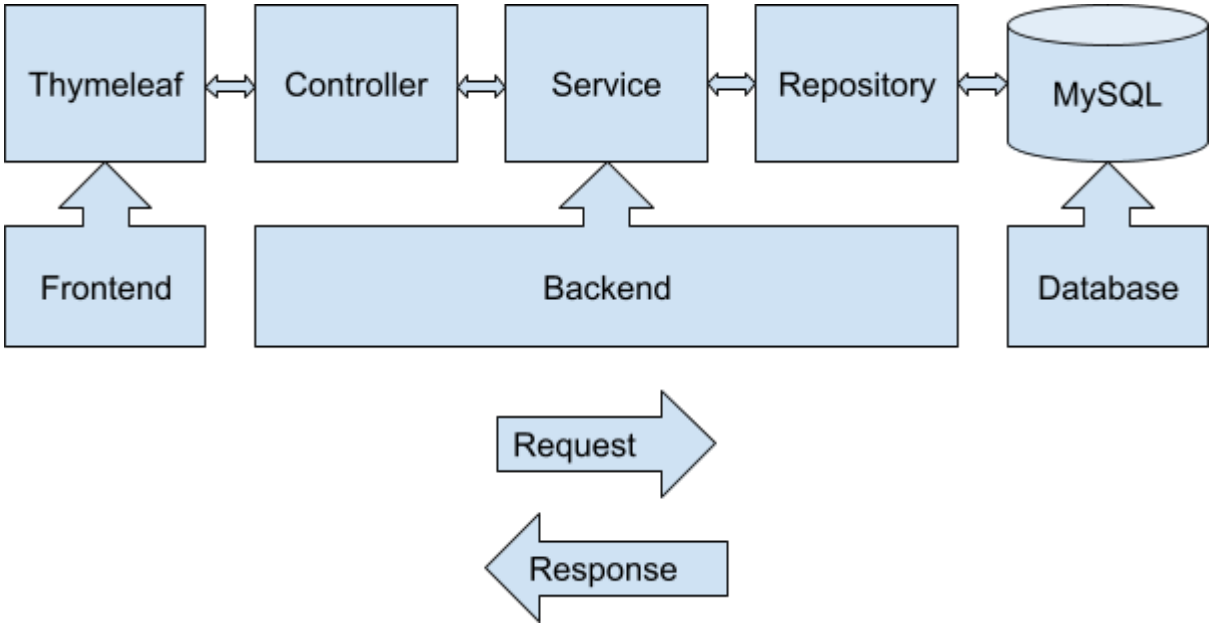
Project 01 - Employee Management System

Project Available at Github: https://github.com/zvdas/EmployeeManagementSystem.git

Application Structure



Features

The following features will be implemented in this project:
- List Employee Feature
- Add Employee Feature
- Update Employee Feature
- Delete Employee Feature
- Pagination Feature
- Sorting Feature
- Login Feature
- Registration Feature
- Logout Feature

Tools & Technologies

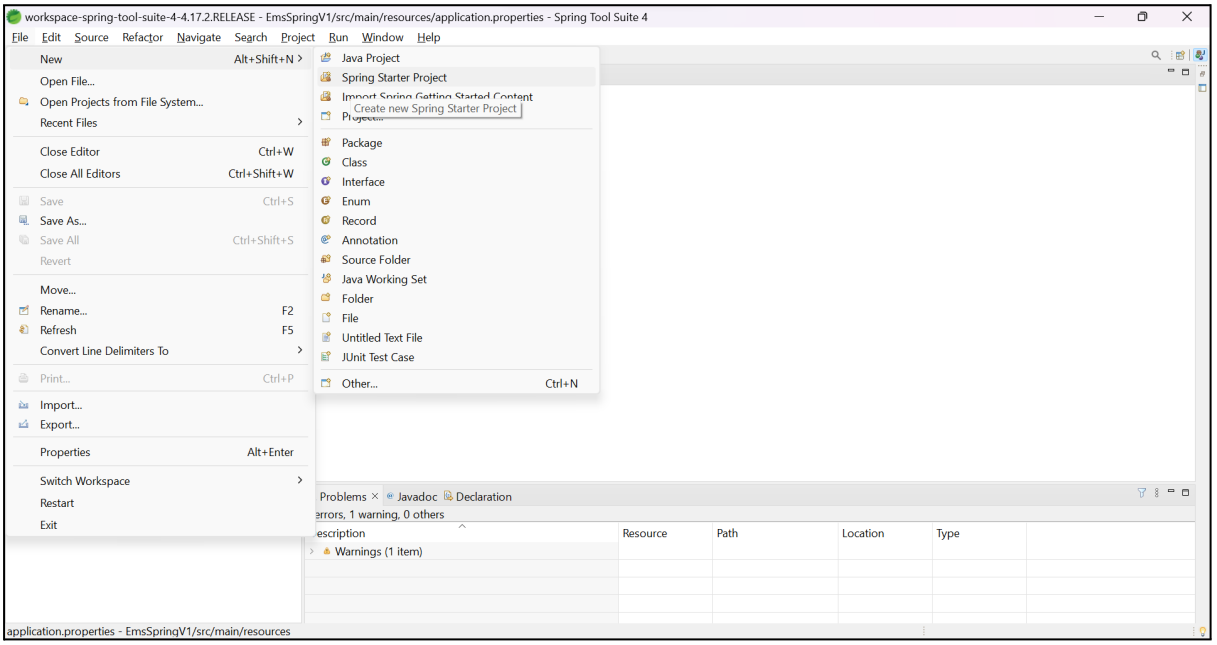
Tools and technologies used:
Java 8+: High-level, class-based, object-oriented programming language
Spring Boot: Application framework to build web applications in Java
Spring Data JPA: Reduce boilerplate code to perform database operations programmatically (internally uses Hibernate ORM)
Spring Security: Framework that provides authentication & authorization to Java applications
MySQL: Relational database management system

Eclipse STS: Java IDE (Integrated Development Environment) tailored for developing Spring-based applications
Maven: Build tool to add dependencies
Tomcat: Server
Thymeleaf: Template rendering engine

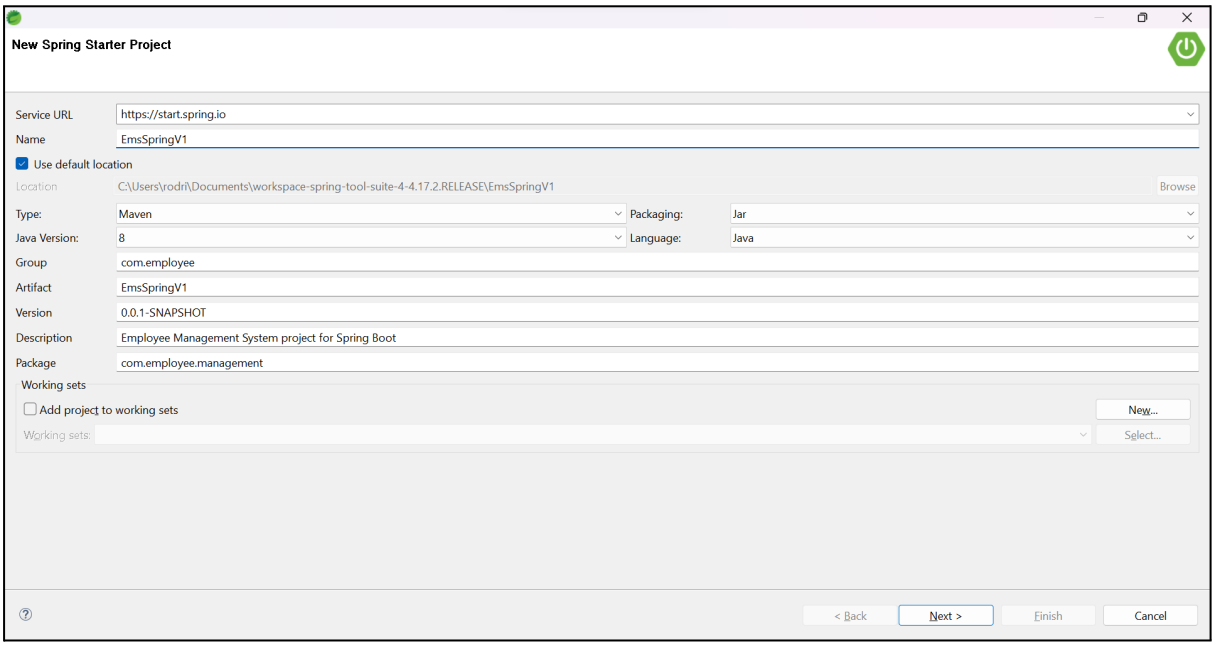
Steps

Step 1: Create Spring Boot Project

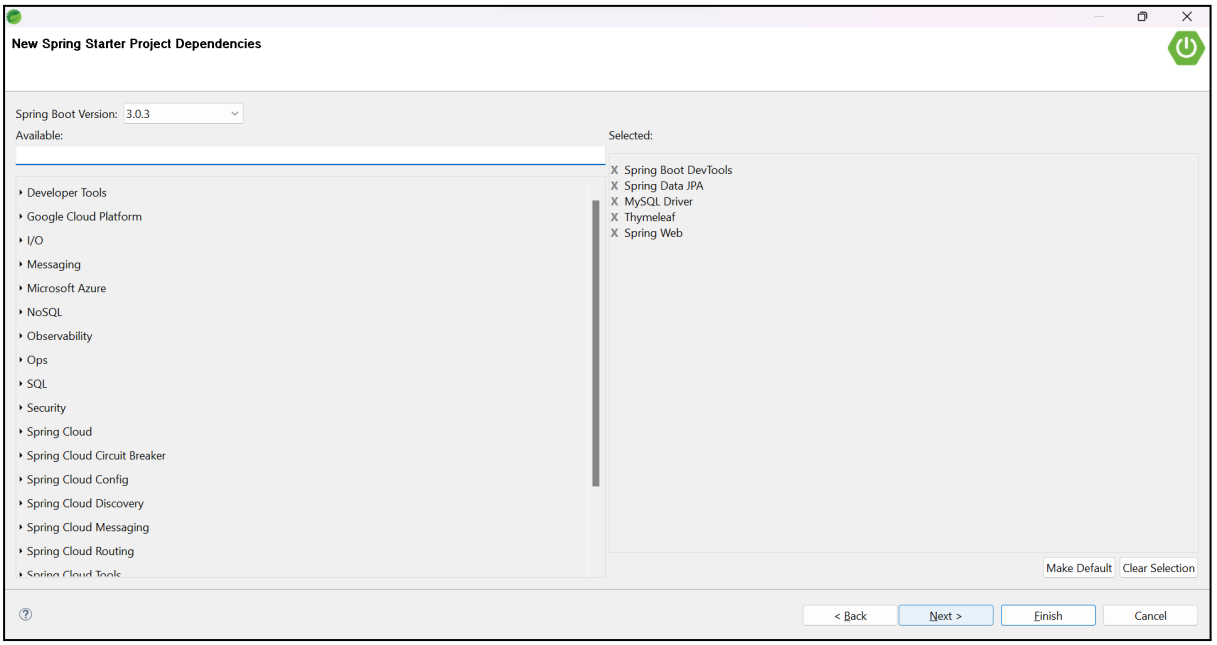
Create a spring boot project in Eclipse STS by clicking *File>New>Spring Starter project*.



Name the application (EmsSpringV1), select Type as *Maven*, packaging as *jar*, Java version as *8*.



