework/reference/web/webmvc.html>
- <https://www.thymeleaf.org/doc/tutorials/3.1/extendingthymeleaf.html>
- <https://getbootstrap.com/docs/5.3/getting-started/introduction/>
- Articles and blogs on Internet.