

## Company Reimbursement Management System

---

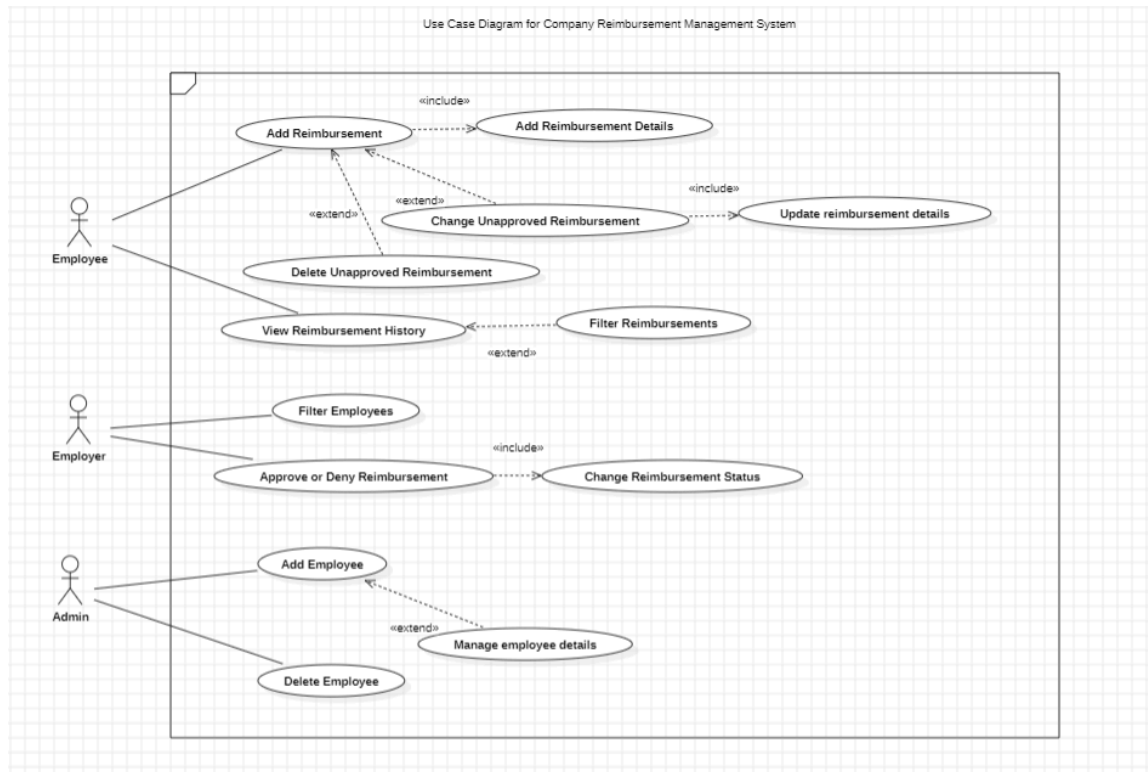
### Problem Statement

The project is a company reimbursement management system designed for an organization to streamline the process of handling reimbursement requests from employees. It encompasses various roles such as employees, employers, and administrators. Employees initiate reimbursement requests by submitting details such as category, amount requested, receipt ID, document date, and vendor name. They can also keep track of their previous reimbursements. These requests are then processed by employers, who have the authority to approve or deny them. Employers can also view pending, approved, or denied reimbursement requests, along with their respective details like request ID, date submitted, category, amount requested, receipt ID, document date, and vendor name. Employers can filter through reimbursements based on category, employee id as well as status. Administrators, on the other hand, seem to have broader access, including the ability to manage employee accounts and add new employees across the organization. Additionally, there's functionality for calculating refunds for approved reimbursements, involving intricate calculations based on category limits and percentages. Overall, the system aims to automate and streamline the reimbursement process, providing transparency and efficiency for both employees and administrators.