Add the dependencies *Spring Web, Spring Data JPA, Spring Boot DevTools, MySQL Driver, Thymeleaf. Spring Web* enables building of RESTful web services and spring applications, *Spring Data JPA* implements Hibernate and allows model connection to database, *Spring Boot DevTools* provides autocompletion features in Eclipse STS for annotations, *MySQL Driver* allows connection of MySQL database to spring project, *Thymeleaf* allows generation of MVC template.



Step 2: Maven Dependencies

Below is the pom.xml file for reference

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project
	xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
	https://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.3.0.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>
	<groupId>com.employee</groupId>
	<artifactId>EmsSpringV1</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<name>EmsSpringV1</name>
	<description>Employee Management System project for Spring Boot</description>
	<properties>
		<java.version>1.8</java.version>
	</properties>
```

```
<dependencies>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-data-jpa</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-security</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-thymeleaf</artifactId>
	</dependency>
	<dependency>
		<groupId>org.thymeleaf.extras</groupId>
		<artifactId>thymeleaf-extras-springsecurity5</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-web</artifactId>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-devtools</artifactId>
		<scope>runtime</scope>
		<optional>true</optional>
	</dependency>
	<dependency>
		<groupId>mysql</groupId>
		<artifactId>mysql-connector-java</artifactId>
		<scope>runtime</scope>
	</dependency>
	<dependency>
		<groupId>com.h2database</groupId>
		<artifactId>h2</artifactId>
		<scope>runtime</scope>
	</dependency>
	<dependency>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-test</artifactId>
		<scope>test</scope>
		<exclusions>
			<exclusion>
				<groupId>org.junit.vintage</groupId>
				<artifactId>junit-vintage-engine</artifactId>
			</exclusion>
		</exclusions>
	</dependency>
	<dependency>
		<groupId>org.springframework.security</groupId>
		<artifactId>spring-security-test</artifactId>
		<scope>test</scope>
	</dependency>
</dependencies>
```

```
	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

</project>
```

Step 3: Configure MySQL Database

Spring Boot tries to auto-configure a DataSource if spring-data-jpa dependency is in the classpath by reading the database configuration from the *application.properties* file. The configurations must be added and Spring Boot will handle the rest.

Open the application.properties file and add the following properties to it.

application.properties

```
# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)

spring.datasource.url=jdbc:mysql://localhost:3306/ems?useSSL=false&serverTimezone=U
TC&useLegacyDatetimeCode=false
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate

# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

# Hibernate debug & logging to console
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type=TRACE
```

Create a database named *ems* in MySQL, and change the spring.datasource.username & spring.datasource.password properties as per the MySQL installation.

In the above properties file, the last two properties are for Hibernate. Spring Boot uses Hibernate as the default JPA implementation.

The property spring.jpa.hibernate.ddl-auto is used for database initialization. The value “update” for this property has been used, which will update the current record instead of creating a new separate record.

Step 4: Create Models

Create models or entities for the Employee Management System application by creating a new package called *model* inside *com.employee.management*. Annotations are used to connect the model to the database.

Employee Class

com.employee.management.model > Employee.java

```
package com.employee.management.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "employees")
public class Employee {

	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private long id;

	@Column(name = "first_name")
	private String firstName;

	@Column(name = "last_name")
	private String lastName;

	@Column(name = "email")
	private String email;

	public long getId() {
		return id;
	}
	public void setId(long id) {
		this.id = id;
	}
	public String getFirstName() {
		return firstName;
	}
	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}
	public String getLastName() {
		return lastName;
	}
	public void setLastName(String lastName) {
		this.lastName = lastName;
```

```
	}
	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}

}
```

User Class

create a *User* class inside the *model* package

com.employee.management.model > User.java

```
package com.employee.management.model;

import java.util.Collection;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.Table;
import jakarta.persistence.UniqueConstraint;

@Entity
@Table(name = "user", uniqueConstraints = @UniqueConstraint(columnNames =
"email"))
public class User {

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

	@Column(name = "first_name")
	private String firstName;

	@Column(name = "last_name")
	private String lastName;

	private String email;

	private String password;

	/*
```

```
	 * One user can have multiple roles,
	 * and one role can be associated with multiple users
	 * roles retrieved 'eagerly' with user (else only on demand - 'lazy')
	 * JoinTable creates a third table for M2M mapping between both entities
	*/
	@ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
	@JoinTable(
			name = "users_roles",
			joinColumns = @JoinColumn(
					name = "user_id", referencedColumnName = "id"),
			inverseJoinColumns = @JoinColumn(
					name = "role_id", referencedColumnName = "id"))
	private Collection<Role> roles;

	public User() {
		// TODO Auto-generated constructor stub
	}

	public User(String firstName, String lastName, String email, String password,
Collection<Role> roles) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
		this.email = email;
		this.password = password;
		this.roles = roles;
	}

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public String getEmail() {
		return email;
```

```
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

	public Collection<Role> getRoles() {
		return roles;
	}

	public void setRoles(Collection<Role> roles) {
		this.roles = roles;
	}

}
```

Role Class

create a *Role* class inside the *model* package

com.employee.management.model > Role.java

```
package com.employee.management.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "role")
public class Role {

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

	private String name;

	public Role() {
		// TODO Auto-generated constructor stub
	}
```

```
	public Role(String name) {
		super();
		this.name = name;
	}

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

}
```

Step 5: Create Configuration

Create a class *SecurityConfiguration* inside the *configuration* package, where the custom configuration with spring security support for login will be stored using Configuration and EnableWebSecurity annotations.

com.employee.management.config > SecurityConfiguration.java

```
package com.employee.management.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationMan
agerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerA
dapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;

import com.employee.management.service.UserService;

@Configuration
@EnableWebSecurity
```

```
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

	@Autowired
	private UserService userService;

	/* encode passwords with Bcrypt */
	@Bean
	public BCryptPasswordEncoder passwordEncoder() {
		return new BCryptPasswordEncoder();
	}

	/* authorize userService using Bcrypt */
	@Bean
	public DaoAuthenticationProvider authenticationProvider() {
		DaoAuthenticationProvider auth = new DaoAuthenticationProvider();
		auth.setUserDetailsService(userService);
		auth.setPasswordEncoder(passwordEncoder());
		return auth;
	}

	/* pass authentication provided to configure method */
	@Override
	protected void configure(AuthenticationManagerBuilder auth) throws Exception {
		auth.authenticationProvider(authenticationProvider());
	}

	/* configure url access, login & logout */
	@Override
	protected void configure(HttpSecurity http) throws Exception {
		http.authorizeRequests().antMatchers(
					"/registration**",
					"/js/**",
					"/css/**",
					"/img/**").permitAll()
		.anyRequest().authenticated()
		.and()
		.formLogin()
		.loginPage("/login")
		.permitAll()
		.and()
		.logout()
		.invalidateHttpSession(true)
		.clearAuthentication(true)
		.logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
		.logoutSuccessUrl("/login?logout")
		.permitAll();
	}

}
```

Step 6: Create Repositories

EmployeeRepository interface

Create an *EmployeeRepository* interface inside the *repository* package, which will extend JpaRepository (defines methods for all the CRUD operations on the entity; default implementation of the SimpleJpaRepository)

com.employee.management.repository > EmployeeRepository.java

```
 package com.employee.management.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.employee.management.model.Employee;

@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long>{

}
```

UserRepository interface

Create a UserRepository interface inside the repository package, which will extend JpaRepository (defines methods for all the CRUD operations on the entity; default implementation of the SimpleJpaRepository)

com.employee.management.repository > UserRepository.java

```
 package com.employee.management.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.employee.management.model.User;

@Repository
public interface UserRepository extends JpaRepository<User, Long>{
	/* retrieve user details by email during login spring security */
	User findByEmail(String email);
}
```

Step 7: Create Services

EmployeeService class

Create an *EmployeeService* interface inside the *service* package

com.employee.management.service > EmployeeService.java

```
package com.employee.management.service;
```

```
import java.util.List;

import org.springframework.data.domain.Page;

import com.employee.management.model.Employee;

public interface EmployeeService {
	List<Employee> getAllEmployees();
	void saveEmployee(Employee employee);
	Employee getEmployeeById(long id);
	void deleteEmployeeId(long id);
	Page<Employee> findPaginated(int pageNumber, int pageSize, String sortField,
String sortDirection);
}
```

EmployeeServiceImpl class

Create an *EmployeeServiceImpl* class which implements *EmployeeService* interface. @Autowired is used to inject the employee repository.

com.employee.management.service > EmployeeServiceImpl.java

```
package com.employee.management.service;

import java.util.List;
import java.util.Optional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.stereotype.Service;

import com.employee.management.model.Employee;
import com.employee.management.repository.EmployeeRepository;

@Service
public class EmployeeServiceImpl implements EmployeeService {

	@Autowired
	private EmployeeRepository employeeRepository;

	@Override
	public List<Employee> getAllEmployees() {
		return employeeRepository.findAll();
	}

	@Override
	public void saveEmployee(Employee employee) {
		employeeRepository.save(employee);
	}
```

```
	@Override
	public Employee getEmployeeById(long id) {
		Optional<Employee> optional = employeeRepository.findById(id);
		Employee employee = null;
		if(optional.isPresent()) {
			employee = optional.get();
		} else {
			throw new RuntimeException("Employee with id::"+id+"not found");
		}
		return employee;
	}

	@Override
	public void deleteEmployeeId(long id) {
		employeeRepository.deleteById(id);
	}

	@Override
	public Page<Employee> findPaginated(int pageNumber, int pageSize, String
sortField, String sortDirection) {
		// toggle sorting by ascending/descending order using ternary operator
		Sort sort = sortDirection.equalsIgnoreCase(Sort.Direction.ASC.name()) ?
Sort.by(sortField).ascending() : Sort.by(sortField).descending();
		// spring data JPA considers pages starting from 0 internally, hence
pageNumber - 1
		Pageable pageable = PageRequest.of(pageNumber - 1, pageSize, sort);
		return employeeRepository.findAll(pageable);
	}

}
```

UserRegistrationDto class

Create a *UserRegistrationDto* class is a DTO (Data Transfer Object) class which sends & receives bulk information to & from the database.

com.employee.management.dto > UserRegistrationDto.java

```
package com.employee.management.web.dto;

public class UserRegistrationDto {
	private String firstName;
	private String lastName;
	private String email;
	private String password;

	public UserRegistrationDto(String firstName, String lastName, String email, String
password) {
		super();
		this.firstName = firstName;
		this.lastName = lastName;
		this.email = email;
		this.password = password;
	}
```

```
	public UserRegistrationDto() {
		// TODO Auto-generated constructor stub
	}

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public String getLastName() {
		return lastName;
	}

	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public String getEmail() {
		return email;
	}

	public void setEmail(String email) {
		this.email = email;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

}
```

UserService class

Create a *UserService* interface inside the *service* package

com.employee.management.service > UserService.java

```
package com.employee.management.service;

import org.springframework.security.core.userdetails.UserDetailsService;

import com.employee.management.dto.UserRegistrationDto;
import com.employee.management.model.User;

public interface UserService extends UserDetailsService {
	User save(UserRegistrationDto registrationDto);
```

```
}
```

UserServiceImpl class

Create a *UserServiceImpl* class which implements *UserService* interface.

com.employee.management.service > UserServiceImpl.java