## Company Reimbursement Management System

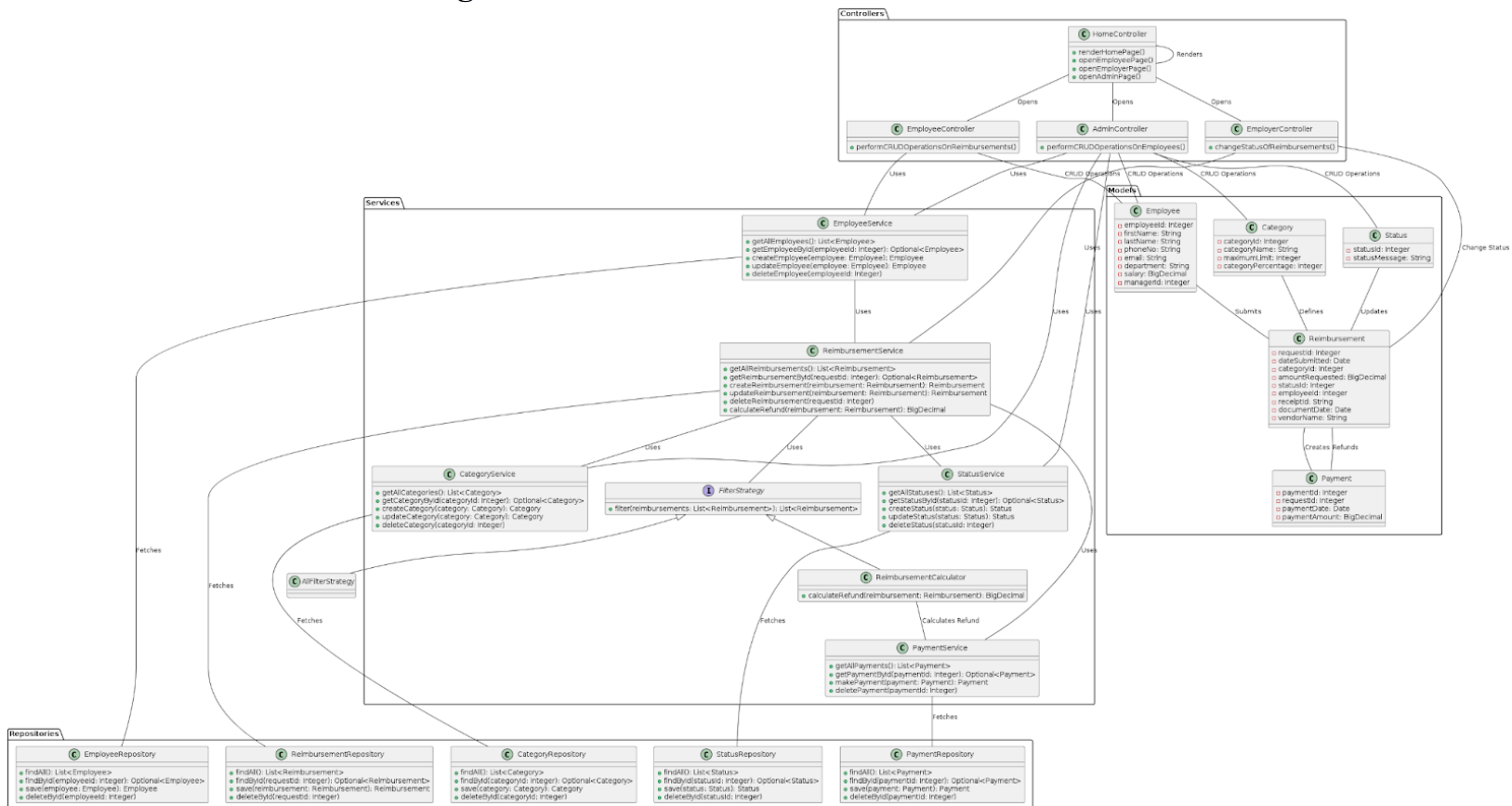
### Models

- Use Case diagram



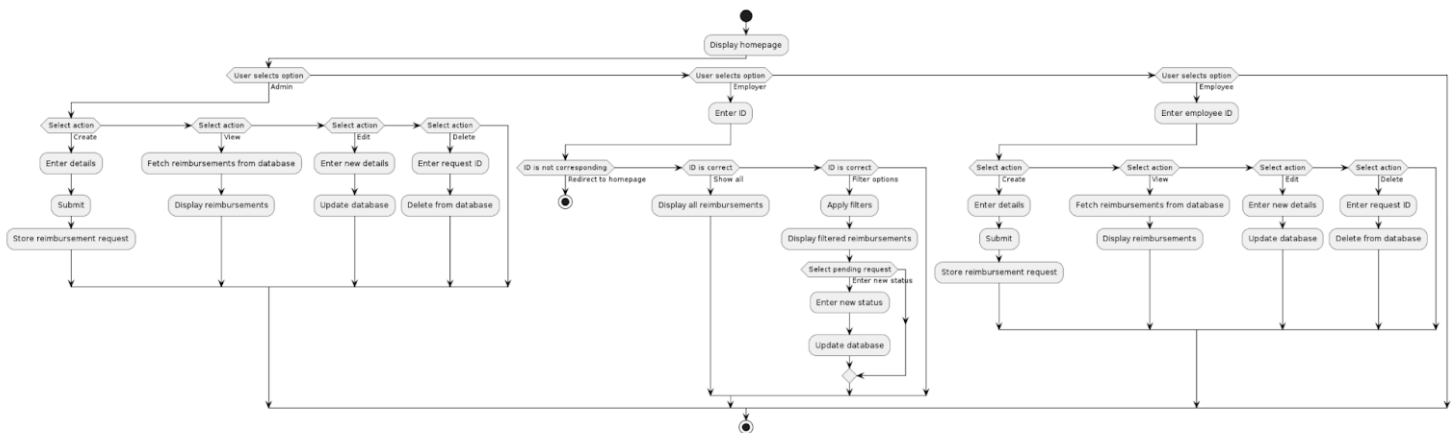
## Company Reimbursement Management System

## ● Class Diagram

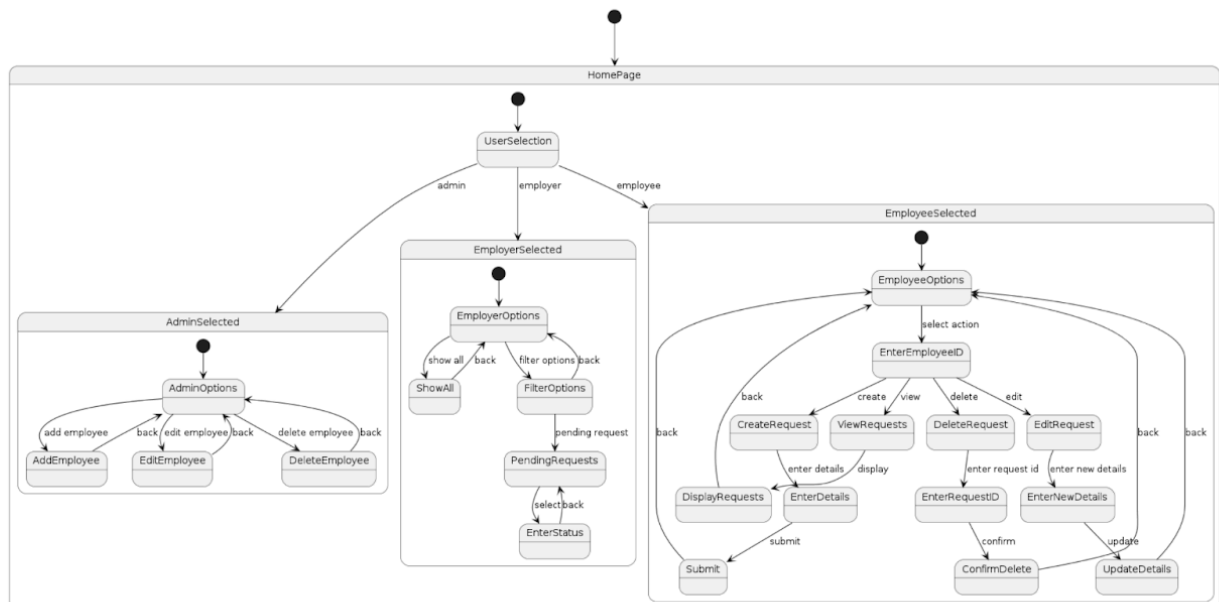


## Company Reimbursement Management System

### ● Activity Diagram



### ● State Diagram



## Architecture Patterns:

## Company Reimbursement Management System

---

We have used the Model-View-Controller (MVC) architecture as it follows separation of principle, with clear distinctions between different layers and responsibilities, promoting modularity, maintainability, and testability of the application.