```
package com.employee.management.service;

import java.util.Arrays;
import java.util.Collection;
import java.util.stream.Collectors;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import com.employee.management.dto.UserRegistrationDto;
import com.employee.management.model.Role;
import com.employee.management.model.User;
import com.employee.management.repository.UserRepository;

@Service
public class UserServiceImpl implements UserService {

	private UserRepository userRepository;

	@Autowired
	private BCryptPasswordEncoder passwordEncoder;

	public UserServiceImpl(UserRepository userRepository) {
		super();
		this.userRepository = userRepository;
	}

	@Override
	public User save(UserRegistrationDto registrationDto) {
		User user = new User(registrationDto.getFirstName(),
				registrationDto.getLastName(), registrationDto.getEmail(),
				passwordEncoder.encode(registrationDto.getPassword()),
Arrays.asList(new Role("ROLE_USER")));
		return userRepository.save(user);
	}

	@Override
	public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
		User user = userRepository.findByEmail(username);
		if(user == null) {
```

```
                        throw new UsernameNotFoundException("Invalid username or
password");
                }
                return new
org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(),
mapRolesToAuthorities(user.getRoles()));
        }

        private Collection<? extends GrantedAuthority>
mapRolesToAuthorities(Collection<Role> roles) {
                return roles.stream().map(role -> new
SimpleGrantedAuthority(role.getName())).collect(Collectors.toList());
        }

}
```

Step 8: Create Controllers

EmployeeController class

Create an *EmployeeController* class inside the *controller* package

com.employee.management.controller > EmployeeController.java

```
package com.employee.management.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

import com.employee.management.model.Employee;
import com.employee.management.service.EmployeeService;

@Controller
public class EmployeeController {

        @Autowired
        private EmployeeService employeeService;

        // display list of employees
        @GetMapping("/")
        public String viewHomePage(Model model) {
                /*
                // return all employees without pagination
                model.addAttribute("listEmployees", employeeService.getAllEmployees());
```

```
		return "index";
		*/

		// return all employees with pagination & sorting
		return findPaginated(1, "firstName", "asc", model);
	}

	// show new employee form
	@GetMapping("/showNewEmployeeForm")
	public String showNewEmployeeForm(Model model) {
		// create model attribute to bind form data
		Employee employee = new Employee();
		model.addAttribute("employee", employee);
		return "new_employee";
	}

	// save employee
	@PostMapping("/saveEmployee")
	public String saveEmployee(@ModelAttribute("employee") Employee employee) {
		// save employee to database
		employeeService.saveEmployee(employee);
		return "redirect:/";
	}

	// show form for update
	@GetMapping("/showFormForUpdate/{id}")
	public String showFormForUpdate(@PathVariable (value = "id") long id, Model
model) {
		// get employee from service
		Employee employee = employeeService.getEmployeeById(id);
		// set employee as a model attribute to pre-populate the form
		model.addAttribute("employee", employee);
		return "new_employee";
	}

	// delete employee
	@GetMapping("/deleteEmployee/{id}")
	public String deleteEmployee(@PathVariable (value = "id") long id) {
		// delete employee
		employeeService.deleteEmployeeId(id);
		return "redirect:/";
	}

	// handle pagination & sorting
	@GetMapping("/page/{pageNumber}")
	public String findPaginated(
			@PathVariable (value = "pageNumber") int pageNumber,
			@RequestParam ("sortField") String sortField,
			@RequestParam ("sortDirection") String sortDirection,
			Model model
			) {
		// set number of records per page
		int pageSize = 5;
		// find employees by page number & page size
```

```
		Page<Employee> page = employeeService.findPaginated(pageNumber,
pageSize, sortField, sortDirection);
		// list the employees found above
		List<Employee> listEmployees = page.getContent();
		// add attributes to the index page for pagination
		model.addAttribute("currentPage", pageNumber);
		model.addAttribute("totalPages", page.getTotalPages());
		model.addAttribute("totalRecords", page.getTotalElements());
		// add attributes to the index page for sorting
		model.addAttribute("sortField", sortField);
		model.addAttribute("sortDirection", sortDirection);
		model.addAttribute("reverseSortDirection", sortDirection.equals("asc") ?
"desc" : "asc");
		// add attributes to the index page for display
		model.addAttribute("listEmployees", listEmployees);
		return "index";
	}

}
```

UserRegistrationController class

Create a *UserRegistrationController* class inside the *controller* package

com.employee.management.controller > UserRegistrationController.java

```
package com.employee.management.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import com.employee.management.dto.UserRegistrationDto;
import com.employee.management.service.UserService;

@Controller
@RequestMapping("/registration")
public class UserRegistrationController {

	private UserService userService;

	public UserRegistrationController(UserService userService) {
		super();
		this.userService = userService;
	}

	@ModelAttribute("user")
	public UserRegistrationDto userRegistrationDto() {
		return new UserRegistrationDto();
	}

	@GetMapping
```

```
	public String showRegistrationForm() {
		return "registration";
	}

	@PostMapping
	public String registerUserAccount(@ModelAttribute("user") UserRegistrationDto
registrationDto){
		userService.save(registrationDto);
		return "redirect:/registration?success";
	}
}
```

MainController class

Create a *MainController* class inside the *controller* package

com.employee.management.controller > MainController.java

```
package com.employee.management.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class MainController {

	@GetMapping("/login")
	public String login() {
		return "login";
	}
}
```

Step 9: Create View Templates (Thymeleaf)

Create the thymeleaf templates in the *templates* folder of *src/main/resources*.

Navbar - Include in Index & New Employee Page

Templates > navbar.html

```
<!-- create navbar -->
<nav class="navbar navbar-expand-lg bg-dark">
	<div class="container-fluid">
		<a class="navbar-brand text-primary" href="#">Employee Management
System</a>
		<ul class="navbar-nav me-auto mb-2 mb-lg-0">
	<li class="nav-item">
		<a class="nav-link active text-primary" aria-current="page" href="#"
th:href="@{/}">Home</a>
	</li>
      </ul>
      <ul class="navbar-nav ms-auto mb-2 mb-lg-0">
	<li class="nav-item">
```

```
                <a class="nav-link disabled text-primary">Welcome, <span
sec:authentication="principal.username">User</span></a>
        </li>
                        <li class="nav-item" sec:authorize="isAuthenticated()">
                                <a class="nav-link active text-primary"
th:href="@{/logout}">Logout</a>
                        </li>
                </ul>
        </div>
</nav>
```

Index - Read All Employees

templates > index.html

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="ISO-8859-1">
                <!-- bootstrap cdn -->
                <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7c
mDA6j6gD" crossorigin="anonymous">
                <title>Employee Management System</title>
        </head>

        <body>
                <div th:insert="navbar"></div>
                <div align="center" class="container">
                        <h1>Employee List</h1>
                        <a class="btn btn-primary my-2"
th:href="@{/showNewEmployeeForm}">Add Employee</a>
                        <table class="table table-dark table-striped table-responsive
table-hover" border="1">
                                <thead>
                                        <tr>
                                                <th>
                                                        <a th:href="@{'/page/' +
${currentPage} + '?sortField=firstName&sortDirection=' + ${reverseSortDirection}}">
                                                                First Name
                                                        </a>
                                                </th>
                                                <th>
                                                        <a th:href="@{'/page/' +
${currentPage} + '?sortField=lastName&sortDirection=' + ${reverseSortDirection}}">
                                                                Last Name
                                                        </a>
                                                </th>
                                                <th>
                                                        <a th:href="@{'/page/' +
${currentPage} + '?sortField=email&sortDirection=' + ${reverseSortDirection}}">
```

```
                                                Email
                                            </a>
                                        </th>

                                        <th>Actions</th>
                                    </tr>
                                </thead>
                                <tbody>
                                    <tr th:each="employee: ${listEmployees}">
                                        <td th:text="${employee.firstName}"></td>
                                        <td th:text="${employee.lastName}"></td>
                                        <td th:text="${employee.email}"></td>
                                        <td>
                                            <a
th:href="@{/showFormForUpdate/{id}(id=${employee.id})}" class="btn btn-primary">
                                                Update
                                            </a>
                                            <a
th:href="@{/deleteEmployee/{id}(id=${employee.id})}" class="btn btn-danger">
                                                Delete
                                            </a>
                                        </td>
                                    </tr>
                                </tbody>
                            </table>

                            <!-- pagination -->
                            <div th:if="${totalPages > 1}">
                                <div class="row col-10">
                                    <div class="col-2">
                                        Total rows: [[${totalRecords}]]
                                    </div>
                                    <div class="col-2">
                                        <!-- print First page link -->
                                        <a th:if="${currentPage > 1}"
th:href="@{'/page/' + 1 + '?sortField=' + ${sortField} + '&sortDirection=' +
${sortDirection}}">
                                            First
                                        </a>
                                        <!-- print First in first page as text -->
                                        <span th:unless="${currentPage >
1}">First</span>
                                    </div>
                                    <div class="col-2">
                                        <!-- print Previous page link -->
                                        <a th:if="${currentPage > 1}"
th:href="@{'/page/' + ${currentPage - 1} + '?sortField=' + ${sortField} + '&sortDirection=' +
${sortDirection}}">
                                            Previous
                                        </a>
                                        <!-- print Previous in first page as text -->
                                        <span th:unless="${currentPage >
1}">Previous</span>
                                    </div>
```

```
                                        <div class="col-2">
                                                <!-- print sequence of page numbers starting
from 1 -->
                                                <span th:each="i:
${#numbers.sequence(1,totalPages)}">
                                                        <!-- print page numbers other than
current page as links -->
                                                        <a th:if="${currentPage != i}"
th:href="@{'/page/' + ${i} + '?sortField=' + ${sortField} + '&sortDirection=' +
${sortDirection}}">
                                                                [[${i}]]
                                                        </a>
                                                        <!-- print current page number as text
-->
                                                        <span th:unless="${currentPage !=
i}">[[${i}]]</span>
                                                </span>
                                        </div>
                                        <div class="col-2">
                                                <!-- print Next page link -->
                                                <a th:if="${currentPage < totalPages}"
th:href="@{'/page/' + ${currentPage + 1} + '?sortField=' + ${sortField} + '&sortDirection=' +
${sortDirection}}">
                                                        Next
                                                </a>
                                                <!-- print Next in last page as text -->
                                                <span th:unless="${currentPage <
totalPages}">Next</span>
                                        </div>
                                        <div class="col-2">
                                                <!-- print Last page link -->
                                                <a th:if="${currentPage < totalPages}"
th:href="@{'/page/' + ${totalPages} + '?sortField=' + ${sortField} + '&sortDirection=' +
${sortDirection}}">
                                                        Last
                                                </a>
                                                <!-- print Last in last page as text -->
                                                <span th:unless="${currentPage <
totalPages}">Last</span>
                                        </div>
                                </div>
                        </div>

                </div>
        </body>
</html>
```

New_employee - Add New Employee

Templates > new_employee.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="ISO-8859-1">
		<!-- bootstrap cdn -->
		<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7c
mDA6j6gD" crossorigin="anonymous">
		<title>Employee Management System</title>
	</head>

	<body>
		<div th:insert="navbar"></div>
		<div align="center" class="container">
			<h1>Add/Update Employee</h1>
			<div class="card m-3 p-3">
				<form action="#" th:action="@{/saveEmployee}"
th:object="${employee}" method="POST">
					<!-- add hidden form field to handle update employee
-->
					<input type="hidden" th:field="*{id}"/>
					<input type="text" th:field="*{firstName}"
placeholder="John" class="form-control mb-3"/>
					<input type="text" th:field="*{lastName}"
placeholder="Doe" class="form-control mb-3"/>
					<input type="email" th:field="*{email}"
placeholder="john@example.com" class="form-control mb-3"/>
					<div class="row justify-content-around">
						<a th:href="@{/}" class="btn btn-secondary
col mx-3">Cancel</a>
						<input type="submit" value="Submit"
class="btn btn-primary col mx-3"/>
					</div>
				</form>
			</div>
		</div>
	</body>
</html>
```

Registration - Register New User

Templates > register.html

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="ISO-8859-1">
		<!-- bootstrap cdn -->
		<link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet"
```

```
integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">
        <title>Employee Management System</title>
    </head>

    <body>
        <!-- create HTML registration form -->
        <div class="container">
            <div class="card m-5">
                <div class="card-title text-center"><h2>Registration</h2></div>
                <div class="card-body">
                    <form th:action="@{/registration}" method="POST" th:object="${user}">
                        <div class="input-group mb-3">
                            <label for="firstName" class="input-group-text">First Name</label>
                            <input type="text" class="form-control" th:field="*{firstName}" id="firstName" placeholder="John" required autofocus>
                        </div>

                        <div class="input-group mb-3">
                            <label for="lastName" class="input-group-text">Last Name</label>
                            <input type="text" class="form-control" th:field="*{lastName}" id="LastName" placeholder="Doe" required autofocus>
                        </div>

                        <div class="input-group mb-3">
                            <label for="email" class="input-group-text">Email</label>
                            <input type="email" class="form-control" th:field="*{email}" id="email" placeholder="john@example.com" required autofocus>
                        </div>

                        <div class="input-group mb-3">
                            <label for="password" class="input-group-text">Password</label>
                            <input type="password" class="form-control" th:field="*{password}" id="password" placeholder="Password" required autofocus>
                        </div>

                        <div align="center" class="mb-3">
                            <input type="submit" class="btn btn-primary col-4" value="Submit">
                            <br/>
                            <span>Already Registered? Click <a href="/" th:href="@{/login}">here</a> to login.</span>
                        </div>
                    </form>

                    <!-- success message -->
```

```
                                        <div th:if="${param.success}">
                                                <div class="alert alert-success">Registration
Successful.</div>
                                        </div>

                                        <!-- failure message -->
                                        <div th:if="${param.error}">
                                                <div class="alert alert-danger">Registration
Failed. Please try again.</div>
                                        </div>
                                </div>
                        </div>
                </div>

        </body>
</html>
```

Login - Login Registered User

Templates > login.html

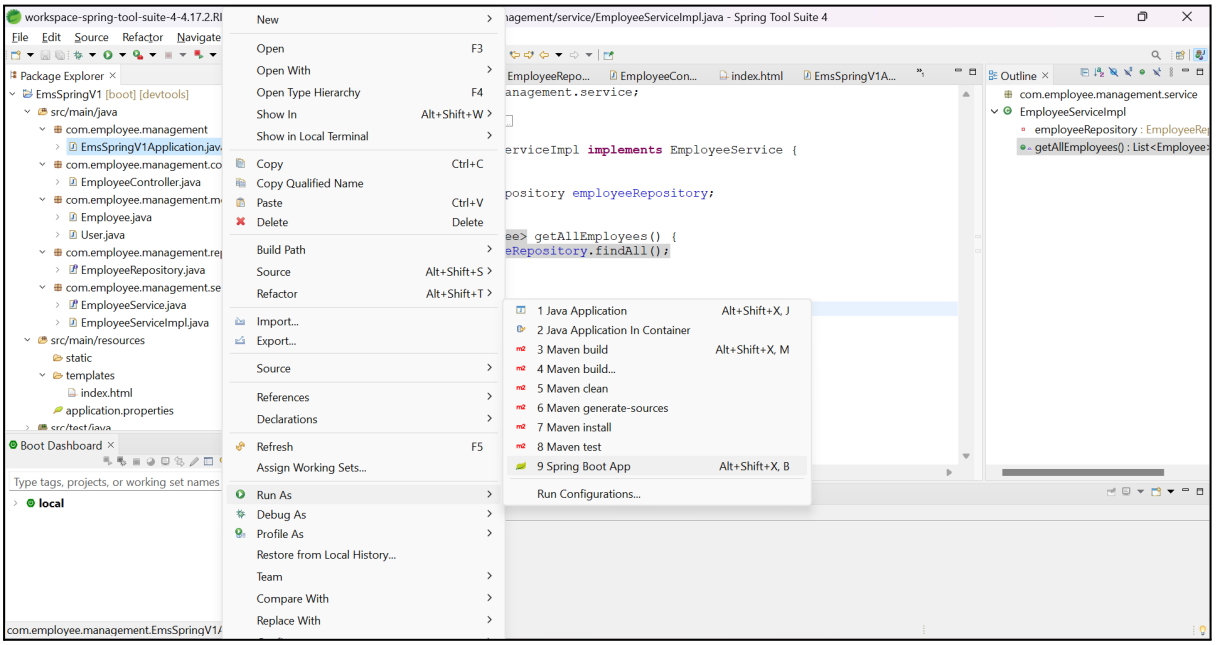
```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="ISO-8859-1">
                <!-- bootstrap cdn -->
                <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/bootstrap.min.css"
rel="stylesheet"
integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7c
mDA6j6gD" crossorigin="anonymous">
                <title>Employee Management System</title>
        </head>

        <body>
                <!-- create HTML login form -->
                <div class="container">
                        <div class="card m-5">
                                <div class="card-title text-center"><h2>Login</h2></div>
                                <div class="card-body">
                                        <!-- <form th:action="@{/login}" method="POST"
th:object="${user}"> -->
                                        <form th:action="@{/login}" method="POST">
                                                <div class="input-group mb-3">
                                                        <label for="username"
class="input-group-text">Username</label>
                                                        <!-- <input type="email"
class="form-control" th:field="*{username}" id="username" name="username"
placeholder="john@example.com" required autofocus/> -->
                                                        <input type="email"
class="form-control" id="username" name="username" placeholder="john@example.com"
required autofocus/>
                                                </div>
```

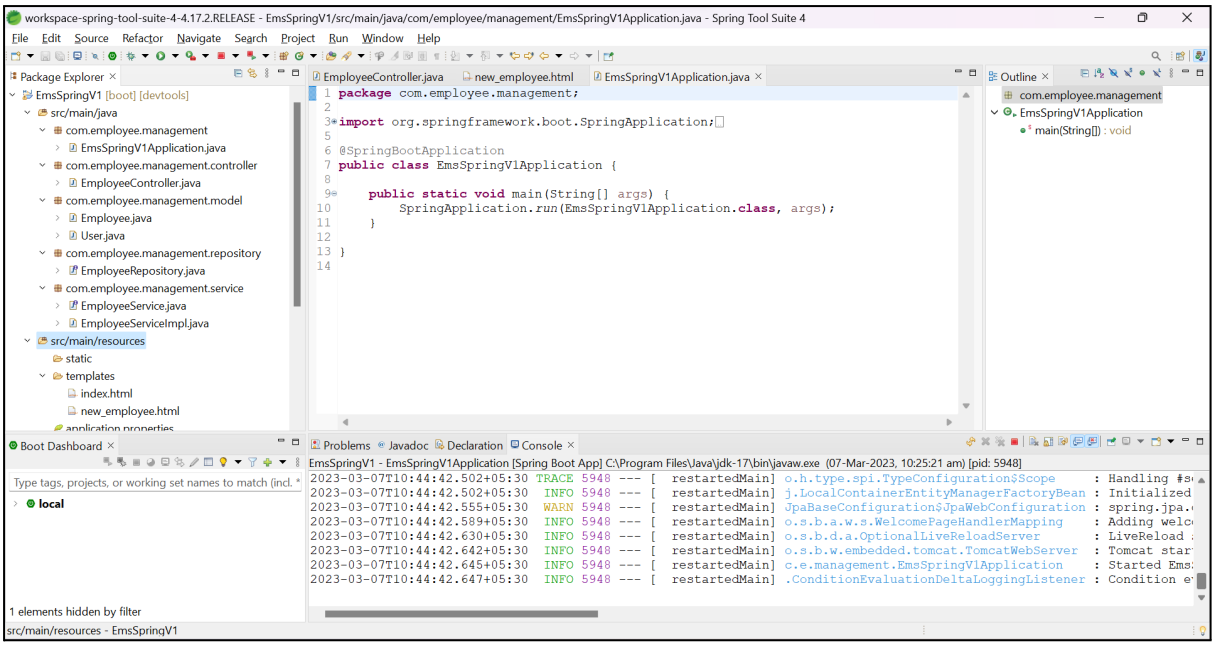
```
                                        <div class="input-group mb-3">
                                            <label for="password"
class="input-group-text">Password</label>
                                            <!-- <input type="password"
class="form-control" th:field="*{password}" id="password" name="password"
placeholder="Password" required autofocus/> -->
                                            <input type="password"
class="form-control" id="password" name="password" placeholder="Password" required
autofocus/>
                                        </div>

                                        <div align="center" class="mb-3">
                                            <input type="submit" class="btn
btn-primary col-4" value="Submit">
                                            <br/>
                                            <span>New user? Click <a href="/"
th:href="@{/registration}">here</a> to register.</span>
                                        </div>
                                    </form>

                                    <!-- success message -->
                                    <div th:if="${param.success}">
                                        <div class="alert alert-success">Login
Successful.</div>
                                    </div>

                                    <!-- failure message -->
                                    <div th:if="${param.error}">
                                        <div class="alert alert-danger">Invalid
username or password.</div>
                                    </div>

                                    <!-- logout message -->
                                    <div th:if="${param.logout}">
                                        <div class="alert alert-primary">You have
been logged out.</div>
                                    </div>
                                </div>
                            </div>
                        </div>

                </body>
</html>
```

Step 10: Run the Spring Project

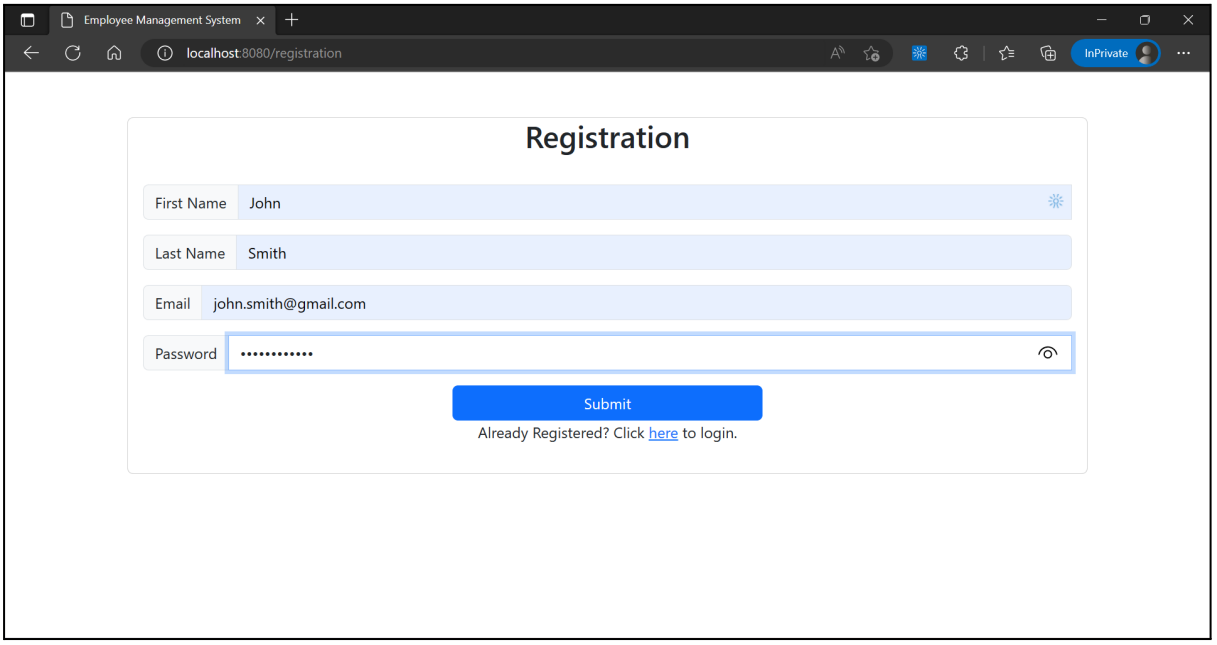
Run the spring project by right-clicking the starting file of the project (com.employee.management > EmsSpringV1Application.java), select *Run As*, then *Spring Boot App*