In the MVC (Model-View-Controller) architectural pattern, the application is divided into three interconnected components:

1. Model: The model represents the application's data and business logic. It encapsulates the state of the application and provides methods to manipulate that state. In our project, the model layer is represented by the models package, which includes classes such as Category, Employee, Payment, Reimbursement, and Status. These classes define the structure and behavior of various entities within the application, such as employees, reimbursements, categories, etc. They encapsulate the data and behavior associated with these entities.
2. View: The view represents the presentation layer of the application. It is responsible for rendering the user interface and displaying data to the user. In our project, the view layer consists of HTML templates located in the templates directory. Each HTML template corresponds to a specific page or view in the application and is responsible for rendering the UI elements and content associated with that view. For example, admin.html, employee.html, employer.html, etc., represent different views in the application.
3. Controller: The controller acts as an intermediary between the model and the view. It receives user input, processes it, interacts with the model to perform business logic, and updates the view accordingly. In our project, the controller layer is represented by classes such as AdminController, EmployeeController, EmployerController, HomeController, etc., located in the controllers package. Each controller is responsible for handling requests from the client, invoking the appropriate services to perform business logic, and returning an appropriate response to the client. For example, EmployerController handles requests related to employer actions, such as viewing reimbursements, changing reimbursement status, etc.

## Design Principles:

1. Single Responsibility Principle (SRP): Each class has a single responsibility. For example, EmployerController handles HTTP requests related to employer actions, FilterService provides filtering functionality for reimbursements, and AllFilterService handles fetching all reimbursements. Each class is responsible for a specific aspect of the application. Employee controller handles HTTP requests related to employee actions and even for Admin Controller.
2. Interface Segregation Principle (ISP): Both FilterService and AllFilterService implement the Filter interface. This ensures that each service only provides the

## Company Reimbursement Management System

---

necessary methods required for filtering reimbursements, promoting a more granular and focused interface.

3. Separation of Concerns (SoC): The code separates concerns between the controller, services, and repository layers. The controller handles HTTP requests and responses, services handle business logic, and repositories handle data access. This separation makes the codebase more modular and easier to understand, maintain, and test.
4. Dependency Injection (DI): Dependency injection (DI) is used in the employee part of the code primarily in the controllers and services. In the controllers, dependencies such as service classes are injected, enabling the controllers to interact with the business logic encapsulated in the services. For example, in the `EmployeeController`, dependencies like `CategoryService` and `EmployeeService` are injected via constructor or setter injection. Similarly, in the services layer, dependencies like repositories (e.g., `CategoryRepository`, `EmployeeRepository`) are injected, allowing services to access and manipulate data from the database without tightly coupling them to specific data access mechanisms. Similarly it is utilized in `Admin` and `Employer` controllers.

## Design Patterns :

### Singleton Design Pattern:

The Singleton method or Singleton Design pattern is one of the simplest design patterns. It ensures a class only has one instance, and provides a global point of access to it.

The singleton design pattern ensures that there's only one instance of the reimbursement calculator throughout the application's lifecycle. This guarantees centralized control over refund calculations, preventing multiple instances from conflicting or duplicating computations. By maintaining a single, shared instance, we enhance efficiency, resource management, and consistency in handling refund calculations across various parts of the system, ultimately promoting scalability and maintainability.

Code:

```
@Service
public class ReimbursementCalculator {

    // Singleton instance
    private static ReimbursementCalculator instance;

    // Private constructor to prevent instantiation
    private ReimbursementCalculator() {}
```

## Company Reimbursement Management System

---

```
// Method to get the singleton instance
public static synchronized ReimbursementCalculator getInstance() {
    if (instance == null) {
        instance = new ReimbursementCalculator();
    }
    return instance;
}

// Method to perform reimbursement calculation
public double calculateReimbursement(double amountRequested, double categoryLimit,
double percentage) {
    // Apply reimbursement calculation logic here
    // For example:
    double reimbursementAmount = amountRequested * percentage / 100.0;
    if (reimbursementAmount > categoryLimit) {
        reimbursementAmount = categoryLimit; // Apply category limit
    }
    return reimbursementAmount;
}
}
```

This single instance is utilized as a centralized location to get the reimbursement calculations to be done.

### Observer Design Pattern:

The Observer Pattern is a behavioral design pattern that defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The main idea is to have one object (subject) that manages a list of other objects (observers) that are interested in its state changes. When the subject's state changes, it notifies all the observers, and they can update themselves.

Key Components:

Subject (Publisher):

- Maintains a list of observers.

- Provides methods to register, remove, and notify observers.

Observer (Subscriber):

- Defines an updating interface for objects that should be notified of changes in a subject.

Concrete Subject:

## Company Reimbursement Management System

---

Implements the Subject interface.

Sends notification to observers about state changes.

Concrete Observer:

Implements the Observer interface.

Stores a reference to the Concrete Subject.

Implements the update method, which is called by the Subject whenever it notifies its observers.

### How it is used in the code:

In the above example, when a new reimbursement request is submitted, all the observers (Employers and Administrators) are notified and updated about this change.

### Code:

```
from abc import ABC, abstractmethod
```

```
class Subject(ABC):
```

```
    @abstractmethod
```

```
    def attach(self, observer):
```

```
        pass
```

```
    @abstractmethod
```

```
    def detach(self, observer):
```

```
        pass
```

```
    @abstractmethod
```

```
    def notify(self, data):
```

```
        pass
```

```
class Observer(ABC):
```

```
    @abstractmethod
```

```
    def update(self, data):
```

```
        pass
```

```
class ReimbursementRequest(Subject):
```

```
    def __init__(self):
```

```
        self._observers = []
```

```
    def attach(self, observer):
```

```
        self._observers.append(observer)
```

```
    def detach(self, observer):
```

```
        self._observers.remove(observer)
```