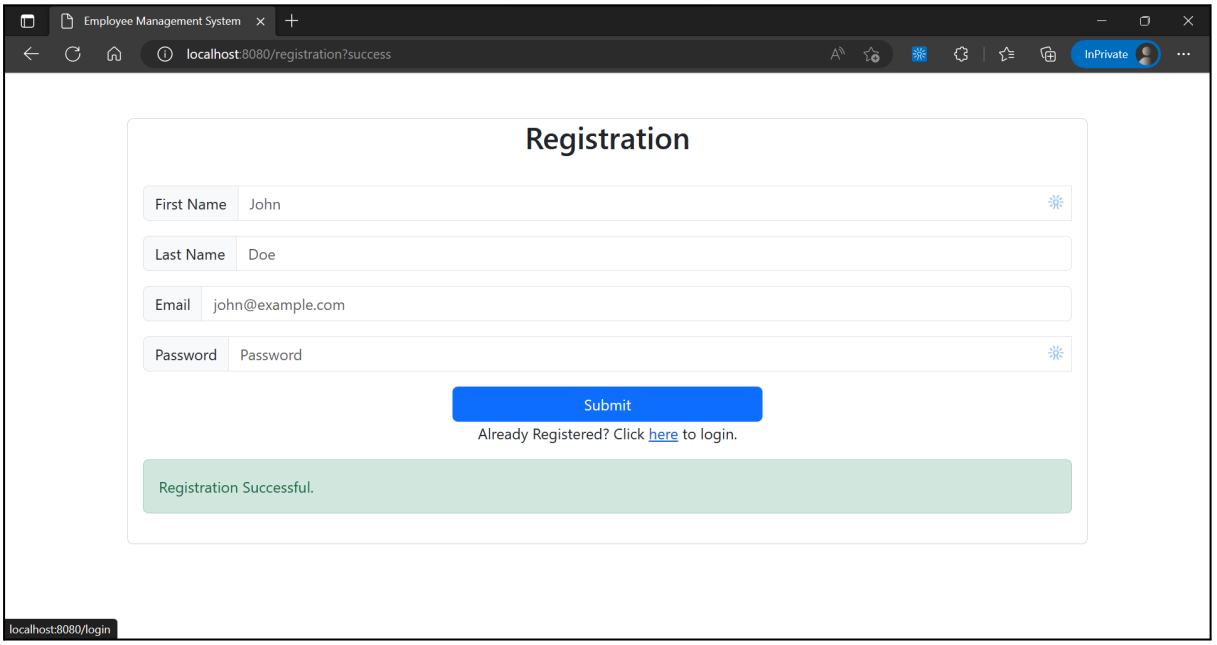
Screenshots



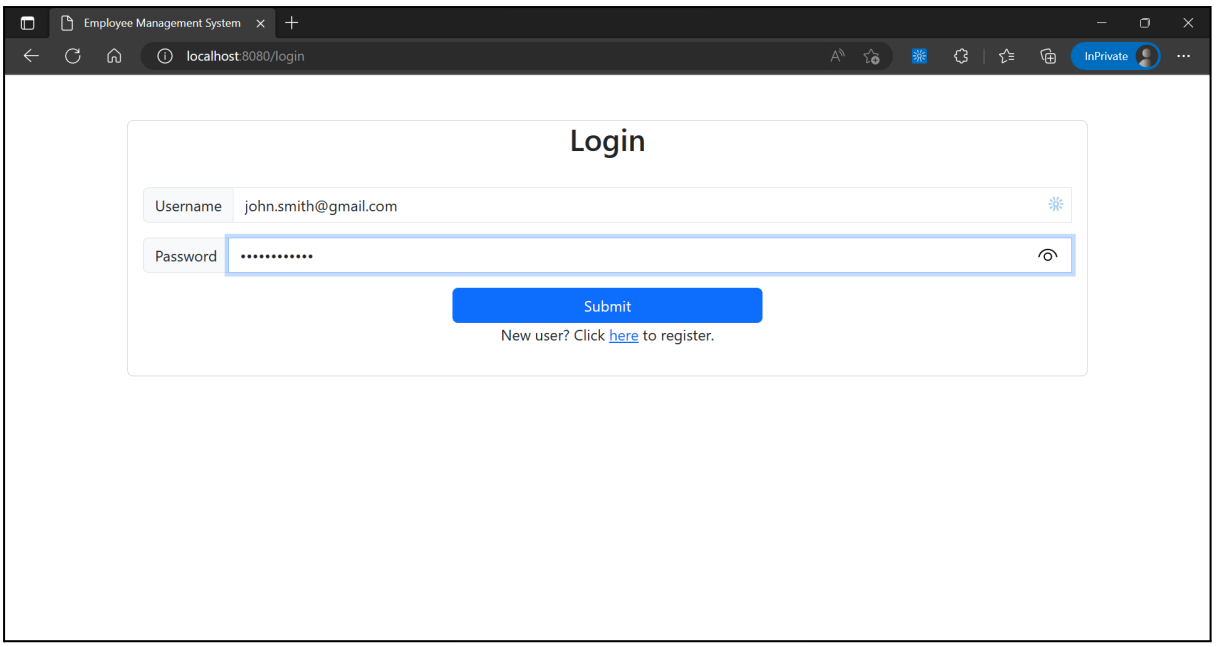


Register new user

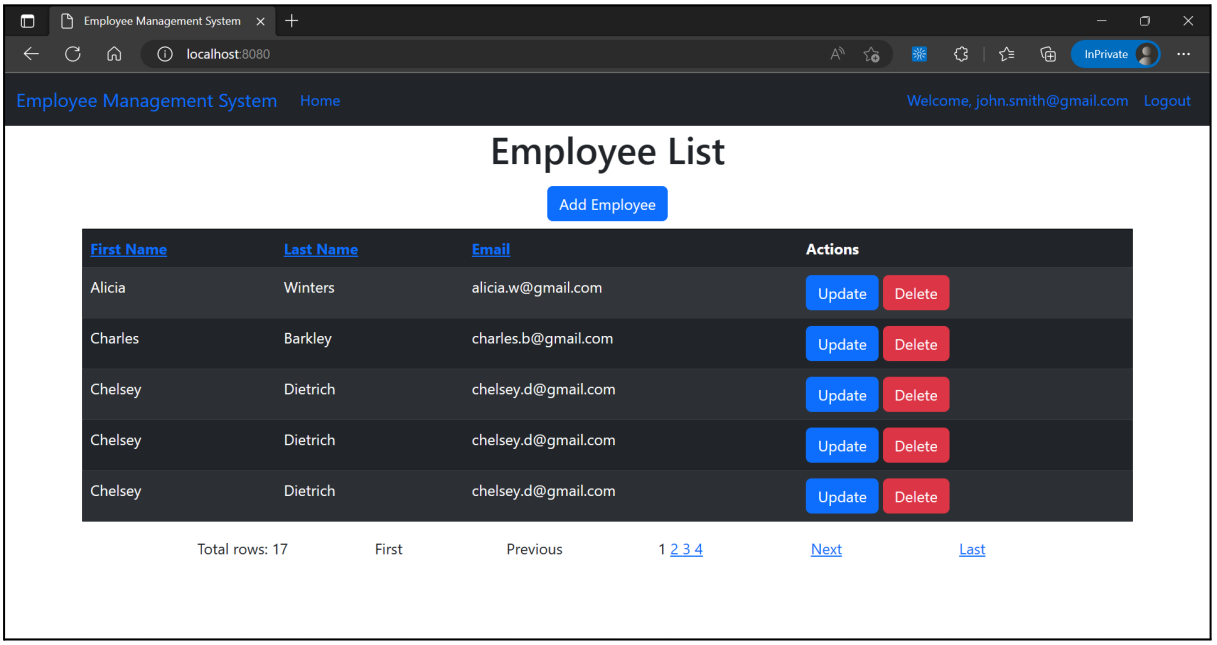




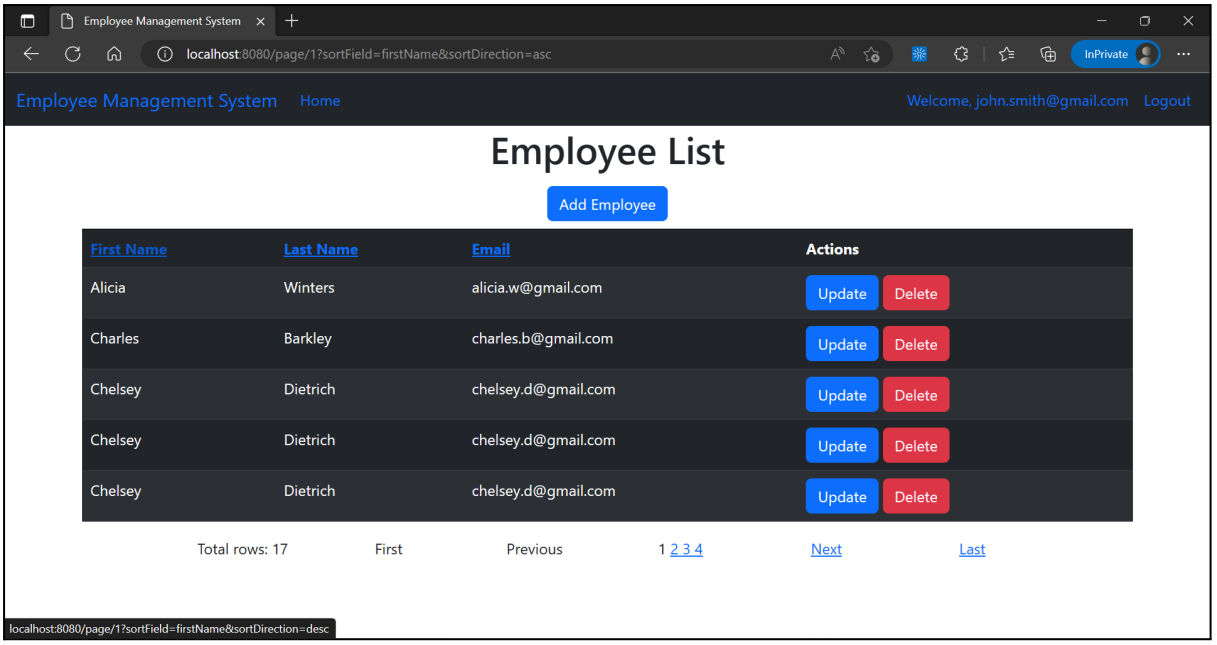
Login with registered user



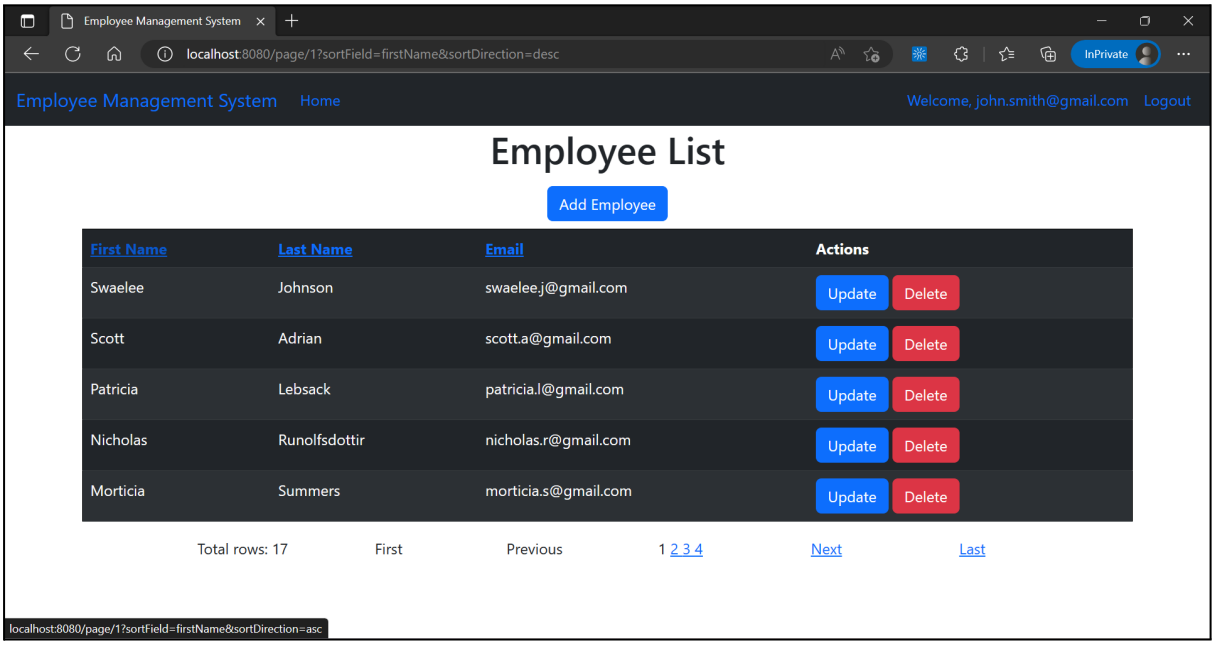