### Company Reimbursement Management System

---

```
def notify(self, data):
    for observer in self._observers:
        observer.update(data)

class Employer(Observer):
    def __init__(self, name):
        self._name = name

    def update(self, data):
        print(f'Employer {self._name} has been notified: {data}')

class Administrator(Observer):
    def __init__(self, name):
        self._name = name

    def update(self, data):
        print(f'Administrator {self._name} has been notified: {data}')

# Example of usage:
if __name__ == "__main__":
    # Creating the reimbursement request
    request = ReimbursementRequest()

    # Creating employers and administrators
    employer1 = Employer("John")
    employer2 = Employer("Alice")

    administrator1 = Administrator("Robert")
    administrator2 = Administrator("Diana")

    # Attaching employers and administrators to the reimbursement request
    request.attach(employer1)
    request.attach(employer2)
    request.attach(administrator1)
    request.attach(administrator2)

    # Notifying the observers about the state change
    request.notify("New reimbursement request submitted")
```

## Company Reimbursement Management System

---

### Strategy Design Pattern:

The Strategy Pattern is a behavioral design pattern that enables an object, called the context, to vary its behavior at runtime by encapsulating different algorithms or strategies and making them interchangeable. This pattern allows the client to choose from a family of algorithms dynamically without altering the client's code.

We have used the Strategy Pattern to implement different filtering strategies for viewing reimbursement requests. The FilterStrategy interface represents the strategy, defining the filter() method. Concrete implementations of this interface

(CategoryEmployeeStatusFilterStrategy and AllFilterStrategy) encapsulate specific filtering algorithms:

- CategoryEmployeeStatusFilterStrategy: This strategy filters reimbursement requests based on specific category, employee, and status.
- AllFilterStrategy: This strategy filters all reimbursement requests without any specific criteria, essentially retrieving all requests.

#### Code:

Filter Strategy Interface:

```
public interface Filter{
```

```
    Iterable<Reimbursement> fetchReimbursements(Integer employeeId, Integer category,Integer employerId);
}
```

All Filter Implementation:

```
public class AllFilterService implements Filter {
```

```
    private final ReimbursementRepository reimbursementRepository;
```

```
    @Autowired
```

```
    public AllFilterService(ReimbursementRepository reimbursementRepository) {
        this.reimbursementRepository = reimbursementRepository;
    }
```

```
    @Override
```

```
    public Iterable<Reimbursement> fetchReimbursements(Integer employeeId, Integer category,Integer employerId) {
        // Implement logic to fetch all reimbursements
        return reimbursementRepository.getAllReimbursements(employerId);
    }
}
```

Input Based Filter Implementation:

```
public class FilterService implements Filter {
```

## Company Reimbursement Management System

---

```
private final ReimbursementRepository reimbursementRepository;

@Autowired
public FilterService(ReimbursementRepository reimbursementRepository) {
    this.reimbursementRepository = reimbursementRepository;
}

@Override
public Iterable<Reimbursement> fetchReimbursements(Integer employeeId, Integer
category,Integer employerId) {
    // Implement logic to fetch pending reimbursements
    return
reimbursementRepository.findReimbursements(employeeId,category,employerId);
}
}
```

### Github link:

<https://github.com/ananya-jaianand/company-reimbursement-management-system-java>

## Individual contributions of the team members:

### Ananya Jha:

Implemented Manager interface (Frontend + Backend) which enables employers to view the reimbursements the employee's they manage have submitted and also change the status of any pending reimbursements. The employer can change a pending request status to either of the following : Pending, Approved, Denied-Non-Compliance with Company Policy, Denied-Lack of Sufficient Documentation, Denied-Expense Not Business-Related, 'Denied-Duplicate Submissions. Utilized Strategy pattern as discussed above to display results based on filters.

### Ananya J :

Implemented the Employee interface (Frontend and Backend).Employees can create a new reimbursement , edit or delete a pending reimbursement as well can filter through reimbursements as pending , approved , denied reimbursements.Approved reimbursements contain the amount refunded after the approval.Implemented Singleton Pattern when creating reimbursement calculator.

### Animesh Bhajji:

Done the whole Admin interface, which includes the Admin UI, Admin Controller and all operations involving CRUD operations by the admin

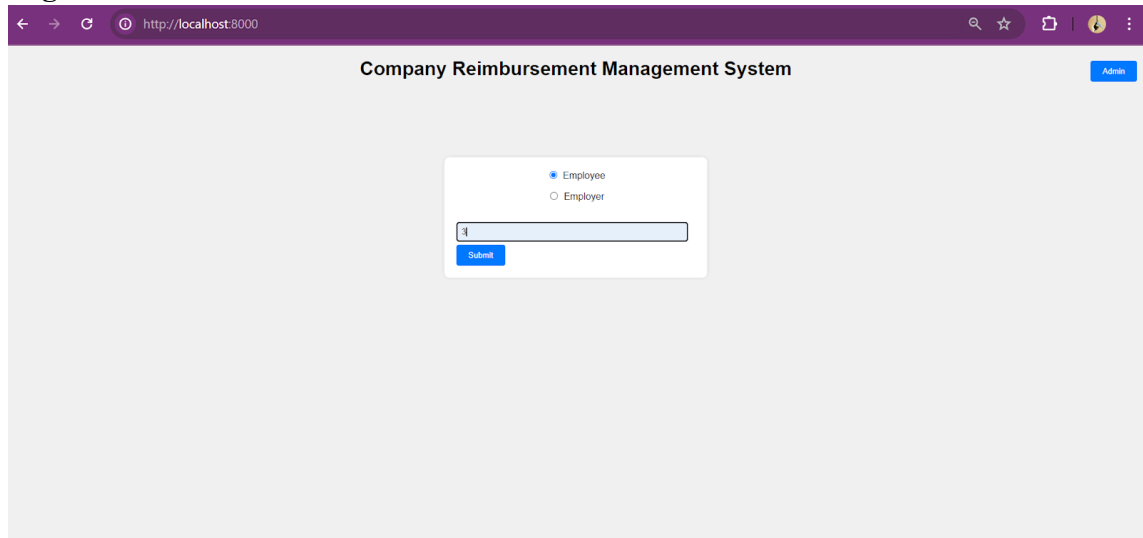
## Company Reimbursement Management System