Show all employees with pagination, sorting, and navbar



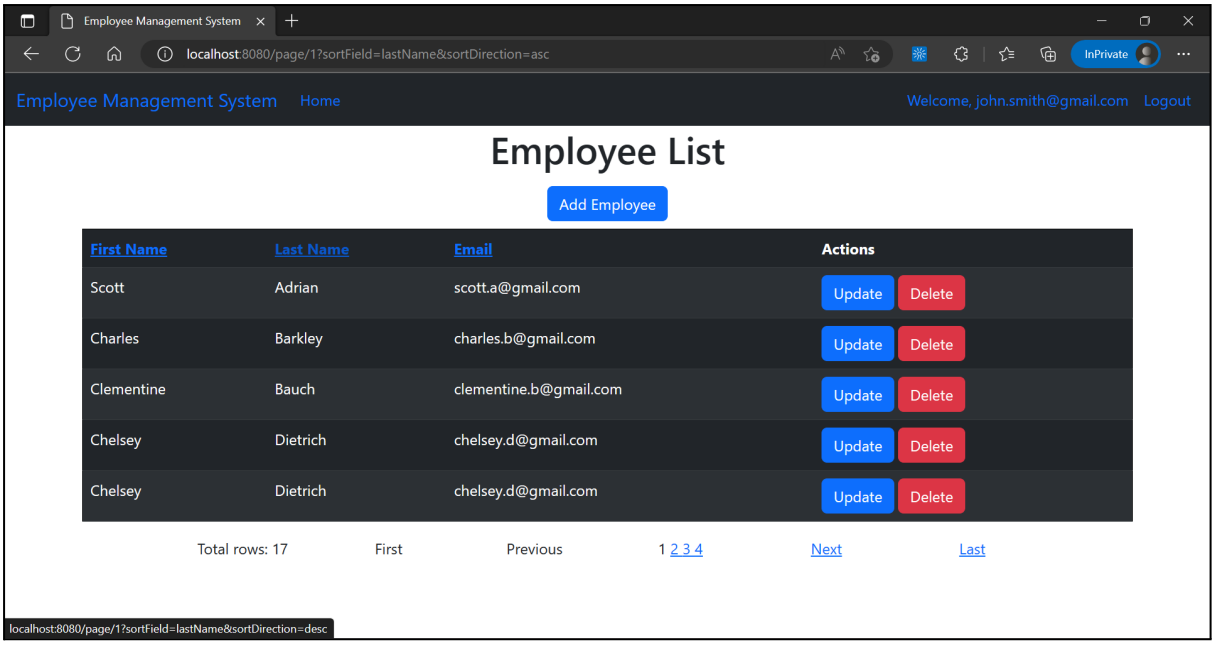
Pagination & sorting (ascending - first name)



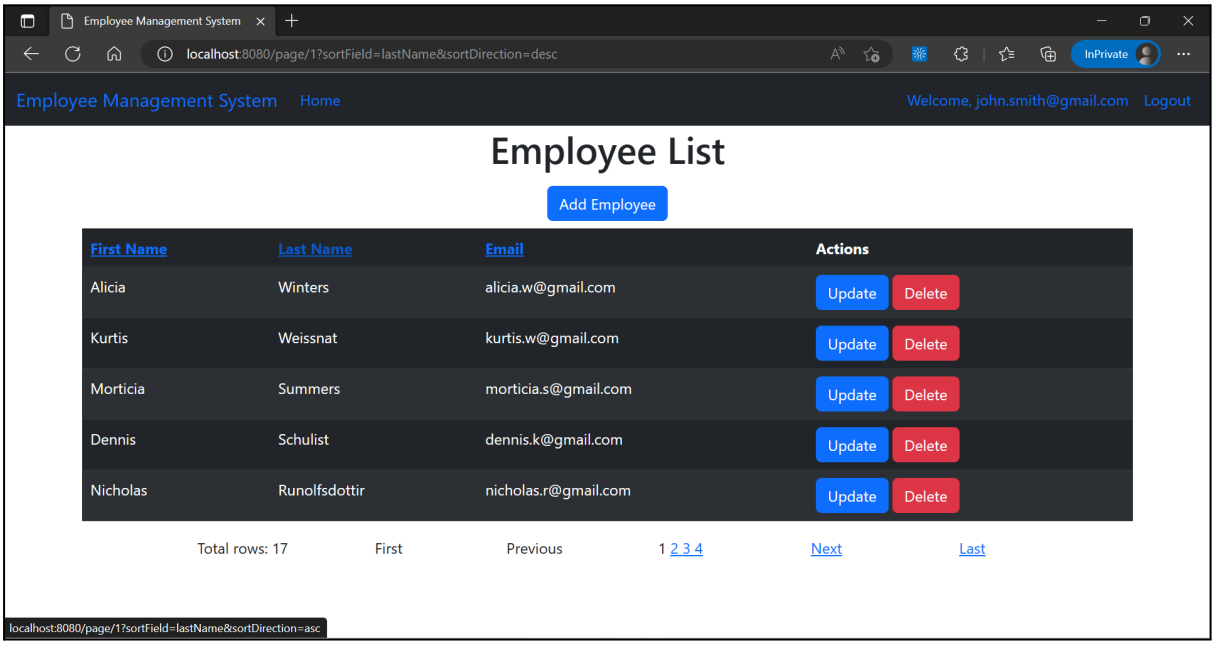
Pagination & sorting (descending - first name)



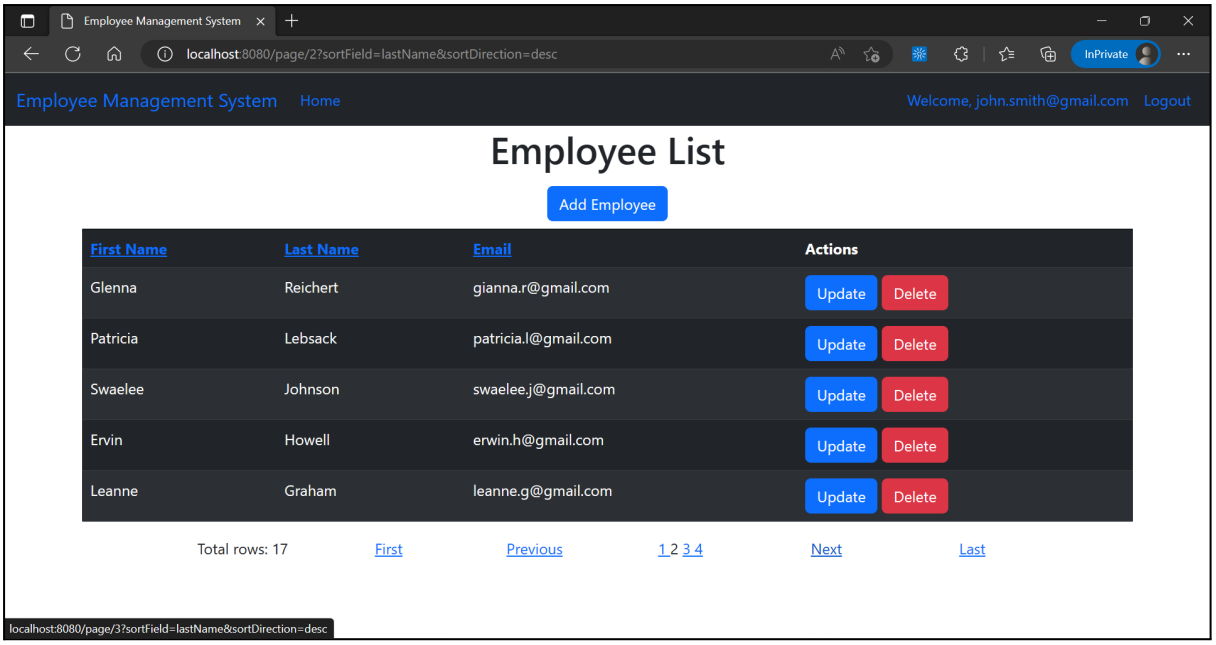
Pagination & sorting (ascending - last name)



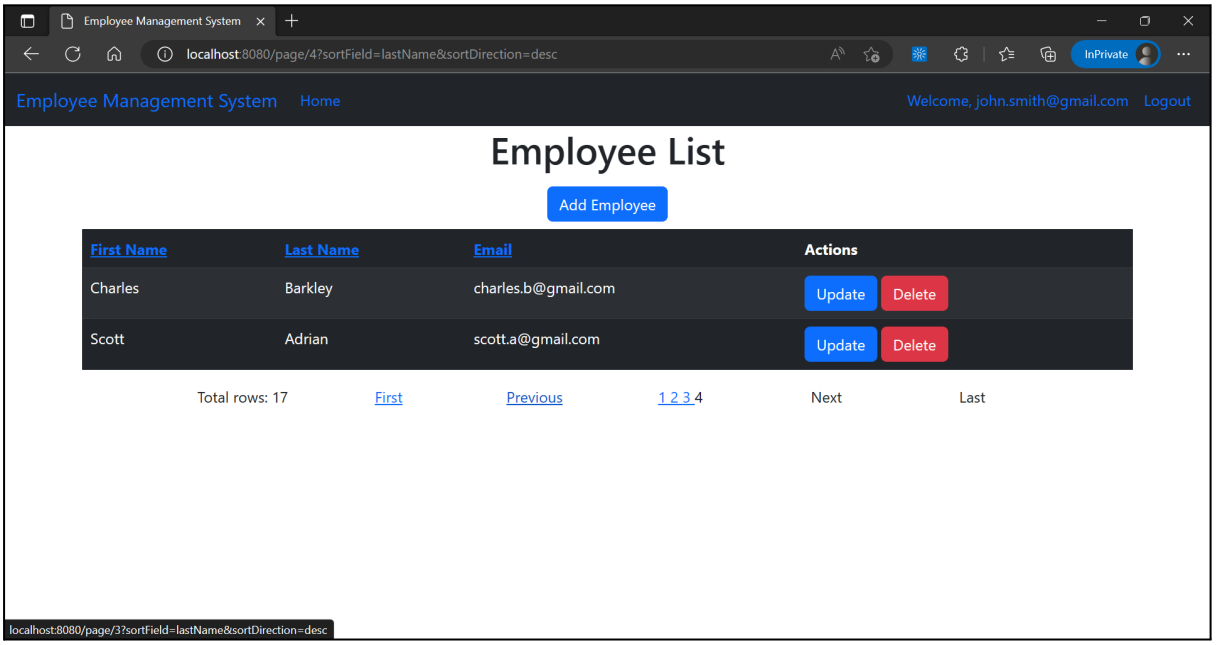
Pagination & sorting (descending - last name)