---

### Screenshots with input values populated and output shown

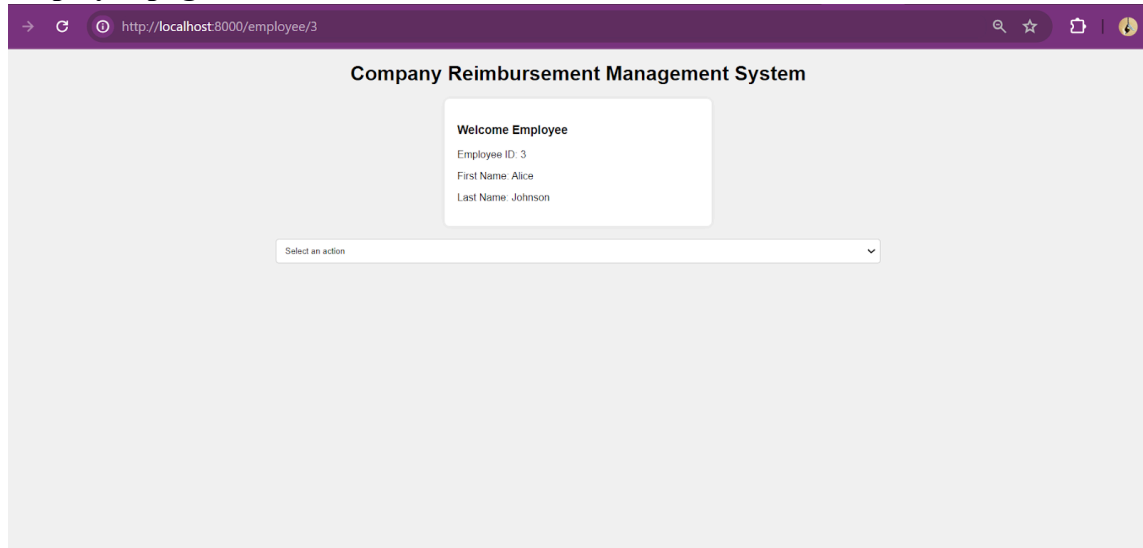
#### Employee Interface:

##### Login:



The screenshot shows a web browser window with the URL `http://localhost:8000`. The page title is "Company Reimbursement Management System". In the top right corner, there is a blue button labeled "Admin". The main content area is light gray and contains a white login box. Inside the box, there are two radio buttons: "Employee" (which is selected) and "Employer". Below the radio buttons is a text input field with a small icon on the left. At the bottom of the box is a blue button labeled "Submit".

##### Employee page:



The screenshot shows a web browser window with the URL `http://localhost:8000/employee/3`. The page title is "Company Reimbursement Management System". The main content area is light gray and contains a white box with the following text: "Welcome Employee", "Employee ID: 3", "First Name: Alice", and "Last Name: Johnson". Below this box is a white dropdown menu with the text "Select an action" and a downward arrow.

Company Reimbursement Management System

Create Reimbursement:

http://localhost:8000/employee/3

Welcome Employee

Employee ID: 3  
First Name: Alice  
Last Name: Johnson

Create a reimbursement

Create a Reimbursement

Category:

Travel

Amount Requested:

242

Receipt ID:

4940230

Document Date:

15 - 04 - 2024

Vendor Name:

abc vendor

Submit

Viewing Pending reimbursements:  
(Created reimbursement seen as request id 22)

http://localhost:8000/employee/3

Company Reimbursement Management System

Welcome Employee

Employee ID: 3  
First Name: Alice  
Last Name: Johnson

View pending reimbursement history

Pending Reimbursements

Request ID	Date Submitted	Category	Amount Requested	Receipt ID	Document Date	Vendor Name	Action
1	2023-01-05 00:00:00	Meals	300.00	RCPT001	2023-01-01 00:00:00	Restaurant A	<a href="#">Edit</a> <a href="#">Delete</a>
20	2024-04-21 22:16:27.336	Miscellaneous	100.00	4940230	2024-04-05 00:00:00	adadi vendor	<a href="#">Edit</a> <a href="#">Delete</a>
21	2024-04-21 22:29:05.96	Meals	190.00	4940980	2024-04-09 00:00:00	ramu hotel	<a href="#">Edit</a> <a href="#">Delete</a>
22	2024-04-21 22:36:45.494	Travel	242.00	4940230	2024-04-15 00:00:00	abc vendor	<a href="#">Edit</a> <a href="#">Delete</a>

Company Reimbursement Management System

Editing Pending Reimbursement:

http://localhost:8000/employee/3/edit/22

Company Reimbursement Management System

Employee Edit Reimbursement

Category:

Training

Amount Requested:

242.00

Receipt ID:

4940230

Document Date:

15-04-2024

Vendor Name:

abc vendor

Submit

(Category was updated from travel to training in request id 22)

http://localhost:8000/employee/3

Company Reimbursement Management System

Welcome Employee

Employee ID: 3

First Name: Alice

Last Name: Johnson

View pending reimbursement history

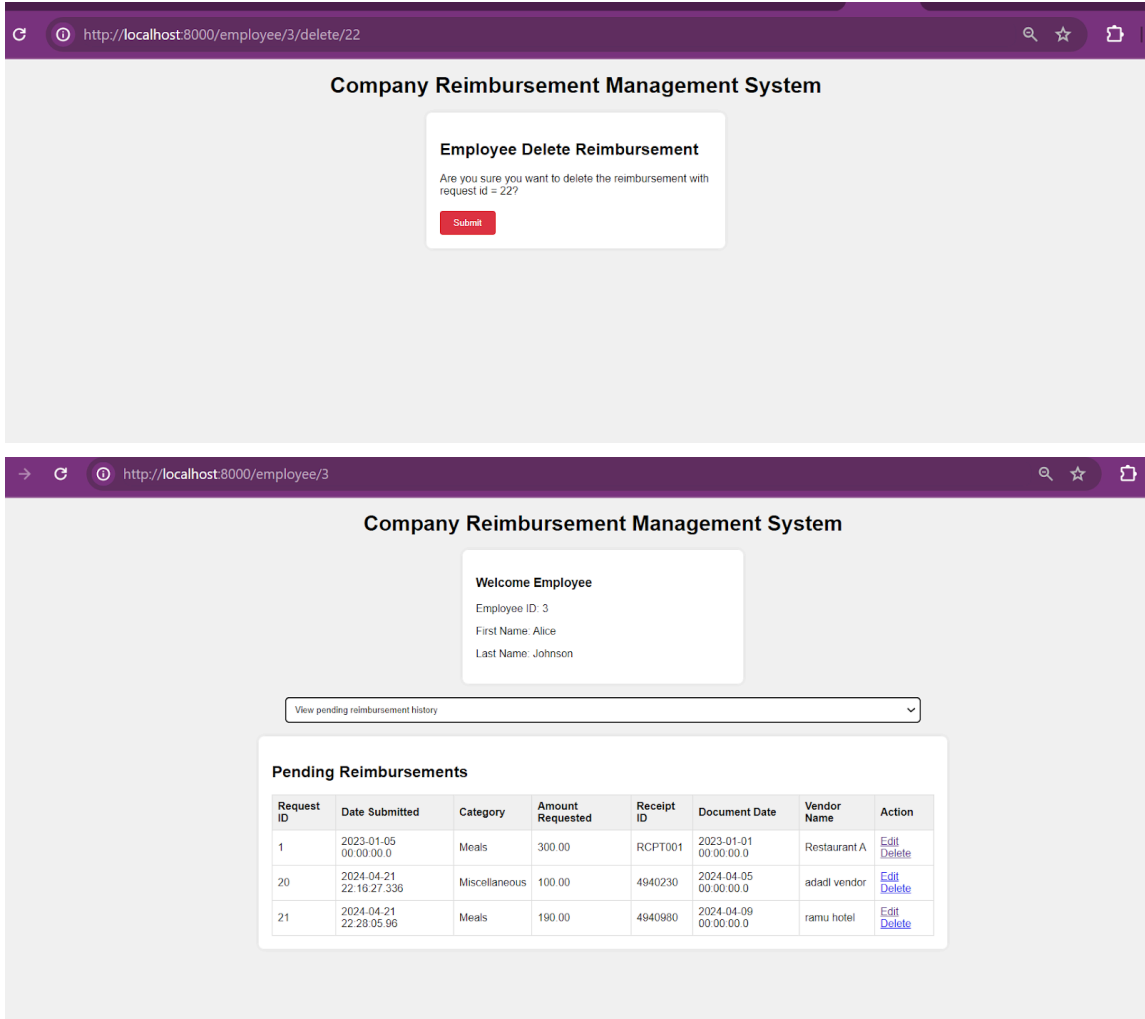
Pending Reimbursements

Request ID	Date Submitted	Category	Amount Requested	Receipt ID	Document Date	Vendor Name	Action
1	2023-01-05 00:00:00	Meals	300.00	RCPT001	2023-01-01 00:00:00	Restaurant A	<a href="#">Edit</a> <a href="#">Delete</a>
20	2024-04-21 22:16:27.336	Miscellaneous	100.00	4940230	2024-04-05 00:00:00	adadi vendor	<a href="#">Edit</a> <a href="#">Delete</a>
21	2024-04-21 22:28:05.96	Meals	190.00	4940980	2024-04-09 00:00:00	ramu hotel	<a href="#">Edit</a> <a href="#">Delete</a>
22	2024-04-21 22:36:45.494	Training	242.00	4940230	2024-04-15 00:00:00	abc vendor	<a href="#">Edit</a> <a href="#">Delete</a>

Changes reflected

Company Reimbursement Management System

Deleting a reimbursement:



successfully deleted request id 22

Company Reimbursement Management System

Viewing approved reimbursements:

http://localhost:8000/employee/3

Company Reimbursement Management System

Welcome Employee

Employee ID: 3

First Name: Alice

Last Name: Johnson

View approved reimbursement history

Approved Reimbursements

Request ID	Date Submitted	Category	Amount Requested	Receipt ID	Document Date	Vendor Name	Amount reimbursed	Date reimbursed
3	2023-03-15 00:00:00	Office Supplies	50.00	RCPT003	2023-03-10 00:00:00	Office Supply Store C	20.00	2024-04-18 18:24:30
11	2024-04-12 09:15:54	Travel	223.00	2224345	2024-03-09 00:00:00	abc vendor	178.40	2024-04-18 18:49:13

Viewing denied Reimbursement history:

http://localhost:8000/employee/3

Company Reimbursement Management System

Welcome Employee

Employee ID: 3

First Name: Alice

Last Name: Johnson

View denied reimbursement history

Denied Reimbursements

Request ID	Date Submitted	Category	Amount Requested	Receipt ID	Document Date	Vendor Name
12	2024-04-12 09:22:33	Office Supplies	242.00	24325665	2024-04-03 00:00:00	dfjkg vendor
13	2024-04-13 18:36:32	Travel	494.00	34825835	2024-02-28 00:00:00	dkjldg vendor
16	2024-04-18 16:26:14	Transportation	423.00	3243536	2024-03-01 00:00:00	ertolt vndor



Company Reimbursement Management System

Employer Interface:

Login:

Company Reimbursement Management System

Admin

Employee

Employer

1

Submit

Show all reimbursements:

Company Reimbursement Management System

Welcome Employer!

Employee ID: 1

First Name: John

Last Name: Doe

To view requests please fill the following:

Show All

Category:

Select category

Employeeid:

Select Employee ID

Status:

Select Status

Submit

The employer will only be able to change status of a pending request (as can be seen by the term “final” for non-pending requests and a radio box a pending request):

Reimbursements List

	Reimbursement ID	Category	Employee ID	Status	Date Submitted	Amount Requested	Receipt ID	Document Date	Vendor Name
<input type="radio"/>	6	Conference Fees	6	Pending	2023-06-20 00:00:00.0	350.00	RCPT006	2023-06-12 00:00:00.0	Conference Organizer F

Company Reimbursement Management System

Reimbursements List

	Reimbursement ID	Category	Employee ID	Status	Date Submitted	Amount Requested	Receipt ID	Document Date	Vendor Name
Final	5	Travel	12	Approved	2023-05-24 00:00:00.0	150.00	RCPT005	2023-05-20 00:00:00.0	Travel Agency E

Once the employer selects a request they can enter the new status:

Reimbursements List

	Reimbursement ID	Category	Employee ID	Status	Date Submitted	Amount Requested	Receipt ID	Document Date	Vendor Name
<input checked="" type="radio"/>	6	Conference Fees	6	Pending	2023-06-20 00:00:00.0	350.00	RCPT006	2023-06-12 00:00:00.0	Conference Organizer F

Enter new status id:

Submit

The employer can apply category, employee, status filters:

Company Reimbursement Management System

Welcome Employer!