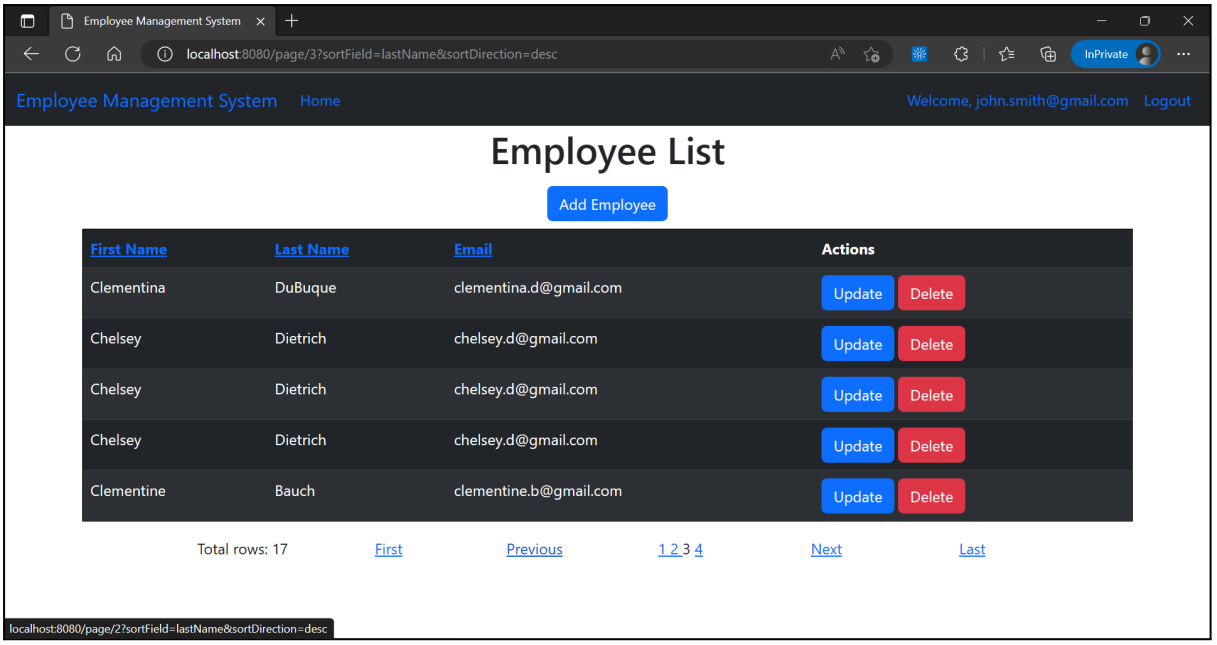
Next page



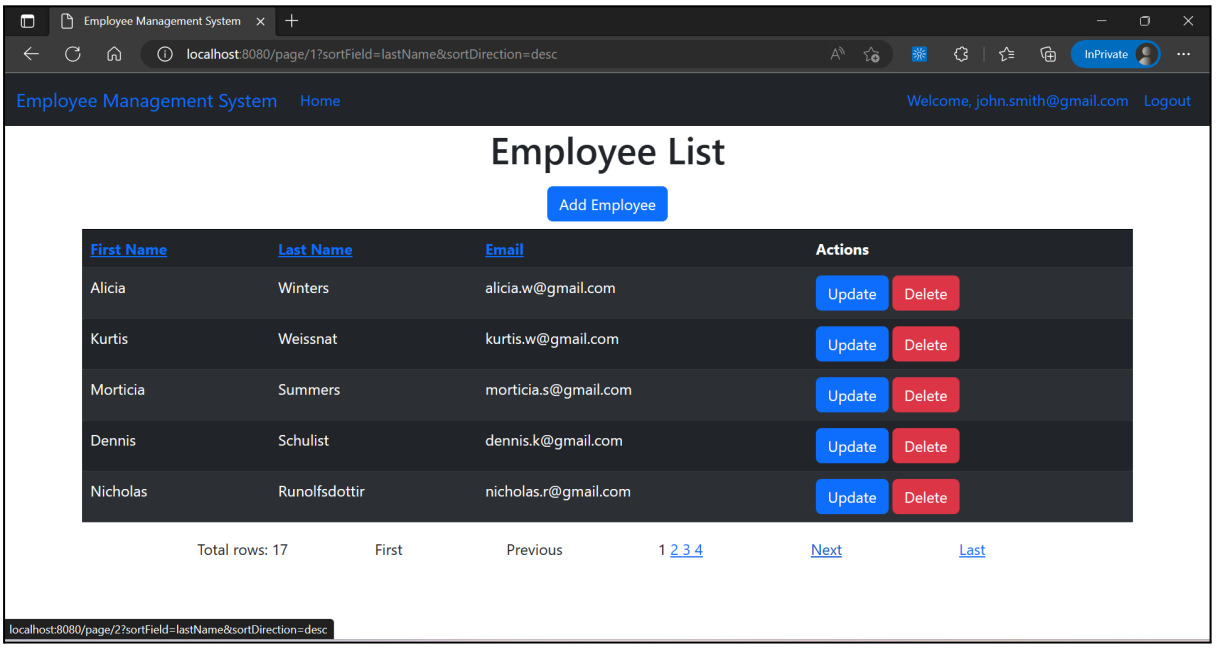
Last page



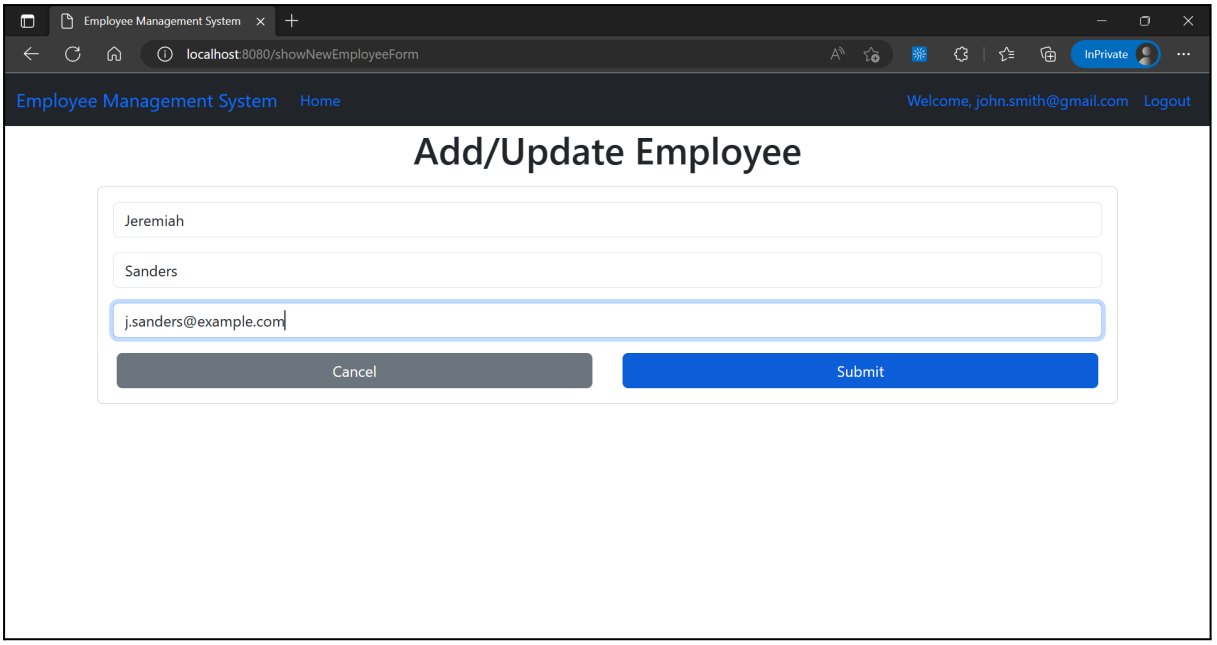
Previous Page

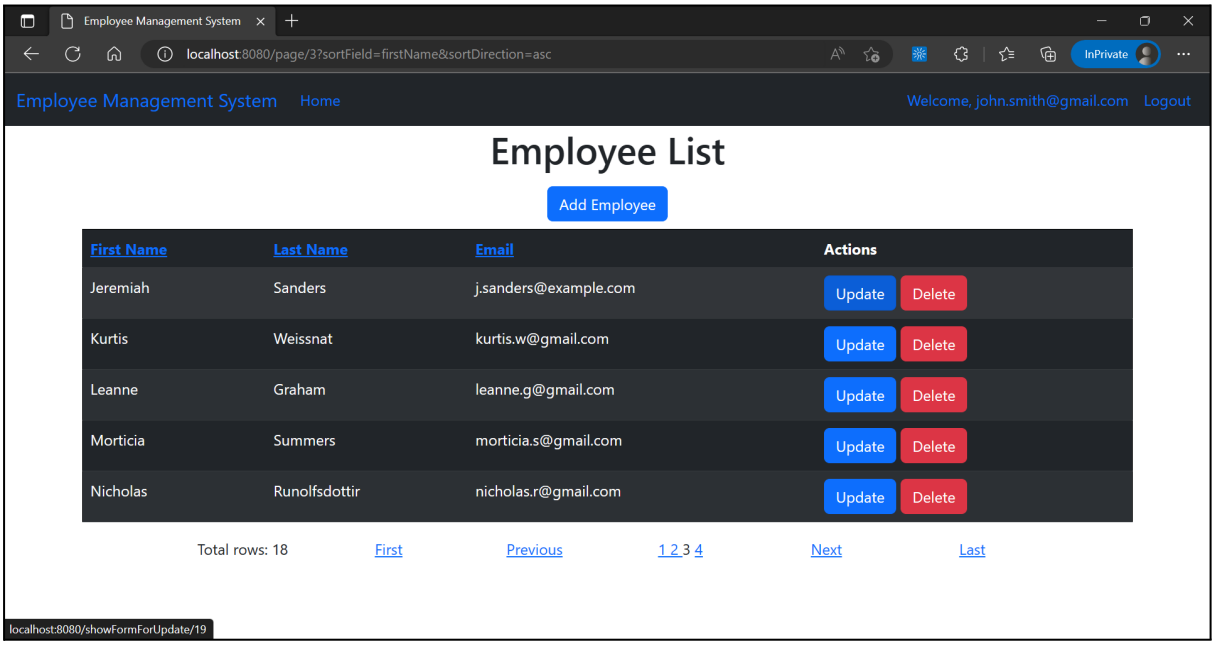


First Page

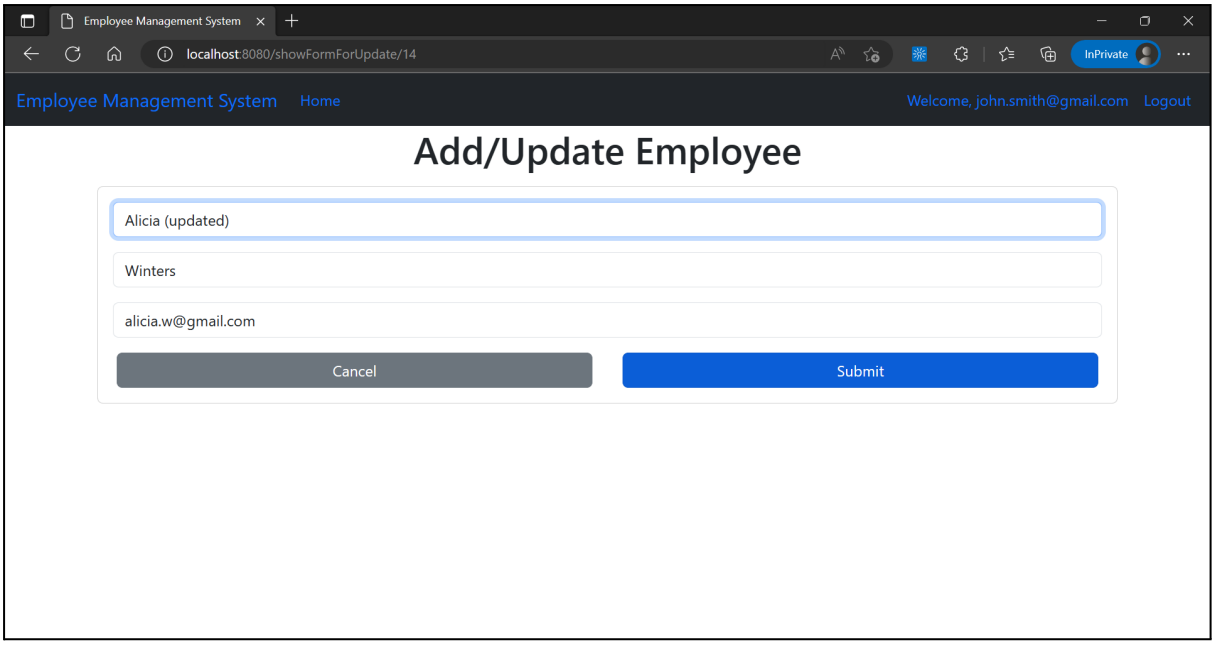


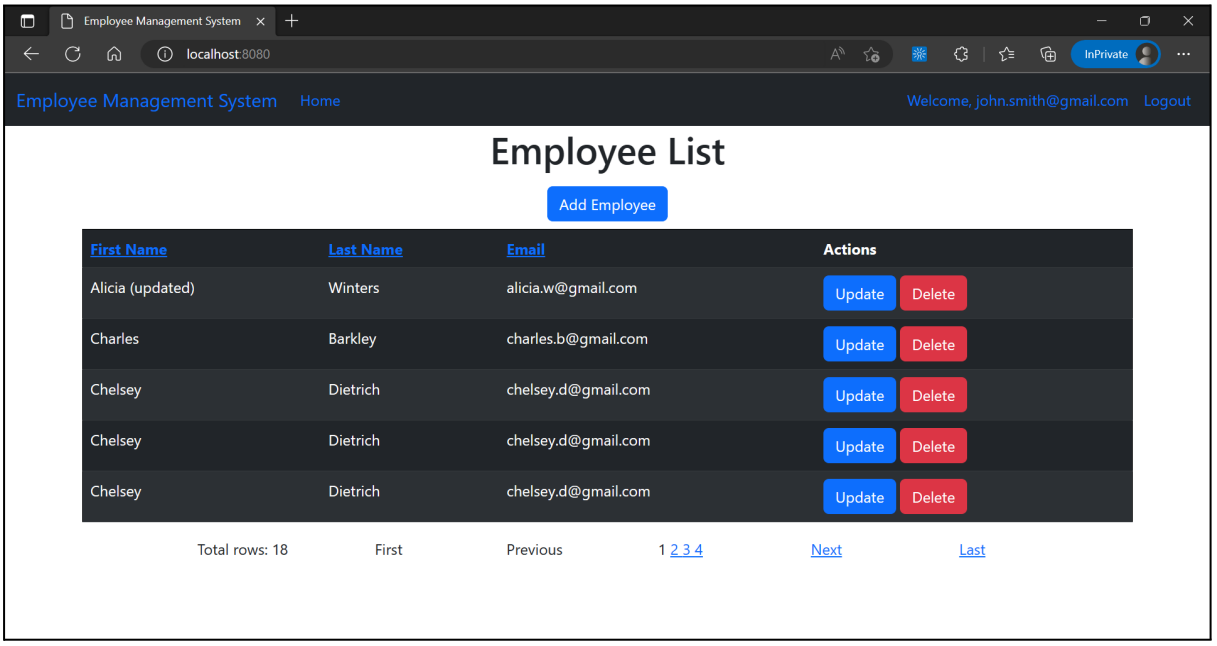
Add new employee



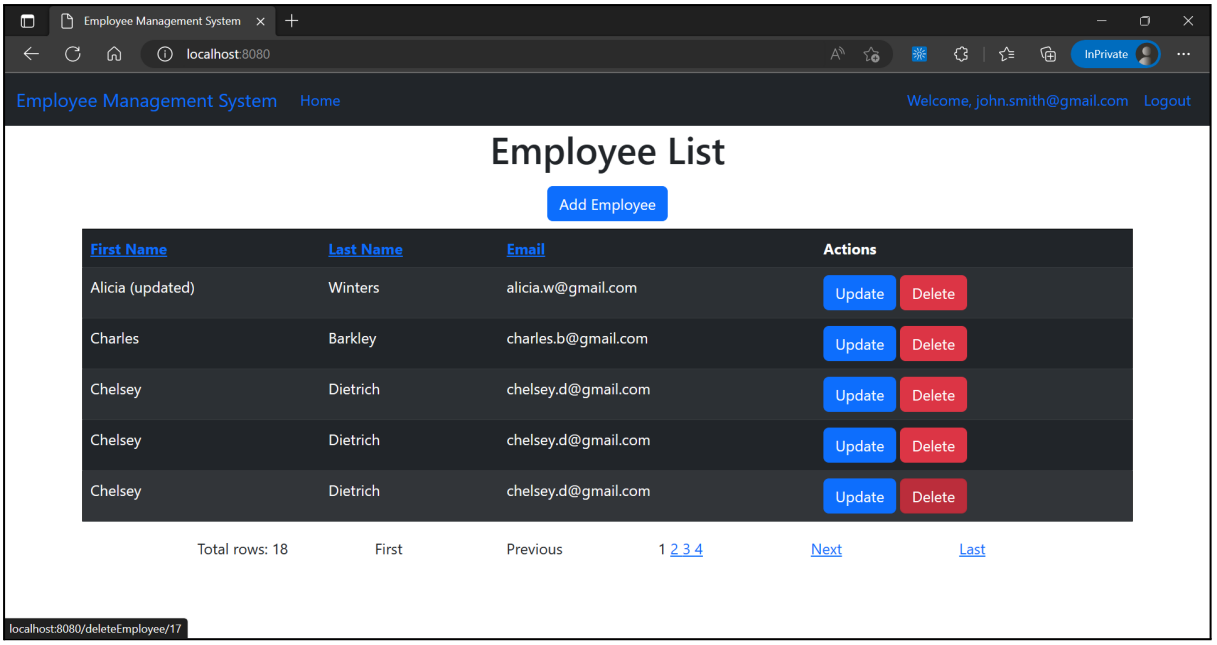


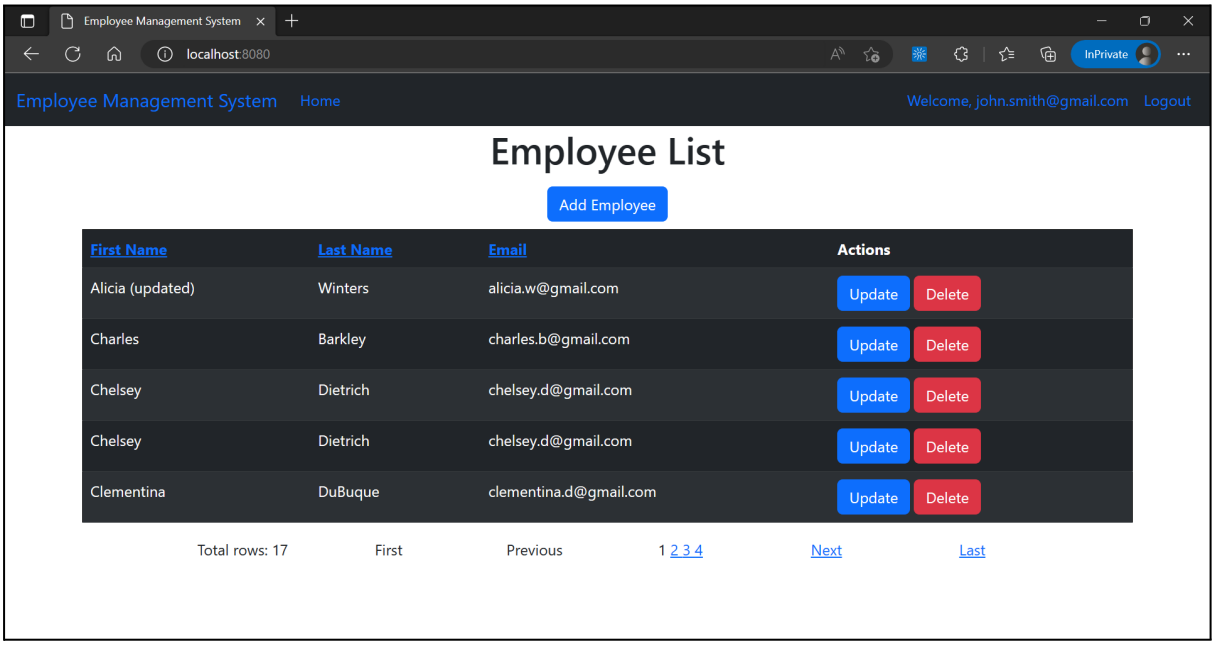
Update employee details





Delete employee





X --- END OF PROJECT --- X