Employee ID: 1  
First Name: John  
Last Name: Doe

To view requests please fill the following:

☐ Show All

Category:

Employeeid:

Status:

Submit

Company Reimbursement Management System

Reimbursements List

	Reimbursement ID	Category	Employee ID	Status	Date Submitted	Amount Requested	Receipt ID	Document Date	Vendor Name
Final	3	Office Supplies	3	Denied-Expense Not Business-Related	2023-03-15 00:00:00.0	50.00	RCPT003	2023-03-10 00:00:00.0	Office Supply Store C

Admin Interface:

Company Reimbursement Management System - Admin Page

All Employees

ID	First Name	Last Name	Email	Department	Actions
1	John	Doe	john.doe@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
2	Jane	Smith	jane.smith@example.com	HR	<a href="#">Edit</a> <a href="#">Delete</a>
3	Alice	Johnson	alice.johnson@example.com	Finance	<a href="#">Edit</a> <a href="#">Delete</a>
4	Bob	Williams	bob.williams@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>
5	Charlie	Miller	charlie.miller@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
6	Diana	Brown	diana.brown@example.com	HR	<a href="#">Edit</a> <a href="#">Delete</a>
7	Edward	Taylor	edward.taylor@example.com	Finance	<a href="#">Edit</a> <a href="#">Delete</a>
8	Fiona	Clark	fiona.clark@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
9	George	Wilson	george.wilson@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>
10	Helen	Moore	helen.moore@example.com	HR	<a href="#">Edit</a> <a href="#">Delete</a>
11	Isaac	Anderson	isaac.anderson@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
12	Jasmine	Perez	jasmine.perez@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>

Add Employee

First Name:

Last Name:

Email:

Department:

Salary:

Adding employee:

4	Bob	Williams	bob.williams@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>
5	Charlie	Miller	charlie.miller@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
6	Diana	Brown	diana.brown@example.com	HR	<a href="#">Edit</a> <a href="#">Delete</a>
7	Edward	Taylor	edward.taylor@example.com	Finance	<a href="#">Edit</a> <a href="#">Delete</a>
8	Fiona	Clark	fiona.clark@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
9	George	Wilson	george.wilson@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>
10	Helen	Moore	helen.moore@example.com	HR	<a href="#">Edit</a> <a href="#">Delete</a>
11	Isaac	Anderson	isaac.anderson@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
12	Jasmine	Perez	jasmine.perez@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>

Add Employee

First Name:

Animesh

Last Name:

Bhaiji

Email:

Animesh.Bhaiji@gmail.com

Department:

CS

Salary:

999999

Phone Number:

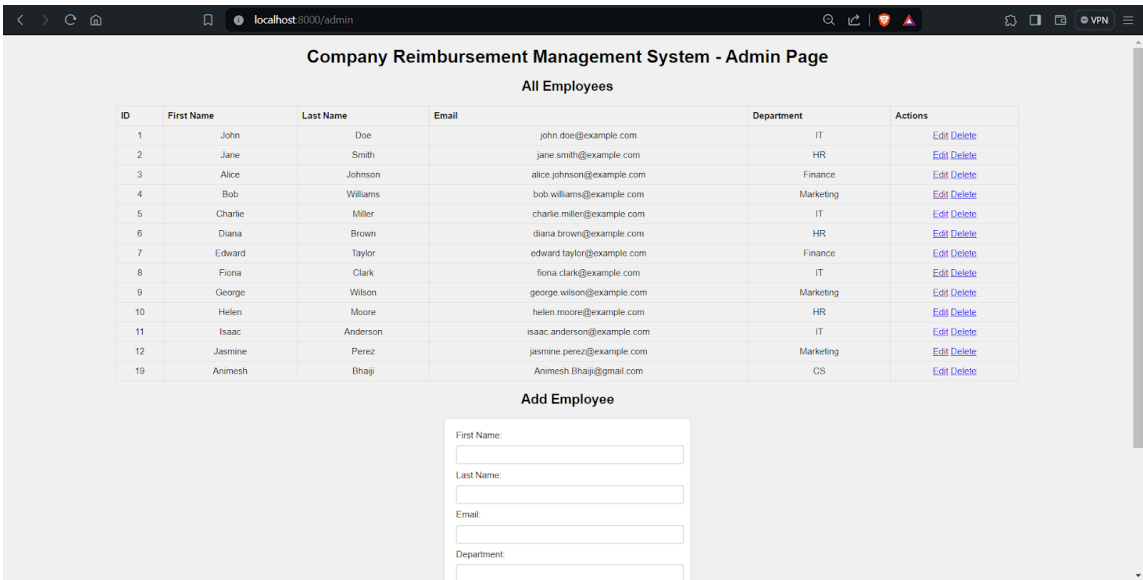
1234567890

Manager ID:

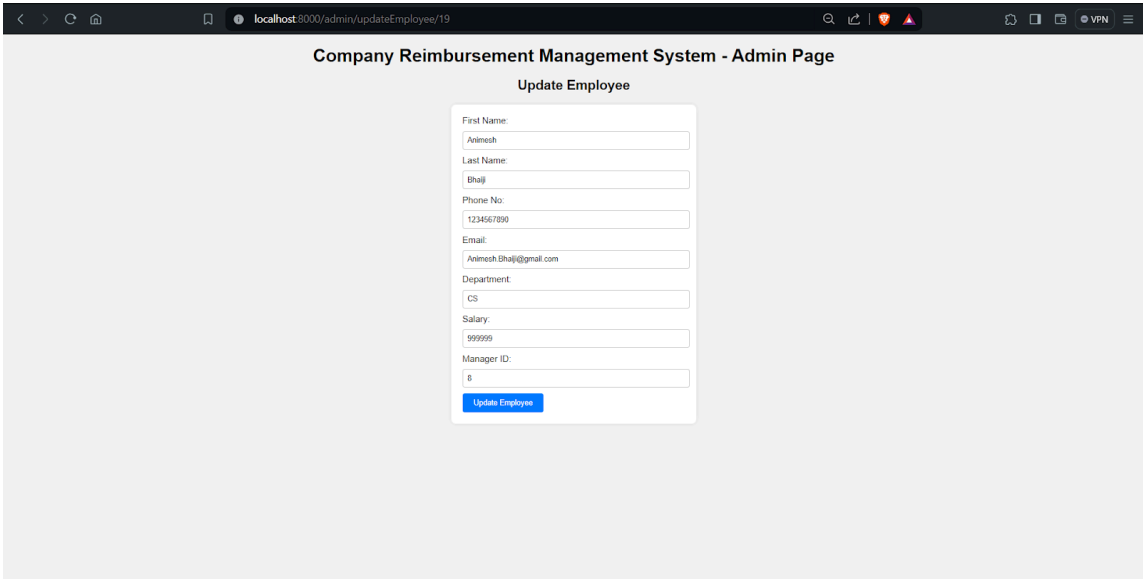
8

Add Employee

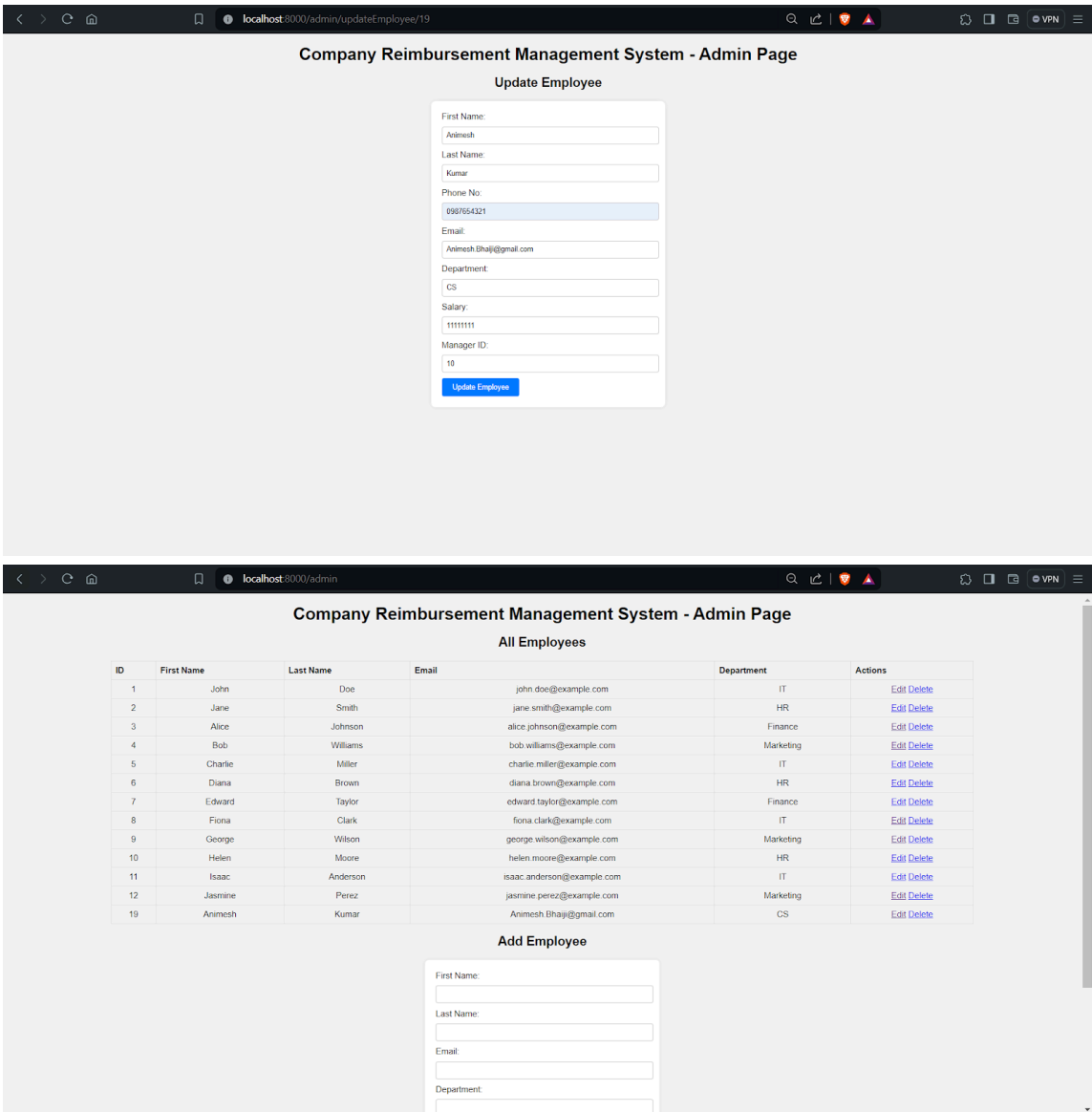
Company Reimbursement Management System



Editing Employee details:



Company Reimbursement Management System



Company Reimbursement Management System

Deleting an Employee:

localhost:8000/admin

Company Reimbursement Management System - Admin Page

All Employees

ID	First Name	Last Name	Email	Department	Actions
1	John	Doe	john.doe@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
2	Jane	Smith	jane.smith@example.com	HR	<a href="#">Edit</a> <a href="#">Delete</a>
3	Alice	Johnson	alice.johnson@example.com	Finance	<a href="#">Edit</a> <a href="#">Delete</a>
4	Bob	Williams	bob.williams@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>
5	Charlie	Miller	charlie.miller@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
6	Diana	Brown	diana.brown@example.com	HR	<a href="#">Edit</a> <a href="#">Delete</a>
7	Edward	Taylor	edward.taylor@example.com	Finance	<a href="#">Edit</a> <a href="#">Delete</a>
8	Fiona	Clark	fiona.clark@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
9	George	Wilson	george.wilson@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>
10	Helen	Moore	helen.moore@example.com	HR	<a href="#">Edit</a> <a href="#">Delete</a>
11	Isaac	Anderson	isaac.anderson@example.com	IT	<a href="#">Edit</a> <a href="#">Delete</a>
12	Jasmine	Perez	jasmine.perez@example.com	Marketing	<a href="#">Edit</a> <a href="#">Delete</a>

Add Employee

First Name:

Last Name:

Email:

Department:

Salary: